NTNU

Norwegian University of
Science and Technology

Investigation of Multivariate Freight Rate Prediction Using Machine Learning and AIS Data

Patrick Andre Næss

Marine Technology
Submission date: June 2018
Supervisor: Bjørn Egil Asbjørnslett, IMT
Co-supervisor: Carl Fredrik Rehn, IMT

Norwegian University of Science and Technology
Department of Marine Technology

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER THESIS IN MARINE TECHNOLOGY
SPRING 2018

For stud.techn. Patrick André Næss

Investigation of Multivariate Freight Rate Prediction
Using Machine Learning and AIS Data

Background
We live in an age of digitization with an increasing amount of data being generated daily. Some of the largest corporations in the world base their income on exploiting vast amounts of data. In shipping, the digitization age and its opportunities are relatively unexplored terrain. With more or less all existing merchant vessels being online in real time through AIS, new possibilities are emerging. The motivation for looking into AIS data is the opportunity to provide useful insight into the nature of maritime transportation. In the context of maritime economics, this information may provide the ability to give added predictive value in forecasting short-term fluctuations in shipping commodities. To anticipate such fluctuations is a crucial element to long-term profitability for both operators and shipowners. Thus making the purpose of this project an essential aspiration in maritime research.

The importance of gas as a cleaner energy source is growing. Liquid Petroleum Gas (LPG), a generic name for commercial propane and butane, is derived from the extraction rate of crude oil refining and natural gas. One of the main reasons for wanting to investigate the LPG market is that it has hardly been subject to previous academic research, presumably because of the comparatively small size of the total market. To the candidate's knowledge, there has not been a study investigating freight rate prediction of LPG with AIS data.

As the amount of available data in the shipping industry increases, methods to take advantage of these emerge. Machine learning methods are not a new development. However, these sophisticated methods have shown to outperform traditional methods when the amount of available data increases. From image recognition in healthcare to driverless cars, these methods can model complex problems. In time series forecasting, new developments within neural networks show promise in forecasting rates within crude oil and dry bulk commodities. To the candidate's knowledge, there has not been a study investigating spot price prediction with machine learning and AIS data.

Objective
The overall objective is to investigate whether multivariate machine learning forecasting methods using features extracted from AIS-data adds additional information in predicting short-term freight rates.

Tasks
The candidate shall/is recommended to cover the following tasks in the master thesis:
a. Extract and explore patterns for relevant market specific vessels from global AIS data.
b. Create and identify valuable features extracted from AIS data through features selection methods.
c. Develop machine learning prediction models to forecast short-term freight rates using multivariate data.
d. Evaluate the prediction models with statistical metrics and traditional models used in ocean freight markets.

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

General

In the thesis the candidate shall present his personal contribution to the resolution of a problem within the scope of the thesis work.

Theories and conclusions should be based on a relevant methodological foundation that through mathematical derivations and/or logical reasoning identify the various steps in the deduction.

The candidate should utilize the existing possibilities for obtaining relevant literature.

The thesis should be organized in a rational manner to give a clear statement of assumptions, data, results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Telegraphic language should be avoided.

The thesis shall contain the following elements: A text defining the scope, preface, list of contents, summary, main body of thesis, conclusions with recommendations for further work, list of symbols and acronyms, reference and (optional) appendices. All figures, tables and equations shall be numerated.

The supervisor may require that the candidate, in an early stage of the work, present a written plan for the completion of the work. The original contribution of the candidate and material taken from other sources shall be clearly defined. Work from other sources shall be properly referenced using an acknowledged referencing system.

The work shall follow the guidelines given by NTNU for the MSc Thesis work. The work load shall be in accordance with 30 ECTS, corresponding to 100% of one semester.

The thesis shall be submitted electronically on DAIM:
- Signed by the candidate.
- The text defining the scope included.
- Computer code, input files, videos and other electronic appendages can be uploaded in a zip-file in DAIM. Any electronic appendages shall be listed in the thesis.

Supervision:
Main supervisor: Prof. Bjørn Egil Asbjørnslett
Co-supervisor: Dr. Carl Fredrik Rehn

Deadline: 25.06.2018

Date: 22.06.18

[illegible]

Prof. Bjørn Egil Asbjørnslett

Preface

This thesis marks the final part of my Master of Science degree with specialization in Marine Systems Design & Logistics at the Department of Marine Technology (IMT). The work has been carried out during the spring semester of 2018 at the Norwegian University of Science and Technology (NTNU), and corresponds to 30 ECTs.

For those interested, I strongly recommend to read the thesis in its entirety. The thesis encompasses a wide range of disciplines. The target audience for this work includes both researchers and practitioners with interest in one or more of the following topics: freight rate prediction, machine learning, time series forecasting and the Liquid Petroleum Gas (LPG) shipping market.

Trondheim, June 25, 2018

[illegible]

Patrick André Næss

Acknowledgment

I would like to thank the following persons for their great help during this project:

Prof. Bjørn Egil Asbjørnslett, my supervisor, for highly competent guidance throughout the project.
Dr. Carl Fredrik Rehn, my co-supervisor, for supporting me and for valuable input and rewarding discussions.
Bjørnar Brende Smestad and Jon Hovem Leonhardsen for their previous work within AIS data exploitation, simplifying the data handling part of the project.
Laurent Viguier at Equinor, for providing med with market-specific data and giving valuable input on market dynamics.
PhD cand. Vit Prochazka at NHH, for valuable input in the feature construction process.
Jon Arve Røyset and the Norwegian Coastal Administration for providing me with AIS data.

P.A.N.

Summary

This thesis investigates whether multivariate machine learning forecasting methods, using information extracted from Automatic Identification System (AIS) data, add additional information in predicting short-term freight rates. The focus in this thesis is the Liquid Petroleum Gas (LPG) shipping market, directed on the prediction of the Mont Belvieu propane spot price. Forecasting LPG spot prices is challenging due to a comparatively small fleet, niche trading patterns and cargo bases, creating high volatility. Nonetheless, anticipating fluctuations in freight rates is a crucial element to long-term profitability for both operators and shipowners.

A thorough literature review is presented to investigate the state-of-the-art within the topic. The data used in this research is provided by the Norwegian Coastal Authorities and Equinor. Several methods are proposed to extract and create predictive features. A significant amount of market specific features are generated and tested with selection methods to identify the most powerful predictors of the Mont Belvieu propane spot price.

Two machine learning models are employed; the Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM) neural networks. To baseline the results, the no-change persistence model and the Vector Autoregressive (VAR) model are utilized, where the latter is widely used for multivariate forecasting in maritime econometrics. By optimizing the neural network architectures with a genetic algorithm, the LSTM-model is identified to perform better than the baseline model and the traditional multivariate VAR-model. Moreover, the results indicate that both the machine learning and traditional models with information extracted from AIS data are able to predict short-term fluctuations in freight rates more accurately than without this information.

In conclusion, there is evidence in favor of using information extracted from AIS data in short-term freight rate prediction. Furthermore, the results also suggest favorability of using multivariate machine learning forecasting models over traditional models. This thesis suggests that further work within the use of AIS data in the context of maritime econometrics is recommended. After surveying the performance in the case study, it is finally recommended to investigate whether the approach used in this study can be readily applied to other ocean freight industries.

Sammendrag

Denne oppgaven undersøker om multivariate maskinlæringsmodeller, ved hjelp av informasjon hentet fra Automatisk identifikasjonssystem (AIS) data, gir tilleggsinformasjon for å predikere kortsiktige fraktrater. Fokuset i denne oppgaven er fraktmarkedet for flytende petroleumsgass (LPG), rettet mot prediksjonen av spotpris på propan ved Mont Belvieu. Det å predikere LPG-spotpriser er utfordrende på grunn av en forholdsvis liten flåte og nisje-handelsmønstre, noe som skaper høy uforutsigbarhet. Det å kunne forutsi svingninger i fraktrater er et avgjørende element for langsiktig lønnsomhet for både operatører og redere.

En grundig litteraturgjennomgang presenteres for å undersøke *state-of-the-art* innen emnet. Dataene som brukes i denne undersøkelsen er gitt av Kystverket og Equinor. Flere metoder foreslås for å finne og lage gode prediktive variable. En betydelig mengde markedsspesifikke variable har blitt generert og testet med utvalgsmetoder for å identifisere de beste variablene til å predikere Mont Belvieu-propanprisen.

To maskinlæringsmodeller er utforsket; Multilayer Perceptron (MLP) og Long Short Term Memory (LSTM) nevrale nettverk. For å sammenligne resultatene, benyttes en vedvarende prediksjonsmodell og en Vector Autoregressive (VAR) modell, der sistnevnte er mye brukt for å gjennomføre multivariate prediksjoner i maritim økonometri. Ved å optimalisere de nevrale nettverksarkitekturene med en genetisk algoritme, er LSTM-modellen funnet å gi bedre resultat enn de tradisjonelle modellene. Videre viser resultatene at både maskinlæring og tradisjonelle modeller med informasjon hentet fra AIS-data er i stand til å forutsi kortvarige svingninger i fraktrater mer nøyaktig enn uten denne informasjonen.

Det er konkludert med at resultatene indikerer for bruk av informasjon hentet fra AIS-data i kortsiktig fraktprognoser. Videre tyder resultatene også på fordeler for bruk av multivariate maskinlæringsmodeller over tradisjonelle modeller. Denne oppgaven antyder at videre arbeid innen bruk av AIS data i sammenheng med maritim økonometri anbefales. Basert på de gode resultatene fra denne metoden er det til slutt anbefalt å undersøke om tilnærmingen som brukes i dette studiet kan benyttes i andre shippingsegmenter.

Contents

List of Figures

List of Tables

Acronyms

AIC	Akaike Information Criterion
AIS	Automatic Identification System
ANN	Artificial Neural Network
Atl	Atlantic
ARIMA	Autoregressive Integrated Moving Average
BDI	Baltic Dry Index
BIC	Bayesian Information Criterion
cbm	Cubic Meters, m^3
COG	Course Over Ground
CP	Arabian Gulf
DAR	Directional Accuracy Ratio
DWT	Deadweight Tonnage
EstP	East Pacific
ETA	Estimated Time of Arrival
FEI	Far East Index
GA	Genetic Algorithm
GARCH	Autoregressive Conditional Heteroskedasticity
GB	Gigabytes
GOM	Gulf of Mexico
IMO	International Maritime Organization
Ind	Indian Ocean
LNG	Liquefied Natural Gas
LPG	Liquefied Petroleum Gas
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MB	Mont Belvieu
Med	Mediterranean
MI	Mutual Information
MIC	Maximal Information Coefficient
MLP	Multilayer Perceptron
MMSI	Maritime Mobile Service Identity
MSE	Mean Square Error
mt	Metric Tonne
NWE	North West Europe
OLS	Ordinary Least Squares
RF	Random Forests
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RSS	Residual Sum of Squares
SGD	Stochastic Gradient Decent
SOG	Speed Over Ground
SOLAS	Safety of Life At Sea
SQL	Structured Query Language
STW	Speed Through Water
SVM	Support Vector Machine
S-AIS	Satellite Automatic Identification System
TSS	Total Sum of Squares
UTC	Coordinated Universal Time
VAR	Vector Autoregressive
VHS	Very High Frequency System
VLCC	Very Large Crude Carrier
VLGC	Very Large Gas Carrier
WTI	West Texas Intermediate

1 | Introduction

1.1 Background

We live in an age of digitization with an increasing amount of data being generated daily. Some of the largest corporations in the world base their income on exploiting vast amounts of data. In shipping, the digitization age and its opportunities are relatively unexplored terrain. With more or less all existing merchant vessels being online in real time through the Automatic Identification System (AIS), new possibilities are emerging. For the stakeholders within shipping, including but not limited to shipowners, operators, and shipyards, the data appears to be a relatively untapped resource and can provide valuable decision support regarding vessel operation, trends and trade.

The motivation for looking into AIS data is the opportunity to provide useful insight into the nature of maritime transportation. In the context of maritime economics, this information may provide the ability to give added predictive value in forecasting short-term fluctuations in shipping commodities. To anticipate such fluctuations is a crucial element to long-term profitability for both operators and shipowners. Thus making the purpose of this project an essential aspiration in maritime research.

The importance of gas as a cleaner energy source is growing. Liquid Petroleum Gas (LPG), a generic name for commercial propane and butane, is not a raw material but extracted as a by-product of gas processing and crude oil refining. LPG as a commodity has become an increasingly important energy source for cooking and heating, as well as a vital petrochemical resource. One of the main reasons for wanting to investigate the LPG market is that it has hardly been subject to previous academic research, presumably because of the comparatively small size of the total market. To the candidate's knowledge, there has not been a study investigating freight rate prediction of LPG with AIS data.

As the amount of available data in the shipping industry increases, methods to take advantage of these emerge. Machine learning methods are not a new development. However, these sophisticated methods have shown to outperform traditional methods when the amount of available data increases. From image recognition in healthcare to driverless cars, these methods can model complex problems. In time series forecasting, new developments within neural networks show promise in forecasting rates within crude oil and dry bulk commodities. To the candidate's knowledge, there has not been a study investigating freight rate prediction with machine learning and AIS data.

1.2 Literature Review

The papers reviewed represent the foundation for the work and give an understanding of the currently existing methods and applications within AIS data exploitation. The use of satellites to receive AIS data is still a relatively new concept, so there have been relatively few studies using AIS data. A collection of AIS-related papers have been reviewed as a part of the work done in this project. Multivariate forecasting of LPG shipping freight rates using machine learning includes a wide range of disciplines. In addition to previous work regarding AIS data, both literature concerning maritime economics, the LPG freight market, forecasting techniques and machine learning methods are among other important disciplines. This section will summarize some previous work within the respective fields to identify the state-of-the-art within the them.

A large part of AIS-related literature is related to traffic estimation and shipping networks and patterns. Kaluza et al. (2010) present an interpretation of the global cargo ship movements as a complex network. The overall purpose is to understand global trade patterns and the influence it has on bioinvasion. Spiliopoulos et al. (2017) present a four-step approach on how to transform AIS data to information for understanding global trade patterns. The results can be used to see changes in shipping trade patterns, which again is connected to global trade patterns. Haji et al. (2013) present the development of a model capable of representing container flows at a global level. AIS data is utilized to detect the positions and sizes of container vessels, and this is used to estimate container flow. In their work, Arguedas et al. (2014) develop an algorithm to construct maritime shipping lanes from AIS data. The lanes are detected by behavioral changes, such as the Course Over Ground (COG) and port calls. Wu et al. (2017) apply methods for mapping the global vessel density and traffic density. Vector-based and grid-based methods are utilized for traffic density, and the latter has some of the same characteristics as geo-fencing, a method of extracting data based on geographical boundaries.

Another segment of previous AIS related literature concerns operational issues such as vessel speeds, where Assmann et al. (2015) looks at the relationship between vessel speeds under conditions of high freight rates and low bunker prices. They find some support for this theory and that speed optimizing behavior is much more pronounced on backhaul trips than on laden trips. Leonhardsen (2017) investigates through his master thesis the possible fuel savings from rapidly re-configurable bulbous bows. A vast amount of historical speed records from AIS data is analyzed. The results are used to confirm significant variations in speed during transits, and from transit to transit. A stochastic representation of the speed is proposed, and used in further work to analyze the possible fuel savings. Jia et al. (2017) identify empirically how Very Large Crude Carriers (VLCC) can save fuel and emissions by implementing Virtual Arrival, an operational agreement that involves reducing speed when there is known delay at the discharge port.

In the context of AIS data quality, Smestad and Rødseth (2015) investigates the utility of AIS data, and show how to use heuristics to establish specific ship type with sole use of AIS data. The purpose of predicting ship type without additional data is to avoid the cost of acquiring commercial ship data. Satellite AIS (S-AIS) data is used as a basis to create the heuristics, and a data cleaning process is carried out to exclude vessels that have conflicting and inaccurate data. To verify the accuracy of the heuristics, AIS data is matched with data from Clarkson's Ship Register for various ship types. The heuristics results to classify gas carriers (LPG and LNG) shows that they can be established with a 99 % certainty.

Most of this previous literature is related to vessel operation, safety by anomaly detection, shipping networks, traffic estimation and environmental issues. There have been relatively few studies that have used AIS data in the context of maritime economics. Previous work that utilizes AIS data in maritime economic studies includes Adland et al. (2017) who investigates the reliability of AIS-based trade volumes. They find that AIS-derived data for seaborne crude exports show proper alignment with official export numbers in aggregate. To our knowledge, the only study using features extracted from AIS data to analyze shipping rates is by Olsen and da Fonseca (2017). This thesis investigates the predictive ability of AIS data in the case of prediction Arabian Gulf oil tanker rates. They utilize the traditional Vector Autoregressive (VAR) model in their multivariate forecast and compare it to the results of a univariate forecast. They combine both data extracted from AIS and market-specific data related to Arabian Gulf oil tanker rates. They find that multivariate models perform relatively better than univariate models in predict future freight rates. Conclusive, they find weak evidence in favor of using information from AIS-derived data for predictive purposes.

As all shipping commodities, the LPG freight rates are determined by the balance between supply and demand. In turn, each of these is driven by different factors. Therefore, to get a good picture of this dynamic relationship, it is important to look into how these interact with each other. Concerning academic work, the LPG market has hardly been subject to previous research (Adland et al., 2008). LPG is a young market in shipping context. Engelen and Dullaert (2010) shows that the LPG market is slowly transforming into a competitive setting with an increasing number of private buyers and sellers, rendering the market more efficient. According to Bai and

Lam (2017), the main Very Large Gas Carrier (VLGC) trading routes, unlike other bulk shipping markets with diversified trading patterns and cargo bases, are niche, thus creating more price volatility. This is also acknowledged by Adland et al. (2008), who states that a low number of main trading routes and the comparatively small fleet may increase price volatility compared to other ocean freight markets.

LPG as a shipping commodity, have some distinct features. Unlike crude oil, coal or iron ore, LPG is not a raw material but extracted as a by-product of gas processing and crude oil refining (Engelen and Dullaert, 2010). The main forwarders of LPG are therefore dominated by the major oil and gas companies. This aspect makes the LPG market to be driven by "supply push" rather than "demand pull" in the sense that the volume of LPG shipments is derived from their products (LNG and crude oil) and is therefore not independently set (Adland et al., 2008). A large part of the LPG market consists of shipments between the major exporters in the Arabian Gulf area and import markets in the Far East. These shipments are often under long-term contracts, creating highly volatile demand in the spot markets. Adland et al. (2008) states that changes in the LPG spot market demand may be dictated by the presence of geographical price arbitrage, supply disruptions or surplus gas production. Other markets are the exports from the oil and gas companies in the north of Europe and the Gulf of Mexico, as the USA has gone from an importer of LPG to a net-exporter. This mainly caused by the effect of the shale gas revolution Dooho (2013). Engelen and Dullaert (2010) affirm that due to an inelastic demand function in the LPG market, every small change in the market balance will, in turn, impel large adjustments in rates. They also state that the market would observe nonlinearities or volatility clustering in the very short run.

The research done by Kavussanos and Nomikos (2003) investigates the causal relationship between the freight futures markets and the spot market in dry bulk shipping. They find that futures prices tend to discover new information more rapidly than spot prices, thus utilizing futures data might provide valuable forecasting information in addition to features extracted from AIS data. This study also employs multivariate forecasting methods as they find evidence in favor of using the co-integrated relationship between spot freight rates and forward contracts in prediction spot prices. The study by Batchelor et al. (2007) states that even though shipping is a non-storable service, meaning that the forward price is not tied to the spot by any arbitrage relationship, they find forward rates to help in forecasting spot prices.

In shipping theory, when freight rates are low or the oil price high, the operators reduce sailing speed to reduce operational expenses (Ronen, 1982). In the dry bulk shipping market, shipowners tend to speed up when rates are high to get the most out of the favorable market conditions (Tsioumas, 2016). Like dry bulk shipping, the LPG shipping market is also dominated by spot cargoes, and one can assume that many of the same dynamics are observed likewise.

Concerning forecasting in maritime economics, adequate research has been done in trying to forecast crude oil or dry bulk indices, prices and freight rates. Barely any of these studies use AIS data. They mainly focus on customs data, shipping indices, and other data. Han et al. (2014) presents an improved Support Vector Machine (SVM) model to forecast dry bulk freight index (BDI). This work compares forecasting results of three other forecasting methods and concludes that the proposed method has higher accuracy in forecasting the short-term trend of the BDI. Yu et al. (2008) proposes an empirical mode decomposition (EMD) based neural network to forecast world crude oil spot price. The results from the estimation of West Texas Intermediate (WTI) crude oil spot price demonstrate the attractiveness of the proposed method. The study done by Li and Parsons (1997) shows that neural networks significantly outperforms traditional time series models, like the Autoregressive Integrated Moving Average (ARIMA) or Autoregressive Conditional Heteroskedasticity (GARCH) models, in forecasting oil tanker freight rates. Xia Zhang et al. (2015) forecast the price of chemical products with multivariate data. They use neural network algorithms in addition to a machine learning model using crawled web data related to chemical products and expert experience data. Compared with traditional prediction models, the results using multivariate data has higher accuracy. In the work of Gao and Lei (2017), neural networks are implemented to predict crude oil spot prices. In their work, they developed a new stream learning algorithm and states that this is advantageous because it can handle a continuous data stream for a non-stationary process. The model introduced is a basic univariate neural network model that only considers previous price history. Kulkarni and Haidar (2009) present a methodology for forecasting crude oil price using artificial neural networks and commodity futures prices. They test both the traditional feed-forward neural network, a multilayer perceptron (MLP), and recurrent neural networks (RNN). Their findings were favorable towards the traditional networks as recurrent networks took very long time to converge. The goal of their work were risk management as they focus on the directional accuracy of the prediction.

What remains to be done?

From the literature review, it became clear that the use of information extracted from AIS data to forecast in maritime economics is a relatively unexplored field. To our knowledge, only one study utilizes this relationship. AIS data in the context of maritime economics is a relatively recent development, and the literature review shows that even though several methods and applications already are explored, opportunities to expand the area are present.

The LPG ocean freight rate market as a topic has hardly been subject to academic research, thus creating opportunities for the author. The market characteristics of the LPG, being supply-driven and spot dictated by the presence of geographical price arbitrage, implies that looking in to global ship positioning and vessel operation may provide valuable insight in forecasting. The spot market is one of the most volatile markets, and there exists a considerable uncertainty about the future development of these prices.

Furthermore, machine learning methods, especially neural networks, show promise in forecasting short-term movements in shipping commodities. Notably, also in forecasting multivariate data and nonlinear relationships, found in the ocean freight markets. Thus, the contribution of this thesis is valuable as it may provide important decision support, potentially leading to profitability for both shipowners and operators.

1.3 Objectives

The main research question of this thesis is to investigate whether multivariate machine learning forecasting methods using features extracted from AIS data adds additional information in predicting short-term freight rates. This objective is two-faced, and incorporates the two following questions:

1. Does AIS data add additional information in predicting short-term freight rates?
2. Can machine learning models achieve better forecasting prediction in predicting short-term freight rates, than traditional models?

To address the goal and main research question, we identify five research objectives that are addressed in this thesis. These objectives are characterized in the context of being applicable for predicting freight rates:

1. Extract and filter relevant market specific vessels from global AIS data.
2. Explore vessel movements and patterns to get market insight.
3. Create and identify valuable features extracted from AIS data through feature selection methods.
4. Develop machine learning prediction models to predict freight rates using multivariate data.
5. Evaluate the prediction models with traditional models used in ocean freight markets.

1.4 Scope and Limitations

The forecasting scope of this study is week-to-week one-step-ahead forecasts. Meaning that for every time step we know the current observation and try to predict the next step. The reason for choosing a weekly forecast is thoroughly reviewed in Section 2.2.

The focus of this study is on the potential added information by including AIS in predicting spot prices with machine learning methods, and not to make discoveries in the field of machine learning. Therefore, analyses of the underlying mechanics of the algorithms are also considered outside the scope of this thesis.

In their work Olsen and da Fonseca (2017) found that multivariate models perform relatively better than univariate models in predict future freight rates. The only way to incorporate AIS information in forecasting is with multivariate methods since univariate only takes previous price history into consideration. The scope is therefore delimited to encompass multivariate forecasting methods. Based on the findings of Li and Parsons (1997); Gao and Lei (2017); Kulkarni and Haidar (2009) we utilize machine learning in forecasting and test against the Vector Autoregressive (VAR) model based on the findings of Olsen and da Fonseca (2017). The no-change persistence model is used as a baseline as it is frequently used in forecasting oil prices (Alquist et al., 2013). Despite the models simplicity, it appeared to be a good baseline and performed better than other heuristic approaches. The scope is therefore limited to these methods, and traditional univariate methods like the Autoregressive Integrated Moving Average (ARIMA) is considered outside the scope of this thesis.

1.5 Structure of the Report

The remainder of this report is structured as follows:

Chapter 2 will give an introduction the general methodological approach used in this project and explanation of the flow-chart in Figure 1.1. A case study is conducted to quantify freight rate prediction using AIS data and machine learning. The study is introduced in this chapter.

Chapter 3 introduces the data foundation used to create predictors and conduct the case study. Fundamentals of AIS data is addressed, before the methods for extracting and filtering relevant market specific vessels is elaborated.

Chapter 4 concerns the creation and identification of valuable features extracted from AIS data.

The chapter include exploratory analysis of the LPG shipping market and how the features created are pre processed and selected based on various feature selection methods. Results of the features selection models are included in the case study.

Chapter 5 will go through the forecasting technique and machine learning methods considered. Also, the traditional time series models are introduced, to be used as a baseline for the machine learning methods. Finally the evaluation methods are presented.

Chapter 6 conducts the case study in order to quantify whether multivariate machine learning forecasting methods using features extracted from AIS data adds additional information in predicting short-term freight rates.

Chapter 7 gives a discussion of the work done, results attained and choices made in this thesis.

Chapter 8 concludes the work done in this thesis in the light of the objectives, and recommendations for further work is elaborated.

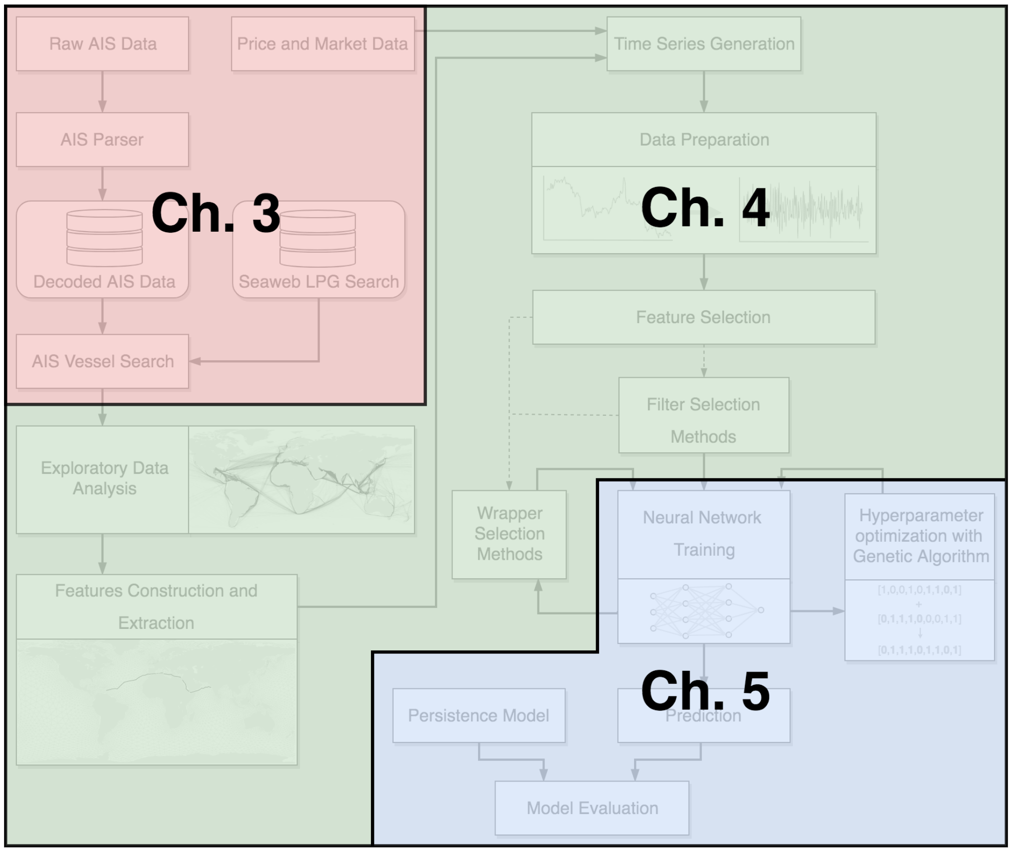


Figure 1.1: Report structure

2 | General Methodology

In the chapter, we present the general approach of the methodology in this study. Furthermore, we introduce the Liquid Petroleum Gas (LPG) shipping market case study with the forecasting horizon selected.

2.1 Methodological Approach

To go from raw AIS data to forecasting freight rates, involves many complex steps. This section will introduce these steps. Figure 2.1 presents a flow chart of the key parts and project flow. Starting from raw AIS data, the main flow follows the arrows, resulting in model evaluation.

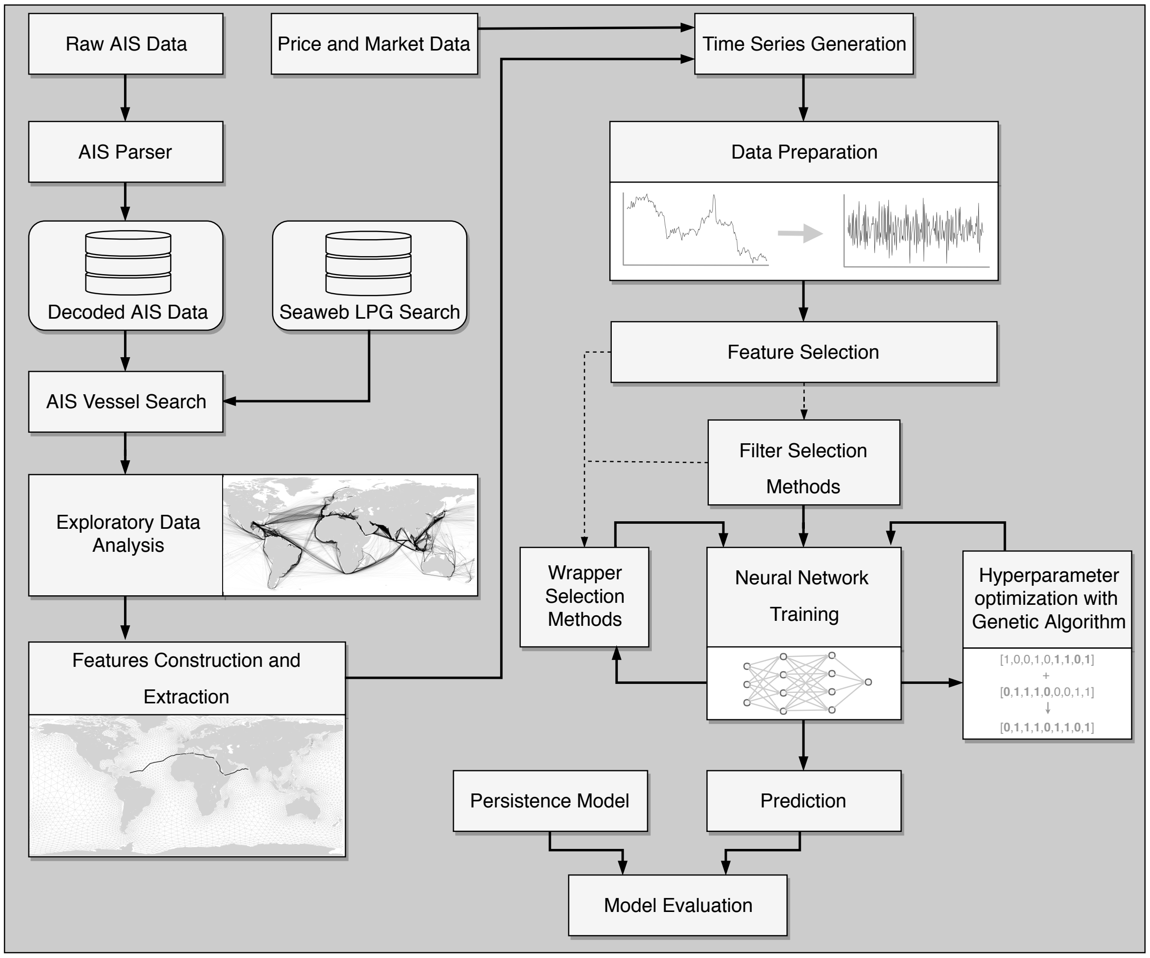


Figure 2.1: Flowchart of project flow and methodology

We have acquired global raw AIS data from the Norwegian Coastal Authorities. To investigate the objective of this study; whether multivariate machine learning forecasting methods using features extracted from AIS data adds additional information in predicting short-term freight rates, we propose the methodology overviewed in Figure 2.1. The first step is to decode the global data. Using a external AIS parser provided by Lane (2006) and a proposed decoding method by Smestad and Rødseth (2015) and Leonhardsen (2017), the AIS data can be decoded and transferred to a database. To investigate the freight rates of a specific market, a vessel search methodology needs to be developed to filter out market specific vessels. We have in this study utilized the maritime IHS' Seaweb database to identify all LPG vessel and their characteristics. Using the to be introduced vessel search methodology, we identify vessels of interest and combine the two to a reduced AIS database. This is done so that both computational time and opportunities for error is reduced.

When the data is extracted, and the search narrowed down, we can explore the data, both vessel characteristics, and movements, to get a general picture of the market and investigate patterns and potential areas of exploitation. When this is established, we can create, prepare and selected features to be used in freight rate prediction. These features are variables, also known as predictors, that are used in the multivariate models on order to forecast the prediction variable, the freight rate. Deciding which features to construct is a market specific question, underlying the importance of exploratory data analysis. The approach taken in this study is to generate a significant amount of features that are thought to be of interest based on previous literature and exploratory analysis.

With a significant amount of AIS-features generated in addition to market specific publicly available data, we can test the importance of the features in predicting the freight rate based on statistical filter methods. These methods select features independently on the model that shall subsequently use them but are good at pinpointing important features. The next step in the approach is wrapper selection. Wrapper methods select the best subset of features, taking into account the model that shall use them. Testing all subset combinations for all features in a neural network is very computationally expensive. Therefore, the proposed method is to test the best subsets from the filter selection methods. Based on the features with high scores, different combinations of these can be tested.

With the subsets generated, we can try different combinations in forecasting the freight rate. We utilize the sliding window forecasting technique. This method is the basis for how any time series dataset can be turned into a supervised machine learning problem. The methodology introduced is a supervised learning approach since the inputs (features) and outputs (freight rate) are known, and the objective is to discover a relationship between the two (Shapiro, 2003). To forecast the freight rate, we use artificial neural networks based on its ability to model complex problems, being a mapping model viewed as non-parametric, non-linear and assumption-free, meaning that it does not make a priori assumption about the problem (Kulkarni and Haidar, 2009). The models are mainly chosen based on their promise in forecasting shipping commodities (Li and Parsons, 1997; Gao and Lei, 2017; Kulkarni and Haidar, 2009). Precisely, we test the basic Multilayer Perceptron (MLP) network and the Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN). To create the best neural network model we utilize an external optimization model. Namely, a genetic algorithm to optimize the network architecture and how many lags of the features to include. Miikkulainen et al. (2017) states that this method may be suitable for the suggested approach, and Shapiro (2003) describes that a genetic algorithm is an intelligent approach to trial and error.

To potentially identify the added performance of including AIS features in forecasting the spot price, the models developed will be executed for both a subset including AIS features and a subset without any features extracted from AIS data, but only previous price and market history. In this way, the value of the added information might be seen through the forecast error and accuracy metrics. On the other note, to potentially identify the added performance of using machine learning methods over traditional methods, we compare the results with a Vector Autoregressive (VAR) model to baseline the results. This model is used by Olsen and da Fonseca (2017) to investigate the predictive ability of AIS data in the case of Arabian Gulf oil tanker rates. In addition to the VAR-model, we baseline all the results of the models above to the no-change persistence model. The persistence model is frequently used in forecasting of oil prices (Alquist et al., 2013). Despite the model's simplicity, it appeared to be a good baseline and performed better than other heuristic approaches.

In order to quantify the performance of the proposed methodology, a case study is conducted and defined in Section 2.3. To perform the analysis in this project, the Python programming environment is utilized. Several scripts have been created to carry through the complex calculations. The contents of these scripts are overviewed in Table 2.1 and further described in the following chapters and Appendix E.

Table 2.1: List with description of Python scripts created in this study

Name	Description
MASTER.py	Serves as a dashboard for selecting analyses of interest. All files and functions except the prediction models and model evaluation are ran from this script.
AIS_Analysis.py	Extraction of AIS data from the databases, feature construction and time series generation is done in this script. Includes several plotting functions in order to effectively preform exploratory data analysis.
Seaweb2AIS.py	Vessels search for LPG vessels in global AIS data in addition to plotting of fleet characteristics.
Interpolate_data.py	Daily interpolation of AIS data for every vessel identity, if applicable.
Ocean_mesh.py	Creation and plotting of an ocean mesh of nodes to calculate shortest sailing distance between two arbitrary points using Dijkstra's shortest path algorithm.
Price_data.py	Importing and cleaning of price and market data into a structured form.
Data_preparation.py	Data preparation of generated time series so that it is applicable to supervised learning problems.
Feature_importance.py	Incorporates the filter selection methods, where features are ranked based on a statistical scores from various variable importance methods.
Subset_selection.py	Contains linear subset selection methods where subset of features are tested linearly to get the best subset combination.
MLP.py	Training, prediction and construction of a Multilayer Perceptron (MLP) model with hyperparameter optimization using a genetic algorithm.
LSTM.py	Training, prediction and construction of a Long Short-Term Memory (LSTM) model with hyperparameter optimization using a genetic algorithm.
VAR.py	Training, prediction and construction of a Vector Autoregressive (VAR) model to baseline neural network prediction.
Model_evaluation.py	Evaluation of LSTM, MLP and VAR models with performance metrics, a persistence model baseline and forecast visualization.

2.2 Forecasting Horizon

It is an essential issue to asses the right horizon for forecasting and horizon to extract observations. Previous studies forecasting ocean freight rates vary from daily to monthly forecasting windows. To use AIS-features in predicting monthly freight rates, in a month a vessel may have sailed a long distance. Accordingly, looking at average vessel positioning within a month might give misleading or averaged out effects. Besides, in the context of machine learning prediction, the models predict better with more data (Sun et al., 2017). Therefore, by investigating monthly or quarterly price fluctuations, the number of observations would become very low due to the short time span of AIS data relative to the history of the ocean shipping industry.

To investigate daily fluctuations might also be insufficient as the AIS data quality used in this study have some gaps and errors, thus looking at daily vessel positioning may exclude some vessels for which AIS data was not recorded this specific day. This issue was investigated in the initial stages of the study by interpolating the AIS data to daily observations. These types of gaps are common in the early years of S-AIS data before 2013 as the number of satellites used to collect data was low in addition to data interference (Eriksen et al., 2010). Hence, forecasting daily fluctuations in price was not seemed fit for this study.

Freight rates and shipping commodity prices, in general, are very volatile in nature (Stopford, 2009). Especially LPG relative to other ocean freight markets, because of the low number of main trading routes and the comparatively small fleet (Adland et al., 2008). This makes for large fluctuations in price, hence a shorter forecasting window seems fit. Accordingly, one-step-ahead weekly predictions are investigated in this study. The only other study utilizing AIS data for freight rate prediction (Olsen and da Fonseca, 2017) also employ this forecasting horizon. From this point, to maintain coherence all through the thesis, the time series data examined are stated as weekly time steps.

2.3 Case Study Definition

To investigate multivariate freight rate prediction with machine learning and AIS data, a case study has been carried through. The goal of the case study is in line with the objectives of the work. Specifically, to investigate if including features extracted from AIS data adds additional information in predicting short-term freight rates. Also, we will explore if machine learning models can provide additional forecasting performance over traditional time series models. To be able to forecast the short-term fluctuations in rates makes for better decision support on contracts and vessel operation, which may provide higher profitability for both shipowners and operators.

The commodity we will investigate is the Mont Belvieu propane spot price. In addition to the propane spot prices in the Arabian Gulf, the North of Europe and the Far East, the Mont Belvieu spot price is one of leading market indicators in the LPG shipping market. To forecast the Mont Belvieu propane spot price, we will utilize both market and spot prices data in addition to extracting new features from AIS data and test prediction model performance on datasets including AIS data and sets without AIS data. From this point on, these are denoted as AIS features and non-AIS features, respectively. We assess this problem by analyzing the spot price and AIS data from May 2011 to December 2017. Both the machine learning and traditional models will be fitted to 90 % of the dataset and tested on the last 10 %, called the in-sample and out-of-sample dataset. In machine learning, it is common to refer to these as the training and testing sets, where we identify and fit the best models based on the training set, and test the performance on the testing set to represent the model's performances on unseen data.

This thesis will have a general approach, with a separate focus on the case study. However, deciding which features to construct is a market specific question. Thus, the case study will be referred to throughout this report. We will introduce the data in Chapter 3, methods for creating and extracting features in Chapter 4, forecasting models in Chapter 5 and finally use all this to make predictions in Chapter 6.

3 | Data Foundation

This chapter introduces how we can proceed from having raw AIS data from the world fleet to a filtered database including only vessels of interest. The process of developing this database includes decoding raw data, identifying vessels of interest and combine the two to a reduced database. This is done so that both computational time and opportunities for error is reduced. Besides, this chapter also introduces the various market specific data used to complete the LPG case study. Mainly, this chapter will introduce the data, and Chapter 4 will explain how this data is utilized to extract relevant features.

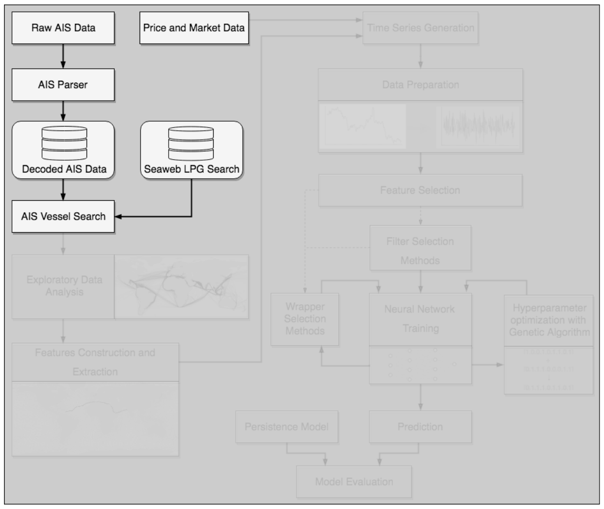


Figure 3.1: Sub-flowchart of project flow and methodology in *Data Foundation*

3.1 AIS Data

3.1.1 Introduction to AIS Data

Automatic Identification System (AIS) is an automatic tracking system based on Very High Frequency (VHS) system, installed on more or less all merchant vessels. AIS Data from vessels can be exchanged with other vessels nearby, AIS base stations and satellites (S-AIS). The information within AIS messages includes static data such as navigational data, dynamic data such as speed, and voyage related data such as draught and estimated time of arrival.

AIS was developed with the purpose of enhancing safety, more specifically to avoid collisions. It remains as a supplement to the marine radar, which is considered the primary instrument to avoid collisions. The AIS technology itself was developed in the 1990s, and from the early 2000s, it became mandatory to have AIS on board most vessels. The International Convention for the Safety of Life at Sea, IMO (1974), requires AIS to be fitted aboard all ships of 300 gross tonnages and upwards engaged on international voyages, cargo ships of 500 gross tonnages and upwards not engaged on international voyages and all passenger ships regardless of size. Around 2008 S-AIS was introduced, meaning that satellites can receive the messages in addition to base stations and other vessels. AIS messages can only reach around 70 kilometers horizontal at sea level, but up to 400 km in vertical direction (Skauen et al., 2013). This enabled the collection of the data with low orbiting satellites and allowed a more coherent investigation of marine traffic.

3.1.2 Message Types and Content

The International Telecommunication (ITU) has defined 27 different AIS message types (Itu-R, 2014), and the five most common ones can be found in Table 3.1. Some of the information included in message type 1 is presented in Table 3.2. According to Smestad and Rødseth (2015), message type 1 contributes to 72,5 % of all AIS messages. Message type 5, which include static vessel and voyage information, is presented in Table 3.3.

Table 3.1: Message types, AIS

ID	Name	Description
1	Position report	Scheduled position report
2	Position report	Assigned scheduled position report
3	Position report	Special position report
4	Base station report	Position, UTC, date and current slot number of base station
5	Static and voyage report	Scheduled static and voyage related vessel data report

Table 3.2: Key features in Message type 1

Information	Description
Unixtime	Number of seconds elapsed since 1 January 1970
Position	Coordinates, longitude and latitude
Speed	Speed over ground (SOG) in knots
Course	Course over ground (COG)
MMSI	Maritime Mobile Service Identity (Vessel ID)
Status	Navigational status

Table 3.3: Message type 5

Information	Description
Unixtime	Number of seconds elapsed since 1 January 1970
Vessel specifications	Length and breadth, in meters
Draught	Current draught in meters
IMO Number	International Maritime Organization number
Origin	Origin of current voyage
Destination	Destination of current voyage
ETA	Estimated time of arrival, in Unixtime
MMSI	Maritime Mobile Service Identity (Vessel ID)
Ship type	Vessel type category

The way the system identifies unique vessels is through the Maritime Mobile Service Identity (MMSI). The number is connected to the AIS-gear onboard the vessel, and only change if ownership changes. Another identification number is the IMO number which was made mandatory by IMO (1974). All ships over 100 gross tonnage, with exception to vessels solely engaged in fishing, ships without mechanical means of propulsion, pleasure yachts, ships engaged on special service, hopper barges, hydrofoils and hovercrafts, floating docks, ships of war and wooden ships, should be identified with an IMO number. The IMO number can be found in static AIS messages while the MMSI is in all as it is the prime identification number in the system.

The AIS ship type described is reported as a double-digit number between 10 and 99. The first digit represents the ship type, as seen in Table 3.4. The second digit represents whether a cargo is dangerous, hazardous or a marine pollutant.

Table 3.4: First digit representation of ship types

First Digit	Ship Type
1	Reserved for future use
2	WIG (Wing In Ground)
3	Other vessels
4	High-speed carrier, or vessels < 100 Gross Tonnes
5	Special craft
6	Passenger ships > 100 Gross Tonnes
7	Cargo ships
8	Tankers
9	Other types of ships

The frequency of the AIS messages varies with different intervals. Static and voyage data is sent every 6 minutes or upon request, but dynamic data is sent according to speed and operational status. The different intervals can be found in Table 3.5

Table 3.5: Dynamic AIS data and their general reporting intervals

Vessel Operational Status	**General reporting interval**
Vessel at anchor	3 min
Vessel at 0-14 knots	12 sec
Vessel at 0-14 knots and changing course	4 sec
Vessel at 14-23 knots	6 sec
Vessel at 14-24 knots and changing course	2 sec
Vessel at > 23 knots	3 sec
Vessel at > 23 knots and changing course	2 sec

AIS data was traditionally collected using land-based receivers able to detect messages up to 40-50 nautical miles off-shore (Skauen et al., 2013). Messages outside this area would not be detected. A solution to this is to utilize satellites to collect the messages. This poses some problems further described in Section 3.1.3. Messages collected with satellites, known as S-AIS data, are collected on a worldwide scale. The Norwegian Coastal Authorities currently have 4 satellites (Norwegian Space Centre), AISSat-1 (launched 2011), AISSat-2 (launched 2014), NorSat-1 (launched 2017) and NorSat-2 (launched 2017) collecting data. The combination of data from these satellites is used in this project.

Detailed information on AIS messages is given in Table A.1, A.2 and A.3, in Appendix A, for static, dynamic and voyage related messages respectively. The data content is given by 'Guidelines for the onboard operational use of shipborne AIS' by IMO (2016).

3.1.3 AIS Data Quality

There are several issues to discuss when it comes to the quality of the AIS data. Some of the most important aspects of this will be discussed in this section. This is also covered extensively by Næss et al. (2017), Smestad and Rødseth (2015) and Leonhardsen (2017). This section will cover quality issues related to S-AIS data, general imperfection with AIS data and some human errors.

Smestad and Rødseth (2015) points out that the variations in traffic from different time periods can have increased coverage in an area and therefore the traffic density may look higher. Eriksen et al. (2010) state that over a time span of 24 hours, the High North and South is covered up to 15 times, while the areas around the equator are covered around two to three times. With the launch of AISSat-2, the coverage was extended. The newly launched NorSat-1 and NorSat-2 will able the Norwegian Coastal Authorities to detect 60 % more vessels and collect 2.5-3 times more data than the previous use of only AISSat-1 and AISSat-2. The satellites can also have interference problems. A satellite will have a much larger coverage area than the AIS system of receivers were designed for, so high traffic areas would cause problems. Combined with low orbiting rates over the area, there could be significant gaps in the data.

There are other possible sources of errors than the ones discussed regarding the satellites. These errors can either be caused by a failure in the automatic reports or by human errors. Regarding the former, Smestad and Rødseth (2015) discovered that there were several thousands of vessels that had at least some erroneous data. This includes, for instance, wrong IMO numbers. However, this only affects the static messages, so the total number of distinct IMO numbers does not reflect the total number of vessels present in the S-AIS data. Leonhardsen (2017) discovered that the total number of unique MMSI numbers in the database exceeded the total number of vessels in the world fleet at that time. This may be caused by vessels changing owners over the period for the data set. Other errors may include wrongly reported ship dimensions and erroneous ship positions.

There are several kinds of human errors with regards to the AIS data. This mainly includes manually reported data. The manually reported data include for instance the draught, destination, ETA, route plan and navigational status (see Appendix A for more static data feature). To illustrate these errors we look at the navigational status. The crew can set the status to 1 or 5 when a vessel is not moving, meaning *at anchor* and *moored*, respectively. It happens that the crew forgets to change the status while sailing. This can be seen in Figure 3.2, where the data plotted is for speeds above 5 knots and navigational status 1 or 5, for an arbitrary set of ships.

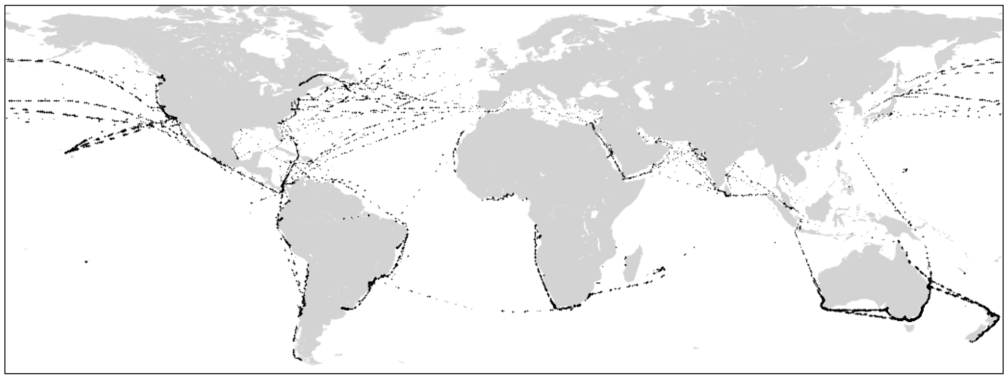


Figure 3.2: Vessels sailing with speeds above 5 knots and navigational status 1 or 5

3.1.4 Decoding Data

The AIS data used in this study was granted by the Norwegian Coastal Authorities. The data was provided in two stages. Firstly, raw S-AIS data form the period 2011-2015, also utilized by the two graduated naval architects Smestad and Rødseth (2015) and Leonhardsen (2017), was acquired. Secondly, S-AIS data in comma-separated files form the period 2016-2017 was attained, as the data was already decoded by the Norwegian Coastal Authorities.

Smestad and Rødseth (2015) developed a Python script used to extract data to an SQLite database by an external AIS parser provided by Lane (2006). The parser decodes raw AIS messages into logic information based on specified patterns and notations. An arbitrary AIS message is visualized in Table 3.6. To use the AIS parser and the python script by Smestad and Rødseth (2015), an Ubuntu operating system was utilized, and decoded files were extracted to an external hard drive. All the data handling, analysis, and visualization in this project are done using Python. SQL is used to create and extract data from the database created.

Table 3.6: Structure of a raw AIS data message

Arbitrary message
/s:ASM//Port=669//MMSI=,c:1439078400*7D/!BSVDM,1,1,,A,1:UF6@001OMO>0au3G5mbDT2081<,0*5E

3.2 Price Data

The price and market data utilized in this study was granted by Equinor. This data includes spot prices of propane for the four major LPG markets, Mont Belvieu in the Gulf of Mexico (Propane MB), North West Europe (Propane CIF NWE), the Arabian Gulf (Propane CP) and the Asian market (Propane FEI). The data is in USD/mt and contains daily reported prices from 2011-2017.

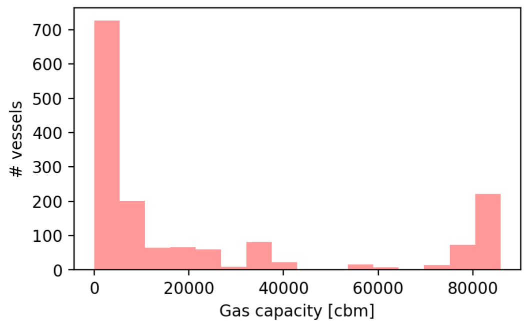
In addition to propane spot prices, the price of crude oil is also used in this study. Based on the fact that the LPG market is driven by “supply push” rather than “demand pull” in the sense that LPG is derived as a by-product of natural gas processing and crude oil refining (Engelen and Dullaert, 2010), it was deemed fit to include the price of crude oil in the prediction model. Specifically, as the case study investigates Mont Belvieu propane prices, the West Texas Intermediate (WTI) Cushing, Oklahoma crude oil price is utilized. The data was acquired from the U.S. Department of Energy (2018), with daily reported prices in USD/barrel from 2011-2017.

3.3 Fleet Data

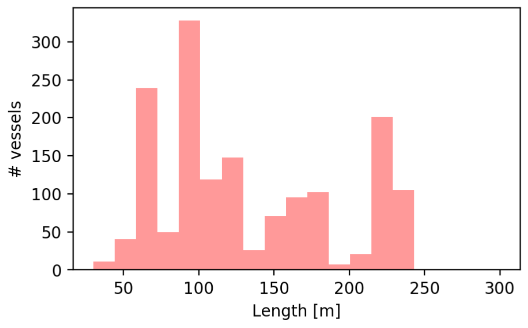
Smestad and Rødseth (2015) show how to use heuristics to establish specific ship type, with sole use of AIS data. The purpose of predicting ship type without additional data is to avoid the cost of acquiring commercial ship data. In their work, S-AIS data was used as a basis to create the heuristics, and a data cleaning process is carried out to exclude vessels that have conflicting and inaccurate data. To verify the accuracy of the heuristics, AIS data was matched with data from Clarkson's Ship Register. In their study, large commercial gas carrier, both LPG and LNG, was characterized as a single group with 99 % accuracy. First of all, the cargoes transported by LPG and LNG carriers are two different commodities, and most of the fleet can not switch cargoes based on the vessels technical design. Secondly, in our work, no significant characteristics were found to separate LPG form LNG carriers. Thirdly, in the study of Smestad and Rødseth (2015), the vessels investigated was in the size of 270-345 m in length. In fact, there are also a lot of smaller LPG vessels trading on shorter routes. This is further reviewed below and in Section 4.2. Therefore, to obtain information about the LPG fleet, external data is needed in addition to AIS data. The maritime IHS' Seaweb was used to collect this material, in addition to a new method to identify the LPG fleet.

3.3.1 Vessel Search

Using maritime IHS' Seaweb database, we can find all LPG vessels and their characteristics. The database includes some no longer operating dead and demolished vessels. Extracting the vessel information from Seaweb, disregarding vessels built in 2018 and future new-buildings, the search results in 1469 unique vessels. From the plots in Figure 3.3, we can see that the LPG fleet is dominated by smaller vessels with gas capacity in the range of 1.000-10.000 cbm. Thus, the approach by Smestad and Rødseth (2015) is not applicable to this project. Another group that stands out is the Very Large Gas Carriers (VLGC) with a gas capacity larger than 60.000 cbm. Exploring the trade patterns of these vessels are further investigated in Section 4.2.



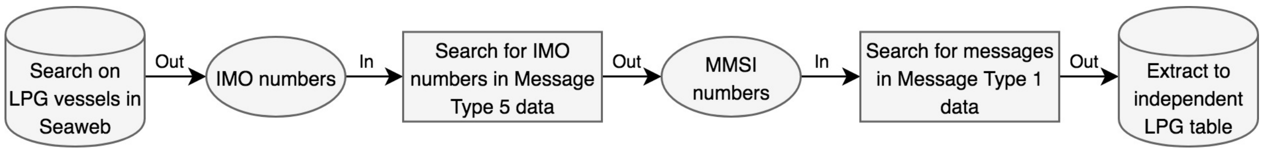
(a) Gas capacity distribution



(b) Length distribution

Figure 3.3: Characteristics of LPG vessels from Seaweb

Concerning vessel identification, the database includes the IMO number for every vessel and MMSI for almost every vessel. As stated above, the MMSI is the prime form of identification in AIS signals, and the IMO number is only included in Message Type 5. For the analysis to be correct and complete, we need to know all the current vessels in the fleet as well as all previous vessels. To do the analysis, we need to know each vessels MMSI at each specific time point. If a vessel change ownership, the MMSI might change. Therefore, only using the MMSI numbers from Seaweb will give a faulty vessel set. The method introduced here is based on using the fact that IMO numbers follow the hull of the vessel through its time span, to find old MMSI numbers. The methodology is sketched in the flowchart in Figure 3.4.



Search on LPG vessels in Seaweb
Out
IMO numbers
In
Search for IMO numbers in Message Type 5 data
Out
MMSI numbers
In
Search for messages in Message Type 1 data
Out
Extract to independent LPG table

Figure 3.4: Flowchart of vessel search methodology

Firstly, with a quick vessel search in the static Message Type 5 database table, we can see in Figure 3.5 that a single IMO number might correspond to several MMSI during a time span of several years. This is done with the Python script *Seaweb2AIS.py* in Appendix E.3, using the following SQL query:

```
SELECT imo,userid,ship_type FROM MessageType5
    WHERE imo IN {IMO_LIST} AND ship_type >= 80 AND ship_type < 90
```

Where IMO_LIST is the list of IMO numbers acquired from Seaweb. LPG vessels are characterized as tankers with the first ship type digit being 8. This is used as a safety-net in case of erroneous data and interference issues.

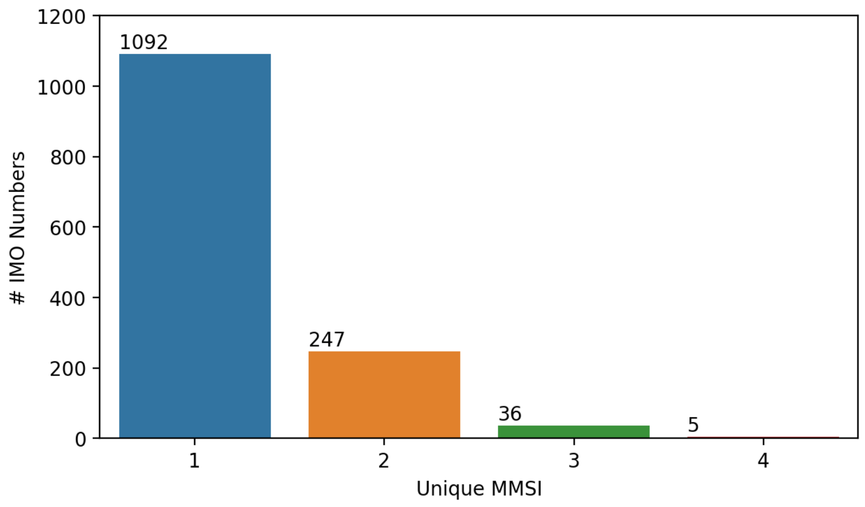


Figure 3.5: Number of unique MMSI numbers connected to an IMO number

The result of this vessel search is a set of 1714 MMSI numbers corresponding to 1469 IMO numbers. The list of these MMSI numbers, MMSI_LIST, are used to extract the Message Type 1 signals with corresponding MMSI. The resulting messages from this analysis are extracted to a standalone database with only LPG data with the query below. This database is described below in Section 3.3.2.

```
CREATE TABLE LPG1 AS
    SELECT * FROM MessageType1
        WHERE userid IN {MMSI_LIST}
```

3.3.2 LPG AIS Database

The method used above in addition to the comma separated file with newer AIS data results in 33.958.372 Messages Type 1 messages from 01/01/2011 to 31/12/2017. Due to some gaps in the data, the horizon used in the case study is from May 2011 to December 2017. The S-AIS Message Type 1 data are plotted in Figure 3.6.

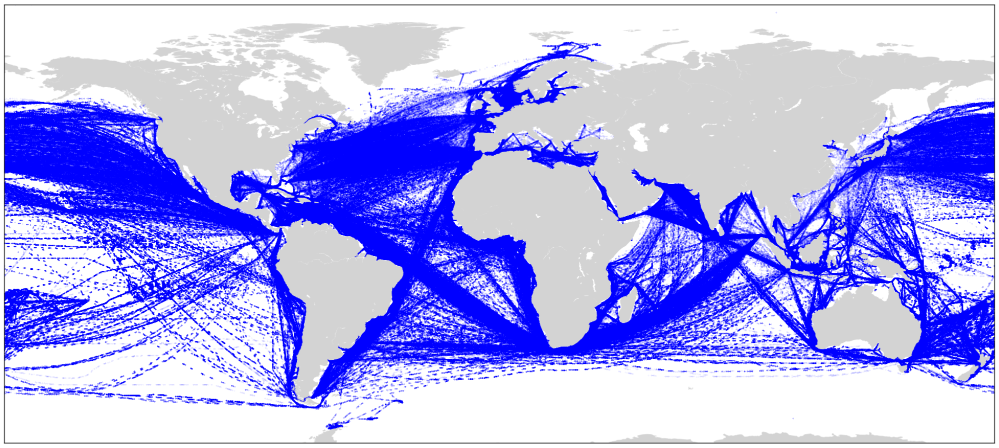


Figure 3.6: AIS Message Type 1 signal plot, 2011-2017

When extracting data from the database it is essential only to gather what is needed in the analysis, or else it will have a significant influence on the running time. It is also important to point out that sorting data with SQL is also significantly faster than sorting in Python. So when searching for vessels in time and space, a continuous focus has been on sorting as much as possible in SQL before the analysis is done in Python. The following SQL query setup is used throughout this study for data extraction:

```
SELECT unixtime,sog,latitude,longitude,userid,nav_status FROM LPG1
    WHERE longitude <= max_lon AND latitude <= max_lat
        AND longitude >= min_lon AND latitude >= min_lat
        AND sog >= min_speed AND sog <= max_speed
        AND unixtime >= min_time AND unixtime <= max_time
        ORDER BY userid,unixtime ASC
```

where max_lon, max_lat, min_lon and min_lat defines the area of interest based on maximum and minimum latitude and longitude coordinates. min_speed, max_speed, min_time and max_time defines the time and speed window of interest respectively.

4 | Feature Engineering

This chapter we will introduce how the process of feature engineering have been carried through in addition to construction features relevant to the LPG case study. This chapter goes through the methodology for creating, preparing and selecting these features, while the results of the feature selection methods are presented in the case study in Chapter 6.

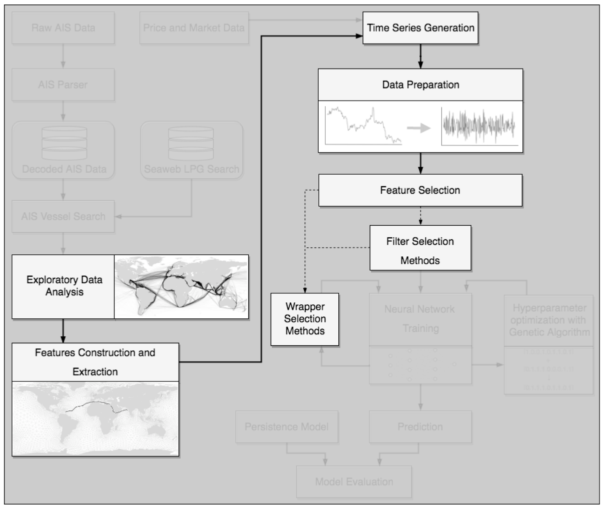


Figure 4.1: Sub-flowchart of project flow and methodology in *Feature Engineering*

4.1 Feature Engineering Methodology

In machine learning and statistics, feature engineering is the process of developing features from the AIS data to be used in machine learning algorithms. These features are attributes or variables that may be important to the model and should help in understanding the context of a problem. One way to generalize feature engineering is that based on a set of features X, we want to predict Y. So the process of feature engineering is to create and select the best set X so that the model function $\hat{f}(X)$ gives the best possible prediction, $\hat{f}(X) = \hat{Y}$, of the actual Y. Feature engineering is not a formal topic but is considered by many researchers as an important and essential part of applied machine learning. Ng (2011) states that *Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.* Domingos (2015) considers that the features used in a machine learning problem are the most important factor to why some projects succeed and some fail.

In the context of AIS data, this chapter will introduce how transforming raw data into features that better represent the underlying problem, will result in improved model accuracy on unseen data. In this thesis, long time has been spent on feature engineering as it usually offers more impact on the project than the choice and tuning of models. The process of feature engineering used in this study is listed below. It is important to keep in mind that this is an iterative process. When new features are created and tested, we can extract new features from the once created and so on as long the added complexity is seen valuable through increased model performance.

1. **Exploratory data analysis**: Brainstorm features and use maritime domain knowledge to narrow down the search.
2. **Feature construction and extraction**: Create and extract features from AIS and price data.
3. **Data preparation**: Transform, normalize and scale data, so that it applies to machine learning problems. Split the data into training and testing set.
4. **Feature selection**: Selects a subset of the features created/extracted based on various statistical tests and methods.

4.2 Exploratory Data Analysis

The idea behind exploratory data analysis is to get a general picture of the data and investigate patterns and potential areas of exploitation. One of the topics in this thesis is machine learning, but the human mind is still better in understanding context. Thus, the exploratory analysis is fundamental.

The AIS Message Type 1 signal plot in Figure 3.6 is chaotic, and it is hard to see a general trading pattern in this form of visualization. Therefore, a density plot function *line_map()* in the code *AIS_Analysis.py* in Appendix E.2, is made to better visualize the sailing patterns. This code is basically a line plot between points for each vessel, where signals with large distance gaps have been cleaned out. The lines are made transparent so that layered lines, meaning areas with much traffic, stands out in on the map. A density plot with the same data as in Figure 3.6, is shown in Figure 4.2.

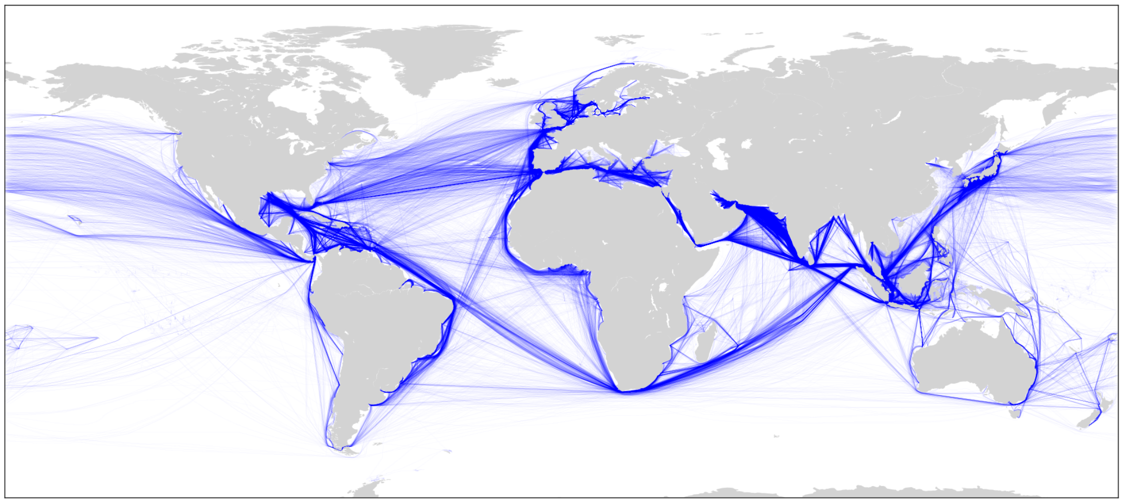


Figure 4.2: Density line plot of the LPG fleet, 2011-2017

As it became clear in the vessel search in Section 3.3.1, the LPG fleet mainly consists of two parts. The first, a large part of smaller vessels in the range of 1.000-10.000 cbm in gas capacity, and the second, the Very Large Gas Carriers (VLGC) with a gas capacity larger than 60.000 cbm. To get a better insight into the trade patterns in the LPG market, we can plot a density plot of the two vessel groups in the same map (Figure 4.3). Separate plots of these vessels segments are provided in Appendix B.1 and B.2.

We observe from Figure 4.3 that the VLGCs has fever but more concentrated longer routes than the smaller vessels. This might indicate that the VLGCs sail on long-haul routes between larger terminals, while the smaller vessels operate in the secondary markets around these hubs. A

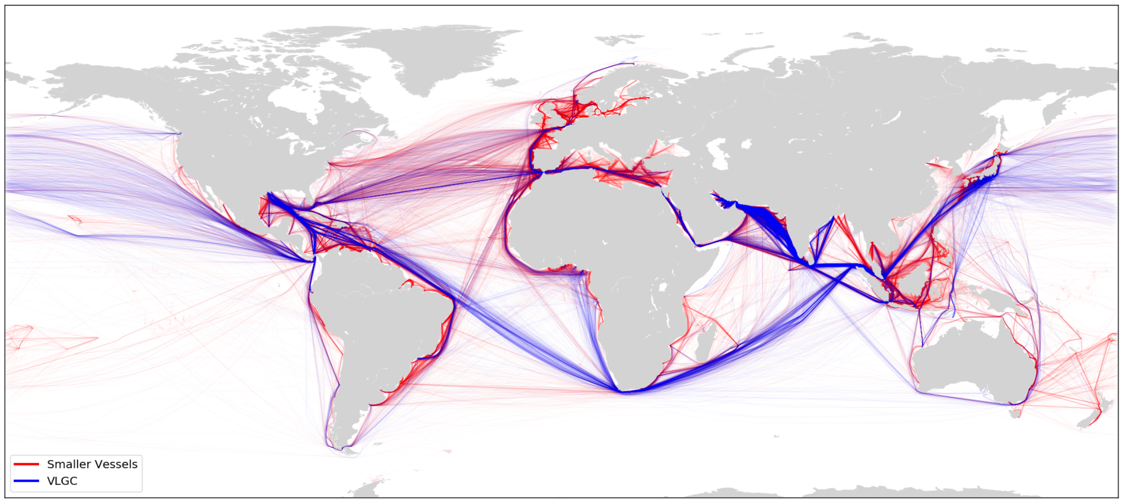
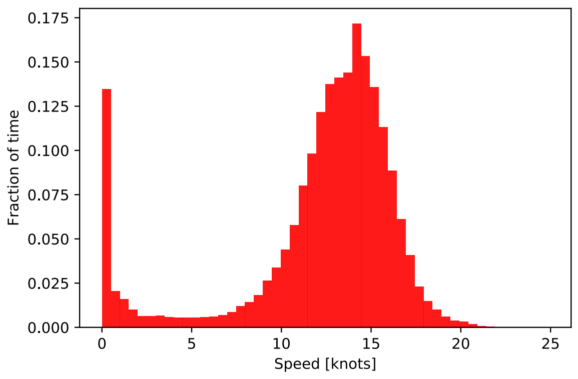
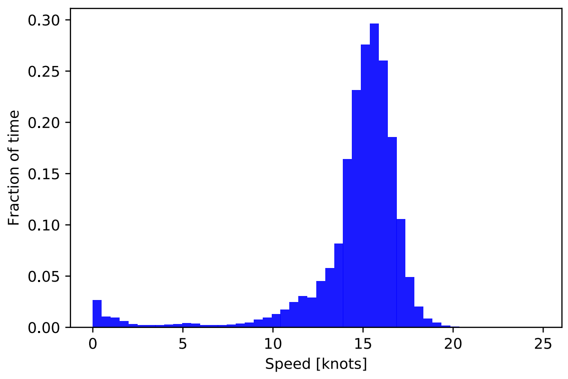


Figure 4.3: Density line plot of VLGC (>60.000 cbm) vs. smaller vessels (<60.000 cbm), 2011-2017

good example of this is seen in the Mediterranean or South East Asia. Investigating the trade patterns of VLGCs, we can observe that some areas are standing out, these are the export market in the Gulf of Mexico and the Arabian Gulf, as well as the import market in the Far East, mainly Japan. Other high activity areas include the Mediterranean, Australia, Northern Europe and South America with Brazil. The observation of longer routes for VLGCs can also be seen through plotting the vessels speed in Figure 4.4. We can observe that the VLGCs have more concentrated speed with low variance, while the smaller vessels sail over a larger specter of speeds in addition to spending more time still, not sailing. The first might be related to the larger VLGC sailing long-haul voyages, where speed is optimized. The second might be related to the smaller vessels sailing shorter routes close to shore and spending a larger fraction of time waiting for shipments. It is also important to mention that the smaller vessels have a larger range of sizes, 1.000-60.000 cbm, thus having different design speeds. However, the plot still shows that the vessels spend more time not sailing,



(a) Smaller vessels (<60.000 cbm)



(b) VLGC (>60.000 cbm)

Figure 4.4: Speed histogram

4.3 Feature Construction and Extraction

Deciding which features to construct is a market specific question, underlying the importance of exploratory data analysis. Thus creating standardized features for AIS that work for all shipping segments is not a sufficient approach. Therefore the features considered in this study are chosen explicitly concerning the LPG shipping market and the case study, but some are also applicable to other shipping segments. The subsections will start by describing the method used and finish with their application in the case study. Creating the features results in 346 weekly observations from May 2011 to December 2017.

4.3.1 Vessel and Capacity Counting in Area or Port

Adland et al. (2008) state that changes in the LPG spot market demand may be dictated by the presence of geographical price arbitrage, supply disruptions or surplus gas production. As the price of a commodity is dictated by the supply-demand relationship, looking at geographical ship positioning is of interest. This can be as easy as just counting vessels in an area or sum up the maximum gas capacity for these vessels. Data on gas capacity is provided by the Seaweb vessel database. Time series based on the number of vessels or summed up gas capacity in an area per week can be generated by constructing geo-fences to extract this information from a specific geographical restricted area. The simplest form of a geo-fence is a box where an area is restricted based on maximum and minimum latitude and longitude values. If it is desirable to geo-fence areas that do not fit for a box geo-fence, like for example the Atlantic ocean, a polygon may be used.

The most common way to decide if a point is inside a polygon or not is the *ray casting algorithm* (Sutherland et al., 1974). The principle is that a point is inside the polygon if it first and foremost lies within the extreme values of the polygon x_{min},x_{max},y_{min} and y_{max}. This is done to increase the computational speed of the analysis so that we do not need to check every global message signal. In general, the following four procedures are carried out and evaluated:

1. Check that the point (x, y) lies within the larger rectangle $[x_{min}, x_{max}, y_{min}, y_{max}]$, the extreme polygon values.
2. Draw a horizontal line in one constant direction for each point
3. Count the number of times the line intersects with polygons edges.
4. A point is inside the polygon or in the edge if the number of times it crosses the edges is odd. If the number of times it crosses is even, then the point is outside the polygon.

This is calculated by the *point_inside_polygon(x,y,poly)* function in the *AIS_Analysis.py* scriptin in Appendix E.2. An example of the *ray casting algorithm* is showed in Figure 4.5, where the algorithm have been carried forward for a Atlantic ocean polygon.

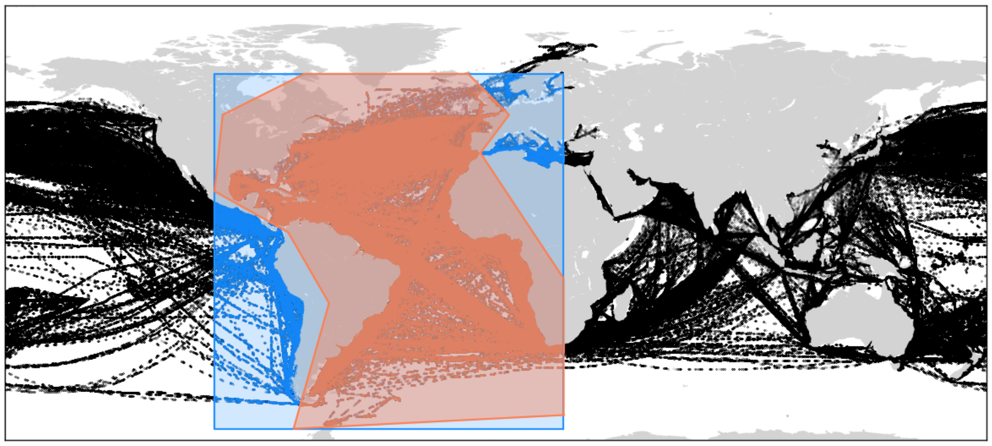


Figure 4.5: Ray casting algorithm used for the Atlantic where black is outside, blue inside extreme points but outside polygon and orange is inside the polygon

Cast Study Application

Given that the LPG spot market demand may be dictated by the presence of geographical price arbitrage (Adland et al., 2008), knowing where the ocean fleet is located might be interesting. With the method described above, we can divide the world into polygons and count vessels and capacity. The following notation is used for the oceans/areas: *Atl* (Atlantic), *FE* (Far East), *CP* (Arabian Gulf), *EstP* (East Pacific), *NWE* (North West Europe), *Ind* (Indian Ocean) and *Med* (Mediterranean). The polygons used are visualized in Figure 4.6. Furthermore, counting weekly unique MMSI in each of these polygons yields the time series in Figure 4.7.

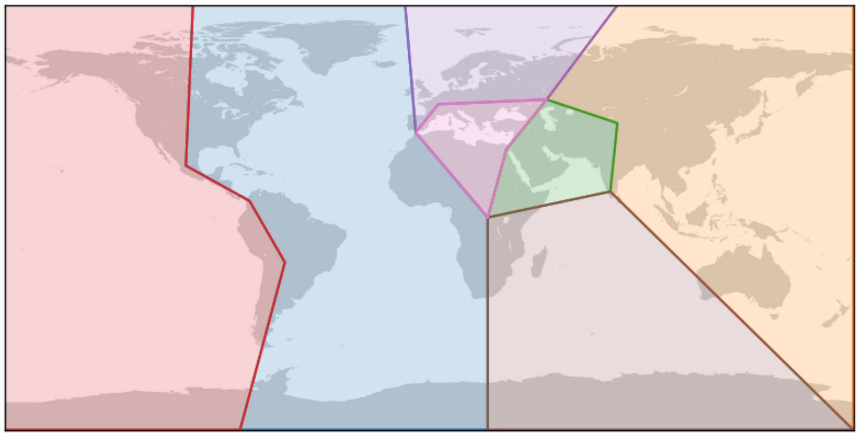


Figure 4.6: Orientation of world polygons

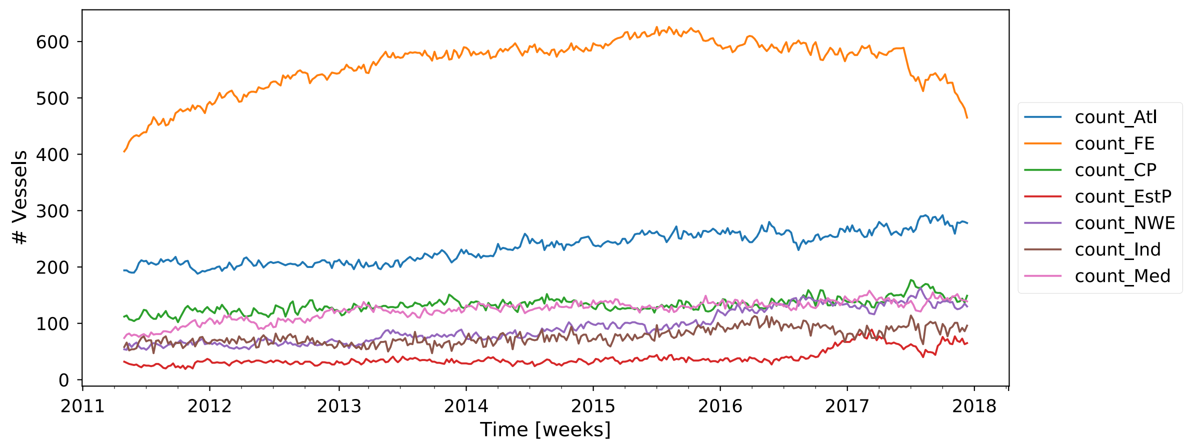


Figure 4.7: Number of vessels observed in each polygon every week from 2011-2017

We observe from Figure 4.7 that a large number of vessels is located in the Far East at all times. As discovered in Section 3.3.1, the LPG fleet consists of many smaller vessels. In Section 4.2 we explored that these two vessels groups have different trading patterns. Therefore, summing up the fleets total gas capacity in a polygon would maybe give a better picture of the supply of vessels. More specifically, the supply of fleet gas capacity in this area. Figure 4.8 show the summed up gas capacity for the vessels located in the respective polygons.

From the plot in Figure 4.8 we observe then the gap between the Far East and the rest of the polygons is relatively smaller concerning the vessel count in Figure 4.7. This observation can understate that there are many smaller markets in the Far East, thus a larger part of smaller vessels sailing from the hubs to these smaller markets over longer distances. This can also be

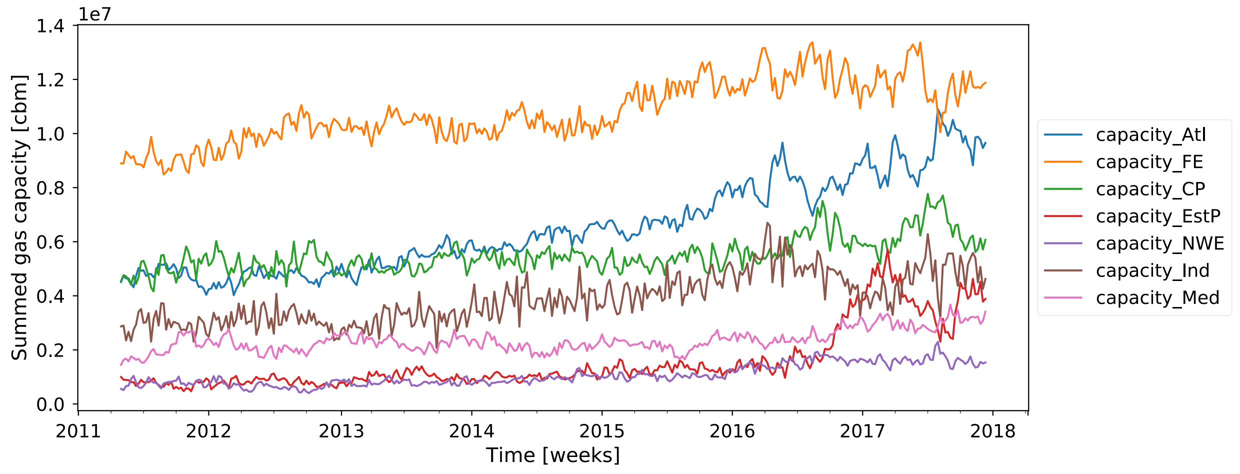


Figure 4.8: Summed up gas capacity for the vessels observed in each polygon every week from 2011-2017

observed in the density plot of the smaller vessels in Figure 4.3.

The vessel capacity and count time series gives a dynamic picture of the vessel movements. However, they do not account for the change in total fleet size. A plot of newbuilt LPG vessels is found in Figure 4.9, with data from Seaweb. Based on the relative small total fleet, the series in Figure 4.9 shows that a significant amount of vessels was built every year through the study horizon (2011-2017). It may be interesting to look at the percentage of the world fleet, or world fleet capacity, observed in the polygons each week. This is visualized in the plots in Figure 4.10 and 4.11.

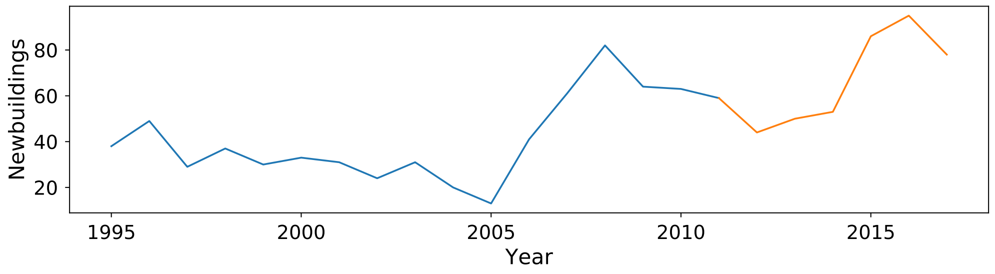


Figure 4.9: Newbuilding in the LPG market, 1995-2017, with study horizon in orange

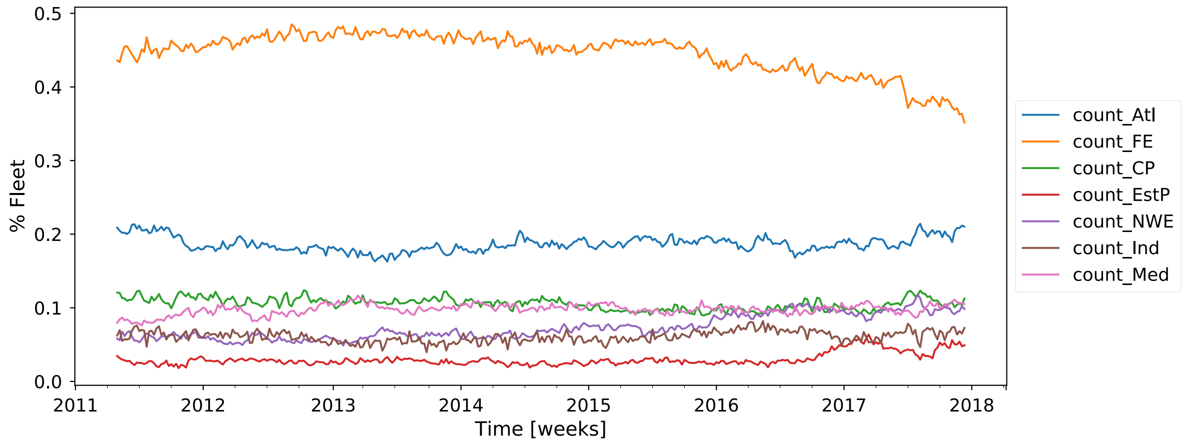


Figure 4.10: Percentage of LPG fleet observed in each polygon every week from 2011-2017

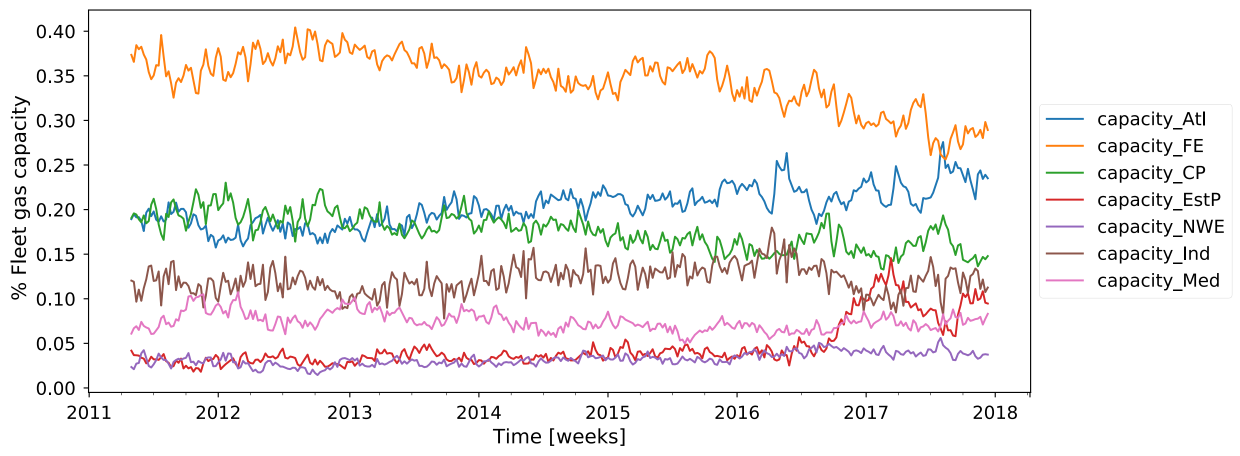


Figure 4.11: Percentage of summed up fleet gas capacity observed in each polygon every week from 2011-2017

We observe from Figure 4.8 that the overall activity, given by capacity, has increased for both the Far East and the Atlantic. This can show that the overall fleet size has increased. However, for the Far East relative to the Atlantic, the capacity has shifted in favor of the latter and other areas, which can be seen in Figure 4.11.

In addition to the polygon presented above, a last polygon is also added to the study. This polygon is the Gulf of Mexico, given by the notation *GOM*. Since the case study investigates the spot price in Mont Belvieu, looking at a separate polygon for this area was done. As the Atlantic is relatively large, it was deemed fit to investigate this smaller portion of the polygon. The same features as for the other polygons have therefore been created for the Gulf of Mexico as well. These are plotted in Figure 4.12 and 4.13.

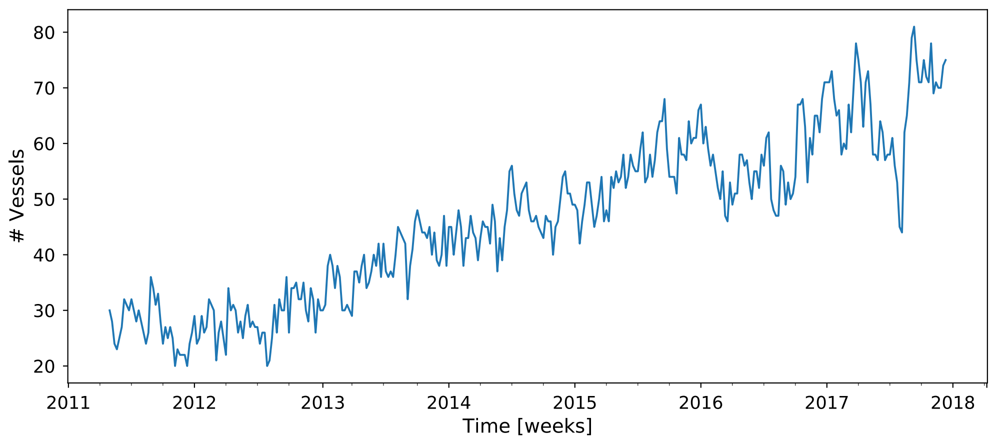


Figure 4.12: Number of vessels observed in the Gulf of Mexico polygon every week from 2011-2017

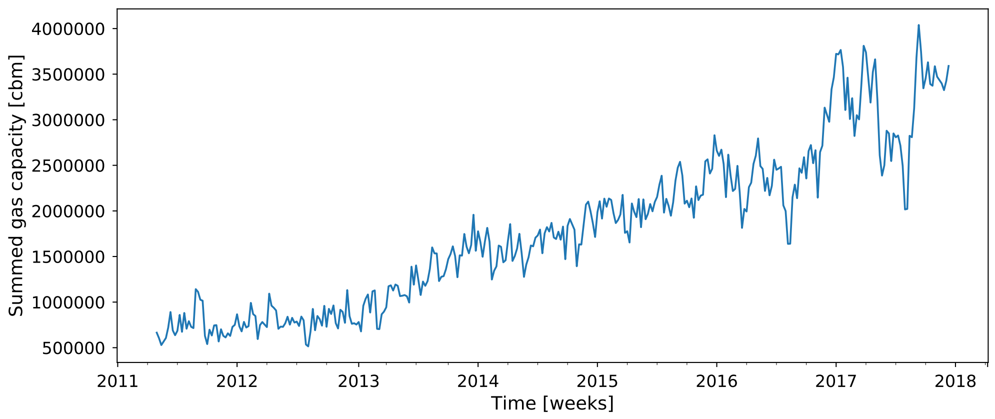


Figure 4.13: Summed up gas capacity for the vessels observed in the Gulf of Mexico polygon every week from 2011-2017

A list with descriptive statistics of all the features created in this section, can be found in Appendix C.1.

4.3.2 Sailing Distance From Current Position To Area or Port

Another feature that may add information about global vessel positioning and effect geographical price arbitrage (Adland et al., 2008), is the vessels sailing distance to a specific port or area. Even more interesting is the total fleets sailing distance to this area. Therefore, the approach introduced in this study, is to add up the mean sailing distances to the specific area and take vessel size into consideration. The mean is of interest because if some vessel is not observed this week, the distance to loading area will drop significantly. This pop-in pop-out effect might fool the prediction model. Since LPG vessels are in the range of 1,000-80,000 cbm, weighting the mean distance based on vessel gas capacity might be of interest because a larger vessel will have a larger impact on the market than a smaller one. The weighted arithmetic mean distance $\bar{d}_{as}$ to area a in time interval s (week) is considered in Equation 4.1. By weighting the sailing distance, we do not need to separately calculate the sailing distance for the two vessel groups.

$$\bar{d}_{as} = \frac{\sum_i^{N_s} C_i \cdot D_{ias}}{\sum_i^{N_s} C_i} \tag{4.1}$$

where C_i is the capacity of vessel i and D_{ias} is the sailing distance from the average position of vessel i in week s to loading port/area a. Here, $i \in N_s$, where N_s is the set of all vessels observed in time s. To find the sailing distance, an ocean mesh grid was provided by Vit Prochazka. The mesh is created as a network of nodes and edges. The distance, or the edge weight, between two

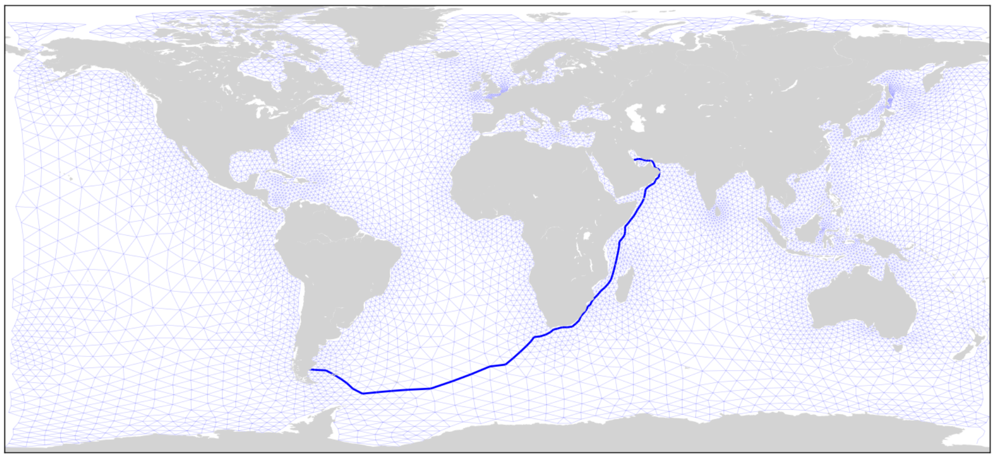


Figure 4.14: Mesh grid with an example shortest sailing distance, provided by Vit Prochazka

neighbouring nodes is calculated based on their coordinates. The mesh is visualized in Figure 4.14.

To find the shortest distance between two arbitrary coordinates, the following procedure is used:

1. Calculate the closest node for both the source and target coordinate.
2. Find the shortest distance between these nodes based on Dijkstra's shortest path algorithm (Dijkstra, 1959). This algorithm finds the shortest path between two nodes in a network based on the weights (distances) of the network edges.

It may be advantageous to know the loading status of the vessel, if it is laden or in ballast. This information can be extracted from the Message Type 5 AIS data since the draught of the vessel is included. First of all, the data from Message Type 1 needs to be connected to Message Type 5 because positioning data is not included in Message Type 5. However, the draught is manually set by the crew of the vessel and is prone to human error. This was investigated, but the quality of Message Type 5 date used in this study was too low, so it did not add additional information. Hence, to include laden/ballast information in the approach was neglected in this study.

Case Study Application

With the method introduced above, we can subsequently calculate the weighted arithmetic mean sailing distance in nautical miles from weekly vessel positions to Mont Belvieu in Texas, US. The feature constructed is plotted in Figure 4.15, and the descriptive statistics can be found in Appendix C.2..

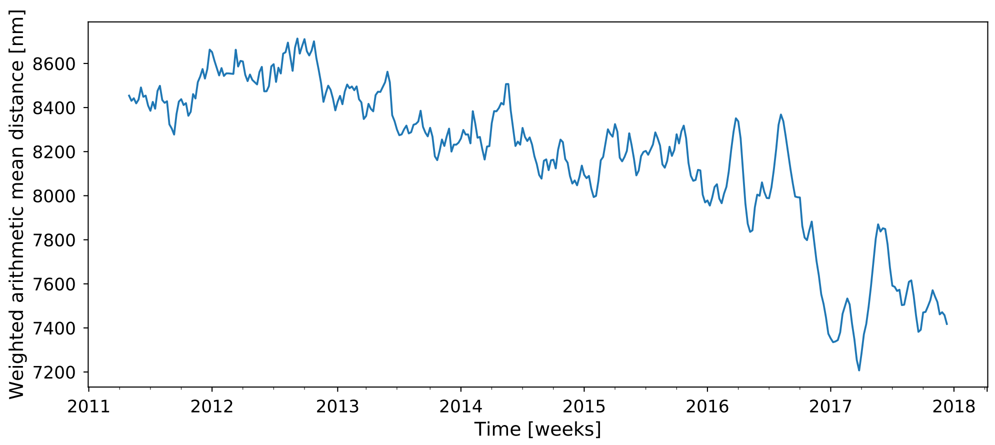


Figure 4.15: Weighted arithmetic mean sailing distance to Mont Belvieu for the LPG fleet, 2011-2017

4.3.3 Flux In and Out of an Area

As well as counting capacity and vessels in an area, knowing the flow in and out of the area might also provide useful market insight. The flow, or the flux, may be calculated by identifying when a vessel is inside and outside a geo-fence. An approach to calculate the flux is proposed as follows:

1. Analyze all messages to identify if they are inside or outside the geo-fence.
2. For each vessel, record when the change from outside to inside, and inside to outside, happens to get the point in time when the crossing happens.
3. Sum up for the whole fleet to get each directional flow or the net-flow, the flux, of the area.

Another approach is to difference the data so that we can investigate the relative change in the area from week to week. This is also a measure of flux.

4.3.4 Fleet Sailing Speed and Variance

Tsioumas (2016) and Ronen (1982) states that shipowners tend to speed up when rates are high to get the most out of the favorable market conditions. The argument favor that freight rates lead changes in speed. On the other hand, because of a small fleet (Adland et al., 2008), the high rates may result in all vessels being used to the max, and thereby the short-term supply can only be altered by higher speed again. Hence, one would expect an increase in average sailing speed for the overall LPG fleet to give rise to more supply, leading to higher competition among LPG vessels and lower freight rates. Therefore, it might provide additional information to include the overall vessel speed and speed variation in the prediction model. The former may be found by extracting the average of the whole fleets vessel speed and variance for every time interval. By doing this, we assume that the speed reporting is consciously distributed over the operational time of the vessels. AIS is sent by signals, and the signals we have in our database are not continuously distributed over time with equal time steps. Especially in the early signals due to interference issues and low satellite coverage (Eriksen et al., 2010). However, if this is the case for the whole study horizon, 2011-2017, we would assume that the difference between speeds at different time points reflect the real speed difference. Thus knowing that the speed average we obtain is not the real average, we can still get some valuable results since we look at changes over time and not the specific magnitude.

Case Study Application

Features for weekly speed average and standard deviation have been created for the whole LPG fleet. The features are plotted in Figure 4.16 and 4.17.

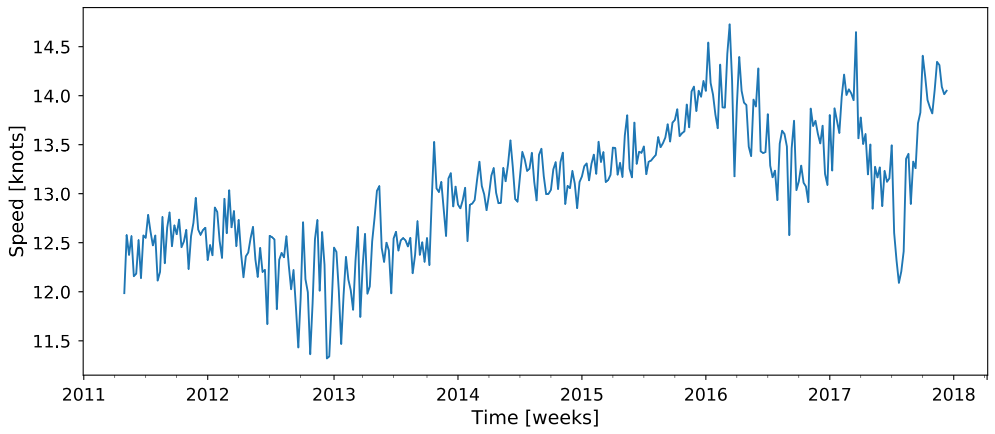


Figure 4.16: Weekly mean steed for the LPG fleet, 2011-2017

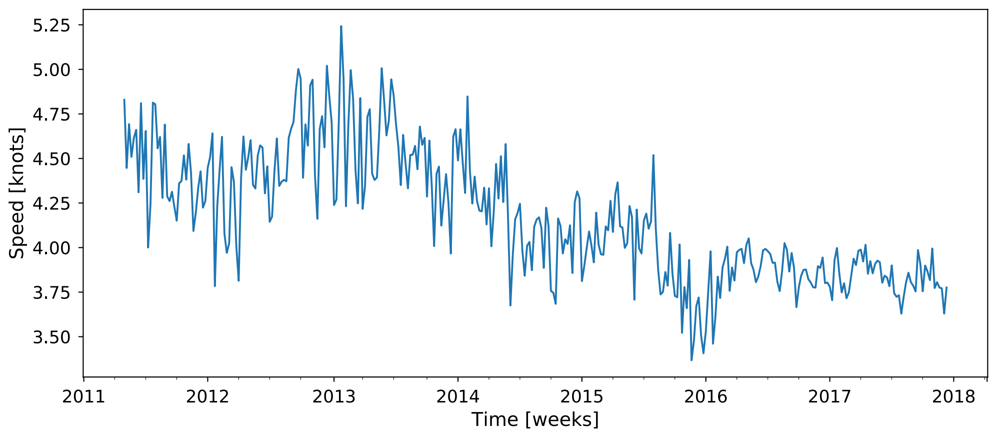


Figure 4.17: Weekly standard deviation in steed for the LPG fleet, 2011-2017

A list with descriptive statistics of all the features created in this section, can be found in Appendix C.3.

4.3.5 Price and Market Features

LPG Spot Price

The feature mostly used in price forecasting is the historical price of the prediction itself. Yu et al. (2008) and Kulkarni and Haidar (2009) only utilizes the historical price with neural network models to forecast future prices. In addition to historical prices of the Mont Belvieu propane spot price, the spot prices in the other major markets are also included in this study. This is done to investigate if some of these rates discover market information before others. Alternatively, the neural network model might be better at predicting the Mont Belvieu spot price by capturing the dynamic relationships between these series. The prices are acquired from Equinor, with daily price reports. As we in this study are looking at weekly observations, we have calculated the weekly averages based on the daily data, from 2011 to 2017. The features are plotted in Figure 4.18, and we observe that the spot prices follow each other evenly with small independent fluctuations. Hence the inclusion of these as separate features might add useful information to the prediction model. The features considered, with their respective notation, are the propane spot prices in Mont Belvieu (*spot_MB*), North West Europe (*spot_NWE*), the Arabian Gulf (*spot_CP*) and the Asian market (*spot_FEI*).

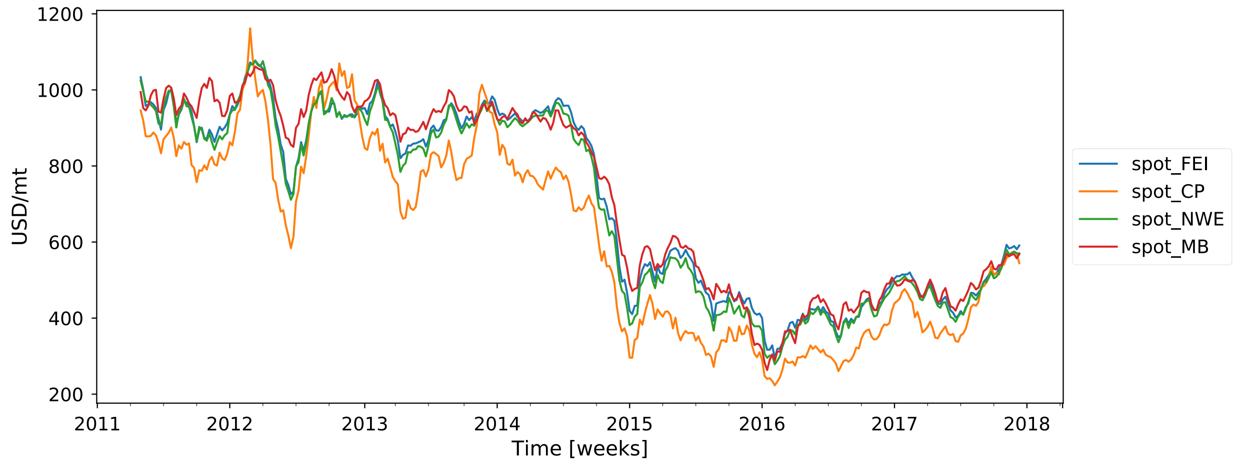


Figure 4.18: Propane spot price

Oil Price

LPG as a shipping commodity, have some distinct features. Unlike crude oil, coal or iron ore, LPG is not a raw material but derived from the extraction rate of LNG and crude oil (Engelen and Dullaert, 2010). The main forwarders of LPG are therefore dominated by the major oil and gas companies. This aspect makes the LPG market to be driven by "supply push" rather than "demand pull" in the sense that the volume of LPG shipments is derived from their products (LNG and crude oil) and is therefore not independently set (Adland et al., 2008). We therefore see it fit to include the crude oil price in our analysis, to see if some of the variations in the propane spot price might be explained by the oil price. Because we look at the Mont Belvieu spot price in Texas, US, we utilize the West Texas Intermediate (WTI) Cushing, Oklahoma, crude oil price acquired from the U.S. Department of Energy (2018).

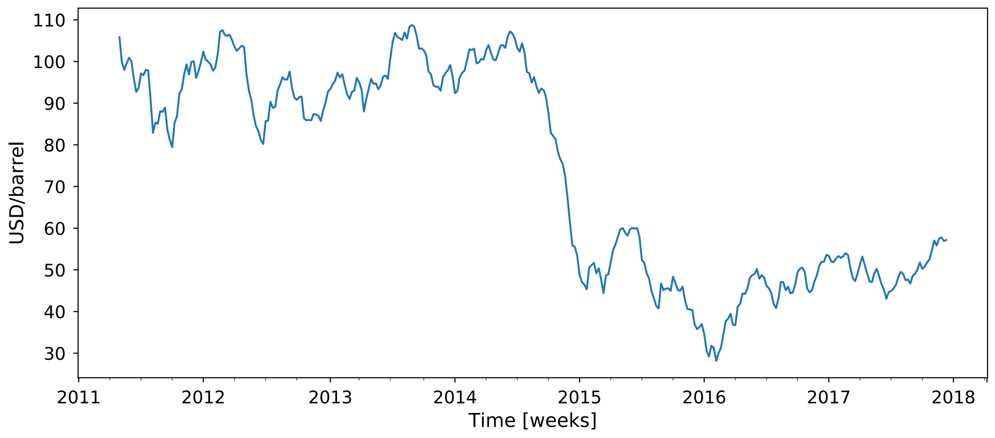


Figure 4.19: Weekly West Texas Intermediate (WTI), from 2011-2017

Comparing the plots in Figure 4.18 and 4.19, we observe that the general characteristics of the graph between movements in oil and propane prices are similar. Hence, the inclusion of the crude oil price might be an interesting feature to include.

A list of descriptive statistics of all the features from this section can be found in Appendix C.4.

4.4 Data Preparation

The data preparation procedure, or pre-processing, in this study mainly consists of three parts. The first part is to make the time series stationary. The second step is to normalize and scale the features. The third is to split the data into a training set and a test set, which is also called the in-sample and out-of-sample data as the first set are used to train the model and the second is to test its performance on unseen data. This is very important not to get an overfitted model that only performs well on seen observations. The data preparation script, *Data_preparation.py*, can be found in Appendix E.7. The following data preparation process is used in this study:

1. **Data transformation**: Analyze the time series and transform the data to stationary form.
2. **Normalize data**: Scale the data so that it is applicable to machine learning problems.
3. **Split the data into training and testing sets**: So that it is applicable to supervised learning problems.

In traditional time series analysis, working with stationary data is crucial. The methods require the data to be stationary to be effective. Stationarity means that the data is independent of time, without trends and seasonal effects. More precisely the traditional models require that for a weakly stationary process, the mean, variance, and autocovariance should be constant in its first and second momentum. Here, constant autocovariance means that the covariance of any sequential values is the same for stationary series (Kulkarni and Haidar, 2009). Working with non-stationary data may lead to a phenomenon called *spurious regression* which states that the variables are wrongly causally related (Brooks, 2002). According to Brooks (2002), stationary is important, for example in case of a shock, where for a stationary series it will have less influence over time. In non-stationary series, the influence of a shock remains for longer time steps and might lead to misleading result. Stationarity is not a requirement for many machine learning methods, but is preferred as it is a much stronger indication of the predictive power of the model. In this way, the model may be able to identify the underlying mechanics rather than just identifying a seasonal trend or dynamic.

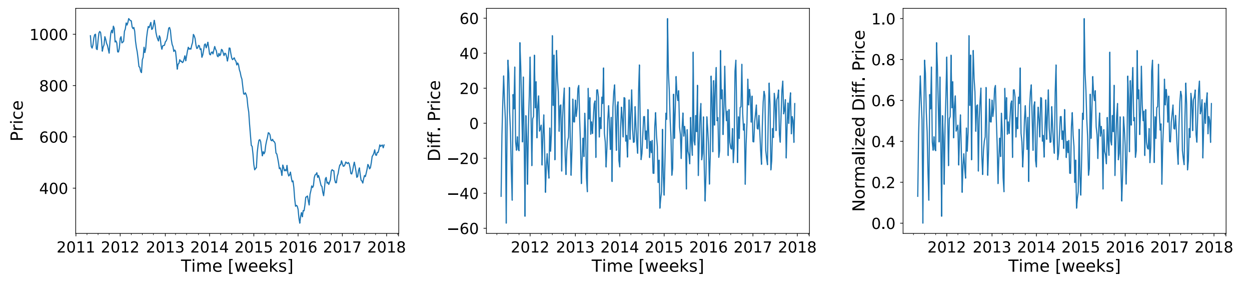
The method used to transform the time series to stationary form is by differencing the data. This is done by subtracting the previous observation from the current observation. Now the series tells how much the change is from one time point to another. To test for stationarity, we visualize the time series and their autocorrelation and partial autocorrelation, in addition to performing an Augmented Dickey-Fuller test (Fuller, 1996). This is a unit root test[1] with a null hypothesis (H0) that the time series can be represented by a unit root, meaning that it has no time dependency. If the null hypothesis is accepted, it suggests the series has a unit root and is non-stationary. The alternate hypothesis (H1) is that the time series does not have a unit root. The result of the test is interpreted by the p-value that states the level of significance. We also get a test statistic that can be compared to a threshold of 10%, 5% and 1%. The null hypothesis is accepted at each of these levels of significance if the test statistic is higher than the thresholds. The results of the tests are shown in Appendix D. Other stationarity methods like the logarithmic difference and other transformation methods were also considered, but the series are shown to be stationary with the first differencing of the data, both from the Augmented Dickey-Fuller test and visualization of the series.

When the features are stationary, the next step is normalization. This is done because of the activation function of the neural network and the variability in range for the different variables. The activation function or transfer function has a given limit for the input. Therefore the data is scaled to fit the function. The values are scaled to the domain [0,1] because of the activation functions of the neural network. Also, with different scales for the features, normalization is important in order to get a statistically sound model that can generalize the underlying problem. There are different kinds of scaling methods, but the most used, and the one used here, is the *Min-Max scaling* method, also known as normalization. The normalization of feature the p-th feature X_p is done in the following way:

$$z_{p,t} = \frac{x_{p,t} - \min(X_p)}{\max(X_p) - \min(X_p)} \tag{4.2}$$

where $x_{p,t} \in X_p$ so that the normalization of X_p is $X_p^{\text{norm}} = (z_{p,t}, \dots, z_{p,n})$ in the scale of $[0,1]$. From here on the normalized time series X_p^{norm} is denoted as just X_p. This transformation does not change the shape of the variable distribution, so the mechanics of the time series are preserved. A visualization of the pre-processing of the time series is shown in Figure 4.20, where the Mont Belvieu spot price is the example used.

[1]A test that determines how strongly a time series is defined by a trend.



(a) Original (b) Differenced (c) Normalized differenced

Figure 4.20: Mont Belvieu spot price data transformation and normalization

The last step in the data preparation process is to split the data into training and testing sets. The training set is used by the algorithm to fit the parameters of the function $\hat{f}(X)$ (see Chapter 5). The training set is again split into a sub training set and a validation set. This because the validation set is used to optimize the hyperparameters of the model with the genetic algorithm described in Section 5.4. The validation set is not included in the testing set because the error rate estimate of the final model on validation data will be biased since the validation set is used to select the final model (optimize hyperparameters). The test set is only used to assess the performance of a fully-trained model. This will show how the algorithm performs on unseen observations. After assessing the final model on the test set, no further tuning is done in order to resemble reality. The size of the training, validation and testing sets are chosen to 80 %, 10 % and 10 % respectively. Features selection will only be based on the training set, because if features selection is performed on a set including testing data, the model will be biased. Using a larger set to select features from reduces the variance of the performance estimate, but not the bias. To get an unbiased performance estimate, the test data is not used to make choices about the model, including feature selection. A visualization of the training, testing and validation set split is shown in Figure 4.21.

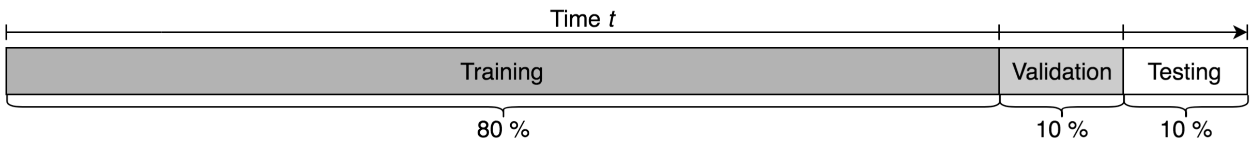


Figure 4.21: Training, testing and validation sets

4.5 Feature Selection

Feature selection is the process of selecting a subset of features to be used in the predictive algorithm. The reason for decreasing the set of created features, is that the set may contain many features that are either redundant or irrelevant, and can thus be removed without suffering much loss of information. James et al. (2013) and Bermingham et al. (2015) states that effective use of feature selection will enhance generalization by reducing overfitting, reducing training time and simplify the model, which makes it more understandable for other researchers. Less misleading data will also mean that the modeling accuracy improves. Also, feature selection avoids the curse of dimensionality, an expression by Bellman (1972), which involves various problems that arise when the model dimension becomes very high. The idea of the curse of dimensionality is that when the amount of features increases, the volume of the dimensional space increases so fast that the available data becomes sparse. This is problematic because, for the model to have statistical significance, the data needed to support this result often grows exponentially with the dimensionality.

The prediction of a variable y, is given by the estimated function $\hat{f}$, so that the prediction of y is $\hat{y} = \hat{f}(X)$. We can take the expected mean square error (MSE) of the prediction error $(y - \hat{f}(X))$, and carry out the calculation (James et al., 2013):

$$E(y - \hat{f}(X))^2 = \mathrm{Var}(\hat{f}(X)) + [\mathrm{Bias}(\hat{f}(X))]^2 + \mathrm{Var}(\epsilon) \tag{4.3}$$

Here $E(y - \hat{f}(X))^2$ denotes the expected MSE, $\mathrm{Var}(\hat{f}(X))$ the variance of the learning model selected, $[\mathrm{Bias}(\hat{f}(X))]^2$ the bias squared of the learning method caused by simplifying assumptions and $\mathrm{Var}(\epsilon)$ the irreducible error which is the lower bound of the expected error on unseen samples (James et al., 2013). This states that to minimize the expected test error, we need to select a subset X and a statistical learning method $\hat{f}(X)$ that simultaneously achieves low variance and low bias. In the context of features, a set with too few might result in high bias with a model that has a low performance on the training set. A selection of too many features might indicate high variance, an overfitted model, where the model will have lower performance on the test set than training set. This is known as the bias-variance trade-off. The relationship also shows that feature selection is a crucial part of the model selection process.

As stated above, we want to select features that improve the model and give relevant information to better make a prediction. A feature that has no impact on the problem, should not be a part of the problem. The added model complexity by including a feature must be justified by an increased model skill or capability. We can objectively estimate the importance of a feature with various statistical test and methods. Firstly, one can divide between univariate and multivariate feature selection methods. The univariate methods consider the variables one by one, while multivariate methods consider the whole groups of variables together as one feature by itself may be useless, but may provide useful information in combination with other. Secondly, we can classify the different feature selection algorithms into two parts (Kojadinovic and Wottka, 2000), namely filter methods and wrapper methods. Filter methods select a subset of features independent of the model that shall use them. Wrapper methods select a subset of variables taking into account the model that shall use them.

Researchers are also considering a third feature selection method, the embedded method (Pošík, 2015), where the feature selection is built into the machine learning algorithm. An example is Random Forests. We can take advantage of these models and use their structure to generate a separate ranking. In this study, Random Forests is used for this purpose.

4.5.1 Filter Methods

As stated in Section 4.5, filter methods selects features independently on the model that will subsequently use them. In this section, both multivariate and univariate selection methods will be introduced. The reason for including univariate selection methods in a multivariate forecasting problem, is that they are good at pinpointing important variables. If some features seem important, they may be further extracted into new features. Filter methods are not only used for subset selection, but also to see interesting statistical relationships between a feature and the predictor. The filter feature selection methods considered in this study are shown in Table 4.1.

Table 4.1: Univariate and multivariate filter methods

Univariate	**Multivariate**
Linear Correlation	Linear Regression
Maximal Information Coefficient	Lasso Regression
	Ridge Regression
	Random Forests
	Linear Subset Selection

These methods are used to generate ratings for each feature based on their relative importance in predicting the predictor. They are used on the features constructed in Section 4.3. The ratings are scaled to ranks in the domain of $[0,1]$, where 1 implies that the feature is statistically important in predicting the prediction variable, and 0 useless. A total overview of the scores are shown in Appendix F, and discussed in Chapter 7. The analysis is done with the Python script *Feature_importance.py* located in Appendix E.8. Most of the calculations are done using the Scikit-learn machine learning package in python, Pedregosa et al. (2011). The following methods are considered:

Linear Correlation

Just by looking at the linear correlation between a feature and the prediction variable is a univariate feature selection method. More specifically, the Pearson correlation coefficient (PCC) is used:

$$\rho_{X_p,Y} = \frac{cov(X_p, Y)}{\sigma_{X_p}\sigma_Y} \tag{4.4}$$

Here, $cov(X_p, Y)$ and σ_{X_p},σ_Y is the covariance and the standard deviation of the variables X_p and Y respectively. X_p denotes the p-th feature in the set of features X. The expression returns a value in the domain of $[-1, 1]$ where the boundaries are the perfect positive and perfect negative linear correlation. To rank a feature on this statistical measure, the absolute value $\mid \rho_{X_p,Y} \mid$, have been carried out so that it is a rank in the domain of $[0, 1]$ where a higher rank implies that the feature is more important since the variables are more linearly correlated. The main drawback of using linear correlation as a measure is that it is only sensitive to a linear relationship. If the relationship is non-linear, the linear correlation can be close to zero even if there is a non-linear relationship between the two variables. However, the method gives valuable insight.

Maximal Information Coefficient (MIC)

In statistical theory, mutual information (MI), is the mutual dependence between two variables. For two random variables, MI is defined as (Cover and Thomas (2006)):

$$I(X_p; Y) = \sum_{y \epsilon Y} \sum_{x_p \epsilon X_p} p(x_p, y) \log\left(\frac{p(x_p, y)}{p(x_p)p(y)}\right) \tag{4.5}$$

where $p(x_p, y)$ is the joint probability function of the p-th feature X_p and the predictor Y. $p(x_p)$ and $p(y)$ are the marginal probability distribution functions of X_p and Y respectively. Here binning[2] is used to discretize the continuous random variables. The selection of the number of bins and the fact that MI is not normalized makes the process sensitive. The maximal information

[2]Sorting continuous data into class intervals.

coefficient (MIC) was introduced to address these shortcomings by searching for optimal binning and turn MI scores into a metric that lies in the range of $[0,1]$. The advantage of MIC is that it is able to find non-linear relationships between variables (Reshef et al., 2011). Therefore, the MIC is used as a univariate filter method in the feature selection process, and function like a non-linear correlation coefficient.

Linear Regression

Fitting the model to a linear function is also a way of doing an univariate ranking. By simply looking at the coefficients of the linear model, the most important features should have the highest coefficients values given that the features are normalized. The multiple linear model, where the fitted relationship between Y and the subset of features X, is given by (James et al., 2013)

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + ... + \beta_p x_{p,t} + \epsilon_t \tag{4.6}$$

where $x_{1,t}, x_{2,t}, ..., x_{pt,}$ is the feature values at time t for features $X_1, X_2, ..., X_p$. Lagged features are also included in the analysis, but is then incorporated as its own feature added to the p features presented here. $\beta_1, \beta_2, ..., \beta_p$ are the coefficients which is the association between the features in X and the response Y, and β_0 is the linear intercept constant. ϵ is the noise of the model. The regression coefficients β are found by minimizing the Residual Sum of Squares (RSS), and the value of β which minimizes this sum is called the Ordinary Least Squares (OLS) estimator for β:

$$RSS = \sum_{t=1}^{n} \epsilon_t^2 = \sum_{t=1}^{n} (y_t - \hat{y}_t)^2 \tag{4.7}$$

where the estimated model is

$$\hat{y}_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + ... + \beta_p x_{p,t} \tag{4.8}$$

This is a form of multivariate method since the model considers all features and selects the coefficients β so that Equation 4.7 is minimized. This optimization procedure will lead to overfitting, but may be solved by introducing regularization, described later in this section. Scaling the coefficients to the domain $[0,1]$ makes it easier to compare with other filter methods. Again, this method is only applicable to linear relationships and does not capture the non-linearities in predicting shipping features. The model is good for datasets with few features, but will be unstable if a large number of features are introduced. This because small changes in the data can cause substantial changes in the model, making model interpretation very difficult. This problem is called multicollinearity. However, as stated before, this simple method gives a pinpoint to important features and shows interesting statistical properties of the features.

Lasso Regression (L1 Regularization)

Regularization is a very important technique in machine learning. These methods add a constraint to the linear regression model to prevent the coefficients to overfit. The regularization technique biases the estimated regression coefficients, but reduces the level of multicollinearity (O'brien, 2007). Since the method is based on regression, this is a multivariate filter method. There are two main forms of regularization, mainly L1 and L2, known as Lasso and Rigid regression. Regularization takes linear regression one step further by applying a regularization term to the minimization of the RSS in Equation 4.7. With L1 Regularization, the following function is minimized to obtain the regression coefficients β in Equation 4.6 (James et al., 2013):

$$RSS + \lambda \sum_{j=1}^{p} \mid \beta_j \mid \tag{4.9}$$

where $\lambda \geq 0$ is a tuning parameter that controls the relative impact of the regularization. $\lambda = 0$ would give an unregularized model, the same as multiple linear regression. To find the tuning parameter, cross-validation is used (Canu, 2014). The regularization term is also called the shrinkage penalty as when β is close to zero; it has the effect of shrinking the estimates of β_j towards zero. Writing out the optimization problem, the following is calculated (James et al., 2013):

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{t=1}^{n} \left(y_t - \beta_0 - \sum_{j=1}^{p} \beta_j x_{j,t} \right)^2 \right\} \qquad \text{subject to} \qquad \sum_{j=1}^{p} \mid \beta_j \mid \leq s \tag{4.10}$$

where for every value of λ there is a corresponding s such that Equation 4.9 and 4.10 will give the same lasso regression coefficient estimates. As for linear regression, lagged features is included but incorporated as its own variable. The β's can be scaled to $[0,1]$ and compared with other filter methods. This method is also linear, but prevents overfitting and might give some useful ranking output.

Ridge Regression (L2 Regularization)

As stated above, the second regularization method considered is the L2 Regularization, also known as Ridge Regression where the following function is minimized to obtain the regression coefficients β:

$$RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{4.11}$$

Writing out the optimization problem, the following is calculated (James et al., 2013):

$$\underset{\beta}{\text{minimize}} \left\{ \sum_{t=1}^{n} \left(y_t - \beta_0 - \sum_{j=1}^{p} \beta_j x_{j,t} \right)^2 \right\} \qquad \text{subject to} \qquad \sum_{j=1}^{p} \beta_j^2 \leq s \tag{4.12}$$

where for every value of λ there is a corresponding s such that Equation 4.11 and 4.12 will give the same lasso regression coefficient estimates. As for linear regression, lagged features is included but incorporated as its own variable. Like lasso regression, the β's can be scaled to $[0,1]$ and compared with other filter methods. This method is also linear, but prevents overfitting and might give some useful ranking output. The key difference between L1 and L2 regularization, is that L1 minimizes the less important feature's coefficients β to zero, thus removing some features altogether. This because Ridge regression will always generate a model involving all features since the regularization term will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero, in contrast to Lasso regression.

Random Forests

Random forests is a statistical learning method that constructs a set of decision trees. Every node in the decision trees is a condition on a single feature. It is designed in a way so that the dataset is split into two parts where similar response values end up in the same set. Breiman (2001) introduced a technique for using random forests for classifying feature importance. For regression problems like this project, the variance is the measure based on which the optimal condition is chosen. This measure is called impurity, and when training a tree, it computes how much each feature decreases the impurity in a tree. How much this impurity decreases for each feature can be averaged and used as a measure of feature importance. Random Forest can be referred to as a model based ranking method, since we take advantage of the model structure to generate a separate ranking using the built-in feature selection process. Since the model considers all the features together, this is a multivariate method.

The advantage with Random Forests is that it can model non-linear relationships well in addition to being a multivariate method. The MIC discussed above is also non-linear, but can only rank the features univariately. A problem with using Random Forests is that when some features are correlated with each other, then the model thinks that any of these can be used with no preference to exactly which. This causes a problem, because when the model selects a feature, the importance of other correlated features is significantly reduced as effectively the impurity they can remove, is already removed by the first feature.

Linear Subset Selection

This is not a variable importance method, but a method for finding the best subset. Instead of comparing all variables or single variables with the response, this method generates subsets with some of the features and compare them with different statistical measures. This method fits a linear regression model from Section 4.5.1 and estimates the parameters with OLS for each possible combination of p features. The model introduced here selects the best subset in the context of linear regression.

The *best subset selection* algorithm, or leaps and bounds procedure by Furnival and Wilson (1974), starts with fitting all subsets that contain exactly one feature to a linear regression model, then all possible models that contain two features, and so forth. So if we have p features, the possible combinations of models to consider is 2^p. For each model that contains k features, there are an amount of $\binom{p}{k}$ model combinations to consider. From these models, the best model is selected based on a selection criterion. As for the linear regression described in Section 4.5.1, RSS in Equation 4.7 is selected. Based on this selection criteria, the best model with regards to linear regression of k features is M_k. Other selection criteria are considered for choosing the best model from $M_0, M_1, ..., M_p$, since choosing based on RSS may lead to overfitting as stated in Section 4.5.1. In the context of training and testing sets, RSS will fit the model as good as possible based on the training set. The training error will decrease as more variables are included in the model, but the test error may not. Therefore these criterion is of interest. The selection criterion considered is the Akaike information criterion (AIC), Bayesian information criterion (BIC) and adjusted R^2. The adjusted R^2 is the feature modified R^2 given in Equation 4.13, so that the R^2 score is reduced when the number of features included $k < p$ are increased.

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2} \tag{4.13}$$

This is the coefficient of determination of the linear correlation between the forecasted value $\hat{y}$ and the actual observation y, where TSS denotes the total sum of squares and RSS the residual sum of squares. $\bar{y}$ is the mean of the observed data. The reason for using AIC, BIC and adjusted R^2 is that they adjust the training error for the models. Given by James et al. (2013):

$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2k\hat{\sigma}^2) \tag{4.14}$$

$$BIC = \frac{1}{n\hat{\sigma}^2}(RSS + \log(n)k\hat{\sigma}^2) \tag{4.15}$$

$$\text{Adjusted R}^2 = 1 - (1 - R^2)\frac{(n-1)}{(n-k-1)} \tag{4.16}$$

where $\hat{\sigma}^2$ is the estimate of the variance of the error ϵ in Equation 4.6, n the number of observations and k the number of features in the model. The optimal subset would have the highest adjusted R^2 and the lowest AIC and BIC.

The original model by Furnival and Wilson (1974) makes this computationally feasible to compare all subsets of up to 30-40 features. A model with 30 features will result in more than one billion possible models to be considered. Since we want to look at many features with many lagged variables, this method is computationally too expensive. Therefore, two greedy[3] algorithm variants of the *best subset selection* technique are used. The first is *forward stepwise regression* that starts with the intercept β_0, and then sequentially adds the feature that most improves the model. The second is *backward stepwise regression* that starts with a model containing all features, and then sequentially drops the feature that has the least impact on the model based on the selection criteria.

Implementing this analysis is done in Python as in James et al. (2013) with the code *Subset_ selection.py* in Appendix E.9. Since using the *best subset selection* algorithm is too computationally expensive, the *forward* and *backward stepwise selection* algorithms are used and compared. The results of the features selection methods are overviewed in the case study in Chapter 6.

It might also be interesting to see the relationship between a subset containing all the features not extracted from AIS (non-AIS features) data and a subset that includes them. This was done in the work of Olsen and da Fonseca (2017). However, they only considered these two combinations. In this study, it was found of importance to look at all features as the non-AIS features also may be redundant for the model. Based on the subset chosen in the forward and backward selection process described above, the optimal subset includes AIS features, which pinpoint to the choice of including them in the prediction model.

4.5.2 Wrapper Methods

Wrapper methods select the best subset of features, taking into account the model that shall use them. Testing all subset combinations for all features in a neural network is very computationally expensive. Therefore, the proposed method is to test the best subsets from the filter selection methods. Based on the features with high scores, different combination of these can be tested. This because the mostly linear filter selection methods might not be able to cap-

[3]Algorithm that selects the locally optimal choice at each stage with the hope of finding a global optimum.

ture some dual dynamics that the complex neural network is able to. Also, testing subsets with non-AIS features against subset including AIS features would give a pinpoint to the added information based on model performance, and thereby the possible advantage of including features extracted from AIS in a prediction model. This method is iterative in a way that different subset configurations will be tested in the case study in chapter 6.

5 | Machine Learning Methodology

This chapter will go through the forecasting technique and machine learning methods considered. Subsequently, the traditional time series models and evaluation metrics are introduced, to be used as means of evaluation for the machine learning methods.

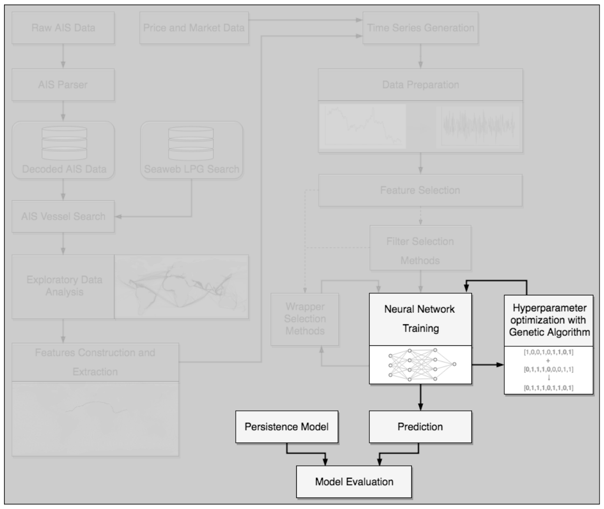


Figure 5.1: Sub-flowchart of project flow and methodology in *Machine Learning Methodology*

The machine learning models to be evaluated in this project is artificial neural networks, given their promise in time series forecasting in complex problems (Li and Parsons, 1997; Gao and Lei, 2017; Kulkarni and Haidar, 2009). The focus of this study is on the potential added information by including AIS in predicting spot prices with machine learning methods, and not to make discoveries in the field of machine learning. Therefore, only a brief introduction to the elements of neural networks is given.

The coding of the machine learning models used in this study is done in the python environment with Keras (Chollet, 2015) using a Tensorflow backend, a deep learning library to model machine learning methods (Abadi et al., 2016).

5.1 Supervised Learning

The goal of the machine learning prediction models is to map a function $\hat{f}$ to an input X so that the result $\hat{y} = \hat{f}(X)$ is the best estimation of the real value y. The way to find this function, is by training the model, or *learning* the model what the output should be given a specific input. In this way, during training, when the model returns a prediction, its told what the prediction should have been, and from there the network makes adjustments to alter this error. This is called supervised learning since the inputs and outputs are known, and the objective is to discover a relationship between the two (Shapiro, 2003). On the other hand, unsupervised learning is when only the input is known, and then the goal is to discover previously unknown patterns in the data. Since we in this study try to estimate the best mapping function between input and output, this is a supervised learning study.

5.2 Sliding Window Forecasting Technique

In order to make predictions with artificial neural networks (ANN), the time series dataset must be turned into a supervised learning problem. To achieve this, a sliding window technique is used. This technique is widely used in time series forecasting with ANN (Vafaeipour et al., 2014). The sliding window is a set of a fixed amount of lagged features. These lagged features are feeded into the model in order to make predictions. This method is the basis for how any time series dataset can be turned into a supervised learning problem.

The window slides one time step at a time, and for a window size of m, the input of a model with p features is $x_{p,t-m},...,x_{p,t}$ in order to predict y_{t+1}. More precisely, we want to fit a function to some historical data from time window $[t-m,t]$ so that $\hat{f}(x_{p,t-m},...,x_{p,t}) = \hat{y}_{t+1}$. The optimal window size is basically evaluated in the feature selection process in Section 4.5 based on the feature importance of lagged variables. The optimal window size can be found by using external optimization algorithms. These can be implemented to optimize the size of the window based on the performance of a model with this window size. In this way, we can get a better solution when features selection methods are used to include the variable itself in the subset of features, while the optimization algorithm is utilized to get the optimal window size based on this combination of variables. This approach is used in this study, with the help of a genetic algorithm described in Section 5.4 and implemented in Section 6.2.

5.3 Artificial Neural Networks (ANN)

Before we introduce how the neural networks in this study are structured for multivariate forecasting, each of the following sections will briefly start by describing the underlying mechanics of neural networks, and how they are able to learn complex problems. Subsequently, we will present how the networks are structured based on the sliding window approach to make multivariate forecasts.

5.3.1 Multilayer Perceptron (MLP)

The artificial neural network can be described as a mapping model, viewed as non-parametric, non-linear and assumption-free, meaning that it does not make a priori assumption about the problem (Kulkarni and Haidar, 2009). This makes it good for fitting complex problems. The network consists of layers with nodes, where the computations happens. An example of the basic form of a neural network, called a Multilayer Perceptron (MLP), is shown in Figure 5.2.

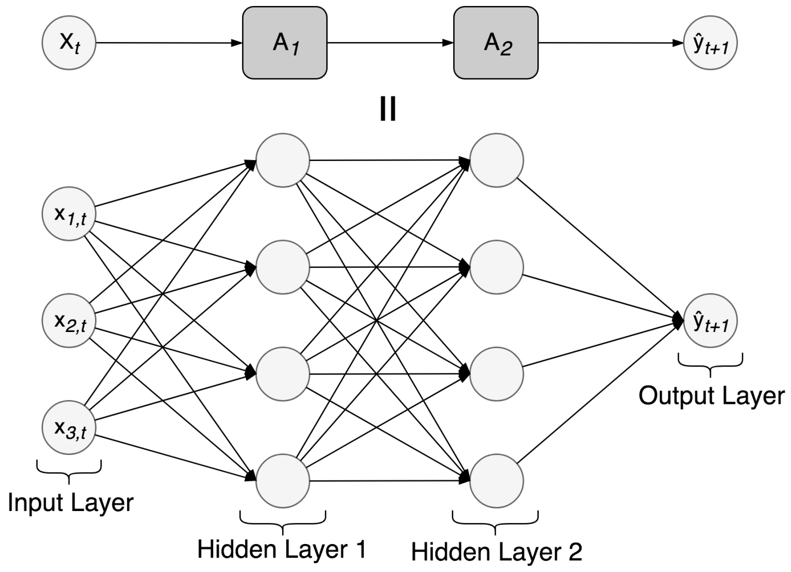


Figure 5.2: Simplified and outstretched MLP with two hidden layers A_1 and A_2

The inputs of the example network in Figure 5.2 is the arbitrary features $x_{1,t}$, $x_{2,t}$ and $x_{3,t}$ at a given time t, with a single output $\hat{y}_{t+1}$. The output of each layer is simultaneously the input of the subsequent layer, starting from an initial input layer receiving the data. The general architecture of the network is given by the number of inputs, number of hidden layers, number of neurons in the hidden layers and number of outputs. As stated above, based on the input X_t, the network is trained to respond with a desired output y_{t+1}. The neuron in the MLP is also called a perceptron. Figure 5.3 shows the components of a basic perceptron.

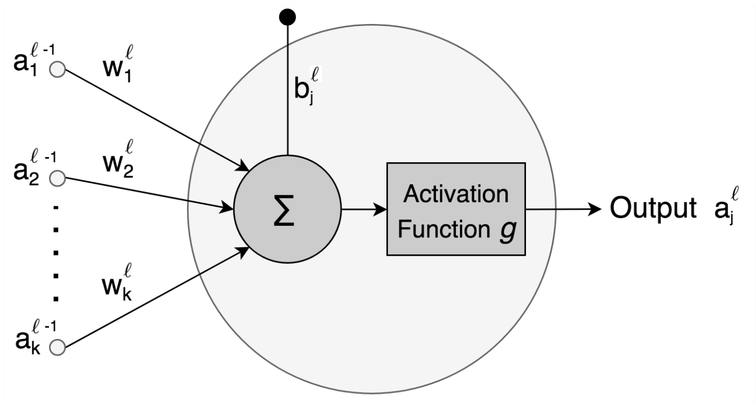


Figure 5.3: Composition of a single neuron j in the ℓ^{th} layer in a MLP

Here, the neuron combines input $a_1^{\ell-1},...,a_k^{\ell-1}$ from k neurons from the $\ell-1^{th}$ layer with a set of weights $w_1^{\ell},..,w_n^{\ell}$, that either amplify or dampen that input. Thereby a significance b_j^{ℓ}, called the bias in the j^{th} neuron, is added to the summation of the input-weight products and all is feeded into an activation function (see Equation 5.1). The input, called the activation, can be from a subsequent layer or the input layer $\ell = 0$, where $x_{1,t},...,x_{k,t}$ is the activations. The activation function determines whether and to what extent that signal progresses further through the network to affect the ultimate outcome. The choice of function varies from problem to problem, and layer to layer. In a basic MLP, all the neurons in the same layer have the same activation function. The most used activation function in machine learning is the *Sigmoid function*[1]. Other, functions like the *Rectified Linear Unit (ReLU)* and *tanh* are also widely used. For regression problems, the linear activation function also known as the *identity function*, meaning signal in equals signal out, is preferred in the output layer (da S. Gomes et al., 2011).

$$a_j^{\ell} = g_{\ell}\left(\sum_{j=1}^{k} w_j^{\ell} a_j^{\ell-1} + b_j^{\ell}\right) \tag{5.1}$$

Here, g_{ℓ} is the activation function in layer ℓ, $a_j^{\ell-1}$ the activations from the $\ell-1^{th}$ layer and a_j^{ℓ} the resulting activation from the j^{th} neuron in the ℓ^{th} layer. From this point, this activation is multiplied with $w_j^{\ell+1}$ to be put into neurons in the next layer and so on. A simplified representation of this is given in Figure 5.4. Here U and V denotes the weight vectors of w in front of and after the hidden layer A respectively. This figure is a neural network with one layer A and a vector input X_t with a single output $\hat{y}_{t+1}$.

[1]WHAT?

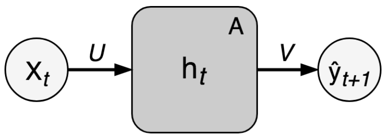


Figure 5.4: Simplified representation of MLP with one hidden layer and weight vectors U and V

Based on Figure 5.4, Equation 5.1 can be simplified to:

$$a^{\ell} = h_t = g(UX_t) \tag{5.2}$$

where h_t is the activations for the hidden layer A and further the output becomes:

$$\hat{y}_{t+1} = g(Vh_t) \tag{5.3}$$

The bias term is not included in these equations. However, it is accounted for as a part of the activation. The goal is to learn the b's and w's, the weights in U and V, so that the error of the mapping model is minimized. This error for a single training example is given by the loss function $\mathscr{L}(\hat{y}_t, y_t)$. The loss summed up for all training examples is called the cost function $J(w, b)$. For regression problems, the loss is usually the mean squared error (MSE) or mean absolute error (MAE) (see Section 5.5.3). The cost function measures how well the parameters w and b does on the training set. The next step is training the network. Backpropagation is a method to calculate the gradient of the cost function, basically saying how much the weights should be altered based on the desired output. This technique is also sometimes called backward propagation of errors, because the error is calculated at the output and distributed back through the network layers. To optimize this process, an optimization algorithm is used. The optimization algorithm repeats a two-phase cycle, propagation and weight update. Any gradient-based optimization algorithm can be used. To optimize the computation time, the choice of optimization algorithm is important. Historically the *Stochastic gradient descent (SGD)* were mostly used, but in later years the *Adam optimization algorithm*, an extension to SGD, have been receiving more popularity as it updates network weights iterative based in training data (Kingma and Ba, 2014).

According to Refenes (1995), there are three primary requirements for any successful neural network model. These are convergence, generalization and stability. Convergence refers to the convergence of the accuracy on a training data set. Generalization is the ability of the model to perform with new data, and is monitored by the models' performance on a testing data set. Stability refers to the consistency of the network output. These training and testing sets are split from the original data in order to resemble reality. Here the training set is used by the algorithm to fit the parameters of the model and the testing set gives an indication of how the algorithm performs on unseen observations. Overfitting occurs when a forecasting model has too few degrees of freedom. In the case of a neural network model, it means that the model has relatively few observations in relation to its parameters and therefore it is able to memorize individual points rather than learn the general patterns. The likelihood of overfitting is determined by the number of weights, known from the number of hidden layers and neurons. By including many features, thereby many input variables, this will result in more weights from the input layer to the first hidden layer. The likelihood of overfitting is also determined by the size of the training set. This is further described in the data preparation process of the features in Section 4.4.

The choice of hyperparameters in an ANN model is also very important. These are parameters that control the model parameters w and b, thereby the name *hyper*. Some of the most important hyperparameters includes the training rate α, number of iterations, number of hidden layers, number of hidden units and choice of activation functions. Kaastra and Boyd (1996) thoroughly review past literature on best hyperparameters, and studies vary from stating that the number of hidden neurons should be 75 % of the number of input neurons, to indications that an optimal number of hidden neurons will generally be found between one-half to three times the number of input neurons. The learning rate determines how fast or how much the model coefficients change. The number of iterations is the number of passes through the network, where each pass contains a number of examples called the batch size. Batch size defines the number of samples that are going to be propagated through the network. The smaller the batch the less accurate estimate of the gradient in the backpropagation. However, the higher the batch size, the more memory space is needed. In context to this, the number of epochs is also important, where one epoch is one forward pass and one backward pass with all the training examples. For example, for 200 training examples with a batch size of 100, it takes two iterations to complete one epoch. The hyperparameters are often chosen based on intuition and domain knowledge, but they can be optimized using external optimization methods. These methods take a combination of hyperparameters and return the associated loss. Methods like *Grid search* and *Random search* are widely used. In later years, evolutionary optimization methods like the *Genetic Algorithm* have been considered for hyperparameter optimization (Miikkulainen et al., 2017). This approach is taken in this study and is further described in Section 5.4, and the case study in Section 6.2.

With respect to multivariate time series forecasting, and the sliding window approach introduced in Section 5.2, the input neurons is the observations of p features in the time window $[t-m, t]$, with window size m. This will result in $(m+1)\cdot p$ input neurons. For example, if we have 8 feature time series with a sliding window size of 10, meaning that we are interested in the lagged features in the time interval of $[t-10, t]$ in order to predict y_{t+1}, this will result in an input of 88 neurons in the input layer of a MLP. An example model with the following notation and two hidden layers are shown in Figure 5.5.

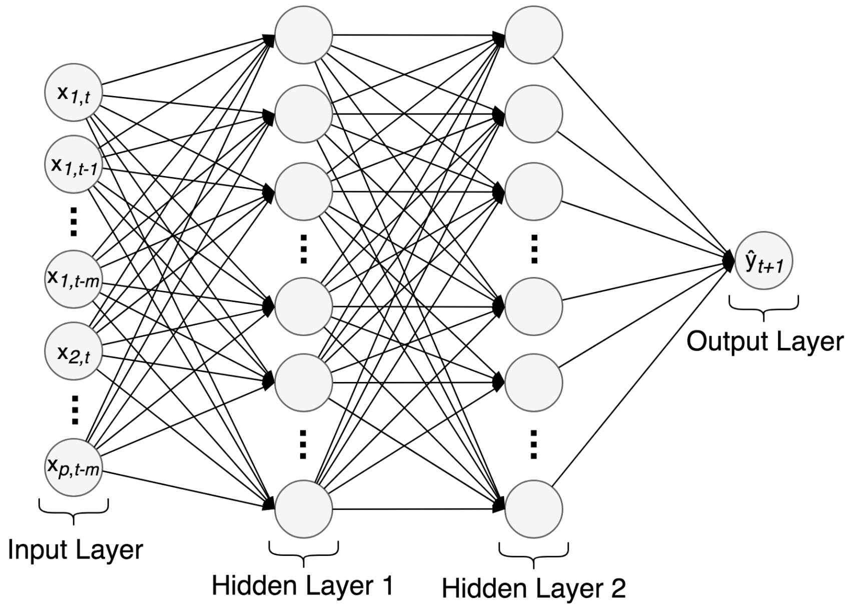


Figure 5.5: Example MLP model with two hidden layers, p features and window size m

5.3.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) also know as Elman networks, was introduced by Elman (1990). These networks were extensions of the original ANN with the idea of including a temporal behavior in the network. This temporal behavior is modeled as an internal state, or memory, at each time step to process sequences of inputs like time series. The general idea of a RNN is shown in Figure 5.6. The memory in the network is a loop, allowing information to persist, and be passed from one time-step to the next. Here, the prediction of y_t, or more specifically the activations from the hidden layer h_{t-1}, is again used to predict y_{t+1}. The hidden layer A is called a cell in the context of RNN. The input x_t can be passed in as a single observation or a one-dimensional vector of several features. In this multivariate study a vector $X_t = (x_{1,t}, x_{2,t}, ..., x_{p,t})$ is passed in for each time step t and p features.

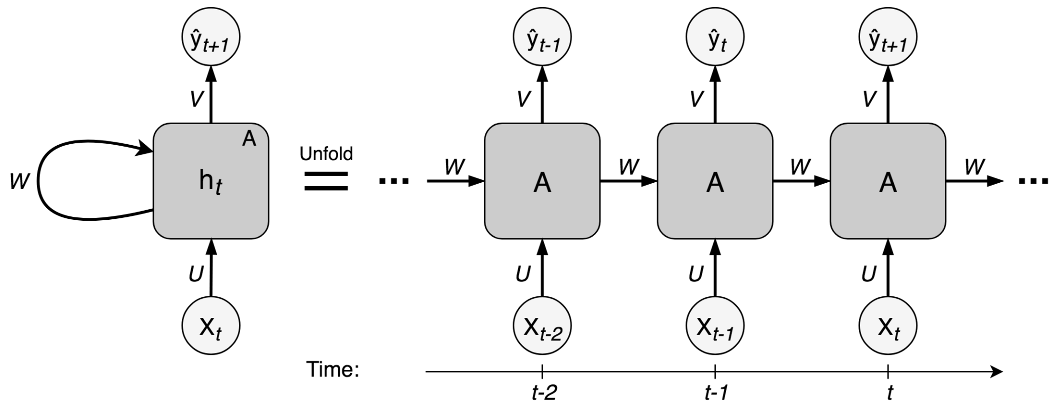


Figure 5.6: General idea of a one layer RNN

In the context of time series forecasting the output index of $\hat{y}$ in Figure 5.6 is shifted one timestep backwards in contrast to traditional literature notation where X_t is used to predict y_t. However, this is only a notational problem and is done so that we use memory from previous history combined with data from time t to predict at time $t+1$. In classical neural network like described in Section 5.3.1, the output $\hat{y}_{t+1}$ are calculated from only the input X_t, by running computations through the weights U and V (vectors of all the weights w) and the hidden part A. In RNN, the activations h_{t-1} from the hidden layer A from the previous time step $t-1$ are again used in computing $\hat{y}_{t+1}$. The activations h_t is calculated as in Equation 5.2, but the new output given by Equation 5.4, where a term taking account for the previous time step, is added:

$$h_t = g(UX_t + Wh_{t-1}) \tag{5.4}$$

This is visualized in Figure 5.7 where the box with the activation function g is where the hidden neurons are located and the computation of h_t happens. The new component in this network is the recurrent weights W. So when training RNN, the new component to fit is these recurrent weights. Therefore, the backpropagation of errors in the case of RNN also calculates how much the weights in W should be altered in addition to V and U, based on the desired output y_{t+1}.

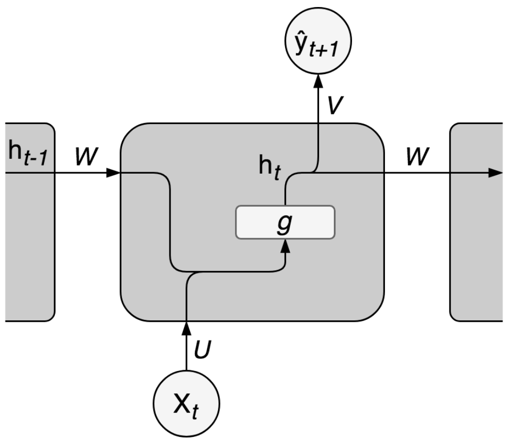


Figure 5.7: Computational flow in a RNN unit, inspired by Olah (2015)

The joining of the arrows in Figure 5.7 is concatenation while splitting is just a copy. Like traditional ANN, it is also possible to stack several RNN layers on top of each other so that the output from the RNN unit, h_t, is passed into a new RNN unit in the same timestep instead of being used to calculate $\hat{y}_{t+1}$.

A problem with training a neural network with gradient-based methods in back-propagation is the vanishing gradient problem (Hochreiter et al., 2001). Under back-propagation, described in Section 5.3.1, when calculating gradients of loss with respect to weights, the gradients tends to get smaller and smaller as the process moves backward in the network. Thus, the neurons in the first hidden layers learn very slowly as compared to the neurons in the later layers. The vanishing gradient problem affects traditional ANN with many hidden layers as well as RNN, since the recurrent action adds more weights. A type of RNN called the Long short-term memory (LSTM) avoids the vanishing gradient problem by introducing *gates*.

5.3.3 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks were introduced by Hochreiter and Schmidhuber (1997) to overcome the shortcomings of RNN with respect to the vanishing gradient problem and long-term dependencies. In theory, RNNs are capable of handling long-term dependencies by adjusting the network weights correctly. However, in practice, these models have problems in learning long-term dependencies where information needs to span over longer time steps. This problem was solved by Hochreiter and Schmidhuber (1997), with the introduction of LSTM with a cell state C_t that runs straight down the entire chain with only some minor linear interactions. The cell state represents the memory of the network. This internal state is a key feature that can aid in forecasting problems. To handle a new model parameter, instead of having a single neural network layer with one activation function (Figure 5.7), LSTMs have four (Figure 5.8) in addition to some pointwise operations. These have different roles. There exist many different LSTM configurations, but according to Greff et al. (2017) the resulting output is not very different.

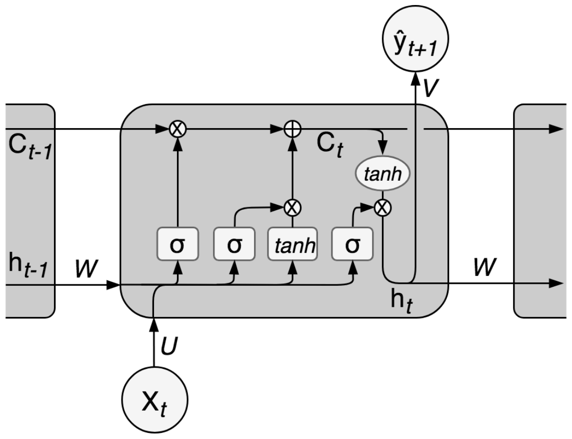


Figure 5.8: Computational flow in a LSTM unit, inspired by Olah (2015)

In Figure 5.8, σ denotes a layer with a Sigmoid activation function and *tanh* denotes a layer with a tanh activation function. The circles with "**x**" denotes a pointwise multiplication operation and circles with "**+**" denotes a pointwise addition operation. This combination of layers and a pointwise operation is called a cell gate. The roles of these gates in the LSTM unit is left to right: The *forget* gate with a Sigmoid layer and a pointwise multiplication. The *update* gate with a sigmoid and tanh layer, where the outputs of these are multiplied and added to the cell state from the first gate, so the state of the cell is updated. The last gate is the *output* gate where the cell state passes through a tanh activation function and is multiplied with an output of a sigmoid layer. This output is again passed on to the next LSTM unit along with the cell state.

To get the best neural network model, the hyperparameters need to be chosen wisely. They can be selected with external optimization methods for RNN as for ANN. When it comes to computational time, LSTMs, as RNNs in general, are time-consuming to train because of more weights in the model. Therefore dimensionality reduction through feature selection is important. This is described in Section 4.5.

With respect to multivariate time series forecasting, we are not interested in the output of the intermediate LSTM units, but only outputting the prediction y_{t+1} based on the input from the time window $[t-m, t]$. This is classified as a many-to-one RNN where a sequential input is used to predict a fixed output, in contrast to a many-to-many visualization in Figure 5.6. The amount of LSTM units will be the window size $m+1$ as m lagged variables are included in addition to the current observation at time t. In contrast to the resulting MLP in Figure 5.5, this representation does not take all the time steps in $[t-m, t]$ for all features p into one network, but takes all features at a given time to then loop the prediction over the window. So the input of the LSTM is a vector $X_t = (x_{1,t}, x_{2,t}, ..., x_{p,t})$ for m timesteps, resulting in a number of $(m+1)$ LSTM units. For example, if we have 8 feature time series with a sliding window size of 10, meaning that we are interested in the lagged features in the time interval of $[t-10, t]$ in order to predict y_{t+1}, this will result in an input 11 LSTM units in the LSTM-model, in contrast to 88 in the MLP-model. Altering the model visualization in Figure 5.6, the context of many-to-one and the sliding window is visualized in Figure 5.9, where a time window of size m with p features are used to predict y_{t+1}.

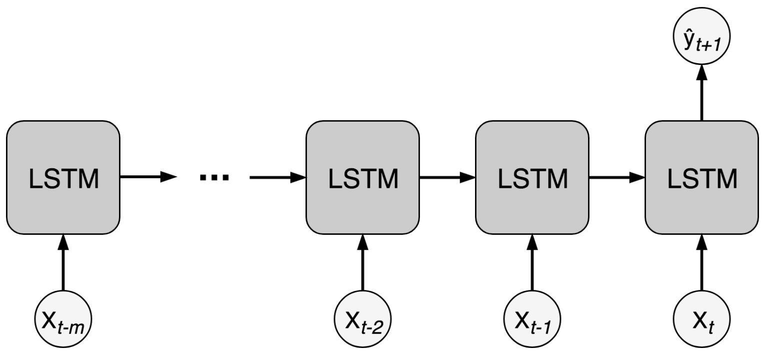


Figure 5.9: Many-to-one LSTM model with one later and window size m

5.4 Genetic Algorithm for Hyperparameter Optimization

As stated in Section 5.3.1, external optimization methods can be used to optimize the hyperparameters of a neural network model. This section will introduce how a genetic algorithm (GA) can be used for this purpose. A genetic algorithm is a metaheuristic[2] inspired by the concepts from evolutionary theory and natural selection[3]. The driving force in GA is the combination and exchange of chromosome material during breeding of individuals. The main parts of the optimization process in a GA is reproduction, crossover and mutation (Bodenhofer, 2003). Firstly, an initial population is created, and each individuals fitness calculated. Regarding optimization, this fitness is the value of the objective function. Reproduction is the creation of a mating pool for randomly selected individuals. The selection is random, but with a bias towards those best fitted. Individuals are mated randomly in the crossover part, where the mating process

[2]Higher-level heuristic designed to generate and find a good feasible solutions that are not (necessarily) optimal.

[3]Term popularized by Charles Darwin describing the survival and reproduction of individuals based on his four postulates (*1859, On the Origin of Species*)

produces individuals with new characteristics by copying parts of the fit individuals to the next generation (Shapiro, 2003). The last part is the mutation of some individuals. This only plays a secondary role in GA, but is useful to produce valuable building blocks which did not exist in the population from the start. Miikkulainen et al. (2017) states that this method may be good for hyperparameter optimization, and Shapiro (2003) describes that a GA is an intelligent approach to trial and error. The procedure of a GA is shown in Figure 5.10.

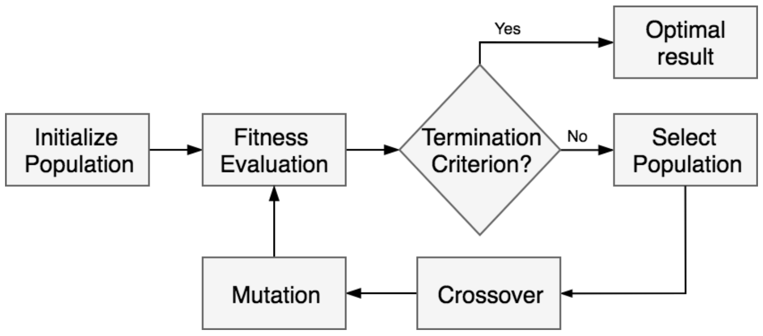


Figure 5.10: Structure and process of genetic algorithms

In the context of hyperparameter optimization, the initial population is created by randomly generating binary tuples of hyperparameters based on some parameter limits. For example like the one in Table 5.1, where each position is assigned to a specific hyperparameter. Thus, the created structure needs the upper limit of the hyperparameter to be defined in advance. Based on hyperparameter ranges, these limits are chosen based on previous work and industry practice. The table is only an example, and the configuration used in this study are overviewed in the case study in Section 6.2.

Table 5.1: Example of a configuration of an individual in a Genetic Algorithm

Individual structure	$[ws, ws, ws, hn, hn, hn, hn]$
Random binary tuple configuration	[1 , 0 , 1 , 1 , 0 , 0 , 1]
Corresponding window size (ws)	Integer([1 , 0 , 1]) = 5
Corresponding hidden neurons (hn)	Integer([1 , 0 , 0 , 1]) = 9

After the initial configuration of individuals, the fitness of each individual is calculated based on cross-validation accuracy of the machine learning algorithm with these specific hyperparameters. These tuples are bit strings where a part of the tuple is connected to, i.e., the number of hidden layers, and another part to the learning rate etc. The fitness is calculated based on the validation set. So the model is trained on a part of the training set, and validated on the validation set (see Section 4.4). After fitness calculation, the individuals are ranked based on their relative fitness, the network performance on the validation set. Then the worst-performing hyperparameter tuples are replaced with new hyperparameter tuples generated through crossover and mutation. This process is repeated until satisfactory algorithm performance is reached or algorithm performance is no longer improving. The termination criterion used in this study is the number of generations. One generation is one cycle through the algorithm. The best individual after all the generations is the optimal result.

Another important notion to mention is the effect of randomness in the neural network. The network will produce different results for separate training sessions with the same hyperparameters. This is a result of randomness when initializing the network. More specifically, the different initial configuration of weights in the network which are randomly set at training start. This will cause two identical networks to produce slightly different results for separate training runs. A solution to this is to train and test the same network multiple times and then average the results. However, to do this for every individual in the genetic algorithm is very computationally expensive. A solution to this is to run the genetic algorithm as usual, and then subsequently test the 5-10 best performing individuals multiple times and the average. The total process can be overviewed as follows:

1. Create a binary tuple structure based on hyperparameter ranges and generate an initial population of solutions.
2. Run the genetic algorithm for all the generations, with fitness calculation, selection, crossover and mutation.
3. Test the best performing individuals multiple times and average out the results to get the overall best individual.

The advantage of using a genetic algorithm is that it is easy to adapt and it is not required to model the function, and the mutation prevents the function to get trapped in local optima. On the other note, it is computationally intensive for complex fitness evaluation. The implementation and the algorithm architecture used in this project is found in the case study in Section 6.2 and as a part of the scripts *MLP.py* and *LSTM.py* in Appendix E.10 and E.11.

5.5 Model Evaluation Method

To evaluate the performance of the machine learning models, various metrics and baseline models are used to quantify the relative error and accuracy of the models. The choice of performance metric should be connected to the ultimate goal of the study. If the goal is to provide a risk management tool, the direction of the price may be more interesting than the magnitude of the prediction. If the aim is profitability, the magnitude would be more interesting to investigate. It is also challenging, if not impossible, to successfully predict the magnitude (Kulkarni and Haidar, 2009). However, the scope of this study is general, so both the magnitude and direction of the prediction is of interest. In addition to statistical error and accuracy metrics, the persistence and Vector Autoregressive (VAR) model is used to baseline the results of the prediction model. These models are widely used in multivariate forecasting (Olsen and da Fonseca, 2017) and to baseline shipping commodities (Alquist et al., 2013), respectively.

5.5.1 Persistence Model

To get a relative view of the models' performance, we can develop a model to baseline and compare the results to. A model that can be used as a baseline, is the persistence model, also called the no-change model. This is a heuristic that uses the previous time step to predict the expected outcome at the next time step. It can be represented in the following way:

$$\hat{y}_t = y_{t-1} \tag{5.5}$$

Here $\hat{y}_t$ is the prediction of y_t at time t, and y_{t-1} is the observation at the previous time step $t-1$. This model is good for baseline forecast because it is simple and fast to implement, in that the method requires no training. The persistence model is frequently used in forecasting of oil prices (Alquist et al., 2013). Despite the models' simplicity, it appeared to be a good baseline and performed better than other heuristic approaches. Therefore, the persistence model is used for its simplicity and fast implementation.

5.5.2 Vector Autoregressive Model (VAR)

Traditional time series models such as the univariate autoregressive integrated moving average (ARIMA) can also be used for baseline forecast and are often used in economic studies. In multivariate forecasting, the vector autoregressive (VAR) model is a common option, as it often provides superior forecasting performance to those from univariate models (Olsen and da Fonseca, 2017). As a traditional time series model, VAR requires the data to be stationary. Stationarity was found by visualization and the Augmented Dickey-Fuller test (Fuller, 1996) from Section 4.4, with results presented in Appendix D. The VAR model can be defined as follows:

$$Y_t = A_1 Y_{t-1} + A_2 Y_{t-2} + ... + A_m Y_{t-m} + u_t \tag{5.6}$$

Here, a vector of all the p features and the prediction is predicted based on the previous m observations, which is the window size. This vector Y_t, is the vector of all features at time t, which includes both the features and the prediction. Hence, $Y_t = [x_{1,t}, x_{2,t}, ..., x_{p,t}, y_t]$. For p features, A_i is the $((p+1)*(p+1))$ coefficient matrix, since we have p features and one variable to predict, the price. u_t is the Gaussian zero mean white noise, also known as the error term.

Using the same features and sliding window technique as for the machine learning models, the optimal window size can be found by using AIC and BIC described in the features selection process in Section 4.5. Thus, finding the optimal window size with AIC using the same features as for the other models, multivariate forecasts can be carried through, and we can baseline our neural network performance against these.

5.5.3 Metrics Used In Statistical Modeling

Various error metrics for measuring the accuracy of the model on the testing set is used. Three different metrics are used to evaluate the magnitude accuracy of the prediction model, and one metric is used to evaluate the directional accuracy. Firstly, the Root Mean Square Error (RMSE) is considered for magnitude accuracy evaluation. This is the most used measure of accuracy for artificial neural networks. RMSE is given by the following equation:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{5.7}$$

where $\hat{y}_t$ is the prediction of the real value y_t at time t on a set of n testing examples. Secondly, the Mean Absolute Error (MAE) are also considered as a performance metric. The MAE of the prediction model is given by:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} \mid y_t - \hat{y}_t \mid \tag{5.8}$$

The MAE is better in describing average error while the RMSE has the benefit of penalizing larger errors. Therefore both are considered in the evaluation. Further another metric, the Mean Absolute Percentage Error (MAPE) can be calculated from MAE, by calculating the relative MAE in relation to the true observation:

$$\text{MAPE} = \left(\frac{1}{n}\sum_{t=1}^{n}\frac{\mid y_t - \hat{y}_t \mid}{\mid y_t \mid}\right) \cdot 100\% \tag{5.9}$$

Lastly, to measure the directional accuracy of the prediction, the Directional Accuracy Ratio (DAR) is used. The metric is used by both Gao and Lei (2017) and Kulkarni and Haidar (2009) to evaluate oil price movements. DAR, also known as the success ratio of direction prediction, can be calculated the following way:

$$\text{DAR} = \frac{1}{n}\sum_{t=1}^{n} d_t \tag{5.10}$$

where $d_t = 1$ if $(\hat{y}_t - y_{t-1})(y_t - y_{t-1}) > 0$ and $d_t = 0$ otherwise. The data is pre-processed with differencing and normalization described in Section 4.4. This means that the results only needs to be inverse-normalized and then the direction is seen from the sign of the differenced data. To compare DAR to the persistence model in Section 5.5.1 is not possible since the direction of the price for one step to another is nothing with the persistence model. Therefore, this metric will be compared to a random guess known as a coin flip, meaning that there is a 50 % chance of the price going up or down, resulting in a DAR = 0.5.

6 | Case Study

This chapter will present the details and procedure for the case study, which investigates whether multivariate machine learning forecasting methods using features extracted from AIS data adds additional information in predicting short-term fluctuations in the Mont Belvieu propane spot price. To potentially identify the added performance of including AIS features in forecasting the spot price, the models developed will be executed for both a subset including AIS features and a subset without any features extracted from AIS data, but only previous price and market history. These sets are denoted as AIS and non-AIS features, respectively. In this way, the value of the added information might be seen through the performance metrics.

6.1 Feature Selection

This section presents the results of the feature selection methods, with the features created in Chapter 4. Keep in mind that all the scores are based on the stationary transformed data. In this way, we try to capture the underlying mechanics of the time series.

6.1.1 Results of Filter Methods

Calculating the feature importance scores from the filter methods; lasso regression, linear correlation, linear regression, Maximal Information Coefficient (MIC), random forests and ridge regression, introduced in Section 4.5, yields the feature importance scores in Appendix F. Here each feature has been tested for lags down to 20. This means that to predict *price(t+1)*, the Mont Belvieu spot price, all features in the range of $[t, t-20]$ are tested. As stated in Section 4.5, the scores are ranked and normalized in the scale of [0,1], where higher rank equals higher score based on a specific method. In addition to the methods, another column has been added to the results in Appendix F, the mean scores of the features. This mean score is the average performance of the feature over all the filter methods. The mean score is again normalized in the scale of [0,1] where the higher score refers to higher importance in predicting *price(t+1)*. For example, from Figure 6.1, the feature on capacity percentage in the East Pacific, *capacity_EstP_percent* at time (t-9), scores 0 with respect to lasso regression, 0.09 with linear correlation, 0.16 with linear

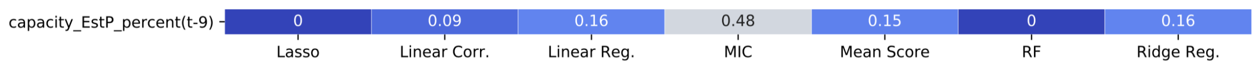


Figure 6.1: Arbitrary row of normalized features importance scores, from Appendix F

regression, 0.48 with MIC, 0 with random forests and 0.16 with ridge regression. This results in a mean score of 0.15.

Based on the mean scores for all the features, the best features using the proposed filter methods are overviewed in Figure 6.2. We observe that in the context of these specific filter methods, the best predictor of the Mont Belvieu price at time $t+1$, is the price at time t in addition to some other spot prices, the crude oil price and some capacity counting features. Namely the capacity in the Atlantic, and the percentage capacity in the East Pacific.

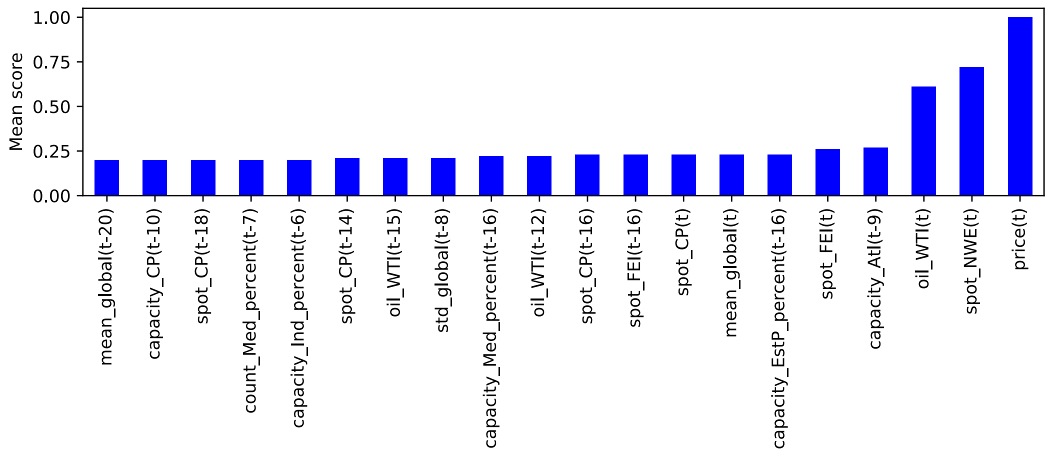


Figure 6.2: Top features based on mean features importance score

The last filter method introduced in Chapter 4, is the linear subset selection. Here, subsets with features are generated and compared with respect to different statistical measures. This method fits a linear regression model from Section 4.5.1 and estimates the parameters with Ordinary Least Squares (OLS) for each possible combination of features. The model introduced here selects the best subset in the context of linear regression. As we did with the linear regression filter method, we can investigate the magnitude of the coefficients of the linear model, so that the most important features should have the highest coefficients values given that the features are normalized. The number of features to include in the optimal subset was chosen based on Akaike Information Criterion (AIC), and the absolute value of the coefficients of these features are plotted in Figure 6.3, where only the best features are visualized. Again, we observe that the previous price is a good predictor in addition to spot prices and some capacity measures. This is further discussed in Chapter 7.

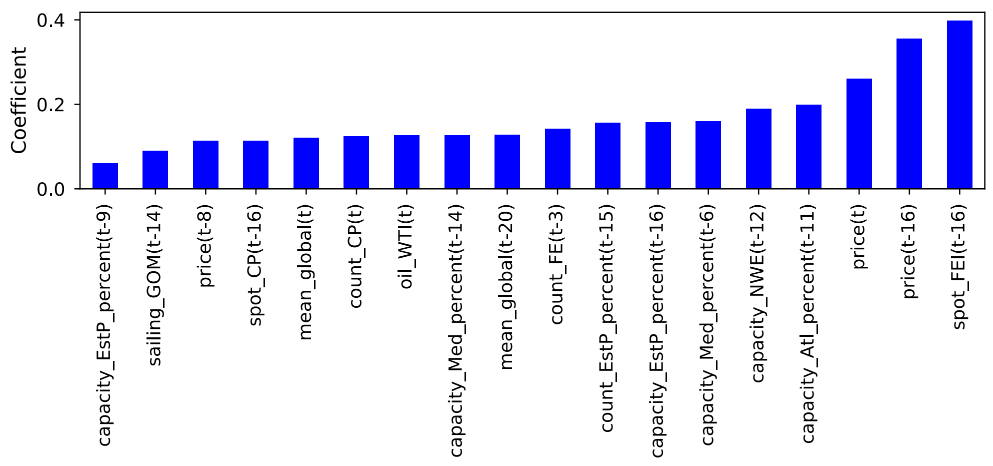


Figure 6.3: Top features from linear subset selection based on model coefficient absolute value

The main takeaway from the results of the filter methods, is that the historical price in addition to other spot prices and the crude oil price, receive high scores all through the selection. With respect to features extracted from AIS data, features that involve the counting of capacity scores high, likewise. These results are further discussed in Chapter 7.

6.1.2 Wrapper Selection

Based on the features with high scores from the filter selection methods, different combinations of these are tested with regard to the model that shall subsequently use them. First of all, the features with high scores are tested, but also tested in combination with other lower ranking features, as the mostly linear filter selection methods might not be able to capture some dual dynamics between features that the complex neural network is able to. The resulting best subset concerning model performance is overviewed in Table 6.1, and further discussed in Chapter 7.

Table 6.1: Best subset of AIS and non-AIS features from filter and wrapper selection

non-AIS subset	**AIS subset**
price (spot_MB)	*price (spot_MB)*
spot_NWE	*spot_NWE*
spot_FEI	*spot_FEI*
spot_CP	*spot_CP*
oil_WTI	*oil_WTI*
	capacity_NWE_percent
	capacity_Atl_percent
	capacity_EstP_percent

6.2 Genetic Algorithm Architecture

As described in Section 5.4, a genetic algorithm (GA) is used to optimize the hyperparameters of the model. Kaastra and Boyd (1996) thoroughly reviewed past literature on best hyperparameters in applications of neural networks in finance. The studies vary from stating that the number of hidden neurons should be 75 % of the number of input neurons, to indications that an optimal number of hidden neurons will generally be found between one-half to three times the number of input neurons. These findings set the boundaries for the search space of the GA. A common practice in time series forecasting using neural networks is also not to have the number of layer exceeding three layers. Kaastra and Boyd (1996) states that networks with one layer and a large number of hidden units are capable of approximating any continuous function, and that in practice, neural networks with one and occasionally two hidden layers are widely used and have performed very well. Further increasing the number of layers might lead to overfitting.

The window size can be estimated through the importance of lagged variables, but can also be a part of the GA as a decision parameter. To get a better solution, features selection methods are used to include the variable itself in the subset of features, while the GA is utilized to get the optimal window size based on this combination of variables. This because the complex neural network may be able to capture the non-linear dynamics between the features.

As described in Section 5.4, an individual in the GA is represented by a binary tuple. Assigning different positions of this tuple to different hyperparameters within their ranges, makes it possible to use GA for hyperparameter optimization. To get the hyperparameter value, the part of the individual's binary tuple representing this hyperparameter, is converted to an integer. There-after, for some of the parameters, a simple calculation is done to avoid pitfalls like the tuple becoming a binary string equal to zero. These calculations also makes the binary tuple shorter in length, thus providing a more efficient computational process. If the tuple becomes a binary string equal to zero, for those who have not an additional calculation of plus one, the fitness is set to an inferior value so that this individual has a low chance of reproduction.

When calculating the fitness, the binary tuple is converted to its respective hyperparameters, and a neural network with these parameters are trained on a part of the training set, and evaluated on the validation set. This performance on the validation set, measured by taking the MAE of the predictions, is the fitness of the individual. Thus lower fitness is better, so to make this a maximization problem to maximize fitness, the fitness is multiplied by negative one.

6.2.1 MLP Architecture

Table 6.2 overviews the hyperparameters and their respective binary tuple structure in the genetic algorithm for our Multilayer Perceptron (MLP) model. Also, the calculation of the converted integer and the resulting set and range of the hyperparameters are showed. The ranges are based on industry practice and previous literature. For example, if the sliding window binary tuple is [1,0,0,1,0], the corresponding window size is 19.

Table 6.2: Genetic algorithm decision parameter structure and corresponding range for MLP

Hyperparameter/component	Binary tuple structure	To integer	Calculation	Range/set
Sliding window size	$[ws, ws, ws, ws, ws]$	**ws**	**ws** + 1	[1, 2, ... , 32]
Number of hidden layers	$[ly, ly]$	**ly**	**ly**	[1, 2, 3]
Neurons in hidden layer 1	$[h1, h1, h1, h1, h1, h1]$	**h1**	(**h1** + 1) · 4	[4, 8, ... , 256]
Neurons in hidden layer 2	$[h2, h2, h2, h2, h2, h2]$	**h2**	(**h2** + 1) · 4	[4, 8, ... , 256]
Neurons in hidden layer 3	$[h3, h3, h3, h3, h3, h3]$	**h3**	(**h3** + 1) · 4	[4, 8, ... , 256]
Learning rate	$[lr]$	**lr**	$10^{-(\mathbf{lr}+1)}$	[0.01, 0.1]

If the optimal number of hidden layers is lower than two or three, the algorithm is designed to neglect the number of hidden neurons in the second and third layer, respectively. Combining all these binary tuple structures into single tuple results in a total tuple length of 25, also known as the generation length. Running the algorithm for 100 individuals, 20 generations and the tuple structure described above, the best performing individual is presented in Table 6.3. As stated in Section 5.4, the 5-10 best individuals for the GA are again tested and reviewed, so Table 6.3 presents the best of these individuals, the individual with best Mean Absolute Error (MAE) on the validation set. Different combinations of activation functions in the layers were also tested.

Table 6.3: Main characteristics of the best MLP

Hyperparameter/component	**Values/types**
Sliding window size	5
Number of hidden layers	3
Number of hidden neurons in layer 1	30
Number of hidden neurons in layer 2	6
Number of hidden neurons in layer 3	2
Activation function in hidden layers	Sigmoid
Learning rate	0.1

The genetic algorithm along with the MLP model can be found in the script *MLP.py* in Appendix E.10.

6.2.2 LSTM Architecture

Table 6.2 overviews the hyperparameters and their respective binary tuple structure in the genetic algorithm for the Long Short-Term Memory (LSTM) model. Also, the calculation of the converted integer and the resulting set and range of the hyperparameters are showed. The ranges are based on industry practice and previous literature. Because of the recurrent nature of LSTM in addition to model tests, no well-preforming architecture had more than two hidden layers. Therefore, the number of hidden layers range was reduced by one. The ranges of the number of neurons are also decreased because of the input difference between LSTM and MLP. In a MLP, the whole window for all features is feeded into the network, giving a number of input neurons equal to $(m+1)\cdot p$ for p features and a window size of m. In a LSTM each time step is subsequently feeded into the network resulting in the number of input neurons being equal to the number of features p.

Table 6.4: Genetic algorithm decision parameter structure and corresponding range for LSTM

Hyperparameter/component	**Binary tuple structure**	**To integer**	**Calculation**	**Range/set**
Sliding window size	$[ws, ws, ws, ws, ws]$	**ws**	**ws** + 1	[1, 2, ... , 32]
Number of hidden layers	$[ly]$	**ly**	**ly** + 1	[1, 2]
Neurons in hidden layer 1	$[h1, h1, h1, h1, h1]$	**h1**	(**h1** + 1) · 2	[2, 4, ... , 64]
Neurons in hidden layer 2	$[h2, h2, h2, h2, h2]$	**h2**	(**h2** + 1) · 2	[2, 4, ... , 64]
Learning rate	$[lr]$	**lr**	$10^{-(\mathbf{lr}+1)}$	[0.01, 0.1]

As for the MLP, combining all these binary tuple structures into single tuple results in a total generation length of 17. Running the algorithm for 100 individuals, 20 generations and the tuple structure described above, the best performing individual is presented in Table 6.5.

Table 6.5: Main characteristics of the best LSTM

Hyperparameter/component	**Values/types**
Sliding window size	27
Number of hidden layers	1
Number of hidden neurons in layer 1	40
Learning rate	0.1

The genetic algorithm along with the LSTM model can be found in the script *LSTM.py* in Appendix E.11.

6.3 Forecasting Results

As we forecast with the sliding window technique, when sliding the window one timestep at a time, we predict the next step based on the past features in the window. The plots in the following subsections are the realized inverse transformed series, as we forecast the difference normalized series in this study.

The neural network models will produce different results for separate training sessions with the same hyperparameters. This is a result of randomness when initializing the network. More specifically, the different initial configuration of weights in the network. Therefore the same models has been trained and tested 10 times with the same hyperparameters, to then take an average of the results.

6.3.1 Results of MLP

Training the MLP on the training set with the optimal hyperparameters found in Table 6.3 results in the one-step-ahead forecasts with non-AIS features in Figure 6.4, and AIS features in Figure 6.5. The code for the neural network model, *MLP.py*, can be found in Appendix E.10.

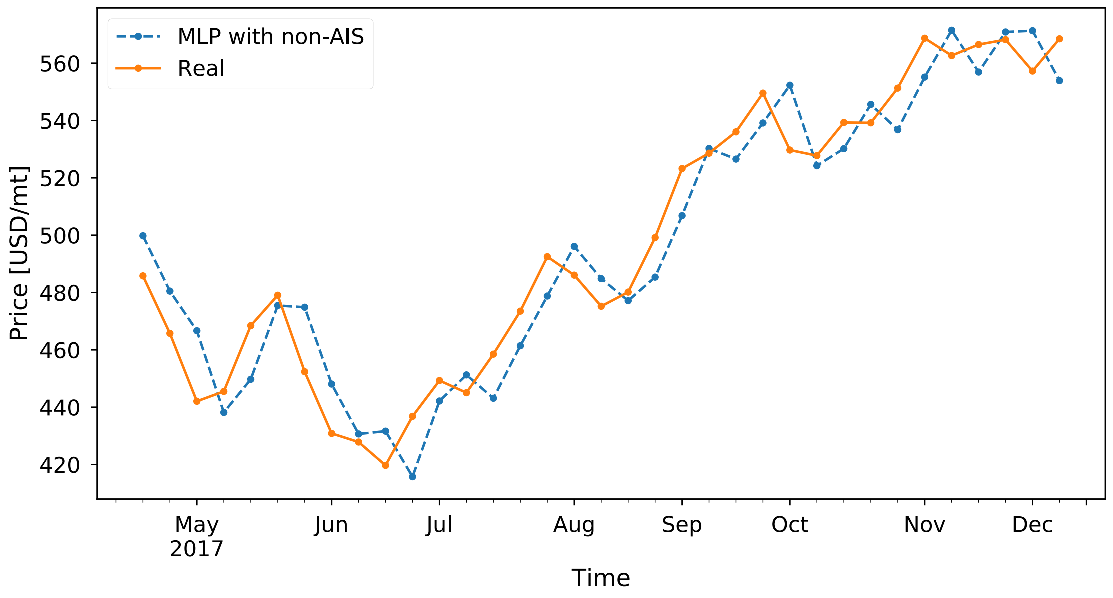


Figure 6.4: One-step-ahead weekly MLP forecasts with non-AIS features

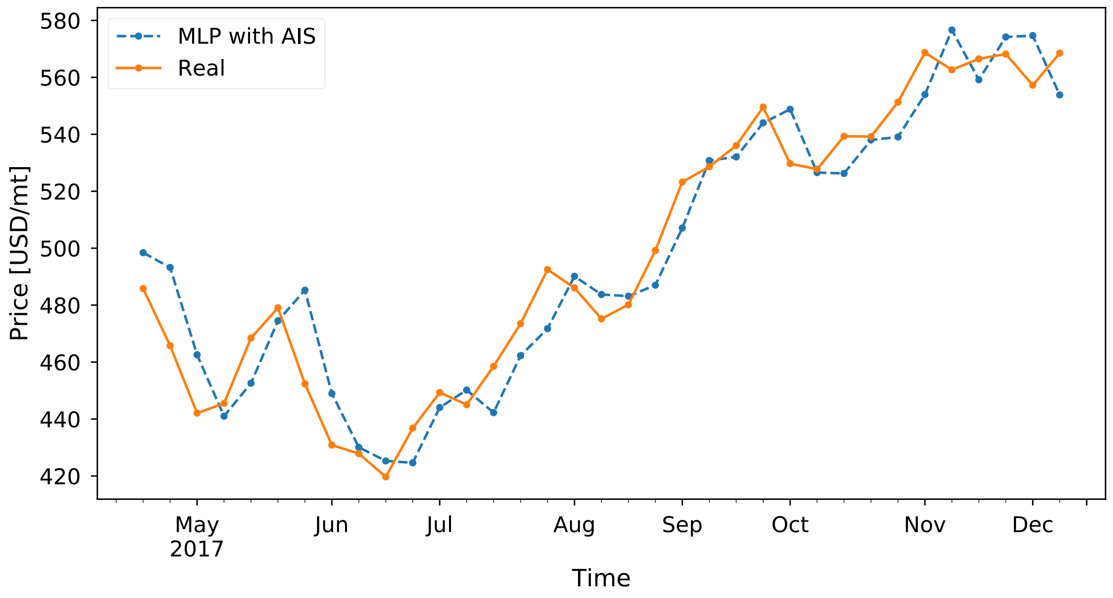


Figure 6.5: One-step-ahead weekly MLP forecasts with AIS features

6.3.2 Results of LSTM

Training the LSTM on the training set with the optimal hyperparameters found in Table 6.5 results in the one-step-ahead forecasts with non-AIS features in Figure 6.6, and AIS features in Figure 6.7. The code for the neural network model, *LSTM.py*, can be found in Appendix E.11.

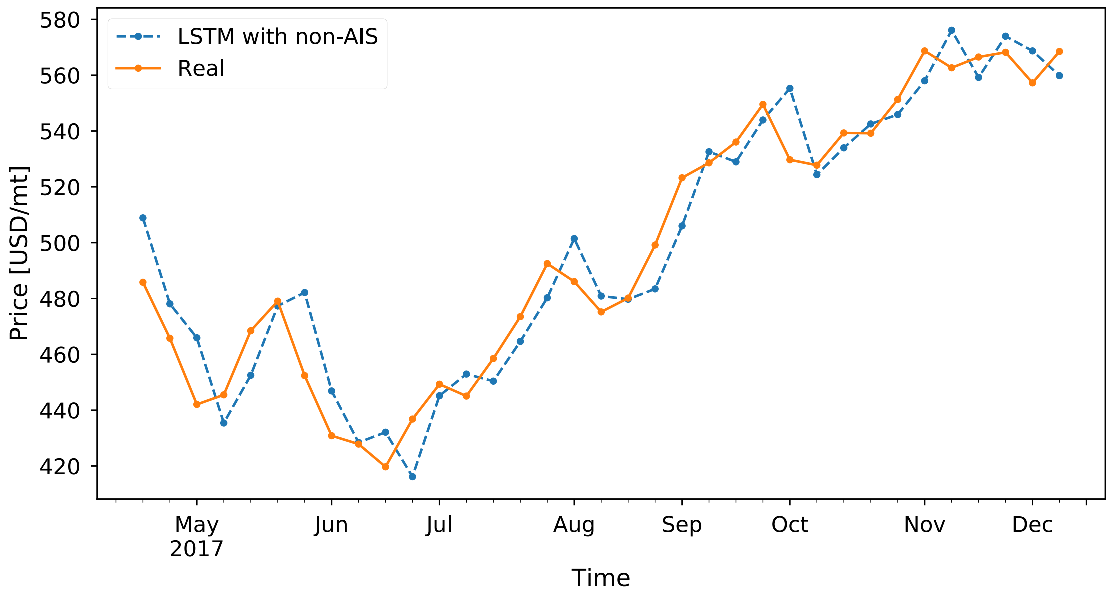


Figure 6.6: One-step-ahead weekly LSTM forecasts with non-AIS features

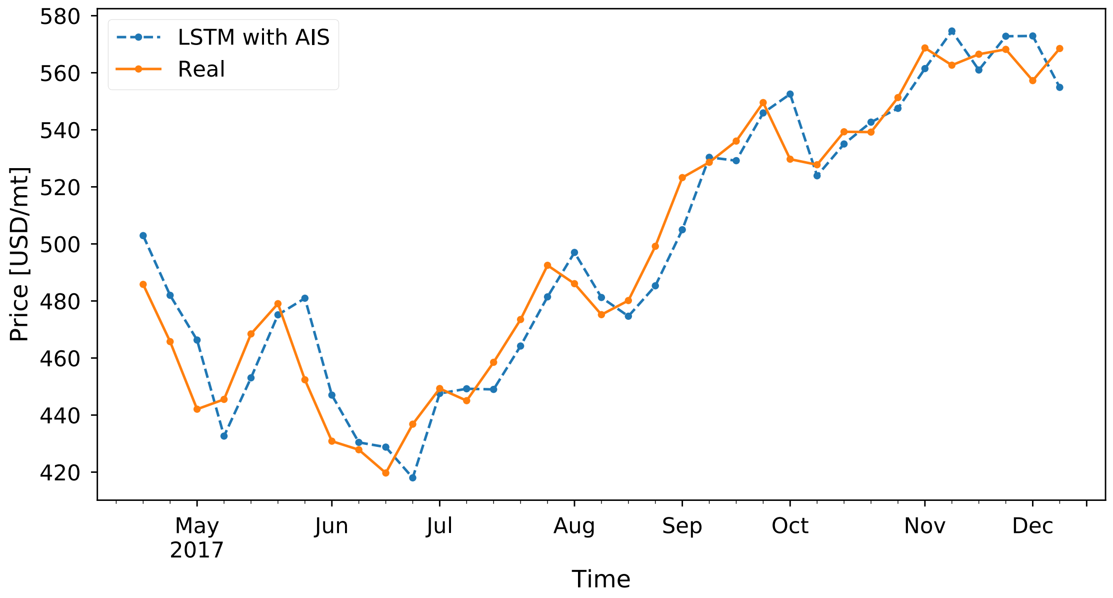


Figure 6.7: One-step-ahead weekly LSTM forecasts with AIS features

6.3.3 Results of VAR

The traditional multivariate Vector Autoregressive (VAR) model is widely used in multivariate forecasting in the maritime industry. It is used as a baseline for the forecasts in this study, in addition to the no-change model, to evaluate the performance of the machine learning models. As for the neural network models, the VAR-model is fitted on the training set and tested on the testing set. Using AIC as a selection criterion for the number of lags, we find that the optimal lag is of order 2. Predicting one-step-ahead with the sliding window method, yield the non-AIS features forecast in Figure 6.8, and AIS features in Figure 6.9. The code for the VAR-model, *VAR.py*, can be found in Appendix E.12.

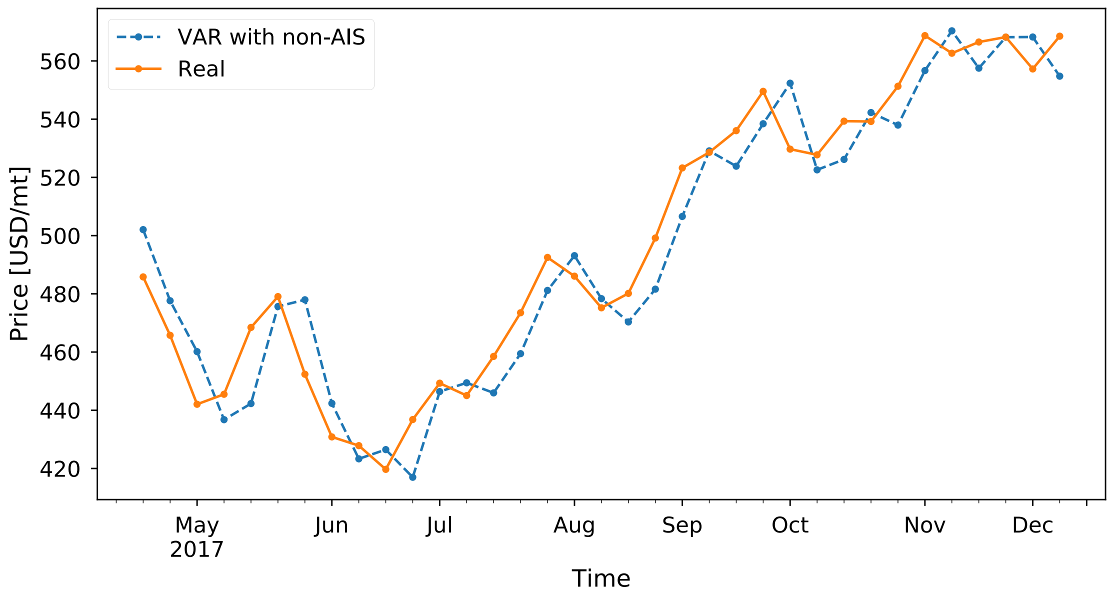


Figure 6.8: One-step-ahead weekly VAR forecasts with non-AIS features

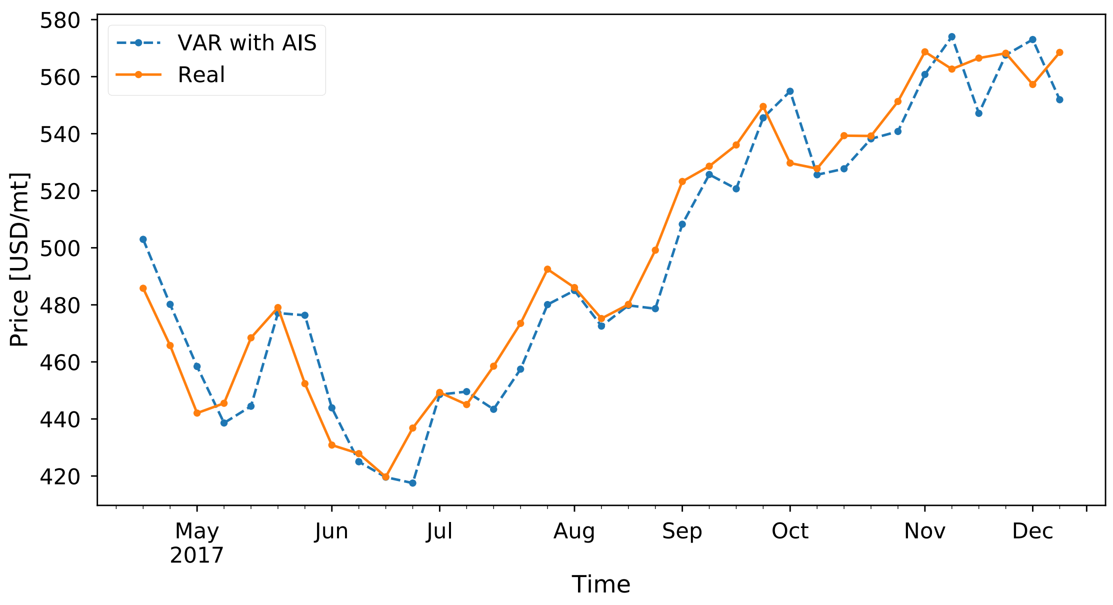


Figure 6.9: One-step-ahead weekly VAR forecasts with AIS features

7 | Discussion

This chapter starts by evaluating the results form the case study. In this way, we can adequately revise the methodology used in this study. Further, a general review of the methodology is presented and the underlying assumptions that have been made discussed.

7.1 Evaluation of Case Study

7.1.1 Evaluation of Forecast Results

Table 7.1 summarizes the results for predictions of the weekly average Mont Belvieu propane spot price. The summary includes all the prediction and baseline models with and without features extracted from AIS data. The values in the table measure the magnitude of the forecast error with Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE), and the directional accuracy measured by the Directional Accuracy Ratio (DAR). By comparing these metrics, we can provide evidence for the predictive ability of each approach. The evaluation of the forecasts is done by the Python script *Model_evaluation.py* in Appendix E.13.

Table 7.1: Performance metrics for the different models on the testing set

Model	**RMSE [USD/mt]**	**MAE [USD/mt]**	**MAPE [%]**	**DAR [-]**
Persistence	14.15	12.15	2.49	0.5
MLP with non-AIS	13.08	11.63	2.37	0.57
MLP with AIS	13.48	11.19	2.28	**0.65**
LSTM with non-AIS	13.03	10.83	2.23	0.63
LSTM with AIS	**12.46**	**10.41**	**2.14**	0.60
VAR with non-AIS	13.17	11.05	2.25	0.54
VAR with AIS	12.83	10.64	2.15	0.57

Note: Boldface indicates the best performance score

Comparing the performance metrics for all the models, we find them to outperform the no-change persistence model along all metrics considered significantly. This indicates that the

Mont Belvieu propane spot price is to some extent predictable in weekly forecasting horizons, utilizing predictors including market features with or without features extracted from AIS data. The above results are indicative of the superiority of the Long Short-Term Memory (LSTM) model with AIS data on all the magnitude error metrics. While in the case of DAR, the directional accuracy, the Multilayer Perceptron (MLP) has the best performance. However, the performance of the MLP is slightly worse than that of Vector Autoregressive (VAR) and LSTM on the other metrics. The good performance of DAR can be due to randomness, as the testing set is relatively small and has a distinctly upwards trend. To get a higher quality result, a more extended dataset will yield more accurate evaluation. However, as quality AIS data is a relatively new development, this was not possible to achieve in this study.

To compare how the performance of the models are against the persistence model, we can investigate the relative metrics for the magnitude errors. This is viewed in Table C.5. In this way, we observe that for example by looking at MAE, the LSTM model performs 14.3 % better than the persistence model, with AIS data, and 10.8 % better than the persistence model, without AIS data.

Table 7.2: Relative performance of the different models to the persistence model

Model	**Rel. RMSE**	**Rel. MAE**	**Rel. MAPE**
Persistence	1	1	1
MLP with non-AIS	0.924	0.957	0.953
MLP with AIS	0.952	0.921	0.914
LSTM with non-AIS	0.921	0.892	0.893
LSTM with AIS	**0.881**	**0.857**	**0.859**
VAR with non-AIS	0.907	0.909	0.862
VAR with AIS	0.930	0.876	0.902

Note: Boldface indicates the best performance score

Furthermore, to better visualize the performance of the different models, we have sorted and plotted the relative metrics in Figures 7.1-7.3.

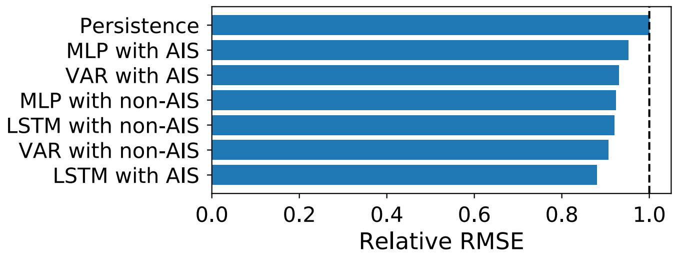


Figure 7.1: Sorted relative RMSE against the persistence model

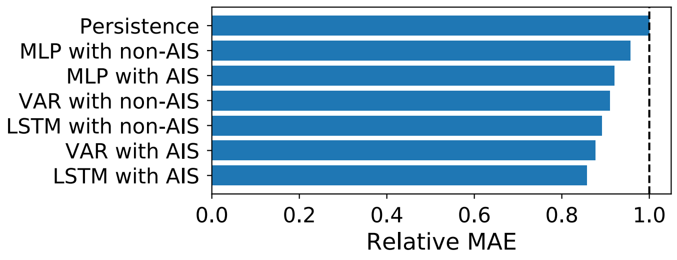


Figure 7.2: Sorted relative MAE against the persistence model

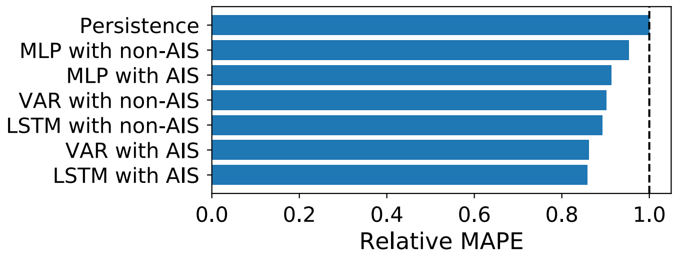


Figure 7.3: Sorted relative MAPE against the persistence model

The first objective of this study was to investigate if AIS data adds additional information in predicting short-term freight rates. From the tables and figures above, we see that both the LSTM and traditional VAR-model perform better with the subset including AIS data. However, further model tuning and higher quality of AIS data may return different feature qualities which again may change the results.

The secondary objective of the study was to investigate if machine learning models achieve higher performance than traditional models in predicting short-term freight rates. From the analysis above, we identify the superior performance of the LSTM model with AIS data. Moreover, concerning MAE and MAPE, the LSTM model outperforms the VAR-model on both subsets of features. With respect to RMSE, the VAR without AIS data performs better than the LSTM without AIS. This performance may be due to the neural network models being trained with MAE as a loss function. Thus, the optimal model selected by the genetic algorithm is the model with the lowest MAE on the validation set. The MAE is better in describing average error while the RMSE has the benefit of penalizing more substantial errors because it is more sensitive to outliers. From a regression standpoint, MAE is considered a better metric as RMSE does not describe average error alone. However, with the testing set only including 34 observations, we can not state for that the results will be identical, or significantly in favor of LSTM, on a different training set or longer time history.

Another important remark is that the MLP-model is not able to generalize as good as the LSTM-model, and the model performs worse than the VAR-model in most of the metrics considered. Further model tuning could produce better results. Still, we observe that the MLP with AIS data performs better than the non-AIS considering MAE and MAPE. On the other note, in the case of RMSE, the performance is worse. This again understates the model training with MAE and the difference in error interpretation between MAE and RMSE.

7.1.2 Evaluation of Features Selected

Evaluating the results of the filter methods in Appendix F, the scores vary not only between the linear and non-linear methods but also between the univariate and multivariate methods. Linear and Ridge regression, both being linear multivariate methods, have similar trends in the resulting scores. While, linear correlation, also being linear, have some different results from these two as this method is univariate. The Lasso regression (L1 regularization) have many features receiving the score of zero, something that is the result of the Lasso minimizing the less important feature's coefficients β to zero, thus removing some features altogether. On the contrary, Ridge regression (L2 regularization) does not have this element. Maximal Information Coefficient (MIC) and Random Forests are the only methods considered that can capture non-linear relationships. Of these, the MIC is univariate while Random Forests are multivariate. We observe in the results of the MIC that many counting and capacity features receive high scores, identifying a possible non-linear relationship between vessel positioning and the Mont Belvieu propane spot price. The resulting best AIS-subset also incorporated these types of features.

Concerning the features used in forecasting, not all features created were found to be good predictors of the spot price, even though some of the literature states otherwise. For example the theory about the causal relationship between speed and freight rates. The LPG marked being dictated by geographical price arbitrage and a small fleet (Adland et al., 2008), the speed might not be a substantial market driver. Tsioumas (2016) and Ronen (1982) states that shipowners tend to speed up when rates are high to get the most out of the favorable market conditions. This argument favor that freight rates lead to changes in speed. However, concerning the AIS data used, some of the data have poor quality with gaps, especially in the early stages before 2013. In addition to the speed being acquired from single AIS signals, we do not have an overall correct speed profile of the fleet. The speed acquired from AIS data is the Speed Over Ground (SOG), and not speed through water, something that also influences this feature.

Counting vessels in large areas have little room for errors and is a straightforward feature extraction task. This might be why the features based on the percentage of fleet capacity in polygons are good predictors, in the context of AIS data quality. Furthermore, these features take the change in fleet size into account, which again gives a better picture of the supply of vessels. The specific features that gave the best result were the percentage capacity counting in the North West Europe, Atlantic, and East Pacific. All of these areas are close to Mont Belvieu in Texas. Thereby, knowing the percentage of ships in the surrounding areas close to the port, was useful information. However, the capacity features created for the Gulf of Mexico did not show this promise. This might be due to a smaller polygon relative to the other polygons. Mont Belvieu is located in this small area, so knowing the weekly fleet density so close to the loading port may be too late in a contract horizon. Also, as the Gulf of Mexico is a high traffic area of vessels, inference errors may have occurred.

Another feature that did not result in more prediction power is the weighted arithmetic mean sailing distance from the whole fleet to Mont Belvieu. Smaller vessels that are continuously sailing in smaller markets on the other side of the globe may be the reason for the poor predictor, damaging some of these results. Another important mention is data quality, as this feature is very sensitive to gaps. Also, as proposed by Vit Prochazka, knowing the vessel position at the point of contract fixation may provide useful input. This could have been done by merging external contract fixtures data with AIS data to find the geographical position of a vessel at the moment it is fixed. Knowing these positions gives an idea about the supply side of the market, how it is changing with different market conditions and where to count the number of vessels. One could then calculate the distance from the position to the loading port and further investigate high-density areas of contract fixation positions. However, we were unable to acquire such contract fixture data. In the context of sailing distances, one could measure weekly sailing distance for the uncontracted vessels, or vessels close to the end of a contract.

Concerning the non-AIS features, we observe from the filter methods and the optimal subset that the other propane spot prices provided valuable information in predicting the Mont Belvieu spot price. This may indicate that some rates discover market information before others, and as it is the price of the same commodity, the relationship between the features is coherent. The

West Texas Intermediate (WTI) Cushing, Oklahoma, crude oil price also scores well in the features selection process. This may be due to the distinctive feature of the LPG shipping market, being driven by “supply push” rather than “demand pull”. The main forwarders of LPG are dominated by the major oil and gas companies, and as LPG is not a raw material but extracted as a by-product of gas processing and crude oil refining, the crude oil price also seems like a good predictor.

7.2 Methodology

7.2.1 Forecasting and Study Horizon

A distinctive characteristic of the maritime shipping industry is the shipping cycles. These cycles arise as a result of times with low freight rates, resulting in less construction in the maritime sector and increasing the number of scrapped vessels. As demand increases and more transport services are needed, the supply cannot be adjusted rapidly, freight rates rise, and construction of new vessels starts again, which subsequently produces excess supply and a lowering of freight rates. The duration of the cycles is highly unpredictable, but studies show that a typical cycle lasts about seven years on average (Stopford, 2009). As the use of S-AIS data to provide useful insight is a relatively new concept in contrast to the old nature of the shipping industry, in the worst case the data collected can be within one of these cycles. Ideally, the data collected should reflect a whole shipping cycle to capture the entire market dynamics. However, for all practical considerations, this is impossible with the data at hand. Thus, as more AIS data is acquired and the collection reflects longer history, the potential advantage of using AIS data to predict freight rates increases.

The forecasting horizon chosen in this study is week-to-week one step ahead forecasts. Also, we forecast the average weekly prices. We reviewed the problem of investigating daily fluctuations in Section 2.2 as we obtain daily price reports. We found that looking at daily vessel positioning may exclude some vessels for which AIS data was not recorded this specific day. Furthermore, interpolating the AIS data to daily observations was also deemed unfit as this might include vessels being laid up over a substantial period or, for large gaps, for example causing the interpolation to give vessel signals over land areas. Another option was to look at monthly averages. However, concerning using AIS-features in predicting freight rates, in a month, a vessel may have sailed a long distance. Accordingly, looking at average vessel positioning within a month may give misleading or averaged out effects. Hence the shorter weekly forecasting window used in this study is evaluated to be sufficient, and the best alternative with the data at hand.

7.2.2 Vessel Search

Reviewed in Section 3.3.1, the heuristic vessel classification method proposed by Smestad and Rødseth (2015) only captures larger vessels. Because we in this study want to capture the whole LPG fleet, we developed a new vessel search method. However, this method requires additional data, in contrast to the approach by Smestad and Rødseth (2015). We utilize data from Seaweb, and find 1469 unique vessels, based on IMO numbers. The database also includes MMSI numbers for these vessels, but only using these will give a faulty vessel set. The method introduced is based on using the fact that IMO numbers follow the hull of the vessel through its time span. We find 1714 unique MMSI numbers from the respective IMO numbers. The main disadvantage of this method is that we do not know if some interference error or erroneous signal reported a wrong MMSI. Thereby identifying a non-LPG vessel and including it in the set. However, based on the overall small fleet size, we assume this error to be relatively low and would not have a large impact on the features created. In general, we find the vessel search method to be of added value in the methodology.

7.2.3 Feature Engineering

As stated by Ng (2011), features selection is an essential part of machine learning. However, if the data used to create these features is bad, we can not expect the features selection methods to perform well. One good example of this is the speed features created in the case study. In Section 4.3.4, we assumed that the speed reporting is consciously distributed over the operational time of the vessels to best reflect the real speed of the fleet. However, because AIS is sent by signals, and the signal obtained in this study are not continuously distributed, the speed collecting may have changed over time. This for example coming from a change in satellite coverage, less interference over time or more signals in high activity areas like ports or close to shore. Based on the training and testing sets, this element may have a faulty effect on the machine learning models. Training and validating the results on two types of data series will cause the model to wrongly set the weights and biases, giving a poor performance on the validation set. This may be the reason for the bad performance of the sailing speed and deviation features. Showing that, since the underlying data collection that the features are based on changes, the consistency of the feature are not kept, making the assumption that the difference between speeds at different time points reflect the real speed difference, nonsufficient. Leonhardsen (2017) found that the speed through water (STW) showed less volatility than the speed over ground (SOG). Ronen (1982) states that the alternative cost of time exceeds the gains from fuel optimization when rates are high. This theory promotes a higher sailing speed. As the best measure for the resistance of the vessel, and subsequently the fuel consumption, is given by STW, this is again affecting the reliability of the speed features extracted from AIS data.

In the context of the Mont Belvieu spot price, and the LPG market, AIS data seem to add additional information in prediction the spot price. However, this might not be the case in other industries, as creating features is a market specific question. The LPG market has some distinctive features, one being driven by supply push rather than demand pull. Another distinctive characteristic is that the changes in the LPG spot market demand may be dictated by the presence of geographical price arbitrage (Adland et al., 2008). The latter may be an important pinpoint to why information about global vessel positioning gave better results for the prediction models.

The features selected in the case study are thoroughly discussed in Section 7.1.2. The features selection methods utilized in this study incorporates a wide range of methods, being both linear and non-linear, but also univariate and multivariate. With the diverse results from the filter methods, we see it advantageous to combine these methods to get a better overview of the potential underlying predictive ability. By utilizing different methods, we can evaluate the results of each one, increasing the decision support for which features to subsequently select. Using the wrapper method by evaluating different subset combinations based on the best filter features, we achieve forecasts that are better than the persistence model. With this combination of methods in an iterative process, we find it favorable to utilize this approach.

7.2.4 Machine Learning Process

It is essential to keep in mind that we forecast the transformed series, differenced and normalized, from Section 4.4. We therefore try to learn the models to generalize the weekly difference, and subsequently inverse transform the series to its original form after the predictions to evaluate the results. To just do this, in this way, might give misleading performance results. By looking at the forecast plot in Figures 6.4-6.9 we see that the fit between the forecasts and real values looks good. However, this might not be the case, because visualizing the inverse transformation can give an optimistic picture of the performance. To picture this, we have plotted the no-change persistence prediction in Figure 7.4, which visually also seems to fit. Actually, the persistence model is in theory not able to generalize the problem, as the previous observation is just carried one step forward as a forecast. However, in this study, we use the persistence model as a baseline for the model performances. Thus, by comparing the model results to the persistence model, we explicitly evaluate if our model performance is better than a coin toss. Therefore, we deem it fit to use this forecasting and evaluation approach.

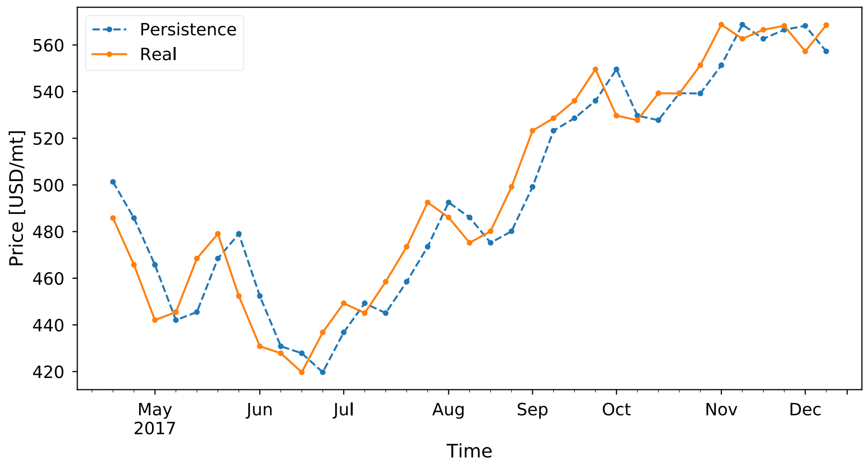


Figure 7.4: Visualization of persistence model forecast

Using a genetic algorithm to decide the optimal network architectures, proved to be of added value in forecasting freight rates. Other research utilizes grid search methods, architectures previously used by other papers or simple rules of thumb. We have no knowledge of other studies utilizing genetic algorithms in the context of forecasting shipping commodities with machine learning. The algorithm starts by randomly creating individuals. The advantage of using a genetic algorithm is that it is easy to adapt and it is not required to model the function behavior, and the mutation prevents the function to get trapped in local optima. However, we found the method to be computationally intensive for complex fitness evaluation as neural network training. We were therefor only able to generate 100 individual initial solutions over 20 generations. However, given the promising results discussed in Section 7.1.1, we find this to be sufficient for this scope of the study.

As discussed in Section 7.1.1, we achieved bad performance with the MLP-model. Bad relative to the traditional VAR-model and the LSTM-model, but better than the baseline persistence model. Further model tuning may increase the predictive ability of this model, which in general also relates to the LSTM-model. We were not able to fully optimize the model tuning in this study. This because the scope of study is rather large and investigative, including data cleaning and vessel search, exploratory analysis, feature engineering and several machine learning methods.

8 | Conclusion

8.1 Concluding Remarks

The objective of this thesis has been to investigate whether multivariate machine learning forecasting methods using features extracted from Automatic Identification System (AIS) data adds additional information in predicting short-term freight rates. Specifically, we have assessed this with a case study on the Liquid Petroleum Gas (LPG) shipping market directed on the weekly prediction of the Mont Belvieu propane spot price. Forecasting LPG spot prices is a very challenging problem due to the high volatility of spot prices. Nonetheless, anticipating such fluctuations is a crucial element to long-term profitability for both operators and shipowners.

Overall, the results from our final analysis establish evidence in favor of using features extracted from AIS data in freight rate prediction. Furthermore, the Long Short-Term Memory (LSTM) neural network model performs better than the baseline no-change persistence model and the traditional multivariate Vector Autoregressive (VAR) model. This result suggests favourability of using multivariate machine learning forecasting methods in short-term freight rate prediction. Even though the project results show promise, we acknowledge that there are several limitations to our research, especially concerning the quality of the AIS data. The quality of the data at hand varies, mostly due to interference errors and gaps in the early stages as a result of satellite coverage. This should be put into consideration before any studies using AIS data are conducted. Accordingly, as more AIS data is acquired and the collection reflects longer history, the potential advantage of using AIS data to predict freight rates increases.

There are some important takeaways from the methodology. First of all, we find the vessel search methodology introduced to be of added value. We found 1714 unique Maritime Mobile Service Identity (MMSI) numbers from 1469 International Maritime Organization (IMO) numbers obtained form Seaweb. With this, we capture most of the market through the horizon of the study. Secondly, the combinational feature selection approach, utilizing different filter methods with the iterative element of the wrapper method, is found to be favorable as it increases the decision support for which features to subsequently select. Thirdly, using a genetic algorithm to decide the optimal network architectures proved to be sufficient, as the mutation prevents the function to get trapped in local optima, resulting in an LSTM architecture with superior performance.

8.2 Recommendations for Further Work

Further research within the use of AIS data in the context of maritime economics is recommended. Based on the finding in this study, and the relatively unexplored field of combining AIS data with freight rates, there is potential in exploiting the information incorporated in AIS-messages. With the numerous material in both static and dynamic messages, more sophisticated features can be created. As more AIS data is acquired and the collection reflects longer history with higher quality data, elements like draught and course information can be exploited.

We also acknowledge that with the scope of this study being wide, further development of each step in the methodology can improve model performance. It is finally recommended that, after surveying the performance of the methodology in the case study, to investigate whether the approach used in this study applies to other ocean freight industries. Deciding which features to construct is a market specific question, and understanding the market dynamics is equally as important as constructing complex features.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. *Brain, Google*.

Adland, R., Jia, H., and Lu, J. (2008). Price dynamics in the market for Liquid Petroleum Gas transport. *Energy Economics*, 30(3):818–828.

Adland, R., Jia, H., and Strandenes, S. P. (2017). Are AIS-based trade volume estimates reliable? The case of crude oil exports. *Maritime Policy and Management*.

Alquist, R., Kilian, L., and Vigfusson, R. J. (2013). Forecasting the Price of Oil. *Handbook of Economic Forecasting*, 2:427–507.

Arguedas, V. F., Pallotta, G., and Vespe, M. (2014). Automatic generation of geographical networks for maritime traffic surveillance - Semantic Scholar. *17th International Conference on Information Fusion (FUSION)*, pages 1–8.

Assmann, L., Andersson, J., and Eskeland, G. S. (2015). Missing in Action? Speed optimization and slow steaming in maritime shipping.

Bai, X. and Lam, J. S. L. (2017). An integrated analysis of interrelationships within the very large gas carrier (VLGC) shipping market. *Maritime Economics & Logistics*, pages 1–18.

Batchelor, R., Alizadeh, A., and Visvikis, I. (2007). Forecasting spot and forward prices in the international freight market. *International Journal of Forecasting*.

Bellman, R. (1972). *Dynamic programming*. Princeton University Press.

Bermingham, M. L., Pong-Wong, R., Spiliopoulou, A., Hayward, C., Rudan, I., Campbell, H., Wright, A. F., Wilson, J. F., Agakov, F., Navarro, P., and Haley, C. S. (2015). Application of high-dimensional feature selection: evaluation for genomic prediction in man. *Scientific Reports*, 5(1):10312.

Bodenhofer, U. (2003). Genetic Algorithms: Theory and Applications. (Third Edition—Winter 2003/2004).

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

Brooks, C. (2002). *Introductory econometrics for finance*. Cambridge University press, Cambridge, second edition edition.

Canu, S. (2014). Tuning hyperparameters using cross validation.

Chollet, F. (2015). Keras.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*. John Wiley & Sons, Inc., Hoboken, New Jersey, second edition edition.

da S. Gomes, G. S., Ludermir, T. B., and Lima, L. M. M. R. (2011). Comparison of new activation functions in neural network for forecasting financial time series. *Neural Computing and Applications*, 20(3):417–439.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.

Domingos, P. (2015). A Few Useful Things to Know about Machine Learning.

Dooho, C. (2013). The effect of shale gas revolution on oil industry. *IEEJ*.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179–211.

Engelen, S. and Dullaert, W. (2010). Transformations in gas shipping: Market structure and efficiency. *Maritime Economics & Logistics*, 12(3):295–325.

Eriksen, T., Skauen, A. N., Narheim, B., Helleren, O., Olsen, O., and Olsen, R. B. (2010). Tracking ship traffic with Space-Based AIS: Experience gained in first months of operations. In *2010 International WaterSide Security Conference*, pages 1–8. IEEE.

Fuller, W. A. (1996). *Introduction to statistical time series*. Wiley, 2nd edition edition.

Furnival, G. M. and Wilson, R. W. (1974). Regressions by Leaps and Bounds. *Technometrics*, 16(4):499–511.

Gao, S. and Lei, Y. (2017). A new approach for crude oil price prediction based on stream learning. *Geoscience Frontiers*, 8(1):183–187.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A Search Space Odyssey. 2.

Haji, S., O'Keeffe, E., and Smith, T. (2013). Estimating the global container shipping network using data and models. *Low Carbon Shipping 2013*.

Han, Q., Yan, B., Ning, G., and Yu, B. (2014). Forecasting Dry Bulk Freight Index with Improved SVM. *Mathematical Problems in Engineering*, 2014:1–12.

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

IMO (1974). International Convention For The Safety of Life At Sea.

IMO (2016). Revised guidelines for the onboard operational use of shipborne Automatic Identification Systems (AIS). Technical report.

Itu-R (2014). Technical characteristics for an automatic identification system using time division multiple access in the VHF maritime mobile frequency band M Series Mobile, radiodetermination, amateur and related satellite services. pages 1371–5.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY.

Jia, H., Adland, R., Prakash, V., and Smith, T. (2017). Energy efficiency with the application of Virtual Arrival policy. *Transportation Research Part D: Transport and Environment*, 54:50–60.

Kaastra, I. and Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236.

Kaluza, P., Kölzsch, A., Gastner, M. T., and Blasius, B. (2010). The complex network of global cargo ship movements. *Journal of the Royal Society, Interface*, 7(48):1093–103.

Kavussanos, M. G. and Nomikos, N. K. (2003). Price Discovery, Causality and Forecasting in the Freight Futures Market. *Review of Derivatives Research*, 6:203–230.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization.

Kojadinovic, I. and Wottka, T. (2000). Comparison between a filter and a wrapper approach to variable subset selection in regression problems.

Kulkarni, S. and Haidar, I. (2009). Forecasting Model for Crude Oil Price Using Artificial Neural Networks and Commodity Futures Prices. *International Journal of Computer Science and Information Security (IJCSIS)*, 2(1).

Lane, B. C. (2006). AIS Parser SDK - Accessed 2018-02-03.

Leonhardsen, J. H. (2017). Estimation of Fuel Savings from Rapidly Reconfigurable Bulbous Bows Exemplifying the Value of Agility in Marine Systems Design.

Li, J. and Parsons, M. G. (1997). Forecasting tanker freight rate using neural networks. *Maritime Policy & Management*, 24(1):9–30.

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., and Hodjat, B. (2017). Evolving Deep Neural Networks.

Næss, P. A., Grundt, E. H., and Axelsen, J. J. (2017). Exploration of Methods for Analysing AIS Data. (December).

Ng, A. (2011). Machine Learning and AI via Brain simulations.

Norwegian Space Centre. New Norwegian satellite to detect radar signals - https://www.romsenter.no/eng/News/News/New-Norwegian-satellite-to-detect-radar-signals - Accessed 2018-02-12.

Olah, C. (2015). Understanding LSTM Networks.

Olsen, M. and da Fonseca, T. R. K. (2017). Investigating the Predictive Ability of AIS-data: The case of Arabian Gulf tanker rates.

O'brien, R. M. (2007). A Caution Regarding Rules of Thumb for Variance Inflation Factors. *Quality & Quantity*, 41(5):673–690.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830.

Pošík, P. (2015). Feature selection and extraction.

Refenes, P. (1995). *Neural networks in the capital markets.* John Wiley & Sons.

Reshef, D. N., Reshef, Y. A., Finucane, H. K., Grossman, S. R., McVean, G., Turnbaugh, P. J., Lander, E. S., Mitzenmacher, M., and Sabeti, P. C. (2011). Detecting novel associations in large data sets. *Science (New York, N.Y.)*, 334(6062):1518–24.

Ronen, D. (1982). The Effect of Oil Price on the Optimal Speed of Ships. *Journal of the Operational Research Society*, 33(11):1035–1040.

Seaweb. IHS Maritime and Trade - Accessed 2018-03-01.

Shapiro, A. F. (2003). Capital Market Applications of Neural Networks, Fuzzy Logic and Genetic Algorithms.

Skauen, A. N., Helleren, , Olsen, , and Olsen, R. (2013). *Operator and User Perspective of Fractionated AIS Satellite Systems.* PhD thesis.

Smestad, B. B. and Rødseth, J. (2015). A Study of Satellite AIS Data and the Global Ship Traffic Through the Singapore Strait.

Spiliopoulos, G., Zissis, D., and Chatzikokolakis, K. (2017). A Big Data Driven Approach to Extracting Global Trade Patterns. pages 109–121. Springer, Cham.

Stopford, M. (2009). *Maritime economics*. Routledge Taylor Francis Group, 3rd edition.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era.

Sutherland, E. E., Sproull, R. F., and Schumacker, R. A. (1974). A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys*, 6(1):1–55.

Tsioumas, V. (2016). *Quantitative analysis of the dry bulk freight market, including forecasting and decision making*. PhD thesis, Department of Maritime Studies University of Piraeus.

U.S. Department of Energy (2018). Energy Information Administration, Independent Statistics and Analysis.

Vafaeipour, M., Rahbari, O., Rosen, M. A., Fazelpour, F., and Ansarirad, P. (2014). Application of sliding window technique for prediction of wind velocity time series.

Wu, L., Xu, Y., Wang, Q., Wang, F., and Xu, Z. (2017). Mapping Global Shipping Density from AIS Data. *Journal of Navigation*, 70(01):67–81.

Xia Zhang, Hong Yin, Changbo Wang, Jin Wang, and Yanping Zhang (2015). Forecast the price of chemical products with multivariate data. In *2015 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC)*, pages 76–82. IEEE.

Yu, L., Wang, S., and Lai, K. K. (2008). Forecasting crude oil price with an EMD-based neural network ensemble learning paradigm. *Energy Economics*, 30(5):2623–2635.

Appendix A

AIS Data Contents

Detailed information of the AIS information transmitted by a ship, as issued by IMO (2016).

Table A.1: Information on static messages

Information item	**Information generation, type and quality of information**
MMSI	Set on installation
Call sign and name	Set on installation
IMO Number	Set on installation
Length and beam	Set on installation
Type of ship	Select from pre-installed list
Location of position-fixing antenna	Set on installation

Table A.2: Information on dynamic messages

Information item	**Information generation, type and quality of information**
Ship's position with accuracy indication and integrity status	Automatically updated from the position sensor connected to AIS. The accuracy indication is for better or worse than 10 m
Position Time stamp in UTC	Automatically updated from ship's main position sensor connected to AIS.
Course over ground (COG)	Automatically updated from ship's main position sensor connected to AIS, if that sensor calculates COG. This information might not be available.
Speed over ground (SOG)	Automatically updated from the position sensor connected to AIS.
Heading	Automatically updated from the ship's heading sensor connected to AIS.
Navigational status	Navigational status information has to be manually entered by the OOW and changed, as necessary, for example: - underway by engines - at anchor - not under command (NUC) - restricted in ability to manoeuvre (RIATM) - moored - constrained by draught - aground - engaged in fishing - underway by sail In practice, since all these relate to the COLREGS, any change that is needed could be undertaken at the same time that the lights or shapes were changed. [1]
Rate of turn (ROT)	Automatically updated from the ship's ROT sensor or derived from the gyro. This information might not be available

Table A.3: Information on voyage related messages

Information item	**Information generation, type and quality of information**
Ship's draught	To be manually entered at the start of the voyage using the maximum draught for the voyage and amended as required. (e.g. – result of de-ballasting prior to port entry.)
Hazardous cargo (type)	To be manually entered at the start of the voyage confirming whether or not hazardous cargo is being carried, namely: - DG (Dangerous goods) - HS (Harmful substances) - MP (Marine pollutants) Indications of quantities are not required.
Destination and ETA	To be manually entered at the start of the voyage and kept up to date as necessary.
Route plan (waypoints)	To be manually entered at the start of the voyage, at the discretion of the master and updated when required.

Appendix B

Density Line Plots

B.1 Density Line Plot of VLGC

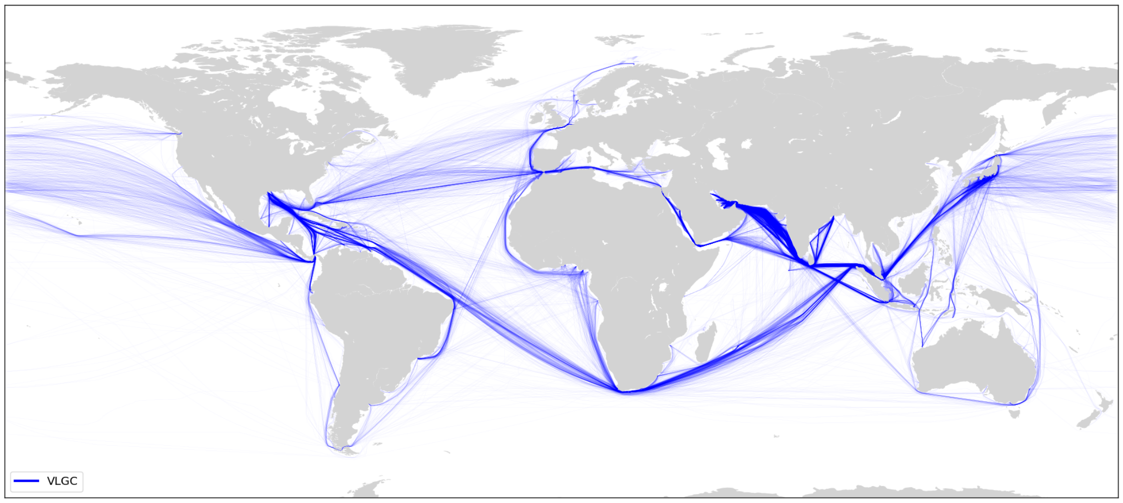


Figure B.1: Density line plot of VLGC, 2011-2017

B.2 Density Line Plot of Smaller Vessels

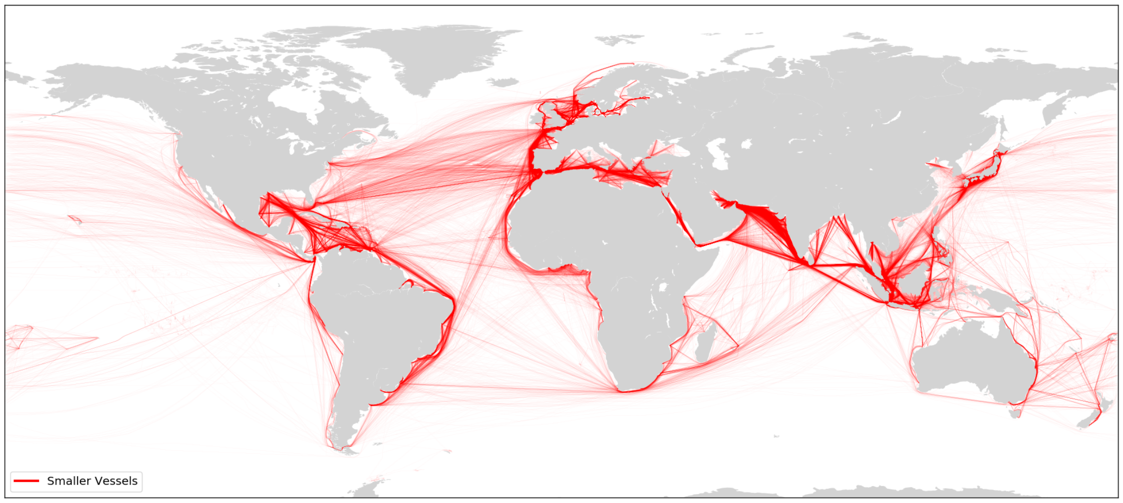


Figure B.2: Density line plot of smaller vessels, 2011-2017

Appendix C

Descriptive Statistics of Features

C.1 Descriptive Statistics of Count and Capacity Features

Table C.1: Descriptive statistics of count and capacity features created in Section 4.3.1

Feature name	**Unit**	**Mean**	**Std.**	**Min**	**Max**
count_Atl	# vessels	235.18	27.22	188	292
count_FE	# vessels	559.53	46.72	405	626
count_CP	# vessels	132.40	12.21	102	177
count_EstP	# vessels	38.18	13.62	19	89
count_NWE	# vessels	93.84	29.03	53	163
count_Ind	# vessels	76.99	13.89	47	114
count_Med	# vessels	123.39	17.35	74	158
count_GOM	# vessels	46.14	14.53	20	81
capacity_Atl	cbm	6,514,093	1,681,181	4,024,201	10,910,420
capacity_FE	cbm	10,807,360	1,124,184	8,484,979	13,375,450
capacity_CP	cbm	5,503,227	633,368	4,162,451	7,766,029
capacity_EstP	cbm	1,563,341	1,140,090	469,080	5,707,081
capacity_NWE	cbm	1,051,836	383,346	401,663	2,294,160
capacity_Ind	cbm	3,882,010	916,236	2,068,188	6,701,294
capacity_Med	cbm	2,340,653	405,816	1,450,451	3,674,603
capacity_GOM	cbm	1,780,195	870,157	514,107	4,039,428

Table C.2: Descriptive statistics of count and capacity percentage features created in Section 4.3.1

Feature name	Unit	Mean	Std.	Min	Max
count_Atl_percent	%	0.187	0.010	0.163	0.214
count_FE_percent	%	0.446	0.028	0.352	0.485
count_CP_percent	%	0.106	0.008	0.090	0.124
count_EstP_percent	%	0.030	0.008	0.018	0.064
count_NWE_percent	%	0.073	0.016	0.049	0.118
count_Ind_percent	%	0.061	0.008	0.040	0.082
count_Med_percent	%	0.098	0.007	0.076	0.117
count_GOM_percent	%	0.036	0.008	0.017	0.058
capacity_Atl_percent	%	0.203	0.021	0.157	0.276
capacity_FE_percent	%	0.346	0.031	0.256	0.404
capacity_CP_percent	%	0.176	0.019	0.133	0.230
capacity_EstP_percent	%	0.046	0.025	0.018	0.146
capacity_NWE_percent	%	0.032	0.007	0.014	0.056
capacity_Ind_percent	%	0.122	0.016	0.077	0.180
capacity_Med_percent	%	0.075	0.010	0.050	0.107
capacity_GOM_percent	%	0.054	0.019	0.019	0.099

C.2 Descriptive Statistics of Sailing Feature

Table C.3: Descriptive statistics of sailing feature from Section 4.3.2

Feature name	Unit	Mean	Std.	Min	Max
sailing_GOM	nm	8,176.43	350.15	7,206.97	8,713.28

C.3 Descriptive Statistics of Speed Features

Table C.4: Descriptive statistics of speed features from Section 4.3.4

Feature name	Unit	Mean	Std.	Min	Max
mean_global	knots	13.04	0.66	11.32	14.73
std_global	knots	4.17	0.36	3.37	5.24

C.4 Descriptive Statistics of Price and Market Features

Table C.5: Descriptive statistics of price and market features from Section 4.3.5

Feature name	**Unit**	**Mean**	**Std.**	**Min**	**Max**
spot_MB*	USD/mt	727.93	247.22	263.40	1,061.25
spot_NWE	USD/mt	694.21	243.79	278.75	1,076.40
spot_CP	USD/mt	616.07	250.67	223.33	1,162.00
spot_FEI	USD/mt	708.46	240.04	294.33	1,077.35
oil_WTI	USD/barrel	73.65	24.96	28.14	108.77

* Denoted as *price* in the text and code

Appendix D

Results of Augmented Dickey-Fuller Test

Table D.1: Results of ADF test with acceptance measure of H0 at 1, 5, or 10 %

Feature	ADF Statistic	p-value	1 %	5 %	10 %	Accept H0
count_GOM	-10.8849	1.2671e-19	-3.4497	-2.8701	-2.5713	False
mean_global	-7.1024	4.1354e-10	-3.4500	-2.8702	-2.5714	False
std_global	-8.4971	1.2666e-13	-3.4501	-2.8702	-2.5714	False
sailing_GOM	-9.9795	2.1324e-17	-3.4496	-2.8700	-2.5713	False
capacity_GOM	-9.1707	2.3920e-15	-3.4499	-2.8701	-2.5713	False
count_Atl	-10.7656	2.4567e-19	-3.4497	-2.8700	-2.5713	False
count_FE	-15.6903	1.4366e-28	-3.4495	-2.8700	-2.5712	False
count_CP	-11.7404	1.2777e-21	-3.4497	-2.8701	-2.5713	False
count_EstP	-9.5933	2.0083e-16	-3.4497	-2.8700	-2.5713	False
count_NWE	-13.3768	5.0685e-25	-3.4496	-2.8700	-2.5712	False
count_Ind	-12.6484	1.3857e-23	-3.4498	-2.8701	-2.5713	False
count_Med	-12.8395	5.6469e-24	-3.4496	-2.8700	-2.5713	False
capacity_Atl	-9.2690	1.3426e-15	-3.4499	-2.8701	-2.5713	False
capacity_FE	-8.1053	1.2674e-12	-3.4502	-2.8702	-2.5714	False
capacity_CP	-6.5264	1.0120e-08	-3.4500	-2.8702	-2.5714	False
capacity_EstP	-3.9528	1.6783e-03	-3.4503	-2.8703	-2.5714	False
capacity_NWE	-11.7037	1.5472e-21	-3.4497	-2.8701	-2.5713	False
capacity_Ind	-11.3528	9.9014e-21	-3.4499	-2.8701	-2.5713	False
capacity_Med	-12.8635	5.0509e-24	-3.4496	-2.8700	-2.5713	False
count_Atl_percent	-11.6334	2.2339e-21	-3.4496	-2.8700	-2.5713	False
count_FE_percent	-12.3253	6.6085e-23	-3.4497	-2.8700	-2.5713	False
count_CP_percent	-11.6896	1.6652e-21	-3.4497	-2.8701	-2.5713	False
count_EstP_percent	-12.8422	5.5765e-24	-3.4496	-2.8700	-2.5713	False
count_NWE_percent	-13.1708	1.2517e-24	-3.4496	-2.8700	-2.5712	False
count_Ind_percent	-12.5881	1.8467e-23	-3.4498	-2.8701	-2.5713	False
count_Med_percent	-13.0233	2.4286e-24	-3.4496	-2.8700	-2.5713	False
capacity_Atl_percent	-7.3560	9.7730e-11	-3.4503	-2.8703	-2.5714	False
capacity_FE_percent	-13.0317	2.3379e-24	-3.4497	-2.8700	-2.5713	False
capacity_CP_percent	-13.3169	6.5751e-25	-3.4497	-2.8700	-2.5713	False
capacity_EstP_percent	-8.7425	2.9816e-14	-3.4497	-2.8700	-2.5713	False
capacity_NWE_percent	-5.5536	1.5991e-06	-3.4505	-2.87042	-2.5715	False

Feature	**ADF Statistic**	**p-value**	**1 %**	**5 %**	**10 %**	**Accept H0**
capacity_Ind_percent	-11.2370	1.8464e-20	-3.4499	-2.8701	-2.5713	False
capacity_Med_percent	-12.9628	3.1986e-24	-3.4496	-2.8700	-2.5713	False
oil_WTI	-13.8890	6.0158e-26	-3.4495	-2.8699	-2.5712	False
spot_FEI	-6.1330	8.3130e-08	-3.4500	-2.8702	-2.5714	False
spot_CP	-5.8708	3.2434e-07	-3.4501	-2.8702	-2.5714	False
spot_NWE	-6.2918	3.5838e-08	-3.4500	-2.8702	-2.5714	False
count_GOM_percent	-10.8969	1.1858e-19	-3.4497	-2.8701	-2.5713	False
capacity_GOM_percent	-9.1849	2.2002e-15	-3.4499	-2.8701	-2.5713	False
spot_MB*	-13.3259	6.3237e-25	-3.4495	-2.8699	-2.5712	False

* Denoted as *price* in the text and code

Appendix E

Code

E.1 Master Script (MASTER.py)

This script runs the whole study and serves as a dashboard by selection of inputs and analysis-methods of interest. All files and functions except the prediction models and model evaluation are ran from this script.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov  1 13:41:00 2017

@author: PatrickAndreNaess
"""
import AIS_Analysis as AIS
import datetime
import pandas as pd
import Seaweb2AIS as SW
import Price_data as PD
import Ocean_mesh as OM
import Feature_importance as FI
import Subset_selection as SS
import Interpolate_data as ID

# AIS data:
Database = '/Users/PatrickAndreNaess/Desktop/LPG.db'
Database_Total = '/Volumes/LaCie/NTNUfilesMac/SAISGlobalFinal.db'
new_data = 'merged_cleaned.csv'

# LPG vessel data from Seaweb:
vessel_path = '/Users/PatrickAndreNaess/Documents/PYTHON/mmsi_capacity.xlsx'
vessel_data = pd.read_excel(vessel_path)

# All methods are run from this script, 1 = run

MessageType1 = 0 #1 = extract Message Type 1 signals from database 2011-2016
MessageType5 = 0 #1 = extract Message Type 5 signals from database 2011-2016
NewData = 0 # Extract external 2016-2018 AIS data from csv

# Table names:
n1 = 'LPG1' # Message Type 1
n5 = '' # Message Type 5

# Locate positions and assign draught-factor for MessageType5:
DraughtPositionFraction = 0

# Split on VLGC vs. Smaller vessels:
DataSplitOnVesselSize = 0

# Vessel search:
VesselSearch = 0

# Inpterpolate signals:
Interpolate = 0

# Mark each signal with a specific zone (polygon):
SelectZones = 0

# Map plotting:
```

```
LocalMap = 0 # Regular plot
HeatMap = 0 # Heat plot based on ship density
SpeedHeatMap = 0 # Heat plot based on ship speed
LineMap = 0 # Line plot
SplitLineMap = 0 # Line plot on vessel size (requiers DataSplitOnVesselSize)
DraughtMap = 0 # Heat plot based on draught−factor
PlotOceanMesh = 0 # Plot the ocean mesh

# Fleet analysis:
DraughtHistogram = 0 # Draught histogram
SpeedHistogram = 0 # Speed histogram

# Extract features and generated time series:
ImportPriceData = 0 # Price data
GenerateGlobalSpeed = 0 # Global speeds
GenerateGOMTimeSeries = 0 # Gulf of Mexico count, capacity + sailing
GeneratePolygonSeries = 0 # Polygon count, capacity and percentage

# Combine all the generated variables:
ConsolidateVariables = 0
# Features importance and subset selection ranking:
FeaturesSelection = 0
# Select the final features :
FinalSelection = 0

# Area of interest: [lonmax,latmax,lonmin,latmin]
Global = [180,90,−180,−90] # 1
Atlantic = [0,53,−80,33] # 2
GulfOfM = [−80,32,−99,17] # 3

# Choose location of intrest:
Loc = 1

if Loc == 1:
    Pos = Global
elif Loc == 2:
    Pos = Atlantic
elif Loc == 3:
    Pos = GulfOfM

# Time window of intrest:
lowtime = '01/01/2011'
hightime = '01/01/2018'

# Speed of interst:
maxspeed = 30
minspeed = 0

# Convert from date to unixtime:
unixlow = datetime.datetime.strptime(lowtime, "%d/%m/%Y").timestamp()
unixhigh = datetime.datetime.strptime(hightime, "%d/%m/%Y").timestamp()

# The analysis script is called to do all the analysis and plotting
# The following returns dataframes df1 and df5 with message type 1 and 5 data:
if MessageType1 == 1 or MessageType5 == 1:
    df1,df5 = AIS.ExtractData(Database,Pos[0],Pos[1],Pos[2],Pos[3],
                              unixlow,unixhigh,maxspeed,minspeed,
                              MessageType1,MessageType5,n1,n5)
if NewData == 1:
    df1 = AIS.AdditionalCSVData(new_data,df1,Pos[0],Pos[1],Pos[2],Pos[3],
                                unixlow,unixhigh,maxspeed,minspeed)

if DataSplitOnVesselSize == 1:
    df1_VLGC,df1_rest = AIS.DataSplitOnVesselSize(df1,vessel_data,50000)

if DraughtPositionFraction == 1:
    df5 = AIS.DraughtPositionFraction(df1,df5,vessel_data)

if VesselSearch == 1:
    SW.main(Database_Total)

if DraughtHistogram == 1:
    AIS.DraughtHistogram(df5)

```

```
if SpeedHistogram == 1:
    AIS.SpeedHistogram(df1)

if LocalMap == 1:
    AIS.LocalMap(df1)

if HeatMap == 1:
    AIS.HeatMap(df1)

if SpeedHeatMap == 1:
    AIS.SpeedHeatMap(df1)

if LineMap == 1:
    AIS.LineMap(df1)

if SplitLineMap == 1:
    AIS.DoubleLineMap(df1_rest,df1_VLGC,df1)

if DraughtMap == 1:
    AIS.DraughtMap(df5[df5['Timediff']<86400].dropna()) # exclude if diff larger than 24 hours

if PlotOceanMesh == 1:
    OM.PlotOceanMesh(1000,5000)

if Interpolate == 1:
    NEW = ID.main(df1)

if SelectZones == 1:
    df1_zones = AIS.zones(df1)

# Time series generation:

if ImportPriceData == 1:
    # Import price data
    spot_MB,spot_FEI,spot_NWE,spot_CP,oil = PD.main()
    w_oil= AIS.WeeklyPriceAverage(df1,oil,'oil_WTI')
    w_spot_MB = AIS.WeeklyPriceAverage(df1,spot_MB,'price')
    w_spot_FEI = AIS.WeeklyPriceAverage(df1,spot_FEI,'spot_FEI')
    w_spot_NWE = AIS.WeeklyPriceAverage(df1,spot_NWE,'spot_NWE')
    w_spot_CP = AIS.WeeklyPriceAverage(df1,spot_CP,'spot_CP')

if GenerateGlobalSpeed == 1:
    # Global speed:
    w_speed_global = AIS.WeekSpeedInArea(df1,Global,'global')
    w_speed_mean_global = w_speed_global['mean_global']
    w_speed_std_global = w_speed_global['std_global']

if GenerateGOMTimeSeries == 1:
    # Sailing in nm to MB
    w_sailing = AIS.WeekDistance2area(NEW,(-95,30),vessel_data,'GOM')
    # GOM count:
    w_count_GOM = AIS.WeekCountInArea(NEW,GulfOfM,'GOM')
    # GOM capacity:
    w_capacity_GOM = AIS.WeekCapacityInArea(NEW,GulfOfM,vessel_data,'GOM')

if GeneratePolygonSeries == 1:
    zones = ['Atl','FE','CP','EstP','NWE','Ind','Med']

    # Capacity and counting:
    counts = pd.DataFrame(index = AIS.GenerateWeeklyTimeWindow(df1)['timestamp'])
    capacities = pd.DataFrame(index = AIS.GenerateWeeklyTimeWindow(df1)['timestamp'])

    for zone in zones:
        counts = counts.join(AIS.WeekCountInArea(df1_zones[df1_zones['Zone'] == zone],
                                                  Global,zone))
        capacities = capacities.join(AIS.WeekCapacityInArea(df1_zones[df1_zones['Zone'] == zone],
                                                             Global,vessel_data,zone))
    # Capacity and counting percentage:
    percent_counts = counts.div(counts.sum(axis=1), axis=0)
    percent_capacities = capacities.div(capacities.sum(axis=1), axis=0)
    percent_counts.columns = [str(col) + '_percent' for col in percent_counts.columns]
    percent_capacities.columns = [str(col) + '_percent' for col in percent_capacities.columns]

if ConsolidateVariables == 1:
    # Join all features:
    data = w_count_GOM.join([w_speed_mean_global,w_speed_std_global,w_sailing,
                             w_capacity_GOM,counts,capacities,percent_counts,
                             percent_capacities])
    price_data = w_oil.join([w_spot_FEI,w_spot_CP,w_spot_NWE,w_spot_MB])
    # Important that price is last (right-most)

    data = data.join(price_data)

    data[18:-2].to_csv('timeseries_data.csv') # First and last
    price_data[18:-2].to_csv('nonAIS_features.csv') #[18:-2]

# Feature Selection:

if FeaturesSelection == 1:
    ts = pd.read_csv('timeseries_data.csv').set_index('timestamp')
    # Max lag of interest
    lag = 20
    # Features importance ranking
    ratings = FI.main(lag,ts)
    # Linear subset selection:
    #Select method: 'best', 'forward', 'backward'
    method = 'forward'
    models_best = SS.main(lag,method,ts)
    # Review k best features
    k = 50
    features = models_best.loc[k, "model"].params
    # Top features included
    print(pd.Series(features.index).apply(lambda x: x.split('(')[0]).value_counts())

if FinalSelection == 1:
    selected = ts[['capacity_NWE_percent','capacity_Atl_percent','capacity_EstP_percent',
                   'oil_WTI','spot_NWE','spot_FEI','spot_CP','price']]
    selected.to_csv('selected_AISfeatures.csv')

    price_selected = ts[['oil_WTI','spot_NWE','spot_FEI','spot_CP','price']]
    price_selected.to_csv('selected_nonAISfeatures.csv')
```

E.2 Data Extraction and Exploration (AIS_Analysis.py)

Extraction of AIS data from the databases, feature construction and time series generation is done in this script. In addition, several plotting and visualization functions are created in order to effectively preform exploratory data analysis.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov  1 13:45:53 2017

@author: PatrickAndreNaess
"""
import sqlite3
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import time
import datetime as dt
from mpl_toolkits.basemap import Basemap
import numpy as np
import pandas as pd
import seaborn as sns
import Ocean_mesh as OM

# THE FOLLOWING PART EXTRACT DATA FROM THE DATABASE:

def ExtractData(file,a,b,c,d,lowtime,hightime,maxspeed,minspeed,t1,t5,n1,n5):
    '''
    Extracts data from the an AIS database:
        file: Filepath to AIS data database
        a,b,c,d: Corners of the area of interest
        lowtime,hightime: Time window of interest
        maxspeed,minspeed: Vessel speed range of interest
        t1: 1 if MessageType1 will be extracted, 0 otherwise
        t5: 1 if MessageType5 will be extracted, 0 otherwise
        n1: Name of table with MessageType1 data in "file"
        n5: Name of table with MessageType5 data in "file"
    returns: DataFrames with Message Type 1 and 5 data
    '''
    speeds = list()
    plotlat = list()
    plotlon = list()
    timestep = list()
    mmsi = list()
    navstat = list()
    draughttime = list()
    draught = list()
    useridDraught = list()
    destination = list()
    name = list()

    # MessageType1:
    if t1 == 1:
        conn = sqlite3.connect(file)

        # For navigational status, insert portstatus into '':
        #portstatus = 'and (nav_status==1 or nav_status ==5)'

        SQLstring1 = "SELECT unixtime,sog,latitude,longitude,userid,\
            nav_status FROM %s WHERE longitude<= %s and latitude <= %s\
            and longitude >= %s and latitude >= %s and sog >= %s and \
            sog <= %s and unixtime >= %s and unixtime <= %s %s ORDER BY userid,\
            unixtime ASC" % (n1,str(a),str(b),str(c),str(d),
            str(minspeed),str(maxspeed),str(lowtime),str(hightime),'')

        # Extract data from database:
        A = time.time()
        with conn:
            cur = conn.cursor()

            cur.execute(SQLstring1)
            VesselData = cur.fetchall()

            for i in range(0,len(VesselData)):
                Datastrip = VesselData[i]
                timestep.append(Datastrip[0])
                speeds.append(Datastrip[1])
                plotlat.append(Datastrip[2])
                plotlon.append(Datastrip[3])
                mmsi.append(Datastrip[4])
                navstat.append(Datastrip[5])

        cur.close()
        print('MessageType1 database extraction time: %f s' % (time.time()-A))

        df = pd.DataFrame({'Speed': speeds, 'MMSI': mmsi, 'Unixtime': timestep,
                           'Lat': plotlat, 'Lon': plotlon})
    else:
        df = False

    # MessageType5:
    if t5 == 1:
        conn = sqlite3.connect(file)
        cur = conn.cursor()

        SQLstring5 = "SELECT unixtime,draught,userid,dest,name from %s where \
            unixtime >= %s and unixtime <= %s ORDER BY unixtime \
            ASC" % (n5,str(lowtime),str(hightime))

        # Extract draught data from database:
        A = time.time()
        with conn:
            cur = conn.cursor()

            cur.execute(SQLstring5)
            draughtdata = cur.fetchall()

            for i in range(0,len(draughtdata)):
                draughtstrip = draughtdata[i]
                if draughtstrip[1]/10 > 5:
                    draughttime.append(draughtstrip[0])
                    draught.append(draughtstrip[1]/10)
                    useridDraught.append(draughtstrip[2])
                    destination.append(draughtstrip[3])
                    name.append(draughtstrip[4])

        cur.close()
        print('MessageType5 database extraction time: %f s' % (time.time()-A))

        dfD = pd.DataFrame({'MMSI': useridDraught, 'Unixtime': draughttime,
                            'Draught': draught, 'Destination': destination,
                            'Name': name})

        # Some messages have error in destination (integer, not string)
        for i in range(0,len(dfD['Destination'])):
            if type(dfD.at[i,'Destination']) == str:
                    dfD.at[i,'Destination'] = dfD.at[i,'Destination'].strip()

        dfD['Name'] = dfD['Name'].apply(lambda x: x.strip())

    else:
        dfD = False

    return df,dfD

###############################################################################

def AdditionalCSVData(file,df_1,a,b,c,d,lowtime,hightime,maxspeed,minspeed):
    '''
    Extracts additional AIS-data from csv:
        file: Filepath to AIS data database
        df_1: Excisting DataFrame with MessageType1 data
        a,b,c,d: Corners of the area of interest
        lowtime,hightime: Time window of interest
        maxspeed,minspeed: Vessel speed range of interest
    returns: Resulting DataFrame with Message Type 1 data
    '''
    A = time.time()
    # Import
    new = pd.read_csv(file).drop('Unnamed: 0',axis=1)
    # Time-constraints
    new = new[(new['Unixtime'] <= hightime) & (new['Unixtime'] >= lowtime)]
    # Speed-constraints
    new = new[(new['Speed'] <= maxspeed) & (new['Speed'] >= minspeed)]
    # Geo-contraints
    new = new[(new['Lon'] <= a) & (new['Lon'] >= c) &
                (new['Lat'] <= b) & (new['Lat'] >= d)]

    print('Additional database extraction time: %f s' % (time.time()-A))
    return df_1.append(new, ignore_index=True)

################################################################################

def DraughtPositionFraction(df_1,df_5,v_data):
    '''
    Connects the load factor and positions to draught data:
        df_1: DataFrame with MessageType1 data
        df_5: DataFrame with MessageType5 data
        v_data: DataFrame with vessel dimensions and characteristics
    returns: Resulting DataFrame with Message Type 5 data
    '''
    positions = df_1.drop('Speed',axis=1).set_index('MMSI')
    design_draught = v_data.set_index('MMSI')['Draught']

    for index, vessel_df5 in df_5.iterrows():
        MMSI = vessel_df5['MMSI']
        if MMSI in positions.index:
            # Vessel positions
            vp = positions.xs(MMSI)
            vp['dif'] = abs(vp['Unixtime']-vessel_df5['Unixtime'])
            line = vp[vp['dif']==vp['dif'].min()].iat[0]

            df_5.at[index,'Timediff'] = line['dif']

            df_5.at[index,'Lat'] = line['Lat']
            df_5.at[index,'Lon'] = line['Lon']

            load = vessel_df5['Draught']/design_draught.at[MMSI]

            if load > 1:
                df_5.at[index,'D/Dmax'] = 1
            else:
                df_5.at[index,'D/Dmax'] = load
    return df_5

################################################################################

def DataSplitOnVesselSize(df_1,vessels,size):
    '''
    Splits the data on vessel size:
        df_1: DataFrame with MessageType1 data
        vessels: DataFrame with vessel characteristics
        size: Capacity size on where to split the dataset
    returns: 2 DataFrames larger and smaller vessels, respectively
    '''
    capacity = vessels.set_index('MMSI')['Gas_Capacity']
    df = df_1.set_index('MMSI')
    df = df.join(capacity).reset_index()

    df_VLGC = df[df['Gas_Capacity'] > size].drop('Gas_Capacity',axis=1)
    df_rest = df[df['Gas_Capacity'] <= size].drop('Gas_Capacity',axis=1)

    return df_VLGC,df_rest

################################################################################

def DraughtHistogram(df_5):
    '''
    Plots a histogram of the draught distribution:
        df_5: DataFrame with MessageType5 data
    '''
    plt.figure()
    sns.distplot(df_5['Draught'],kde=False,norm_hist=True,color='r')
    plt.xlabel('Draught [meters]')
    plt.ylabel('Fraction of time')
    plt.show()

###############################################################################

def SpeedHistogram(df_1):
    '''
    Plots a histogram of the speed distribution:
        df_1: DataFrame with MessageType1 data
    '''
    plt.figure()
    sns.distplot(df_1['Speed'],kde=False,norm_hist=True,color='b',hist_kws=dict(alpha=0.9))
    plt.xlabel('Speed [knots]')
    plt.ylabel('Fraction of time')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Speed_Hist_Current.pdf',
                bbox_inches='tight')
    plt.show()

###############################################################################

def LocalMap(df_1):
    '''
    Plots scatter positions on a map:
        df_1: DataFrame with MessageType1 data
    '''
    minlon = max(-180,min(df_1['Lon'])-5) #-5
    minlat = max(-90,min(df_1['Lat'])-5) #-5
    maxlon = min(180,max(df_1['Lon'])+5) #+5
    maxlat = min(90,max(df_1['Lat'])+5) #+5
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    fig,ax=plt.subplots(figsize=(25,25))
    #fig.add_axes([0.1,0.1,0.8,0.8])
    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,
                urcrnrlat=maxlat,rsphere=(6378137.00,6356752.3142),
                resolution='l',projection='cyl',lat_0=lat0,lon_0=lon0,
                lat_ts = lat1)

    #Black n White
    #m.drawmapboundary(fill_color='w')
    #m.fillcontinents(color='lightgrey',lake_color='w') #,zorder=0
    #c='k'

    #White n Blue
    #m.drawmapboundary(fill_color='#003253')
    #m.fillcontinents(color='k',lake_color='#003253')
    #c='w'

    #Master
    m.drawmapboundary(fill_color='w')
    m.fillcontinents(color='lightgrey',lake_color='w')
    #c='b'

    x, y = m(df_1['Lon'],df_1['Lat'])
    #Ships:
    m.scatter(x,y,0.01,marker='.',c='b')#

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Local_Map_Current.png',
                bbox_inches='tight',dpi = 100)

###############################################################################

def HeatMap(df_1):
    '''
    Plots a hexbin heatmap of vessel positions:
        df_1: DataFrame with MessageType1 data
    '''
    minlon = max(-180,min(df_1['Lon'])-5) #-5
    minlat = max(-90,min(df_1['Lat'])-5) #-5
    maxlon = min(180,max(df_1['Lon'])+5) #+5
    maxlat = min(90,max(df_1['Lat'])+5) #+5
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    fig,ax=plt.subplots(figsize=(25,25))

    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,
                urcrnrlat=maxlat,rsphere=(6378137.00,6356752.3142),
                resolution='l',projection='cyl',lat_0=lat0,lon_0=lon0,
                lat_ts = lat1)

    m.drawmapboundary(fill_color='black')
    m.fillcontinents(color='darkgrey',lake_color='black',zorder=0) #,zorder=0

    x, y = m(df_1['Lon'],df_1['Lat'])

    m.hexbin(np.array(x),np.array(y),gridsize=500,mincnt=1,cmap='inferno',
             bins='log',zorder=0)
    m.colorbar(location='bottom', format='%.1f', label='log(# messages)')
    # in hexbin: ,C=np.array(speeds)
    #mincnt = 2: cells with one signal gest neglected
    #magma or inferno with balck background

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Heat_Plot_Current.png',
        bbox_inches='tight',dpi = 100)

##############################################################################

def SpeedHeatMap(df_1):
    '''
    Plots a hexbin heatmap of the vessel speeds:
        df_1: Dataframe with MessageType1 data
    '''
    minlon = max(-180,min(df_1['Lon'])-5) #-5
    minlat = max(-90,min(df_1['Lat'])-5) #-5
    maxlon = min(180,max(df_1['Lon'])+5) #+5
    maxlat = min(90,max(df_1['Lat'])+5) #+5
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    fig,ax=plt.subplots(figsize=(25,25))

    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,
                urcrnrlat=maxlat,rsphere=(6378137.00,6356752.3142),
                resolution='l',projection='cyl',lat_0=lat0,lon_0=lon0,
                lat_ts = lat1)

    m.drawmapboundary(fill_color='w')
    m.fillcontinents(color='darkgrey',lake_color='w') #,zorder=0

    x, y = m(df_1['Lon'],df_1['Lat'])

    m.hexbin(np.array(x),np.array(y),C=np.array(df_1['Speed']),gridsize=75,
             mincnt=250,cmap='coolwarm',zorder=0)
    m.colorbar(location='bottom', format='%.1f', label='Speed [knots]')
    #mincnt = 2: cells with one signal gest neglected
    #magma or inferno with balck background

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Speed_Heat_Current.png',
            bbox_inches='tight',dpi = 100)

##############################################################################

def LineMap(df_1):
    '''
    Density line plot, connecting the vessel positions, excluding large gaps:
        df_1: DataFrame with MessageType1 data
    '''
    minlon = max(-180,min(df_1['Lon'])-5) #-5
    minlat = max(-90,min(df_1['Lat'])-5) #-5
    maxlon = min(180,max(df_1['Lon'])+5) #+5
    maxlat = min(90,max(df_1['Lat'])+5) #+5
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    dfLine = df_1.set_index(['MMSI','Unixtime'])

    fig,ax=plt.subplots(figsize=(25,25))

    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,
                urcrnrlat=maxlat,rsphere=(6378137.00,6356752.3142),
                resolution='l',projection='cyl',lat_0=lat0,lon_0=lon0,
                lat_ts = lat1)
    #Linkedin
    #m.drawmapboundary(fill_color='#F3F3F3')
    #m.fillcontinents(color='#243642',lake_color='#F3F3F3') #,zorder=0,
    #c='#0068A7'

    #Blue-white
    m.drawmapboundary(fill_color='w')
    m.fillcontinents(color='lightgrey',lake_color='w') #,zorder=0,
    #c='b'

    #Western Bulk
    #m.drawmapboundary(fill_color='#003253')
    #m.fillcontinents(color='k',lake_color='#003253')
    #c='w'

    #Neon:
    #m.drawmapboundary(fill_color='#323232')
    #m.fillcontinents(color='#black',lake_color='#black') #,zorder=0,
    #c='palegreen'

    # Plot for each vessel:
    for MMSI, new_df in dfLine.groupby(level=0):
        new_df = new_df.xs(MMSI)[['Lat','Lon']]
        diff_df = new_df[['Lat','Lon']].diff().abs()
        # Exclude line where difference in lat/lon > 30
        new_df[(diff_df['Lat']>10) | (diff_df['Lon']>10)] = np.nan

        x, y = m(new_df['Lon'],new_df['Lat'])
        m.plot(x,y,linewidth=0.05,c='b',alpha = 0.2,zorder=0)

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Line_Plot_Current.png',
                bbox_inches='tight',dpi = 100)

###############################################################################

def DoubleLineMap(df_1_small,df_1_large,df_1):
    '''
    Density line plot for several vessels DataFrames, excluding large gaps:
        df_1_small: DataFrame for smaller vessels with MessageType1 data
        df_1_large: DataFrame for VLGC with MessageType1 data
        df_1: DataFrame with MessageType1 data
    '''
    minlon = max(-180,min(df_1['Lon'])-5) #-5
    minlat = max(-90,min(df_1['Lat'])-5) #-5
    maxlon = min(180,max(df_1['Lon'])+5) #+5
    maxlat = min(90,max(df_1['Lat'])+5) #+5
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    dfLineSmall = df_1_small.set_index(['MMSI','Unixtime'])
    dfLineLarge = df_1_large.set_index(['MMSI','Unixtime'])

    fig,ax=plt.subplots(figsize=(25,25))

    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,
                urcrnrlat=maxlat,rsphere=(6378137.00,6356752.3142),
                    resolution='l',projection='cyl',lat_0=lat0,lon_0=lon0,
                    lat_ts = lat1)

    m.drawmapboundary(fill_color='w')
    m.fillcontinents(color='lightgrey',lake_color='w') #,zorder=0,

    # Plot for each vessel:
    for MMSI, new_df in dfLineSmall.groupby(level=0):
        new_df = new_df.xs(MMSI)[['Lat','Lon']]
        diff_df = new_df[['Lat','Lon']].diff().abs()
        # Exclude line where difference in lat/lon > 30
        new_df[(diff_df['Lat']>10) | (diff_df['Lon']>10)] = np.nan

        x, y = m(new_df['Lon'],new_df['Lat'])
        m.plot(x,y,linewidth=0.05,c='r',alpha = 0.2,zorder=0)

    for MMSI, new_df in dfLineLarge.groupby(level=0):
        new_df = new_df.xs(MMSI)[['Lat','Lon']]
        diff_df = new_df[['Lat','Lon']].diff().abs()
        # Exclude line where difference in lat/lon > 30
        new_df[(diff_df['Lat']>10) | (diff_df['Lon']>10)] = np.nan

        x, y = m(new_df['Lon'],new_df['Lat'])
        m.plot(x,y,linewidth=0.05,c='b',alpha = 0.2,zorder=0)

    legend_elements = [Line2D([0],[0],color='r',lw=3,label='Smaller Vessels'),
                       Line2D([0],[0],color='b',lw=3,label='VLGC')]
    plt.legend(handles=legend_elements, loc=3, fontsize = 'x-large')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Line_Vessel_split.png',
                bbox_inches='tight',dpi = 100)

################################################################################

def DraughtMap(df_5):
    '''
    Plots a hexbin heatmap of the vessel draught rate:
        df_5: DataFrame with MesaageType5 data
    '''
    minlon = max(-180,min(df_5['Lon'])-5) #-5
    minlat = max(-90,min(df_5['Lat'])-5) #-5
    maxlon = min(180,max(df_5['Lon'])+5) #+5
    maxlat = min(90,max(df_5['Lat'])+5) #+5
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    fig,ax=plt.subplots(figsize=(25,25))

    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,
                urcrnrlat=maxlat,rsphere=(6378137.00,6356752.3142),
                resolution='l',projection='cyl',lat_0=lat0,lon_0=lon0,
                lat_ts = lat1)

    m.drawmapboundary(fill_color='w')
    m.fillcontinents(color='darkgrey',lake_color='w') #,zorder=0

    x, y = m(df_5['Lon'],df_5['Lat'])

    m.hexbin(np.array(x),np.array(y),C=np.array(df_5['D/Dmax']),gridsize=75,
             mincnt=1,cmap='coolwarm',zorder=0)
    m.colorbar(location='bottom', format='%.1f',label='Draught/Max Draught from Seaweb [-]')
    #mincnt = 2: cells with one signal gest neglected
    #magma or inferno with balck background

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Draught_Map_Current.png',
            bbox_inches='tight',dpi = 100)

################################################################################

def GenerateWeeklyTimeWindow(df):
    '''
    Generates weekly time windows based on first and last message in df:
        df: DataFrame with MessageType1 data
    returns: DataFrame with weekly time intervals
    '''
    mintime = df['Unixtime'].min()
    maxtime = df['Unixtime'].max()

    min_week = dt.datetime.fromtimestamp(mintime).isocalendar()[1]
    min_year = dt.datetime.fromtimestamp(mintime).isocalendar()[0]
    start = dt.datetime.strptime(str(min_year)+'-W'+str(min_week)+'-1',"%Y-W%W-%w")

    max_week = dt.datetime.fromtimestamp(maxtime).isocalendar()[1]
    max_year = dt.datetime.fromtimestamp(maxtime).isocalendar()[0]
    end = dt.datetime.strptime(str(max_year)+'-W'+str(max_week)+'-0',"%Y-W%W-%w")

    window = pd.DataFrame(columns=['timestamp'])
    delta = dt.timedelta(days=7)

    tid = start

    while (tid <= end):
        window = window.append({'timestamp': tid},ignore_index=True)
        tid = tid + delta

    return window

##############################################################################

def WeeklyPriceAverage(df,daily,name):
    '''
    Calculate weekly price average from daily price data:
        df: DataFrame with MessageType1 data
        daily: Daily price DataFrame
        name: Wanted feature name (Ex: oil_WTI, spot_MB)
    returns: Weekly price averages based on time intervals
    '''
    price = GenerateWeeklyTimeWindow(df)

    for i in range(0,len(price)-1):
        start_stamp = price.at[i,'timestamp']
        end_stamp = price.at[i+1,'timestamp']

        weekly = daily[(daily.index>=start_stamp) & (daily.index<end_stamp)]['price']
        average = weekly.mean()

        price.at[i,name] = average

    return price.set_index('timestamp')

##############################################################################

def WeekCountInArea(df,area,name):
    '''
    The function counts unique vessels present in an area each week:
        df: DataFrame with MesaageType1 data
        area: List with corners (global = [180,90,-180,-90]) on where to count
        name: Wanted feature name (Ex: Atl, GOM)
    returns: Weekly count in area
    '''
    count = GenerateWeeklyTimeWindow(df)

    df = df[(df['Lat']<area[1]) & (df['Lat']>area[3])]
    df = df[(df['Lon']>area[2]) & (df['Lon']<area[0])]

    for i in range(0,len(count)-1):
        start_stamp = count.at[i,'timestamp']
        end_stamp = count.at[i+1,'timestamp']

        start_u = time.mktime(start_stamp.timetuple())
        end_u = time.mktime(end_stamp.timetuple())

        N_unique = df[(df['Unixtime']>=start_u) & (df['Unixtime']<end_u)]['MMSI'].nunique()

        count.at[i,'count_'+name] = N_unique

    return count.set_index('timestamp')

```

```python
###############################################################################

def WeekCapacityInArea(df,area,vessels,name):
    '''
    The function sums the capacity of vessels present in an area each week:
        df: DataFrame with MesaageType1 data
        area: List with corners (global = [180,90,-180,-90]) on where to count
        vessels: DataFrame with vessel characteristics
        name: Wanted feature name (Ex: Atl, GOM)
    returns: Weekly summed capacity in area
    '''
    cap = GenerateWeeklyTimeWindow(df)

    capacity = vessels.set_index('MMSI')['Gas_Capacity']

    df = df[(df['Lat']<area[1]) & (df['Lat']>area[3])]
    df = df[(df['Lon']>area[2]) & (df['Lon']<area[0])]

    for i in range(0,len(cap)-1):
        start_stamp = cap.at[i,'timestamp']
        end_stamp = cap.at[i+1,'timestamp']

        start_u = time.mktime(start_stamp.timetuple())
        end_u = time.mktime(end_stamp.timetuple())

        N_unique = df[(df['Unixtime']>=start_u) & (df['Unixtime']<end_u)]['MMSI'].nunique()
        unique = df[(df['Unixtime']>=start_u) & (df['Unixtime']<end_u)]['MMSI'].unique()

        weekly_cap = 0

        if N_unique > 0:
            for vessel in unique:
                weekly_cap += capacity.at[vessel]

        cap.at[i,'capacity_'+name] = weekly_cap

    return cap.set_index('timestamp')

###############################################################################

def WeekSpeedInArea(df,area,name):
    '''
    The function calculates mean and std of the flet speed in an area each week:
        df: DataFrame with MesaageType1 data
        area: List with corners (global = [180,90,-180,-90]) on where to count
        name: Wanted feature name (Ex: Atl, GOM)
    returns: Weekly mean speed and std of speed in area
    '''
    speed = GenerateWeeklyTimeWindow(df)

    df = df[(df['Lat']<area[1]) & (df['Lat']>area[3])]
    df = df[(df['Lon']>area[2]) & (df['Lon']<area[0])]

    for i in range(0,len(speed)-1):
        start_stamp = speed.at[i,'timestamp']
        end_stamp = speed.at[i+1,'timestamp']

        start_u = time.mktime(start_stamp.timetuple())
        end_u = time.mktime(end_stamp.timetuple())

        w_speed = df[(df['Unixtime']>=start_u) & (df['Unixtime']<end_u)]['Speed']

        speed.at[i,'mean_'+name] = w_speed.mean()
        speed.at[i,'std_'+name] = w_speed.std()

    return speed.set_index('timestamp')

###############################################################################

def generate_polygons():
    '''
    Generate the ocean plygons:
    returns: List with polygons
    '''
    polygons = list()
    polygon_atlantic = [[25,-90],[25,0],[-5.5,36],[-10,90],[-100,90],
                         [-103,22],[-76,7],[-61,-19],[-80,-90],[25,-90]]
    polygon_FE = [[180,90],[80,90],[50,50],[80,40],[77,11],[180,-90],[180,90]]
    polygon_CP = [[33,29],[50,50],[80,40],[77,11],[25,0],[33,29]]
    polygon_EstP = [[-180,90],[-100,90],[-103,22],[-76,7],[-61,-19],[-80,-90],
                    [-180,-90]]
    polygon_NWE = [[-5.5,36],[-10,90],[80,90],[50,50],[4,48],[-5.5,36]]

    polygon_indi = [[25,-90],[25,0],[77,11],[180,-90],[25,-90]]

    polygon_Med = [[25,0],[-5.5,36],[4,48],[50,50],[33,29],[25,0]]

    polygons.append(polygon_atlantic)
    polygons.append(polygon_FE)
    polygons.append(polygon_CP)
    polygons.append(polygon_EstP)
    polygons.append(polygon_NWE)
    polygons.append(polygon_indi)
    polygons.append(polygon_Med)

    return polygons

###############################################################################

def ocean_polygons(polygons = generate_polygons()):
    '''
    Plots the ocean polygons:
        polygons: List with polygons
    '''
    fig, ax = plt.subplots(figsize=(10,10))
    m = Basemap(projection='cyl',lon_0=0,resolution='l')
    m.drawmapboundary(fill_color='white')
    m.fillcontinents(color='lightgrey',lake_color='white')

    for polygon in polygons:
        x,y = zip(*polygon)
        m.plot(x,y,markersize=0)
        ax.fill(x, y,alpha=0.2)
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/ocean_polygons.png',
        bbox_inches='tight',dpi = 100)

###############################################################################

def point_inside_polygon(x,y,poly):
    '''
    Ray casting algorithm to check if a point is inside a polygon:
        x: Longitude
        y: Latitude
        poly: Polygon to check for
    returns: True/False if inside/not inside the polygon
    '''
    n = len(poly)
    inside =False

    p1x,p1y = poly[0]
    for i in range(n+1):
        p2x,p2y = poly[i % n]
        if y > min(p1y,p2y):
            if y <= max(p1y,p2y):
                if x <= max(p1x,p2x):
                    if p1y != p2y:
                        xinters = (y-p1y)*(p2x-p1x)/(p2y-p1y)+p1x
                    if p1x == p2x or x <= xinters:
                        inside = not inside
        p1x,p1y = p2x,p2y

    return inside

###############################################################################

def zones(df_1):
    '''
    States in which polygon the signal is sent from:
        df_1: DataFrame with MessageType1 data
    returns: DataFrame with MessageType1 data, with additional column stating Zone
    '''
    df = df_1.copy()
    polygons = generate_polygons()

    #Creating empty column in the dataframe
    df['Zone'] = 'Outside'

    plotlon = list(df['Lon'])
    plotlat = list(df['Lat'])

    #Taking time:
    for i in range(0,len(plotlat)):
        if point_inside_polygon(plotlon[i],plotlat[i],polygons[0]):
            df.at[i,'Zone'] = 'Atl'
        elif point_inside_polygon(plotlon[i],plotlat[i],polygons[1]):
            df.at[i,'Zone'] = 'FE'
        elif point_inside_polygon(plotlon[i],plotlat[i],polygons[2]):
            df.at[i,'Zone'] = 'CP'
        elif point_inside_polygon(plotlon[i],plotlat[i],polygons[3]):
            df.at[i,'Zone'] = 'EstP'
        elif point_inside_polygon(plotlon[i],plotlat[i],polygons[4]):
            df.at[i,'Zone'] = 'NWE'
        elif point_inside_polygon(plotlon[i],plotlat[i],polygons[5]):
            df.at[i,'Zone'] = 'Ind'
        elif point_inside_polygon(plotlon[i],plotlat[i],polygons[6]):
            df.at[i,'Zone'] = 'Med'
    return df

################################################################################

def inside_polygon(df_1,polygon):
    '''
    Another function to check if a signal is inside a polygon:
        df_1: DataFrame with MessageType1 data
        polygon: Polygon
    returns: DataFrame with MessageType1 data, with additional column stating 1/0
    '''
    df = df_1.copy()

    # Narrow down the search to make faster iterations:
    x,y = zip(*polygon)
    max_lon,max_lat,min_lon,min_lat = max(x),max(y),min(x),min(y)

    df_search = df[(df['Lon'] < max_lon) & (df['Lon'] > min_lon) &
                   (df['Lat'] < max_lat) & (df['Lat'] > min_lat)]

    lon = list(df_search['Lon'])
    lat = list(df_search['Lat'])
    index = list(df_search.index)

    df['in_area'] = 0
    j = 100
    for i in range(0,len(lon)):
        if point_inside_polygon(lon[i],lat[i],polygon):
            df.at[index[i],'in_area'] = 1
        if i > j:
            print(i)
            j = j*2
    return df

################################################################################

def WeekCountFlux(df_1,name):
    '''
    Using inside_polygon to find the vessel flux over a polygon boundary:
        df_1: DataFrame with MessageType1 data
        name: Name of polygon
    returns: The weekly capacity flux represented by inflow and outfolw from the polygon
    '''
    df_poly = df_1.copy()
    flux = GenerateWeeklyTimeWindow(df_poly)

    df_poly['diff'] = df_poly.groupby('MMSI')['in_area'].transform(pd.Series.diff)

    for i in range(0,len(flux)-1):
        start_stamp = flux.at[i,'timestamp']
        end_stamp = flux.at[i+1,'timestamp']

        start_u = time.mktime(start_stamp.timetuple())
        end_u = time.mktime(end_stamp.timetuple())

        weekly = df_poly[(df_poly['Unixtime']>=start_u) & (df_poly['Unixtime']<end_u)]

        flux.at[i,'in_count_'+name] = len(weekly[weekly['diff']==1])
        flux.at[i,'out_count_'+name] = len(weekly[weekly['diff']==-1])

    return flux.set_index('timestamp')

###############################################################################

def WeekCapacityFlux(df_1,vessels,name):
    '''
    Using inside_polygon to find the vessel capacity flux over a polygon boundary:
        df_1: DataFrame with MessageType1 data
        name: Name of polygon
        vessels: DataFrame with vessel characteristics
    returns: The weekly count flux represented by inflow and outfolw from the polygon
    '''
    df_poly = df_1.copy()
    flux = GenerateWeeklyTimeWindow(df_poly)

    capacity = vessels.set_index('MMSI')['Gas_Capacity']

    df_poly['diff'] = df_poly.groupby('MMSI')['in_area'].transform(pd.Series.diff)

    for i in range(0,len(flux)-1):
        start_stamp = flux.at[i,'timestamp']
        end_stamp = flux.at[i+1,'timestamp']

        start_u = time.mktime(start_stamp.timetuple())
        end_u = time.mktime(end_stamp.timetuple())

        weekly = df_poly[(df_poly['Unixtime']>=start_u) & (df_poly['Unixtime']<end_u)]

        weekly_in = weekly[weekly['diff']==1]
        weekly_out = weekly[weekly['diff']==-1]

        N_unique_in = weekly_in['MMSI'].nunique()
        N_unique_out = weekly_out['MMSI'].nunique()
        unique_in = weekly_in['MMSI'].unique()
        unique_out = weekly_out['MMSI'].unique()

        weekly_cap_in = 0

        if N_unique_in > 0:
            for vessel in unique_in:
                weekly_cap_in += capacity.at[vessel]

        weekly_cap_out = 0

        if N_unique_out > 0:
            for vessel in unique_out:
                weekly_cap_out += capacity.at[vessel]

        flux.at[i,'in_capacity_'+name] = weekly_cap_in
        flux.at[i,'out_capacity_'+name] = weekly_cap_out

    return flux.set_index('timestamp')

###############################################################################

def WeekDistance2area(df_1,coordinate,vessels,name):
    '''
    Calculating weekly weighted arithmetic mean sailing distance:
        df_1: Dataframe with MesaageType1 data
        coordiante: The point (lon0,lat0) to calculate distanse to
        vessels: Dataframe with vessel characteristics
        name: Name target area
    returns: Weekly weighted arithmetic mean sailing distance
    '''
    distance = GenerateWeeklyTimeWindow(df_1)

    capacity = vessels.set_index('MMSI')['Gas_Capacity']

    df = df_1.drop('Speed',axis=1)

    # Generate the ocean mesh and get the mesh node positions:
    G,points = OM.GenerateOceanMesh()
    nodes = list(zip(points['lon'],points['lat']))

    for i in range(0,len(distance)-1):
        start_stamp = distance.at[i,'timestamp']
        end_stamp = distance.at[i+1,'timestamp']

        start_u = time.mktime(start_stamp.timetuple())
        end_u = time.mktime(end_stamp.timetuple())

        df_week = df[(df['Unixtime']>=start_u) & (df['Unixtime']<end_u)].set_index('MMSI')

        # Sum(C_i * D_i)
        dist_cap = 0
        # Sum(C_i)
        cap_week = 0
        for vessel in df_week.index.unique():
            # Add vessel capacity to monthly world capacity
            cap_week += capacity.at[vessel]

            df_vessel = df_week.xs(vessel)

            # Select last recorded position this month:
            # If only one signal:
            if 'Lat' in df_vessel.index:
                # If only one record
                lat = df_vessel['Lat']
                lon = df_vessel['Lon']
            else:
                # If several signals, select the last
                lat = df_vessel['Lat'].iat[-1]
                lon = df_vessel['Lon'].iat[-1]

            # Sailing distance in nm from vessel position to area
            vessel_dist = OM.ShortestSeaDistance((lon,lat),coordinate,nodes,G)
            # C_i * D_i
            dist_cap += vessel_dist*capacity.at[vessel]

        # Sum(C_i * D_i)/Sum(C_i)
        arithm = dist_cap/cap_week
        distance.at[i,'sailing_'+name] = arithm

    return distance.set_index('timestamp')
```

E.3 Vessel Search and Fleet Characteristics Extraction (Seaweb2AIS.py)

The following code runs the vessels search for LPG vessels in global AIS data. In addition, plotting of fleet characteristics is included.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 28 15:28:34 2018

@author: PatrickAndreNaess
"""
import sqlite3
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def plotSeaweb(df):
    '''
    Plots statistics:
        df: DataFrame with vessel data
    '''
    #Pairplot of the dataset:
    sns.pairplot(df.drop(['MMSI','IMO/LR/IHS_No.'],axis=1).dropna(),
                 size=2.5,plot_kws={"s": 10})

    #Plot of gas capacity vs. DWT by type
    sns.lmplot(x='Gas_Capacity',y='Deadweight',data = df,hue='Ship_Type',
               fit_reg=False,scatter_kws={"s": 20},legend_out=False,size=7)

    #Density lot of gas capacity vs. DWT
    sns.jointplot(x='Gas_Capacity',y='Deadweight',data=df,kind='kde',size=7)

    # Newwuildings
    plt.figure(figsize=(13,3))
    df['Built'].value_counts().sort_index()[35:52].plot()
    df['Built'].value_counts().sort_index()[51:].plot()
    plt.xlabel('Year')
    plt.ylabel('Newbuildings')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Newbuilding.pdf',
            bbox_inches='tight')

    #Vessels with missing MMSI number
    nan_mmsi = df[df['MMSI'].isnull()]
    num_nan_mmsi = len(nan_mmsi)

    #Size plot of capacity
    fig,ax=plt.subplots(figsize=(5,3))
    sns.distplot(df['Gas_Capacity'],kde=False,color='r')
    plt.ylabel('# vessels')
    plt.xlabel('Gas capacity [cbm]')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Gas_Capacity.png',
            bbox_inches='tight',dpi = 200)

    #Size plot of length
    fig,ax=plt.subplots(figsize=(5,3))
    sns.distplot(df['Length'],kde=False,color='r')
    plt.ylabel('# vessels')
    plt.xlabel('Length [m]')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/LPG_Length.png',
            bbox_inches='tight',dpi = 200)

    #Flag plot:
    df['Flag'].value_counts().plot(kind='pie')

    print('\nFive largest opeartors:')
    print(df['Operator'].value_counts().head(5))
    print('\nFive most registered flags:')
    print(df['Flag'].value_counts().head(5))
    print('\nVessels without MMSI: %i' % num_nan_mmsi)
    print('Vessels without listet DWT: %i' % len(df[df['Deadweight'] == 0]))
    print('Vessels without listet Gas_Capacity: %i' % len(df[df['Gas_Capacity'] == 0]))
    print('\nThese are:\n')
    print(df[df['Gas_Capacity'] == 0])

def imo2mmsi(databasepath,imo_numbers):
    '''
    Vessel search in database:
        databasepath: Path to database
        imo_numbers: List with imo numbers from SeaWeb
    returns: DataFrame with MMSIs numbers found for each IMO number
    '''
    con = sqlite3.connect(databasepath)
    imo = list()
    mmsi = list()

    imo_string = str(imo_numbers[0])
    imo_numbers.remove(imo_numbers[0])

    for number in imo_numbers:
        imo_string = imo_string + ',%s' % str(number)

    #In case erroneous, select tanker also
    SQL = "SELECT imo,userid,ship_type FROM MessageType5 where imo in (%s) \
            and ship_type >= 80 and ship_type < 90" % imo_string

    with con:
        cur = con.execute(SQL)
        VesselData = cur.fetchall()

        for i in range(0,len(VesselData)):
            Datastrip = VesselData[i]
            imo.append(Datastrip[0])
            mmsi.append(Datastrip[1])

    con.close()

    x = pd.DataFrame({'MMSI': mmsi,'imo': imo})
    x.set_index('imo',inplace = True)

    # Same ship might not be dropped as duplicate since the ship_type might be changed
    return x.drop_duplicates('MMSI')

# Div statistics:
def imo2mmsiStatistics(lpg,imoTommsi):
    '''
    Plots vessel search statistics:
        lpg: Vessel data from SeaWeb
        imoTommsi: All MMSIs found per IMO number
    '''
    no_found = len(lpg[lpg['Num MMSI'].isnull()])
    found = len(lpg)
    print('\nDid not find MMSI relating to IMO in MessageType5: %i in %i vessels'
          % (no_found,found))
    print('Found: %.4f' % (1-no_found/found))
    print('Unique MMSI: %i\n' % len(imoTommsi))

    plt.figure(figsize=(6,3))
    ax = sns.countplot(imoTommsi.index.value_counts()) # AIS + Not found (Only SW)
    plt.xlabel('Unique MMSI')
    plt.ylabel('# IMO Numbers')
    plt.ylim([0,1200])
    for p in ax.patches:
        ax.annotate(p.get_height(), (p.get_x(), p.get_height()+15))

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/imoMMSIhist.png',
        bbox_inches='tight',dpi = 200)

def main(DB):
    # Read CSV file from Seaweb
    file = '/Users/PatrickAndreNaess/Documents/PYTHON/SeawebLPG.csv'
    df = pd.DataFrame(pd.read_csv(file))
    df.drop('OPERATOR_URL',axis=1,inplace=True)
    # Drop month in year built if wanted:
```

```
    df['Built'] = df['Built'].apply(lambda x: int(x.split('-')[0]))
    # Plot statistics:
    plotSeaweb(df)
    # Whole fleet
    LPG = df[df['Built']<2018]
    # List of IMO numbers
    IMO_LIST = list(LPG['IMO/LR/IHS_No.'])
    # All MMSIs found per IMO number
    imoMMSI = imo2mmsi(DB,IMO_LIST)
    # Order LPG set by IMO number and match with imoTommsi to get capacity per MMSI
    LPG.set_index('IMO/LR/IHS_No.',inplace=True)
    # All MMSIs from Seaweb
    seaMMSI = pd.DataFrame(LPG['MMSI'].dropna().apply(lambda x: int(x)))
    # Add MMSIs from SW not found in imoMMSI, to imoMMSI. Might be in MessageType1
    imoTommsi = imoMMSI.append(seaMMSI).drop_duplicates()
    # Number of MMSIs found per IMO
    LPG = LPG.join(pd.DataFrame(imoMMSI.index.value_counts(),columns=['Num MMSI']))
    #Add those not found to imoMMSI, might be in MessageType1

    # New LPG df indexed by all MMSI found and not found in Mt5
    mmsi_capacity = imoTommsi.join(LPG.drop(['MMSI'],axis=1))
    mmsi_capacity['IMO'] = mmsi_capacity.index
    mmsi_capacity.set_index('MMSI',inplace=True)

    # Save mmsi_capacity to excel
    output = '/Users/PatrickAndreNaess/Documents/PYTHON/mmsi_capacity.xlsx'
    writer = pd.ExcelWriter(output)
    mmsi_capacity.to_excel(writer,'Sheet1')
    writer.save()

    # MMSIs used to create database:
    MMSI_LIST = str()
    for i,mi in imoTommsi['MMSI'].iteritems():
        MMSI_LIST += ','+str(mi)

    return MMSI_LIST
```

E.4 Daily Data Interpolation (Interpolate_data.py)

Daily interpolation of AIS data for every vessel identity. If applicable.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  6 18:52:43 2018

@author: PatrickAndreNaess
"""
import pandas as pd

def main(df_1):
    '''
    Interpolates Message Type 1 data:
        df_1: DataFrame with Message Type 1 data
    returns: DataFrame with daily interpolated data
    '''
    df = df_1.copy().sort_values(by=['Unixtime'])

    df.index = pd.to_datetime(df['Unixtime'],unit='s')

    df.index = df.index.normalize()

    NEW = pd.DataFrame()

    for MMSI, new_df in df.groupby('MMSI'):

        new_df = new_df[~new_df.index.duplicated(keep='last')]

        new_df = new_df.resample('D').asfreq().interpolate(axis=0,method='linear')

        NEW = NEW.append(new_df)

    NEW['MMSI'] = NEW['MMSI'].astype(int)
    NEW['Unixtime'] = NEW['Unixtime'].astype(int)
    return NEW
```

E.5 Price Data Importation (Ocean_mesh.py)

Creation and plotting of an ocean mesh of nodes to calculate shortest sailing distance between two arbitrary points using Dijkstra's shortest path algorithm.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Apr  6 16:59:10 2018

@author: PatrickAndreNaess
"""
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

def GenerateOceanMesh():
    '''
    Generate the ocean mesh based on imported csv
    returns: NetworkX Graph and the node coordinates
    '''
    # Import CSV with points and distances
    file_p = '/Users/PatrickAndreNaess/Documents/PYTHON/mesh_points.csv'
    file_d = '/Users/PatrickAndreNaess/Documents/PYTHON/mesh_hrany.csv'

    points = pd.DataFrame(pd.read_csv(file_p,sep=';'))
    dist = pd.DataFrame(pd.read_csv(file_d,sep=';'))

    # Make a network graph:
    G=nx.Graph()

    # Make nodes:
    for index,row in points.iterrows():
        G.add_node(index,pos=(row['lon'],row['lat']))

    # Add edges with weights:
    for index,row in dist.iterrows():
        G.add_edge(int(row['i']),int(row['j']),weight=row['dist'])

    return G,points

def PlotOceanMesh(i,j):
    '''
    Plots the mesh/grid with/without shortest path between two points:
        i,j: Node number (set (None,None) if only mesh is wanted)
    '''
    # Import CSV with points and distances
    file_p = '/Users/PatrickAndreNaess/Documents/PYTHON/mesh_points.csv'
    file_d = '/Users/PatrickAndreNaess/Documents/PYTHON/mesh_hrany.csv'

    points = pd.DataFrame(pd.read_csv(file_p,sep=';'))
    dist = pd.DataFrame(pd.read_csv(file_d,sep=';'))

    # Make a network graph:
    G=nx.Graph()

    # Make nodes:
    for index,row in points.iterrows():
        G.add_node(index,pos=(row['lon'],row['lat']))

    # Add edges with weights:
    for index,row in dist.iterrows():
        G.add_edge(int(row['i']),int(row['j']),weight=row['dist'])

        # Remove edges that cross the lon 180/-180 line (only for plotting):
        lon_i = points.iat[int(row['i'])]['lon']
        lon_j = points.iat[int(row['j'])]['lon']
        x = max(lon_i,lon_j) # far right on map
```

```
            y = min(lon_i,lon_j) # far left on map
            if x > 160 and y < -160:
                G.remove_edge(int(row['i']),int(row['j']))

    # Visualizing the network:
    fig, ax = plt.subplots(figsize=(18,18))

    minlon = -180
    minlat = -80
    maxlon = 180
    maxlat = 83
    lat0 = (maxlat+minlat)/2
    lon0 = (maxlon+minlon)/2
    lat1 = (maxlat+minlat)/2-20

    m = Basemap(llcrnrlon=minlon,llcrnrlat=minlat,urcrnrlon=maxlon,urcrnrlat=maxlat,\
                rsphere=(6378137.00,6356752.3142),\
                resolution='l',projection='cyl',\
                lat_0=lat0,lon_0=lon0,lat_ts = lat1)
    m.drawmapboundary(fill_color='white')
    m.fillcontinents(color='lightgray',lake_color='white')

    pos=nx.get_node_attributes(G,'pos')
    nx.draw_networkx_nodes(G,pos,node_size=0)
    nx.draw_networkx_edges(G,pos,alpha=0.2,edge_color='b',width=0.5)

    if i or j:
        path = nx.dijkstra_path(G,source=i,target=j)
        path_edges = list(zip(path,path[1:]))
        nx.draw_networkx_nodes(G,pos,nodelist=path,node_size=0)
        nx.draw_networkx_edges(G,pos,edgelist=path_edges,edge_color='b',width=2)

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/OceanMesh.png',
        bbox_inches='tight',dpi = 100)

def ShortestSeaDistance(source,target,nodes,Graph):
    '''
    Calculate the shortest sea distance between two arbitrary coordinates:
        source: (lon,lat) coordinate of source position
        target: (lon,lat) coordinate of target position
        nodes: DataFrame with node positions generated by GenerateOceanMesh()
        Graph: NetworkX graph generated by GenerateOceanMesh()
    returns: Shortest path between the source and target based on Dijkstra's
             algorithm
    '''
    nodes = np.asarray(nodes)

    deltas_s = nodes - source
    dist_2_s = np.einsum('ij,ij->i', deltas_s, deltas_s)
    i = np.argmin(dist_2_s)

    deltas_t = nodes - target
    dist_2_t = np.einsum('ij,ij->i', deltas_t, deltas_t)
    j = np.argmin(dist_2_t)

    length = nx.dijkstra_path_length(Graph,source=i,target=j)

    return length
```

E.6 Price Data Importation (Price_data.py)

Importing and cleaning of price and market data into a structured form.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 11 09:31:25 2018

@author: PatrickAndreNaess
"""
import pandas as pd

def main():
    '''
    Imports price data from file:
    returns: Separate spot and crude oil prices
    '''
    # Soot prices
    spot = pd.read_excel('2018-06-11 Spot & Forward prices Propane.xlsx',
                         sheet_name='Spot Prices MB-ARA-FEI-CP').set_index('Date')

    oil = pd.read_csv('crude_oil_WTI.csv').set_index('DATE').dropna()
    oil.index = pd.to_datetime(oil.index)
    oil.columns = ['price']

    spot['Prices'] = spot['Prices'].astype('float32')
    spot_ = pd.DataFrame()

    for name in spot['Quote'].unique():
        spot_[name] = spot[spot['Quote']==name]['Prices']

    spot_MB = pd.DataFrame()
    spot_MB['price'] = spot_['Propane MB']
    spot_FEI = pd.DataFrame()
    spot_FEI['price'] = spot_['Propane FEI']
    spot_NWE = pd.DataFrame()
    spot_NWE['price'] = spot_['Propane CIF NWE']
    spot_CP = pd.DataFrame()
    spot_CP['price'] = spot_['Propane CP M01']

    return spot_MB,spot_FEI,spot_NWE,spot_CP,oil
```

E.7 Data Preparation (Data_preparation.py)

Data preparation of generated time series so that it is applicable to supervised learning problems.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu May 10 19:40:52 2018

@author: PatrickAndreNaess
"""
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller

def plot_time_series(data):
    '''
    Plotting all the times series:
        data: DataFrame with time series data
    '''
    values = data.values

    train_size = round(len(values)*0.9)
    validation_size = round(train_size*0.9)

    train_data = values[0:validation_size+1]
    validation_data = values[validation_size:train_size+1]
    test_data = values[train_size:]

    groups = list(range(0,len(data.columns)))
    i = 1
    # plot each column
    plt.figure(figsize=(20,20))
    for group in groups:
        plt.subplot(len(groups), 1, i)
        plt.plot(train_data[:, group],label='training_set')
        plt.plot(range(validation_size,train_size+1),validation_data[:, group],label='validation_set')
        plt.plot(range(train_size,len(values)),test_data[:, group],label='testing_set')
        plt.title(data.columns[group],y=0.85,loc='right')
        if i == 1:
            plt.legend()
        plt.xlim([-1,len(data)+1])
        i += 1
        plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/plot_time_series.pdf',
            bbox_inches='tight')

def normalize(data):
    '''
    Normalizing the time series data:
        data: DataFrame with time series data
    returns: DataFrame with normalized data and the scaler
    '''
    # Normalize data
    for i,col in data.iteritems():
        scaler = MinMaxScaler(feature_range=(0,1))
        data[i] = scaler.fit_transform(data[i].values.reshape(-1, 1))
        data = data[data.columns].astype('float32')

    return data,scaler

def difference_normalize(data):
    '''
    Normalizing and first difference of the time series data:
        data: DataFrame with time series data
    returns: DataFrame with differenced normalized data and the scaler
    '''
    # Difference the data
    data = data.diff().dropna()

    # Normalize data
    for i,col in data.iteritems():
        scaler = MinMaxScaler(feature_range=(0,1))
        data[i] = scaler.fit_transform(data[i].values.reshape(-1, 1))
        data = data[data.columns].astype('float32')

    return data,scaler

def log_difference_normalize(data):
    '''
    Normalizing and logarithmic differencing the time series data:
        data: DataFrame with time series data
    returns: DataFrame with log differenced normalized data and the scaler
    '''
    # Difference the data
    data = data.apply(np.log)
    #data = data.replace([np.inf, -np.inf], np.nan)
    data = data.dropna().diff().dropna().reset_index(drop=True)

    # Normalize data
    for i,col in data.iteritems():
        scaler = MinMaxScaler(feature_range=(0,1))
        data[i] = scaler.fit_transform(data[i].values.reshape(-1, 1))
        data = data[data.columns].astype('float32')

    return data,scaler

def stationarity_check(data):
    '''
    Stationarity check with the Augmented Dickey-Fuller test:
        data: DataFrame with time series data
    returns: DataFrame with test statistics
    '''
    AF = pd.DataFrame()

    for name,series in data.iteritems():
        result = adfuller(series)
        AF.at['ADF Statistic',name] = result[0]
        AF.at['p-value',name] = result[1]
        for key, value in result[4].items():
            AF.loc[key,name] = value
        for key, value in result[4].items():
            AF.loc['Accept H0 at '+key,name] = (result[0]>value)

    return AF

def supervised_learning(lag,df):
    '''
    Creates a DataFrame with features and lags:
        lag: Max lag to output
        df: DataFrame with time series data
    returns: DataFrame with the series and lags
    '''
    df_data,scaler = difference_normalize(df)

    dataframe = pd.DataFrame()
    for name,d in df_data.iteritems():
        series = df_data[name]
        for i in range(lag,0,-1):
            dataframe[name+'(t-'+str(i)+')'] = series.shift(i)
        dataframe[name+'(t)'] = series
    dataframe = dataframe[lag:]

    dataframe['price(t+1)']=dataframe['price(t)'].shift(-1)

    # Only perform features selection on in-sample data (training data)
    train_size = round(len(df)*0.9)
    train_data = dataframe[0:train_size]

    return train_data.dropna()

def inverse(pred,test,last_obs,scl):
    '''
    Inverese scaling the predicted dataset:
        pred: Predictions
        test: Real values
        last_obs: Last observed value in the training set
        scl: The scaler for which to inverse transform
    returns: DataFrame with inverse transformed date
    '''
    inverted = scl.inverse_transform(pred)
    return inverted

def inverse_transfrom(pred,test,last_obs,scl):
    '''
    Inverese scaling and differencing the predicted dataset:
        pred: Predictions
        test: Real values
        last_obs: Last observed value in the training set
        scl: The scaler for which to inverse transform
    returns: DataFrame with inverse transformed date
    '''
    pred = scl.inverse_transform(pred)
    inverted = list()
    inverted.append(pred[0] + last_obs)

    for i in range(1, len(pred)):
        inverted.append(pred[i] + test[i-1])
    return inverted

def inverse_log_transfrom(pred,test,last_obs,scl):
    '''
    Inverese scaling and log differencing the predicted dataset:
        pred: Predictions
        test: Real values
        last_obs: Last observed value in the training set
        scl: The scaler for which to inverse transform
    returns: DataFrame with inverse transformed date
    '''
    last_ln_obs = np.log(last_obs)
    pred = scl.inverse_transform(pred)
    inverted = list()
    inverted.append(pred[0] + last_ln_obs)

    for i in range(1, len(pred)):
        inverted.append(pred[i] + np.log(test[i-1]))
    return np.exp(inverted)

if __name__ == '__main__':
    data = pd.read_csv('timeseries_data.csv').set_index('timestamp')
    data_processed,scaler = difference_normalize(data)

    plot_time_series(data)
    plot_time_series(data_processed)
    adf = stationarity_check(data_processed)
```

E.8 Feature Importance Scores (Feature_importance.py)

Incorporates the filter selection methods, where features are ranked based on a statistical scores from various variable importance methods.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 18 16:33:32 2018

@author: PatrickAndreNaess
"""
import pandas as pd
import numpy as np
import Data_preparation as DP
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import f_regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import RidgeCV,LassoCV,LinearRegression
import matplotlib.pyplot as plt
from minepy import MINE
import seaborn as sns

def scale_rank(rank,name,order=1):
    '''
    Scale the feature importance score in range [0,1]:
        rank: List with scores
        name: Name of method
    returns: Dictionary with scores for the specific method
    '''
    scale = MinMaxScaler()
    rank = scale.fit_transform(order*np.array([rank]).T).T[0]
    rank = map(lambda x: round(x, 2), rank)
    return dict(zip(name, rank))

def main(lag,data):
    '''
    Calculate and plot filter features selection scores:
        lag: Max lag of interest
        data: DataFrame with transformed time series data
    returns: DataFrame with normalized filter feature importance ratings
    '''
    df = DP.supervised_learning(lag,data)

    array = df.dropna().values.astype('float32')

    # Split into input and output (right most is price(t+1))
    X = array[:,0:-1]
    Y = array[:,-1]

    names = df.columns.values[0:-1]
    ranks = {}

    # Linear Regression
    lr = LinearRegression()
    lr.fit(X, Y)
    ranks['Linear Reg.'] = scale_rank(np.abs(lr.coef_), names)

    # Ridge Regression with cross validation to find the tuning parameter
    ridge = RidgeCV()
    ridge.fit(X, Y)
    ranks['Ridge Reg.'] = scale_rank(np.abs(ridge.coef_), names)

    # Lasso Regression with cross validation to find the tuning parameter
    lasso = LassoCV()
    lasso.fit(X, Y)
    ranks['Lasso'] = scale_rank(np.abs(lasso.coef_), names)

    # Random Forests
    rf = RandomForestRegressor()
    rf.fit(X,Y)
    ranks['RF'] = scale_rank(rf.feature_importances_, names)

    # Linear Correlation
    f, pval  = f_regression(X, Y, center=True)
    ranks['Linear Corr.'] = scale_rank(f, names)

    # MIC
    mine = MINE()
    mic_scores = []
    for i in range(X.shape[1]):
        mine.compute_score(X[:,i], Y)
        m = mine.mic()
        mic_scores.append(m)
    ranks['MIC'] = scale_rank(mic_scores, names)

    # Mean score
    r = {}
    for name in names:
        r[name] = round(np.mean([ranks[method][name] for method in ranks.keys()]), 2)
    ranks['Mean Score'] = r

    ratings = pd.DataFrame(ranks)

    # Plot top features based on mean score
    ratings['Mean Score'].sort_values()[-20:].plot(kind='bar',figsize = (10,2),color='b')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/mean_score.pdf',
            bbox_inches='tight')
    j = 0
    for i in range(0,len(ratings),63):
        j += 1
        plt.subplots(figsize=(12,0.3*len(ratings[i:i+63])))
        sns.heatmap(ratings[i:i+63],cmap='coolwarm', annot=True,cbar=False,
                    linewidths=.1, vmin=0, vmax=1)
        plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/FI/FI_'+str(j)+'.pdf',
                bbox_inches='tight')
    return ratings
```

E.9 Subset Selection (Subset_selection.py)

Contains linear subset selection methods where subset of features are tested linearly to get the best linear subset combination.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri May  4 19:21:45 2018

@author: PatrickAndreNaess
"""
import pandas as pd
import Data_preparation as DP
import itertools
import time
import statsmodels.api as sm
import matplotlib.pyplot as plt

def processSubset(feature_set,y,X):
    '''
    Fits the subset to a linear model based on OLS:
        feature_set: Subset of features
        y: Price to predict
        X: Features data
    returns: Fitted model with respective RSS
    '''
    # Fit model on feature_set and calculate RSS
    model = sm.OLS(y,X[list(feature_set)])
    reg = model.fit()
    rss = ((reg.predict(X[list(feature_set)]) - y) ** 2).sum()
    return {'model':reg, 'RSS':rss}

def calculateCombinations(k,y,X):
    '''
    Calculate all feature combinations to find the best subset:
        k: Number of features to include
        y: Price to predict
        X: Features data
    returns: All fitted models, the best model and processing time
    '''
    tic = time.time()
    results = []

    for combo in itertools.combinations(X.columns, k):
        results.append(processSubset(combo,y,X))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
    models['k'] = k
    # Calculate adjusted R^2 statistic
    models['rsquared_adj'] = models.apply(lambda row: row[1].rsquared_adj, axis=1)
    # Calculate adjusted AIC
    models['aic'] = models.apply(lambda row: row[1].aic, axis=1)
    # Calculate adjusted BIC
    models['bic'] = models.apply(lambda row: row[1].bic, axis=1)

    best_model = models.loc[models['RSS'].idxmin()]

    toc = time.time()
    t = (toc-tic)
    print('Processed',models.shape[0],'models on',k,'features in',t,'seconds.')

    return models.drop('model',axis=1),best_model,t

def forward(features,y,X):
    '''
    Finds the best model based on forward selection:
        features: Subset of features
        y: Price to predict
        X: Features data
    returns: Best forward selection model
    '''
    # Pull out predictors we still need to process
    remaining_predictors = [p for p in X.columns if p not in features]

    tic = time.time()

    results = []

    for p in remaining_predictors:
        results.append(processSubset(features+[p],y,X))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
    # Calculate adjusted R^2 statistic
    models['rsquared_adj'] = models.apply(lambda row: row[1].rsquared_adj, axis=1)
    # Calculate adjusted AIC
    models['aic'] = models.apply(lambda row: row[1].aic, axis=1)
    # Calculate adjusted BIC
    models['bic'] = models.apply(lambda row: row[1].bic, axis=1)
    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].idxmin()]

    toc = time.time()
    t = (toc-tic)
    print("Processed ",models.shape[0],'models on',len(features)+1,'features in',t,'seconds.')

    # Return the best model, along with some other useful information about the model
    return best_model,t

def backward(features,y,X):
    '''
    Finds the best model based on backward selection:
        features: Subset of features
        y: Price to predict
        X: Features data
    returns: Best backward selection model
    '''
    tic = time.time()

    results = []

    for combo in itertools.combinations(features, len(features)-1):
        results.append(processSubset(combo,y,X))

    # Wrap everything up in a nice dataframe
    models = pd.DataFrame(results)
    # Calculate adjusted R^2 statistic
    models['rsquared_adj'] = models.apply(lambda row: row[1].rsquared_adj, axis=1)
    # Calculate adjusted AIC
    models['aic'] = models.apply(lambda row: row[1].aic, axis=1)
    # Calculate adjusted BIC
    models['bic'] = models.apply(lambda row: row[1].bic, axis=1)
    # Choose the model with the highest RSS
    best_model = models.loc[models['RSS'].idxmin()]

    toc = time.time()
    t = (toc-tic)
    print('Processed ',models.shape[0],'models on',len(features)-1,'features in',t,'seconds.')

    # Return the best model, along with some other useful information about the model
    return best_model,t

############################## CALCULATIONS ###############################

def main(lag,method,data):
    '''
    Preforms the linear subset selection analysis:
        lag: Max lag of interest
        method: 'best' or 'forward', 'backward'
        data: DataFrame with transformed time series data
    returns: Best models
    '''
    # Import data
    dataframe = DP.supervised_learning(lag,data)

    # Separate the features and predicted variable
    y = dataframe['price(t+1)']
    X = dataframe.drop('price(t+1)',axis=1)

    #Select method: 'best', 'forward', 'backward'
    #method = 'backward'

    ################################ BEST SUBSET ############################
    if method == 'best':
        models = pd.DataFrame(columns=['RSS','rsquared_adj','k','aic','bic'])
        models_best = pd.DataFrame(columns=['RSS', 'model','rsquared_adj','aic','bic'])
        tid = pd.DataFrame(columns=['time'])

        max_subset_size = 4

        tic = time.time()
        for k in range(1,max_subset_size+1):
            mod,models_best.loc[k],tid.loc[k] = calculateCombinations(k,y,X)
            models = models.append(pd.DataFrame(mod),ignore_index=True)

        toc = time.time()
        print('Total elapsed time:', (toc-tic), 'seconds.')


    ################################## FORWARD ##############################
    elif method == 'forward':
        models_best = pd.DataFrame(columns=['RSS', 'model','rsquared_adj','aic','bic'])
        tid = pd.DataFrame(columns=['time'])

        tic = time.time()
        features = []

        for i in range(1,len(X.columns)+1):
            models_best.loc[i],tid.loc[i] = forward(features,y,X)
            features = models_best.loc[i]['model'].model.exog_names

        toc = time.time()
        print('Total elapsed time:', (toc-tic), 'seconds.')

    ################################# BACKWARD ##############################
    elif method == 'backward':
        models_best = pd.DataFrame(columns=['RSS','model','rsquared_adj','aic','bic'])
        tid = pd.DataFrame(columns=['time'])

        tic = time.time()
        features = X.columns

        while(len(features) > 1):
            models_best.loc[len(features)-1],tid.loc[len(features)-1] = backward(features,y,X)
            features = models_best.loc[len(features)-1]['model'].model.exog_names

        toc = time.time()
        print('Total elapsed time:', (toc-tic), 'seconds.')

    models_best[['RSS','rsquared_adj','aic','bic']] = models_best[['RSS',
                'rsquared_adj','aic','bic']].astype('float64')

    ############################### PLOT FIGURE #############################

    plt.figure(figsize=(20,10))
    plt.subplot(2, 2, 1)

    # Plot RSS
    #plt.scatter(models['k'],models['RSS'],s=5,marker='.',c='grey')
    plt.plot(models_best['RSS'])
    plt.plot(models_best['RSS'].idxmin(), models_best['RSS'].min(),'or')
    plt.xlabel('# Predictors')
    plt.ylabel('RSS')

    # Plot adjusted R^2 statistic
    plt.subplot(2, 2, 2)
    #plt.scatter(models['k'],models['rsquared_adj'],s=5,marker='.',c='grey')
    plt.plot(models_best['rsquared_adj'])
    plt.plot(models_best['rsquared_adj'].idxmax(), models_best['rsquared_adj'].max(),'or')
    plt.xlabel('# Predictors')
    plt.ylabel('adjusted R^2')

    # Plot adjusted AIC
    plt.subplot(2, 2, 3)
    #plt.scatter(models['k'],models['aic'],s=5,marker='.',c='grey')
    plt.plot(models_best['aic'])
    plt.plot(models_best['aic'].idxmin(), models_best['aic'].min(),'or')
    plt.xlabel('# Predictors')
    plt.ylabel('AIC')

    # Plot adjusted BIC
    plt.subplot(2, 2, 4)
    #plt.scatter(models['k'],models['bic'],s=5,marker='.',c='grey')
    plt.plot(models_best['bic'])
    plt.plot(models_best['bic'].idxmin(), models_best['bic'].min(),'or')
    plt.xlabel('# Predictors')
    plt.ylabel('BIC')

    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/subset_selection.pdf',
                bbox_inches='tight')

    plt.figure(figsize=(10,5))
    plt.plot(tid)
    plt.xlabel('# Predictors')
    plt.ylabel('Processing time [seconds]')
    plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/subset_selection_time.pdf',
          bbox_inches='tight')

    return models_best
```

E.10 Multilayer Perceptron Model (MLP.py)

Training, prediction and construction of a Multilayer Perceptron (MLP) model with hyperparameter optimization using a genetic algorithm.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 16 18:38:49 2018

@author: PatrickAndreNaess
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import Data_preparation as DP
from sklearn.metrics import mean_squared_error,mean_absolute_error
from Model_evaluation import DAR

from keras.layers import Dense
from keras.models import Sequential
from keras.optimizers import Adam
from keras import backend as K

from deap import base, creator, tools, algorithms
from scipy.stats import bernoulli
from bitstring import BitArray

def prepare_dataset_MLP(data, window_size):
    '''
    Prepare dataset for MLP based on window method:
        data: DataFrame with time series data
        window_size: Sliding window size
    returns: Right shape numpy array with predictors X and prediction Y
    '''
    n_features = len(data[0])

    # MAKE SURE RIGHT MOST COLUMN IS PREDICTED VARIABLE
    x, y = data[:,:n_features], data[:,-1].reshape(-1,1)

    X, Y = np.empty((0,window_size * n_features)), np.empty((0))

    for i in range(len(x)-window_size):
        X = np.vstack((X,[item for sublist in x[i:(i + window_size),:] for item in sublist]))
        Y = np.append(Y,y[i + window_size])
    Y = np.reshape(Y,(len(Y),1))

    return X, Y

def train_evaluate_MLP(ga_individual_solution):
    '''
    Test an individual from the genetic algorithm:
        ga_individual_solution: INdividual represented by binary tuple
    returns: MAE performance of MLP (fitness of the individual)
    '''
    K.clear_session()

    # Decode GA solution to integer for window_size and num_units
    window_size = 1+BitArray(ga_individual_solution[0:5]).uint
    neurons_hidden_1 = (BitArray(ga_individual_solution[5:11]).uint+1)*4
    neurons_hidden_2 = (BitArray(ga_individual_solution[11:17]).uint+1)*4
    neurons_hidden_3 = (BitArray(ga_individual_solution[17:23]).uint+1)*4
    learning_rate_power = 1+BitArray(ga_individual_solution[23:24]).uint
    learning_rate = 10**(-learning_rate_power)
    number_of_layers = BitArray(ga_individual_solution[24:]).uint

    if number_of_layers == 0:
        return 100,

    # Print solution:
```

```
    print('\nWindow Size:',window_size)
    print('Num of layers:',number_of_layers)
    print('Num of neurons in layer 1:',neurons_hidden_1)
    if number_of_layers == 2:
        print('Num of neurons in layer 2:',neurons_hidden_2)
    elif number_of_layers == 3:
        print('Num of neurons in layer 2:',neurons_hidden_2)
        print('Num of neurons in layer 3:',neurons_hidden_3)
    print('Learning rate:',learning_rate)

    # Prepare data based on window size
    X,Y = prepare_dataset_MLP(train_data,window_size)

    # Split into traingin and validation set
    train_size = round(len(Y)*(8/9))
    X_train, X_val = X[:train_size], X[train_size:]
    y_train, y_val = Y[:train_size], Y[train_size:]

    model = Sequential()
    # Hidden layer 1
    model.add(Dense(neurons_hidden_1, activation='sigmoid', input_dim = X_train.shape[1]))

    if number_of_layers == 2:
        # Hidden layer 2
        model.add(Dense(neurons_hidden_2, activation='sigmoid'))

    elif number_of_layers == 3:
        # Hidden layer 2
        model.add(Dense(neurons_hidden_2, activation='sigmoid'))
        # Hidden layer 3
        model.add(Dense(neurons_hidden_3, activation='sigmoid'))

    # Output layer
    model.add(Dense(1, activation = 'linear'))

    optimizer = Adam(lr = learning_rate)
    model.compile(loss = 'mae', optimizer = optimizer)

    # Fit data to model
    model.fit(X_train,y_train,
              epochs=400,batch_size=128,
              verbose = 0,
              shuffle=True)

    # Make prediction
    y_pred = model.predict(X_val)

    # Calculate the MAE score as fitness for GA
    mae = mean_absolute_error(y_val,y_pred)
    print('\nValidation MAE: ', mae,'\n')
    return mae,

def geneticAlgorithm():
    '''
    Run the genetic algorithm:
    returns: Resulting population, log of algorithm and Hall of Fame individual
    '''
    # Select the parameters of the Genetic Algorithm
    population_size = 100 # Number of individuals
    num_generations = 20 # Times the individuals should evolve
    gene_length = 26 # Binary tuple length individuals

    # Structured in the following way:
    # [ws*5,h1*6,h2*6,h3*6,lr,ly*2] = length 26

    # Defining the GA from the Deap library
    # Minimize the RMSE score, using -1.0. In case, maximize accuracy, use 1.0

    creator.create('FitnessMax', base.Fitness, weights = (-1.0,))
    creator.create('Individual', list , fitness = creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register('binary', bernoulli.rvs, 0.5)
    toolbox.register('individual', tools.initRepeat, creator.Individual,
                    toolbox.binary, n = gene_length)
    toolbox.register('population', tools.initRepeat, list , toolbox.individual)
    toolbox.register('mate', tools.cxOrdered)
    toolbox.register('mutate', tools.mutShuffleIndexes, indpb = 0.6)
    toolbox.register('select', tools.selBest) #selRoulette, selBest
    toolbox.register('evaluate', train_evaluate_MLP)

    pop = toolbox.population(n = population_size)
    hof = tools.HallOfFame(1)

    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("min", np.min)

    pop, log = algorithms.eaSimple(pop, toolbox, cxpb = 0.4, mutpb = 0.1,
                                   ngen = num_generations, halloffame=hof,
                                   verbose = False)
    return pop,log,hof

######################## SELECT FEATURES OF INTEREST ########################

# Select features of interest:
    # AIS
    # nonAIS
dataset = 'AIS'

########################## IMPORT AND PROCESS DATA ##########################

if dataset == 'AIS':
    unprocessed_data = pd.read_csv('selected_AISfeatures.csv').set_index('timestamp')
elif dataset == 'AIS':
    unprocessed_data = pd.read_csv('selected_nonAISfeatures.csv').set_index('timestamp')
else:
    print('No datasets selected!')

df_data,scaler = DP.difference_normalize(unprocessed_data)
data = df_data.values.astype('float32')

# 90-10 training and testing split
train_size = round(len(data)*0.9)

train_data = data[0:train_size]

########################### GENETIC ALGORITHM ###########################

# If custom solution:(Only select Train_model = True
best_window_size = ''
best_neurons_hidden_1 = ''
best_neurons_hidden_2 = ''
best_neurons_hidden_3 = ''
best_learning_rate = ''
best_number_of_layers = ''

# Select:
Run_GA = True
Train_model = True # Set True and rest False if custom solution above
Test_individual = True # Train on solution from GA
Individual_to_test = 1 # 1 is best, 2 is second best, ...

if Run_GA:
    # Run Genetic Algorithm to get best individuals
    population,logbook,hof = geneticAlgorithm()

    # Get Algorithm statistics
    fitness = pd.DataFrame()
    for l in logbook:
        fitness.loc[l.get('gen'),'average'] = l.get('avg')
        fitness.loc[l.get('gen'),'best'] = l.get('min')

if Test_individual:
    best_individuals = tools.selBest(population,k = Individual_to_test)
    print('\nTraining model with, ' + dataset)
    print('Nr. %i best individual:' % Individual_to_test)
    for bi in best_individuals:

        best_window_size = 1+BitArray(bi[0:5]).uint
        best_neurons_hidden_1 = (BitArray(bi[5:11]).uint+1)*4
        best_neurons_hidden_2 = (BitArray(bi[11:17]).uint+1)*4
        best_neurons_hidden_3 = (BitArray(bi[17:23]).uint+1)*4
        learning_rate_power = 1+BitArray(bi[23:24]).uint
        best_learning_rate = 10**(-learning_rate_power)
        best_number_of_layers = BitArray(bi[24:]).uint

elif Train_model:
    print('\nTraining model with, ' + dataset)
    print('Costum solution:')

if Train_model:
    # Print solution:
    print('\nWindow Size:',best_window_size)
    print('Num of layers:',best_number_of_layers)
    print('Num of neurons in layer 1:',best_neurons_hidden_1)
    if best_number_of_layers == 2:
        print('Num of neurons in layer 2:',best_neurons_hidden_2)
    if best_number_of_layers == 3:
        print('Num of neurons in layer 2:',best_neurons_hidden_2)
        print('Num of neurons in layer 3:',best_neurons_hidden_3)
    print('Learning rate:',best_learning_rate,'\n')

    ########################### CREATE BEST MODEL ###########################

    # Define datasets based on window size
    X_train,y_train = prepare_dataset_MLP(train_data,best_window_size)

    test_data = data[train_size-best_window_size:]
    X_test, y_test = prepare_dataset_MLP(test_data,best_window_size)

    # Make dataframes to take averages of in the end:
    df_history_loss = pd.DataFrame()
    df_history_val_loss = pd.DataFrame()
    df_y_pred = pd.DataFrame()

    for run in range(2):
        print('Runn nr. %i' % (run+1))

        K.clear_session()

        model = Sequential()
        # Hidden layer 1
        model.add(Dense(best_neurons_hidden_1, activation='sigmoid',
                        input_dim = X_train.shape[1]))

        if best_number_of_layers == 2:
            # Hidden layer 2
            model.add(Dense(best_neurons_hidden_2, activation='sigmoid'))

        elif best_number_of_layers == 3:
            # Hidden layer 2
            model.add(Dense(best_neurons_hidden_2, activation='sigmoid'))
            # Hidden layer 3
            model.add(Dense(best_neurons_hidden_3, activation='sigmoid'))

        # Output layer
        model.add(Dense(1, activation = 'linear'))

        optimizer = Adam(lr = best_learning_rate)
        model.compile(loss = 'mae', optimizer = optimizer)
        # Fit data to model
        history = model.fit(X_train,y_train,
                            epochs=200,batch_size=128,
                            validation_data=(X_test, y_test),
                            verbose=0,
                            shuffle=True)

        ########################## MAKE PREDICTION ##########################

        y_pred = model.predict(X_test)

        ############################## SAVE RUN ##############################

        df_history_loss[str(run)] = pd.Series(history.history['loss'])
        df_history_val_loss[str(run)] = pd.Series(history.history['val_loss'])
        df_y_pred[str(run)] = pd.Series(y_pred[:,0])

    ########################### AVERAGE THE RESULTS ###########################

    y_pred = df_y_pred.mean(axis=1).values.reshape(-1,1)
    history_loss = df_history_loss.mean(axis=1).values
    history_val_loss = df_history_val_loss.mean(axis=1).values

    def plot_mlp_training():
        # Summarize history for loss
        plt.plot(history_loss)
        plt.plot(history_val_loss)
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.ylim(ymax=0.25)
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()

    plot_mlp_training()

    ######################## INVERSE TRANSFORM DATA ########################

    y_real = unprocessed_data['price'].iloc[-len(y_test):].values.reshape(-1,1)

    last_obs = unprocessed_data['price'].iloc[-len(y_test)-1]

    y_forecast = DP.inverse_transfrom(y_pred,y_real,last_obs,scaler)

    ############################ EVALUATE RESULTS ############################

    def evaluate_results(real,pred):

        rmse = np.sqrt(mean_squared_error(real, pred))
        print('\nTest RMSE: ', rmse)
        mae = mean_absolute_error(real,pred)
        print('Test MAE: ', mae)
        mape = np.mean(np.abs((real-pred)/real))*100
        print('Test MAPE: ',mape,'%')
        dar = DAR(real,pred,last_obs)
        print('Test DAR: ',dar)

        forecast = pd.DataFrame(data = pred, columns=['pred'])
        forecast['real'] = pd.DataFrame(real)
        forecast.index = pd.to_datetime(unprocessed_data[-len(y_test):].index)
        forecast.to_csv('MLP_forecast_'+dataset+'.csv')

        # Plot forecast
        forecast.plot(figsize=(8,4))
        plt.scatter(forecast.index,forecast['pred'],marker='.')
        plt.scatter(forecast.index,forecast['real'],marker='.')
        plt.show()

    #evaluate_results(y_test,y_pred)
    evaluate_results(y_real,y_forecast)
```

E.11 Long Short-Term Memory Model (LSTM.py)

Training, prediction and construction of a Long Short-Term Memory (LSTM) model with hyper-parameter optimization using a genetic algorithm.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon May  7 21:07:36 2018

@author: PatrickAndreNaess
"""
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import Data_preparation as DP
from sklearn.metrics import mean_squared_error,mean_absolute_error
from Model_evaluation import DAR

from keras.layers import LSTM, Dense, Dropout
from keras.models import Sequential
from keras.optimizers import Adam
from keras import backend as K

from deap import base, creator, tools, algorithms
from scipy.stats import bernoulli
from bitstring import BitArray

def prepare_dataset_LSTM(data, window_size):
    '''
    Prepare dataset for LSTM based on window method:
        data: DataFrame with time series data
        window_size: Sliding window size
    returns: Right shape numpy array with predictors X and prediction Y
    '''
    n_features = len(data[0])

    # MAKE SURE RIGHT MOST COLUMN IS PREDICTED VARIABLE
    x, y = data[:,:n_features], data[:,-1].reshape(-1,1)

    X, Y = np.empty((0,window_size,n_features)), np.empty((0))

    for i in range(len(x)-window_size):
        X = np.vstack((X,[x[i:(i + window_size),:]]))
        Y = np.append(Y,y[i + window_size])
    Y = np.reshape(Y,(len(Y),1))

    return X, Y

def train_evaluate_LSTM(ga_individual_solution):
    '''
    Test an individual from the genetic algorithm:
        ga_individual_solution: INdividual represented by binary tuple
    returns: MAE performance of LSTM (fitness of the individual)
    '''
    K.clear_session()

    # Decode GA solution:
    window_size = 1+BitArray(ga_individual_solution[0:5]).uint
    num_units_1 = (1+BitArray(ga_individual_solution[5:10]).uint)*2
    num_units_2 = (1+BitArray(ga_individual_solution[10:15]).uint)*2
    learning_rate_power = 1+BitArray(ga_individual_solution[15:16]).uint
    learning_rate = 10**(-(learning_rate_power))
    number_of_layers = 1+BitArray(ga_individual_solution[16:]).uint

    # Print solution:
    print('\nWindow Size:',window_size)
    print('Num of layers:',number_of_layers)
    print('Num of neurons in layer 1:',num_units_1)
    if number_of_layers == 2:
        print('Num of neurons in layer 2:',num_units_2)
    print('Learning rate:',learning_rate)

    # Prepare data based on window size
    X,Y = prepare_dataset_LSTM(train_data,window_size)

    # Split into traingin and validation set
    train_size = round(len(Y)*(8/9))
    X_train, X_val = X[:train_size], X[train_size:]
    y_train, y_val = Y[:train_size], Y[train_size:]

    model = Sequential()

    if number_of_layers == 1:
        # Hidden layer 1
        model.add(LSTM(num_units_1, input_shape=(X_train.shape[1],X_train.shape[2])))
        model.add(Dropout(0.3))
        # Output 1 neuron
        model.add(Dense(1, activation = 'linear'))
    else:
        # Hidden layer 1
        model.add(LSTM(num_units_1, input_shape=(X_train.shape[1],X_train.shape[2]),
                       return_sequences=True))
        model.add(Dropout(0.3))
        # Hidden layer 2
        model.add(LSTM(num_units_2))
        model.add(Dropout(0.3))
        # Output 1 neuron
        model.add(Dense(1, activation = 'linear'))

    optimizer = Adam(lr = learning_rate)
    model.compile(loss = 'mae', optimizer = optimizer)

    # Fit data to model
    for i in range(200):
        model.fit(X_train,y_train,
                  epochs=1,batch_size=128,
                  verbose=0,
                  shuffle=False) #very important shuffle = False for LSTM
        model.reset_states()

    y_pred = model.predict(X_val)

    # Calculate the MAE score as fitness for GA
    mae = mean_absolute_error(y_val,y_pred)
    print('\nValidation MAE: ', mae,'\n')
    return mae,

def geneticAlgorithm():
    '''
    Run the genetic algorithm:
    returns: Resulting population, log of algorithm and Hall of Fame individual
    '''
    # Select the parameters of the Genetic Algorithm
    population_size = 100 # Number of individuals
    num_generations = 20 # Times the individuals should evolve
    gene_length = 17 # Binary tuple length individuals

    # Structured in the following way:
    # [wi*5,h1*5,h2*5,lr,ly] = length 17

    # Defining the GA from the Deap library
    # Minimize the RMSE score, using -1.0. In case, maximize accuracy, use 1.0

    creator.create('FitnessMax', base.Fitness, weights = (-1.0,))
    creator.create('Individual', list , fitness = creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register('binary', bernoulli.rvs, 0.5)
    toolbox.register('individual', tools.initRepeat, creator.Individual,
                     toolbox.binary, n = gene_length)
    toolbox.register('population', tools.initRepeat, list , toolbox.individual)
    toolbox.register('mate', tools.cxOrdered)
    toolbox.register('mutate', tools.mutShuffleIndexes, indpb = 0.6)
    toolbox.register('select', tools.selBest) #selRoulette
    toolbox.register('evaluate', train_evaluate_LSTM)

    pop = toolbox.population(n = population_size)
    hof = tools.HallOfFame(1)

    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("min", np.min)

    pop,log = algorithms.eaSimple(pop, toolbox, cxpb = 0.4, mutpb = 0.1,
                                   ngen = num_generations, stats=stats,
                                   halloffame=hof, verbose = False)
    return pop,log,hof

######################## SELECT FEATURES OF INTEREST ########################

# Select features of interest:
    # AIS
    # nonAIS
dataset = 'nonAIS'

########################## IMPORT AND PROCESS DATA ##########################

if dataset == 'AIS':
    unprocessed_data = pd.read_csv('selected_AISfeatures.csv').set_index('timestamp')
elif dataset == 'nonAIS':
    unprocessed_data = pd.read_csv('selected_nonAISfeatures.csv').set_index('timestamp')
else:
    print('No datasets selected!')

df_data,scaler = DP.difference_normalize(unprocessed_data)
data = df_data.values.astype('float32')

# 90-10 training and testing split
train_size = round(len(data)*0.9)

train_data = data[0:train_size]

############################# GENETIC ALGORITHM #############################

# If custom solution:(Only select Train_model = True
best_window_size = ''
best_num_units_1 = ''
best_num_units_2 = ''
best_learning_rate = ''
best_number_of_layers = ''

# Select:
Run_GA = True
Train_model = True
Test_individual = True # Set True and rest False if custom solution above
Individual_to_test = 1 # 1 is best, 2 is second best, ...

if Run_GA:
    # Run Genetic Algorithm to get best individuals
    population,logbook,hof = geneticAlgorithm()

    # Get Algorithm statistics
    fitness = pd.DataFrame()
    for l in logbook:
        fitness.loc[l.get('gen'),'average'] = l.get('avg')
        fitness.loc[l.get('gen'),'best'] = l.get('min')

if Test_individual:
    best_individuals = tools.selBest(population,k = Individual_to_test)
    print('\nTraining model with, ' + dataset)
    print('Nr. %i best individual:' % Individual_to_test)
    for bi in best_individuals:

        best_window_size = 1+BitArray(bi[0:5]).uint
        best_num_units_1 = (1+BitArray(bi[5:10]).uint)*2
        best_num_units_2 = (1+BitArray(bi[10:15]).uint)*2
        learning_rate_power = 1+BitArray(bi[15:16]).uint
        best_learning_rate = 10**(-(learning_rate_power))
        best_number_of_layers = 1+BitArray(bi[16:]).uint

elif Train_model:
    print('\nTraining model with, ' + dataset)
    print('Costum solution:')

if Train_model:
    # Print solution:
    print('\nWindow Size:',best_window_size)
    print('Num of layers:',best_number_of_layers)
    print('Num of neurons in layer 1:',best_num_units_1)
    if best_number_of_layers == 2:
        print('Num of neurons in layer 2:',best_num_units_2)
    print('Learning rate:',best_learning_rate,'\n')

    ########################### CREATE BEST MODEL ###########################

    # Define datasets based on window size:
    X_train,y_train = prepare_dataset_LSTM(train_data,best_window_size)

    test_data = data[train_size-best_window_size:]
    X_test, y_test = prepare_dataset_LSTM(test_data,best_window_size)

    # Make dataframes to take averages of in the end:
    df_history_loss = pd.DataFrame()
    df_history_val_loss = pd.DataFrame()
    df_y_pred = pd.DataFrame()

    for run in range(10):
        print('Run nr. %i' % (run+1))

        K.clear_session()

        model = Sequential()

        if best_number_of_layers == 1:
            # Hidden layer 1
            model.add(LSTM(best_num_units_1, input_shape=(X_train.shape[1],
                                                          X_train.shape[2])))
            model.add(Dropout(0.3))
            # Output 1 neuron
            model.add(Dense(1, activation = 'linear'))
        else:
            # Hidden layer 1
            model.add(LSTM(best_num_units_1,
                           input_shape=(X_train.shape[1],X_train.shape[2]),
                           return_sequences=True))
            model.add(Dropout(0.3))
            # Hidden layer 2
            model.add(LSTM(best_num_units_2))
            model.add(Dropout(0.3))
            # Output 1 neuron
            model.add(Dense(1, activation = 'linear'))

        optimizer = Adam(lr = best_learning_rate)
        model.compile(loss = 'mae', optimizer = optimizer)

        ############################ TRAIN MODEL ############################

        history_acc = list()
        history_val_acc = list()
        history_loss = list()
        history_val_loss = list()

        # Train model on number of epochs: range(epochs)
        for i in range(200):
            history = model.fit(X_train,y_train,
                                epochs=1,batch_size=128,
                                validation_data=(X_test, y_test),
                                verbose = 0,
                                shuffle=False) #important shuffle=False for LSTM
            model.reset_states()

            history_loss.append(history.history['loss'])
            history_val_loss.append(history.history['val_loss'])

        ########################## MAKE PREDICTION ##########################

        y_pred = model.predict(X_test)

        ############################## SAVE RUN ##############################

        df_history_loss[str(run)] = pd.Series(np.array(history_loss)[:,0])
        df_history_val_loss[str(run)] = pd.Series(np.array(history_val_loss)[:,0])
        df_y_pred[str(run)] = pd.Series(y_pred[:,0])

    ########################## AVERAGE THE RESULTS ##########################

    y_pred = df_y_pred.mean(axis=1).values.reshape(-1,1)
    history_loss = df_history_loss.mean(axis=1).values
    history_val_loss = df_history_val_loss.mean(axis=1).values

    def plot_lstm_training():
        '''
        Plot training results
        '''
        # Summarize history for loss
        plt.plot(history_loss)
        plt.plot(history_val_loss)
        plt.title('model loss')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.ylim(ymax=0.2)
        plt.show()

    plot_lstm_training()

    ######################## INVERSE TRANSFORM DATA ########################

    y_real = unprocessed_data['price'].iloc[-len(y_test):].values.reshape(-1,1)

    last_obs = unprocessed_data['price'].iloc[-len(y_test)-1]

    y_forecast = DP.inverse_transfrom(y_pred,y_real,last_obs,scaler)

    ########################### EVALUATE RESULTS ###########################

    def evaluate_results(real,pred):
        '''
        Evaluate forecasting results and save to csv:
            real: Real price
            pred: Prediction of price
        '''
        rmse = np.sqrt(mean_squared_error(real, pred))
        print('\nTest RMSE: ', rmse)
        mae = mean_absolute_error(real,pred)
        print('Test MAE: ', mae)
        mape = np.mean(np.abs((real-pred)/real))*100
        print('Test MAPE: ',mape,'%')
        dar = DAR(real,pred,last_obs)
        print('Test DAR: ',dar)

        forecast = pd.DataFrame(data = pred, columns=['pred'])
        forecast['real'] = pd.DataFrame(real)
        forecast.index = pd.to_datetime(unprocessed_data[-len(y_test):].index)
        forecast.to_csv('LSTM_forecast_'+dataset+'.csv')

        # Plot forecast
        forecast.plot(figsize=(8,4))
        plt.scatter(forecast.index,forecast['pred'],marker='.')
        plt.scatter(forecast.index,forecast['real'],marker='.')
        plt.show()

    #evaluate_results(y_test,y_pred)
    evaluate_results(y_real,y_forecast)
```

E.12 Vector Autoregressive Model (VAR.py)

Construction and prediction with a Vector Autoregressive (VAR) model to baseline neural network predictions.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May  6 11:01:53 2018

@author: PatrickAndreNaess
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import Data_preparation as DP
from sklearn.metrics import mean_squared_error,mean_absolute_error
from statsmodels.tsa.api import VAR
from Model_evaluation import DAR

######################## SELECT FEATURES OF INTEREST ########################

# Select features of interest:
    # AIS
    # nonAIS
dataset = 'nonAIS'

########################## IMPORT AND PROCESS DATA ##########################

if dataset == 'AIS':
    unprocessed_data = pd.read_csv('selected_AISfeatures.csv').set_index('timestamp')
elif dataset == 'nonAIS':
    unprocessed_data = pd.read_csv('selected_nonAISfeatures.csv').set_index('timestamp')
else:
    print('No datasets selected!')

print('\nFitting model with, ' + dataset)

data,scaler = DP.difference_normalize(unprocessed_data)
data.index = unprocessed_data[1:].index

# 90-10 training and testing split
train_size = round(len(data)*0.90)
train_data = data[0:train_size]

# Create model:
model = VAR(train_data)

# Fit model based on AIC with maumum number of lags = 20
results = model.fit(maxlags=20, ic='aic')

# Window size
window_size = results.k_ar

# Testing data
history = data.values[train_size-window_size:]

y_pred = list()
y_test = data['price'].values[train_size:]

for t in range(0,len(y_test)):
    h = history[t:(t+window_size)]
    pred = results.forecast(h, 1)
    y_pred.append(pred[-1,-1])

y_pred = np.array(y_pred).reshape(-1,1)

y_real = unprocessed_data['price'].iloc[-len(y_test):].values.reshape(-1,1)
last_obs = unprocessed_data['price'].iloc[-len(y_test)-1]
y_forecast = DP.inverse_transfrom(y_pred,y_real,last_obs,scaler)

```

```python

def evaluate_results(real,pred):
    '''
    Evaluate forecasting results and save to csv:
        real: Real price
        pred: Prediction of price
    '''
    rmse = np.sqrt(mean_squared_error(real, pred))
    print('\nTest RMSE: ', rmse)
    mae = mean_absolute_error(real,pred)
    print('Test MAE: ', mae)
    mape = np.mean(np.abs((real-pred)/real))*100
    print('Test MAPE: ',mape,'%')
    dar = DAR(real,pred,last_obs)
    print('Test DAR: ',dar)

    forecast = pd.DataFrame(data = pred, columns=['pred'])
    forecast['real'] = pd.DataFrame(real)
    forecast.index = pd.to_datetime(unprocessed_data[-len(y_test):].index)
    forecast.to_csv('VAR_forecast_'+dataset+'.csv')

    # Plot forecast
    forecast.plot(figsize=(8,4))
    plt.scatter(forecast.index,forecast['pred'],marker='.')
    plt.scatter(forecast.index,forecast['real'],marker='.')
    plt.show()

evaluate_results(y_real,y_forecast)
```

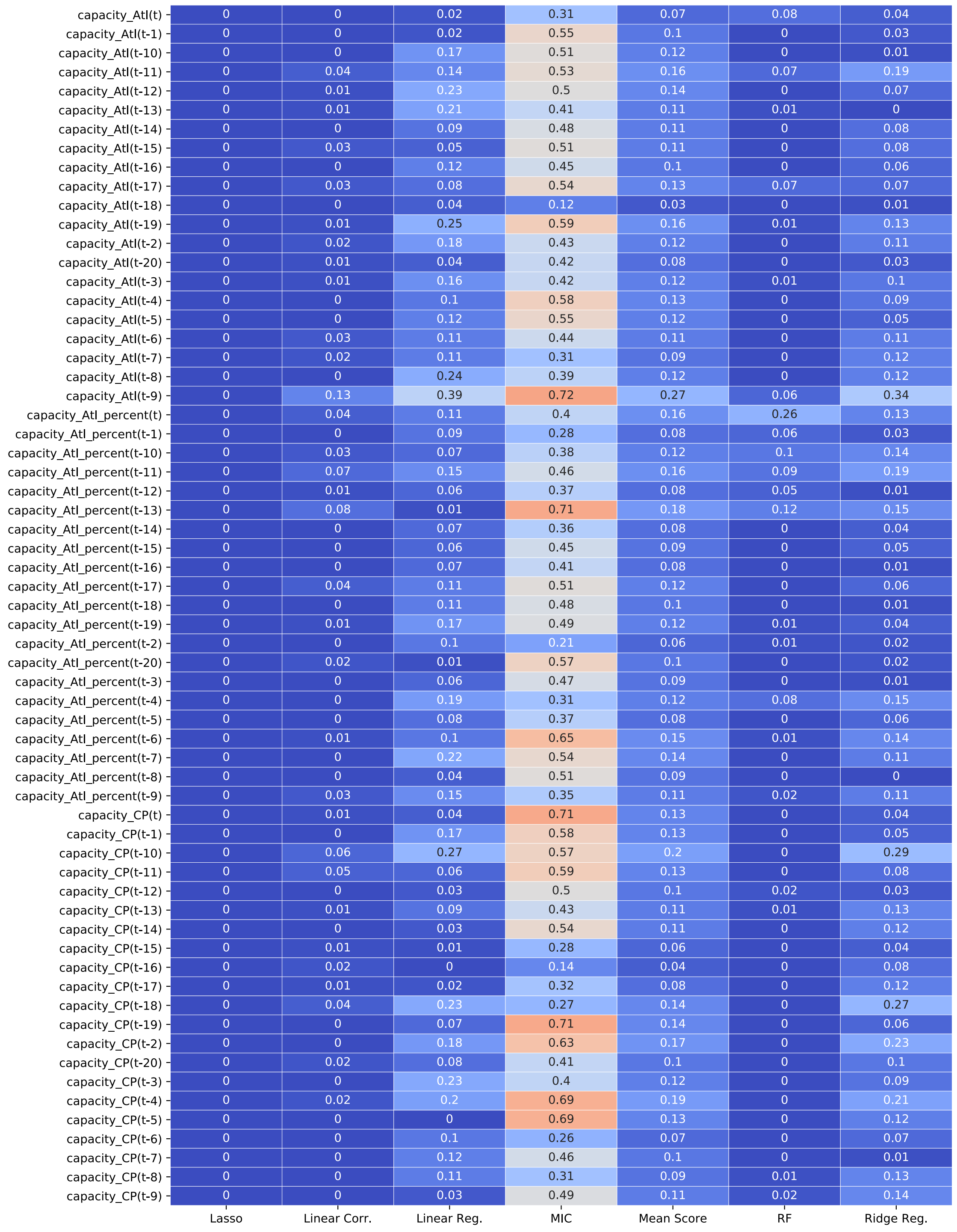
E.13 Model Evaluation (Model_evaluation.py)

Evaluation of LSTM, MLP and VAR models with performance metrics, a persistence model baseline and forecast visualization.

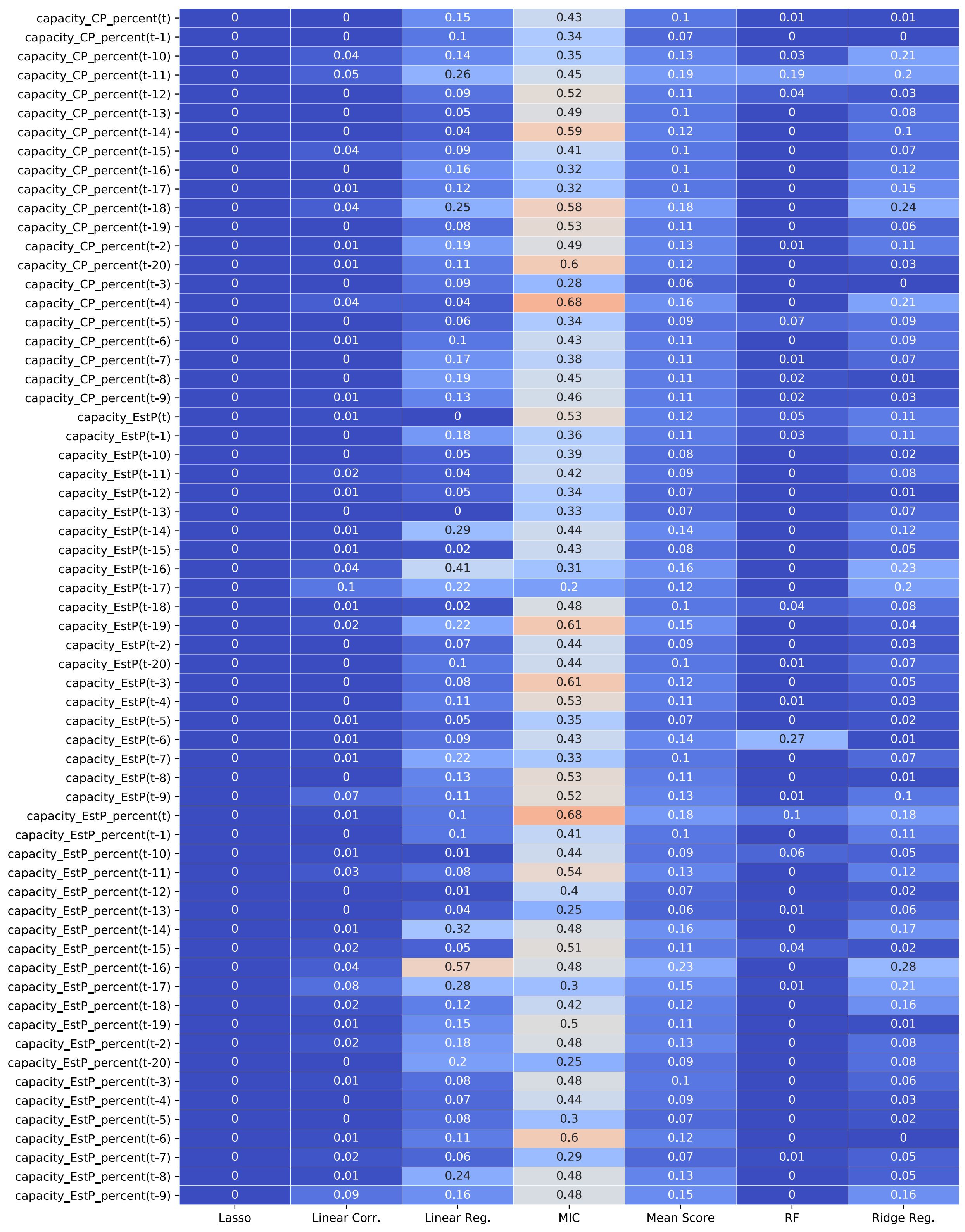
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jun  5 13:24:13 2018

@author: PatrickAndreNaess
"""
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error,mean_absolute_error

######################## DIRECTIONAL ACCURACY RATIO #########################

def DAR(real,pred,last_obs):
    '''
    Calculates the Directional Accuracy Ratio (DAR) of predictions:
        real: Real price
        pred: Prediction of price
        last_obs: Last observed value in the training set
    returns: DAR
    '''
    d = list()

    if (pred[0]-last_obs)*(real[0]-last_obs) > 0:
        d.append(1)
    else:
        d.append(0)

    for i in range(1,len(pred)):
        if (pred[i]-real[i-1])*(real[i]-real[i-1]) > 0:
            d.append(1)
        else:
            d.append(0)

    return sum(d)/len(d)

if __name__ == '__main__':
    ########################### IMPORT FORECAST #############################

    MLP_AIS = pd.read_csv('MLP_forecast_AIS.csv').set_index('timestamp')
    LSTM_AIS = pd.read_csv('LSTM_forecast_AIS.csv').set_index('timestamp')
    VAR_AIS = pd.read_csv('VAR_forecast_AIS.csv').set_index('timestamp')
    MLP_nonAIS = pd.read_csv('MLP_forecast_nonAIS.csv').set_index('timestamp')
    LSTM_nonAIS = pd.read_csv('LSTM_forecast_nonAIS.csv').set_index('timestamp')
    VAR_nonAIS = pd.read_csv('VAR_forecast_nonAIS.csv').set_index('timestamp')

    price = pd.read_csv('timeseries_data.csv').set_index('timestamp')['price']

    MLP_AIS.index = pd.to_datetime(MLP_AIS.index)
    LSTM_AIS.index = pd.to_datetime(LSTM_AIS.index)
    VAR_AIS.index = pd.to_datetime(VAR_AIS.index)
    MLP_nonAIS.index = pd.to_datetime(MLP_nonAIS.index)
    LSTM_nonAIS.index = pd.to_datetime(LSTM_nonAIS.index)
    VAR_nonAIS.index = pd.to_datetime(VAR_nonAIS.index)

    price.index = pd.to_datetime(price.index)

    last_obs = price.iloc[-len(MLP_AIS)-1]

    forecast = pd.DataFrame()
    forecast['real'] = MLP_AIS['real']
    forecast['MLP with non-AIS'] = MLP_nonAIS['pred']
    forecast['MLP with AIS'] = MLP_AIS['pred']
    forecast['LSTM with non-AIS'] = LSTM_nonAIS['pred']
    forecast['LSTM with AIS'] = LSTM_AIS['pred']
    forecast['VAR with AIS'] = VAR_AIS['pred']
    forecast['VAR with non-AIS'] = VAR_nonAIS['pred']
    forecast['Persistence'] = price.shift(1).loc[LSTM_AIS.index]

    ################# CALCULATE AND PLOT METRICS AND FORECAST #################

    metrics = pd.DataFrame()

    sns.set_context("paper", font_scale=1.5)

    for name,col in forecast.drop('real',axis=1).iteritems():
        real = forecast['real']
        pred = forecast[name]

        pred.plot(figsize = (10,5),label = name,style='--',marker='o',markersize=4)
        real.plot(label = 'Real',marker='o',markersize=4)
        plt.legend()
        plt.xlabel('Time')
        plt.ylabel('Price [USD/mt]')
        plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Forecast_'+name+'.pdf',
                    bbox_inches='tight')
        plt.show()

        metrics.loc['RMSE',name] = np.sqrt(mean_squared_error(real,pred))
        metrics.loc['MAE',name] = mean_absolute_error(real,pred)
        metrics.loc['MAPE',name] = np.mean(np.abs((real-pred)/real))*100

        if name == 'Persistence':
            dar = 0.5
        else:
            dar = DAR(real,pred,last_obs)
        metrics.loc['DAR',name] = dar

    error_metrics = pd.DataFrame()

    for metr,row in metrics[:3].iterrows():
        error_metrics[metr] = row/row['Persistence']

    sns.set_context("paper", font_scale=1.9)

    for metr,col in error_metrics.iteritems():
        sort = col.sort_values(ascending=True)
        plt.subplots(figsize = (6,2.5))
        plt.barh(range(len(sort.index)), sort)
        plt.yticks(range(len(sort.index)), sort.index)
        plt.xlabel('Relative '+metr)
        plt.axvline(x=1,color='k',linestyle='--',linewidth=1.5)
        plt.savefig('/Users/PatrickAndreNaess/Desktop/PyPlots/Metirc_'+metr+'.pdf',
                        bbox_inches='tight')
        plt.show()

    sns.set_context("paper", font_scale=1.4)

    print('\nMetrics:\n')
    print(metrics,'\n')
    print(error_metrics)
```

Appendix F

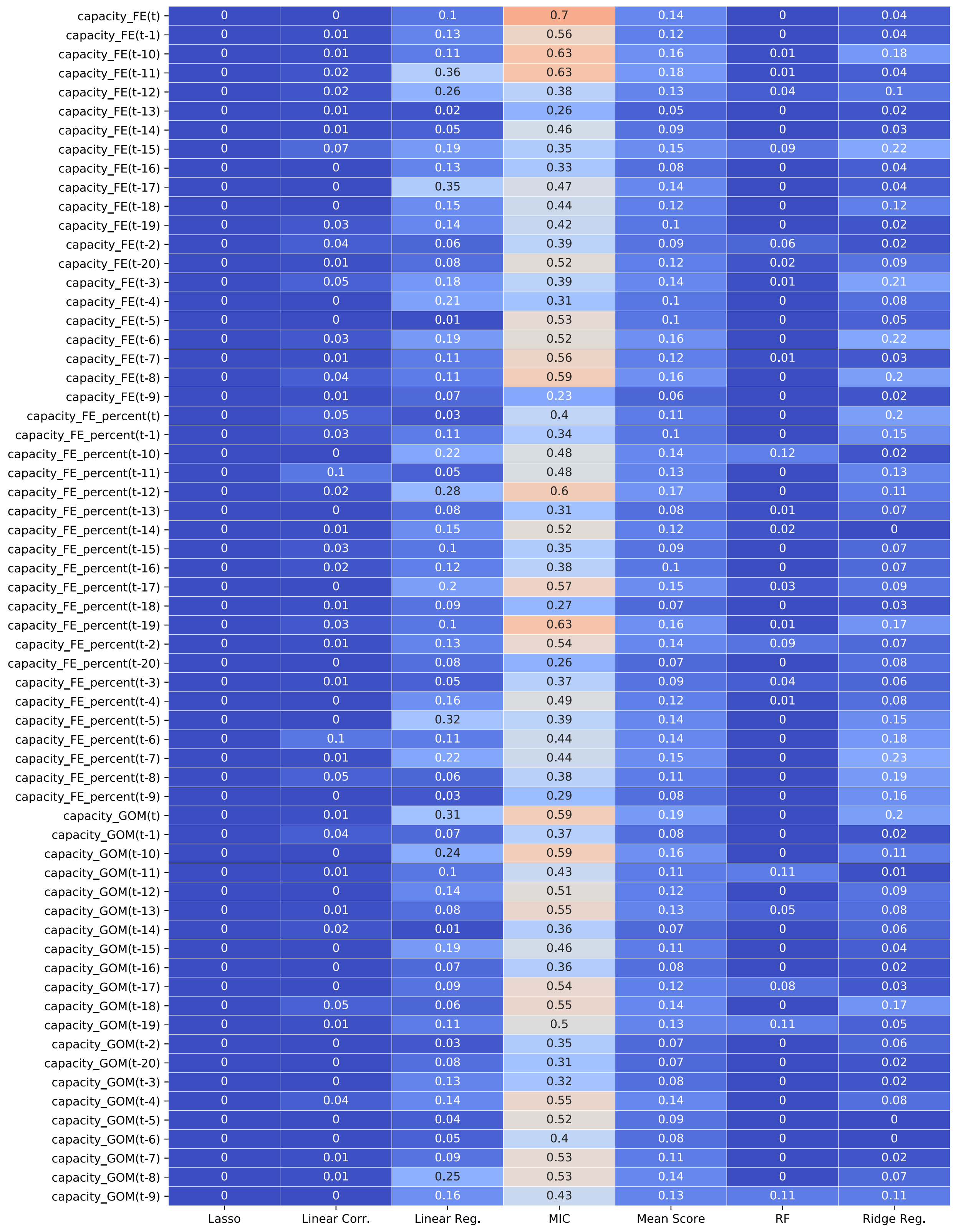
Features Importance Scores of Filter Selection Methods



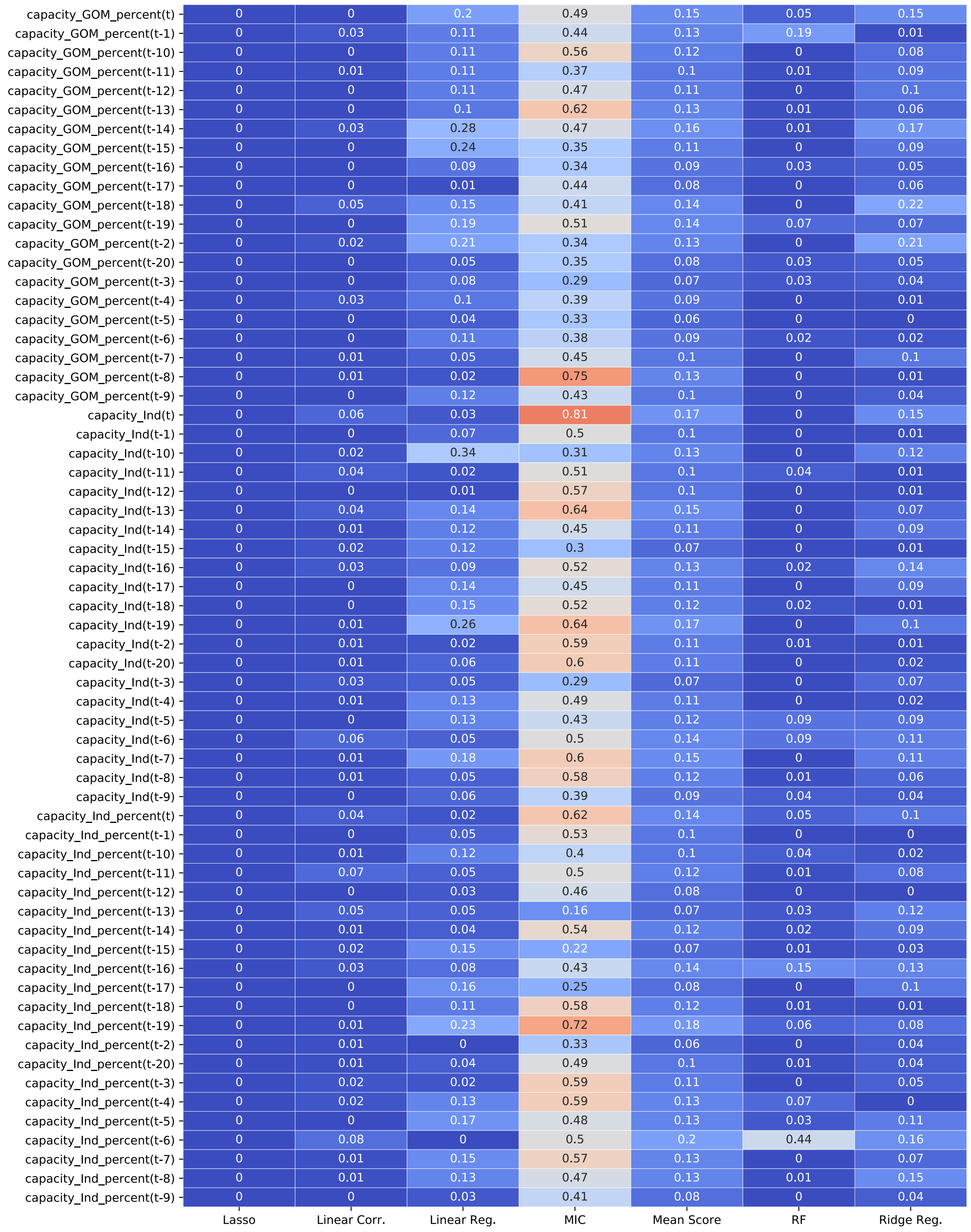
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
capacity_Atl(t)	0	0	0.02	0.31	0.07	0.08	0.04
capacity_Atl(t-1)	0	0	0.02	0.55	0.1	0	0.03
capacity_Atl(t-10)	0	0	0.17	0.51	0.12	0	0.01
capacity_Atl(t-11)	0	0.04	0.14	0.53	0.16	0.07	0.19
capacity_Atl(t-12)	0	0.01	0.23	0.5	0.14	0	0.07
capacity_Atl(t-13)	0	0.01	0.21	0.41	0.11	0.01	0
capacity_Atl(t-14)	0	0	0.09	0.48	0.11	0	0.08
capacity_Atl(t-15)	0	0.03	0.05	0.51	0.11	0	0.08
capacity_Atl(t-16)	0	0	0.12	0.45	0.1	0	0.06
capacity_Atl(t-17)	0	0.03	0.08	0.54	0.13	0.07	0.07
capacity_Atl(t-18)	0	0	0.04	0.12	0.03	0	0.01
capacity_Atl(t-19)	0	0.01	0.25	0.59	0.16	0.01	0.13
capacity_Atl(t-2)	0	0.02	0.18	0.43	0.12	0	0.11
capacity_Atl(t-20)	0	0.01	0.04	0.42	0.08	0	0.03
capacity_Atl(t-3)	0	0.01	0.16	0.42	0.12	0.01	0.1
capacity_Atl(t-4)	0	0	0.1	0.58	0.13	0	0.09
capacity_Atl(t-5)	0	0	0.12	0.55	0.12	0	0.05
capacity_Atl(t-6)	0	0.03	0.11	0.44	0.11	0	0.11
capacity_Atl(t-7)	0	0.02	0.11	0.31	0.09	0	0.12
capacity_Atl(t-8)	0	0	0.24	0.39	0.12	0	0.12
capacity_Atl(t-9)	0	0.13	0.39	0.72	0.27	0.06	0.34
capacity_Atl_percent(t)	0	0.04	0.11	0.4	0.16	0.26	0.13
capacity_Atl_percent(t-1)	0	0	0.09	0.28	0.08	0.06	0.03
capacity_Atl_percent(t-10)	0	0.03	0.07	0.38	0.12	0.1	0.14
capacity_Atl_percent(t-11)	0	0.07	0.15	0.46	0.16	0.09	0.19
capacity_Atl_percent(t-12)	0	0.01	0.06	0.37	0.08	0.05	0.01
capacity_Atl_percent(t-13)	0	0.08	0.01	0.71	0.18	0.12	0.15
capacity_Atl_percent(t-14)	0	0	0.07	0.36	0.08	0	0.04
capacity_Atl_percent(t-15)	0	0	0.06	0.45	0.09	0	0.05
capacity_Atl_percent(t-16)	0	0	0.07	0.41	0.08	0	0.01
capacity_Atl_percent(t-17)	0	0.04	0.11	0.51	0.12	0	0.06
capacity_Atl_percent(t-18)	0	0	0.11	0.48	0.1	0	0.01
capacity_Atl_percent(t-19)	0	0.01	0.17	0.49	0.12	0.01	0.04
capacity_Atl_percent(t-2)	0	0	0.1	0.21	0.06	0.01	0.02
capacity_Atl_percent(t-20)	0	0.02	0.01	0.57	0.1	0	0.02
capacity_Atl_percent(t-3)	0	0	0.06	0.47	0.09	0	0.01
capacity_Atl_percent(t-4)	0	0	0.19	0.31	0.12	0.08	0.15
capacity_Atl_percent(t-5)	0	0	0.08	0.37	0.08	0	0.06
capacity_Atl_percent(t-6)	0	0.01	0.1	0.65	0.15	0.01	0.14
capacity_Atl_percent(t-7)	0	0	0.22	0.54	0.14	0	0.11
capacity_Atl_percent(t-8)	0	0	0.04	0.51	0.09	0	0
capacity_Atl_percent(t-9)	0	0.03	0.15	0.35	0.11	0.02	0.11
capacity_CP(t)	0	0.01	0.04	0.71	0.13	0	0.04
capacity_CP(t-1)	0	0	0.17	0.58	0.13	0	0.05
capacity_CP(t-10)	0	0.06	0.27	0.57	0.2	0	0.29
capacity_CP(t-11)	0	0.05	0.06	0.59	0.13	0	0.08
capacity_CP(t-12)	0	0	0.03	0.5	0.1	0.02	0.03
capacity_CP(t-13)	0	0.01	0.09	0.43	0.11	0.01	0.13
capacity_CP(t-14)	0	0	0.03	0.54	0.11	0	0.12
capacity_CP(t-15)	0	0.01	0.01	0.28	0.06	0	0.04
capacity_CP(t-16)	0	0.02	0	0.14	0.04	0	0.08
capacity_CP(t-17)	0	0.01	0.02	0.32	0.08	0	0.12
capacity_CP(t-18)	0	0.04	0.23	0.27	0.14	0	0.27
capacity_CP(t-19)	0	0	0.07	0.71	0.14	0	0.06
capacity_CP(t-2)	0	0	0.18	0.63	0.17	0	0.23
capacity_CP(t-20)	0	0.02	0.08	0.41	0.1	0	0.1
capacity_CP(t-3)	0	0	0.23	0.4	0.12	0	0.09
capacity_CP(t-4)	0	0.02	0.2	0.69	0.19	0	0.21
capacity_CP(t-5)	0	0	0	0.69	0.13	0	0.12
capacity_CP(t-6)	0	0	0.1	0.26	0.07	0	0.07
capacity_CP(t-7)	0	0	0.12	0.46	0.1	0	0.01
capacity_CP(t-8)	0	0	0.11	0.31	0.09	0.01	0.13
capacity_CP(t-9)	0	0	0.03	0.49	0.11	0.02	0.14



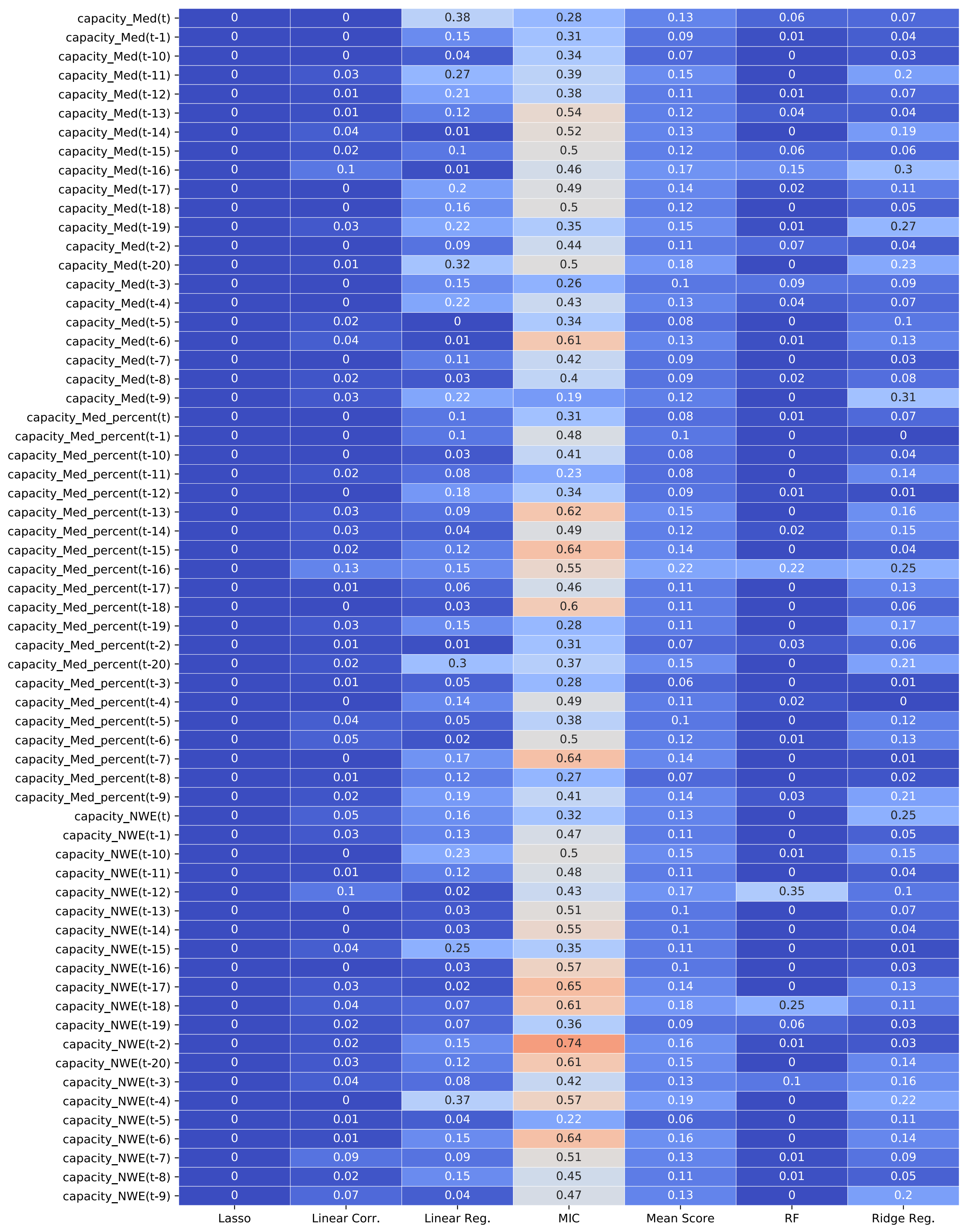
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
capacity_CP_percent(t)	0	0	0.15	0.43	0.1	0.01	0.01
capacity_CP_percent(t-1)	0	0	0.1	0.34	0.07	0	0
capacity_CP_percent(t-10)	0	0.04	0.14	0.35	0.13	0.03	0.21
capacity_CP_percent(t-11)	0	0.05	0.26	0.45	0.19	0.19	0.2
capacity_CP_percent(t-12)	0	0	0.09	0.52	0.11	0.04	0.03
capacity_CP_percent(t-13)	0	0	0.05	0.49	0.1	0	0.08
capacity_CP_percent(t-14)	0	0	0.04	0.59	0.12	0	0.1
capacity_CP_percent(t-15)	0	0.04	0.09	0.41	0.1	0	0.07
capacity_CP_percent(t-16)	0	0	0.16	0.32	0.1	0	0.12
capacity_CP_percent(t-17)	0	0.01	0.12	0.32	0.1	0	0.15
capacity_CP_percent(t-18)	0	0.04	0.25	0.58	0.18	0	0.24
capacity_CP_percent(t-19)	0	0	0.08	0.53	0.11	0	0.06
capacity_CP_percent(t-2)	0	0.01	0.19	0.49	0.13	0.01	0.11
capacity_CP_percent(t-20)	0	0.01	0.11	0.6	0.12	0	0.03
capacity_CP_percent(t-3)	0	0	0.09	0.28	0.06	0	0
capacity_CP_percent(t-4)	0	0.04	0.04	0.68	0.16	0	0.21
capacity_CP_percent(t-5)	0	0	0.06	0.34	0.09	0.07	0.09
capacity_CP_percent(t-6)	0	0.01	0.1	0.43	0.11	0	0.09
capacity_CP_percent(t-7)	0	0	0.17	0.38	0.11	0.01	0.07
capacity_CP_percent(t-8)	0	0	0.19	0.45	0.11	0.02	0.01
capacity_CP_percent(t-9)	0	0.01	0.13	0.46	0.11	0.02	0.03
capacity_EstP(t)	0	0.01	0	0.53	0.12	0.05	0.11
capacity_EstP(t-1)	0	0	0.18	0.36	0.11	0.03	0.11
capacity_EstP(t-10)	0	0	0.05	0.39	0.08	0	0.02
capacity_EstP(t-11)	0	0.02	0.04	0.42	0.09	0	0.08
capacity_EstP(t-12)	0	0.01	0.05	0.34	0.07	0	0.01
capacity_EstP(t-13)	0	0	0	0.33	0.07	0	0.07
capacity_EstP(t-14)	0	0.01	0.29	0.44	0.14	0	0.12
capacity_EstP(t-15)	0	0.01	0.02	0.43	0.08	0	0.05
capacity_EstP(t-16)	0	0.04	0.41	0.31	0.16	0	0.23
capacity_EstP(t-17)	0	0.1	0.22	0.2	0.12	0	0.2
capacity_EstP(t-18)	0	0.01	0.02	0.48	0.1	0.04	0.08
capacity_EstP(t-19)	0	0.02	0.22	0.61	0.15	0	0.04
capacity_EstP(t-2)	0	0	0.07	0.44	0.09	0	0.03
capacity_EstP(t-20)	0	0	0.1	0.44	0.1	0.01	0.07
capacity_EstP(t-3)	0	0	0.08	0.61	0.12	0	0.05
capacity_EstP(t-4)	0	0	0.11	0.53	0.11	0.01	0.03
capacity_EstP(t-5)	0	0.01	0.05	0.35	0.07	0	0.02
capacity_EstP(t-6)	0	0.01	0.09	0.43	0.14	0.27	0.01
capacity_EstP(t-7)	0	0.01	0.22	0.33	0.1	0	0.07
capacity_EstP(t-8)	0	0	0.13	0.53	0.11	0	0.01
capacity_EstP(t-9)	0	0.07	0.11	0.52	0.13	0.01	0.1
capacity_EstP_percent(t)	0	0.01	0.1	0.68	0.18	0.1	0.18
capacity_EstP_percent(t-1)	0	0	0.1	0.41	0.1	0	0.11
capacity_EstP_percent(t-10)	0	0.01	0.01	0.44	0.09	0.06	0.05
capacity_EstP_percent(t-11)	0	0.03	0.08	0.54	0.13	0	0.12
capacity_EstP_percent(t-12)	0	0	0.01	0.4	0.07	0	0.02
capacity_EstP_percent(t-13)	0	0	0.04	0.25	0.06	0.01	0.06
capacity_EstP_percent(t-14)	0	0.01	0.32	0.48	0.16	0	0.17
capacity_EstP_percent(t-15)	0	0.02	0.05	0.51	0.11	0.04	0.02
capacity_EstP_percent(t-16)	0	0.04	0.57	0.48	0.23	0	0.28
capacity_EstP_percent(t-17)	0	0.08	0.28	0.3	0.15	0.01	0.21
capacity_EstP_percent(t-18)	0	0.02	0.12	0.42	0.12	0	0.16
capacity_EstP_percent(t-19)	0	0.01	0.15	0.5	0.11	0	0.01
capacity_EstP_percent(t-2)	0	0.02	0.18	0.48	0.13	0	0.08
capacity_EstP_percent(t-20)	0	0	0.2	0.25	0.09	0	0.08
capacity_EstP_percent(t-3)	0	0.01	0.08	0.48	0.1	0	0.06
capacity_EstP_percent(t-4)	0	0	0.07	0.44	0.09	0	0.03
capacity_EstP_percent(t-5)	0	0	0.08	0.3	0.07	0	0.02
capacity_EstP_percent(t-6)	0	0.01	0.11	0.6	0.12	0	0
capacity_EstP_percent(t-7)	0	0.02	0.06	0.29	0.07	0.01	0.05
capacity_EstP_percent(t-8)	0	0.01	0.24	0.48	0.13	0	0.05
capacity_EstP_percent(t-9)	0	0.09	0.16	0.48	0.15	0	0.16



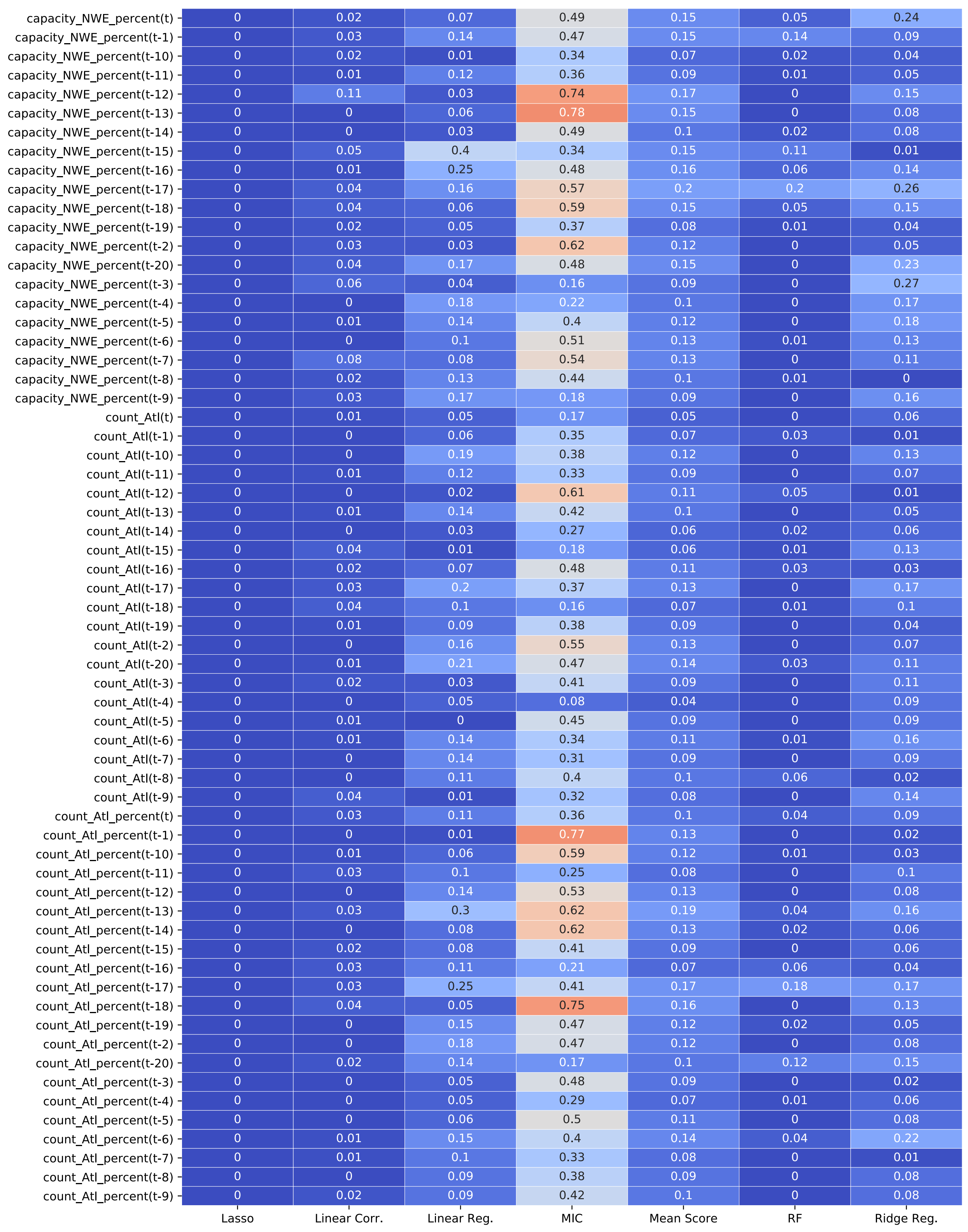
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
capacity_FE(t)	0	0	0.1	0.7	0.14	0	0.04
capacity_FE(t-1)	0	0.01	0.13	0.56	0.12	0	0.04
capacity_FE(t-10)	0	0.01	0.11	0.63	0.16	0.01	0.18
capacity_FE(t-11)	0	0.02	0.36	0.63	0.18	0.01	0.04
capacity_FE(t-12)	0	0.02	0.26	0.38	0.13	0.04	0.1
capacity_FE(t-13)	0	0.01	0.02	0.26	0.05	0	0.02
capacity_FE(t-14)	0	0.01	0.05	0.46	0.09	0	0.03
capacity_FE(t-15)	0	0.07	0.19	0.35	0.15	0.09	0.22
capacity_FE(t-16)	0	0	0.13	0.33	0.08	0	0.04
capacity_FE(t-17)	0	0	0.35	0.47	0.14	0	0.04
capacity_FE(t-18)	0	0	0.15	0.44	0.12	0	0.12
capacity_FE(t-19)	0	0.03	0.14	0.42	0.1	0	0.02
capacity_FE(t-2)	0	0.04	0.06	0.39	0.09	0.06	0.02
capacity_FE(t-20)	0	0.01	0.08	0.52	0.12	0.02	0.09
capacity_FE(t-3)	0	0.05	0.18	0.39	0.14	0.01	0.21
capacity_FE(t-4)	0	0	0.21	0.31	0.1	0	0.08
capacity_FE(t-5)	0	0	0.01	0.53	0.1	0	0.05
capacity_FE(t-6)	0	0.03	0.19	0.52	0.16	0	0.22
capacity_FE(t-7)	0	0.01	0.11	0.56	0.12	0.01	0.03
capacity_FE(t-8)	0	0.04	0.11	0.59	0.16	0	0.2
capacity_FE(t-9)	0	0.01	0.07	0.23	0.06	0	0.02
capacity_FE_percent(t)	0	0.05	0.03	0.4	0.11	0	0.2
capacity_FE_percent(t-1)	0	0.03	0.11	0.34	0.1	0	0.15
capacity_FE_percent(t-10)	0	0	0.22	0.48	0.14	0.12	0.02
capacity_FE_percent(t-11)	0	0.1	0.05	0.48	0.13	0	0.13
capacity_FE_percent(t-12)	0	0.02	0.28	0.6	0.17	0	0.11
capacity_FE_percent(t-13)	0	0	0.08	0.31	0.08	0.01	0.07
capacity_FE_percent(t-14)	0	0.01	0.15	0.52	0.12	0.02	0
capacity_FE_percent(t-15)	0	0.03	0.1	0.35	0.09	0	0.07
capacity_FE_percent(t-16)	0	0.02	0.12	0.38	0.1	0	0.07
capacity_FE_percent(t-17)	0	0	0.2	0.57	0.15	0.03	0.09
capacity_FE_percent(t-18)	0	0.01	0.09	0.27	0.07	0	0.03
capacity_FE_percent(t-19)	0	0.03	0.1	0.63	0.16	0.01	0.17
capacity_FE_percent(t-2)	0	0.01	0.13	0.54	0.14	0.09	0.07
capacity_FE_percent(t-20)	0	0	0.08	0.26	0.07	0	0.08
capacity_FE_percent(t-3)	0	0.01	0.05	0.37	0.09	0.04	0.06
capacity_FE_percent(t-4)	0	0	0.16	0.49	0.12	0.01	0.08
capacity_FE_percent(t-5)	0	0	0.32	0.39	0.14	0	0.15
capacity_FE_percent(t-6)	0	0.1	0.11	0.44	0.14	0	0.18
capacity_FE_percent(t-7)	0	0.01	0.22	0.44	0.15	0	0.23
capacity_FE_percent(t-8)	0	0.05	0.06	0.38	0.11	0	0.19
capacity_FE_percent(t-9)	0	0	0.03	0.29	0.08	0	0.16
capacity_GOM(t)	0	0.01	0.31	0.59	0.19	0	0.2
capacity_GOM(t-1)	0	0.04	0.07	0.37	0.08	0	0.02
capacity_GOM(t-10)	0	0	0.24	0.59	0.16	0	0.11
capacity_GOM(t-11)	0	0.01	0.1	0.43	0.11	0.11	0.01
capacity_GOM(t-12)	0	0	0.14	0.51	0.12	0	0.09
capacity_GOM(t-13)	0	0.01	0.08	0.55	0.13	0.05	0.08
capacity_GOM(t-14)	0	0.02	0.01	0.36	0.07	0	0.06
capacity_GOM(t-15)	0	0	0.19	0.46	0.11	0	0.04
capacity_GOM(t-16)	0	0	0.07	0.36	0.08	0	0.02
capacity_GOM(t-17)	0	0	0.09	0.54	0.12	0.08	0.03
capacity_GOM(t-18)	0	0.05	0.06	0.55	0.14	0	0.17
capacity_GOM(t-19)	0	0.01	0.11	0.5	0.13	0.11	0.05
capacity_GOM(t-2)	0	0	0.03	0.35	0.07	0	0.06
capacity_GOM(t-20)	0	0	0.08	0.31	0.07	0	0.02
capacity_GOM(t-3)	0	0	0.13	0.32	0.08	0	0.02
capacity_GOM(t-4)	0	0.04	0.14	0.55	0.14	0	0.08
capacity_GOM(t-5)	0	0	0.04	0.52	0.09	0	0
capacity_GOM(t-6)	0	0	0.05	0.4	0.08	0	0
capacity_GOM(t-7)	0	0.01	0.09	0.53	0.11	0	0.02
capacity_GOM(t-8)	0	0.01	0.25	0.53	0.14	0	0.07
capacity_GOM(t-9)	0	0	0.16	0.43	0.13	0.11	0.11



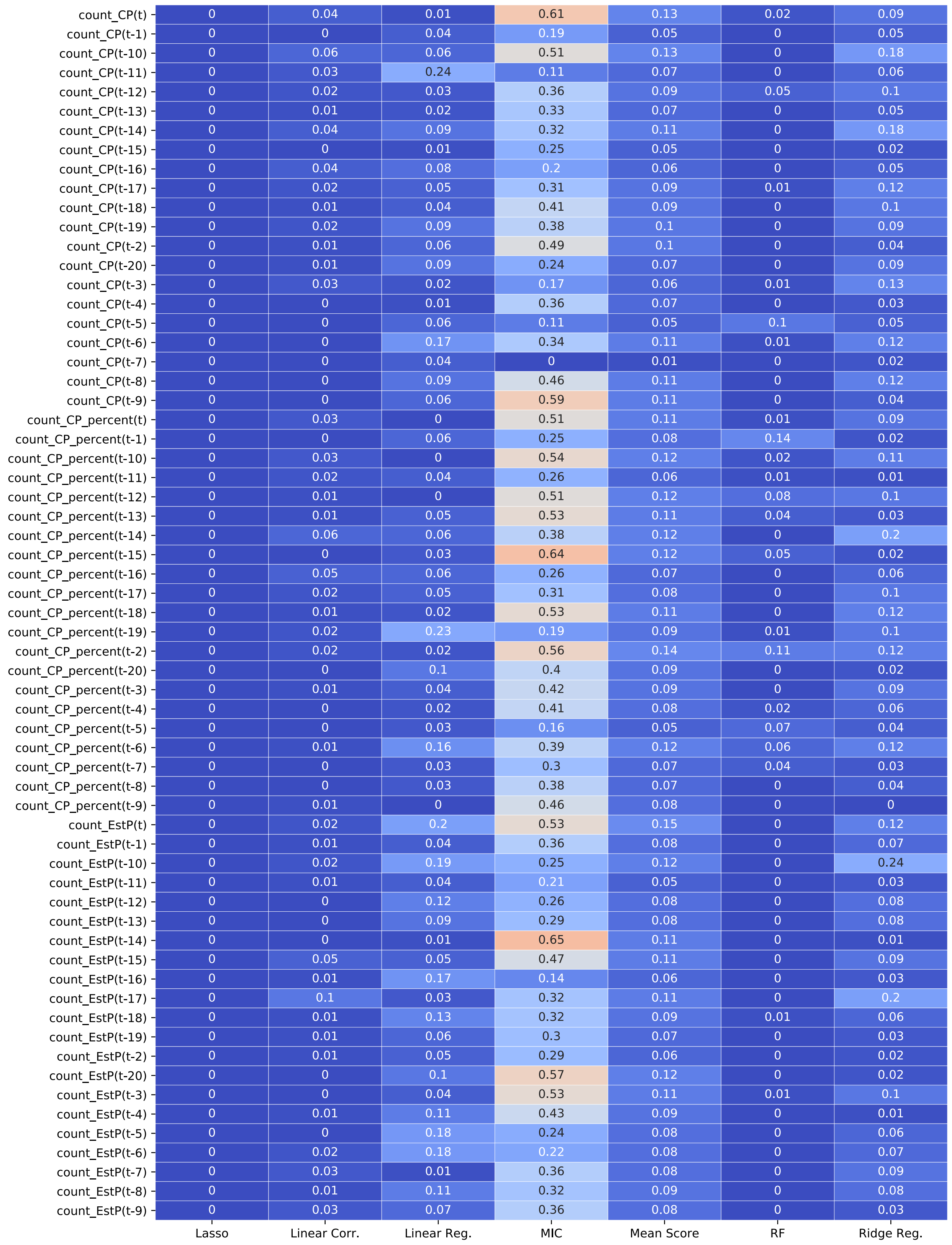
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
capacity_GOM_percent(t)	0	0	0.2	0.49	0.15	0.05	0.15
capacity_GOM_percent(t-1)	0	0.03	0.11	0.44	0.13	0.19	0.01
capacity_GOM_percent(t-10)	0	0	0.11	0.56	0.12	0	0.08
capacity_GOM_percent(t-11)	0	0.01	0.11	0.37	0.1	0.01	0.09
capacity_GOM_percent(t-12)	0	0	0.11	0.47	0.11	0	0.1
capacity_GOM_percent(t-13)	0	0	0.1	0.62	0.13	0.01	0.06
capacity_GOM_percent(t-14)	0	0.03	0.28	0.47	0.16	0.01	0.17
capacity_GOM_percent(t-15)	0	0	0.24	0.35	0.11	0	0.09
capacity_GOM_percent(t-16)	0	0	0.09	0.34	0.09	0.03	0.05
capacity_GOM_percent(t-17)	0	0	0.01	0.44	0.08	0	0.06
capacity_GOM_percent(t-18)	0	0.05	0.15	0.41	0.14	0	0.22
capacity_GOM_percent(t-19)	0	0	0.19	0.51	0.14	0.07	0.07
capacity_GOM_percent(t-2)	0	0.02	0.21	0.34	0.13	0	0.21
capacity_GOM_percent(t-20)	0	0	0.05	0.35	0.08	0.03	0.05
capacity_GOM_percent(t-3)	0	0	0.08	0.29	0.07	0.03	0.04
capacity_GOM_percent(t-4)	0	0.03	0.1	0.39	0.09	0	0.01
capacity_GOM_percent(t-5)	0	0	0.04	0.33	0.06	0	0
capacity_GOM_percent(t-6)	0	0	0.11	0.38	0.09	0.02	0.02
capacity_GOM_percent(t-7)	0	0.01	0.05	0.45	0.1	0	0.1
capacity_GOM_percent(t-8)	0	0.01	0.02	0.75	0.13	0	0.01
capacity_GOM_percent(t-9)	0	0	0.12	0.43	0.1	0	0.04
capacity_Ind(t)	0	0.06	0.03	0.81	0.17	0	0.15
capacity_Ind(t-1)	0	0	0.07	0.5	0.1	0	0.01
capacity_Ind(t-10)	0	0.02	0.34	0.31	0.13	0	0.12
capacity_Ind(t-11)	0	0.04	0.02	0.51	0.1	0.04	0.01
capacity_Ind(t-12)	0	0	0.01	0.57	0.1	0	0.01
capacity_Ind(t-13)	0	0.04	0.14	0.64	0.15	0	0.07
capacity_Ind(t-14)	0	0.01	0.12	0.45	0.11	0	0.09
capacity_Ind(t-15)	0	0.02	0.12	0.3	0.07	0	0.01
capacity_Ind(t-16)	0	0.03	0.09	0.52	0.13	0.02	0.14
capacity_Ind(t-17)	0	0	0.14	0.45	0.11	0	0.09
capacity_Ind(t-18)	0	0	0.15	0.52	0.12	0.02	0.01
capacity_Ind(t-19)	0	0.01	0.26	0.64	0.17	0	0.1
capacity_Ind(t-2)	0	0.01	0.02	0.59	0.11	0.01	0.01
capacity_Ind(t-20)	0	0.01	0.06	0.6	0.11	0	0.02
capacity_Ind(t-3)	0	0.03	0.05	0.29	0.07	0	0.07
capacity_Ind(t-4)	0	0.01	0.13	0.49	0.11	0	0.02
capacity_Ind(t-5)	0	0	0.13	0.43	0.12	0.09	0.09
capacity_Ind(t-6)	0	0.06	0.05	0.5	0.14	0.09	0.11
capacity_Ind(t-7)	0	0.01	0.18	0.6	0.15	0	0.11
capacity_Ind(t-8)	0	0.01	0.05	0.58	0.12	0.01	0.06
capacity_Ind(t-9)	0	0	0.06	0.39	0.09	0.04	0.04
capacity_Ind_percent(t)	0	0.04	0.02	0.62	0.14	0.05	0.1
capacity_Ind_percent(t-1)	0	0	0.05	0.53	0.1	0	0
capacity_Ind_percent(t-10)	0	0.01	0.12	0.4	0.1	0.04	0.02
capacity_Ind_percent(t-11)	0	0.07	0.05	0.5	0.12	0.01	0.08
capacity_Ind_percent(t-12)	0	0	0.03	0.46	0.08	0	0
capacity_Ind_percent(t-13)	0	0.05	0.05	0.16	0.07	0.03	0.12
capacity_Ind_percent(t-14)	0	0.01	0.04	0.54	0.12	0.02	0.09
capacity_Ind_percent(t-15)	0	0.02	0.15	0.22	0.07	0.01	0.03
capacity_Ind_percent(t-16)	0	0.03	0.08	0.43	0.14	0.15	0.13
capacity_Ind_percent(t-17)	0	0	0.16	0.25	0.08	0	0.1
capacity_Ind_percent(t-18)	0	0	0.11	0.58	0.12	0.01	0.01
capacity_Ind_percent(t-19)	0	0.01	0.23	0.72	0.18	0.06	0.08
capacity_Ind_percent(t-2)	0	0.01	0	0.33	0.06	0	0.04
capacity_Ind_percent(t-20)	0	0.01	0.04	0.49	0.1	0.01	0.04
capacity_Ind_percent(t-3)	0	0.02	0.02	0.59	0.11	0	0.05
capacity_Ind_percent(t-4)	0	0.02	0.13	0.59	0.13	0.07	0
capacity_Ind_percent(t-5)	0	0	0.17	0.48	0.13	0.03	0.11
capacity_Ind_percent(t-6)	0	0.08	0	0.5	0.2	0.44	0.16
capacity_Ind_percent(t-7)	0	0.01	0.15	0.57	0.13	0	0.07
capacity_Ind_percent(t-8)	0	0.01	0.13	0.47	0.13	0.01	0.15
capacity_Ind_percent(t-9)	0	0	0.03	0.41	0.08	0	0.04



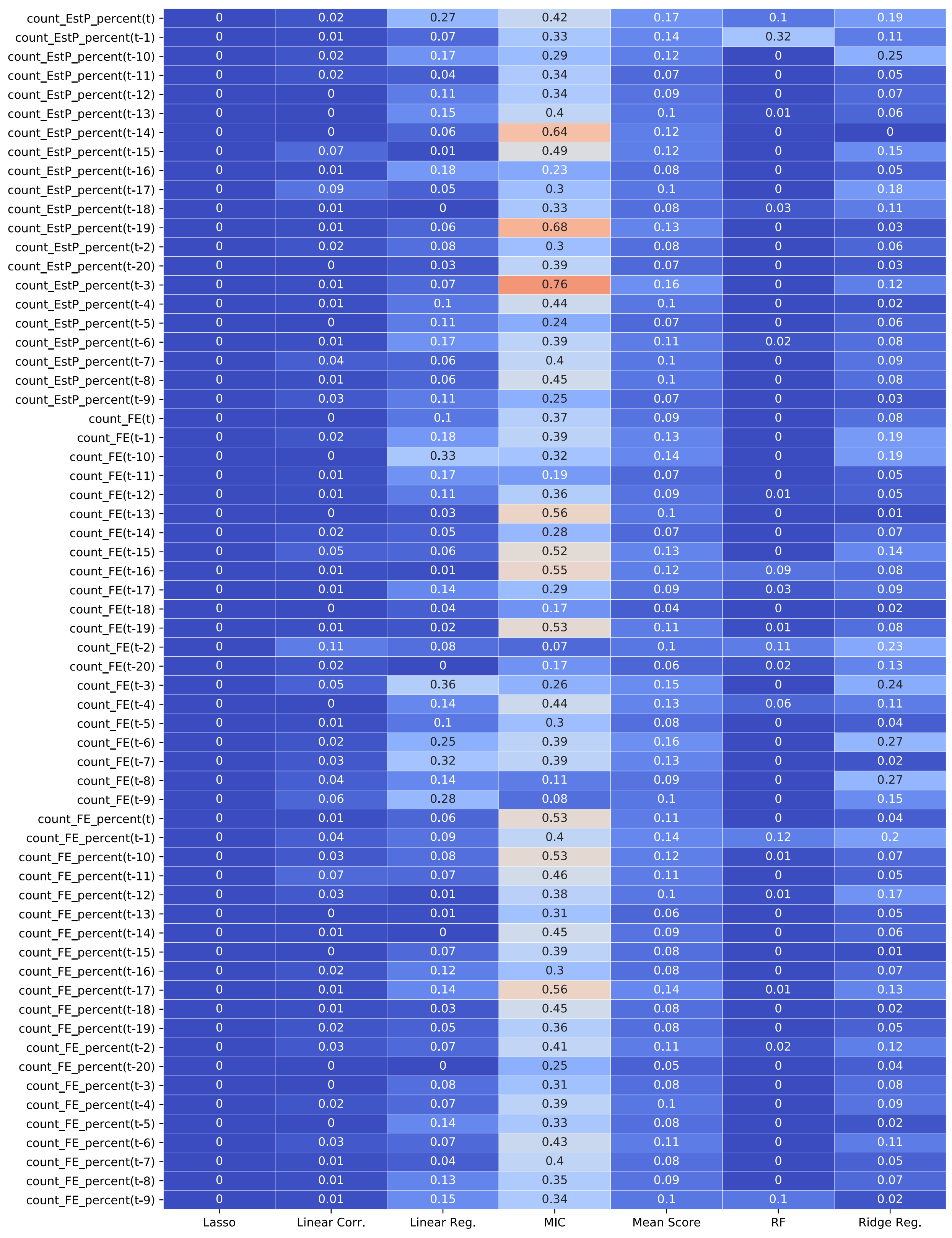
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
capacity_Med(t)	0	0	0.38	0.28	0.13	0.06	0.07
capacity_Med(t-1)	0	0	0.15	0.31	0.09	0.01	0.04
capacity_Med(t-10)	0	0	0.04	0.34	0.07	0	0.03
capacity_Med(t-11)	0	0.03	0.27	0.39	0.15	0	0.2
capacity_Med(t-12)	0	0.01	0.21	0.38	0.11	0.01	0.07
capacity_Med(t-13)	0	0.01	0.12	0.54	0.12	0.04	0.04
capacity_Med(t-14)	0	0.04	0.01	0.52	0.13	0	0.19
capacity_Med(t-15)	0	0.02	0.1	0.5	0.12	0.06	0.06
capacity_Med(t-16)	0	0.1	0.01	0.46	0.17	0.15	0.3
capacity_Med(t-17)	0	0	0.2	0.49	0.14	0.02	0.11
capacity_Med(t-18)	0	0	0.16	0.5	0.12	0	0.05
capacity_Med(t-19)	0	0.03	0.22	0.35	0.15	0.01	0.27
capacity_Med(t-2)	0	0	0.09	0.44	0.11	0.07	0.04
capacity_Med(t-20)	0	0.01	0.32	0.5	0.18	0	0.23
capacity_Med(t-3)	0	0	0.15	0.26	0.1	0.09	0.09
capacity_Med(t-4)	0	0	0.22	0.43	0.13	0.04	0.07
capacity_Med(t-5)	0	0.02	0	0.34	0.08	0	0.1
capacity_Med(t-6)	0	0.04	0.01	0.61	0.13	0.01	0.13
capacity_Med(t-7)	0	0	0.11	0.42	0.09	0	0.03
capacity_Med(t-8)	0	0.02	0.03	0.4	0.09	0.02	0.08
capacity_Med(t-9)	0	0.03	0.22	0.19	0.12	0	0.31
capacity_Med_percent(t)	0	0	0.1	0.31	0.08	0.01	0.07
capacity_Med_percent(t-1)	0	0	0.1	0.48	0.1	0	0
capacity_Med_percent(t-10)	0	0	0.03	0.41	0.08	0	0.04
capacity_Med_percent(t-11)	0	0.02	0.08	0.23	0.08	0	0.14
capacity_Med_percent(t-12)	0	0	0.18	0.34	0.09	0.01	0.01
capacity_Med_percent(t-13)	0	0.03	0.09	0.62	0.15	0	0.16
capacity_Med_percent(t-14)	0	0.03	0.04	0.49	0.12	0.02	0.15
capacity_Med_percent(t-15)	0	0.02	0.12	0.64	0.14	0	0.04
capacity_Med_percent(t-16)	0	0.13	0.15	0.55	0.22	0.22	0.25
capacity_Med_percent(t-17)	0	0.01	0.06	0.46	0.11	0	0.13
capacity_Med_percent(t-18)	0	0	0.03	0.6	0.11	0	0.06
capacity_Med_percent(t-19)	0	0.03	0.15	0.28	0.11	0	0.17
capacity_Med_percent(t-2)	0	0.01	0.01	0.31	0.07	0.03	0.06
capacity_Med_percent(t-20)	0	0.02	0.3	0.37	0.15	0	0.21
capacity_Med_percent(t-3)	0	0.01	0.05	0.28	0.06	0	0.01
capacity_Med_percent(t-4)	0	0	0.14	0.49	0.11	0.02	0
capacity_Med_percent(t-5)	0	0.04	0.05	0.38	0.1	0	0.12
capacity_Med_percent(t-6)	0	0.05	0.02	0.5	0.12	0.01	0.13
capacity_Med_percent(t-7)	0	0	0.17	0.64	0.14	0	0.01
capacity_Med_percent(t-8)	0	0.01	0.12	0.27	0.07	0	0.02
capacity_Med_percent(t-9)	0	0.02	0.19	0.41	0.14	0.03	0.21
capacity_NWE(t)	0	0.05	0.16	0.32	0.13	0	0.25
capacity_NWE(t-1)	0	0.03	0.13	0.47	0.11	0	0.05
capacity_NWE(t-10)	0	0	0.23	0.5	0.15	0.01	0.15
capacity_NWE(t-11)	0	0.01	0.12	0.48	0.11	0	0.04
capacity_NWE(t-12)	0	0.1	0.02	0.43	0.17	0.35	0.1
capacity_NWE(t-13)	0	0	0.03	0.51	0.1	0	0.07
capacity_NWE(t-14)	0	0	0.03	0.55	0.1	0	0.04
capacity_NWE(t-15)	0	0.04	0.25	0.35	0.11	0	0.01
capacity_NWE(t-16)	0	0	0.03	0.57	0.1	0	0.03
capacity_NWE(t-17)	0	0.03	0.02	0.65	0.14	0	0.13
capacity_NWE(t-18)	0	0.04	0.07	0.61	0.18	0.25	0.11
capacity_NWE(t-19)	0	0.02	0.07	0.36	0.09	0.06	0.03
capacity_NWE(t-2)	0	0.02	0.15	0.74	0.16	0.01	0.03
capacity_NWE(t-20)	0	0.03	0.12	0.61	0.15	0	0.14
capacity_NWE(t-3)	0	0.04	0.08	0.42	0.13	0.1	0.16
capacity_NWE(t-4)	0	0	0.37	0.57	0.19	0	0.22
capacity_NWE(t-5)	0	0.01	0.04	0.22	0.06	0	0.11
capacity_NWE(t-6)	0	0.01	0.15	0.64	0.16	0	0.14
capacity_NWE(t-7)	0	0.09	0.09	0.51	0.13	0.01	0.09
capacity_NWE(t-8)	0	0.02	0.15	0.45	0.11	0.01	0.05
capacity_NWE(t-9)	0	0.07	0.04	0.47	0.13	0	0.2



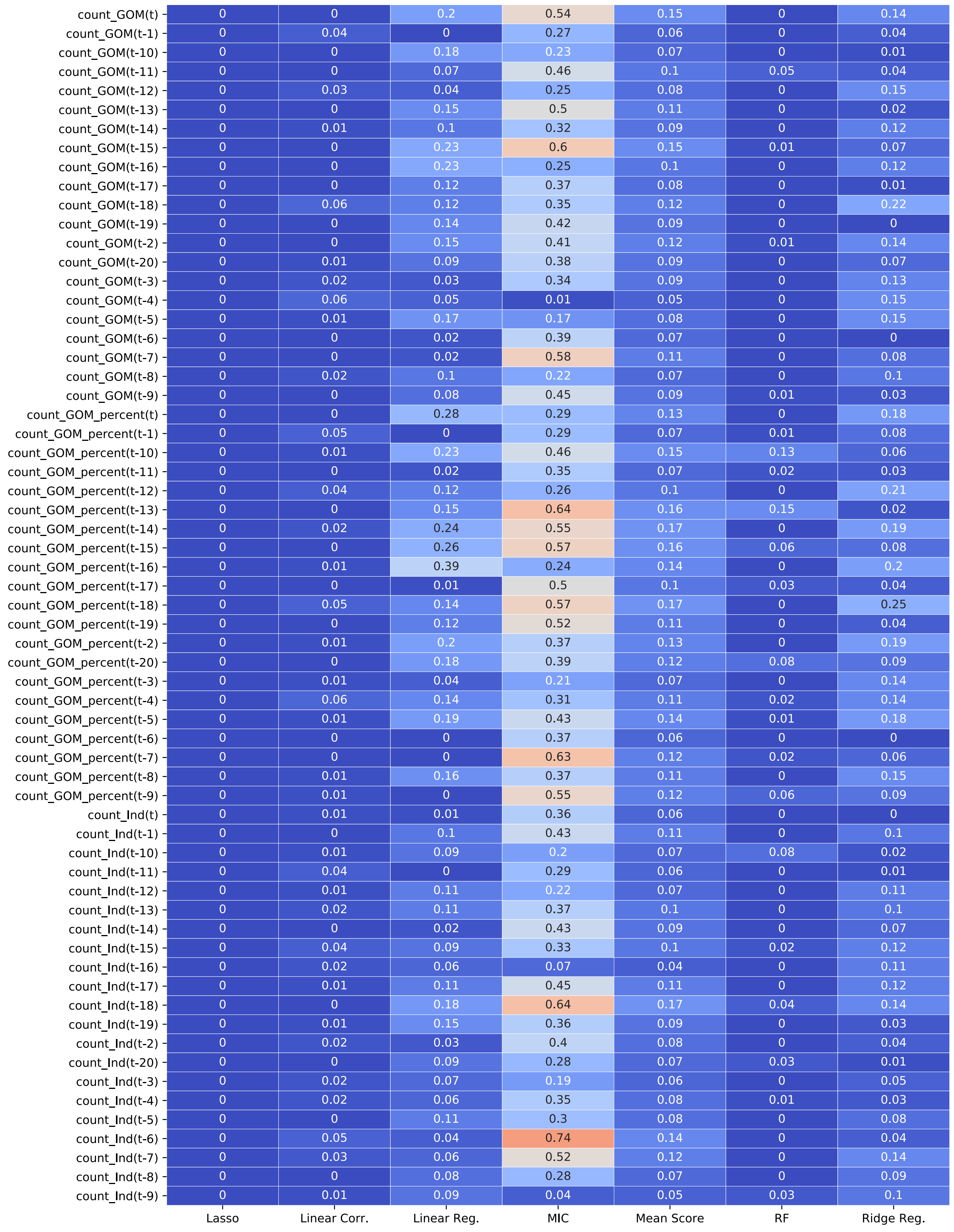
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
capacity_NWE_percent(t)	0	0.02	0.07	0.49	0.15	0.05	0.24
capacity_NWE_percent(t-1)	0	0.03	0.14	0.47	0.15	0.14	0.09
capacity_NWE_percent(t-10)	0	0.02	0.01	0.34	0.07	0.02	0.04
capacity_NWE_percent(t-11)	0	0.01	0.12	0.36	0.09	0.01	0.05
capacity_NWE_percent(t-12)	0	0.11	0.03	0.74	0.17	0	0.15
capacity_NWE_percent(t-13)	0	0	0.06	0.78	0.15	0	0.08
capacity_NWE_percent(t-14)	0	0	0.03	0.49	0.1	0.02	0.08
capacity_NWE_percent(t-15)	0	0.05	0.4	0.34	0.15	0.11	0.01
capacity_NWE_percent(t-16)	0	0.01	0.25	0.48	0.16	0.06	0.14
capacity_NWE_percent(t-17)	0	0.04	0.16	0.57	0.2	0.2	0.26
capacity_NWE_percent(t-18)	0	0.04	0.06	0.59	0.15	0.05	0.15
capacity_NWE_percent(t-19)	0	0.02	0.05	0.37	0.08	0.01	0.04
capacity_NWE_percent(t-2)	0	0.03	0.03	0.62	0.12	0	0.05
capacity_NWE_percent(t-20)	0	0.04	0.17	0.48	0.15	0	0.23
capacity_NWE_percent(t-3)	0	0.06	0.04	0.16	0.09	0	0.27
capacity_NWE_percent(t-4)	0	0	0.18	0.22	0.1	0	0.17
capacity_NWE_percent(t-5)	0	0.01	0.14	0.4	0.12	0	0.18
capacity_NWE_percent(t-6)	0	0	0.1	0.51	0.13	0.01	0.13
capacity_NWE_percent(t-7)	0	0.08	0.08	0.54	0.13	0	0.11
capacity_NWE_percent(t-8)	0	0.02	0.13	0.44	0.1	0.01	0
capacity_NWE_percent(t-9)	0	0.03	0.17	0.18	0.09	0	0.16
count_Atl(t)	0	0.01	0.05	0.17	0.05	0	0.06
count_Atl(t-1)	0	0	0.06	0.35	0.07	0.03	0.01
count_Atl(t-10)	0	0	0.19	0.38	0.12	0	0.13
count_Atl(t-11)	0	0.01	0.12	0.33	0.09	0	0.07
count_Atl(t-12)	0	0	0.02	0.61	0.11	0.05	0.01
count_Atl(t-13)	0	0.01	0.14	0.42	0.1	0	0.05
count_Atl(t-14)	0	0	0.03	0.27	0.06	0.02	0.06
count_Atl(t-15)	0	0.04	0.01	0.18	0.06	0.01	0.13
count_Atl(t-16)	0	0.02	0.07	0.48	0.11	0.03	0.03
count_Atl(t-17)	0	0.03	0.2	0.37	0.13	0	0.17
count_Atl(t-18)	0	0.04	0.1	0.16	0.07	0.01	0.1
count_Atl(t-19)	0	0.01	0.09	0.38	0.09	0	0.04
count_Atl(t-2)	0	0	0.16	0.55	0.13	0	0.07
count_Atl(t-20)	0	0.01	0.21	0.47	0.14	0.03	0.11
count_Atl(t-3)	0	0.02	0.03	0.41	0.09	0	0.11
count_Atl(t-4)	0	0	0.05	0.08	0.04	0	0.09
count_Atl(t-5)	0	0.01	0	0.45	0.09	0	0.09
count_Atl(t-6)	0	0.01	0.14	0.34	0.11	0.01	0.16
count_Atl(t-7)	0	0	0.14	0.31	0.09	0	0.09
count_Atl(t-8)	0	0	0.11	0.4	0.1	0.06	0.02
count_Atl(t-9)	0	0.04	0.01	0.32	0.08	0	0.14
count_Atl_percent(t)	0	0.03	0.11	0.36	0.1	0.04	0.09
count_Atl_percent(t-1)	0	0	0.01	0.77	0.13	0	0.02
count_Atl_percent(t-10)	0	0.01	0.06	0.59	0.12	0.01	0.03
count_Atl_percent(t-11)	0	0.03	0.1	0.25	0.08	0	0.1
count_Atl_percent(t-12)	0	0	0.14	0.53	0.13	0	0.08
count_Atl_percent(t-13)	0	0.03	0.3	0.62	0.19	0.04	0.16
count_Atl_percent(t-14)	0	0	0.08	0.62	0.13	0.02	0.06
count_Atl_percent(t-15)	0	0.02	0.08	0.41	0.09	0	0.06
count_Atl_percent(t-16)	0	0.03	0.11	0.21	0.07	0.06	0.04
count_Atl_percent(t-17)	0	0.03	0.25	0.41	0.17	0.18	0.17
count_Atl_percent(t-18)	0	0.04	0.05	0.75	0.16	0	0.13
count_Atl_percent(t-19)	0	0	0.15	0.47	0.12	0.02	0.05
count_Atl_percent(t-2)	0	0	0.18	0.47	0.12	0	0.08
count_Atl_percent(t-20)	0	0.02	0.14	0.17	0.1	0.12	0.15
count_Atl_percent(t-3)	0	0	0.05	0.48	0.09	0	0.02
count_Atl_percent(t-4)	0	0	0.05	0.29	0.07	0.01	0.06
count_Atl_percent(t-5)	0	0	0.06	0.5	0.11	0	0.08
count_Atl_percent(t-6)	0	0.01	0.15	0.4	0.14	0.04	0.22
count_Atl_percent(t-7)	0	0.01	0.1	0.33	0.08	0	0.01
count_Atl_percent(t-8)	0	0	0.09	0.38	0.09	0	0.08
count_Atl_percent(t-9)	0	0.02	0.09	0.42	0.1	0	0.08



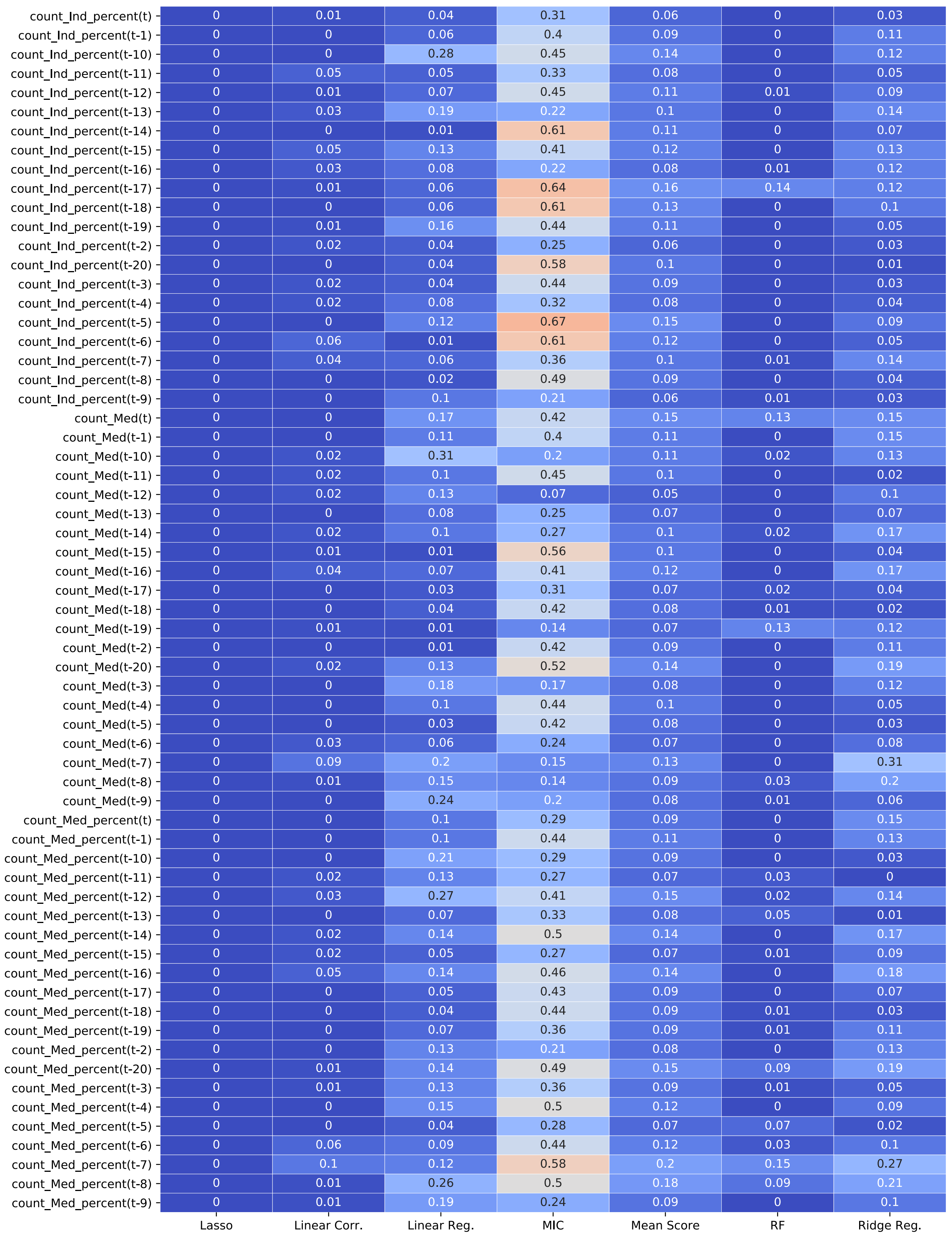
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
count_CP(t)	0	0.04	0.01	0.61	0.13	0.02	0.09
count_CP(t-1)	0	0	0.04	0.19	0.05	0	0.05
count_CP(t-10)	0	0.06	0.06	0.51	0.13	0	0.18
count_CP(t-11)	0	0.03	0.24	0.11	0.07	0	0.06
count_CP(t-12)	0	0.02	0.03	0.36	0.09	0.05	0.1
count_CP(t-13)	0	0.01	0.02	0.33	0.07	0	0.05
count_CP(t-14)	0	0.04	0.09	0.32	0.11	0	0.18
count_CP(t-15)	0	0	0.01	0.25	0.05	0	0.02
count_CP(t-16)	0	0.04	0.08	0.2	0.06	0	0.05
count_CP(t-17)	0	0.02	0.05	0.31	0.09	0.01	0.12
count_CP(t-18)	0	0.01	0.04	0.41	0.09	0	0.1
count_CP(t-19)	0	0.02	0.09	0.38	0.1	0	0.09
count_CP(t-2)	0	0.01	0.06	0.49	0.1	0	0.04
count_CP(t-20)	0	0.01	0.09	0.24	0.07	0	0.09
count_CP(t-3)	0	0.03	0.02	0.17	0.06	0.01	0.13
count_CP(t-4)	0	0	0.01	0.36	0.07	0	0.03
count_CP(t-5)	0	0	0.06	0.11	0.05	0.1	0.05
count_CP(t-6)	0	0	0.17	0.34	0.11	0.01	0.12
count_CP(t-7)	0	0	0.04	0	0.01	0	0.02
count_CP(t-8)	0	0	0.09	0.46	0.11	0	0.12
count_CP(t-9)	0	0	0.06	0.59	0.11	0	0.04
count_CP_percent(t)	0	0.03	0	0.51	0.11	0.01	0.09
count_CP_percent(t-1)	0	0	0.06	0.25	0.08	0.14	0.02
count_CP_percent(t-10)	0	0.03	0	0.54	0.12	0.02	0.11
count_CP_percent(t-11)	0	0.02	0.04	0.26	0.06	0.01	0.01
count_CP_percent(t-12)	0	0.01	0	0.51	0.12	0.08	0.1
count_CP_percent(t-13)	0	0.01	0.05	0.53	0.11	0.04	0.03
count_CP_percent(t-14)	0	0.06	0.06	0.38	0.12	0	0.2
count_CP_percent(t-15)	0	0	0.03	0.64	0.12	0.05	0.02
count_CP_percent(t-16)	0	0.05	0.06	0.26	0.07	0	0.06
count_CP_percent(t-17)	0	0.02	0.05	0.31	0.08	0	0.1
count_CP_percent(t-18)	0	0.01	0.02	0.53	0.11	0	0.12
count_CP_percent(t-19)	0	0.02	0.23	0.19	0.09	0.01	0.1
count_CP_percent(t-2)	0	0.02	0.02	0.56	0.14	0.11	0.12
count_CP_percent(t-20)	0	0	0.1	0.4	0.09	0	0.02
count_CP_percent(t-3)	0	0.01	0.04	0.42	0.09	0	0.09
count_CP_percent(t-4)	0	0	0.02	0.41	0.08	0.02	0.06
count_CP_percent(t-5)	0	0	0.03	0.16	0.05	0.07	0.04
count_CP_percent(t-6)	0	0.01	0.16	0.39	0.12	0.06	0.12
count_CP_percent(t-7)	0	0	0.03	0.3	0.07	0.04	0.03
count_CP_percent(t-8)	0	0	0.03	0.38	0.07	0	0.04
count_CP_percent(t-9)	0	0.01	0	0.46	0.08	0	0
count_EstP(t)	0	0.02	0.2	0.53	0.15	0	0.12
count_EstP(t-1)	0	0.01	0.04	0.36	0.08	0	0.07
count_EstP(t-10)	0	0.02	0.19	0.25	0.12	0	0.24
count_EstP(t-11)	0	0.01	0.04	0.21	0.05	0	0.03
count_EstP(t-12)	0	0	0.12	0.26	0.08	0	0.08
count_EstP(t-13)	0	0	0.09	0.29	0.08	0	0.08
count_EstP(t-14)	0	0	0.01	0.65	0.11	0	0.01
count_EstP(t-15)	0	0.05	0.05	0.47	0.11	0	0.09
count_EstP(t-16)	0	0.01	0.17	0.14	0.06	0	0.03
count_EstP(t-17)	0	0.1	0.03	0.32	0.11	0	0.2
count_EstP(t-18)	0	0.01	0.13	0.32	0.09	0.01	0.06
count_EstP(t-19)	0	0.01	0.06	0.3	0.07	0	0.03
count_EstP(t-2)	0	0.01	0.05	0.29	0.06	0	0.02
count_EstP(t-20)	0	0	0.1	0.57	0.12	0	0.02
count_EstP(t-3)	0	0	0.04	0.53	0.11	0.01	0.1
count_EstP(t-4)	0	0.01	0.11	0.43	0.09	0	0.01
count_EstP(t-5)	0	0	0.18	0.24	0.08	0	0.06
count_EstP(t-6)	0	0.02	0.18	0.22	0.08	0	0.07
count_EstP(t-7)	0	0.03	0.01	0.36	0.08	0	0.09
count_EstP(t-8)	0	0.01	0.11	0.32	0.09	0	0.08
count_EstP(t-9)	0	0.03	0.07	0.36	0.08	0	0.03



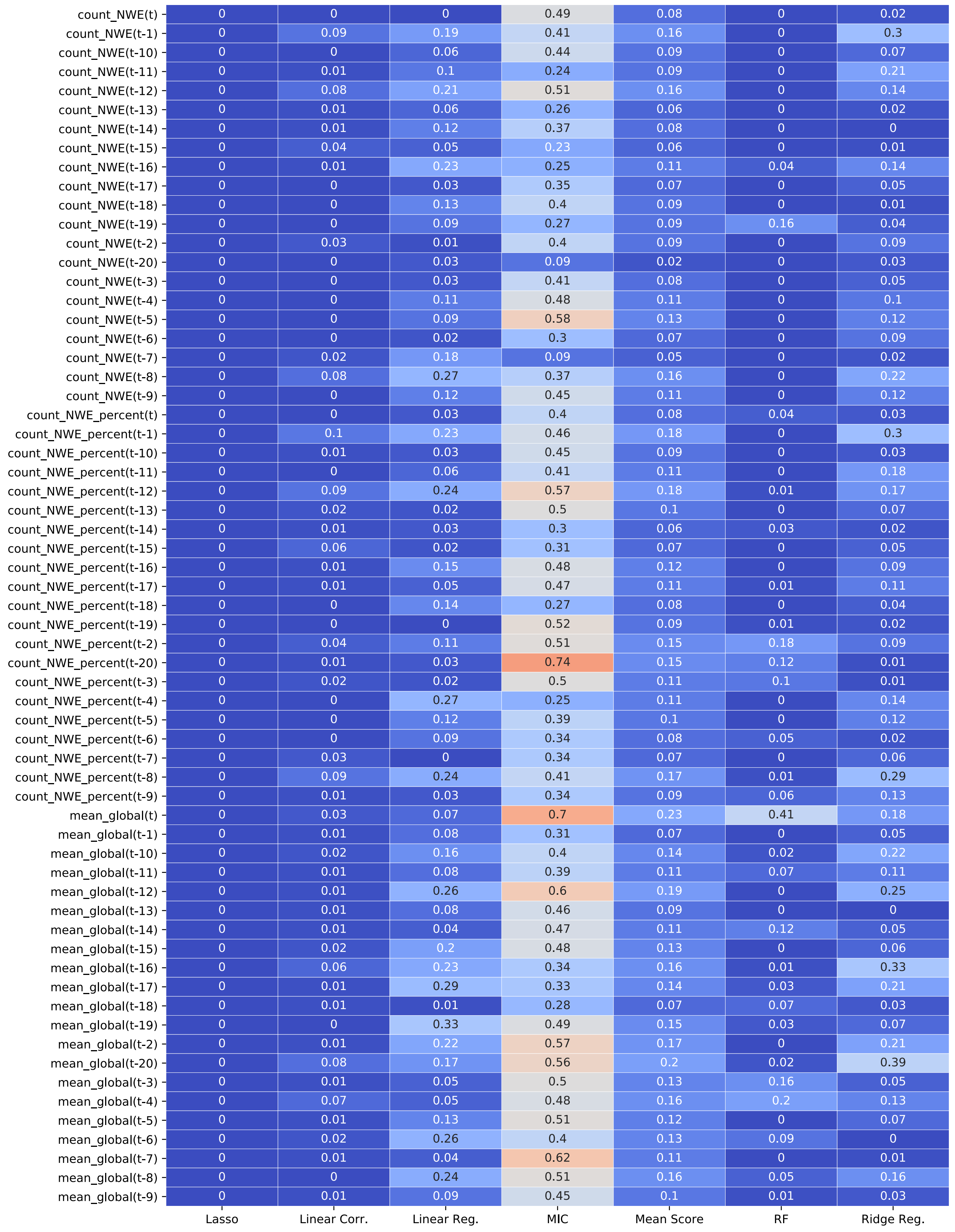
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
count_EstP_percent(t)	0	0.02	0.27	0.42	0.17	0.1	0.19
count_EstP_percent(t-1)	0	0.01	0.07	0.33	0.14	0.32	0.11
count_EstP_percent(t-10)	0	0.02	0.17	0.29	0.12	0	0.25
count_EstP_percent(t-11)	0	0.02	0.04	0.34	0.07	0	0.05
count_EstP_percent(t-12)	0	0	0.11	0.34	0.09	0	0.07
count_EstP_percent(t-13)	0	0	0.15	0.4	0.1	0.01	0.06
count_EstP_percent(t-14)	0	0	0.06	0.64	0.12	0	0
count_EstP_percent(t-15)	0	0.07	0.01	0.49	0.12	0	0.15
count_EstP_percent(t-16)	0	0.01	0.18	0.23	0.08	0	0.05
count_EstP_percent(t-17)	0	0.09	0.05	0.3	0.1	0	0.18
count_EstP_percent(t-18)	0	0.01	0	0.33	0.08	0.03	0.11
count_EstP_percent(t-19)	0	0.01	0.06	0.68	0.13	0	0.03
count_EstP_percent(t-2)	0	0.02	0.08	0.3	0.08	0	0.06
count_EstP_percent(t-20)	0	0	0.03	0.39	0.07	0	0.03
count_EstP_percent(t-3)	0	0.01	0.07	0.76	0.16	0	0.12
count_EstP_percent(t-4)	0	0.01	0.1	0.44	0.1	0	0.02
count_EstP_percent(t-5)	0	0	0.11	0.24	0.07	0	0.06
count_EstP_percent(t-6)	0	0.01	0.17	0.39	0.11	0.02	0.08
count_EstP_percent(t-7)	0	0.04	0.06	0.4	0.1	0	0.09
count_EstP_percent(t-8)	0	0.01	0.06	0.45	0.1	0	0.08
count_EstP_percent(t-9)	0	0.03	0.11	0.25	0.07	0	0.03
count_FE(t)	0	0	0.1	0.37	0.09	0	0.08
count_FE(t-1)	0	0.02	0.18	0.39	0.13	0	0.19
count_FE(t-10)	0	0	0.33	0.32	0.14	0	0.19
count_FE(t-11)	0	0.01	0.17	0.19	0.07	0	0.05
count_FE(t-12)	0	0.01	0.11	0.36	0.09	0.01	0.05
count_FE(t-13)	0	0	0.03	0.56	0.1	0	0.01
count_FE(t-14)	0	0.02	0.05	0.28	0.07	0	0.07
count_FE(t-15)	0	0.05	0.06	0.52	0.13	0	0.14
count_FE(t-16)	0	0.01	0.01	0.55	0.12	0.09	0.08
count_FE(t-17)	0	0.01	0.14	0.29	0.09	0.03	0.09
count_FE(t-18)	0	0	0.04	0.17	0.04	0	0.02
count_FE(t-19)	0	0.01	0.02	0.53	0.11	0.01	0.08
count_FE(t-2)	0	0.11	0.08	0.07	0.1	0.11	0.23
count_FE(t-20)	0	0.02	0	0.17	0.06	0.02	0.13
count_FE(t-3)	0	0.05	0.36	0.26	0.15	0	0.24
count_FE(t-4)	0	0	0.14	0.44	0.13	0.06	0.11
count_FE(t-5)	0	0.01	0.1	0.3	0.08	0	0.04
count_FE(t-6)	0	0.02	0.25	0.39	0.16	0	0.27
count_FE(t-7)	0	0.03	0.32	0.39	0.13	0	0.02
count_FE(t-8)	0	0.04	0.14	0.11	0.09	0	0.27
count_FE(t-9)	0	0.06	0.28	0.08	0.1	0	0.15
count_FE_percent(t)	0	0.01	0.06	0.53	0.11	0	0.04
count_FE_percent(t-1)	0	0.04	0.09	0.4	0.14	0.12	0.2
count_FE_percent(t-10)	0	0.03	0.08	0.53	0.12	0.01	0.07
count_FE_percent(t-11)	0	0.07	0.07	0.46	0.11	0	0.05
count_FE_percent(t-12)	0	0.03	0.01	0.38	0.1	0.01	0.17
count_FE_percent(t-13)	0	0	0.01	0.31	0.06	0	0.05
count_FE_percent(t-14)	0	0.01	0	0.45	0.09	0	0.06
count_FE_percent(t-15)	0	0	0.07	0.39	0.08	0	0.01
count_FE_percent(t-16)	0	0.02	0.12	0.3	0.08	0	0.07
count_FE_percent(t-17)	0	0.01	0.14	0.56	0.14	0.01	0.13
count_FE_percent(t-18)	0	0.01	0.03	0.45	0.08	0	0.02
count_FE_percent(t-19)	0	0.02	0.05	0.36	0.08	0	0.05
count_FE_percent(t-2)	0	0.03	0.07	0.41	0.11	0.02	0.12
count_FE_percent(t-20)	0	0	0	0.25	0.05	0	0.04
count_FE_percent(t-3)	0	0	0.08	0.31	0.08	0	0.08
count_FE_percent(t-4)	0	0.02	0.07	0.39	0.1	0	0.09
count_FE_percent(t-5)	0	0	0.14	0.33	0.08	0	0.02
count_FE_percent(t-6)	0	0.03	0.07	0.43	0.11	0	0.11
count_FE_percent(t-7)	0	0.01	0.04	0.4	0.08	0	0.05
count_FE_percent(t-8)	0	0.01	0.13	0.35	0.09	0	0.07
count_FE_percent(t-9)	0	0.01	0.15	0.34	0.1	0.1	0.02



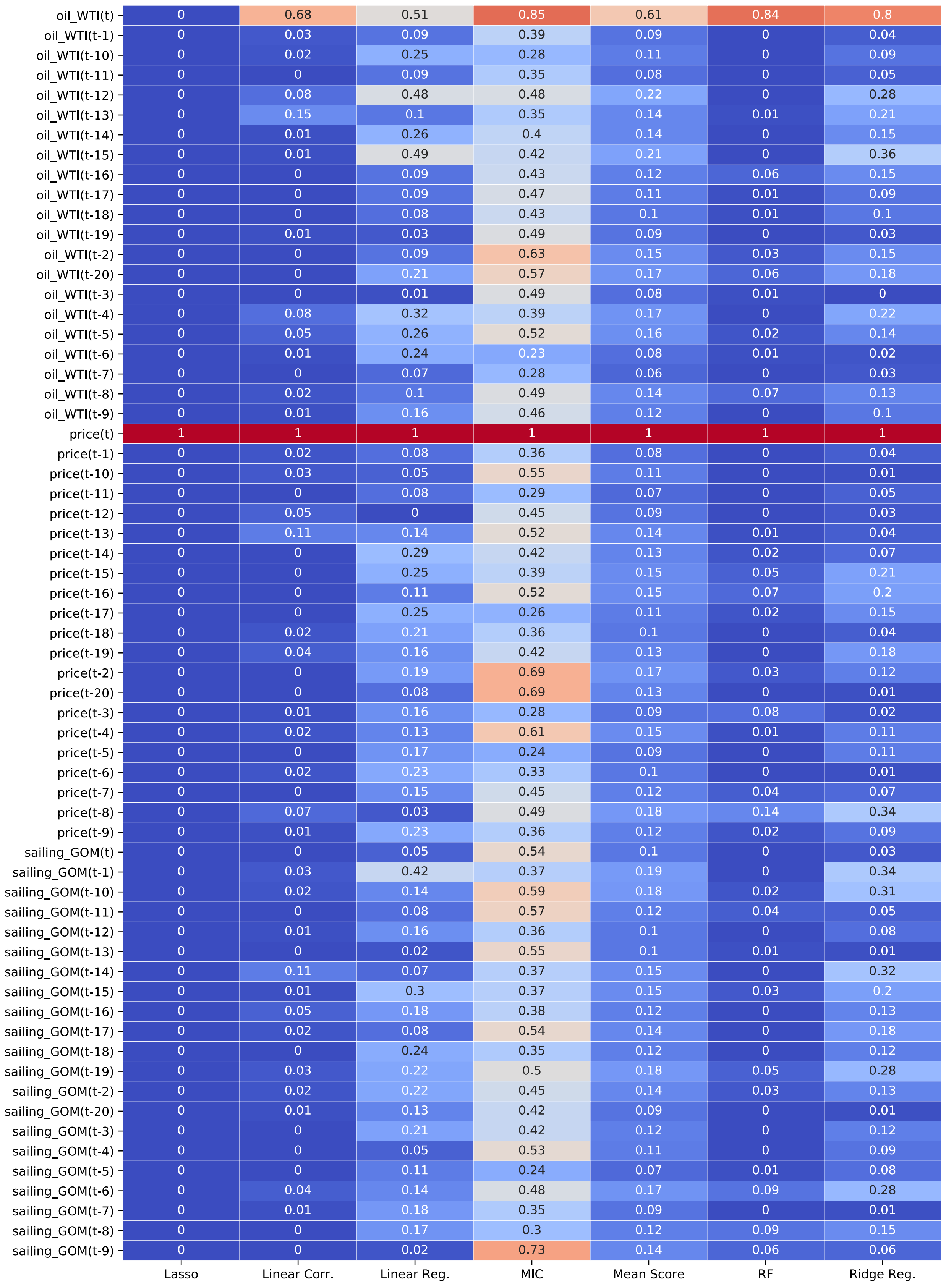
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
count_GOM(t)	0	0	0.2	0.54	0.15	0	0.14
count_GOM(t-1)	0	0.04	0	0.27	0.06	0	0.04
count_GOM(t-10)	0	0	0.18	0.23	0.07	0	0.01
count_GOM(t-11)	0	0	0.07	0.46	0.1	0.05	0.04
count_GOM(t-12)	0	0.03	0.04	0.25	0.08	0	0.15
count_GOM(t-13)	0	0	0.15	0.5	0.11	0	0.02
count_GOM(t-14)	0	0.01	0.1	0.32	0.09	0	0.12
count_GOM(t-15)	0	0	0.23	0.6	0.15	0.01	0.07
count_GOM(t-16)	0	0	0.23	0.25	0.1	0	0.12
count_GOM(t-17)	0	0	0.12	0.37	0.08	0	0.01
count_GOM(t-18)	0	0.06	0.12	0.35	0.12	0	0.22
count_GOM(t-19)	0	0	0.14	0.42	0.09	0	0
count_GOM(t-2)	0	0	0.15	0.41	0.12	0.01	0.14
count_GOM(t-20)	0	0.01	0.09	0.38	0.09	0	0.07
count_GOM(t-3)	0	0.02	0.03	0.34	0.09	0	0.13
count_GOM(t-4)	0	0.06	0.05	0.01	0.05	0	0.15
count_GOM(t-5)	0	0.01	0.17	0.17	0.08	0	0.15
count_GOM(t-6)	0	0	0.02	0.39	0.07	0	0
count_GOM(t-7)	0	0	0.02	0.58	0.11	0	0.08
count_GOM(t-8)	0	0.02	0.1	0.22	0.07	0	0.1
count_GOM(t-9)	0	0	0.08	0.45	0.09	0.01	0.03
count_GOM_percent(t)	0	0	0.28	0.29	0.13	0	0.18
count_GOM_percent(t-1)	0	0.05	0	0.29	0.07	0.01	0.08
count_GOM_percent(t-10)	0	0.01	0.23	0.46	0.15	0.13	0.06
count_GOM_percent(t-11)	0	0	0.02	0.35	0.07	0.02	0.03
count_GOM_percent(t-12)	0	0.04	0.12	0.26	0.1	0	0.21
count_GOM_percent(t-13)	0	0	0.15	0.64	0.16	0.15	0.02
count_GOM_percent(t-14)	0	0.02	0.24	0.55	0.17	0	0.19
count_GOM_percent(t-15)	0	0	0.26	0.57	0.16	0.06	0.08
count_GOM_percent(t-16)	0	0.01	0.39	0.24	0.14	0	0.2
count_GOM_percent(t-17)	0	0	0.01	0.5	0.1	0.03	0.04
count_GOM_percent(t-18)	0	0.05	0.14	0.57	0.17	0	0.25
count_GOM_percent(t-19)	0	0	0.12	0.52	0.11	0	0.04
count_GOM_percent(t-2)	0	0.01	0.2	0.37	0.13	0	0.19
count_GOM_percent(t-20)	0	0	0.18	0.39	0.12	0.08	0.09
count_GOM_percent(t-3)	0	0.01	0.04	0.21	0.07	0	0.14
count_GOM_percent(t-4)	0	0.06	0.14	0.31	0.11	0.02	0.14
count_GOM_percent(t-5)	0	0.01	0.19	0.43	0.14	0.01	0.18
count_GOM_percent(t-6)	0	0	0	0.37	0.06	0	0
count_GOM_percent(t-7)	0	0	0	0.63	0.12	0.02	0.06
count_GOM_percent(t-8)	0	0.01	0.16	0.37	0.11	0	0.15
count_GOM_percent(t-9)	0	0.01	0	0.55	0.12	0.06	0.09
count_Ind(t)	0	0.01	0.01	0.36	0.06	0	0
count_Ind(t-1)	0	0	0.1	0.43	0.11	0	0.1
count_Ind(t-10)	0	0.01	0.09	0.2	0.07	0.08	0.02
count_Ind(t-11)	0	0.04	0	0.29	0.06	0	0.01
count_Ind(t-12)	0	0.01	0.11	0.22	0.07	0	0.11
count_Ind(t-13)	0	0.02	0.11	0.37	0.1	0	0.1
count_Ind(t-14)	0	0	0.02	0.43	0.09	0	0.07
count_Ind(t-15)	0	0.04	0.09	0.33	0.1	0.02	0.12
count_Ind(t-16)	0	0.02	0.06	0.07	0.04	0	0.11
count_Ind(t-17)	0	0.01	0.11	0.45	0.11	0	0.12
count_Ind(t-18)	0	0	0.18	0.64	0.17	0.04	0.14
count_Ind(t-19)	0	0.01	0.15	0.36	0.09	0	0.03
count_Ind(t-2)	0	0.02	0.03	0.4	0.08	0	0.04
count_Ind(t-20)	0	0	0.09	0.28	0.07	0.03	0.01
count_Ind(t-3)	0	0.02	0.07	0.19	0.06	0	0.05
count_Ind(t-4)	0	0.02	0.06	0.35	0.08	0.01	0.03
count_Ind(t-5)	0	0	0.11	0.3	0.08	0	0.08
count_Ind(t-6)	0	0.05	0.04	0.74	0.14	0	0.04
count_Ind(t-7)	0	0.03	0.06	0.52	0.12	0	0.14
count_Ind(t-8)	0	0	0.08	0.28	0.07	0	0.09
count_Ind(t-9)	0	0.01	0.09	0.04	0.05	0.03	0.1



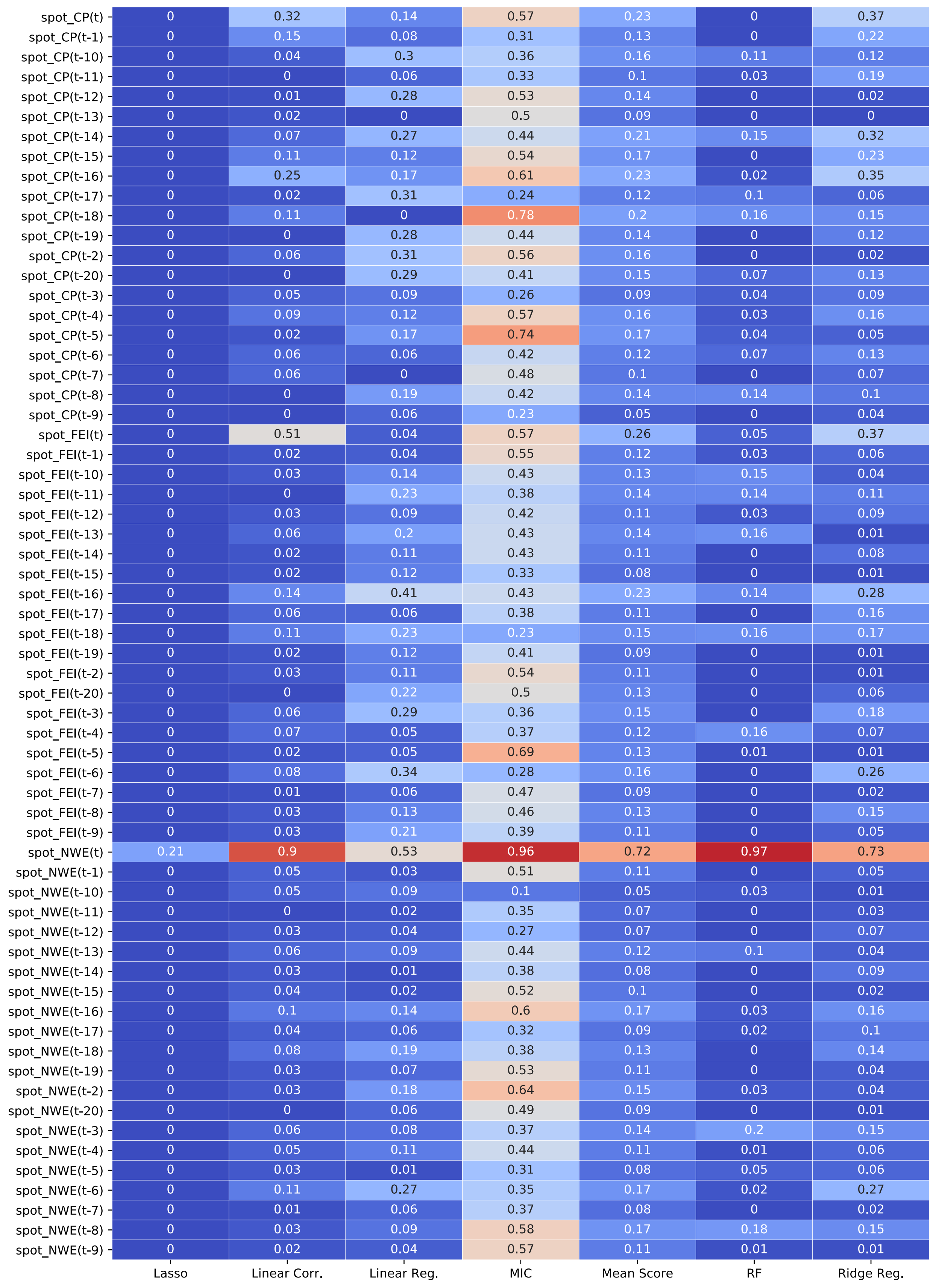
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
count_Ind_percent(t)	0	0.01	0.04	0.31	0.06	0	0.03
count_Ind_percent(t-1)	0	0	0.06	0.4	0.09	0	0.11
count_Ind_percent(t-10)	0	0	0.28	0.45	0.14	0	0.12
count_Ind_percent(t-11)	0	0.05	0.05	0.33	0.08	0	0.05
count_Ind_percent(t-12)	0	0.01	0.07	0.45	0.11	0.01	0.09
count_Ind_percent(t-13)	0	0.03	0.19	0.22	0.1	0	0.14
count_Ind_percent(t-14)	0	0	0.01	0.61	0.11	0	0.07
count_Ind_percent(t-15)	0	0.05	0.13	0.41	0.12	0	0.13
count_Ind_percent(t-16)	0	0.03	0.08	0.22	0.08	0.01	0.12
count_Ind_percent(t-17)	0	0.01	0.06	0.64	0.16	0.14	0.12
count_Ind_percent(t-18)	0	0	0.06	0.61	0.13	0	0.1
count_Ind_percent(t-19)	0	0.01	0.16	0.44	0.11	0	0.05
count_Ind_percent(t-2)	0	0.02	0.04	0.25	0.06	0	0.03
count_Ind_percent(t-20)	0	0	0.04	0.58	0.1	0	0.01
count_Ind_percent(t-3)	0	0.02	0.04	0.44	0.09	0	0.03
count_Ind_percent(t-4)	0	0.02	0.08	0.32	0.08	0	0.04
count_Ind_percent(t-5)	0	0	0.12	0.67	0.15	0	0.09
count_Ind_percent(t-6)	0	0.06	0.01	0.61	0.12	0	0.05
count_Ind_percent(t-7)	0	0.04	0.06	0.36	0.1	0.01	0.14
count_Ind_percent(t-8)	0	0	0.02	0.49	0.09	0	0.04
count_Ind_percent(t-9)	0	0	0.1	0.21	0.06	0.01	0.03
count_Med(t)	0	0	0.17	0.42	0.15	0.13	0.15
count_Med(t-1)	0	0	0.11	0.4	0.11	0	0.15
count_Med(t-10)	0	0.02	0.31	0.2	0.11	0.02	0.13
count_Med(t-11)	0	0.02	0.1	0.45	0.1	0	0.02
count_Med(t-12)	0	0.02	0.13	0.07	0.05	0	0.1
count_Med(t-13)	0	0	0.08	0.25	0.07	0	0.07
count_Med(t-14)	0	0.02	0.1	0.27	0.1	0.02	0.17
count_Med(t-15)	0	0.01	0.01	0.56	0.1	0	0.04
count_Med(t-16)	0	0.04	0.07	0.41	0.12	0	0.17
count_Med(t-17)	0	0	0.03	0.31	0.07	0.02	0.04
count_Med(t-18)	0	0	0.04	0.42	0.08	0.01	0.02
count_Med(t-19)	0	0.01	0.01	0.14	0.07	0.13	0.12
count_Med(t-2)	0	0	0.01	0.42	0.09	0	0.11
count_Med(t-20)	0	0.02	0.13	0.52	0.14	0	0.19
count_Med(t-3)	0	0	0.18	0.17	0.08	0	0.12
count_Med(t-4)	0	0	0.1	0.44	0.1	0	0.05
count_Med(t-5)	0	0	0.03	0.42	0.08	0	0.03
count_Med(t-6)	0	0.03	0.06	0.24	0.07	0	0.08
count_Med(t-7)	0	0.09	0.2	0.15	0.13	0	0.31
count_Med(t-8)	0	0.01	0.15	0.14	0.09	0.03	0.2
count_Med(t-9)	0	0	0.24	0.2	0.08	0.01	0.06
count_Med_percent(t)	0	0	0.1	0.29	0.09	0	0.15
count_Med_percent(t-1)	0	0	0.1	0.44	0.11	0	0.13
count_Med_percent(t-10)	0	0	0.21	0.29	0.09	0	0.03
count_Med_percent(t-11)	0	0.02	0.13	0.27	0.07	0.03	0
count_Med_percent(t-12)	0	0.03	0.27	0.41	0.15	0.02	0.14
count_Med_percent(t-13)	0	0	0.07	0.33	0.08	0.05	0.01
count_Med_percent(t-14)	0	0.02	0.14	0.5	0.14	0	0.17
count_Med_percent(t-15)	0	0.02	0.05	0.27	0.07	0.01	0.09
count_Med_percent(t-16)	0	0.05	0.14	0.46	0.14	0	0.18
count_Med_percent(t-17)	0	0	0.05	0.43	0.09	0	0.07
count_Med_percent(t-18)	0	0	0.04	0.44	0.09	0.01	0.03
count_Med_percent(t-19)	0	0	0.07	0.36	0.09	0.01	0.11
count_Med_percent(t-2)	0	0	0.13	0.21	0.08	0	0.13
count_Med_percent(t-20)	0	0.01	0.14	0.49	0.15	0.09	0.19
count_Med_percent(t-3)	0	0.01	0.13	0.36	0.09	0.01	0.05
count_Med_percent(t-4)	0	0	0.15	0.5	0.12	0	0.09
count_Med_percent(t-5)	0	0	0.04	0.28	0.07	0.07	0.02
count_Med_percent(t-6)	0	0.06	0.09	0.44	0.12	0.03	0.1
count_Med_percent(t-7)	0	0.1	0.12	0.58	0.2	0.15	0.27
count_Med_percent(t-8)	0	0.01	0.26	0.5	0.18	0.09	0.21
count_Med_percent(t-9)	0	0.01	0.19	0.24	0.09	0	0.1



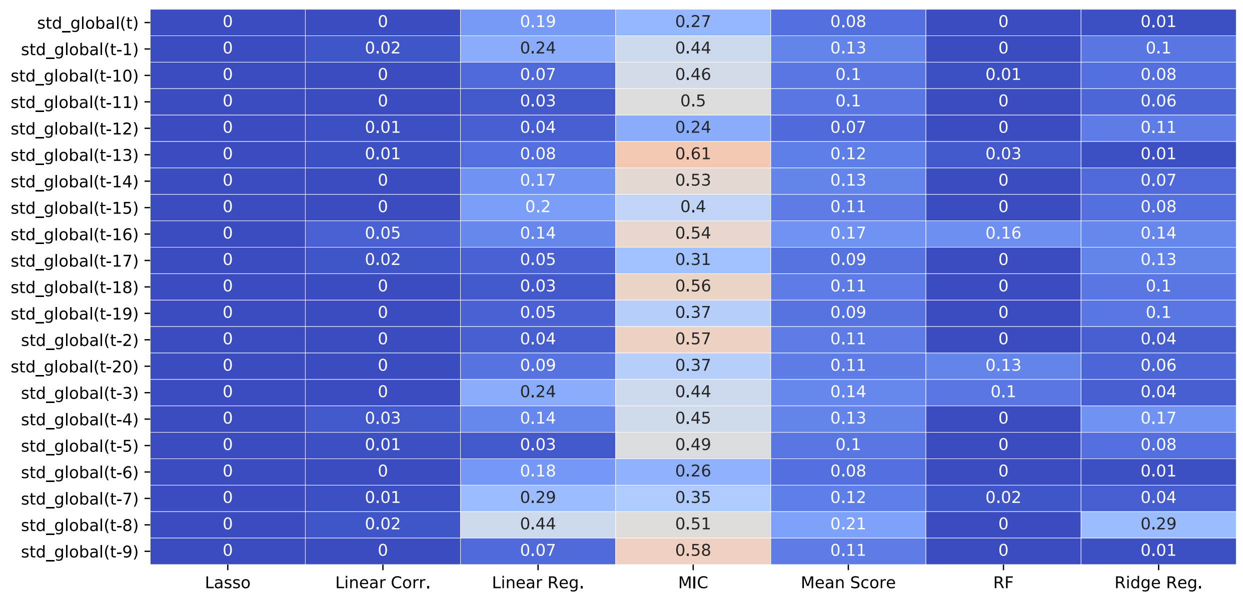
	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
count_NWE(t)	0	0	0	0.49	0.08	0	0.02
count_NWE(t-1)	0	0.09	0.19	0.41	0.16	0	0.3
count_NWE(t-10)	0	0	0.06	0.44	0.09	0	0.07
count_NWE(t-11)	0	0.01	0.1	0.24	0.09	0	0.21
count_NWE(t-12)	0	0.08	0.21	0.51	0.16	0	0.14
count_NWE(t-13)	0	0.01	0.06	0.26	0.06	0	0.02
count_NWE(t-14)	0	0.01	0.12	0.37	0.08	0	0
count_NWE(t-15)	0	0.04	0.05	0.23	0.06	0	0.01
count_NWE(t-16)	0	0.01	0.23	0.25	0.11	0.04	0.14
count_NWE(t-17)	0	0	0.03	0.35	0.07	0	0.05
count_NWE(t-18)	0	0	0.13	0.4	0.09	0	0.01
count_NWE(t-19)	0	0	0.09	0.27	0.09	0.16	0.04
count_NWE(t-2)	0	0.03	0.01	0.4	0.09	0	0.09
count_NWE(t-20)	0	0	0.03	0.09	0.02	0	0.03
count_NWE(t-3)	0	0	0.03	0.41	0.08	0	0.05
count_NWE(t-4)	0	0	0.11	0.48	0.11	0	0.1
count_NWE(t-5)	0	0	0.09	0.58	0.13	0	0.12
count_NWE(t-6)	0	0	0.02	0.3	0.07	0	0.09
count_NWE(t-7)	0	0.02	0.18	0.09	0.05	0	0.02
count_NWE(t-8)	0	0.08	0.27	0.37	0.16	0	0.22
count_NWE(t-9)	0	0	0.12	0.45	0.11	0	0.12
count_NWE_percent(t)	0	0	0.03	0.4	0.08	0.04	0.03
count_NWE_percent(t-1)	0	0.1	0.23	0.46	0.18	0	0.3
count_NWE_percent(t-10)	0	0.01	0.03	0.45	0.09	0	0.03
count_NWE_percent(t-11)	0	0	0.06	0.41	0.11	0	0.18
count_NWE_percent(t-12)	0	0.09	0.24	0.57	0.18	0.01	0.17
count_NWE_percent(t-13)	0	0.02	0.02	0.5	0.1	0	0.07
count_NWE_percent(t-14)	0	0.01	0.03	0.3	0.06	0.03	0.02
count_NWE_percent(t-15)	0	0.06	0.02	0.31	0.07	0	0.05
count_NWE_percent(t-16)	0	0.01	0.15	0.48	0.12	0	0.09
count_NWE_percent(t-17)	0	0.01	0.05	0.47	0.11	0.01	0.11
count_NWE_percent(t-18)	0	0	0.14	0.27	0.08	0	0.04
count_NWE_percent(t-19)	0	0	0	0.52	0.09	0.01	0.02
count_NWE_percent(t-2)	0	0.04	0.11	0.51	0.15	0.18	0.09
count_NWE_percent(t-20)	0	0.01	0.03	0.74	0.15	0.12	0.01
count_NWE_percent(t-3)	0	0.02	0.02	0.5	0.11	0.1	0.01
count_NWE_percent(t-4)	0	0	0.27	0.25	0.11	0	0.14
count_NWE_percent(t-5)	0	0	0.12	0.39	0.1	0	0.12
count_NWE_percent(t-6)	0	0	0.09	0.34	0.08	0.05	0.02
count_NWE_percent(t-7)	0	0.03	0	0.34	0.07	0	0.06
count_NWE_percent(t-8)	0	0.09	0.24	0.41	0.17	0.01	0.29
count_NWE_percent(t-9)	0	0.01	0.03	0.34	0.09	0.06	0.13
mean_global(t)	0	0.03	0.07	0.7	0.23	0.41	0.18
mean_global(t-1)	0	0.01	0.08	0.31	0.07	0	0.05
mean_global(t-10)	0	0.02	0.16	0.4	0.14	0.02	0.22
mean_global(t-11)	0	0.01	0.08	0.39	0.11	0.07	0.11
mean_global(t-12)	0	0.01	0.26	0.6	0.19	0	0.25
mean_global(t-13)	0	0.01	0.08	0.46	0.09	0	0
mean_global(t-14)	0	0.01	0.04	0.47	0.11	0.12	0.05
mean_global(t-15)	0	0.02	0.2	0.48	0.13	0	0.06
mean_global(t-16)	0	0.06	0.23	0.34	0.16	0.01	0.33
mean_global(t-17)	0	0.01	0.29	0.33	0.14	0.03	0.21
mean_global(t-18)	0	0.01	0.01	0.28	0.07	0.07	0.03
mean_global(t-19)	0	0	0.33	0.49	0.15	0.03	0.07
mean_global(t-2)	0	0.01	0.22	0.57	0.17	0	0.21
mean_global(t-20)	0	0.08	0.17	0.56	0.2	0.02	0.39
mean_global(t-3)	0	0.01	0.05	0.5	0.13	0.16	0.05
mean_global(t-4)	0	0.07	0.05	0.48	0.16	0.2	0.13
mean_global(t-5)	0	0.01	0.13	0.51	0.12	0	0.07
mean_global(t-6)	0	0.02	0.26	0.4	0.13	0.09	0
mean_global(t-7)	0	0.01	0.04	0.62	0.11	0	0.01
mean_global(t-8)	0	0	0.24	0.51	0.16	0.05	0.16
mean_global(t-9)	0	0.01	0.09	0.45	0.1	0.01	0.03



	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
oil_WTI(t)	0	0.68	0.51	0.85	0.61	0.84	0.8
oil_WTI(t-1)	0	0.03	0.09	0.39	0.09	0	0.04
oil_WTI(t-10)	0	0.02	0.25	0.28	0.11	0	0.09
oil_WTI(t-11)	0	0	0.09	0.35	0.08	0	0.05
oil_WTI(t-12)	0	0.08	0.48	0.48	0.22	0	0.28
oil_WTI(t-13)	0	0.15	0.1	0.35	0.14	0.01	0.21
oil_WTI(t-14)	0	0.01	0.26	0.4	0.14	0	0.15
oil_WTI(t-15)	0	0.01	0.49	0.42	0.21	0	0.36
oil_WTI(t-16)	0	0	0.09	0.43	0.12	0.06	0.15
oil_WTI(t-17)	0	0	0.09	0.47	0.11	0.01	0.09
oil_WTI(t-18)	0	0	0.08	0.43	0.1	0.01	0.1
oil_WTI(t-19)	0	0.01	0.03	0.49	0.09	0	0.03
oil_WTI(t-2)	0	0	0.09	0.63	0.15	0.03	0.15
oil_WTI(t-20)	0	0	0.21	0.57	0.17	0.06	0.18
oil_WTI(t-3)	0	0	0.01	0.49	0.08	0.01	0
oil_WTI(t-4)	0	0.08	0.32	0.39	0.17	0	0.22
oil_WTI(t-5)	0	0.05	0.26	0.52	0.16	0.02	0.14
oil_WTI(t-6)	0	0.01	0.24	0.23	0.08	0.01	0.02
oil_WTI(t-7)	0	0	0.07	0.28	0.06	0	0.03
oil_WTI(t-8)	0	0.02	0.1	0.49	0.14	0.07	0.13
oil_WTI(t-9)	0	0.01	0.16	0.46	0.12	0	0.1
price(t)	1	1	1	1	1	1	1
price(t-1)	0	0.02	0.08	0.36	0.08	0	0.04
price(t-10)	0	0.03	0.05	0.55	0.11	0	0.01
price(t-11)	0	0	0.08	0.29	0.07	0	0.05
price(t-12)	0	0.05	0	0.45	0.09	0	0.03
price(t-13)	0	0.11	0.14	0.52	0.14	0.01	0.04
price(t-14)	0	0	0.29	0.42	0.13	0.02	0.07
price(t-15)	0	0	0.25	0.39	0.15	0.05	0.21
price(t-16)	0	0	0.11	0.52	0.15	0.07	0.2
price(t-17)	0	0	0.25	0.26	0.11	0.02	0.15
price(t-18)	0	0.02	0.21	0.36	0.1	0	0.04
price(t-19)	0	0.04	0.16	0.42	0.13	0	0.18
price(t-2)	0	0	0.19	0.69	0.17	0.03	0.12
price(t-20)	0	0	0.08	0.69	0.13	0	0.01
price(t-3)	0	0.01	0.16	0.28	0.09	0.08	0.02
price(t-4)	0	0.02	0.13	0.61	0.15	0.01	0.11
price(t-5)	0	0	0.17	0.24	0.09	0	0.11
price(t-6)	0	0.02	0.23	0.33	0.1	0	0.01
price(t-7)	0	0	0.15	0.45	0.12	0.04	0.07
price(t-8)	0	0.07	0.03	0.49	0.18	0.14	0.34
price(t-9)	0	0.01	0.23	0.36	0.12	0.02	0.09
sailing_GOM(t)	0	0	0.05	0.54	0.1	0	0.03
sailing_GOM(t-1)	0	0.03	0.42	0.37	0.19	0	0.34
sailing_GOM(t-10)	0	0.02	0.14	0.59	0.18	0.02	0.31
sailing_GOM(t-11)	0	0	0.08	0.57	0.12	0.04	0.05
sailing_GOM(t-12)	0	0.01	0.16	0.36	0.1	0	0.08
sailing_GOM(t-13)	0	0	0.02	0.55	0.1	0.01	0.01
sailing_GOM(t-14)	0	0.11	0.07	0.37	0.15	0	0.32
sailing_GOM(t-15)	0	0.01	0.3	0.37	0.15	0.03	0.2
sailing_GOM(t-16)	0	0.05	0.18	0.38	0.12	0	0.13
sailing_GOM(t-17)	0	0.02	0.08	0.54	0.14	0	0.18
sailing_GOM(t-18)	0	0	0.24	0.35	0.12	0	0.12
sailing_GOM(t-19)	0	0.03	0.22	0.5	0.18	0.05	0.28
sailing_GOM(t-2)	0	0.02	0.22	0.45	0.14	0.03	0.13
sailing_GOM(t-20)	0	0.01	0.13	0.42	0.09	0	0.01
sailing_GOM(t-3)	0	0	0.21	0.42	0.12	0	0.12
sailing_GOM(t-4)	0	0	0.05	0.53	0.11	0	0.09
sailing_GOM(t-5)	0	0	0.11	0.24	0.07	0.01	0.08
sailing_GOM(t-6)	0	0.04	0.14	0.48	0.17	0.09	0.28
sailing_GOM(t-7)	0	0.01	0.18	0.35	0.09	0	0.01
sailing_GOM(t-8)	0	0	0.17	0.3	0.12	0.09	0.15
sailing_GOM(t-9)	0	0	0.02	0.73	0.14	0.06	0.06



	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
spot_CP(t)	0	0.32	0.14	0.57	0.23	0	0.37
spot_CP(t-1)	0	0.15	0.08	0.31	0.13	0	0.22
spot_CP(t-10)	0	0.04	0.3	0.36	0.16	0.11	0.12
spot_CP(t-11)	0	0	0.06	0.33	0.1	0.03	0.19
spot_CP(t-12)	0	0.01	0.28	0.53	0.14	0	0.02
spot_CP(t-13)	0	0.02	0	0.5	0.09	0	0
spot_CP(t-14)	0	0.07	0.27	0.44	0.21	0.15	0.32
spot_CP(t-15)	0	0.11	0.12	0.54	0.17	0	0.23
spot_CP(t-16)	0	0.25	0.17	0.61	0.23	0.02	0.35
spot_CP(t-17)	0	0.02	0.31	0.24	0.12	0.1	0.06
spot_CP(t-18)	0	0.11	0	0.78	0.2	0.16	0.15
spot_CP(t-19)	0	0	0.28	0.44	0.14	0	0.12
spot_CP(t-2)	0	0.06	0.31	0.56	0.16	0	0.02
spot_CP(t-20)	0	0	0.29	0.41	0.15	0.07	0.13
spot_CP(t-3)	0	0.05	0.09	0.26	0.09	0.04	0.09
spot_CP(t-4)	0	0.09	0.12	0.57	0.16	0.03	0.16
spot_CP(t-5)	0	0.02	0.17	0.74	0.17	0.04	0.05
spot_CP(t-6)	0	0.06	0.06	0.42	0.12	0.07	0.13
spot_CP(t-7)	0	0.06	0	0.48	0.1	0	0.07
spot_CP(t-8)	0	0	0.19	0.42	0.14	0.14	0.1
spot_CP(t-9)	0	0	0.06	0.23	0.05	0	0.04
spot_FEI(t)	0	0.51	0.04	0.57	0.26	0.05	0.37
spot_FEI(t-1)	0	0.02	0.04	0.55	0.12	0.03	0.06
spot_FEI(t-10)	0	0.03	0.14	0.43	0.13	0.15	0.04
spot_FEI(t-11)	0	0	0.23	0.38	0.14	0.14	0.11
spot_FEI(t-12)	0	0.03	0.09	0.42	0.11	0.03	0.09
spot_FEI(t-13)	0	0.06	0.2	0.43	0.14	0.16	0.01
spot_FEI(t-14)	0	0.02	0.11	0.43	0.11	0	0.08
spot_FEI(t-15)	0	0.02	0.12	0.33	0.08	0	0.01
spot_FEI(t-16)	0	0.14	0.41	0.43	0.23	0.14	0.28
spot_FEI(t-17)	0	0.06	0.06	0.38	0.11	0	0.16
spot_FEI(t-18)	0	0.11	0.23	0.23	0.15	0.16	0.17
spot_FEI(t-19)	0	0.02	0.12	0.41	0.09	0	0.01
spot_FEI(t-2)	0	0.03	0.11	0.54	0.11	0	0.01
spot_FEI(t-20)	0	0	0.22	0.5	0.13	0	0.06
spot_FEI(t-3)	0	0.06	0.29	0.36	0.15	0	0.18
spot_FEI(t-4)	0	0.07	0.05	0.37	0.12	0.16	0.07
spot_FEI(t-5)	0	0.02	0.05	0.69	0.13	0.01	0.01
spot_FEI(t-6)	0	0.08	0.34	0.28	0.16	0	0.26
spot_FEI(t-7)	0	0.01	0.06	0.47	0.09	0	0.02
spot_FEI(t-8)	0	0.03	0.13	0.46	0.13	0	0.15
spot_FEI(t-9)	0	0.03	0.21	0.39	0.11	0	0.05
spot_NWE(t)	0.21	0.9	0.53	0.96	0.72	0.97	0.73
spot_NWE(t-1)	0	0.05	0.03	0.51	0.11	0	0.05
spot_NWE(t-10)	0	0.05	0.09	0.1	0.05	0.03	0.01
spot_NWE(t-11)	0	0	0.02	0.35	0.07	0	0.03
spot_NWE(t-12)	0	0.03	0.04	0.27	0.07	0	0.07
spot_NWE(t-13)	0	0.06	0.09	0.44	0.12	0.1	0.04
spot_NWE(t-14)	0	0.03	0.01	0.38	0.08	0	0.09
spot_NWE(t-15)	0	0.04	0.02	0.52	0.1	0	0.02
spot_NWE(t-16)	0	0.1	0.14	0.6	0.17	0.03	0.16
spot_NWE(t-17)	0	0.04	0.06	0.32	0.09	0.02	0.1
spot_NWE(t-18)	0	0.08	0.19	0.38	0.13	0	0.14
spot_NWE(t-19)	0	0.03	0.07	0.53	0.11	0	0.04
spot_NWE(t-2)	0	0.03	0.18	0.64	0.15	0.03	0.04
spot_NWE(t-20)	0	0	0.06	0.49	0.09	0	0.01
spot_NWE(t-3)	0	0.06	0.08	0.37	0.14	0.2	0.15
spot_NWE(t-4)	0	0.05	0.11	0.44	0.11	0.01	0.06
spot_NWE(t-5)	0	0.03	0.01	0.31	0.08	0.05	0.06
spot_NWE(t-6)	0	0.11	0.27	0.35	0.17	0.02	0.27
spot_NWE(t-7)	0	0.01	0.06	0.37	0.08	0	0.02
spot_NWE(t-8)	0	0.03	0.09	0.58	0.17	0.18	0.15
spot_NWE(t-9)	0	0.02	0.04	0.57	0.11	0.01	0.01



	Lasso	Linear Corr.	Linear Reg.	MIC	Mean Score	RF	Ridge Reg.
std_global(t)	0	0	0.19	0.27	0.08	0	0.01
std_global(t-1)	0	0.02	0.24	0.44	0.13	0	0.1
std_global(t-10)	0	0	0.07	0.46	0.1	0.01	0.08
std_global(t-11)	0	0	0.03	0.5	0.1	0	0.06
std_global(t-12)	0	0.01	0.04	0.24	0.07	0	0.11
std_global(t-13)	0	0.01	0.08	0.61	0.12	0.03	0.01
std_global(t-14)	0	0	0.17	0.53	0.13	0	0.07
std_global(t-15)	0	0	0.2	0.4	0.11	0	0.08
std_global(t-16)	0	0.05	0.14	0.54	0.17	0.16	0.14
std_global(t-17)	0	0.02	0.05	0.31	0.09	0	0.13
std_global(t-18)	0	0	0.03	0.56	0.11	0	0.1
std_global(t-19)	0	0	0.05	0.37	0.09	0	0.1
std_global(t-2)	0	0	0.04	0.57	0.11	0	0.04
std_global(t-20)	0	0	0.09	0.37	0.11	0.13	0.06
std_global(t-3)	0	0	0.24	0.44	0.14	0.1	0.04
std_global(t-4)	0	0.03	0.14	0.45	0.13	0	0.17
std_global(t-5)	0	0.01	0.03	0.49	0.1	0	0.08
std_global(t-6)	0	0	0.18	0.26	0.08	0	0.01
std_global(t-7)	0	0.01	0.29	0.35	0.12	0.02	0.04
std_global(t-8)	0	0.02	0.44	0.51	0.21	0	0.29
std_global(t-9)	0	0	0.07	0.58	0.11	0	0.01