

Oscar Arne Rosfjord Thorstensen

# Applicability of Machine Learning Algorithms for the Capesize Shipping Segment

Master's thesis in Marine Technology

Supervisor: Bjørn Egil Asbjørnslett

Co-supervisor: Bjørnar Brende Smestad

June 2021



Oscar Arne Rosfjord Thorstensen

# **Applicability of Machine Learning Algorithms for the Capesize Shipping Segment**

Master's thesis in Marine Technology  
Supervisor: Bjørn Egil Asbjørnslett  
Co-supervisor: Bjørnar Brende Smestad  
June 2021

Norwegian University of Science and Technology  
Faculty of Engineering  
Department of Marine Technology







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

## **MASTER THESIS IN MARINE TECHNOLOGY**

**SPRING 2021**

**For stud.techn.**

**Oscar Arne Rosfjord Thorstensen**

### ***Applicability of Machine Learning Algorithms for the Capesize Shipping Segment***

#### **Background**

The world is becoming more digitized by the minute, creating a continuous production of an increasing amount of data. This data can provide valuable information and insight for the development of new technological advancements. A crucial business strategy is to properly exploit this abundant amount of readily accessible data, analyze it, and use it for high-quality decision support. Numerous large corporations already practice methods for this purpose. McKinsey & Company produced an article in November 2020 where they express the necessity for actors in the maritime industry, particularly the bulk and tanker segment, to utilize data-driven insight for decision making. Furthermore, they identify four primary areas to seize opportunities with data-led analytics for shipowners and operators in bulk shipping. These four areas are defined as:

1. *Finding attractive subsectors and niches through insight into end customers,*
2. *Optimizing portfolios based on relative attractiveness and risk level of different vessel classes,*
3. *Improving commercial choices, and*
4. *Operating vessels more effectively.*

For this study, we are primarily interested in the last two key areas. The shipping industry is generally considered to have fallen behind on its digitization process, and these possibilities will most likely require substantial investments. We also recognize that the number of studies and approaches for efficiently employing the Automatic Identification System (AIS) data are rapidly increasing. This study will explore and investigate the opportunities for shipping behavior analysis by utilizing a combination of AIS- and market-relevant data.

The work of the project thesis focused on exploring previously conducted research studies on AIS data and investigated the development of various shipping segments. Given the considerable size and efficiency of the dry bulk sector, a behavioral analysis of this shipping niche was determined to have more considerable potential and applicability. We, therefore, selected it for further investigation in this master thesis. There has not been a study investigating freight rate prediction of the Capesize bulker segment combining AIS and market-relevant data to the candidate's knowledge.



Utilizing machine learning algorithms for modeling complex problems and time-series forecasting to obtain decision-support material is nothing new. Historically, decision-making in the shipping industry has primarily been based on judgment and experience, especially in the bulk shipping sector. There are proven results using existing machine learning models; it is highly possible to outperform traditional methods. This applies particularly when the amount of readily available data increases, which is the case of AIS- and market-relevant data. Previous research studies have been done on the utilization of machine learning algorithms to predict freight rate movements in different segments of the shipping industry. However, a study has not focused on the Capesize bulker segment to the candidate's knowledge.

### **Objective**

The overall objective of this thesis is to investigate the applicability of machine learning algorithms to predict short-term freight rate movements on the C3 route in the Capesize bulker shipping niche. Furthermore, we aim to identify elements that significantly influence this shipping segment.

### **Tasks**

The candidate should cover the following main points:

- a. Conduct a thorough literature review on relevant topics for the problem objective.
- b. Construct and provide a detailed problem description.
- c. Document the methodology used to approach the problem.
- d. Retrieve, pre-process and clean global AIS data for further exploration of the Capesize bulker segment.
- e. Extract and explore patterns in the Capesize bulker segment from global AIS data.
- f. Identify and construct valuable features from AIS- and market-relevant data.
- g. Choose relevant machine learning algorithms to forecast short-term freight rate movements based on multivariate data input.
- h. Evaluate the forecasting results from the employed models of various feature subsets, using statistical metrics and results from benchmark model.
- i. Discuss results and approaches employed in relation to the problem.
- j. Provide a short and concise conclusion of the problem.

### **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.



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

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.

**Supervision:**

Main supervisor: Bjørn Egil Asbjørnslett

Co-supervisor: Bjørnar Brende Smestad

**Deadline: 10.06.2020**

# Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science with specialization in Marine Systems Design at the Department of Marine Technology (IMT). The work presented in this thesis has been carried out during the spring of 2021 at the Norwegian University of Science and Technology (NTNU), and corresponds to 30 ECTs.

The thesis was conducted during the SARS-CoV-2 pandemic. Despite both regional and national restrictions throughout the semester, it has been possible to organize digital guidance sessions thanks to the adaptability of both the supervisor and the Department of Marine Technology.

Trondheim, June 10, 2021  
Oscar A. R. Thorstensen

A handwritten signature in black ink, reading "Oscar A. R. Thorstensen". The signature is written in a cursive style with a prominent flourish at the end.



# Acknowledgment

I would like to thank the following persons for their great help and to express my sincerest gratitude for making this thesis possible:

Professor Bjørn Egil Asbjørnslett, my supervisor, for valuable advice and enriching guidance throughout the work of this thesis.

PhD cand. Bjørnar Brende Smestad, my co-supervisor, for retrieving AIS data from the Norwegian Coastal Administration database and useful insight into AIS data exploitation.

Ingeborg Almås, analyst at Clarksons Platou, for extremely rewarding discussions of the shipping industry and providing me with market-specific data.

I would also like to thank my office mates for helpful discussions and creating an educationally rewarding environment at Tyholt.

Finally, I would like to thank my family for their support, and in particular my father and sister for their valuable feedback on the report.

# Abstract

The thesis investigates the applicability of selected machine learning algorithms to predict short-term freight rate movements on the Capesize route from Tubarao to Qingdao (C3). The dry bulk shipping segment was selected for this exercise in consideration of its market size, potential and applicability to the performance of a behavioral study. Input from Automatic Identification System (AIS) and market-relevant data has been employed in attempts to identify significant elements (features) influencing Capesize bulk operations. A comprehensive literature review was conducted for benchmark topic relevance to the problem objective. The study has used AIS data retrieved from the Norwegian Coastal Administration (NCA) database as well as market derived information provided by Clarksons Platou. Several methods were explored in the process of extracting and constructing relevant features from the different data sources. In attempts to determine the more important influencing elements, different feature selection methods were utilized. The study considered three different machine learning models; (1) the linear ridge regression (LRR) model, (2) the random forest regression (RFR) model, and (3) the long short-term memory (LSTM) model. The persistence model was utilized for comparison purposes of the forecasting results and to provide a benchmark performance level. Lastly, the study employed a *grid search* method to analyze and describe the optimal combination of hyperparameters for the various employed machine learning algorithms. The results of the conducted investigation indicate that all selected models employed in this thesis show capabilities in predicting short-term freight rate movements on the C3 route. Differing sets of features proved influential in development of prediction accuracy in machine learning algorithms. However, results attained provided no definitive conclusions or identification as to which feature specification set that showed greatest market influence. All examined feature sets included a combination of AIS- and market-derived data and consequently supported the objective formulated for this thesis. We can therefore conclude that the employed machine learning models can to some degree predict short-term freight rate movements on the C3 route.

# Sammen drag

Denne avhandlingen undersøker anvendeligheten til utvalgte maskinlæringsalgoritmer for å kunne forutsi kortsiktige bevegelser i fraktrater ved Capesize-ruten mellom Tubarao og Qingdao (C3). Med et utgangspunkt i markedsstørrelse, potensial og anvendelighet for gjennomføring av en atferds studie, valgte vi å se på skipsfartssegmentet for tørr bulk til denne øvelsen. I et forsøk på å identifisere viktige elementer som påvirker Capesize bulk-operasjoner benyttet vi input fra Automatic Identification System (AIS) og annen markedsrelevant data. Det ble gjennomført en omfattende litteraturstudie for å undersøke relevansen av referansetemaet for målet med avhandlingen. Studien evaluerte tre ulike maskinlæringsmodeller; en *linear ridge regression* (LRR) modell, en *random forest regression* (RFR) modell, og en *long short-term memory* (LSTM) modell. En *persistence model* ble brukt til å sammenligne prognoseresultatene og for å etablere et ytelsesnivå. Til sist ble det brukt en *grid search* metode for å analysere og beskrive den mest gunstige kombinasjonen av hyperparametere ved de benyttede maskinlærings algoritmene. Resultatene våre indikerer at de utvalgte modellene viser evne til å kunne forutsi kortsiktige bevegelser i fraktraten til C3-ruten. Alle undersøkte funksjonssett inkluderte en kombinasjon av AIS og markedsavlede data og støttet følgelig målet av avhandlingen. Vi kan derfor konkludere med at maskinlæringsmodellene som ble brukt i denne studien til en viss grad kan forutsi kortsiktige bevegelser i fraktrater ved C3-ruten.

# Table of Contents

<b>Preface</b>	<b>i</b>
<b>Acknowledgment</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Sammendrag</b>	<b>iv</b>
<b>Table of Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xii</b>
<b>Nomenclature</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	2
1.2 Motivation . . . . .	3
1.3 Problem Description . . . . .	3
1.3.1 Objective . . . . .	5
1.4 Thesis outline . . . . .	6
<b>2 Literature Review</b>	<b>7</b>
2.1 Dry bulk shipping . . . . .	7
2.1.1 Freight Rate Modeling . . . . .	8
2.2 AIS applications . . . . .	8
2.2.1 Data handling . . . . .	9
2.2.2 Previous applications with machine learning . . . . .	9
2.3 Data science literature and theory . . . . .	10
2.3.1 Exploratory data analysis . . . . .	10

2.3.2	Data mining . . . . .	11
2.3.3	Machine learning . . . . .	12
2.3.3.1	Building a machine learning model . . . . .	12
2.3.3.2	Supervised, unsupervised or semi-supervised . . . . .	13
2.3.4	Algorithm selection . . . . .	13
<b>3</b>	<b>Methodological Approach</b>	<b>15</b>
3.1	Exploratory data analysis . . . . .	15
3.2	Feature engineering . . . . .	18
3.2.1	Feature construction and extraction . . . . .	18
3.2.1.1	Vessel capacity count in world regions and port locations . . . . .	18
3.2.1.2	Fleet percentage in world regions . . . . .	21
3.2.1.3	Active vessels and fleet capacity . . . . .	22
3.2.1.4	Fleet utilization . . . . .	23
3.2.1.5	Price and market features . . . . .	23
3.2.2	Data preparation . . . . .	26
3.2.3	Feature selection . . . . .	29
3.3	Algorithm selection . . . . .	31
3.3.1	Linear ridge regression . . . . .	31
3.3.2	Random forest regressor . . . . .	33
3.3.3	Long short-term memory . . . . .	35
3.3.4	Model tuning with hyperparameter optimization . . . . .	38
3.4	Model evaluation method . . . . .	40
3.4.1	Baseline model . . . . .	40
3.4.2	Statistical modeling metrics . . . . .	41
<b>4</b>	<b>Computational Study</b>	<b>43</b>
4.1	The bulk shipping case . . . . .	43
4.1.1	Commodity types . . . . .	44
4.1.2	Classification of vessel types . . . . .	45
4.1.3	Route segments . . . . .	46
4.1.4	One-step-ahead forecasting . . . . .	46
4.2	Description of raw data . . . . .	48
4.2.1	Capesize Bulker Data . . . . .	48
4.2.1.1	Vessel type classification data . . . . .	48
4.2.1.2	Historical time-series data . . . . .	49
4.2.2	AIS Data . . . . .	50
4.2.2.1	Message content and frequency . . . . .	51
4.2.2.2	Data extraction . . . . .	52
4.2.2.3	Data assembly and preprocessing . . . . .	53
4.3	Results . . . . .	55
4.3.1	Benchmark model . . . . .	55
4.3.2	Feature selection . . . . .	55
4.3.3	Hyperparameter optimization . . . . .	56
4.3.4	Model training . . . . .	61

4.3.5	Model forecasting . . . . .	62
4.3.6	Train vs test performance . . . . .	67
<b>5</b>	<b>Discussion</b>	<b>71</b>
5.1	Evaluation of forecasting results . . . . .	71
5.2	Evaluation of methodological approach . . . . .	73
5.2.1	Hyperparameter optimization technique . . . . .	73
5.2.2	Employed feature selection methods . . . . .	74
5.2.3	Feature construction process . . . . .	74
5.3	Limitations and considerations . . . . .	75
<b>6</b>	<b>Conclusion</b>	<b>77</b>
6.1	Further work . . . . .	78
	<b>Bibliography</b>	<b>79</b>
<b>A</b>	<b>AIS Data Contents</b>	<b>83</b>
<b>B</b>	<b>Descriptive Statistics of Features</b>	<b>85</b>
<b>C</b>	<b>Augmented Dickey-Fuller Results</b>	<b>89</b>
<b>D</b>	<b>Feature Importance Scores</b>	<b>91</b>
<b>E</b>	<b>Python Scripts</b>	<b>97</b>
E.1	Description of Python scripts . . . . .	97
E.2	project_settings.py . . . . .	98
E.3	AIS SQL Script . . . . .	99
E.4	data_processing.py . . . . .	101
E.5	clarksons.py . . . . .	105
E.6	polygons.py . . . . .	109
E.7	FE.py . . . . .	115
E.8	data_preparation.py . . . . .	123
E.9	feature_importance_score.py . . . . .	126
E.10	feature_selection.py . . . . .	128
E.11	ML_models.py . . . . .	132



# List of Figures

1.1	Baltic Exchange Capesize Index . . . . .	4
1.2	Illustration of Anthony’s framework . . . . .	5
2.1	Summary of studies on the applications of AIS data . . . . .	9
2.2	Data Science Venn Diagram . . . . .	10
2.3	Gartner’s analytics maturity model . . . . .	11
2.4	Comparison of different learning styles . . . . .	13
2.5	Decision tree for algorithm selection process . . . . .	14
3.1	Scatter plot of worldwide AIS Capesize recordings . . . . .	16
3.2	Density plot of worldwide AIS Capesize recordings . . . . .	16
3.3	Map of major Capesize routes and port locations . . . . .	17
3.4	Velocity histogram of worldwide AIS recordings . . . . .	17
3.5	Orientation of world polygons . . . . .	19
3.6	Orientation of port polygons . . . . .	19
3.7	Zoomed example of a port polygon (Tubarao) . . . . .	20
3.8	Vessel distribution over time in world regions . . . . .	21
3.9	Capacity distribution of the world polygons . . . . .	21
3.10	Comparison of different vessel count features on global scale . . . . .	22
3.11	Comparison of different sum of fleet capacity features on global scale . . . . .	23
3.12	Capesize bulker fleet utilization factor . . . . .	24
3.13	Historical spot rate progression of selected Capesize routes . . . . .	24
3.14	Historical progression of the BCI and BDRY features . . . . .	25
3.15	BCI data transformation and normalization . . . . .	28
3.16	Training and test set split . . . . .	28
3.17	Simple decision tree example . . . . .	34
3.18	Random forest algorithm structure . . . . .	35
3.19	MLP with 3 hidden layers and $n$ input features . . . . .	35
3.20	Composition of neuron $i$ in the first hidden layer of an MLP . . . . .	36
3.21	Unrolled recurrent neural network . . . . .	37



3.22	Structure of the repeating module in an LSTM model . . . . .	38
4.1	Vessel Size Groups (in deadweight tons) . . . . .	45
4.2	Spot rate C3 route . . . . .	46
4.3	Historical plot of two different spot rates . . . . .	50
4.4	Overview of AIS applications in maritime research . . . . .	51
4.5	Velocity histogram of worldwide AIS recordings (>25 knots) . . . . .	54
4.6	Persistence model forecasting results . . . . .	55
4.7	Grid search scores for linear ridge regression model . . . . .	58
4.8	Grid search scores for random forest regression model . . . . .	59
4.9	Grid search scores for long short-term memory model . . . . .	60
4.10	Linear ridge regression model forecast for different feature subsets . . . . .	64
4.11	Random forest regression model forecast for different feature subsets . . . . .	65
4.12	Long short-term memory model forecast for different feature subsets . . . . .	66
4.13	RMSE performance measurements utilizing all features . . . . .	67
4.14	RMSE performance measurements utilizing top features . . . . .	68
4.15	RMSE performance measurements utilizing selected features . . . . .	69
5.1	Copy of figure 4.8c . . . . .	73
D.1	Top 20 most important features based on random forest regressor model performance . . . . .	91
D.2	Cumulative feature importance score with respect to number of fea- tures . . . . .	92
D.3	Top 20 features based on mean feature importance score . . . . .	92
D.4	Feature importance scores of selected algorithms (1/3) . . . . .	93
D.5	Feature importance scores of selected algorithms (2/3) . . . . .	94
D.6	Feature importance scores of selected algorithms (3/3) . . . . .	95

# List of Tables

1.1	Comparison of planning and decision levels . . . . .	4
3.1	Overview of engineered features for final dataset . . . . .	25
3.2	Univariate and multivariate filter methods to identify top $n$ features	30
3.3	Overview of selected activation functions for ANN [33] . . . . .	37
3.4	Overview of optimized hyperparameters in selected machine learning algorithms . . . . .	39
4.1	Vessel size groups according to commonly used shipping terminology	45
4.2	Overview of dry bulk shipping routes . . . . .	47
4.3	Vessel type and size indication, with corresponding aggregated data from Clarksons Research Services database . . . . .	48
4.4	Correlation matrix for time charter and trip charter rates . . . . .	49
4.5	Descriptive statistics of correlation matrix in Table 4.4 . . . . .	50
4.6	AIS data types . . . . .	52
4.7	Reporting intervals of dynamic AIS data . . . . .	52
4.8	AIS ship types . . . . .	53
4.9	Structure of the original data retrieved from the AIS database . . . . .	53
4.10	Structure of the post-processed AIS data . . . . .	53
4.11	AIS data file comparison pre-, peri- and post-processing . . . . .	54
4.12	Composition of different feature combination sets . . . . .	56
4.13	Optimal hyperparameter combinations . . . . .	56
4.14	Performance metrics on training set for all models, with the employed data scaled in domain $[0,1]$ . . . . .	61
4.15	Performance metrics on test set for all models, with the employed data scaled in the domain $[0,1]$ . . . . .	62
4.16	Performance metrics on forecast results for all models . . . . .	63
B.1	Descriptive statistics of count features in port locations . . . . .	85
B.2	Descriptive statistics of capacity features in port locations . . . . .	86

B.3	Descriptive statistics of count and capacity features in world regions	86
B.4	Descriptive statistics fleet percentage features in world regions . . .	87
B.5	Descriptive statistics of fleet count and capacity features . . . . .	87
B.6	Descriptive statistics of fleet utilization feature . . . . .	87
B.7	Descriptive statistics of market and price derived features . . . . .	87

# Nomenclature

ADF	=	Augmented Dickey-Fuller
AIS	=	Automatic Identification System
ANN	=	Artificial Neural Network
BCI	=	Baltic Exchange Capesize Index
BDRY	=	Breakwave Dry Bulk Shipping ETF
BPPT	=	Backpropagation through time
CNN	=	Convolutional Neural Network
DT	=	Decision Tree
EDA	=	Exploratory Data Analysis
ETF	=	Exchange-Traded Fund
GAM	=	Generalized Additive Models
IMO	=	International Maritime Organisation
KDD	=	Knowledge Discovery in Databases
KNN	=	K-Nearest Neighbour
LRR	=	Linear Ridge Regression
LSTM	=	Long Short-Term Memory
MAE	=	Mean Absolute Error
MAPE	=	Mean Absolute Percentage Error
ML	=	Machine Learning
MLP	=	Multilayer Perceptron
MLR	=	Multiple Linear Regression
NCA	=	Norwegian Coastal Administration
RFE	=	Recursive Feature Elimination
RFR	=	Random Forest Regressor
RMSE	=	Root Mean Squared Error
RNN	=	Recurrent Neural Network
SIN	=	Shipping Intelligence Network
SLR	=	Simple Linear Regression
SVM	=	Support Vector Machine
VHF	=	Very High Frequency



# Chapter 1

## Introduction

The maritime shipping and transport industry, and in particular the dry bulk segments, has long been considered a prime example of a perfectly competitive and efficient market (Norman [1]). According to Hayes [2], a perfectly competitive market requires the satisfaction of the following criteria;

- *Companies sell identical products*
- *Market share does not influence price*
- *Companies are able to enter or exit without barrier*
- *Buyers have perfect information*
- *Companies cannot determine prices*

In a paper by Adland et al. [3], concerns regarding the shipping industry's market efficiency at a micro-level are raised. They identify various elements that indicate a flawed hypothesis of a perfectly competitive shipping market.

Meanwhile, the maritime shipping industry includes a multitude of important stakeholders, such as shipowners, charterers, classification societies, and shipyards, to name a few; is by its very nature the most global industry and effects all citizens in all nations throughout the world, but has yet to gainfully employ the vast amounts of historical data available through advanced technological modeling and analysis. This paradox must surely be about to change as we see the enormous advances in other industries who have successfully employed big data analysis. This paper is perhaps a small step towards addressing this obvious flaw.

## 1.1 Background

The world is becoming increasingly digitized, more by the minute, creating a continuous production of an increasing amount of data. This data can provide valuable information and insight for the development of new technological advancements. A crucial business strategy is to properly exploit this abundant amount of readily accessible data, analyze it, and use it for high-quality decision support. Numerous large corporations already practice methods for this purpose. Jie et al. [4] at McKinsey & Company produced an article in November 2020 where they express the necessity for actors in the maritime industry, particularly the bulk and tanker segments, to utilize data-driven insight to support decision making. In addition, they identify four key areas where opportunity for application of data-led analytics exists:

1. *finding attractive subsectors and niches through insight into end customers,*
2. *optimizing portfolios based on relative attractiveness and risk level of different vessel classes,*
3. *improving commercial choices, and*
4. *operating vessels more effectively.*

Our discussion is focused primarily towards the last two key fields.

Despite its global reach and fundamental nature, the shipping industry is generally considered to have fallen behind the general digitization process. In all likelihood this is due to challenging market conditions at a time when such digitization requires substantial investments. However, we recognize that the number of studies and approaches for efficiently utilizing the Automatic Identification System (AIS) data are rapidly increasing. This study will explore and investigate the opportunities for predicting shipping behavior by employing a combination of AIS- and market-relevant data analysis.

In the preliminary project presentation, the candidate explored previously conducted research studies on AIS data and investigated the development of various shipping segments. Given the considerable size and efficiency of the dry bulk sector, a behavioral analysis of this shipping niche was determined to have more potential and applicability, and it was consequently selected for further investigation in this master thesis. Indeed, to the candidates knowledge there has not been a study investigating freight rate prediction of the Capesize bulker segment combining AIS and market-relevant data.

Utilizing machine learning models for modeling complex problems and time-series forecasting to obtain decision-support material is nothing new. Historically, decision-making in the shipping industry has primarily been based on judgment and experience, especially in the bulk shipping sector. There are numerous results showing that with the use of existing sophisticated machine learning models, there is a significant likelihood to outperform traditional methods. This applies particularly when the amount of readily available data increases, which is the case of AIS-

and market-relevant data. There have been previous studies on the utilization of machine learning models for the prediction of freight rate movements in different segments of the shipping industry ([5], [6]), but none focused on the Capesize bulker segment.

## 1.2 Motivation

The primary motivation for investigating these opportunities is to produce new and gainful insight into the maritime shipping and transportation industry. Initially, the focus of this thesis was to examine extreme changes in the bulker segment and in particular, to concentrate on the impact of various global and regional crises. Considering that several significant situations have had great effect on the financial markets, particularly the energy prices, the idea of gaining insight into how these elements have affected the bulk market was an exciting starting point. Such understanding might provide information that will contribute to developing new methods for predicting the consequential impact of future calamities. Since however, the bulk segment in world shipping and trade encompasses an unfathomable amount of influencing elements, it is undeniably difficult to single out significant factors from historical crises that have had an identifiable effect on the market development of the bulker segment.

Rather than identifying crises specific elements, we therefore aim to identify factors influencing the Capesize dry bulk sector in general, with the combined use of AIS- and market-relevant data. The Capesize bulker segment is a highly volatile market, and the opportunity of retrieving helpful insight into this sector is highly motivating. Moreover, the possibility of utilizing AIS data to provide added predictive value in anticipating significant fluctuations in this sector is essential for the long-term profitability for both shipowners and operators in this market. Figure 1.1 properly illustrates the volatile Baltic Exchange Capesize Index (BCI). We immediately observe that the index plummets to a negative value after a period of more than 300 weeks, which is the first time the index has dropped into negative territory [7]. Numerous influences were among the root causes of this historical dip; amongst them seasonality, flooding in Brazil, and undoubtedly the most significant, the global coronavirus outbreak. With China being the largest importer of dry bulk commodities, accounting for roughly 40% of the market<sup>1</sup>, we observed the significant impact of a locked-down China in the aftermath of the outbreak of the SARS-CoV-2 pandemic.

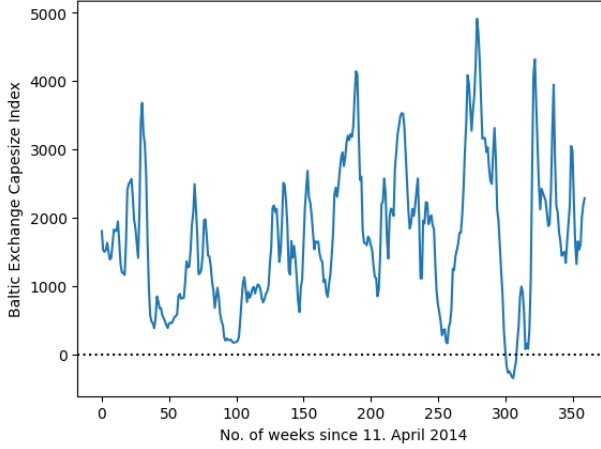
## 1.3 Problem Description

The model presented in Figure 1.2 is a commonly employed framework to categorize different planning levels, namely *strategic*-, *tactical*- and *operational*-planning. The triangular model is based on Robert Anthony's fundamental beliefs that companies

---

<sup>1</sup><https://www.hellenicshippingnews.com/chinas-import-surge-drives-optimism-in-dry-bulk-shipping-demand/>





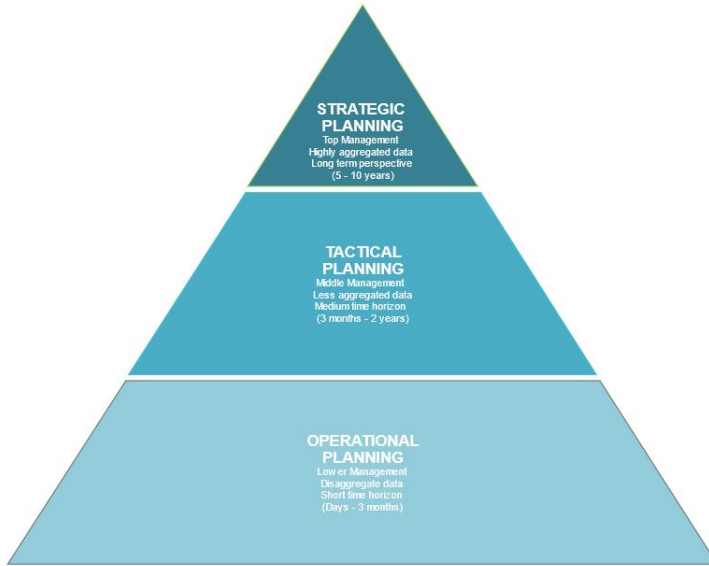
**Figure 1.1:** Baltic Exchange Capesize Index

and organizations are constructed as a hierarchy of decision-making levels. Top-level decisions are considered strategic and are seldom very specific, while the further down the hierarchy we descend, the more detailed and operation-specific the decisions become. Since we aim to predict short-term freight rate movements, this thesis falls into the category of operational planning. A more detailed comparison of the different planning levels is provided in Table 1.1.

Among the most critical elements of any data science project is to gain a complete business understanding of the addressed problem. Failure to properly understand the problem will result in less efficient model scoping that in turn will not produce the desired outcomes for further evaluation. The below section of this paper will therefore also describe the objectives and approaches selected to address the problems encountered.

**Table 1.1:** Comparison of planning and decision levels

<b>Factor</b>	<b>Strategic Planning</b>	<b>Tactical Planning</b>	<b>Operational Planning</b>
<i>Purpose</i>	Management of change, resource acquisition	Resource utilization	Execution, evaluation, and control
<i>Implementation instruments</i>	Policies, objectives, capital investments	Budgets	Procedures, reports
<i>Planning horizon</i>	Long	Medium	Short
<i>Scope</i>	Broad, corporate level	Medium, plant level	Narrow, job shop level
<i>Level of management involved</i>	Top	Middle	Low
<i>Frequency of re-planning</i>	Low	Medium	High
<i>Source of information</i>	Largely external	External and internal	Largely internal
<i>Level of aggregation of information</i>	Highly aggregated	Moderately aggregated	Detailed
<i>Required accuracy</i>	Low	Medium	High
<i>Degree of uncertainty</i>	High	Medium	Low
<i>Degree of risk</i>	High	Medium	Low



**Figure 1.2:** Illustration of Anthony's framework

### 1.3.1 Objective

The central objective of this thesis is to investigate the applicability of sophisticated machine learning algorithms to predict short-term freight rate movements on the Capesize route from Tubarao to Qingdao (C3). In continuation, we aim to identify critical elements that significantly influence this shipping segment.

To best approach these objectives, we intend to utilize data retrieved from the Norwegian Coastal Administration (NCA) AIS database in combination with freight market data provided from Clarksons Platou and Breakwave Dry Bulk Shipping exchange-traded fund data. With such extensive datasets available, we are presented with a challenge to construct, evaluate and determine which features and combinations of features that are most influential and relevant for various machine learning algorithms to produce the best possible predictions. It is therefore critical to study and analyze all available data thoroughly. Following this, we have addressed the following research objectives in this thesis:

- a *Conduct a thorough literature review on relevant topics for the problem objective.*
- b *Construct and provide a detailed problem description.*
- c *Document the methodology used to approach the problem.*
- d *Retrieve, pre-process and clean global AIS data for further exploration of the Capesize bulker segment.*

- e *Extract and explore patterns in the Capesize bulker segment from global AIS data.*
- f *Identify and construct valuable features from AIS- and market-relevant data.*
- g *Choose relevant machine learning algorithms to forecast short-term freight rate movements based on multivariate data input.*
- h *Evaluate the forecasting results from the employed models of various feature subsets, using statistical metrics and results from benchmark model.*
- i *Discuss results and approaches employed in relation to the problem.*
- j *Provide a short and concise conclusion of the problem.*

## 1.4 Thesis outline

The thesis is structured in the following order:

**Chapter 2:** Reviews literature for benchmark topic relevance to the problem objective.

**Chapter 3:** Provides insight to the employed methods for feature engineering, algorithm selection and model evaluation.

**Chapter 4:** Presents the obtained raw data, in addition to the produced results from the benchmark and machine learning models.

**Chapter 5:** Discusses the produced results and the methodology used to approach the problem.

**Chapter 6:** Provides a short and concise conclusion and presents further work.

# Literature Review

This part of the paper will provide insight into previously conducted research and applications reviewed as part of the work performed for this master thesis. The articles presented in this section have given a better understanding of influencing elements in dry bulk shipping and currently utilized AIS data application approaches. They also provide better insight into suitable methods commonly employed in data science. A notable portion of the literature review was conducted as part of the project thesis. However, all the literature combined sets the foundation for further work in this thesis.

## 2.1 Dry bulk shipping

Adland et al. [3] conduct a study where they propose a model to extract freight rate information in individual contracts from the transportation of crude oil and dry bulk commodities. Their study's purpose is twofold; (1) to expand already existing models on microeconomic determinants on freight rates to account for relationship effects between buyers and sellers, and (2) investigate the influence of these relationships on the freight rates for individual fixtures empirically. The study is meant to assess buyers' and sellers' impact on fluctuations in freight rates. According to their findings, market conditions and routes remain the most important covariates. However, they also acknowledge the significant role of charterers and shipowners and their influence in individual contracts.

Alizadeh and Talley [8] utilize a large sample of individual dry bulk charter contracts to investigate several important aspects of the dry bulk shipping market. They primarily study the microeconomic determinants of freight rates while simultaneously examining; (1) how the freight rates vary with regards to major dry bulk routes, (2) how shipping activities are distributed geographically, and (3) the laycan period duration in shipping contracts. Their conclusions indicate a strong

correlation between dry bulk freight rates and laycan periods. Furthermore, they identify voyage routes, vessel deadweight, and age as significant and influential determinants of the dry bulk shipping freight rates.

Köhn and Thanopoulou [9] proposes a methodology to assess the non-linearity nexus between charter rates and their determinants in dry bulk shipping. They utilize semi-parametric methods to construct various generalized additive models (GAMs) to examine different factors influencing the physical time-charter rates. The paper aims to recognize general market trends and further explain the resulting variations on physical T/C rates. According to their findings, both vessel and fixture-specific traits are revealed to impact time-charter rates for different ships.

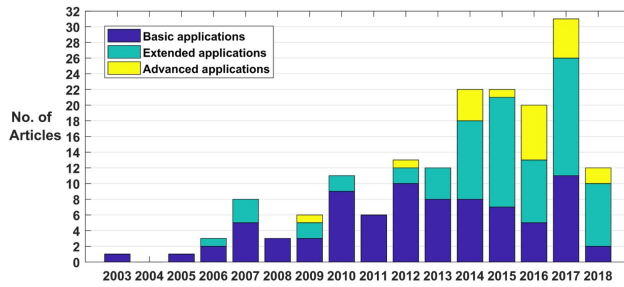
### 2.1.1 Freight Rate Modeling

The purpose of the study by Århus and Salen [5] is twofold; (1) utilize already developed machine learning techniques to predict future shipping freight rates in the crude oil tanker market, and (2) examine the predictiveness of employing satellite AIS data. They combine the use of AIS-derived information with non-AIS-derived data. Furthermore, they attempt several experiments with various forecasting horizons and complexity levels to evaluate the model's accuracy. Their findings indicate that for predicting the freight rates in the tanker market, AIS-derived data does not provide any significant additional value. It will be interesting to compare the impact of AIS-derived information on the tanker market with the dry bulk segment.

## 2.2 AIS applications

The study conducted by Yang et al. [10] illustrates applications for AIS data in marine research. Their paper demonstrates the rapid growth in the utilization of AIS data applications. The authors have identified three main categories for these applications: basic applications, extended applications, and advanced applications. Figure 2.1, extracted from their study, illustrates the division of these categories. Furthermore, they also identify a list of the major categories for methods used with AIS applications. Data mining, causality analysis, and operational research are respectively fascinating and relevant for further investigation. The information and findings from this study have provided a more comprehensive understanding of the shipping market dynamics. Additionally, it has helped to provide better insight into what elements to research for more significant multi-disciplinary studies with AIS data in the center.

Adland et al. [11] investigate the accuracy of trade volume estimates in the shipping industry based on AIS data combined with a detailed crude oil shipments database. More specifically, the study strictly only applies to shipping segments where the cargo type is observable and homogenous due to limitations for AIS-based estimates of trade volumes. According to their research, utilization of AIS-derived data to determine total exported quantities provide somewhat good alignment to the aggregated customs-based export numbers. However, when examining the exported



**Figure 2.1:** Summary of studies on the applications of AIS data

values at more micro-levels, the estimates become less accurate with more unstable deviations of the exported volumes.

Yan et al. [12] present their study of the global marine oil trade, as a combination of traffic route analysis, trade volume analysis, and trade network analysis, based on AIS data. While Adland et al. [11] focuses on determining the level of precision for the trade volume estimates, this study aims to construct a framework for estimating the trade volume.

### 2.2.1 Data handling

Brende Smestad et al. [13] presents heuristic methods for identifying vessel types using AIS-data. The study intends to predict ship types with a high level of accuracy and demonstrate the unnecessary purchasing of additional information from commercial databases, thereby avoiding additional costs. The paper provides a detailed and thorough preprocessing of the AIS database, which is essential to prevent inaccurate data in the heuristics. According to the final results, the developed heuristics provide highly accurate predictions compared to data from the Clarksons Ship Register.

### 2.2.2 Previous applications with machine learning

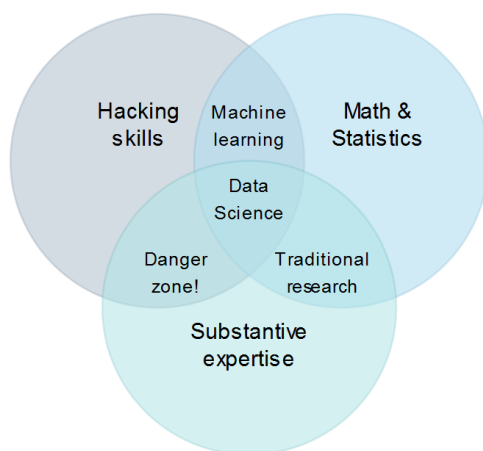
Chen et al. [14] conducts a ship movement classification analysis with the combination of AIS data with machine learning algorithms. Their study focuses on the use of Convolutional Neural Network (CNN) for ship movement classification. However, they also compare the results with other commonly utilized algorithms such as K-Nearest Neighbours (KNN), Support Vector Machine (SVM), and Decision Tree (DT). The results from the study indicate that the use of CNN provides better performance for the classification of AIS data.

Århus and Salen [5] conduct a fascinating study with a twofold purpose, namely to (1) apply machine learning techniques to predict shipping freight rates and (2) to investigate the possibilities for prediction with the use of AIS data. Furthermore, they describe the process of transforming raw AIS data into usable time-series

data and identifying relevant features and crucial non-AIS-derived data elements. A detailed description of their methodology for constructing the machine learning program is also presented in the paper.

## 2.3 Data science literature and theory

Data science's primary purpose is to provide solutions to real business problems by utilizing available data resources. Figure 2.2 illustrates the essential elements that combined results in data science and is extracted from Conway [15]. The hacking skills represent computer science, data engineering, and programming, while math and statistics knowledge define the necessary numerical techniques and algorithms to derive insight. Lastly, the substantive expertise element means to represent the necessity for domain knowledge and business value.

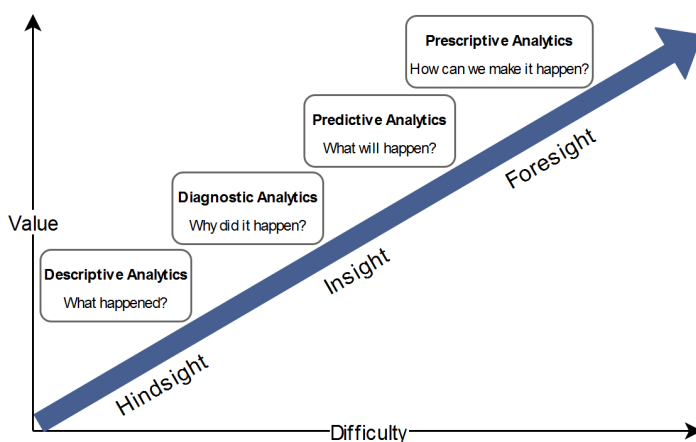


**Figure 2.2:** Data Science Venn Diagram

According to Anadiotis [16], we can classify data analytics applications into the following categories: *Descriptive Analytics*, *Diagnostic Analytics*, *Predictive Analytics*, and *Prescriptive Analytics*. The different classifications present different levels of complexity with a corresponding level of business value that is achievable. Figure 2.3 properly demonstrates the correlation between difficulty and benefit for the different classifications.

### 2.3.1 Exploratory data analysis

Exploring the data is commonly the first step in data science projects. The Exploratory Data Analysis (EDA), according to Bruce et al. [17], includes a set of approaches and techniques used for examining datasets and provides a summary of the main features. Tukey [18] defines EDA as "*Procedures for analyzing data*,



**Figure 2.3:** Gartner’s analytics maturity model

*techniques for interpreting the results of such procedures, ways of planning the gathering of data to make its analysis easier, more precise or more accurate, and all the machinery and results of (mathematical) statistics which apply to analyzing data.”*

Data visualization tools are essential in exploratory analysis. They intend to provide insight into trends and patterns and visually illustrate how they are correlated using different plots. Commonly used plots for this type of analysis include but are not limited to box plots, scatter plots, and histograms.

### 2.3.2 Data mining

Twin [19] defines data mining as converting raw data into useful information, which is done by discovering and recognizing patterns and structures in the datasets. They also mention that the process is highly dependent on computer processing capabilities and the level of efficiency for data collection.

Mannila [20] identifies and presents five steps as part of the knowledge discovery in databases (KDD) process, more commonly known as data mining. These steps are listed below. Furthermore, they consider some data mining methods for pattern recognition and demonstrate possible applications of these techniques.

1. *understanding the domain,*
2. *preparing the data set,*
3. *discovering pattern (data mining),*
4. *postprocessing of discovered patterns, and*
5. *putting the results into use.*



### 2.3.3 Machine learning

The purpose of Machine learning (ML) is to allow computers to continuously improve their performance regarding decision-making or predictive accuracy based on their previous experience [21]. In general, we consider ML as a branch or subfield of artificial intelligence. ML utilizes algorithms and statistical modeling combined with feature data to create predictions or make decisions without being explicitly programmed.

As mentioned previously, it is common to utilize visualization tools as part of the exploratory analysis; the same applies to the start of an ML study. A correlation matrix is an example of this type of instrument that intends to identify the impact the dataset's different features have on the target variable. Therefore, this method is widespread for feature selection in ML studies.

#### 2.3.3.1 Building a machine learning model

Before starting to work on building an ML model, it is essential to obtain a comprehensive business understanding. This step might seem obvious; however, the data scientist must understand the problem correctly to construct the best possible system. The next step in creating an ML model is to examine the data that comprise the available raw material, forming the final feature set's foundation. For this step, it is vital to verify the data provided and investigate the applied processing techniques.

Following the proper understanding of available data material, it is necessary to prepare and define training and test sets. The process of preparing the datasets can be quite extensive; however, it is crucial to make sure that the selected data for training is clean, properly formatted, and does not contain any imbalances that would impact the practice model.

The fourth step for this method is to determine what algorithm to use for the training model. There are currently numerous developed algorithms for various purposes; therefore, conducting thoroughgoing research when selecting what algorithm to use is of high importance.

After concluding the algorithm and set of features to apply in training, the next step is fitting the model. This part of the approach for building an ML model is an iterative process. This step aims to identify the optimum parameters' values and adjust the algorithm's weights until the model returns satisfying results. The product that produces acceptable outcomes will be the trained algorithm, namely the machine learning model.

The final step is to utilize the ML model on new test data, to solve the proposed business problem. Ideally, the system will improve over time and produce more accurate results regarding the end goal.

### 2.3.3.2 Supervised, unsupervised or semi-supervised

We categorize the different learning styles in ML into three primary categories: Supervised, unsupervised, and semi-supervised. Figure 2.4 presents a clear overview of the different learning types and provides examples of commonly applied algorithms. Additionally, there exists a learning method called reinforcement learning. This technique focuses on learning from interactions with an agent and trial and error methodology. For this study, we will not consider this technique for further use.

	Overview	Process	Subtypes	Subtypes
Supervised	Majority of algorithms Machine is trained using well-labeled data; inputs and outputs are matched	Mapping function takes inputs and matches to outputs, creating a target function	Classification, Regression	Linear regression, Random forest, SVM
Unsupervised	Unlabeled data (inputs only) is analyzed. Learning happens without supervision.	Inputs are used to create a model of the data	Clustering, Association	PCA, k-Means, Hierarchical clustering
Semi supervised	Some data is labeled, some not. Goal: better results than labeled data alone. Good for real world data.	Combination of above processes	All the above	Self training, Mixture models, Semi-supervised SVM

Figure 2.4: Comparison of different learning styles<sup>1</sup>

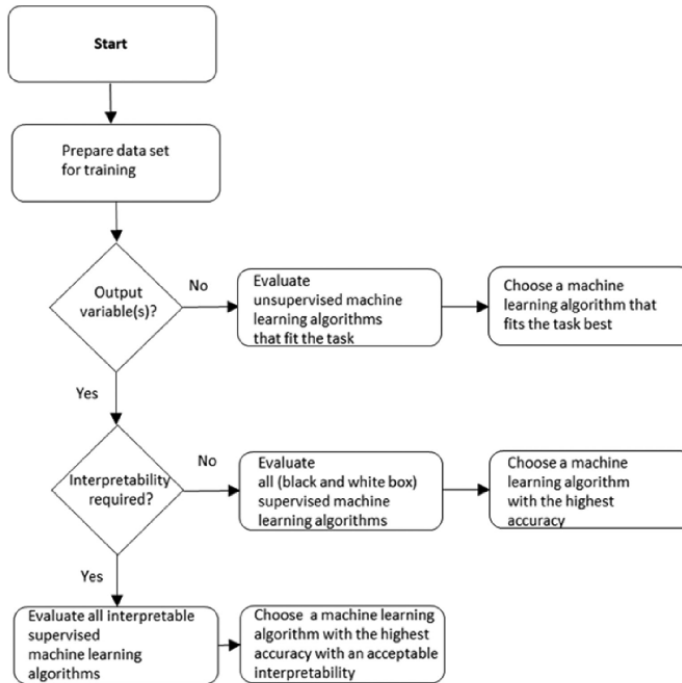
The typical goal of any ML prediction model is to utilize input values  $\mathbf{X}$  to map a function  $\hat{f}$  that returns  $\hat{y} = \hat{f}(X)$  as best possible estimation for the real value  $y$ . For the model to identify said function, it needs to be trained. When we say that the model requires training, it must be given input values to learn from progressively. With supervised learning, both input and output values are known, and the model employs both during the training process. The model does so by first utilizing input values to return a prediction. Secondly, it gets instructed of the true value of what the prediction should have been, after which the system makes adjustments to account for the error. The objective of a supervised learning model is to identify the relationship between the two known values, input and output. While an unsupervised learning model only employs input values as known to discover unknown patterns in the data.

### 2.3.4 Algorithm selection

We previously pointed out in Section 2.3.3.1 the importance of deciding what algorithm to select for the ML model. There exists a wide selection of ML algorithms to choose from, and it is often extremely challenging to determine which will yield the best results to the proposed problem. Lee and Shin [22] address the issues of algorithm selection in ML. As mentioned in their paper, each case comprises different variables and data that influence algorithms' performance. Additionally, they highlight that the main challenge is to select the algorithm that results in

<sup>1</sup><https://www.datasciencecentral.com/profiles/blogs/supervised-learning-vs-unsupervised-in-one-picture>

the best trade-off between accuracy and interpretability. Figure 2.5 is extracted from their paper and illustrates a process they designed to determine what machine learning algorithm to select. They recognize that without any particular time- and or processing limitations, it is possible to test out different algorithms and methods commonly employed in ML models. However, this type of strategy, which aims to test as many algorithms as possible to identify the best possible algorithm, can be quite extensive and time-consuming. Therefore, a proper understanding of previous use cases for different algorithms is highly beneficial.



**Figure 2.5:** Decision tree for algorithm selection process

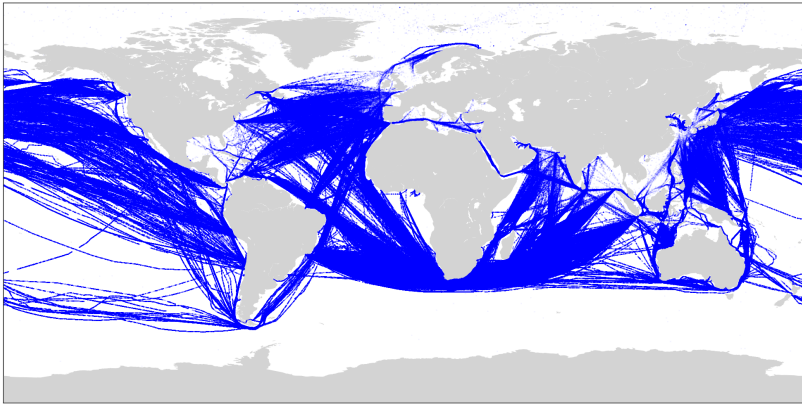
# Methodological Approach

With a detailed definition and understanding of the stated problem and objectives of this study, we can move on to the next work considerations of this thesis, namely the methodological approach. Initially however, we must conduct a thorough exploratory data analysis of the readily available data before we begin with the feature engineering process.

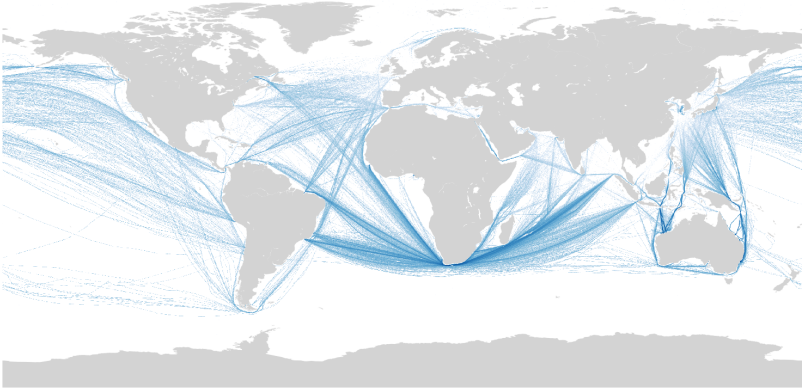
## 3.1 Exploratory data analysis

As discussed in Section 2.3.1 under literature review, conducting an EDA to get a more comprehensive understanding of the data available is typically the first step of any data science project. The primary intention of this EDA is to examine and identify any anomalies, trends, correlations, or patterns of interest that can be employed in the feature engineering process described later. To ensure that the analysis is as thorough as possible, we have investigated the available data quantitatively, with statistics, and visually, with various plots and figures.

We begin the EDA by examining the geospatial data from AIS. Figure 3.1 illustrates a scatter plot of all registered vessel recordings of Capesize vessels retrieved from the AIS database. Other than the fact that this figure confirms that we have managed to retrieve AIS data messages worldwide, there is no additional insight to gain from studying this figure. We have therefore produced a more insightful illustration in Figure 3.2, which correctly shows a density plot of the worldwide recordings. This figure allows us to better understand the major trading routes serviced by Capesize vessels in 2019. To some extent we can also determine what port locations are more commonly accessed. Clearly illustrated is the heavy traffic around Cape of Good Hope, as well as significant movement to and from Brazil and Australia. It is challenging to derive from either figure whether or not there are irregular records that we should remove from the dataset.



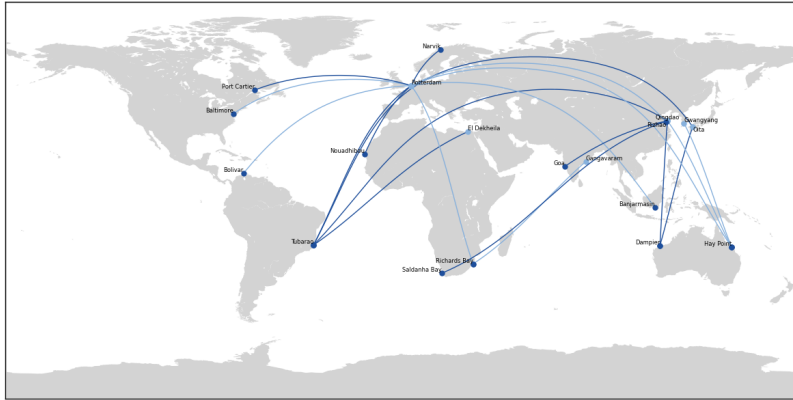
**Figure 3.1:** Scatter plot of worldwide AIS Capesize recordings



**Figure 3.2:** Density plot of worldwide AIS Capesize recordings

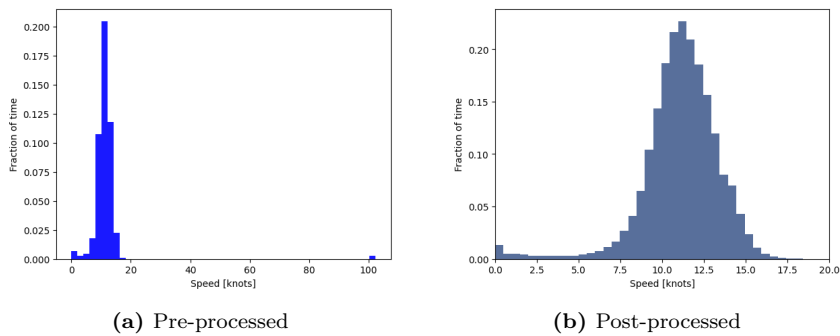
The findings from the density plot corresponds well to several known and established Capesize trading routes and port locations. In Figure 3.3 we have plotted a series of major trading routes and port locations, provided with insight from Clarksons Platou. The figure also indicates whether or not a port is importing or exporting goods and the transported commodity type. Port locations in dark blue color indicate exporting ports, while light blue colors symbolize importing ports. The two commodity types presented in the figure are iron ore and coal, with dark blue colored lines also indicating the transportation of iron ore, and light blue colored lines show the transportation of coal. However, we also observe that there are specific port locations and routes that appear to be heavily traversed according to the density plot that is apparent from Figure 3.3.

In continuation, we have addressed the vessel velocity recordings in the retrieved data from AIS. Figure 3.4 presents two velocity histogram plots, with Figure 3.4a



**Figure 3.3:** Map of major Capesize routes and port locations

illustrating original raw data, whilst Figure 3.4b shows a histogram of the data post-processing. From Figure 3.4a we see that the original raw data retrieved from AIS contains certain records with abnormal vessel velocities. Due to the abnormal velocity recordings, it is challenging to display a proper illustration of the velocity distribution without further data processing. In Figure 3.4b therefore, we illustrate the velocity distribution after employing simple processing techniques on the raw data. According to this figure, it is easy to deduce that Capesize vessels spend most of their operational time in the velocity range of 7.5 to 15 knots. However, it is also crucial to bear in mind that the reporting intervals vary significantly; a discussion further elaborated in Section 4.2.2.1. In consequence, the figures do not all together correctly illustrate the Capesize fleet’s operational status concerning time.



**Figure 3.4:** Velocity histogram of worldwide AIS recordings

## 3.2 Feature engineering

After having obtained greater insight and understanding of the available data, we can begin the process of feature engineering. It is however, essential to bear in mind that there is still substantial exploratory data analysis that we can conduct to gain further insight. Indeed, there are various steps included in the method of feature engineering, as is listed below. The approach is considered to be an iterative process as new features may constantly develop from preceedingly explored and constructed features. In consequence, this part of any data science project is generally regarded as highly time-consuming, but is perhaps also the most critical part of any such project. In short, the process aims to continuously transform the retrieved raw data into valuable features that better the representation of the underlying problem.

1. **Feature construction and extraction:** Identify, create and extract features from the retrieved raw data.
2. **Data preparation:** Transform, normalize and scale data for better application in different machine learning algorithms. Define training and test sets for the machine learning model to employ.
3. **Feature selection:** Select a subset of the final feature dataset to utilize in the machine learning model based on various statistical tests and methods.

### 3.2.1 Feature construction and extraction

Ultimately selecting what features to construct from the raw data is a market-specific question underlying the importance of conducting a proper EDA. The features built as part of this thesis are primarily chosen for the Capesize bulker segment, but may be applicable to other shipping niches. The following subsection of the chapter will describe the process of developing specific features later employed in the machine learning algorithms. A total of 65 unique features have been identified, with 52 weekly observations from January 2019 to January 2020.

#### 3.2.1.1 Vessel capacity count in world regions and port locations

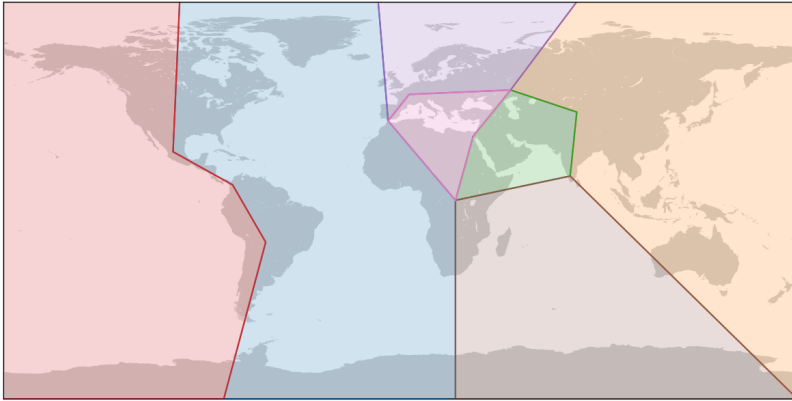
Perhaps the most intuitive features derived from the AIS data are vessel and capacity count in different zones and port locations. First, it is necessary to define the areas of interest before aggregating any data regarding geospatial locations. The python script *polygons.py* attached in Appendix E.6 was designed for this exact purpose, and it utilizes a variety of built-in packages to accomplish this. The reader can find a short and concise description of the different functions in the script. The included set of central port locations for Capesize vessels is identical to the collection of port locations previously illustrated in Figure 3.3. Utilizing the classic-sea-routes website<sup>1</sup>, it was possible to obtain and determine the physical areas of all port locations. In combination with the interactive geojson-website<sup>2</sup>, it

---

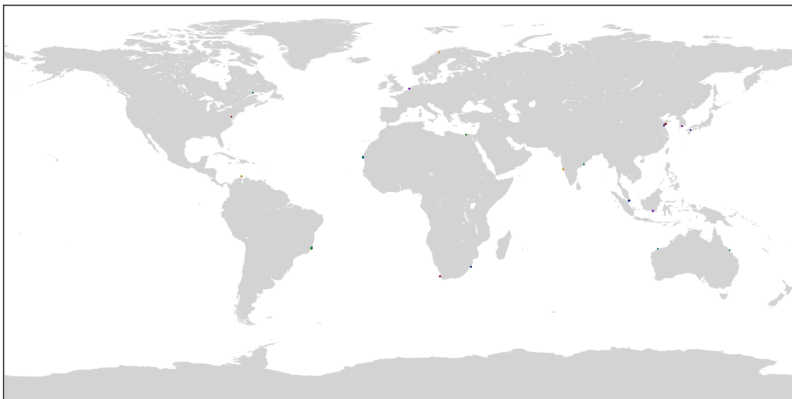
<sup>1</sup><https://classic.searoutes.com/routing>

<sup>2</sup><http://geojson.io>

enables the process of retrieving all coordinates to identify and construct the designated port sites. Figure 3.5, and Figure 3.6 presents the constructed orientation of the world polygons and port polygons, respectively. Although it may be difficult to observe from Figure 3.6, the port locations are not just small dots marked on the map. A zoomed example is presented in Figure 3.7 to illustrate better how the port locations are constructed. All coordinates are saved to individual area-based geojson-files to utilize further the established areas of interest in the feature construction process. This is accomplished with the functions *ocean\_polygons\_geojson* and *port\_polygons\_geojson* in the *polygons.py*-script. From Figure 3.5, one can observe that the world has been divided into seven polygon areas or world regions, identified as the following: Atlantic, Far East, Arabian Gulf, East Pacific, North West Europe, Indian Ocean, and the Mediterranean.



**Figure 3.5:** Orientation of world polygons

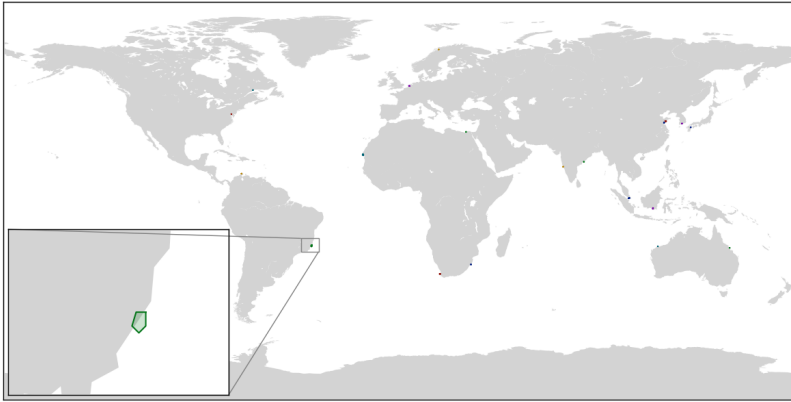


**Figure 3.6:** Orientation of port polygons

With the various areas of interest adequately established, feature construction by



deriving data from AIS may commence. The function *geofence\_processing*, included in the *FE.py*-script attached in Appendix E.7, processes data from AIS and returns a binary value indicating whether or not a recording has been made inside the area of interest. Utilizing the manufactured binary values enables the possibility of determining the set of vessels recorded at the respective regions of interest at a weekly frequency regarding their identification values, i.e., the pre-defined MMSI numbers. This is achieved with the function *vessels\_dict*. With direct access to a complete overview of all registered vessels in each pre-ordained location at any given week, deriving the count of vessels and corresponding sum of capacity for the respective areas is easily accomplished. Combining the complete dictionary<sup>3</sup> of recorded vessels with the Capesize database provided by Clarksons Platou enables the applied method in acquiring the correct sum of capacity.



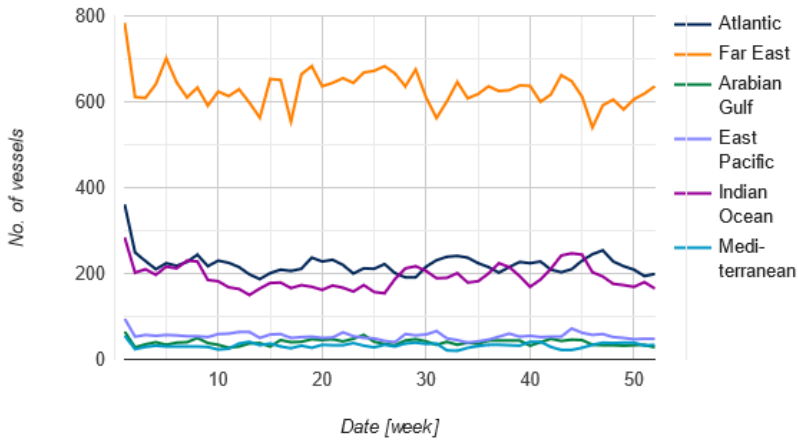
**Figure 3.7:** Zoomed example of a port polygon (Tubarao)

The distribution and progression of the reported vessels recorded in the pre-defined world regions are illustrated in Figure 3.8. The figure demonstrates that the majority of the Capesize fleet operates in the categorized *Far East*-region. A large part of the fleet is also recorded in both the *Atlantic*- and *Indian Ocean*-regions. The distribution for the remaining areas however, shows significantly lower activity levels. Considering that Capesize vessels have historically been forced to transit via the Cape of Good Hope or Cape Horn, the significant trading routes are observed from Brazil, China, and West Australia; corresponding well with our output values. Unfortunately, the data retrieved from AIS covers only a single year, which in turn makes it challenging to conclude any seasonal movements or typical trend developments.

An overview of the descriptive statistics of all features constructed in this section can be found in Table B.1, Table B.2, and Table B.3 located in Appendix B.

---

<sup>3</sup>Python dictionary, generally known as an associative array: <https://realpython.com/python-dicts/>

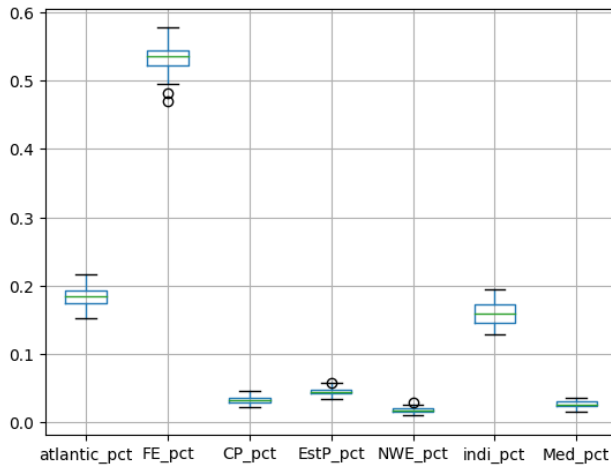


**Figure 3.8:** Vessel distribution over time in world regions

### 3.2.1.2 Fleet percentage in world regions

The fleet percentage features for the respective world regions are derived from the previously mentioned features. These features indicate the capacity percentage of the fleet in the various regions. Figure 3.9 illustrates a boxplot to demonstrate the distribution of the Capesize bulker fleet visually.

An overview of the descriptive statistics of all fleet percentage features for the respective world regions can be found in Table B.4 located in Appendix B.

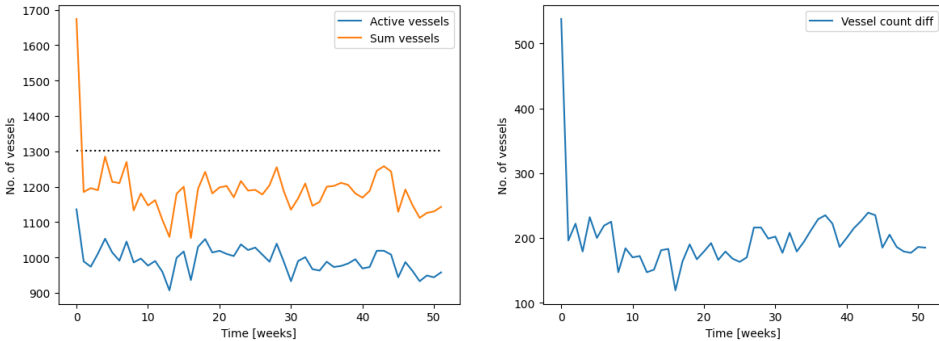


**Figure 3.9:** Capacity distribution of the world polygons

### 3.2.1.3 Active vessels and fleet capacity

In addition to providing features for the count of vessels and sum of capacity in selected ports and world regions, deriving the number of active ships and fleet capacity on a global scale at a weekly frequency may be of great significance. When developing the features for vessel count in various world locations, there is a possibility of registering several vessels in multiple areas for the same week. Consider for instance, a particular ship may report its location Wednesday morning for a specific week in the Far East region. If the same vessel is active and sailing, it is possible that it also records a location in the Indian Ocean region for the same week. Thus, the features indicating the count of vessels in the various areas of interest may contain misleading or disruptive information.

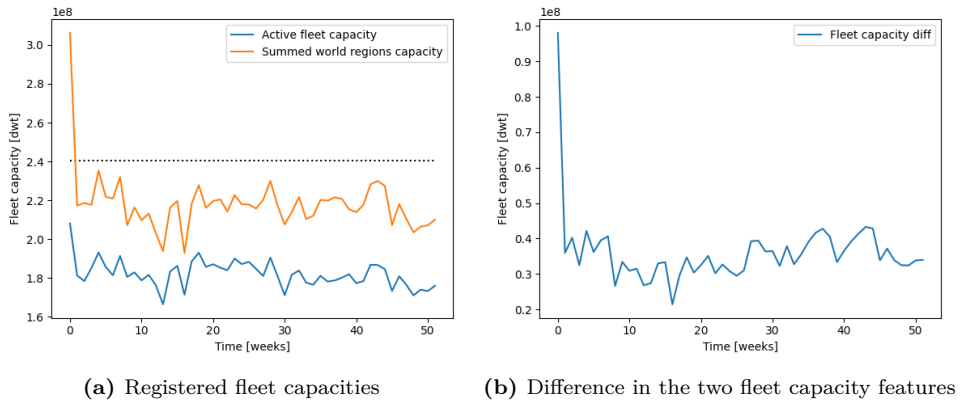
Furthermore, when deriving the number of active vessels on a global scale at a weekly frequency, additional data input is utilized, namely *nav-status*<sup>4</sup>. Moreover, the previously mentioned features, i.e. vessel count and capacity in areas of interest, do not consider ships' navigational status. This results in additional misleading information. Figure 3.10a and Figure 3.10b properly illustrates the level of conflicting information in the different vessels count features. In Figure 3.10a, it is easily observed that the two different vessel count features show relatively similar developments over time. In fact, the correlation between them is 0.84. We also recognize the dotted horizontal line as the total number of vessels registered in the fleet and investigated in this thesis. In addition, the figure shows that the *sum vessels*-feature manages to identify more ships than included in the database, which should not be possible. The second figure, Figure 3.10b, is included to provide a better visual understanding of how these two features correlate. The same issues therefore are present for the sum of fleet capacities at weekly frequencies, illustrated in Figure 3.11a and Figure 3.11b.



(a) Count of active vessels vs the sum of vessels (b) Difference in the two vessel count features

**Figure 3.10:** Comparison of different vessel count features on global scale

<sup>4</sup>A more detailed description of employed data for the works of this thesis is provided in Section 4.2.2.3



**Figure 3.11:** Comparison of different sum of fleet capacity features on global scale

An overview of the descriptive statistics of all features constructed in this section can be found in Table B.5 located in Appendix B.

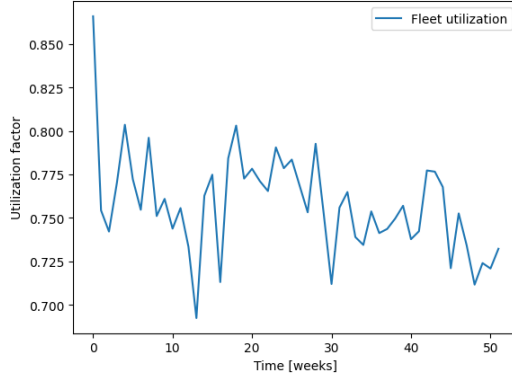
#### 3.2.1.4 Fleet utilization

The final feature derived from AIS data concerns the fleet utilization factor, intended to indicate at what level the Capesize bulker fleet is employed at a weekly frequency. This feature is constructed by the previously mentioned *fleet\_capacity*-feature, derived from the *active\_vessels*-feature, in combination with the total fleet capacity of the investigated Capesize fleet. Figure 3.12 presents the change in the fleet utilization factor over time, which indicates significant fluctuations over shorter periods. There can be numerous reasons behind these volatile variations, i.e. such as seasonal trends, cultural and political events, environmental incidents, and the global economy to name a few. Identifying and quantifying these influencing elements can provide indispensable learning material to a machine learning algorithm. However, the process of quantifying these factors is enormously challenging and likely impossible.

The descriptive statistics of the fleet utilization factor is provided in Table B.6 located in Appendix B.

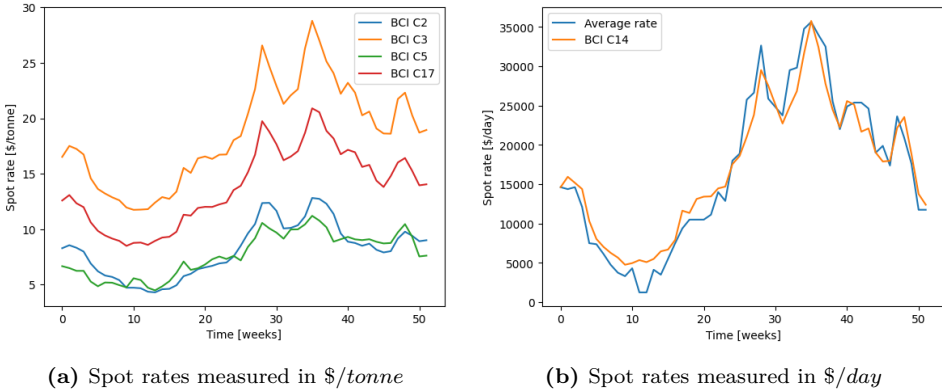
#### 3.2.1.5 Price and market features

In addition to deriving features from AIS data, this thesis also leverages market-derived features. The most commonly employed feature in price or spot-rate forecasting is the historical data of the dependent variable itself, i.e., the unit being predicted. Examples of this are Kulkarni and Haidar [23], and Yu et al. [24], which forecast future prices with only the use of historical prices as input in neural network models. Historical data for the dependent variable is not the only market-derived feature employed in this thesis; a selection of different spot rates for various main routes in the Capesize niche, with assumed relevance to the C3 route, are



**Figure 3.12:** Capesize bulker fleet utilization factor

also employed. These additional historical spot rates are included for the following Capesize routes: *C2*, *C5*, *C14*, *C17*<sup>5</sup>, as well as the *average rate* for Capesize vessels with a capacity of 172,000 dwt. Clarksons Platou provides all the abovementioned historical spot rate data at a weekly frequency. The historical spot rates are presented in both  $\$/tonne$  and  $\$/day$ , and Figure 3.13a and Figure 3.13b plots the selected spot rates against each other, with regards to the unit measurement. The figures present the dynamic relationships between the selected features and clearly illustrate the expected high correlation, with only minor independent fluctuations over time.



**(a)** Spot rates measured in  $\$/tonne$

**(b)** Spot rates measured in  $\$/day$

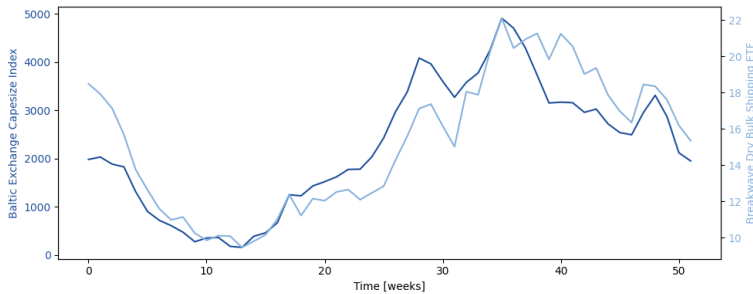
**Figure 3.13:** Historical spot rate progression of selected Capesize routes

The last two features included in this thesis are also market-derived, namely the *Baltic Exchange Capesize Index* (BCI) and the *Breakwave Dry Bulk Shipping ETF* (BDRY). The BCI provides a benchmark measurement for the price of transporting commodities by sea in the Capesize shipping segment and is thus primarily

<sup>5</sup>Table 4.2 provides a detailed overview of trading routes in dry bulk shipping

influenced by fluctuations in the Capesize market. Clarksons Platou also provided historical data of the BCI. In comparison, the BDRY is designed to *reflect the daily price movements of the near-dated dry bulk freight futures*<sup>6</sup> and is more heavily influenced by the progression in the Capesize, Panamax, and Supramax markets combined. Figure 3.14 demonstrates the progression of both features in relevance to each other. The figure also exhibits similar trend developments as the previous market-derived features, highlighting the dynamic response between the various features.

An overview of the descriptive statistics of all features constructed in this section can be found in Table B.7 located in Appendix B.



**Figure 3.14:** Historical progression of the BCI and BDRY features

A complete overview of all constructed feature types, along with a short and concise description and the anticipated influence on the various employed machine learning algorithms, can be found in Table 3.1.

**Table 3.1:** Overview of engineered features for final dataset

Feature type	Description	Unit	Expected impact
<i>AIS-derived features</i>			
port_count	Count of vessels in each port	# vessels	↕
port_capacity	Derived sum of capacity in each port	dwt	↑
zone_count	Count of vessels in each zone	# vessels	↕
zone_capacity	Derived sum of capacity in each zone	dwt	↓
zone_percentage	Calculated percentage of world capacity in each zone	%	↑
sum_zone_capacity	Calculated total sum of capacity for all zones	dwt	↓
active_vessels	Count of active vessels in the world	# vessels	↑
world_capacity	Derived sum of utilized capacity worldwide	dwt	↕
fleet_utilization	Calculated fleet utilization factor	%	↑
<i>Non-AIS-derived features</i>			
calendar_features	Year and week		↓
BDRY	The Breakwave Dry Bulk Shipping exchange-traded-fund	\$	↑
BCI	Baltic Exchange Capesize Index		↑
BCLC2	Spot rate on the BCI C2 route (Tubarao to Rotterdam)	\$/tonne	↕
BCLC3*	Spot rate on the BCI C3 route (Tubarao to Qingdao)	\$/tonne	↑
BCLC5	Spot rate on the BCI C5 route (W. Australia to Qingdao)	\$/tonne	↕
BCLC14	Spot rate on the BCI C14 route (China-Brazil round-voyage)	\$/day	↑
BCLC17	Spot rate on the BCI C17 route (Saldanha Bay to Qingdao)	\$/tonne	↕
avg_rate	Capesize 172,000 dwt average trip rates	\$/day	↕

<sup>6</sup><https://etfmg.com/funds/bdry/>

### 3.2.2 Data preparation

The second step in conducting a proper feature engineering operation is preparing and processing the data to be employed. The following three steps cover the principal data preparation and processing procedure performed as part of the works in this thesis:

1. *Data transformation:* Analyze the time series, transform and investigate whether or not stationarity exists.
2. *Normalize data:* Feature scaling of data for greater applicability in various machine learning algorithms and better comparison of different models.
3. *Split data into training and test sets:* Ensuring that the data is employable in supervised machine learning problems.

#### Step 1: Data transformation

Time series are predominantly identified as non-stationary time series, i.e., they illustrate a certain level of relation to time, for instance, in seasonal variations, trends, or cycles. In the field of machine learning, transforming non-stationary data to stationary is not necessarily a requirement. It is however highly preferred to convert all data to a stationary format as it might provide the employed algorithms with greater predictive power. As highlighted by Brooks [25], stationary data series will not be influenced as much as non-stationary data series over time in the case of a shock event. Thus, stationary data will provide a greater probability for identifying underlying elements rather than just seasonal components.

Before performing any transformation on the existing data, we analyze and investigate the necessity of any data transformation. An Augmented Dickey-Fuller (ADF) unit root test was conducted to determine the non-stationary data in the original feature dataset. The test indicated that a total of 36.92% of the original feature set was non-stationary with a confidence interval of 1%. Adjusting the confidence interval to 10%, the test categorized 25.15% of the feature set as non-stationary. Thus, to achieve greater prediction levels in the employed forecasting models, a transformation of the features dataset to stationary data was essential.

There exist a variety of employable techniques to transform non-stationary data to a stationary format. For the works of this thesis, a differencing data technique is utilized. This technique subtracts the previous observation from the current observation to determine the difference; with a given series  $Z(t)$ , we get the differenced series  $Y(t)$ , as illustrated mathematically in Equation (3.1).

$$Y(t) = Z(t) - Z(t - 1) \tag{3.1}$$

After completing the transformation of the features dataset, a second ADF unit root test is conducted to confirm that all data is stationary. The reader can find the results from the second ADF unit root test in Appendix C. The results show that

the null hypothesis ( $H_0$ ), i.e. that the series presents heteroscedasticity, cannot be rejected for 7.69% of the data with a confidence interval of 1%. However, the ADF unit root test rejects the null hypothesis,  $H_0$ , for all data series in the dataset with a confidence interval of 10%. Thus, the data is regarded as successfully transformed to a stationary format. With this being the desired outcome in regression analysis, no additional data transformation was necessary.

### Step 2: Normalize data

After successfully transforming all features to a stationary format, the second step of the data preparation procedure is to normalize the data. Normalizing the data is primarily done to ensure that all available variables contribute equally to the model. Considering that the employed feature dataset contains a vast collection of various data with a whole range of different units, this step is crucial. To illustrate this, consider two constructed features for this thesis, namely *fleet\_utilization* and *world\_capacity*. The *fleet\_utilization* feature represents the percentage utilization of the world Capesize fleet and ranges therefore from 0 to 1. The *world\_capacity* feature on the other hand, represents the total active capacity utilized and ranges from approximately 150,000,000 dwt to 200,000,000 dwt. These two features operate at totally different scales. Thus, when conducting further analysis, especially regression analysis, the *world\_capacity* feature will intrinsically influence the model more as a direct result of its larger values. However, this feature is not necessarily a better predictor for the model. To avoid these complications, therefore it is essential to normalize the data to ensure that all input variables operate at the same scale.

For the work of this thesis, we employ the *Min-Max scaling* method, also referred to as normalization, and scale all features in the range  $[0,1]$ . The mathematical formulation for scaling the data is presented in Equation (3.2).

$$z_{p,t} = \frac{x_{p,t} - \min(X_p)}{\max(X_p) - \min(X_p)} \quad (3.2)$$

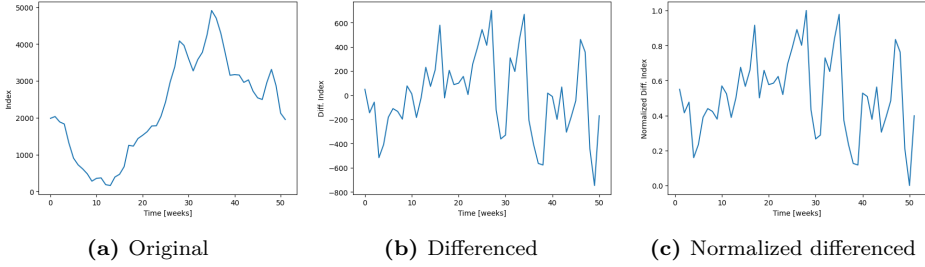
In the equation above,  $p$  indicates feature  $X_p$ , with  $x_{p,t} \in X_p$ . This results in the normalization of  $X_p$  with  $X_p^{\text{norm}} = (z_{p,t}, \dots, z_{p,n})$ , scaled in the predefined domain range, namely  $[0,1]$ . Read more about the implementation of the *Min-Max scaler* in python via scikit-learn<sup>7</sup>. Figure 3.15a through Figure 3.15c are included to provide a visual example of the employed data transformation and normalization procedure, with the *Baltic Exchange Capesize Index* (BCI) employed as an example.

### Step 3: Split data into training and test sets

With all data successfully transformed and normalized, the last part of the exercise in this data preparation process is to divide the dataset into training and test sets. The training set contains a substantial portion of sample data from the original

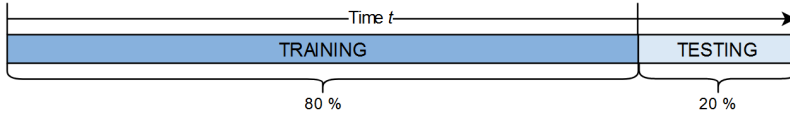
<sup>7</sup><https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>





**Figure 3.15:** BCI data transformation and normalization

dataset and is purposefully used to fit the parameters of the respective machine learning algorithms. On the other hand, the test set contains the remainder of the original dataset that is not included in the training set. The data stored in the test set is only intended for evaluation and assessment purposes on the performance level of the various employed machine learning algorithms. The test dataset provides a reference standard to compare the competing models to determine which model performs best. For the works of this thesis, a train-test ratio of [0.8:0.2] was used. Figure 3.16 correctly illustrates the division of the original dataset into training and test subsets.



**Figure 3.16:** Training and test set split

In addition to splitting the data into training and test subsets, it is essential to include a validation technique. The primary purpose of having a validation methodology on the trained data is to provide an unbiased evaluation of a model while tuning the models hyperparameters [26]. We have employed a time series cross-validation technique for the works in this thesis, namely the time series split technique, which is a variation of the KFold method. Adjusting the number of splits to divide the training set into for validation purposes, correspondingly adjusts the respective training and test sets. Equation (3.3) and Equation (3.4) presents the mathematical expressions for the training and test subset sizes respectively, with regards to the  $i$ -th split. Here,  $n$  represents the number of data samples in the training set, and  $n_s$  is the total number of splits.

$$S_{\text{train},i} = \frac{i \cdot n}{n_s + 1} + n \bmod (n_s + 1) \quad (3.3)$$

$$S_{\text{test},i} = \frac{n}{n_s + 1} \quad (3.4)$$

### 3.2.3 Feature selection

A vast collection of different features and possible predictors were developed as part of the feature construction and extraction stage. The final step in this feature engineering process is selecting various feature subsets to apply in the machine learning algorithms. However, this raises an essential question to be addressed in data science and machine learning problems specifically, namely:

*Why can we not provide the machine learning algorithms with all available features and allow the algorithm to determine which features to use?*

There are multiple reasons why it is not helpful to introduce all features for employment in machine learning. For this thesis, we have primarily been concerned with the following three aspects:

1. *Curse of dimensionality*: To better understand the meaning behind dimensionality in data science, we quote Chris Albon [27]:

*As the dimensionality of the features space increases, the number of configurations can grow exponentially, and thus the number of configurations covered by an observation decreases.*

In other words, with too much available data input, i.e. features, there is a significant risk that the applied machine learning algorithm will fit the training data perfectly. This in turn, will result in the case of overfitting. Thus, the employed algorithm will not generalize and modify the test data, indicating that the model has not learned anything.

2. *Occam's razor*: A philosophical problem-solving principle stating that *entities are not to be multiplied beyond necessity* [28]. This principle gives precedence to simplicity and applies exceptionally well in data science. Therefore, it is preferable to construct machine learning models in a straightforward manner. With more features introduced to a model, the less intuitive and explainable the model becomes.
3. *Garbage in  $\rightarrow$  Garbage out*: This last element is perhaps quite self-explanatory. The majority of data science problems include a multitude of non-informative features, which is likely also the case for this thesis. As presented in Table 3.1, a complete overview of all constructed feature types for this thesis is included, with the rightmost column indicating the expected impact on the employed algorithms for the different feature types. However, these levels of expected impacts are merely assumptions. Providing an algorithm with poor-quality input data will result in poor-quality output data and consequently poor forecasting results. It is therefore necessary to avoid excess feature input. Reducing the number of feature inputs also makes the machine learning algorithms less time-consuming and easier to implement.

Ultimately therefore and as pointed out by Shaikh [29], performing proper feature selection techniques may yield multiple benefits concerning performance measurements, i.e., *reduces overfitting, improves accuracy, and reduces training time.*

For the works of this thesis, three different feature combinations have been employed, i.e., *all features*, *top features*, and *selected features*.

As the first feature combination implies, this utilizes all available features when training the data; this is employed to investigate if the other feature selection methods applied in this thesis yield any beneficial performance enhancements. In comparison the second feature combination, *top features*, employs the top  $n$  features as input data to fit and train the respective machine learning models. The methodology for selecting the top  $n$  features is highly inspired by Næss [6], and the reader can find a thorough and detailed description of the employed filter selection method in the respective paper.

The top feature selection process identifies the top  $n$  features regarding the mean feature score from various filter methods. Table 3.2 presents an overview of the different univariate and multivariate filter methods used in this thesis to determine the respective feature importance scores. The feature importance scores are ratings indicating the relative importance of the individual features in predicting the predictor. These ratings are scaled in the domain of  $[0,1]$ , where 0 implies that the corresponding feature is of no value in predicting the predictor, while 1 signifies the most statistically significant feature. Figure D.4 through Figure D.6 in Appendix D provides a complete overview of all calculated scores for the individual features with the corresponding filter method. The analysis process for this feature selection method is conducted using the *feature\_importance\_score.py*-script, located in Appendix E.9.

**Table 3.2:** Univariate and multivariate filter methods to identify top  $n$  features

Univariate	Multivariate
Linear correlation	Linear regression
Maximal information coefficient	Ridge regression
	Random forests

The last feature combination, *selected features*, utilizes a feature selection approach developed by Koehrsen [30]. The first step in this approach is to remove one out of every pair of feature variables that presents a correlation level greater than a pre-defined value; for this thesis, the value has been set to 97.5%. The original feature set, including all previously constructed features, contains a total of 130 features. The first step in this approach identified a total of 51 features to drop from the original dataset. Thus, the updated features dataset contains only 79 features, a 39.231% decrease in features. The next step is to determine the feature importance scores for the remaining features. After training a random forest regression model on the complete training set, it is possible to obtain an overview of the different feature importances. Normalizing the ranked scores, i.e., summing the importance scores to one, allows for determining the significance of the individual features on the model. Additionally, it is possible to determine the cumulative importance score and identify the least necessary number of features required to reach a desired

cumulative importance threshold. For experimental and investigative purposes of this thesis, we explored a threshold of 95%, which indicated that 30 features were necessary to reach this level of cumulative feature importance. The most important features according to this approach, along with the cumulative feature importance score, are presented in Figure D.1 and Figure D.2 respectively, located in Appendix D.

However, rather than simply identifying the top features, as previously achieved by calculating the mean feature importance scores on various filter methods, this last feature combination aims to select the best possible subset of  $n$  features. Conducting a recursive feature elimination (RFE) process will help reach this goal. An RFE utilizes an external estimator, i.e., the number of trees in a random forest regression model, to select features by recursively considering smaller and smaller sets of features<sup>8</sup>. It was possible to identify the optimal estimator to the random forest regression model utilizing a cross-validation technique. The reader can find the Python script *feature\_selection.py* for the approach of extracting this last feature combination in its entirety in Appendix E.10.

## 3.3 Algorithm selection

It is important to note that the focus of this study is not on making discoveries in the field of machine learning or data science. Instead, the aim is to investigate the applicability of existing sophisticated machine learning algorithms to predict short-term freight rate movements and potentially identify significantly influencing elements in the Capesize shipping segment. One of several objectives for this thesis is to select relevant machine learning algorithms to employ in forecasting short-term freight rate movements. This section will therefore provide the reader with only a brief introduction to a selection of potential algorithms.

### 3.3.1 Linear ridge regression

The first machine learning algorithm we are exploring is the *linear ridge regressor* (LRR) model. To better understand how a ridge regression model functions, we begin by exploring linear regression. There are two different applications of linear regression, namely *simple linear regression* (SLR) and *multiple linear regression* (MLR). Since we investigate multivariate data as input to the selected machine learning algorithms, we are primarily interested in MLR. The principal use for regression models is when the dependent variable, i.e., the predicted value, is a continuous data type; while the employed predictors, i.e., the input data, can be any data type<sup>9</sup>. The purpose of the regression model is to identify the best fit line, which illustrates the relationship between the predictors and dependent variable, which yields the least error value. To find the best fit line for the dependent variable, we employ the following equation:

---

<sup>8</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.feature\\_selection.RFE.html](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html)

<sup>9</sup><https://www.mygreatlearning.com/blog/linear-regression-in-machine-learning/#multiplelinearregression>

$$\hat{y}(w, x)_t = w_0 + w_1x_{1,t} + w_2x_{2,t}\dots + w_px_{p,t} \quad (3.5)$$

In the above equation,  $\hat{y}(w, x)_t$  represents the predicted value, at time  $t$ , given a specific set of input values  $x$  ( $x = [x_1, x_2, x_3, \dots, x_p]$ ) and the respective weighted coefficient for each data input  $w$  ( $w = [w_1, w_2, w_3, \dots, w_p]$ ), with  $p$  being the number of input variables. The equation also includes one additional weight coefficient, namely  $w_0$ , which is purposefully added to provide the model with an extra degree of freedom, more commonly referred to as the intercept or bias coefficient. The final equation for an MLR model with the associated error term for the predicted value at time  $t$  consequently becomes:

$$\hat{y}(w, x)_t = w_0 + w_1x_{1,t} + w_2x_{2,t}\dots + w_px_{p,t} + \epsilon_t \quad (3.6)$$

The overall objective of the MLR model is to determine the weight coefficients ( $w = [w_1, w_2, w_3, \dots, w_p]$ ) by minimizing the error variable. An LRR model is very similar to an MLR model, with the main difference being that an LRR model employs an L2 regularization technique. This makes the LRR the preferable model type in problems where the input data suffers from multicollinearity. Multicollinearity is a term associated with statistics and is defined as *the occurrence of high intercorrelations among two or more independent variables in a multiple regression model* [31]. The primary reasons for applying a regularization technique is to reduce the model complexity via feature selection and address issues regarding over-fitting. Considering that the employed dataset for this thesis contains a vast amount of constructed features, this technique can be highly effective and consequently provide valuable forecasting results. The L2 regularization technique utilizes the squared magnitude of the respective coefficients as penalty terms for the loss function<sup>10</sup>. The LRR model aims to minimize the objective function presented in the equation below, with the penalty term illustrated as the highlighted part of the equation.

$$\min(\|\hat{y} - wx\|^2 + \alpha \|w\|^2) \quad (3.7)$$

It is critical to determine a good value for  $\alpha$  to avoid over-fitting or under-fitting. This method however is generally considered to be an excellent technique to avoid possible over-fitting issues. For more information on LRR models, documentations<sup>11</sup> and user guides<sup>12</sup> on the implementation of an LRR model in python, we refer to scikit-learn.

---

<sup>10</sup><https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c>

<sup>11</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.Ridge.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html)

<sup>12</sup>[https://scikit-learn.org/stable/modules/linear\\_model.html#ridge-regression](https://scikit-learn.org/stable/modules/linear_model.html#ridge-regression)

### 3.3.2 Random forest regressor

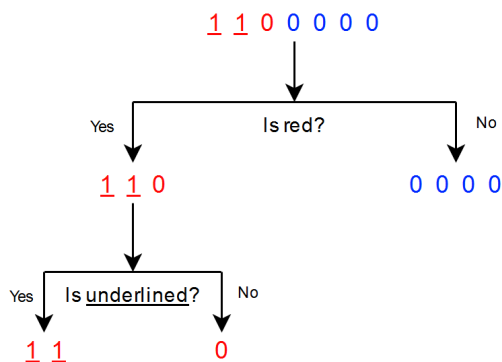
Secondly, we have investigated the applicability of a random forest regression model. Random forest regression models utilize the ensemble learning method, which is a learning technique that focuses on combining multiple predictions from various machine learning algorithms to determine the best possible forecast. These types of machine learning models are therefore categorized as *ensemble models*. Machine learning models that employ ensemble learning can be further classified into the following types:

1. Boosting → The purpose of this ensemble learning technique is to make various algorithms learn from each other by communicating which feature or set of features to best focus on in the upcoming models. This technique therefore focuses on teamwork capabilities. The term *boosting* comes from the fact that the employed machine learning algorithm is provided with instructions from previously exhausted algorithms that augment or enhance the current algorithm's learning abilities.
2. Bootstrap aggregation (bagging) → This type of learning methodology employs only a portion of data from the complete dataset. The small subset of data used is retrieved at random. This technique is a commonly applied procedure when the objective is to reduce the variance in machine learning algorithms with high degrees of variance. Compared to the boosting practice, *bagging* functions with each model running independently and afterward aggregates the output values with no preference to any model.

A random forest machine learning model is a sophisticated and supervised machine learning algorithm with applicability for both regression and classification problems. The difference between a regression and classification problem is further discussed in Section 3.4.2. Random forest models operate with a bagging technique since the constructed trees in the model are run independently, and in parallel, with absolutely no form of communication between them. Considering that random forest models are built by combining several decision trees into one model, it is necessary first to learn and understand how decision trees work.

A simple example of a decision tree is presented in Figure 3.17 [32]. The sample input data set is given at the top of the figure. The objective is to separate and identify smaller sets of data by combining various features. For this particular example, the employed features are the color (red or blue) and format (underlined or not) of the input data. In the first split, the input dataset is evaluated on the color of the data. If the data has the color red, it moves the Yes branch, and if the color is anything but red, it moves down the No branch. The same process is done with the format feature in focus, as is illustrated in the figure. In this example, it was necessary to employ two different features to split all the data perfectly. However, this is only a simple example to understand better how a decision tree functions. Problems encountered with real data are often more challenging to process, but the theory and logic behind a decision tree remain the same. Considering that decision trees are extremely data-sensitive, the produced predictions can vary significantly.

A random forest model allows for combining several decision trees into one model, reducing some data sensitivity in the predictions.



**Figure 3.17:** Simple decision tree example<sup>13</sup>

Random forest models are sophisticated and supervised machine learning algorithms with applicability for both regression and classification problems, as described in Section 3.4.2. The constructed trees in a random forest model utilize the bagging technique, meaning they are run in parallel with absolutely no interaction between the different trees. The algorithm for random forest regression models functions by creating a multitude of trees, combined create a forest, and the resulting prediction is the mean prediction of all individual trees. An illustration of this process and a general structure of the random forest algorithm is given in Figure 3.18. The idea behind this technique is that *a large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models* [32]. It is therefore essential to ensure that the separate trees or models diversify each other, i.e., each tree can not be too correlated with any other tree in the forest. With the utilization of the bagging technique, allowing each tree to sample input data with replacement from the original dataset at random, the algorithm achieves some diversification. Additionally, the random forest algorithm also ensures diversification through *feature randomness*. In contrast to regular decision trees, where all dataset features are available when determining how to split the node, each tree in random forest models can only employ a random subset of features. Thus, a random forest model utilizes a combination of random sample data input and feature randomness to achieve optimal diversification of all constructed trees. For more information on Random Forest models, documentation<sup>14</sup> and user guides<sup>15</sup> on the implementation of a Random Forest Regression model in python, we refer to scikit-learn.

<sup>13</sup><https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

<sup>14</sup><https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

<sup>15</sup><https://scikit-learn.org/stable/modules/ensemble.html#forest>

<sup>16</sup><https://medium.com/swlh/random-forest-and-its-implementation-71824ced454f>

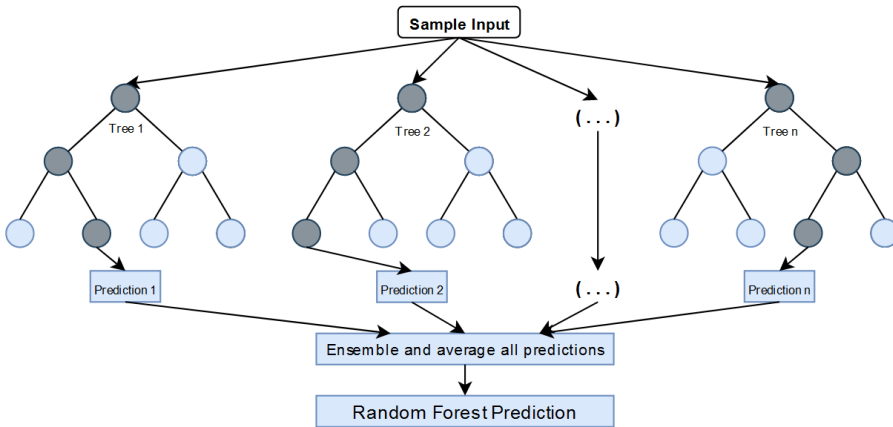


Figure 3.18: Random forest algorithm structure<sup>16</sup>

### 3.3.3 Long short-term memory

The third model we wish to explore is the long short-term memory (LSTM) model. LSTM networks are recognized as a particular type of recurrent neural network (RNN), while RNN is a class of artificial neural networks (ANN). Thus, to properly understand how an LSTM network operates and the benefits of employing such a network, it is essential first to grasp a basic understanding of ANN and RNN. Considering that the focus of this thesis is on the employment of various machine learning algorithms, such as an LSTM network, this section will only provide a brief introduction to both ANN and RNN.

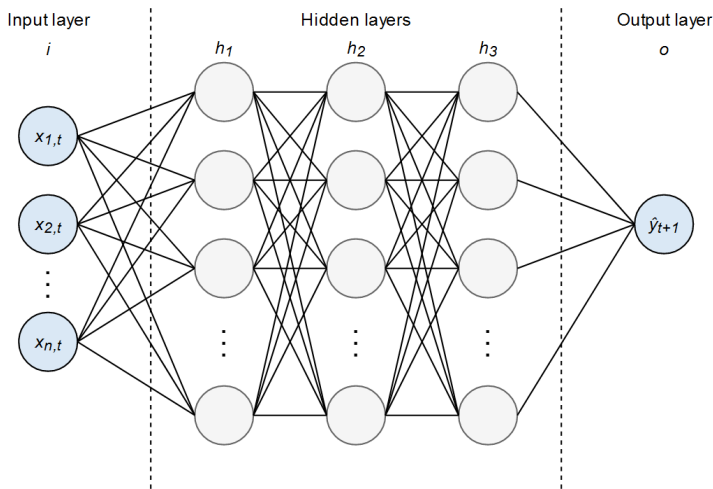
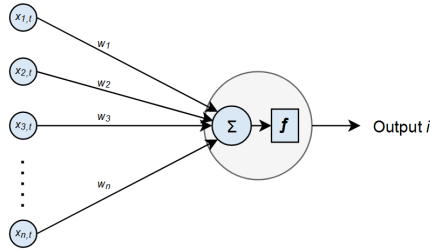


Figure 3.19: MLP with 3 hidden layers and  $n$  input features



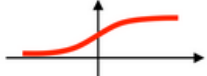
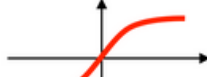
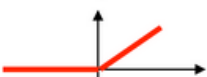
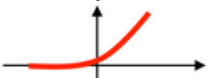
ANN models are considered to be exceptionally well suited for fitting and solving complex problems, as this type of network does not make any a priori assumptions regarding the issue at hand. The Multilayer Perceptron (MLP) is an example of a neural network and is represented in Figure 3.19. From the illustration, we observe several key elements of an artificial network, namely *input layer*, *hidden layers*, and *output layer*. The input layer consists of  $n$  nodes for each feature in the feature dataset, and each input node is connected to all nodes in the first hidden layer, indicated as  $h_1$  in the figure. From the first hidden layer, the input to the subsequent layers is the current layer's outputs. The nodes in the hidden layers are where the computation of the model is performed. The outputs from all nodes in the final hidden layer, indicated as  $h_3$  in the figure, provides the single output value  $\hat{y}_{t+1}$ , i.e., the predicted value at time  $t + 1$ . The architecture of an ANN is given by the number of inputs (nodes in the input layer, i.e.,  $X = \{x_{1,t}, x_{2,t}, \dots, x_{n,t}\}$ ), number of hidden layers ( $h = \{h_1, h_2, h_3, \dots\}$ ), number of neurons in the hidden layers (nodes in the hidden layers) and number of outputs ( $\hat{y}_{t+1}$ ). Figure 3.20 provides an illustration of the composition of a single neuron  $i$  in the first hidden layer of an MLP and is included to give the reader a better understanding of the computational process.



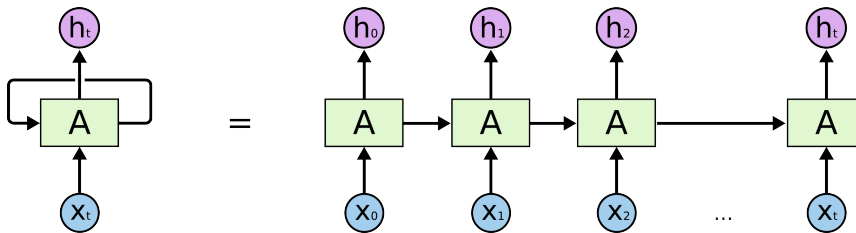
**Figure 3.20:** Composition of neuron  $i$  in the first hidden layer of an MLP

As the figure illustrates, each neuron in the first hidden layer has a set of  $n$  input values  $X_n$  ( $X_n = \{x_{1,t}, x_{2,t}, \dots, x_{n,t}\}$ ). Each input value has a corresponding weight value  $w$ . Additionally, we identify an activation function  $f$ . Thus, the output of node  $i$  becomes the sum of the paired input value  $x$  with the corresponding weight  $w$  multiplied by the applied activation function. The computation process for node  $i$  in the subsequent hidden layer is similar, except that output values of all nodes in the current hidden layer become the input values for the nodes in the subsequent layer. The activation function aims to determine to what extent the received signal progresses through the network and the effect on the concluding outcome. Thus, the choice of activation function varies from problem to problem. This thesis will not go into detail and describe various activation functions. However, Table 3.3 presents a selection of commonly applied activation functions, extracted from the works of Raschka [33].

**Table 3.3:** Overview of selected activation functions for ANN [33]

Activation function	Example use cases	Equation	1D Graph
Logistic (sigmoid)	Logistic regression, Multi-layer NN	$\phi(z) = \frac{1}{1 + e^{-z}}$	
Hyperbolic tangent	Multi-layer NN	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	
Rectified linear unit	Multi-layer NN	$\phi(z) = \max(0, z)$	
Rectifier, softplus	Multi-layer NN	$\phi(z) = \ln(1 + e^z)$	

Now that we have obtained a basic understanding of how an ANN functions, it is necessary to understand the ABCs of RNNs. As previously mentioned, an RNN is a class of ANNs and was first introduced by Elman [34]. In traditional neural networks, there is no persistence, meaning they do not have the foundations for utilizing previously recorded data, they only use the current data as input. RNNs are developed to address this specific issue. The primary difference is that RNNs operate with loops, thus allowing data and information to persist throughout the termination of each iteration in the program. The looping process, which enables data to flow through the network and lasts throughout the entirety of the program, is illustrated in Figure 3.21, retrieved from Olah [35].



**Figure 3.21:** Unrolled recurrent neural network

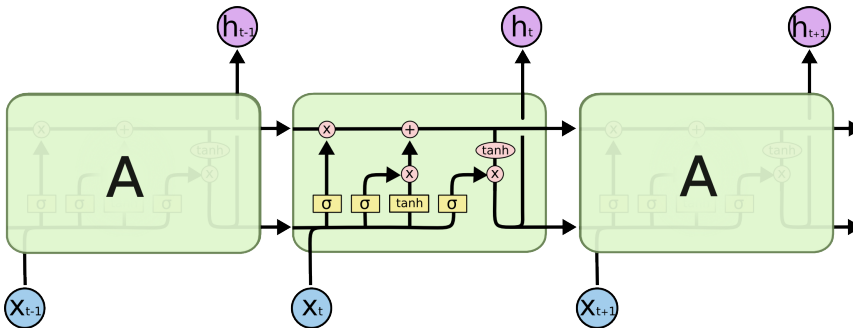
In the process of determining the gradients, which are used for updating the weights in the network, RNNs leverage a backpropagation through time (BPTT) algorithm [36]. The BPTT algorithm ensures that the model trains using the calculated errors from the output layers to its input layers. This in turn, fits the employed parameters of the model correspondingly. Unlike traditional backpropagation, the

BPTT needs to sum errors at each step since the operational parameters are shared across each network layer. However, it is also during this process of determining the gradients that most RNNs run into problems, i.e., the issues of *exploding gradients* and *vanishing gradients*. These issues are described as the following [37]:

1. *The vanishing gradient problem*: A relatively small gradient will continue to decrease until it ultimately becomes insignificant (i.e. = 0), resulting in difficulties for the network to learn from long term dependencies.
2. *The exploding gradient problem*: Opposite to the vanishing gradient problem, when a gradient is too large and goes towards infinity at an exponential rate, the gradients will eventually become NaN, and the model becomes unstable.

The problem concerning the vanishing gradient and the issues of long-term dependencies was explored by Hochreiter [38]<sup>17</sup>. A variant of RNNs, the LSTM network, was principally developed to tackle the problem of the vanishing gradient in RNNs and has been significantly applied since it was first introduced to the field of data science by Hochreiter and Schmidhuber [39].

Figure 3.22 (extracted from the works of Olah [35]) provides an illustration of the chain like LSTM structure and a detailed view of the repeating module is presented. In contrary to a standard neural network model, that only utilize a single neural network layer, in an LSTM network there are four neural network layers, as indicated in Figure 3.22. The key characteristics of an LSTM network is the ability to remove or add information, regulated by different gates. These gates allows the model to control the flow of information used as input in the network model.



**Figure 3.22:** Structure of the repeating module in an LSTM model

### 3.3.4 Model tuning with hyperparameter optimization

After conducting a thorough algorithm screening, i.e., identifying suitable and employable algorithms, it is vital to enhance the respective algorithms as much as possible for the relevant problem. One commonly applied method for doing so is

<sup>17</sup>Conducted as a thesis in computer science, only available in german

tuning the hyperparameters in the respective machine learning models. It is crucial to understand that model parameters and hyperparameters are not the same. *Hyperparameters* are defined as the building blocks for the model architecture. Thus, optimizing the hyperparameters of the respective models is considered an exhaustive but necessary process to achieve greater predictability. The general idea behind the tuning process is to instruct the machine learning model to explore a range of possible values for a set of hyperparameters and ultimately identify and select the structural foundation that yields the best score.

There exist multiple methods for hyperparameter tuning in machine learning problems. Jordan [40] recognizes three commonly applied methods, namely *grid search*, *random search*, and *bayesian optimization*. The *cross validation grid search*-method has been leveraged as the hyperparameter tuning technique for this thesis, which allows for selecting the scoring method utilized the cross validation. The *negative root mean squared error*-scorer has been employed for this hyperparameter optimization technique. This method is considered an extremely exhaustive approach, as it evaluates all possible combinations of the various hyperparameters it is instructed to investigate. If for instance, the objective is to identify the best combination of two different hyperparameters, each with four different possible values, the approach trains the model on all 16 different combinations. Eventually, after all combinations have been trained and evaluated, the grid search will output the combination that performed best. This method is far from being considered the best hyperparameter optimization technique. As pointed out by Johnson [41], the *grid search*-method does not guarantee that it will identify the best solution, as the problem of aliasing tends to occur.

A complete overview of the investigated hyperparameters for the different machine learning algorithms is given in Table 3.4. The table also includes a short description of the various hyperparameters and the explored search range. For documentation on the implementation of the grid search method with cross-validation, we refer to scikit-learn<sup>18</sup>.

**Table 3.4:** Overview of optimized hyperparameters in selected machine learning algorithms

Model	Hyperparameter	Definition	Search Range
Linear Ridge Regressor (LRR)	Alpha	Regularization strength to reduce the variance	[0.01, 0.05, 0.1, 0.5, 1, 2, 5]
	Solver	Computational method for fitting the regression model	['auto', 'lsqr', 'sag', 'saga']
Random Forest Regressor (RFR)	N-estimators	Number of trees in the forest	[20, 50, 100, 150, 200]
	Max_depth	Max depth in the tree	[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
Long Short-Term Memory (LSTM)	Epochs	Number of times the algorithm will work through the training dataset	[10, 25, 50, 75, 100]
	Batch_size	Number of samples the network requires before performing a weight update	[8, 16, 32, 64, 128, 254]

<sup>18</sup>[https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.GridSearchCV.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)

## 3.4 Model evaluation method

To best assess the quality performance of any forecasting model, it is essential to determine a benchmark performance level and define a set of metrics to evaluate against benchmark results. The most common practice for assessing various machine learning models is quantifying the relative error and accuracy of the respective model. This section of the paper will describe a variety of regularly employed forecasting methods that can be utilized as a benchmark model to compare the final results. Additionally, we identify three statistical metrics for evaluating error and accuracy levels for machine learning models.

### 3.4.1 Baseline model

The results from a baseline model are primarily utilized to produce a relative understanding of other employed model's performance. The *last-value* forecasting technique, commonly referred to as the *naive method* or *persistence model*, is perhaps the most frequently applied model to obtain benchmark results. This technique uses the last observation as a forecast for the next period, expressed by the following equation:

$$\hat{y}_t = y_{t-1} \tag{3.8}$$

In the above equation,  $\hat{y}_t$  represents the predicted value of  $y_t$  at time  $t$ , while  $y_{t-1}$  is the real value of  $y$  at the previous time step i.e.  $t - 1$ . Because of the model's simplicity, quick implementation, and the fact that it requires no training, this model provides a good baseline forecast for comparison. This technique can be especially applicable in problems where the underlying assumption of a stationary time series is questionable. In laymen's terms, the time series indicates extreme volatility that can be difficult to capture by more advanced methods.

Another regularly used forecasting method to obtain benchmark results is the *averaging forecasting*-technique. To produce the predicted value  $\hat{y}_t$  at time  $t$ , it employs the average value of all past observations, illustrated by the equation below.

$$\hat{y}_t = \sum_{\tau=1}^t \frac{y_{\tau-1}}{t} \tag{3.9}$$

Hillier [42] highlight that this particular method is best suited for *young processes* (i.e., not too many data points) and that the data should be relatively stable. In other words, large datasets with significant volatility should avoid applying this forecasting technique.

The *moving-average forecasting*-technique is quite similar to the abovementioned *averaging forecasting*-technique. However, this method does not employ the average value of all past observations; it only applies the past  $n$  observations to produce

the predicted value  $\hat{y}_t$  at time  $t$ . The equation below illustrates how this technique is applied in practice.

$$\hat{y}_t = \sum_{\tau=t-n+1}^t \frac{y_{\tau-1}}{t} \quad (3.10)$$

The primary advantage of this method in comparison to the standard *averaging forecasting*-technique is that the old observations, and perhaps irrelevant data, are discarded. On the other hand, when predicting the next forecasting value, all previous and included data have equal weight in determining the value.

The final method we describe for potential use to produce the baseline model is the *exponential smoothing forecasting*-technique. This method defines a recursive relationship between current observation and current forecast, as compared with the previously mentioned techniques that only utilize actual historical values directly to determine the forecasted value. The following equation describes the recursive relationship:

$$\hat{y}_t = \alpha y_{t-1} + (1 - \alpha) \cdot \hat{y}_{t-1} \quad (3.11)$$

In the above equation,  $\alpha$  represents the smoothing constant while  $\hat{y}_{t-1}$  is the predicted forecast value at time  $t - 1$ . This particular technique intends to use a weighted average of previous observations, with more recent observations having a higher weight in determining the value.

After exploring various potential baseline models, we have decided to employ a persistence model for producing the benchmark results for the case investigated in this thesis. This decision is primarily due to the persistence model's level of simplicity and quick implementation. Additionally, we recognize that with the volatile nature of spot rates in the Capesize segment, a persistence model might capture elements of the rapidly changing time series better than some of the more sophisticated models employed in this study.

### 3.4.2 Statistical modeling metrics

Supervised machine learning algorithms are classified into two main types<sup>19</sup>, namely:

1. *Regression*: Used to predict a continuous variable
2. *Classification*: Used to predict a discrete variable

---

<sup>19</sup><https://www.mygreatlearning.com/blog/linear-regression-in-machine-learning/#multiplelinearregression>

Moreover, there are different statistical metrics developed and used in evaluating the performance of regression and classification models. In previous sections, it has been stipulated that the study of this thesis focuses on predicting the value of a continuous variable. Consequently, the statistical metrics used in the evaluation process of this thesis are metrics developed for regression models.

As discussed above, we have employed a combination of three error metrics for measuring the accuracy of the various employed machine learning algorithms. There exist multiple metrics for assessing the performance level of machine learning algorithms on both magnitude and directional scales, but for evaluation purposes in this thesis, we have only utilized statistical metrics to determine the magnitude accuracy. The metrics used in this study are the *Root Mean Squared Error* (RMSE), the *Mean Absolute Error* (MAE) and finally the *Mean Absolute Percentage Error* (MAPE). The equations for calculating the different error metrics are presented in Equation (3.12), Equation (3.13) and Equation (3.14) respectively. In the functions below,  $\hat{y}_t$  is the predicted value of  $y_t$  at time  $t$  for a set with a sample size of  $n$ .

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{t=1}^n (y_t - \hat{y}_t)^2} \quad (3.12)$$

$$\text{MAE} = \frac{1}{n} \sum_{t=1}^n |y_t - \hat{y}_t| \quad (3.13)$$

$$\text{MAPE} = \left( \frac{1}{n} \sum_{t=1}^n \frac{|y_t - \hat{y}_t|}{|y_t|} \right) \cdot 100\% \quad (3.14)$$

The RMSE is widely considered the most popular form of metric in evaluating the predictive performance of machine learning algorithms in regression problems. Performance measurement in machine learning is predominantly efficient when the input data is considerably large. This however is not the case for the problem investigated in this thesis.

# Chapter 4

## Computational Study

This study utilizes two primary data sources to provide input data and insight into the Capesize vessel size of the dry bulk shipping market. These data are sourced from (1) Clarksons Shipping Intelligence Network (SIN)<sup>1</sup> and (2) the NCA AIS database. Additional insight into influencing elements and economic determinants in dry bulk shipping and a greater understanding of the AIS database has been obtained after conducting a thorough literature review from previous studies, which were described in Chapter 2. Discussions with research analysts and shipowners have provided an added understanding of the shipping industry, in general, as well as significant elements of the Capesize vessel segment.

This chapter will present the obtained raw data from the abovementioned sources and the data processing methods employed in this study. Data preparation, processing, and manipulation are critical elements for this study to provide datasets of high quality to utilize as input in the employed prediction models. The last section of this chapter will display the produced results from the benchmark and machine learning models.

### 4.1 The bulk shipping case

As outlined initially, the central objective of this thesis has been to investigate the applicability of sophisticated machine learning algorithms to predict short-term freight rate movements on the Capesize route from Tubarao to Qingdao (C3). The choice of this particular shipping segment as well as route alternative has been primarily as a consequence of the following considerations:

1. *Market efficiency*

---

<sup>1</sup><https://sin.clarksons.net/>



### 2. Manageable data quantity and availability

### 3. Study limitations

It was necessary to approach the thesis objective through the use of available data originating from an efficient marketplace. Whilst the global shipping market is inarguably efficient over the long-run, this is of limited interest in predicting short-term freight rate movements. Conversely, as various studies have sought to show, market efficiency is not necessarily present in smaller shipping segments or niche operations. Generally however, market efficiency is strong in the volume segments such as global tanker or dry bulk operations, both as a consequence of the many operators located worldwide, but also in consideration of the spill-over effect observed between the different sub-segments at times of significant freight shifts. It follows that our thesis objective required a volume segment to achieve apparent market efficiency. However, given the very large number of vessels operating in both the global tanker and dry bulk segments, further reduction criteria was necessary in order to reduce the quantity and scope of study.

From the preliminary work of the specialization project conducted in the fall of 2020, we identified the dry bulk market in preference to the tanker operations, as a segment to perform a behavioral study. This decision was made with a view to the bulk markets considerable size, potential and applicability.

#### 4.1.1 Commodity types

In aggregate, dry bulk shipping accounts for the majority of global shipping volume. However, commodity characteristics are determinants in choice of vessel capacities. Frequently specific commodities require vessel specialization and volume optimization. Consequently, certain bulk vessels are dedicated to transporting iron ore, coal or grain, whilst other vessels may transport a variety of commodities, such as steel, cement, bauxite or minerals<sup>2</sup>. Most commonly commodities are differentiated between minor and major.

##### 1. Major Dry bulk commodities

- (a) Iron Ore
- (b) Coal
- (c) Grain

##### 2. Minor Dry bulk commodities

- (a) Fertilisers
- (b) Steel
- (c) Cement
- (d) Bauxite

---

<sup>2</sup><https://www.eurosender.com/blog/en/shipping-dry-bulk-commodities/>

## (e) Miscellaneous minerals

There are substantially fewer operators handling major dry bulk commodities. Operations in this segment are more regular, easier to handle with fewer charterers and accessible ports for loading and discharge. Capital costs however, are substantially higher than the smaller segments and the fleet consequently numbers fewer vessels.

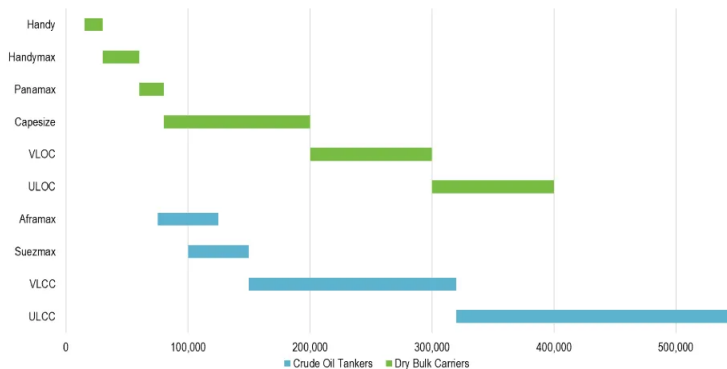
### 4.1.2 Classification of vessel types

The different shipping segments have classified ships employed into a set of vessel subtypes. Table 4.1 below lists the commonly used size classification for ships in the dry bulk segment; the table is extracted from the United Nations Conference on Trade and Development (UNCTAD) Review of Maritime Transport 2020<sup>3</sup>.

**Table 4.1:** Vessel size groups according to commonly used shipping terminology

Dry bulk and ore carriers	Vessel Capacity
Capesize bulk carrier	100,000 dwt and above
Panamax bulk carrier	65,000 - 99,999 dwt
Handymax bulk carrier	40,000 - 64,999 dwt
Handysize bulk carrier	10,000 - 39,999 dwt

However, there exist many different vessel size groupings depending on the source. Figure 4.1 below illustrates an example of this. We note that all the classifications from Table 4.1 are included with slightly different vessel capacity ranges. This figure also introduces two additional classes for dry bulk carriers: *Very Large Ore Carriers* (VLOC) and *Ultra Large Ore Carriers* (ULOC).



**Figure 4.1:** Vessel Size Groups (in deadweight tons)<sup>4</sup>

<sup>3</sup><https://unctad.org/system/files/official-document/rmt2020-en.pdf>

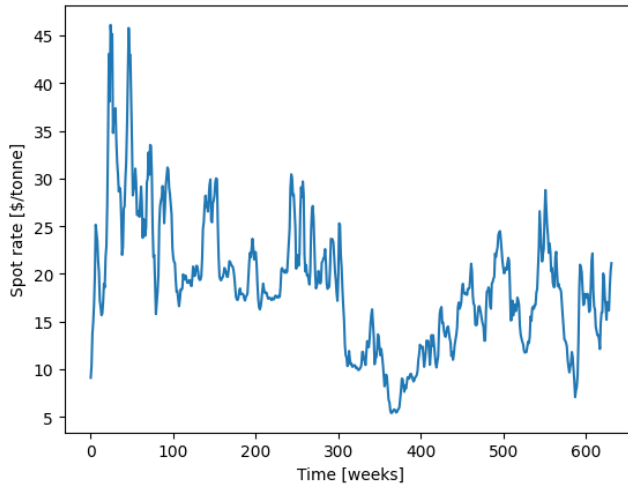
<sup>4</sup><https://transportgeography.org/contents/chapter5/maritime-transportation/vessel-size-groups/>

### 4.1.3 Route segments

A complete overview of all dry bulk shipping routes are presented in Table 4.2. For the purpose of this thesis and in order to eliminate as much unapplicable dry bulk tonnage as possible, we have chosen the C3 route from Tubarao to Qingdao. This is handled strictly by Capesize vessels or larger, and is considered a benchmark trade by commercial operators, with reference rates quoted weekly.

### 4.1.4 One-step-ahead forecasting

A forecast horizon may apply to anything between a day to several years ahead. However, considering the general volatility observed in the Capesize segment as a whole, and in particular the rate movements on the C3 route (as depicted in Figure 4.2), there is limited interest or practical application for the prediction of very short (i.e. daily) forecasting. Moreover, longer forecasting horizons will require substantially longer time-series to generate a satisfactory number of data points. Consequently, we have identified a forecasting horizon of one week as the most attractive and applicable to our study.



**Figure 4.2:** Spot rate C3 route

**Table 4.2:** Overview of dry bulk shipping routes<sup>5</sup>

Notation	Description	Feature
<i>Capesize</i>		
C2	Tubarao to Rotterdam	Yes
C3	Tubarao to Qingdao	Yes
C5	West Australia to Qingdao	Yes
C7	Bolivar to Rotterdam	No
C8.14	Gibraltar/Hamburg transatlantic round voyage	No
C9.14	Continent/Mediterranean trip China-Japan	No
C10.14	China-Japan transpacific round voyage	No
C14	China-Brazil round voyage	Yes
C16	Revised backhaul	No
C17	Saldanha Bay to Qingdao	Yes
<i>Panamax</i>		
P1A.82	Panamax Skaw-Giv transatlantic round voyage	No
P2A.82	Panamax Skaw-Gib trip to Taiwan-Japan	No
P3A.82	Panamax Japan-S. Korea Transpacific round voyage	No
P4.82	Panamax Japan-S. Korea trip to Skaw Passero	No
P5.82	Panamax South China, Indonesian round voyage (BEP Asia)	No
P6.82	Panamax Singapore round voyage via Atlantic	No
P7	Panamax USG to Qingdao grain 66,000 MT	No
P8	Panamax Santos to Qingdao grain 66,000 MT	No
P1A.03	Panamax 74 Skaw-Gib transatlantic round voyage 74,000 MT	No
P2A.03	Panamax 74 Skaw-Gib trip to Taiwan-Japan 74,000 MT	No
P3A.03	Panamax 74 Japan-S. Korea Transpacific round voyage 74,000 MT	No
<i>Supramax</i>		
S1B.58	Canakkale trip via Med or BI Sea to China-South Korea	No
S1C.58	US Gulf trip to China-south Japan	No
S2.58	North China one Australian or Pacific round voyage	No
S3.58	North China trip to West Africa	No
S4A.58	US Gulf trip to Skaw-Passero	No
S4B.58	Skaw-Passero trip to US Gulf	No
S5.58	West Africa trip via east coast South America to north China	No
S8.58	South China trip via Indonesia to east coast India	No
S9.58	West Africa trip via east coast South America to Skaw-Passero	No
S10.58	South China trip via Indonesia to south China	No
<i>Handysize</i>		
HS1.38	Skaw-Passero trip to Rio de Janeiro-Recalada	No
HS2.38	Skaw-Passero trip to Boston-Galveston	No
HS3.38	Rio de Janeiro-Recalada trip to Skaw-Passero	No
HS4.38	US Gulf trip via US Gulf or north coast South America to Skaw-Passero	No
HS5.38	South East Asia trip to Singapore-Japan	No
HS6.38	North China-South Korea-Japan trip to North China-South Korea-Japan	No
HS7.38	North China-South Korea-Japan trip to south east Asia	No

<sup>5</sup><https://www.balticexchange.com/en/data-services/market-information0/dry-services.html>

## 4.2 Description of raw data

For the purpose of this study, we have employed a combination of data from alternative sources. We have retrieved market-related data from the Clarksons SIN database, such as historical spot-, trip- and time-charter rates, in addition to relevant information on the Capesize Bulker world fleet. Static and dynamic voyage-related data from the AIS database have been provided by the NCA. However, due to time limitations, computer processing and data accessibility constraints, only a part of the desired data has been retrieved for further processing and assembly.

### 4.2.1 Capesize Bulker Data

The data extracted from the Clarksons SIN database are twofold; (1) Vessel type classification data and (2) time-series data of historical spot-, trip- and time-charter rates. The following two subsections will provide a more detailed description of the datasets.

#### 4.2.1.1 Vessel type classification data

The original Capesize vessels database extracted from the SIN platform contains all registered Capesize vessels in the Clarksons database. After reviewing the database, we conclude that it includes all dry bulk vessels with a capacity range from approximately 120,000 dwt up to 400,000 dwt. We understand that the original database contains relevant information for vessels not only classified as Capesize but also includes valuable information on the VLOC and ULOC fleets. After performing specific data processing techniques on the database, we have divided it into three separate databases containing only the necessary information for each vessel category, i.e., Capesize, VLOC, and ULOC. Table 4.3 presents and compares all vessel databases, both the original and the extracted and processed versions. It is important to note that the processed version of the Capesize database (as provided in Table 4.3 below) only comprises information on the vessels included in the AIS dataset.

**Table 4.3:** Vessel type and size indication, with corresponding aggregated data from Clarksons Research Services database<sup>6</sup>

Vessel Type	Vessel Capacity [dwt]	No. of Vessels	Total Fleet Capacity [dwt]
<i>Clarksons SIN Original Database</i>			
All	120,397 - 403,919	1,717	349,476,969
<i>Capesize Database</i>			
Capesize	120,397 - 216,656	1,466	271,720,962
<i>Very Large Ore Carrier Database</i>			
Very Large Ore Carrier	215,790 - 299,688	132	34,441,498
<i>Ultra Large Ore Carrier Database</i>			
Ultra Large Ore Carrier	300,000 - 403,919	117	42,964,353
<i>Capesize Database - Processed</i>			
Capesize	120,397 - 216,656	1,301	240,407,734

### 4.2.1.2 Historical time-series data

In addition to the vessel database, we have also obtained extensive time-series data of historical spot-, trip- and time-charter rates from the Clarksons SIN database. The original datasets were very comprehensive, covering a multitude of rates and index values at a weekly frequency. Some of the information dates back to 1999, while more data becomes available around 2009, or even as late as 2014. Due to the quantity of the readily available historical time-series data, a series of processing techniques and methods have been employed to understand and apply the data as input variables to the machine learning model.

**Table 4.4:** Correlation matrix for time charter and trip charter rates

	<b>Bol/Rot</b> \$/Tonne	<b>Aus/Qin</b> \$/Tonne	<b>Tub/Qin</b> \$/Tonne	<b>Tub/Rot</b> \$/Tonne	<b>Cont-FE</b> \$/Day	<b>Tr.Atl R/V</b> \$/Day	<b>FE-Cont</b> \$/Day	<b>Tr.Pac R/V</b> \$/Day	<b>Trip Rates</b> \$/Day
<b>Bol/Rot</b>	1.000000								
<b>Aus/Qin</b>	0.950016	1.000000							
<b>Tub/Qin</b>	0.954561	0.969767	1.000000						
<b>Tub/Rot</b>	0.988988	0.950287	0.953138	1.000000					
<b>Cont-FE</b>	0.933902	0.918133	0.910579	0.921330	1.000000				
<b>Tr.Atl R/V</b>	0.909627	0.850740	0.822159	0.901664	0.954902	1.000000			
<b>FE-Cont</b>	0.747104	0.706187	0.658977	0.770724	0.836398	0.908444	1.000000		
<b>Tr.Pac R/V</b>	0.859625	0.889442	0.830129	0.864048	0.941887	0.942015	0.899288	1.000000	
<b>Trip Rates</b>	0.903680	0.881384	0.847998	0.902822	0.974178	0.985676	0.928376	0.977158	1.000000

Firstly, we observe that the historical rates are not provided on a universal scale. For instance, various trip rates are registered and stored as dollars per day, while other rates are presented in dollars per tonne. Examples of different rates and units of measurement are shown in Table 4.4 above. The table also illustrates the statistical correlation between various rates. After investigating the correlation matrix, we find that the most excellent correlation is between the *Bol/Rot* and *Tub/Rot* rates. Continuously meanwhile, the least similarity is between the *Tub/Qin* and *FE-Cont*, with correlation factors of 0.988988 and 0.658977, respectively.

In Figure 4.3 we plot two different rates, with different units of measurement against each other, to visualize the comparison between them. The figures indicate the existence of somewhat similar trend developments. However, there are also periods where the relative differences vary significantly.

Table 4.5 provides descriptive statistics for the previously mentioned correlation matrix. It is apparent after assessing the presented results that the average correlation between the different rates is significant. Indeed, except for the least two correlating rates, they all indicate a correlation factor of 0.7 or higher.

<sup>6</sup>These values are extracted based on vessel capacity for the respective vessel type, from a database on Capesize vessels provided by Clarksons Research Services Limited 2021

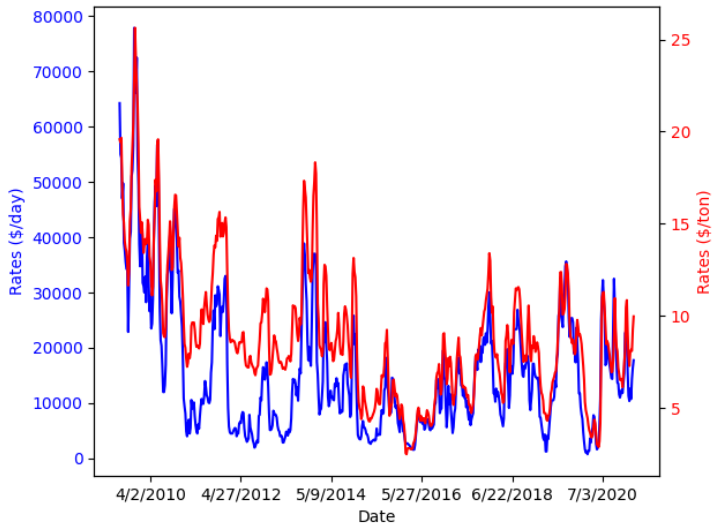


Figure 4.3: Historical plot of two different spot rates

Table 4.5: Descriptive statistics of correlation matrix in Table 4.4

	Bol/Rot	Aus/Qin	Tub/Qin	Tub/Rot	Cont-FE	Tr.Atl R/V	FE-Cont	Tr.Pac R/V	Trip Rates
<i>Count</i>	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000
<i>Mean</i>	0.916389	0.901773	0.883034	0.917000	0.932368	0.919470	0.828388	0.911510	0.933475
<i>Std</i>	0.916389	0.087068	0.106023	0.069947	0.046006	0.058482	0.114299	0.057225	0.053137
<i>Min</i>	0.747104	0.706187	0.658977	0.770724	0.836398	0.822159	0.658977	0.830129	0.847998
<i>25%</i>	0.903680	0.881384	0.830129	0.901664	0.918133	0.901664	0.747104	0.864048	0.902822
<i>50%</i>	0.933902	0.918133	0.910579	0.921330	0.933902	0.909627	0.836398	0.899288	0.928376
<i>75%</i>	0.954561	0.950287	0.954561	0.953138	0.954902	0.954902	0.908444	0.942015	0.977158
<i>Max</i>	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

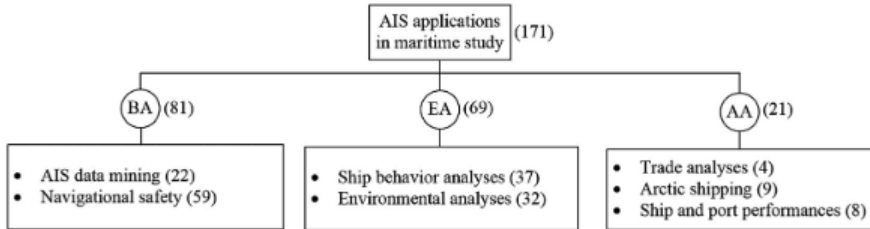
## 4.2.2 AIS Data

Automatic Identification System (AIS) was initially intended to identify hazardous maritime instances and thereby *increase the safety of ships and the environment*<sup>7</sup>. The system has since been employed as a supplementary tool with marine radar to prevent collisions at sea. AIS equipment, is recognized as an automatic tracking method transmitting real-time information over the Very High Frequency (VHF) system. According to Skauen et al. [43], AIS messages reach significantly further when emitted vertically by comparison to horizontally. The launching of AIS-enabled satellites allowed for a more comprehensive data collection, enabling more insightful research areas.

The UN’s International Maritime Organisation (IMO) has established international requirements as to which vessel types and classifications necessitate functional AIS

<sup>7</sup>[https://www.kystverket.no/en/EN\\_Maritime-Services/Reporting-and-Information-Services/ais/Automatic-Identification-System-AIS/](https://www.kystverket.no/en/EN_Maritime-Services/Reporting-and-Information-Services/ais/Automatic-Identification-System-AIS/)

equipment installed<sup>8</sup>. Therefore, a significant proportion of the world shipping fleet has installed AIS transmitters. AIS data can be registered or transferred either by exchange with nearby vessels, or through AIS base stations or satellites. There are three primary categories of data information stored in the AIS database: static, dynamic, and voyage-related. For the research conducted in this study, we employ both static and dynamic AIS data to engineer useful features so as to gain insight into the Capesize Bulk segment.



**Figure 4.4:** Overview of AIS applications in maritime research

In more recent years, as highlighted by Yang et al. [10], study and research employing AIS data for more advanced applications have proliferated as a direct result of the rapidly increasing and readily available high-quality data. As part of their study, they divide recorded AIS applications into three development stages, as illustrated by Figure 4.4. In addition, they identify four significant methodology categories; (1) data processing and mining, (2) index measurement, (3) causality analysis, and (4) operational research. Their research combines AIS data mining and trade analysis, applying it to both *basic applications* and *advanced applications*.

#### 4.2.2.1 Message content and frequency

AIS data consists of a multitude of data types<sup>9</sup>. The information is either constant or updated continuously and done either manually or automatically. Table 4.6 presents a detailed overview and description of commonly employed AIS data types. The reader will find a more comprehensive summary of all data types included in the AIS database in Appendix A.

<sup>8</sup>[https://www.kystverket.no/en/EN\\_Maritime-Services/Reporting-and-Information-Services/ais/AIS-Requirements/](https://www.kystverket.no/en/EN_Maritime-Services/Reporting-and-Information-Services/ais/AIS-Requirements/)

<sup>9</sup>Data storage format, containing either a specific type or range of values, with examples: *integers*, *strings*, *arrays*, *timestamps*, and *boolean values*. <https://techterms.com/definition/datatype>



**Table 4.6:** AIS data types

Information type	Description
MMSI	Vessel ID - Maritime Mobile Service Identity
Unixtime	Number of seconds elapsed, since 1 January 1970
Vessel dimensions	Length and breadth - measured in meters
Draught	Value of reported draught - measured in meters
IMO number	Additional ID number
Voyage origin	Origin of current voyage
Voyage destination	Destination of current voyage
ETA	Estimated time of arrival, current voyage
Vessel type	Vessel type category
Nav-status	Vessel navigational status, indicating ship activity

The frequency of the reporting intervals varies according to operational status. For all static and voyage-related data types, the reporting intervals are every 6 minutes or when data has been amended or requested<sup>10</sup>. For dynamic data, the reporting intervals are adjusted according to different operating conditions, this type of data is reported with substantially higher frequency. Depending on movement, Table 4.7 describes the different intervals for the recording of dynamic data.

**Table 4.7:** Reporting intervals of dynamic AIS data

Operational condition	Reporting interval	
	Not changing course	Changing course
At anchor or moored and moving less than 3 knots	3 min	3 min
At anchor or moored and moving faster than 3 knots	10 sec	10 sec
0 to 14 knots	10 sec	3 1/3 sec
14 to 23 knots	6 sec	2 sec
Over 23 knots	2 sec	2 sec

#### 4.2.2.2 Data extraction

Co-advisor for this thesis, Bjørnar Smestad, has primarily conducted the process of extracting the necessary data from the AIS database. In Appendix E.3 the reader will find a draft script identifying the desired data requested from the AIS SQL database by the author. With limited experience with SQL databases, compiling a draft script to retrieve the necessary data was done to gain a better understanding of what data was available and therefore to determine what type of data would be gainfully employed in continuation.

Considering that a significant portion of the world shipping fleet is required to operate with functioning AIS equipment, it is understandable that the AIS database contains an enormous amount of data. So as to avoid excess outputs, the process of identifying only the necessary data to extract from the AIS database is fundamental. The AIS database contains data for a range of different ship types, with a total of 99 different vessel categories registered<sup>11</sup>. The first digit however, pro-

<sup>10</sup>[https://arundaleais.github.io/docs/ais/ais\\_reporting\\_rates.html](https://arundaleais.github.io/docs/ais/ais_reporting_rates.html)

<sup>11</sup><https://api.vtexplorer.com/docs/ref-aistypes.html>

vides an indication of the vessel type, as presented in Table 4.8. Since we are only concerned with the world Capesize fleet, we have focused on vessel type 7, namely *cargo vessels*.

**Table 4.8:** AIS ship types

First digit	Vessel type
1	Reserved for future use
2	Wing in ground (WIG)
3	Other vessels
4	High speed craft (HSC)
5	Special craft
6	Passenger ships
7	Cargo vessels
8	Tankers
9	Other ship types

#### 4.2.2.3 Data assembly and preprocessing

The original data extracted from the AIS database contained various excess information that required substantial processing. The structure of the retrieved raw data is presented in Table 4.9. Certain elements are either too difficult to interpret or do not serve any apparent use for further examination. The *unixtime* measurement for instance, is a commonly used system for describing a point in time, with the value indicating the number of lapsed seconds since the Unix epoch<sup>12</sup>. However, since this value is not easy to interpret independently, we convert this data into readable dates. Moreover, data components like *heading*, *cog* and *rot* will not be used for further exploration and are therefore removed from the final dataset.

**Table 4.9:** Structure of the original data retrieved from the AIS database

MMSI	Unixtime	Latitude	Longitude	Heading	SOG	Nav-status	COG	ROT
374705000	1557246594	9.912683333333333	-14.819366666666667	3	13.10	0	3.0	0.0
372634000	1553030915	-31.181116666666667	43.646633333333333	64	9.39	0	60.0	0.0
566358000	1571491226	-17.788761666666667	116.70552166666667	171	13.19	0	171.5	0.0

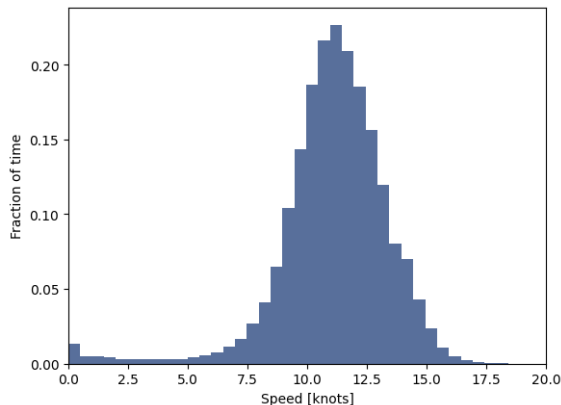
In Table 4.10 we present the structure of the post-processed AIS data. We immediately observe that this dataset is significantly more readable and contains no excess data, allowing accurate and efficient input to the machine learning models employed.

**Table 4.10:** Structure of the post-processed AIS data

MMSI	Date	Year	Week	Latitude	Longitude	SOG	Nav-status
210615000	2019-05-27 18:02:16	2019	22	1.1262	60.30366	12.0	0
256848000	2019-06-20 19:48:52	2019	25	-23.0520	152.80703	14.0	0
247283400	2019-11-11 06:53:38	2019	46	-17.6322	62.40476	4.59	3

<sup>12</sup><https://www.unixtimestamp.com/>

After conducting a short investigation into the registered vessel velocities, we found a series of abnormal speed recordings. We therefore determined to drop all recordings with anomalous speed records ( $>25$  knots) from the dataset. Figure 4.5 illustrates a histogram plot of all filed vessel velocities.



**Figure 4.5:** Velocity histogram of worldwide AIS recordings ( $>25$  knots)

Table 4.11 presents an overview of the data file pre-, peri- and post-processing. It shows a significant amount of excessive data removed from the original file, as we managed to reduce the total data size by 27.69%. Importantly, these numbers are based solely on the data retrieved from 01.01.2019 to 31.12.2019. Due to computer-processing limitations and memory capacities, it was necessary to split the final dataset into four further parts.

**Table 4.11:** AIS data file comparison pre-, peri- and post-processing

	No. of rows	No. of columns	Size
<i>Raw AIS data-file</i>	62,914,023	9	5.49 GB
<i>AIS data processed once</i>	62,914,023	7	4.88 GB
<i>AIS data processed twice</i>	51,264,153	7	3.97 GB
<i>Processed data - Split 1</i>	12,868,382	7	0.99 GB
<i>Processed data - Split 2</i>	12,766,109	7	0.98 GB
<i>Processed data - Split 3</i>	12,835,156	7	0.99 GB
<i>Processed data - Split 4</i>	12,794,506	7	0.99 GB

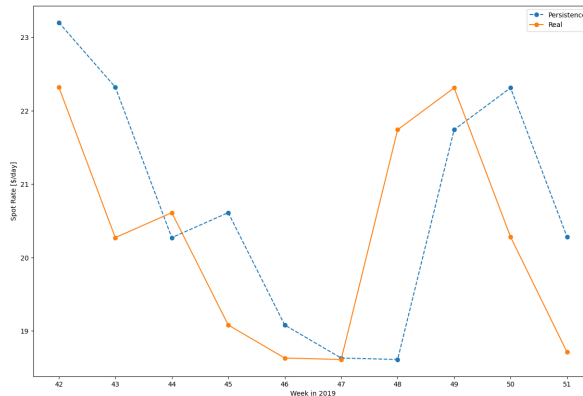
After conducting a thorough investigation of the data and performing the necessary processing methods to avoid excess information, we concentrated on the remaining material. In Figure 3.1 we depict a scatter plot of all recorded vessel locations for the year 2019. Other than the fact that we can confirm that we have retrieved AIS data from the entire world, there is not much insight to gain from examining this illustration. Figure 3.2 on the other hand, provides a better understanding of the data at large. The figure depicts a density map in a scatter format, illustrating the main routes traversed by the world fleet and central port locations.

## 4.3 Results

This section of the chapter will display the findings from the examination conducted as part of this thesis. Initially, the results from the benchmark model, i.e., the persistence model described in Section 3.4.1, will be presented. These results will provide the necessary means for evaluating the performance of the various applied machine learning algorithms in this study. In addition, we will present the findings from the different feature selection processes and the results from the hyperparameter tuning approach. Lastly, we show all produced forecasts.

### 4.3.1 Benchmark model

To produce a benchmark model for this thesis, we employed the persistence model technique. As previously presented, this model is used because of its level of simplicity and fast implementation. The results of the persistence model are depicted in Figure 4.6.



**Figure 4.6:** Persistence model forecasting results

### 4.3.2 Feature selection

In Section 3.2.3, we stated that this thesis will use three different sets of feature combinations as input to the different machine learning models. We identified these feature combinations as: *all features*, *top features*, and *selected features*. The *all features* combination is highly self-explanatory; it utilizes the complete set of all constructed features as part of this thesis. In contrast, the *top features* composition only utilizes the top 20 established features. These features are presented in Table 4.12. The table also includes the list of the *selected features* combination. Features in the table without any asterisk annotation are featured in both sets. Elements marked with one or two asterisk symbols however, indicate that they are only employed in the *top* or *selected* feature sets, respectively. The feature  $price(t)$ , i.e., the dependent variable, was originally included in both sets, but is

removed from the different feature combination training sets. Consequently, the employed training sets for the *top* and *selected* feature combinations contains only 19 features. The *top* feature set consists of 47.4% market-derived features, while the remaining 52.6% features are derived from AIS. By comparison, the *selected* feature combination is built up from 68.4% market-derived features and 31.6% AIS-derived features.

**Table 4.12:** Composition of different feature combination sets

Top 20 features	Selected 20 features
avg_rate(t)	avg_rate(t)
BCLC17(t)	BCLC17(t)
avg_rate(t-1)*	Bdry(t)**
tubarao_cap(t)*	BCLC2(t-1)**
Med_pct(t)*	BCLC5(t)**
tubarao(t)*	BCLC14(t-1)**
BCLC14(t)	BCLC14(t)
price(t-1)	price(t-1)
nouadhibou_cap(t)*	BCLC17(t-1)**
Bdry(t-1)	Bdry(t-1)
indi_pct(t-1)*	portcartier(t-1)**
nouadhibou(t)*	atlantic(t-1)**
portcartier_cap(t)*	Med(t-1)**
BCI(t-1)*	atlantic_pct(t)**
BCLC2(t)	BCLC2(t)
portcartier(t)*	CP_pct(t-1)**
BCI(t)	BCI(t)
Med_cap(t)*	Med_pct(t-1)**
EstP_pct(t-1)*	price(t+1)**

### 4.3.3 Hyperparameter optimization

Earlier, in Section 3.3.4, we describe the employed method used in this thesis to tune machine learning algorithms. Table 3.4 provides an overview of the different investigated hyperparameters and the respective search ranges. In Table 4.13, we have presented the optimal combination of the investigated hyperparameters for the various machine learning algorithms and corresponding feature combinations.

**Table 4.13:** Optimal hyperparameter combinations

Model	Feature Set	Hyperparameter 1	Hyperparameter 2
Linear Ridge Regressor	All	Alpha = 0.01	Solver = auto
	Top	Alpha = 0.01	Solver = saga
	Selected	Alpha = 0.01	Solver = saga
Random Forest Regressor	All	Max depth = 14	N estimators = 50
	Top	Max depth = 9	N estimators = 20
	Selected	Max depth = 7	N estimators = 50
Long Short-Term Memory	All	Batch size = 128	Epochs = 50
	Top	Batch size = 64	Epochs = 50
	Selected	Batch size = 8	Epochs = 75

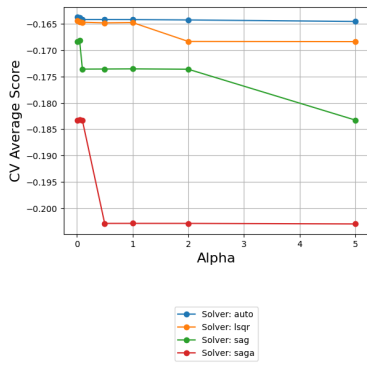
Moreover, below we have included visual illustrations of the grid search results for the different models used in this thesis. Figure 4.7a through Figure 4.7f, presents the mean and standard deviation scores from conducting a grid search to explore the hyperparameters of the linear ridge regression model. The figures also denote what feature combination set has been employed.

Similarly, all subfigures of Figure 4.8 visually illustrate the results of the grid search examination of the random forest regression model. The caption of each subfigure indicates both the score type, i.e., mean or standard deviation, and the respectively feature set employed.

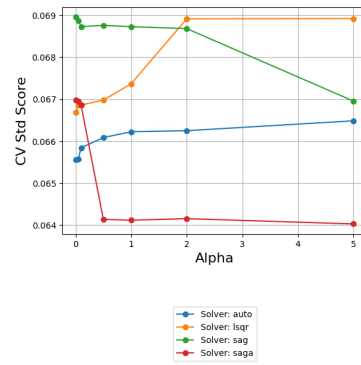
Figure 4.9a through Figure 4.9f present the findings from the grid search investigation of the final algorithm utilized in this thesis, i.e., the long short-term model. Like our previously explained figures, the LRR and RFR models respectively, these figures also indicate which feature combination has been employed.

From the figures included in this subsection, we can extract the following:

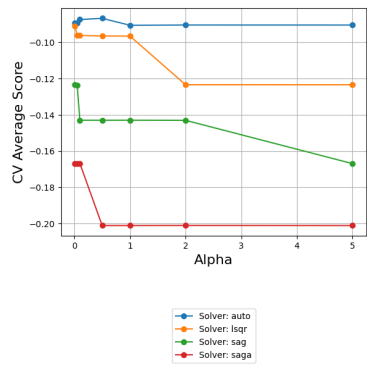
1. The *mean scores* of the grid search for the linear ridge regression model is approximately in the domain  $[-0.2, -0.06]$ . The corresponding *standard deviation scores* are primarily in the range  $[0.02, 0.07]$ .
2. The *mean scores* of the grid search for the random forest regressor model is found in the  $[-0.18, -0.1275]$  region. With the corresponding *standard deviation scores* observed in the domain  $[0.04, 0.075]$ .
3. Lastly, the *mean scores* of the grid search for the long short-term memory model appears in the domain  $[-0.32, -0.075]$ . The corresponding *standard deviation scores* are found in the range  $[0.04, 0.13]$ .



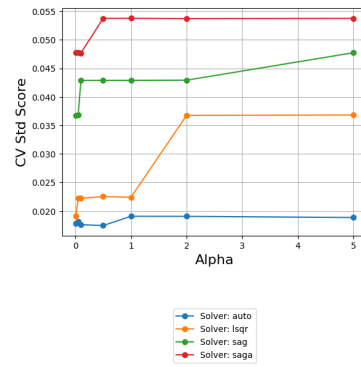
(a) Mean test scores: all features



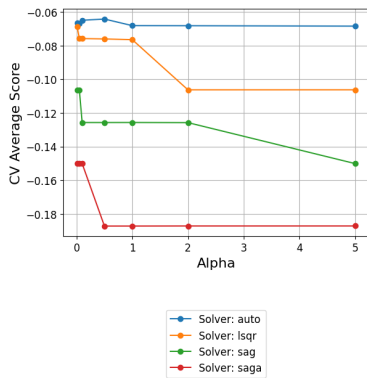
(b) Std. test scores: all features



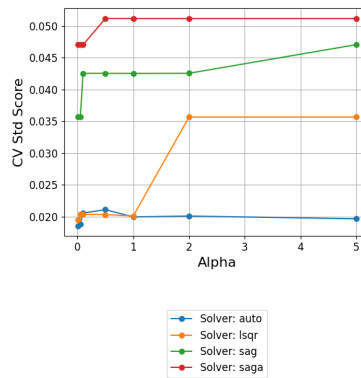
(c) Mean test scores: top features



(d) Std. test scores: top features

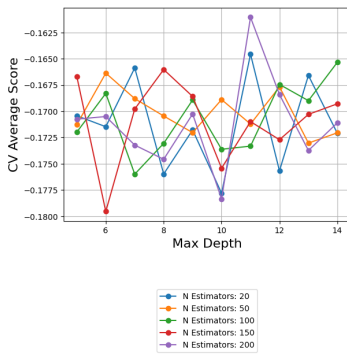


(e) Mean test scores: selected features

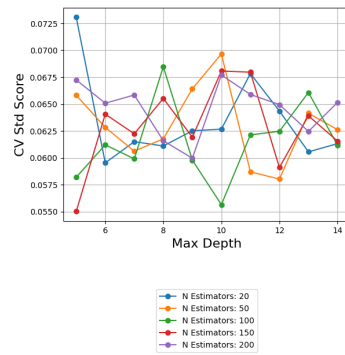


(f) Std. test scores: selected features

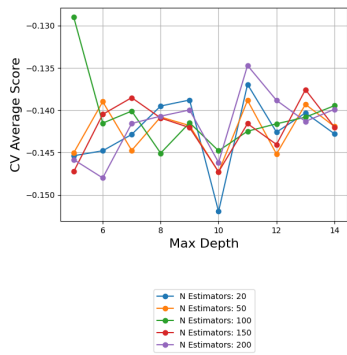
Figure 4.7: Grid search scores for linear ridge regression model



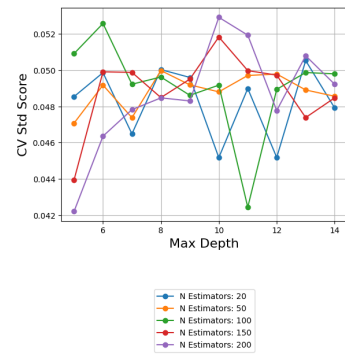
(a) Mean test scores: all features



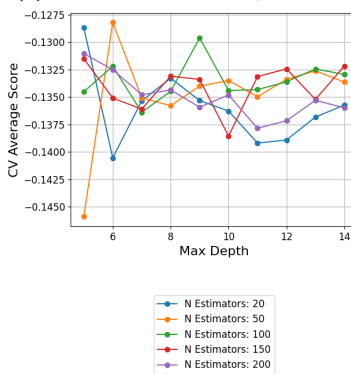
(b) Std. test scores: all features



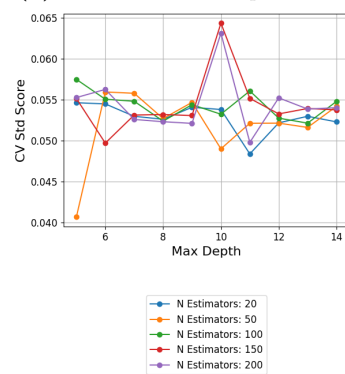
(c) Mean test scores: top features



(d) Std. test scores: top features



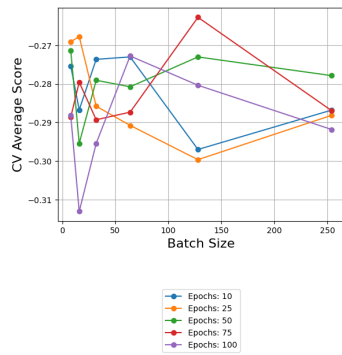
(e) Mean test scores: selected features



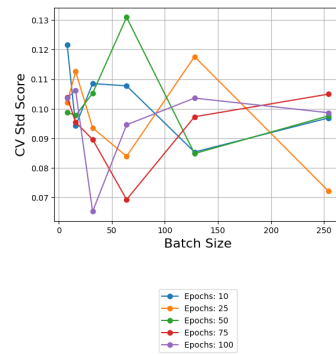
(f) Std. test scores: selected features

**Figure 4.8:** Grid search scores for random forest regression model

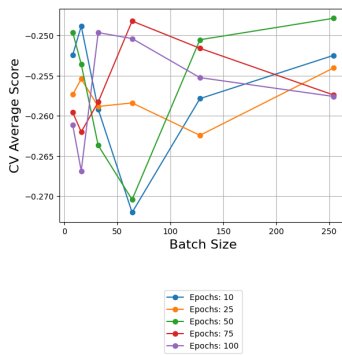




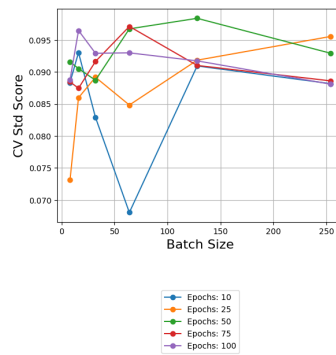
(a) Mean test scores: all features



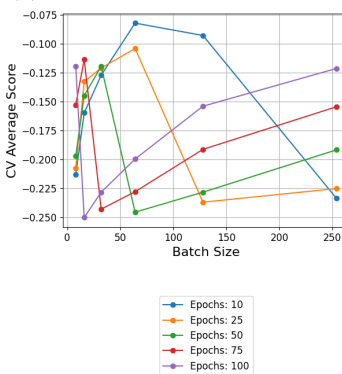
(b) Std. test scores: all features



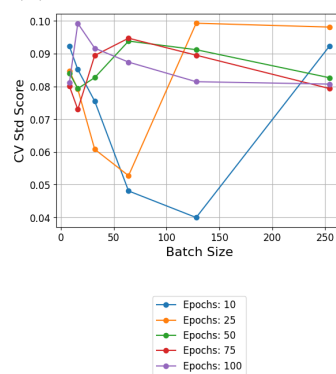
(c) Mean test scores: top features



(d) Std. test scores: top features



(e) Mean test scores: selected features



(f) Std. test scores: selected features

Figure 4.9: Grid search scores for long short-term memory model

#### 4.3.4 Model training

Having obtained the optimal combination of hyperparameters for the models and respective feature sets employed in this thesis, the process of training the models can be initiated. In Table 4.14, we have presented the error estimates for the various models on the training dataset, while employing scaled data input values. The results provided in the table are produced while employing the identified optimal hyperparameter combinations as given by Table 4.13.

Considering that all statistical metrics used for model evaluation in this thesis are error estimates, the objective is to minimize the error values, i.e. the lower the better. Hence, we immediately observe that the LRR model, when using *all features*, outperforms all other model and feature set combinations investigated while training the model. These results are highlighted in bold font in the table. Lastly, we observe that the best results in the table are significantly low, and might indicate the existence of overfitting.

**Table 4.14:** Performance metrics on training set for all models, with the employed data scaled in domain  $[0,1]$

Model	Feature Set	RMSE	MAE	MAPE
Linear Ridge Regressor	All	<b>0.00105</b>	<b>0.00083</b>	Inf
	Top	0.04305	0.02989	Inf
	Selected	0.02412	0.01752	Inf
Random Forest Regressor	All	0.03428	0.02590	Inf
	Top	0.04008	0.02540	Inf
	Selected	0.03044	0.02176	Inf
Long Short-Term Memory	All	0.18085	0.13183	Inf
	Top	0.19750	0.14350	Inf
	Selected	0.03559	0.03288	Inf

### 4.3.5 Model forecasting

We have produced the forecasting results by applying the optimal hyperparameter combinations for the different machine learning models on the test set. Figure 4.10a through Figure 4.10c illustrate the forecasting results for the linear ridge regression model. The results from the random forest regression model are presented in the subplots of figure Figure 4.11. Finally, Figure 4.12a through Figure 4.12c display the results formed by the long short-term model.

In addition, we have presented an overview of the error metrics used in this thesis for the corresponding model and feature combination in Table 4.15 and Table 4.16. While both tables demonstrate the performance metrics for the different forecasting results, Table 4.15 presents the results while employing scaled values, purposely included to compare with the findings given in Table 4.14. In Table 4.16 on the other hand, the performance metrics are calculated with the inverse scaled values, i.e. the real forecasted estimates, for comparison with the persistence model.

**Table 4.15:** Performance metrics on test set for all models, with the employed data scaled in the domain [0,1]

Model	Feature Set	RMSE	MAE	MAPE
Linear Ridge Regressor	All	0.14608	0.12550	1.52896
	Top	0.09308	0.07712	0.70141
	Selected	0.08932	0.06023	0.35733
Random Forest Regressor	All	0.08441	0.06633	1.07919
	Top	0.07169	0.05499	0.75042
	Selected	0.08496	0.06328	0.77296
Long Short-Term Memory	All	0.27967	0.21744	4.96886
	Top	0.25515	0.22622	4.71898
	Selected	<b>0.04195</b>	<b>0.03465</b>	<b>0.30806</b>

We can conclude the following from the results of Table 4.16:

1. All feature combinations of the linear ridge regression model outperform the benchmark model.
2. All feature combinations of the random forest regression model outperform the benchmark model.
3. All feature combinations of the long short-term memory model outperform the benchmark model, with respect to the RMSE performance metrics.
4. The persistence model outperform the LSTM + top features combination, with respect to MAE and MAPE performance levels.
5. The long short-term memory model with the *selected features* combination produces the best forecasting according to all performance metrics employed in this study.
6. The RFR model produce the lowest mean RMSE values

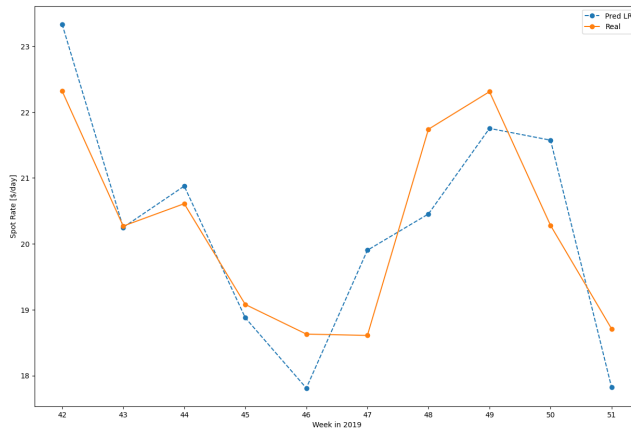
**Table 4.16:** Performance metrics on forecast results for all models

Model	Feature Set	RMSE	MAE	MAPE	RMSE*
Persistence Model	N/A	1.56025	1.25699	6.16010	1.56025
Linear Ridge Regressor	All	0.88964	0.76430	3.78948	0.66501
	Top	0.56611	0.46916	2.33206	
	Selected	0.53928	0.36498	1.83010	
Random Forest Regressor	All	0.46571	0.37224	1.79907	<b>0.46942</b>
	Top	0.44551	0.33029	1.61718	
	Selected	0.49704	0.33850	1.60570	
Long Short-Term Memory	All	1.05658	0.81617	3.94328	0.95530
	Top	1.55385	1.37769	6.74109	
	Selected	<b>0.25548</b>	<b>0.21099</b>	<b>1.04014</b>	

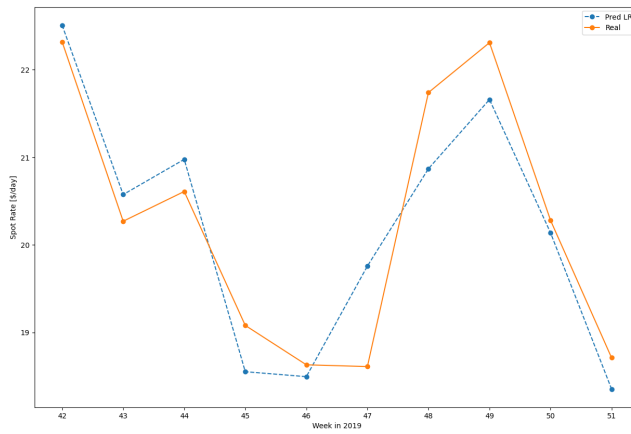
*Note: RMSE\* indicates the mean RMSE score for the different models*

#### 4. COMPUTATIONAL STUDY

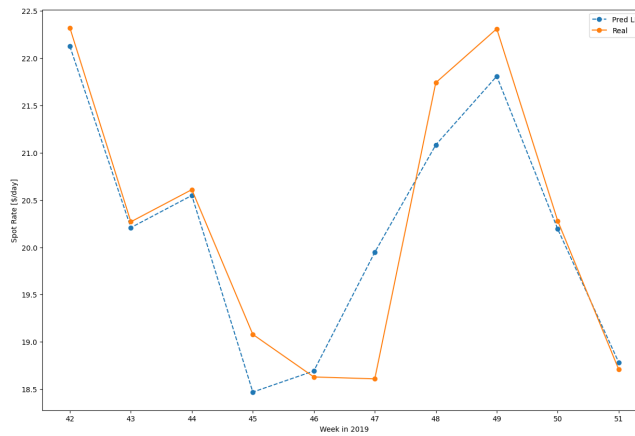
---



(a) All features



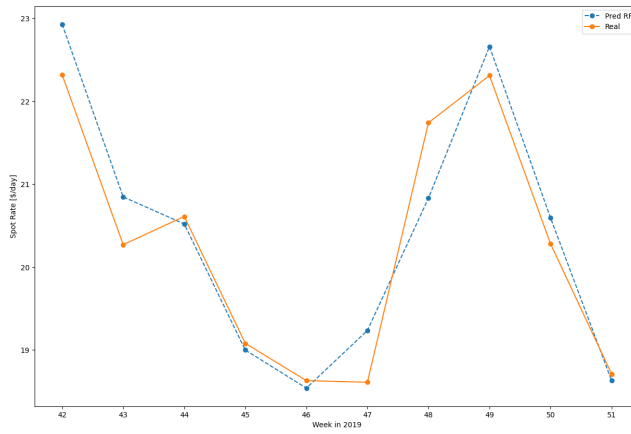
(b) Top features



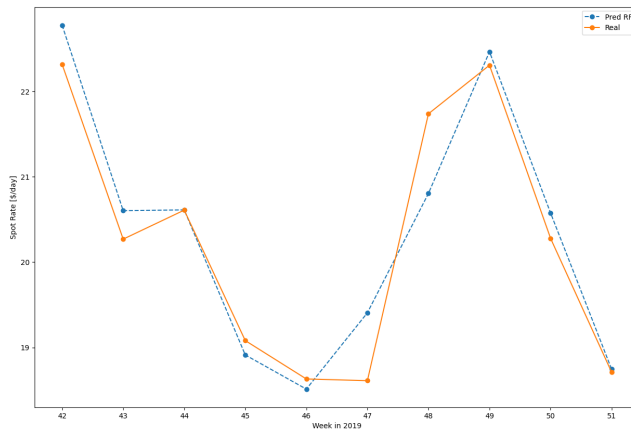
(c) Selected features

**Figure 4.10:** Linear ridge regression model forecast for different feature subsets

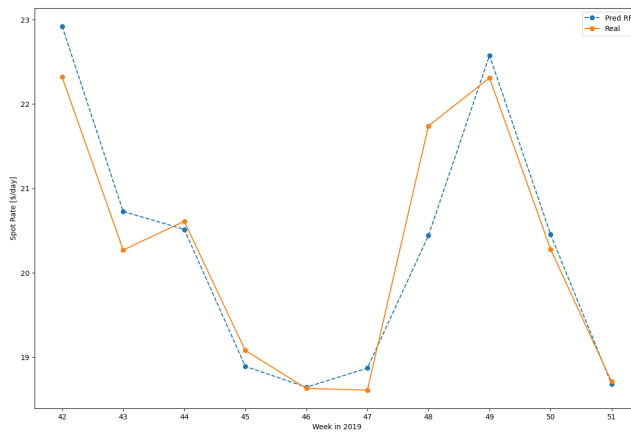
---



(a) All features



(b) Top features

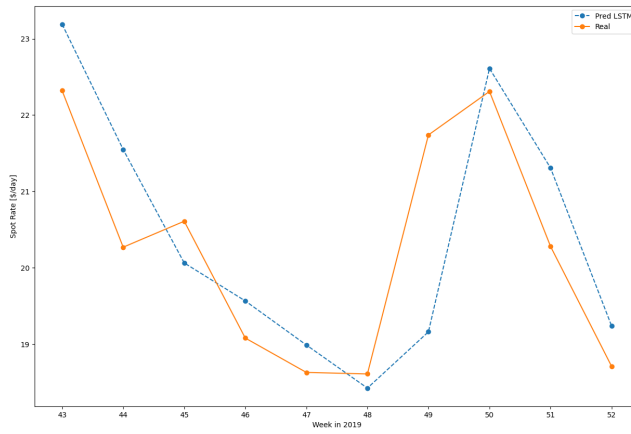


(c) Selected features

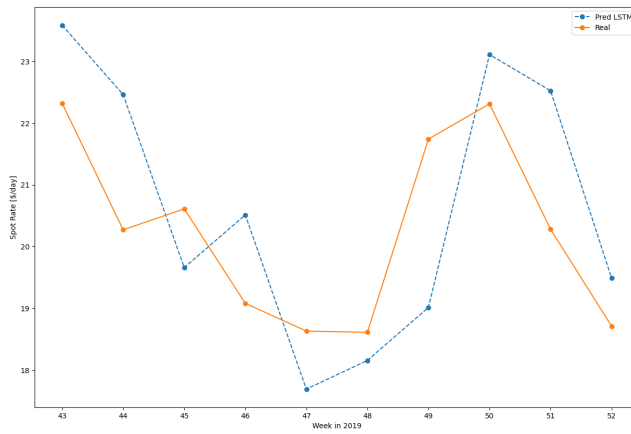
**Figure 4.11:** Random forest regression model forecast for different feature subsets

#### 4. COMPUTATIONAL STUDY

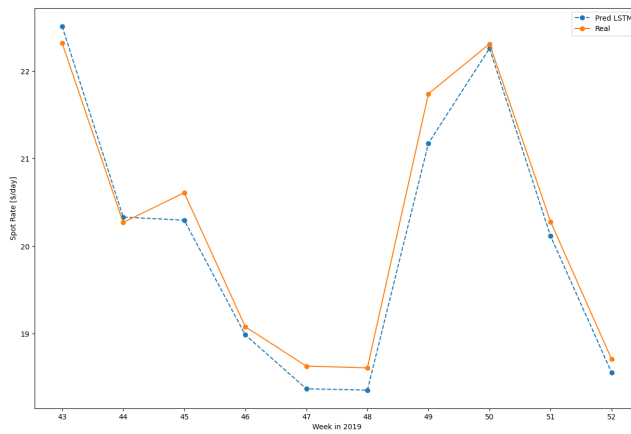
---



(a) All features



(b) Top features



(c) Selected features

**Figure 4.12:** Long short-term memory model forecast for different feature subsets

---

### 4.3.6 Train vs test performance

In previous Section 4.3.4, we present the findings (Table 4.14) of the performance metrics for the various employed machine learning models with corresponding feature set combination on the training set alone. While in Section 4.3.5, the performance metrics on the test set are given (Table 4.15). In Figure 4.13, Figure 4.14, and Figure 4.15, these results are visually illustrated, to provide a better understanding of the RMSE performance progression for the different model and feature combinations.

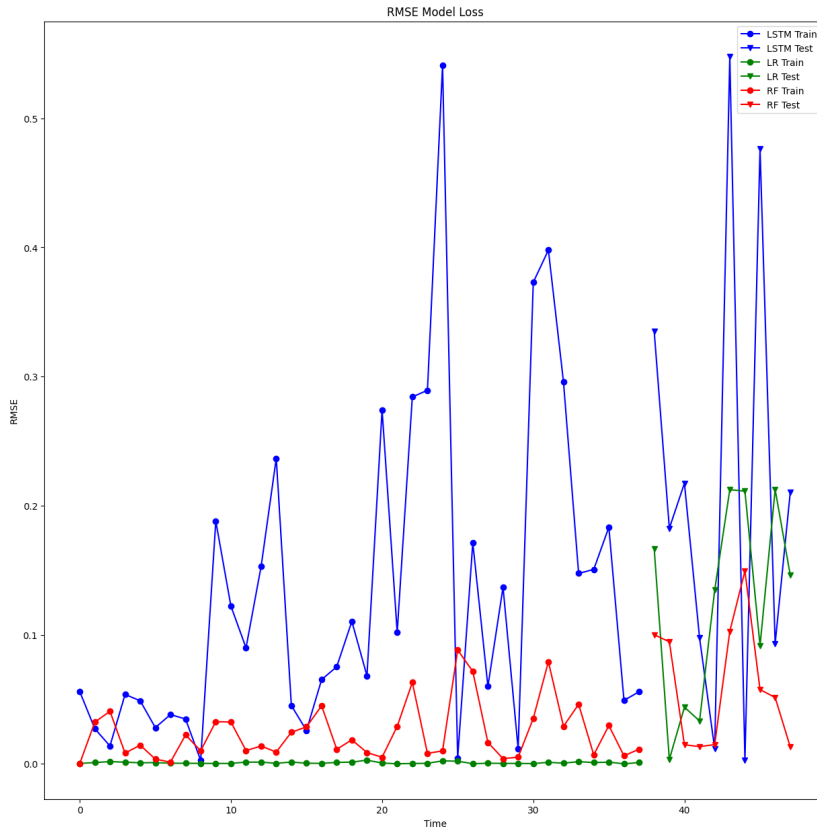


Figure 4.13: RMSE performance measurements utilizing all features



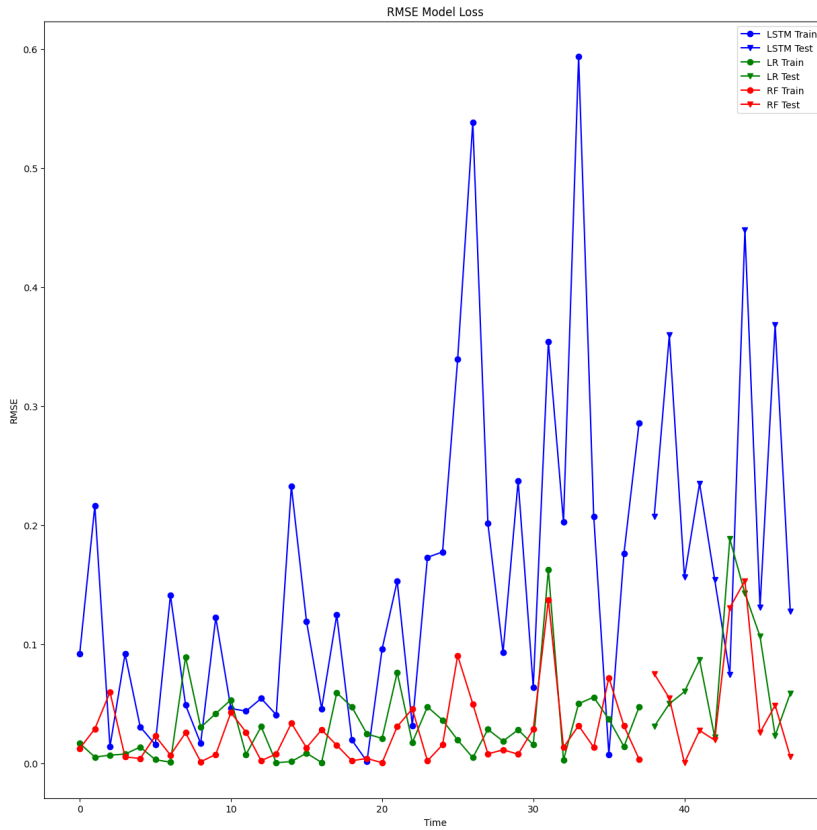
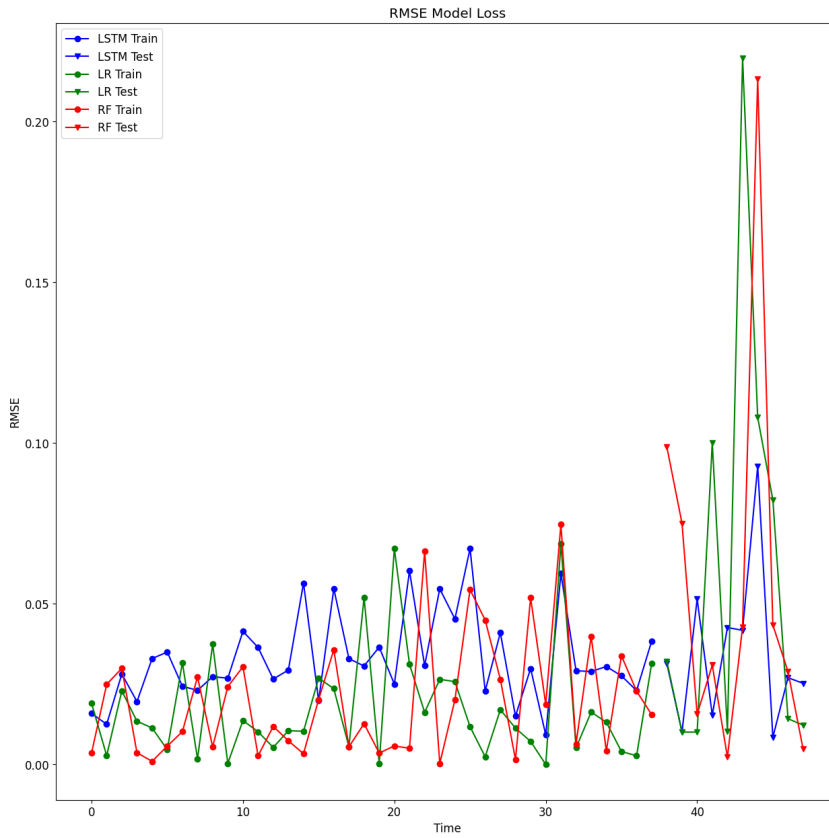


Figure 4.14: RMSE performance measurements utilizing top features



**Figure 4.15:** RMSE performance measurements utilizing selected features



# Discussion

This chapter will provide a discourse on the forecasting results produced and the described approach employed to address the objective of this thesis. The discussion of the employed method will focus on the following: the adopted hyperparameter tuning approach, the feature selection processes used, and the feature construction practice. Finally, we address the topic of encountered limitations as well as considerations for future research.

## 5.1 Evaluation of forecasting results

Despite limited data available from the NCA AIS database, this thesis has managed to produce interesting forecasting results of the short-term freight rate movements on the C3 Capesize route from Tubarao to Qingdao, using AIS- and market-derived information. The employed machine learning models have managed to compile results in accordance with the intended purpose of this study, as illustrated by Figure 4.10a through Figure 4.12c.

As presented in Table 4.14, we have found performance measurements for the respective models on the training dataset. Although these measurements do not indicate the predictiveness in the different models, they can help deduce whether or not a model is overfitted, i.e. if the model learns too much and therefore becomes unreliable. We have observed that a selection of the model- and feature-combinations demonstrate high-performance levels ( $RMSE < 0.1$ ) with the following: all LRR combinations, all RFR combinations, and the LSTM model with the *selected* feature combination. Comparing these performance levels with the computed measurements for the corresponding model- and feature-combinations on the test set presented in Table 4.15, indicates the existence of overfitting. In Figure 4.13, Figure 4.14, and Figure 4.15 we have visualized the RMSE performance levels for the different models and corresponding feature sets, for both training and test periods.

We can conclude a case of overfitting especially in Figure 4.13 for the linear ridge regression model. The figure correctly demonstrates the RMSE performance scores for both training and testing. However, one can clearly observe that the RMSE is relatively non-existent for the training set, whilst for the test set the RMSE values are substantially larger. This collaborates well with the results in Table 4.14 and Table 4.15. In addition, we also identified the existence of minor overfitting cases in other model and feature combinations, namely: *LRR + selected features*, *RFR + all features*, and *RFR + selected features*. Consequently, of all tested algorithms only the LSTM model does not exhibit any case of overfitting.

Applying the models to the dataset, we investigated the predictiveness for comparison with results produced by the benchmark model, i.e. the persistence model. After examining the results presented in Table 4.16, we have observed that **all** employed machine learning models and feature combinations outperform the persistence model under evaluation against the RMSE performance metric. However, the table also implies that the persistence model manage to outperform only the LSTM model with top features combination when compared to the MAE and MAPE performance measurements. Furthermore, we recognize that the *LSTM + selected features* combination delivers the highest performing forecasting results. Interestingly, the remaining two LSTM combinations produce significantly worse forecasting estimates. Careful examination of Table 4.16 reveals that the LSTM model produced the most varying performance estimates by comparison to the LRR and RFR models. Indeed, the RFR model is generally recognized for producing the least fluctuating error measurements, and achieves in addition the lowest mean RMSE score for all experimented feature combinations. Lastly, it is challenging to identify any recognizable pattern when inspecting the performance levels for the different feature sets. For instance, the results from the LRR model show that moving from *all* features to *top* features and subsequently *selected* features, yields more accurate prediction estimates. In the case for the RFR model however, one can observe that the *selected* features combination yields the least accurate forecasting outputs.

In consideration of the applicability of machine learning algorithms in predicting short-term freight rate movements on the C3 route in the Capesize bulk segment, all selected models investigated in this thesis indicate capabilities in predictive-ness. In addition, we have constructed sets of elements that presumably influence the Capesize bulk segment, allowing the employed machine learning algorithms to produce relatively accurate predictions. However, there is no evidence to support which specific set of elements that together influence this shipping segment best. Interestingly, all examined feature sets includes a combination of AIS- and market-derived data.

## 5.2 Evaluation of methodological approach

### 5.2.1 Hyperparameter optimization technique

As mentioned in Section 3.3.4, this study employs the *cross validation grid search*-technique for hyperparameter optimizing with the employment of the *negative root mean squared error*-scorer. The results from the exhaustive grid search to identify the optimal hyperparameter combinations can be found in Table 4.13. In Figure 4.7a through Figure 4.9f the mean and standard deviation scores for the different models and feature set combinations are illustrated. In theory, the employed grid search method selects the hyperparameter combination that yields the best score with regards to the selected scoring-identifier. According to Table 4.13, the optimal combination of hyperparameters for the LRR model with *all* features is;  $\alpha = 0.01$  and solver = auto. This corresponds well to the illustration in Figure 4.7a, since we employ the *negative root mean squared error*-scorer the best combination is the given by the highest score. However, we identify several cases where the presented optimal hyperparameter combination in the table do not correspond to the respectively plotted mean score figure. For instance, the table demonstrate that the optimal hyperparameter combination for the RFR model with *top* features are; max depth = 9 and n estimators = 20. Examining the respective figure, i.e. Figure 4.8c (copy of which is shown below for easy reference), one would assume the optimal combination to be; max depth = 5 and n estimators = 100, as this results in the least negative mean test score. This frequently recurring inconsistency has been exhaustively investigated without any conclusive explanation. To determine whether this has arisen from personal error or system inherit fault is presently unproductive. The author is of the opinion this will require further work in a separate study.

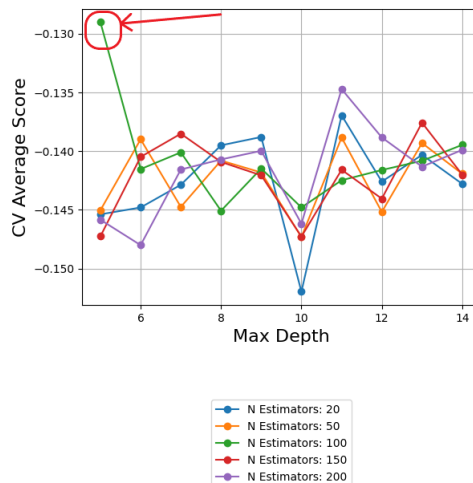


Figure 5.1: Copy of figure 4.8c

### 5.2.2 Employed feature selection methods

For the investigation conducted in this thesis, we have intentionally studied the employment of three different sets of information, i.e. a set of *all* constructed features, a set of *top* features, and lastly a set of *selected* features. From the results previously discussed concerning Figure 4.13, we identified the existence of overfitting in both the LRR and RFR models, which proves the presence of the *curse of dimensionality*<sup>1</sup>. This necessitated feature reduction to best deal with the issue of dimensionality. A feature selection process is recognized as a subprocess of feature reduction. We consequently developed and applied two different feature selection methods, described in Section 3.2.3. The resulting lists of features for the two methods, i.e. top features (inspired by Næss [6]) and selected features (developed by Koehrsen [30]), can be found in Table 4.12. Note that both feature combinations contain both AIS- and non-AIS-derived features. However, after carefully reviewing Figure D.1, which demonstrates the normalized feature importance for the top 20 features using the *selected* method, we observed that the top seven features are exclusively retrieved from market-relevant information. Similarly, we observe from Figure D.3, illustrating the top 20 features using the *top* method, the four most dominant features are also exclusively derived from market data. Interestingly, two of the top features in both selection methods are employed in both feature sets, namely: *avg.rate(t)* and *BCI-C17(t)*. Unfortunately, the limited number of experimental results and time-series data available from AIS do not provide sufficient evidence to draw any firm conclusions on whether or not this data information can be identified as significantly influencing elements in the Capesize bulk segment.

### 5.2.3 Feature construction process

The exploratory data analysis (EDA) is considered an essential milestone in the data exploration process and was conducted as a preliminary exercise to the feature engineering procedure. Performing a comprehensive EDA enabled the possibility of deriving valuable understanding of the data retrieved from the NCA AIS database, and allowed a better insight to the operational activities in the Capesize bulk segment. Consequently, the process of constructing features to utilize as input parameters in the later developed machine learning models became more efficient. However, as a result of certain elements being overlooked in the feature construction phase, later stages of the study showed that several constructed features contained misleading and disruptive information. Features demonstrating inaccurate information were still deemed useful and therefore included in the final features dataset. In addition, whereas the features constructed from AIS-derived input were primarily composed of aggregated data, it may possibly have been more constructive to produce a dynamic feature set. Further study of these circumstances is warranted and suggested by the author. Moreover, considering that the shipping industry and the respective freight rates are highly dependent on demand factors, establishing specific features with this focus for the Capesize bulk segment should be of great interest for further analysis.

---

<sup>1</sup>Definition provided in Section 3.2.3

### 5.3 Limitations and considerations

All computations performed as part of this thesis have been conducted on the author's personal computer. Consequently, processing power set by the employed computer is identified as a major constraint and limitation throughout this study.

Although we previously considered the retrieved data from the NCA AIS database as a limited data set, the actual information is extensive in data volume. As we have presented in Table 4.11, the initially retrieved data set contained more than 60 million rows of information in nine corresponding data columns, requiring a total of 5.49 GB of storage capacity. During attempts to process this AIS data we frequently experienced exceeding the memory capability of the used computer. However, in relation to a time-series window, this data set is limited to only a single year. The author is of the opinion that a longer time-series is necessary to further validate the thesis objective.

Finally, as summarized in Section 4.2, only a minor part of the requested data was made available from the NCA AIS database. Again, due to the relatively modest number of weekly observations, it is challenging to properly train the selected machine learning algorithms to produce acceptable and reliable forecasting results.





# Chapter 6

## Conclusion

For the purpose of this thesis and the construction of machine learning algorithms this study employed three forecasting models; the linear ridge regression (LRR) model, the random forest regression (RFR) model and the long short-term memory (LSTM) model. All models were explored with identical feature combinations and underwent comparable hyperparameter optimization processes.

From a total of 65 features examined, spanning numerous Capesize bulk operations from defining routes to fleet utilization and port capacity; two features were identified as influential in all feature selection processes,  $avg\_rate(t)$ <sup>1</sup> and  $BCI\_C17(t)$ <sup>2</sup>.

To optimize the respectively employed machine learning algorithms, we used a hyperparameter optimization technique: *cross validation grid search*. The use of this hyperparameter optimization method produced frequently recurring inconsistencies. Despite exhaustive investigation, no conclusive explanation for this disparity was established.

Model simulation was conducted on a Intel Core i7-7700HQ CPU, with 16 GB RAM installed. Most accurate results were provided by the LSTM model with our *selected* features set. Overall, the RFR model produced the most consistent and least varied forecasts as well as the lowest mean RMSE metrics.

All selected models investigated in this thesis, indicate applicability of machine learning algorithms in predicting short-term freight rate movements on the C3 route in the Capesize bulk segment. Moreover, varying prediction accuracy corroborates that our selected features influence rate movements. We were however unable to identify, define or prove which specific set of features that bears the greatest influence on rate predictability. This aside, all examined feature sets included a

---

<sup>1</sup>Capesize 172,000 dwt average trip rates

<sup>2</sup>Spot rate on the C17 route, from Saldanha Bay to Qingdao

combination of AIS- and market-derived data. Machine learning algorithms from single source information remains untested.

## 6.1 Further work

Based on the topics discussed in the previous Chapter 5, and conclusions provided in Chapter 6, the necessity for further research of several elements are identified. The list below presents topics that will require further work based on the author's opinion.

- Investigating the use of other hyperparameter optimization techniques, for example the *random search* or *bayesian optimization* mentioned in Section 3.3.4.
- Explore the possibilities of constructing additional features from either AIS or market-specific data, such as demand or supply related features.
- Examine the applicability of features sets containing information from only single source in the Capesize bulk segment.

# Bibliography

- [1] V. D. I. Norman. *Economics of bulk shipping*. Bergen: Institute for Shipping Research, 1979.
- [2] A. Hayes. *Perfect Competition*. 2020. URL: <https://www.investopedia.com/terms/p/perfectcompetition.asp>.
- [3] R. Adland, P. Cariou, and F.-C. Wolff. “The influence of charterers and owners on bulk shipping freight rates”. In: *Transportation Research Part E: Logistics and Transportation Review* 86 (2016), pp. 69–82. ISSN: 1366-5545. DOI: <https://doi.org/10.1016/j.tre.2015.11.014>. URL: <https://www.sciencedirect.com/science/article/pii/S1366554515002252>.
- [4] Y. Jie, A. Kersing, and Q. Xie. *Data will decide success in the next normal of bulk and tanker shipping*. 2020. URL: <https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/data-will-decide-success-in-the-next-normal-of-bulk-and-tanker-shipping>.
- [5] G. H. Århus and S. R. Salen. “Predicting Shipping Freight Rate Movements Using Recurrent Neural Networks and AIS Data-On the tanker route between the Arabian Gulf and Singapore”. MA thesis. NTNU, 2018.
- [6] P. A. Næss. “Investigation of multivariate freight rate prediction using machine learning and ais data”. MA thesis. NTNU, 2018.
- [7] H. S. N. Worldwide. *Capesize index plummets to -133, the first time ever in negative history*. 2020. URL: <https://www.hellenicshippingnews.com/capesize-index-plummets-to-133-the-first-time-ever-in-negative-territory/>.

- 
- [8] A. H. Alizadeh and W. K. Talley. “Microeconomic determinants of dry bulk shipping freight rates and contract times”. In: *Transportation* 38 (2011), pp. 561–579. DOI: <https://doi.org/10.1007/s11116-010-9308-7>. URL: <https://link.springer.com/article/10.1007/s11116-010-9308-7#citeas>.
- [9] S. Köhn and H. Thanopoulou. “A gam assessment of quality premia in the dry bulk time–charter market”. In: *Transportation Research Part E: Logistics and Transportation Review* 47.5 (2011), pp. 709–721. ISSN: 1366-5545. DOI: <https://doi.org/10.1016/j.tre.2011.01.003>. URL: <https://www.sciencedirect.com/science/article/pii/S1366554511000068>.
- [10] D. Yang, L. Wu, S. Wang, H. Jia, and K. X. Li. “How big data enriches maritime research – a critical review of Automatic Identification System (AIS) data applications”. In: *Transport Reviews* 39.6 (2019), pp. 755–773. DOI: 10.1080/01441647.2019.1649315. eprint: <https://doi.org/10.1080/01441647.2019.1649315>. URL: <https://doi.org/10.1080/01441647.2019.1649315>.
- [11] R. Adland, H. Jia, and S. P. Strandenes. “Are AIS-based trade volume estimates reliable? The case of crude oil exports”. In: *Maritime Policy & Management* 44.5 (2017), pp. 657–665. DOI: 10.1080/03088839.2017.1309470. eprint: <https://doi.org/10.1080/03088839.2017.1309470>. URL: <https://doi.org/10.1080/03088839.2017.1309470>.
- [12] Z. Yan, Y. Xiao, L. Cheng, S. Chen, X. Zhou, X. Ruan, M. Li, R. He, and B. Ran. “Analysis of global marine oil trade based on automatic identification system (AIS) data”. In: *Journal of Transport Geography* 83 (2020), p. 102637. ISSN: 0966-6923. DOI: <https://doi.org/10.1016/j.jtrangeo.2020.102637>. URL: <http://www.sciencedirect.com/science/article/pii/S0966692319306076>.
- [13] B. Brende Smestad, B. Asbjørnslett, and Ø. Rødseth. “Expanding the Possibilities of AIS Data with Heuristics”. In: *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation* 11 (June 2017), pp. 93–100. DOI: 10.12716/1001.11.02.10.
- [14] X. Chen, Y. Liu, K. Achuthan, and X. Zhang. “A ship movement classification based on Automatic Identification System (AIS) data using Convolutional Neural Network”. In: *Ocean Engineering* 218 (2020), p. 108182. ISSN: 0029-8018. DOI: <https://doi.org/10.1016/j.oceaneng.2020.108182>. URL: <http://www.sciencedirect.com/science/article/pii/S0029801820311124>.
- [15] D. Conway. *The Data Science Venn Diagram*. 2010. URL: <http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram>.

- 
- [16] G. Anadiotis. *Planet analytics: big data, sustainability, and environmental impact*. 2017. URL: <https://www.zdnet.com/article/planet-analytics-big-data-sustainability-and-environmental-impact/>.
- [17] P. Bruce, A. Bruce, and P. Gedeck. *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python*. O'Reilly Media, 2020.
- [18] J. W. Tukey. "The Future of Data Analysis". In: *The Annals of Mathematical Statistics* 33.1 (1962), pp. 1–67. ISSN: 00034851. URL: <http://www.jstor.org/stable/2237638>.
- [19] A. Twin. *Data Mining*. 2020. URL: <https://www.investopedia.com/terms/d/datamining.asp>.
- [20] H. Mannila. "Methods and problems in data mining". In: *Database Theory — ICDT '97*. Ed. by F. Afrati and P. Kolaitis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 41–55. ISBN: 978-3-540-49682-3.
- [21] C. E. IBM. *Machine Learning*. 2020. URL: <https://www.ibm.com/cloud/learn/machine-learning#toc-machine-le-K7Vsz0k6>.
- [22] I. Lee and Y. J. Shin. "Machine learning for enterprises: Applications, algorithm selection, and challenges". In: *Business Horizons* 63.2 (2020), pp. 157–170. ISSN: 0007-6813. DOI: <https://doi.org/10.1016/j.bushor.2019.10.005>. URL: <http://www.sciencedirect.com/science/article/pii/S0007681319301521>.
- [23] S. Kulkarni and I. Haidar. "Forecasting model for crude oil price using artificial neural networks and commodity futures prices". In: *arXiv preprint arXiv:0906.4838* (2009).
- [24] L. Yu, S. Wang, and K. K. Lai. "Forecasting crude oil price with an EMD-based neural network ensemble learning paradigm". In: *Energy Economics* 30.5 (2008), pp. 2623–2635.
- [25] C. Brooks. *Introductory econometrics for finance*. Cambridge university press, 2019.
- [26] T. Shah. "About Train, Validation and Test Sets in Machine Learning". In: (2017). URL: <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>.
- [27] C. Albon. *Machine learning with python cookbook: Practical solutions from preprocessing to deep learning*. " O'Reilly Media, Inc.", 2018.
- [28] B. Duignan. "Occam's razor - Philosophy". In: (2021). URL: <https://www.britannica.com/topic/Occams-razor>.
- [29] R. Shaikh. "Feature Selection Techniques in Machine Learning with Python". In: (2018). URL: <https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e>.
-

- 
- [30] W. Koehrsen. “A Complete Introduction and Walkthrough”. In: (2018). URL: <https://www.kaggle.com/willkoehrsen/a-complete-introduction-and-walkthrough/notebook#Feature-Selection>.
- [31] A. Hayes. “Fundamental Analysis - Multicollinearity”. In: (2021). URL: <https://www.investopedia.com/terms/m/multicollinearity.asp>.
- [32] T. Yiu. “Understanding Random Forest - How the Algorithm Works and Why it Is So Effective”. In: (2019). URL: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>.
- [33] S. Raschka. “Activation Functions for Artificial Neural Networks”. In: (2016). URL: [http://rasbt.github.io/mlxtend/user\\_guide/general\\_concepts/activation-functions/](http://rasbt.github.io/mlxtend/user_guide/general_concepts/activation-functions/).
- [34] J. L. Elman. “Finding structure in time”. In: *Cognitive science* 14.2 (1990), pp. 179–211.
- [35] C. Olah. “Understanding LSTM Networks”. In: (2015). URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.
- [36] I. C. Education. “Recurrent Neural Networks”. In: (2020). URL: <https://www.ibm.com/cloud/learn/recurrent-neural-networks>.
- [37] B. Or. “The Exploding and Vanishing Gradients Problem in Time Series”. In: (2020). URL: <https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>.
- [38] S. Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen”. In: *Diploma, Technische Universität München* 91.1 (1991).
- [39] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [40] J. Jordan. “Hyperparameter tuning for machine learning models”. In: (2017). URL: <https://www.jeremyjordan.me/hyperparameter-tuning/>.
- [41] A. Johnson. “Common Problems in Hyperparameter Optimization”. In: (2017). URL: <https://sigopt.com/blog/common-problems-in-hyperparameter-optimization>.
- [42] F. S. Hillier. *Introduction to operations research*. Tata McGraw-Hill Education, 2012.
- [43] A. N. Skauen, Ø. Hellenen, Ø. Olsen, and R. Olsen. “Operator and user perspective of fractionated AIS satellite systems”. In: *Proceedings of the AIAA/USU Conference on Small Satellites, Around the Corner, SSC13-XI-5*. 2013.

# Appendix A

## AIS Data Contents

Information item	Information generation, type and quality of information
<b>Static Information</b>	
MMSI	Set on installation
Call sign and name	Set on installation
IMO Number	Set on installation
Length and beam	Set on installation or if changed
Type of ship	Select from pre-installed list
Location of position-fixing antenna	Set on installation or may be changed for bi-directional vessels or those fitted with multiple antennae
<b>Dynamic Information</b>	
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
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: <ul style="list-style-type: none"> <li>- underway by engines</li> <li>- at anchor</li> <li>- not under command (NUC)</li> <li>- restricted in ability to manoeuvre (RIATM)</li> <li>- moored</li> <li>- constrained by draught</li> <li>- aground</li> <li>- engaged in fishing</li> <li>- underway by sail</li> </ul> <p>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</p>
Rate of turn (ROT)	Automatically updated from the ship's ROT sensor or derived from the gyro



---

<b>Voyage-related</b>	
Ship's draught	To be manually entered at the start of the voyage using the maximum draft 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**

## Descriptive Statistics of Features

**Table B.1:** Descriptive statistics of count features in port locations

<b>Feature name</b>	<b>Unit</b>	<b>Mean</b>	<b>Std.</b>	<b>Min</b>	<b>Max</b>
tubarao	# vessels	5.826923	2.826493	1	13
qingdao	# vessels	0.115385	0.378534	0	2
bolivar	# vessels	2.192308	1.547147	0	6
dampier	# vessels	3.961538	1.969986	0	9
oita	# vessels	0.057692	0.307645	0	2
haypoint	# vessels	4.038462	2.195910	0	11
baltimore	# vessels	0.500000	0.939336	0	4
narvik	# vessels	1.596154	1.240803	0	4
portcartier	# vessels	1.326923	1.097612	0	4
richardsbay	# vessels	4.038462	2.384275	0	10
gangavaram	# vessels	0.480769	0.851536	0	4
saldanhabay	# vessels	6.480769	2.469344	2	12
goa	# vessels	0.173077	0.473665	0	2
nouadhibou	# vessels	2.153846	1.363166	0	5
rizhao	# vessels	0.096154	0.495454	0	3
dekheila	# vessels	0.096154	0.297678	0	1

**Table B.2:** Descriptive statistics of capacity features in port locations

Feature name	Unit	Mean	Std.	Min	Max
tubarao_cap	dwt	1,076,131	532,401	174,944	2,518,616
qingdao_cap	dwt	20,971	68,815	0	361,120
bolivar_cap	dwt	392,957	278,127	0	1,085,260
dampier_cap	dwt	714,453	359,079	0	1,608,192
oita_cap	dwt	10,822	56,548	0	354,954
haypoint_cap	dwt	732,136	401,746	0	2,029,211
baltimore_cap	dwt	88,847	167,447	0	724,584
narvik_cap	dwt	292,281	228,058	0	772,584
portcartier_cap	dwt	236,616	195,834	0	714,800
richardsbay_cap	dwt	710,667	414,787	0	1,732,110
gangavaram_cap	dwt	85,151	150,964	0	705,473
saldanhabay_cap	dwt	1,181,196	447,457	349,144	2,161,695
goa_cap	dwt	32,269	87,631	0	365,860
nouadhibou_cap	dwt	382,596	241,140	0	896,086
rizhao_cap	dwt	17,396	90,686	0	566,375
dekheila_cap	dwt	16,697	51,707	0	179,067

**Table B.3:** Descriptive statistics of count and capacity features in world regions

Feature name	Unit	Mean	Std.	Min	Max
atlantic	# vessels	221.21	25.24	187	361
FE	# vessels	630.06	40.02	540	784
CP	# vessels	39.63	7.36	27	65
EstP	# vessels	54.54	8.67	39	95
NWE	# vessels	22.15	4.88	13	33
indi	# vessels	191.44	27.72	150	284
Med	# vessels	32.31	6.29	20	55
atlantic_cap	dwt	40,350,807	4,606,793	34,040,551	65,990,019
FE_cap	dwt	116,124,869	7,340,753	99,998,161	144,061,607
CP_cap	dwt	7,116,248	1,330,829	4,783,052	11,708,818
EstP_cap	dwt	9,846,140	1,587,266	7,114,499	17,382,856
NWE_cap	dwt	4,033,180	890,305	2,309,654	6,140,830
indi_cap	dwt	34,904,410	5,005,928	27,374,150	51,795,173
Med_cap	dwt	5,803,266	1,128,724	3,603,926	9,826,379

---

**Table B.4:** Descriptive statistics fleet percentage features in world regions

Feature name	Unit	Mean	Std.	Min	Max
atlantic_pct	%	0.18488	0.01434	0.15216	0.21605
FE_pct	%	0.53273	0.02118	0.47047	0.57737
CP_pct	%	0.03253	0.00497	0.02280	0.04618
EstP_pct	%	0.04506	0.00563	0.03342	0.05765
NWE_pct	%	0.01849	0.00393	0.01004	0.02923
indi_pct	%	0.15968	0.01698	0.12864	0.19498
Med_pct	%	0.02664	0.00493	0.01625	0.03648

**Table B.5:** Descriptive statistics of fleet count and capacity features

Feature name	Unit	Mean	Std.	Min	Max
n_active	# vessels	993.35	38.21	907	1,136
sum_cap	dwt	182,125,989	6,963,553	166,466,264	208,192,051
sum_tot_cap	dwt	218,178,920	15,057,033	192,887,535	306,204,691

**Table B.6:** Descriptive statistics of fleet utilization feature

Feature name	Unit	Mean	Std.	Min	Max
fleet_util	%	0.75757	0.02897	0.69243	0.86600

**Table B.7:** Descriptive statistics of market and price derived features

Feature name	Unit	Mean	Std.	Min	Max
avg_rate	\$/day	16865.42	9829.43	1250.00	35625.00
Bdry	\$	15.27	3.76	9.46	22.11
BCI	-	2243.74	1319.60	160.80	4910.60
BCLC2	\$/tonne	8.15	2.44	4.31	12.82
BCLC3*	\$/tonne	18.56	4.58	11.75	28.79
BCLC5	\$/tonne	7.72	1.97	4.49	11.21
BCLC14	\$/day	17081.90	8274.28	4764.00	35755.00
BCLC17	\$/tonne	13.78	3.53	8.50	20.90



# Appendix C

## Augmented Dickey-Fuller Results

Feature	ADF Statistics	p-value	1%	5%	10%	$H_0$ at 1%	$H_0$ at 5%	$H_0$ at 10%
avg_rate	-4.3878	0.0003	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
Bdry	-6.0744	0.0000	-3.5685	-2.9214	-2.5987	FALSE	FALSE	FALSE
BCI	-4.1101	0.0009	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
BCLC2	-4.0910	0.0010	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
BCLC5	-5.2287	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
BCLC14	-4.5606	0.0002	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
BCLC17	-4.7116	0.0001	-3.5685	-2.9214	-2.5987	FALSE	FALSE	FALSE
tubarao	-3.7082	0.0040	-3.5925	-2.9315	-2.6041	FALSE	FALSE	FALSE
tubarao_cap	-3.4955	0.0081	-3.5925	-2.9315	-2.6041	TRUE	FALSE	FALSE
qingdao	-10.1735	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
qingdao_cap	-4.3952	0.0003	-3.6056	-2.9371	-2.6070	FALSE	FALSE	FALSE
bolivar	-4.3703	0.0003	-3.5848	-2.9283	-2.6023	FALSE	FALSE	FALSE
bolivar_cap	-4.3555	0.0004	-3.5848	-2.9283	-2.6023	FALSE	FALSE	FALSE
dampier	-7.8557	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
dampier_cap	-7.8826	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
oita	-6.6332	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
oita_cap	-6.6332	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
haypoint	-5.9843	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
haypoint_cap	-6.0871	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
baltimore	-4.0599	0.0011	-3.5848	-2.9283	-2.6023	FALSE	FALSE	FALSE
baltimore_cap	-4.0427	0.0012	-3.5848	-2.9283	-2.6023	FALSE	FALSE	FALSE
narvik	-4.1105	0.0009	-3.5925	-2.9315	-2.6041	FALSE	FALSE	FALSE
narvik_cap	-4.1545	0.0008	-3.5925	-2.9315	-2.6041	FALSE	FALSE	FALSE
portcartier	-5.4991	0.0000	-3.5813	-2.9268	-2.6015	FALSE	FALSE	FALSE
portcartier_cap	-5.4984	0.0000	-3.5813	-2.9268	-2.6015	FALSE	FALSE	FALSE
richardsbay	-7.9060	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
richardsbay_cap	-8.0256	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
gangavaram	-6.9387	0.0000	-3.5778	-2.9253	-2.6008	FALSE	FALSE	FALSE
gangavaram_cap	-6.9626	0.0000	-3.5778	-2.9253	-2.6008	FALSE	FALSE	FALSE
saldanhabay	-8.3633	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
saldanhabay_cap	-8.4392	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
goa	-7.2491	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
goa_cap	-7.2316	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
nouadhibou	-6.1865	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
nouadhibou_cap	-6.1707	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
rizhao	-7.6660	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
rizhao_cap	-7.4851	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE

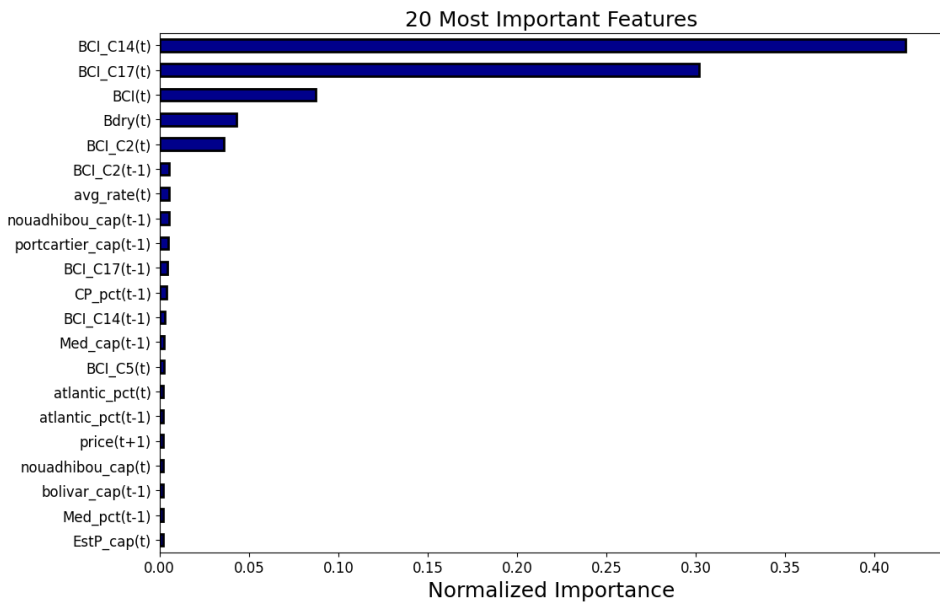
---

Feature	ADF Statistics	p-value	1%	5%	10%	$H_0$ at 1%	$H_0$ at 5%	$H_0$ at 10%
dekheila	-6.6634	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
dekheila_cap	-6.6558	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
atlantic	-2.7514	0.0656	-3.6010	-2.9351	-2.6060	TRUE	TRUE	FALSE
atlantic_cap	-2.8085	0.0571	-3.6010	-2.9351	-2.6060	TRUE	TRUE	FALSE
FE	-9.9724	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
FE_cap	-10.0361	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
CP	-7.4551	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
CP_cap	-7.4230	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
EstP	-5.5200	0.0000	-3.5813	-2.9268	-2.6015	FALSE	FALSE	FALSE
EstP_cap	-5.4311	0.0000	-3.5813	-2.9268	-2.6015	FALSE	FALSE	FALSE
NWE	-5.4499	0.0000	-3.5778	-2.9253	-2.6008	FALSE	FALSE	FALSE
NWE_cap	-5.3289	0.0000	-3.5778	-2.9253	-2.6008	FALSE	FALSE	FALSE
indi	-2.9161	0.0435	-3.5925	-2.9315	-2.6041	TRUE	TRUE	FALSE
indi_cap	-2.9137	0.0438	-3.5925	-2.9315	-2.6041	TRUE	TRUE	FALSE
Med	-5.0555	0.0000	-3.6010	-2.9351	-2.6060	FALSE	FALSE	FALSE
Med_cap	-4.9382	0.0000	-3.6010	-2.9351	-2.6060	FALSE	FALSE	FALSE
Sum_tot_cap	-9.1170	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
sum_cap	-11.0353	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
n_active	-10.9207	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
fleet_util	-11.0353	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
atlantic_pct	-5.9118	0.0000	-3.5746	-2.9240	-2.6000	FALSE	FALSE	FALSE
FE_pct	-5.9252	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
CP_pct	-7.5884	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
EstP_pct	-5.9481	0.0000	-3.5813	-2.9268	-2.6015	FALSE	FALSE	FALSE
NWE_pct	-5.1803	0.0000	-3.5778	-2.9253	-2.6008	FALSE	FALSE	FALSE
indi_pct	-5.4771	0.0000	-3.5715	-2.9226	-2.5993	FALSE	FALSE	FALSE
Med_pct	-4.0392	0.0012	-3.5966	-2.9333	-2.6050	FALSE	FALSE	FALSE
price	-5.1018	0.0000	-3.5685	-2.9214	-2.5987	FALSE	FALSE	FALSE

---

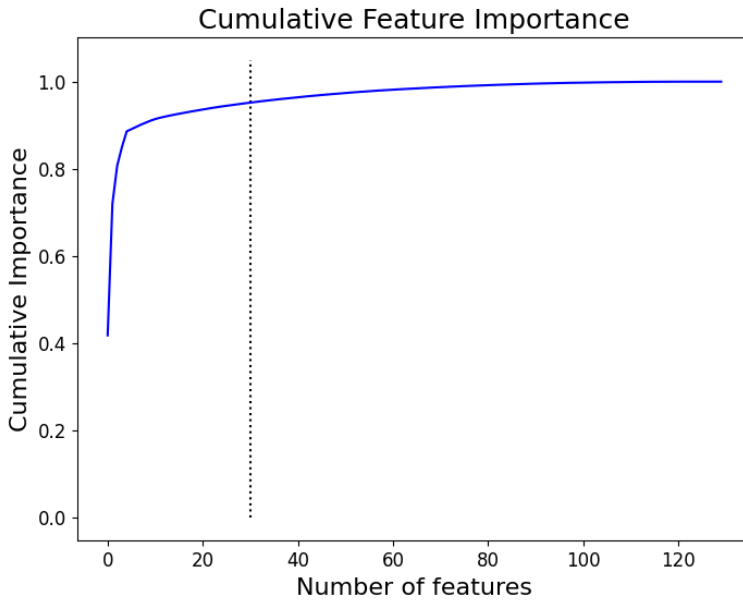
# Appendix D

## Feature Importance Scores

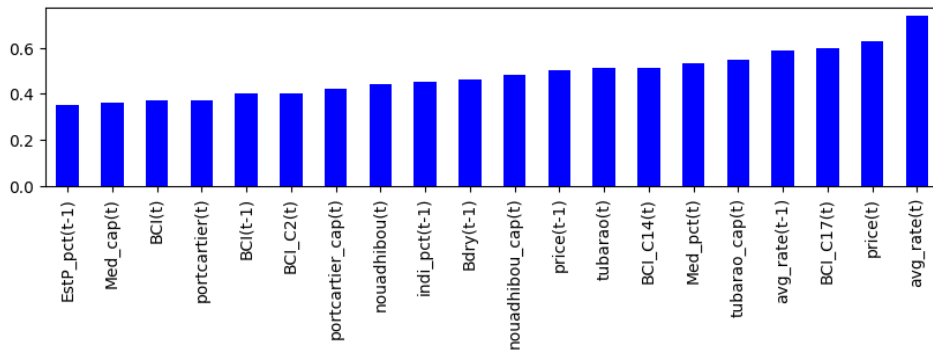


**Figure D.1:** Top 20 most important features based on random forest regressor model performance





**Figure D.2:** Cumulative feature importance score with respect to number of features



**Figure D.3:** Top 20 features based on mean feature importance score

avg_rate(t-1)	0.42	0.66	0.46	0.39	1	0.59
avg_rate(t)	0.62	1	0.45	1	0.61	0.74
Bdry(t-1)	0.68	0.03	0.74	0.03	0.82	0.46
Bdry(t)	0.47	0.34	0.03	0.36	0.22	0.28
BCI(t-1)	1	0.12	0.24	0.04	0.6	0.4
BCI(t)	0.04	0.57	0.1	0.57	0.57	0.37
BCI_C2(t-1)	0.29	0.09	0.15	0	0.46	0.2
BCI_C2(t)	0.18	0.24	0.77	0.26	0.57	0.4
BCI_C5(t-1)	0.49	0.1	0.04	0	0.08	0.14
BCI_C5(t)	0.27	0.39	0.03	0.37	0.2	0.25
BCI_C14(t-1)	0.01	0.17	0.76	0	0.6	0.31
BCI_C14(t)	0.22	0.74	0.16	0.82	0.6	0.51
BCI_C17(t-1)	0.16	0.17	0.12	0	0.45	0.18
BCI_C17(t)	0.29	0.54	1	0.63	0.52	0.6
tubarao(t-1)	0.39	0.07	0.03	0	0.41	0.18
tubarao(t)	0.75	0.93	0.01	0.58	0.27	0.51
tubarao_cap(t-1)	0.14	0.14	0.11	0	0.38	0.15
tubarao_cap(t)	0.77	0.94	0.15	0.57	0.34	0.55
qingdao(t-1)	0.21	0.11	0.01	0.02	0.04	0.08
qingdao(t)	0.04	0	0	0	0.14	0.04
qingdao_cap(t-1)	0.34	0.1	0	0.02	0.04	0.1
qingdao_cap(t)	0.06	0.02	0.03	0	0.14	0.05
bolivar(t-1)	0.12	0.01	0.01	0.01	0.48	0.13
bolivar(t)	0.16	0.12	0	0	0.21	0.1
bolivar_cap(t-1)	0.13	0.02	0.11	0.01	0.59	0.17
bolivar_cap(t)	0.2	0.12	0.07	0	0.24	0.13
dampier(t-1)	0.16	0.1	0.01	0.04	0.34	0.13
dampier(t)	0.21	0.22	0.02	0.06	0.16	0.13
dampier_cap(t-1)	0.27	0.08	0.04	0.04	0.3	0.15
dampier_cap(t)	0.34	0.26	0.08	0.07	0.39	0.23
oita(t-1)	0.11	0.1	0	0.01	0.07	0.06
oita(t)	0.03	0.2	0	0.05	0.21	0.1
oita_cap(t-1)	0.01	0.08	0	0.01	0.07	0.03
oita_cap(t)	0.2	0.18	0	0.04	0.21	0.13
haypoint(t-1)	0.22	0.12	0.05	0.02	0.12	0.11
haypoint(t)	0.12	0.12	0.02	0	0.45	0.14
haypoint_cap(t-1)	0.29	0.12	0.06	0.02	0.14	0.13
haypoint_cap(t)	0.12	0.11	0.14	0	0.52	0.18
baltimore(t-1)	0.2	0.26	0.01	0.04	0.33	0.17
baltimore(t)	0.15	0.38	0	0.15	0.18	0.17
baltimore_cap(t-1)	0.23	0.26	0.02	0.04	0.4	0.19
baltimore_cap(t)	0.1	0.37	0.01	0.14	0.27	0.18
narvik(t-1)	0.4	0.4	0.01	0.13	0.19	0.23
narvik(t)	0.22	0.4	0.03	0.15	0.14	0.19
narvik_cap(t-1)	0.33	0.4	0.05	0.13	0.55	0.29
narvik_cap(t)	0.14	0.36	0.05	0.13	0.21	0.18
portcartier(t-1)	0.32	0.01	0.06	0	0.38	0.15
portcartier(t)	0.32	0.64	0.35	0.17	0.36	0.37
portcartier_cap(t-1)	0.29	0.01	0.27	0	0.45	0.2
portcartier_cap(t)	0.29	0.63	0.72	0.17	0.31	0.42
richardsbay(t-1)	0.26	0.04	0.18	0	0.16	0.13
richardsbay(t)	0.41	0.1	0.02	0.01	0.12	0.13
richardsbay_cap(t-1)	0.21	0.02	0.02	0	0.41	0.13
richardsbay_cap(t)	0.27	0.12	0.08	0.01	0.38	0.17
gangavaram(t-1)	0.17	0.02	0.01	0.04	0.31	0.11
gangavaram(t)	0.2	0.17	0	0	0.06	0.09
gangavaram_cap(t-1)	0.19	0.02	0.02	0.04	0.4	0.13
gangavaram_cap(t)	0.24	0.19	0.08	0	0.21	0.14
saldanhabay(t-1)	0.41	0.16	0.05	0	0.49	0.22
saldanhabay(t)	0.04	0.05	0.02	0	0.22	0.07
saldanhabay_cap(t-1)	0.46	0.13	0.15	0	0.44	0.24
saldanhabay_cap(t)	0	0.06	0.09	0	0.2	0.07
goa(t-1)	0.31	0.03	0.01	0	0.24	0.12
	Linear Reg.	Ridge Reg.	RF	Linear Corr.	MIC	Mean Score

Figure D.4: Feature importance scores of selected algorithms (1/3)

richardsbay(t-1)	0.26	0.04	0.18	0	0.16	0.13
richardsbay(t)	0.41	0.1	0.02	0.01	0.12	0.13
richardsbay_cap(t-1)	0.21	0.02	0.02	0	0.41	0.13
richardsbay_cap(t)	0.27	0.12	0.08	0.01	0.38	0.17
gangavaram(t-1)	0.17	0.02	0.01	0.04	0.31	0.11
gangavaram(t)	0.2	0.17	0	0	0.06	0.09
gangavaram_cap(t-1)	0.19	0.02	0.02	0.04	0.4	0.13
gangavaram_cap(t)	0.24	0.19	0.08	0	0.21	0.14
saldanhabay(t-1)	0.41	0.16	0.05	0	0.49	0.22
saldanhabay(t)	0.04	0.05	0.02	0	0.22	0.07
saldanhabay_cap(t-1)	0.46	0.13	0.15	0	0.44	0.24
saldanhabay_cap(t)	0	0.06	0.09	0	0.2	0.07
goa(t-1)	0.31	0.03	0.01	0	0.24	0.12
goa(t)	0.29	0.2	0.01	0.08	0.01	0.12
goa_cap(t-1)	0.32	0.06	0.02	0.01	0.26	0.13
goa_cap(t)	0.39	0.18	0	0.07	0.21	0.17
nouadhibou(t-1)	0.05	0.21	0.01	0.05	0	0.06
nouadhibou(t)	0.49	0.9	0.01	0.49	0.3	0.44
nouadhibou_cap(t-1)	0.01	0.2	0.13	0.04	0.21	0.12
nouadhibou_cap(t)	0.53	0.89	0.09	0.5	0.38	0.48
nizhao(t-1)	0.17	0.07	0	0	0.21	0.09
nizhao(t)	0.24	0.11	0	0	0.21	0.11
nizhao_cap(t-1)	0.2	0.06	0	0	0.21	0.09
nizhao_cap(t)	0.2	0.11	0	0.01	0.21	0.11
dekheila(t-1)	0.01	0.1	0.01	0.03	0.42	0.11
dekheila(t)	0.29	0.43	0	0.08	0.73	0.31
dekheila_cap(t-1)	0.03	0.08	0.03	0.03	0.42	0.12
dekheila_cap(t)	0.26	0.41	0	0.08	0.73	0.3
atlantic(t-1)	0.19	0.38	0.12	0.16	0.08	0.19
atlantic(t)	0.12	0.04	0.03	0.02	0.56	0.15
atlantic_cap(t-1)	0.18	0.37	0.11	0.15	0.31	0.22
atlantic_cap(t)	0.07	0.05	0.04	0.02	0.55	0.15
FE(t-1)	0.2	0.23	0.03	0.04	0.27	0.15
FE(t)	0	0.19	0.01	0.04	0.14	0.08
FE_cap(t-1)	0.26	0.23	0.08	0.04	0.27	0.18
FE_cap(t)	0.06	0.22	0.04	0.05	0.36	0.15
CP(t-1)	0.07	0.25	0.12	0.16	0.35	0.19
CP(t)	0.37	0.01	0.02	0	0.17	0.11
CP_cap(t-1)	0.09	0.24	0.03	0.15	0.34	0.17
CP_cap(t)	0.23	0.01	0.06	0	0.22	0.1
EstP(t-1)	0.38	0.39	0.07	0.25	0.49	0.32
EstP(t)	0.5	0.18	0.19	0.05	0.18	0.22
EstP_cap(t-1)	0.2	0.35	0.22	0.22	0.49	0.3
EstP_cap(t)	0.4	0.18	0.2	0.05	0.17	0.2
NWE(t-1)	0.01	0.21	0.01	0.02	0.13	0.08
NWE(t)	0.03	0.15	0.03	0.05	0.2	0.09
NWE_cap(t-1)	0.04	0.21	0.08	0.02	0.33	0.14
NWE_cap(t)	0.05	0.16	0.03	0.05	0.34	0.13
indi(t-1)	0	0.17	0.13	0.08	0.08	0.09
indi(t)	0.05	0.04	0.15	0	0.27	0.1
indi_cap(t-1)	0.11	0.15	0.07	0.07	0.21	0.12
indi_cap(t)	0	0.01	0.16	0	0.49	0.13
Med(t-1)	0.04	0.08	0.01	0	0.25	0.08
Med(t)	0.18	0.27	0.22	0.16	0.69	0.3
Med_cap(t-1)	0.04	0.06	0.1	0	0.15	0.07
Med_cap(t)	0.19	0.27	0.51	0.15	0.7	0.36
Sum_tot_cap(t-1)	0.14	0.1	0.07	0	0.36	0.13
Sum_tot_cap(t)	0	0.1	0.07	0.02	0.3	0.1
sum_cap(t-1)	0.09	0.12	0.06	0.01	0.44	0.14
sum_cap(t)	0.11	0.2	0.06	0.06	0.3	0.15
n_active(t-1)	0.23	0.1	0.01	0.01	0.35	0.14
n_active(t)	0.06	0.18	0.03	0.05	0.29	0.12
fleet_util(t-1)	0.09	0.12	0.04	0.01	0.44	0.14
	Linear Reg.	Ridge Reg.	RF	Linear Corr.	MIC	Mean Score

Figure D.5: Feature importance scores of selected algorithms (2/3)

indi_cap(t-1)	0.11	0.15	0.07	0.07	0.21	0.12
indi_cap(t)	0	0.01	0.16	0	0.49	0.13
Med(t-1)	0.04	0.08	0.01	0	0.25	0.08
Med(t)	0.18	0.27	0.22	0.16	0.69	0.3
Med_cap(t-1)	0.04	0.06	0.1	0	0.15	0.07
Med_cap(t)	0.19	0.27	0.51	0.15	0.7	0.36
Sum_tot_cap(t-1)	0.14	0.1	0.07	0	0.36	0.13
Sum_tot_cap(t)	0	0.1	0.07	0.02	0.3	0.1
sum_cap(t-1)	0.09	0.12	0.06	0.01	0.44	0.14
sum_cap(t)	0.11	0.2	0.06	0.06	0.3	0.15
n_active(t-1)	0.23	0.1	0.01	0.01	0.35	0.14
n_active(t)	0.06	0.18	0.03	0.05	0.29	0.12
fleet_util(t-1)	0.09	0.12	0.04	0.01	0.44	0.14
fleet_util(t)	0.11	0.2	0.01	0.06	0.3	0.14
atlantic_pct(t-1)	0.15	0.6	0.2	0.23	0.29	0.29
atlantic_pct(t)	0.05	0.08	0.06	0	0.72	0.18
FE_pct(t-1)	0.26	0.37	0.1	0.14	0.42	0.26
FE_pct(t)	0.05	0.24	0.18	0.06	0.11	0.13
CP_pct(t-1)	0.11	0.47	0.09	0.25	0.55	0.29
CP_pct(t)	0.32	0.09	0.11	0.01	0.16	0.14
EstP_pct(t-1)	0.48	0.72	0.04	0.47	0.04	0.35
EstP_pct(t)	0.61	0.24	0.11	0.04	0.19	0.24
NWE_pct(t-1)	0.18	0.3	0.09	0.03	0.41	0.2
NWE_pct(t)	0.02	0.21	0.03	0.06	0.47	0.16
indi_pct(t-1)	0.2	0.46	0.54	0.17	0.88	0.45
indi_pct(t)	0.11	0.13	0.08	0.02	0.18	0.1
Med_pct(t-1)	0.09	0	0.07	0	0.35	0.1
Med_pct(t)	0.38	0.43	0.71	0.15	0.97	0.53
price(t-1)	0.94	0.46	0.46	0.01	0.61	0.5
price(t)	0.82	0.83	0.15	0.85	0.51	0.63
	Linear Reg.	Ridge Reg.	RF	Linear Corr.	MIC	Mean Score

Figure D.6: Feature importance scores of selected algorithms (3/3)



# Appendix E

## Python Scripts

### E.1 Description of Python scripts

Name	Description
<i>project_settings.py</i>	Generates global variables for file accessibility
<i>ais_sql.py</i>	Draft script for extraction of desired AIS data from SQL databases
<i>data_processing.py</i>	Pre-processes data retrieved from the AIS database to avoid any excess or unwanted data. Also splits the processed data into smaller parts for better implementation.
<i>clarksons.py</i>	Pre-processes and generate plots for data retrieved from the Clarksons Platou database
<i>polygons.py</i>	Creation, plotting and saving of world and port polygons, used for further geospatial analysis of AIS data
<i>FE.py</i>	Script for feature engineering and construction process
<i>data_preparation.py</i>	Data preparation of time series data to enable supervised learning problems
<i>feature_importance_score.py</i>	Rank the features in accordance to the filter selection methods
<i>feature_selection.py</i>	Determine the best feature set combination for an optimized decision tree model
<i>ML_models.py</i>	Serves as the main file, runs and evaluates the selected machine learning models

---

## E.2 project\_settings.py

```
import os

#AIS Data
#AIS_RAW_DATA_PATH = r'oscarsingapore1.csv'
AIS_RAW_DATA_PATH = r'AIS_Data\AIS_2019_raw_processed.csv'
#AIS_RAW_DATA_PATH = r'cotankers2019.csv'
SAMPLE_SET_PATH = r'AIS_Data\sample_ais.csv'
AIS_PROCESSED_DATA_PATH = r'AIS_Data\AIS_2019_processed_02.csv'
AIS_PROCESSED_DATA_VEL_PATH = r'AIS_Data\AIS_2019_processed_vel.csv'
AIS_PROCESSED_DATA_SPLIT1 = r'AIS_Data\AIS_2019_processed_split1.csv'
AIS_PROCESSED_DATA_SPLIT2 = r'AIS_Data\AIS_2019_processed_split2.csv'
AIS_PROCESSED_DATA_SPLIT3 = r'AIS_Data\AIS_2019_processed_split3.csv'
AIS_PROCESSED_DATA_SPLIT4 = r'AIS_Data\AIS_2019_processed_split4.csv'

#Feature Sets
FEATURES = r'AIS_Data\Features.csv'
FEATURE_SPLIT_1 = r'AIS_Data\FeatureSetSplit_v01.csv'
FEATURE_SPLIT_2 = r'AIS_Data\FeatureSetSplit_v02.csv'
FEATURE_SPLIT_3 = r'AIS_Data\FeatureSetSplit_v03.csv'
FEATURE_SPLIT_4 = r'AIS_Data\FeatureSetSplit_v04.csv'
FINAL_FEATURE_SET = r'AIS_Data\FinalFeatureSet.csv'
FINAL_FEATURE_SET_2 = r'AIS_Data\FinalFeatureSet_2.csv'
FEATURE_SPLIT_V1 = r'AIS_Data\FeatureSetSplit_v011.csv'
FEATURE_SPLIT_V2 = r'AIS_Data\FeatureSetSplit_v012.csv'
FEATURE_SPLIT_V3 = r'AIS_Data\FeatureSetSplit_v013.csv'
FEATURE_SPLIT_V4 = r'AIS_Data\FeatureSetSplit_v014.csv'

#Clarksons Data
#Price Data
CLARKSONS_RAW_DATA_PATH = r'ClarksonsData\Capesize_rates_weekly_avg_combined.csv'
CLARKSONS_MERGED_PATH = r''
CLARKSON_PATH = r'AIS_Data\FeatureSet_01.csv'
CLARKSONS_RATES_PATH = r'ClarksonsData\Capesize_rates_only.csv'
CLARKSON_TEST_PATH = r'AIS_Data\FeatureTest01.csv'
#Vessel Databases
CLARKSONS_CAPE_SIZE_DATABASE_PATH = r'CapesizeDatabase\Capesize_database.csv'
CLARKSONS_CAPE_SIZE_PATH = r'CapesizeDatabase\Capesize.csv'
CLARKSONS_CAPE_SIZE_PROCESSED_PATH = r'CapesizeDatabase\Capesize_processed.csv'
CLARKSONS_VLOC_PATH = r'CapesizeDatabase\VLOC.csv'
CLARKSONS_ULOC_PATH = r'CapesizeDatabase\ULOC.csv'
#Vessel Routes
CAPE_SIZE_ROUTES_PATH = r'C:\Users\Bruker\Oscar\NTNU\Div. Programming\Python\paths.csv'
```

---

## E.3 AIS SQL Script

```
import sqlite3
import time
import pandas as pd

def ExtractData(file,a,b,c,d,start_time,end_time,t1,t5,n1,n5, st=70):
    '''
    Extracts data from the AIS database:
    file: Filepath to AIS data database
    a,b,c,d: Corners of the area of interest (The entire globe)
    start_time, end_time: Time window of interest (01.01.2019 - current date)
    t1: 1 if MessageType1 (Position Report) will be extracted, 0 otherwise
    t5: 1 if MessageType5 (Static and voyage related data) will be extracted, 0 otherwise
    n1: Name of table with MessageType1 data in "file"
    n5: Name of table with MessageType5 data in "file"
    st: Ship Type to extract data of (=70 -> Cargo-all ships of this type)
    returns: DataFrames with Message Types 1 and 5 data
    '''
    plotlat = list()
    plotlon = list()
    timestep = list()
    mmsi = list()
    navstat = list()
    voyagetime = list()
    draught = list()
    useridVoyage = list()
    beam = list()
    loa = list()
    shiptype = list()

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

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

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

        #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])
            plotlat.append(Datastrip[1])
            plotlon.append(Datastrip[2])
            mmsi.append(Datastrip[3])
```



---

```

        navstat.append(Datastrip[4])

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

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

else:
    df = False

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

    SQLstring5 = "SELECT unixtime, userid, draught, beam, loa, st from %s where \
        unixtime >= %s and unixtime <= %s and st == %s ORDER BY unixtime ASC" % (n5,
        str(start_time), str(end_time), str(st))

    #Extract draught, beam and loa from database:
    A = time.time()
    with conn:
        cur = conn.cursor()
        cur.execute(SQLstring5)
        VoyageData = cur.fetchall()

        for i in range(0, len(VoyageData)):
            VoyageStrip = VoyageData[i]
            voyagetime.append(VoyageStrip[0])
            useridVoyage.append(VoyageStrip[1])
            draught.append(VoyageStrip[2])
            beam.append(VoyageStrip[3])
            loa.append(VoyageStrip[4])
            shiptype.append(VoyageStrip[5])

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

    dfVoyage = pd.DataFrame({'Unixtime' : voyagetime, 'MMSI' : useridVoyage,
                            'Draught' : draught, 'Beam' : beam, 'LoA' : loa,
                            'Ship Type' : shiptype})

else:
    dfVoyage = False

return df, dfVoyage

```

---

## E.4 data\_processing.py

```
from tqdm import tqdm
from configs.project_settings import AIS_PROCESSED_DATA_SPLIT1, AIS_PROCESSED_DATA_SPLIT2
from configs.project_settings import AIS_PROCESSED_DATA_SPLIT3, AIS_PROCESSED_DATA_SPLIT4
from configs.project_settings import AIS_PROCESSED_DATA_VEL_PATH

def ais_pre_processing():
    with open(r"CapesizeDatabase\Capesize.csv") as f:
        MMSI_vals = [line.split(",")[3] for line in f]

    with open(r"AIS_Data\AIS_2019_raw_processed.csv") as f:
        with open(r"AIS_Data\AIS_2019_processed_02.csv", "w+") as out_f:
            for i, line in enumerate(tqdm(f)):
                if line.split(",")[0] in MMSI_vals:
                    out_f.write(line)
            pass

#####

def capesize_db_processing():
    with open(AIS_PROCESSED_DATA_PATH) as f:
        MMSI_vals = [line.split(",")[0] for line in f]

    with open(r"CapesizeDatabase\Capesize.csv") as f:
        with open(r"CapesizeDatabase\Capesize_processed.csv", "w+") as out_f:
            for i, line in enumerate(tqdm(f)):
                if line.split(",")[3] in MMSI_vals:
                    out_f.write(line)
            pass

#####

def ais_splitting_set():
    with open(r"AIS_Data\AIS_2019_processed_02.csv") as f:
        with open(AIS_PROCESSED_DATA_SPLIT1, "w+") as f_out1:
            with open(AIS_PROCESSED_DATA_SPLIT2, "w+") as f_out2:
                with open(AIS_PROCESSED_DATA_SPLIT3, "w+") as f_out3:
                    with open(AIS_PROCESSED_DATA_SPLIT4, "w+") as f_out4:
                        for i, line in enumerate(tqdm(f)):
                            if i == 0:
                                f_out1.write(line)
                                f_out2.write(line)
                                f_out3.write(line)
                                f_out4.write(line)
                            else:
                                mmsi = int(line.split(",")[0])
                                if mmsi <= 353788000:
                                    f_out1.write(line)
                                elif mmsi <= 431342000:
                                    f_out2.write(line)
                                elif mmsi <= 538090370:
                                    f_out3.write(line)
                                else:
                                    f_out4.write(line)
            pass
```

---

```

#####

def velocity_processing():
    with open(r"AIS_Data\AIS_2019_processed_02.csv") as f:
        with open(AIS_PROCESSED_DATA_VEL_PATH, "w+") as f_out:
            for i, line in enumerate(tqdm(f)):
                if i == 0:
                    f_out.write(line)
                else:
                    speed = float(line.split(",")[6])
                    if speed < 25:
                        f_out.write(line)
            pass

#####

# Alternative Approach for AIS Data Processing #
#----- Extremely more Time Consuming -----#

import pandas as pd
import time
from configs.project_settings import SAMPLE_SET_PATH, AIS_PROCESSED_DATA_PATH
import clarksons as cl

def raw_ais_data(sample=False, processed=False):
    """
    Extracts:
        AIS data from .csv-file
    Returns:
        AIS DataFrame
    """
    if sample == True:
        ais_df_chunk = pd.read_csv(SAMPLE_SET_PATH)#, chunksize=100)
    elif processed == True:
        ais_df_chunk = pd.read_csv(AIS_PROCESSED_DATA_PATH)#, chunksize=1000000)

    return ais_df_chunk

#####

def excess_vessels(db, df):
    """
    Extracts:
        Array of all MMSI values registered in the Clarksons Capesize database
        Array of all MMSI values registered in the raw AIS database
    Returns:
        Set of all excess vessels extracted from the AIS database,
        that does not appear in the Clarksons Capesize database
    """
    vessels_db_array = db[["mmsi"]].to_numpy().astype(int)
    vessels_df_array = df[["mmsi"]].to_numpy().astype(int)
    excess_vessels_set = set()

    for mmsi in vessels_df_array:
        if mmsi[0] not in vessels_db_array:
            excess_vessels_set.add(mmsi[0])

```

---

---

```

    return excess_vessels_set

#####

def excess_vessels_db(db, df):
    """
    Extracts:
        Array of all MMSI values registered in the Clarksons Capesize database
        Array of all MMSI values registered in the raw AIS database
    Returns:
        Set of all excess vessels extracted from the AIS database,
        that does not appear in the Clarksons Capesize database
    """
    vessels_db_array = db[["mmsi"]].to_numpy().astype(int)
    vessels_df_array = df[["mmsi"]].to_numpy().astype(int)
    excess_vessels_db_set = set()

    for mmsi in vessels_db_array:
        if mmsi[0] not in vessels_df_array:
            excess_vessels_db_set.add(mmsi[0])

    return excess_vessels_db_set

#####

def raw_ais_data_process(db, df):
    """
    Utilizes the Set of excess vessels in the AIS database,
    to remove all data for these vessels, and creates a new
    and processed AIS DataFrame for only Capesize vessels
    Returns:
        Processed AIS DataFrame with recordings from only
        Capesize vessels listed in the Capesize database
    """
    excess_vessels_set = excess_vessels(db, df)
    for index, row in df.iterrows():
        if row['mmsi'] in excess_vessels_set:
            df.drop(index, inplace=True)
        if is_integer_num(index / 50000):
            print(index)

    return df

#####

def ais_chunk_processing(db, df):
    chunk_list = [] #Append each chunk DataFrame here

    for chunk in df:
        #Processing the raw AIS data
        df_chunk_filter = raw_ais_data_process(db, chunk)

        #Once the data processing is done, append the chunk to list
        chunk_list.append(df_chunk_filter)

    #Concat the list into DataFrame

```

---

```

df = pd.concat(chunk_list)
df.to_csv(r'AIS_Data\AIS_2019_processed_01.csv', index=False)

#####

def is_integer_num(n):
    if isinstance(n, int):
        return True
    if isinstance(n, float):
        return n.is_integer()
    return False

#####

if __name__ == "__main__":
    start = time.time()
    start_readable = time.ctime(start)
    print(f"\nInitialized run at: {start_readable}\n")

    #---- Extract Excess Vessels in Raw AIS Data ----#
    #ais_df = raw_ais_data(processed=True)
    #spot_df, capesize_db = cl.clarksons_data()
    #excess_vessels_set = excess_vessels(capesize_db, ais_df)
    #excess_vessels_db_set = excess_vessels_db(capesize_db, ais_df)
    #print(len(excess_vessels_set))
    #print(len(excess_vessels_db_set))
    #print(excess_vessels_db_set)

    #---- Remove Excess Vessel Recordings in Raw AIS Data ----#
    #ais_df_chunk = raw_ais_data(processed=True)
    #spot_df, capesize_db = cl.clarksons_data()
    #ais_chunk_processing(capesize_db, ais_df_chunk)

    #ais_splitting_set()
    #df1 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT1)
    #print(df1)
    #df2 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT2)
    #print(df2)
    #df3 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT3)
    #print(df3)
    #df4 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT4)
    #print(df4)

    #capesize_db_processing()
    #ais_pre_processing()

    velocity_processing()

    end = time.time()
    end_readable = time.ctime(end)
    print(f"\nCode ended at: {end_readable}")
    print(f"Runtime of the program is {end - start}")

```

---

## E.5 clarksons.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from pandas import Grouper, DataFrame
from matplotlib.dates import DateFormatter
import time

#plt.style.use('seaborn-whitegrid')

#Importing datapaths
#Price Data:
from configs.project_settings import CLARKSONS_RAW_DATA_PATH, CLARKSONS_MERGED_PATH
#Vessel Databases:
from configs.project_settings import CLARKSONS_CAPE SIZE_DATABASE_PATH, CLARKSONS_CAPE SIZE_PATH
from configs.project_settings import CLARKSONS_VLOC_PATH, CLARKSONS_ULOC_PATH
from configs.project_settings import CLARKSONS_CAPE SIZE_PROCESSED_PATH

#####

def clarksons_data():
    spot_rates_df = pd.read_csv(CLARKSONS_RAW_DATA_PATH, parse_dates=True, squeeze=True)
    capesize_df = pd.read_csv(CLARKSONS_CAPE SIZE_PROCESSED_PATH)
    #capesize_df = pd.read_csv(CLARKSONS_CAPE SIZE_DATABASE_PATH)

    spot_rates_df.rename(columns={'172 dwt' : 'dwt_172', '180 dwt' : 'dwt_180'}, inplace=True)
    spot_rates_df.drop(columns=['dwt_180','Eco'], inplace=True)
    spot_rates_df.dropna(inplace=True)

    return spot_rates_df, capesize_df

#####

def spot_rate_figures(df):
    print(df)

    fig, ax1 = plt.subplots()

    ax1.set_xlabel('Date')
    ax1.set_ylabel('Rates ($/day)', color='b')
    ax1.plot(df['Date'], df['dwt_172'], color='b')
    ax1.tick_params(axis='y', labelcolor='b')

    ax2 = ax1.twinx()

    ax2.set_ylabel('Rates ($/ton)', color='r')
    ax2.plot(df['Date'], df['Tubarao/Rotterdam_160'], color='r')
    ax2.tick_params(axis='y', labelcolor='r')

    plt.xticks(np.arange(36, len(df.index), step=106), rotation=45)

    fig.tight_layout()
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Python\rate_comparison1.png',
                bbox_inches='tight',dpi=100)
```

---

```

#####

def capesize_database_processing(capesize, vloc, uloc):
    '''
    Converts the original database provided by Clarkson into separate Capesize, VLOC and
    ULOC Databases
        capesize = [0 or 1], if 1 -> Saves new Capesize.csv-database file
        vloc = [0 or 1], if 1 -> Saves new vloc.csv-database file
        uloc = [0 or 1], if 1 -> Saves new uloc.csv-database file
    '''
    #Read original Clarkson Bulk Vessel Database with dwt of 200,000+, sort in descending order
    vessel_df = pd.read_csv(CLARKSONS_CAPEXSIZE_DATABASE_PATH)
    vessel_df.sort_values(by='Dwt', ascending=False, ignore_index=True, inplace=True)

    if capesize == 1:
        capesize_df = DataFrame()
        for i in range(len(vessel_df.index)):
            if (vessel_df.loc[i,'Dwt'] < 200000 and vessel_df.loc[i, 'Type'] != 'Slurry Carrier')
            or (vessel_df.loc[i, 'Dwt'] > 200000 and vessel_df.loc[i, 'Type'] == 'Bulk Carrier'):
                capesize_df = capesize_df.append(vessel_df.loc[i,:],
                                                ignore_index=True)[vessel_df.columns.tolist()]
        capesize_df.to_csv(CLARKSONS_CAPEXSIZE_PATH, index=False)

    if vloc == 1:
        vloc_df = DataFrame()
        for i in range(len(vessel_df.index)):
            if vessel_df.loc[i,'Dwt'] >= 200000 and vessel_df.loc[i,'Dwt'] < 300000
            and vessel_df.loc[i, 'Type'] != 'Bulk Carrier':
                vloc_df = vloc_df.append(vessel_df.loc[i,:],
                                        ignore_index=True)[vessel_df.columns.tolist()]
        vloc_df.to_csv(CLARKSONS_VLOC_PATH, index=False)

    if uloc == 1:
        uloc_df = DataFrame()
        for i in range(len(vessel_df.index)):
            if vessel_df.loc[i,'Dwt'] >= 300000:
                uloc_df = uloc_df.append(vessel_df.loc[i,:],
                                        ignore_index=True)[vessel_df.columns.tolist()]
        uloc_df.to_csv(CLARKSONS_ULOC_PATH, index=False)

#####

def mmsi_list(vessel_database):
    '''
    Converts row of mmsi values in the respective vessel database into a list
        vessel_database = 1 -> Capesize
        vessel_database = 2 -> VLOC
        vessel_database = 3 -> ULOC
    returns: list of all mmsi values in desired vessel type database
    '''
    if vessel_database == 1:
        df = pd.read_csv(CLARKSONS_CAPEXSIZE_PATH)
    elif vessel_database == 2:
        df = pd.read_csv(CLARKSONS_VLOC_PATH)
    elif vessel_database == 3:
        df = pd.read_csv(CLARKSONS_ULOC_PATH)

```

---

```

mmsi_list = df['MMSI'].to_numpy().astype(int)

#if 477947300 in mmsi_list:
#    in_list = True

return mmsi_list

#####

def tot_fleet_capacity(vessel_database):
    '''
    Calculates the total fleet capacity (DWT) in the respective vessel database
    vessel_database = 1 -> Capesize
    vessel_database = 2 -> VLOC
    vessel_database = 3 -> ULOC
    returns: total fleet capacity, measured in DWT
    '''
    if vessel_database == 1:
        df = pd.read_csv(CLARKSONS_CAPE_SIZE_PATH)
    elif vessel_database == 2:
        df = pd.read_csv(CLARKSONS_VLOC_PATH)
    elif vessel_database == 3:
        df = pd.read_csv(CLARKSONS_ULOC_PATH)

    dwt = df['Dwt'].to_numpy().astype(int)
    dwt = dwt.sum()

    return dwt

#####

def percentage_change(df):
    df['dwt_172_percentchange'] = df['dwt_172'].pct_change()
    df['Tubarao/Qingdao_160/170mt_percentchange'] = df['Tubarao/Qingdao_160/170mt'].pct_change()

    plt.scatter(df['dwt_172_percentchange'], df['Tubarao/Qingdao_160/170mt_percentchange'])
    plt.show()

    fig, ax1 = plt.subplots()

    color = '#1e4f9c'
    ax1.set_xlabel('Date')
    ax1.set_ylabel('Rates ($/day)', color=color)
    ax1.plot(df['Date'], df['dwt_172_percentchange'], color=color)
    ax1.tick_params(axis='y', labelcolor=color)

    ax2 = ax1.twinx()

    color = ('#87b1dd')
    ax2.set_ylabel('Rates ($/ton)', color=color)
    ax2.plot(df['Date'], df['Tubarao/Qingdao_160/170mt_percentchange'], color=color)
    ax2.tick_params(axis='y', labelcolor=color)

    plt.xticks(np.arange(36, len(df.index), step=106), rotation=45)

    fig.tight_layout()
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Python\rate_comparison_pct.png')

```

---



---

```
        bbox_inches='tight',dpi=100)

correlation = df['dwt_172_percentchange'].corr(df['Tubarao/Qingdao_160/170mt_percentchange'])
print(f"Correlation is: {correlation} ")

#####

#####

if __name__ == "__main__":
    start = time.time()
    spot_rates_df, capesize_df = clarksons_data()
    spot_rate_figures(spot_rates_df)
    #capesize_database_processing(0,1,0)
    #mmsi = mmsi_list(1)
    #dwt = tot_fleet_capacity(3)
    #print(dwt)
    #percentage_change(spot_rates_df)
    end = time.time()
    print(f"Runtime of the program is {end - start}")
```

---

## E.6 polygons.py

```
import pandas as pd
import time
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
from mpl_toolkits.axes_grid1.inset_locator import zoomed_inset_axes
from mpl_toolkits.axes_grid1.inset_locator import mark_inset
from geojson import Polygon, Feature, FeatureCollection, dump
plt.style.use('seaborn-dark-palette')

from configs.project_settings import CAPE_SIZE_ROUTES_PATH

#####

def generate_polygons():
    """
    Generate: ocean polygons
    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]]
    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
    with: polygons = list of polygons
    """
    fig, ax = plt.subplots(figsize=(15, 15))
    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(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Python\ocean_polygons.png',
```

---

```

        bbox_inches='tight',dpi=100)

#####

def ocean_polygons_geojson(ocean_polygons = generate_polygons()):
    '''
    Writes the ocean polygon coordinates generated by generate_polygons() to .geojson-files for
    geofence evaluation in ais.py-script
    '''
    string_ocean_polygons = ['atlantic','FE','CP','EstP','NWE','indi','Med']
    n = len(ocean_polygons)
    for i in range(n):
        polygon = Polygon([ocean_polygons[i]])
        features = []
        features.append(Feature(geometry=polygon))

        feature_collection = FeatureCollection(features)

        with open(r'OceanPolygons\polygon_' + string_ocean_polygons[i] + r'.geojson', 'w') as f:
            dump(feature_collection, f)

#####

def generate_port_polygons():
    '''
    Generate: port polygons
    Returns: List with port polygons
    '''
    port_polygons = list()

    #Get coordinates from: https://www.gps-coordinates.net/
    #Get port locations from: https://classic.searoutes.com/routing
    port_singapore = [[103.5, 1.15], [103.62, 1.27], [103.64, 1.33], [104, 1.33],
        [104, 1.22], [103.78, 1.16], [103.5, 1.15]]
    port_tubarao = [[-40, -20], [-40, -20.5], [-40.25, -20.75], [-40.5, -20.5],
        [-40.35, -20], [-40, -20]]
    port_qingdao = [[120.16159057617186,36.0624217151089],[120.1348114013672,35.86790829906797],
        [120.46440124511719,35.8562222630588],[120.56533813476562,36.029110596631874],
        [120.31677246093749,36.1245647481333],[120.16159057617186,36.0624217151089]]
    port_rotterdam = [[4.178581237792968,51.93145921471351],[4.198322296142578,51.959605695633044],
        [4.08966064453125,52.057557132353644],[3.8520812988281246,51.98065110068555],
        [3.9427185058593746,51.88412064173134],[4.178581237792968,51.93145921471351]]
    port_bolivar = [[-71.97263717651367,12.244230289476999],[-71.95272445678711,12.20245561901314],
        [-71.90071105957031,12.268889343198195],[-71.9384765625,12.319709903080708],
        [-72.02808380126953,12.299248680984558],[-72.04113006591797,12.23148048741301],
        [-71.97263717651367,12.244230289476999]]
    port_dampier = [[116.74999237060547,-20.620493976900747],[116.70810699462889,-20.619529991918],
        [116.72218322753906,-20.639933037853005],[116.74398422241211,-20.647563395349],
        [116.76758766174316,-20.63206132059543],[116.74999237060547,-20.620493976900]]
    port_oita = [[131.66616439819336,33.256489214763576],[131.6757774353027,33.259575428753976],
        [131.66925430297852,33.27464607011705],[131.67963981628418,33.287920432606434],
        [131.6572380065918,33.29466447030428],[131.63869857788086,33.27651177811422],
        [131.64170265197754,33.26746989857436],[131.66616439819336,33.256489214763576]]
    port_haypoint = [[149.29218292236328,-21.279617267606948],[149.3009376525879,-21.281376790374],
        [149.3199920654297,-21.25962121266924],[149.3048858642578,-21.249702161806933],
        [149.2818832397461,-21.27561827396124],[149.2839431762695,-21.28153674594627],
        [149.29218292236328,-21.279617267606948]]

```

---

```

port_baltimore = [[-76.54895782470703,39.20751693226987],[-76.5256118774414,39.21576330385492],
[-76.53350830078125,39.235178353475774],[-76.5582275390625,39.22985969593234],
[-76.56784057617188,39.214167307512355],[-76.54895782470703,39.20751693226987]]
port_gwangyang = [[127.69100189208984,34.92675642136852],[127.63744354248045,34.8898735965285],
[127.63160705566405,34.86480634950137],[127.67074584960938,34.82986845427644],
[127.69546508789061,34.85691844395436],[127.70816802978516,34.91437014491459],
[127.69100189208984,34.92675642136852]]
port_narvik = [[17.430496215820312,68.42835737688752],[17.416076660156246,68.43529882733436],
[17.385005950927734,68.43126042407293],[17.356853485107422,68.42311832857942],
[17.413673400878906,68.40467734676693],[17.437705993652344,68.41617314451344],
[17.430496215820312,68.42835737688752]]
port_portcartier = [[-66.79078102111816,50.03233498004549],[-66.79987907409668,50.02081063581],
[-66.77833557128906,50.01799805556886],[-66.76872253417969,50.026655864097],
[-66.77704811096191,50.034760805448535],[-66.79078102111816,50.03233498004]]
port_richardsbay = [[32.02239990234375,-28.78782076284914],[32.00386047363281,-28.8049701491075],
[32.05089569091797,-28.838057477044515],[32.093467712402344,-28.8924782762],
[32.14977264404297,-28.821815920748026],[32.069435119628906,-28.77217324806],
[32.02239990234375,-28.78782076284914]]
port_gangavaram = [[83.2265853881836,17.626353863611257],[83.20117950439453,17.5602465032949],
[83.2993698120117,17.55991917900598],[83.31893920898438,17.62798987749787],
[83.25748443603516,17.647948051340578],[83.2265853881836,17.626353863611257]]
port_saldanhabay = [[18.051910400390625,-33.04723451447377],[18.002471923828125,-32.9867798933],
[17.896041870117188,-32.99081149087487],[17.765579223632812,-33.0483856287],
[17.86376953125,-33.20134964006387],[17.99560546875,-33.14215083110535],
[18.051910400390625,-33.04723451447377]]
port_banjarmasin = [[114.54826354980469,-3.279686393323821],[114.42466735839844,-3.43117485722],
[114.33540344238281,-3.5798980917712004],[114.444580078125,-3.671039169907],
[114.53453063964844,-3.5408348394316587],[114.57229614257812,-3.27625878298],
[114.54826354980469,-3.279686393323821]]
port_goa = [[73.80992889404297,15.397417842677568],[73.82125854492188,15.407347605586226],
[73.83258819580078,15.445076377324979],[73.72993469238281,15.459967425155801],
[73.71791839599608,15.380867184736008],[73.80992889404297,15.397417842677568]]
port_nouadhibou = [[-17.05078125,20.78693059257028],[-16.772003173828125,20.493918871618803],
[-16.65802001953125,20.704738720055513],[-16.917572021484375,21.208739012],
[-17.04254150390625,21.053744493156348],[-17.05078125,20.78693059257028]]
port_rizhao = [[119.54154968261717,35.39240857605964],[119.454345703125,35.25683378961826],
[119.59648132324219,35.185032937998294],[119.69192504882812,35.294391713301195],
[119.65072631835936,35.41031879581839],[119.54154968261717,35.39240857605964]]
port_dekheila = [[29.7894287109375,31.134370760178317],[29.814147949218746,31.130844260159883],
[29.855690002441406,31.15993396385525],[29.826164245605465,31.19195169537777],
[29.733467102050785,31.139954117071454],[29.752693176269528,31.10556717407662],
[29.7894287109375,31.134370760178317]]

port_polygons.append(port_singapore)
port_polygons.append(port_tubarao)
port_polygons.append(port_qingdao)
port_polygons.append(port_rotterdam)
port_polygons.append(port_bolivar)
port_polygons.append(port_dampier)
port_polygons.append(port_oita)
port_polygons.append(port_haypoint)
port_polygons.append(port_baltimore)
port_polygons.append(port_gwangyang)
port_polygons.append(port_narvik)
port_polygons.append(port_portcartier)
port_polygons.append(port_richardsbay)
port_polygons.append(port_gangavaram)

```

---

---

```

port_polygons.append(port_saldanhabay)
port_polygons.append(port_banjarmasin)
port_polygons.append(port_goa)
port_polygons.append(port_nouadhibou)
port_polygons.append(port_rizhao)
port_polygons.append(port_dekheila)

return port_polygons

#####

def port_polygons_plot(port_polygons = generate_port_polygons()):
    '''
    Plots the port polygons
    with: port_polygons = list of port polygons
    '''
    zoom = 1

    fig, ax = plt.subplots(figsize=(15,15))
    m = Basemap(projection='cyl', lon_0=0, resolution='l')
    m.drawmapboundary(fill_color='white')
    m.fillcontinents(color='lightgrey', lake_color='white')

    for port_polygon in port_polygons:
        x, y = zip(*port_polygon)
        m.plot(x,y,markersize=0)
        ax.fill(x,y,alpha=0.2)

    if zoom == 1:
        axins = zoomed_inset_axes(ax, 12.5, loc=3)
        axins.set_xlim(-45, -37)
        axins.set_ylim(-23, -17)

        plt.xticks(visible=False)
        plt.yticks(visible=False)

        m2 = Basemap(llcrnrlon=-45,llcrnrlat=-23,urcrnrlon=-37,urcrnrlat=-17, ax=axins)
        m2.drawmapboundary(fill_color='white')
        m2.fillcontinents(color='lightgrey', lake_color='white')

        for port_polygon in port_polygons:
            x, y = zip(*port_polygon)
            m2.plot(x,y,markersize=0)
            axins.fill(x,y,alpha=0.2)

        mark_inset(ax, axins, loc1=4, loc2=2, fc="none", ec="0.5")

    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Python\port_polygons_zoom.png',
                bbox_inches='tight', dpi=100)

#####

def port_polygons_geojson(port_polygons = generate_port_polygons()):
    '''
    Writes the port polygon coordinates generated by generate_port_polygons to .geojson-files for
    geofence evaluation in ais.py-script
    '''

```

---

```

string_port_polygons = ['singapore', 'tubarao', 'qingdao', 'rotterdam', 'bolivar', 'dampier',
                        'oita', 'haypoint', 'baltimore', 'gwangyang', 'narvik', 'portcartier',
                        'richardsbay', 'gangavaram', 'saldanhabay', 'banjarmasin', 'goa',
                        'nouadhibou', 'rizhao', 'dekheila']

n = len(port_polygons)
for i in range(n):
    polygon = Polygon([port_polygons[i]])
    features = []
    features.append(Feature(geometry=polygon))

    feature_collection=FeatureCollection(features)

    with open(r'PortPolygons\polygon_' + string_port_polygons[i] + r'.geojson', 'w') as f:
        dump(feature_collection, f)

#####

def path_map():
    df = pd.read_csv(CAPESIZE_ROUTES_PATH)
    print(df)

    fig, ax = plt.subplots(figsize=(15,15))
    m = Basemap(projection='cyl', lon_0=0, resolution='l')
    m.drawmapboundary(fill_color='white')
    m.fillcontinents(color='lightgrey', lake_color='white')

    already_plotted_poo = list()
    already_plotted_pod = list()

    for i in range(len(df)):
        lon_start = df.loc[i, 'Lon_Po0']
        lat_start = df.loc[i, 'Lat_Po0']

        lon_end = df.loc[i, 'Lon_PoD']
        lat_end = df.loc[i, 'Lat_PoD']

        if df.loc[i, 'Commodity'] == 'Ore':
            color = '#1e4f9c'
        else:
            color = '#87b1dd'

        m.drawgreatcircle(lon_start, lat_start, lon_end, lat_end, linewidth=1,color=color)

        m.plot(lon_start, lat_start, marker='o', color='#1e4f9c', markersize=5, alpha=1)
        m.plot(lon_end, lat_end, marker='o', color='#87b1dd', markersize=5, alpha=1)

        if df.loc[i, 'Po0'] not in already_plotted_poo:
            already_plotted_poo.append(df.loc[i, 'Po0'])
            label = df.loc[i, 'Po0']
            if df.loc[i, 'Po0'] == 'Rizhao':
                plt.text(lon_start, lat_start, label, ha='right', va='top', fontsize=6)
            else:
                plt.text(lon_start, lat_start, label, ha='right', va='bottom', fontsize=6)

        if df.loc[i, 'PoD'] not in already_plotted_pod:
            already_plotted_pod.append(df.loc[i, 'PoD'])
            label = df.loc[i, 'PoD']

```

---

---

```

if df.loc[i,'PoD'] == 'Qingdao':
    plt.text(lon_end, lat_end, label, ha='center', va='bottom', fontsize=6)
elif df.loc[i,'PoD'] == 'Oita':
    plt.text(lon_end, lat_end, label, ha='left', va='top', fontsize=6)
else:
    plt.text(lon_end, lat_end, label, ha='left',va='bottom',fontsize=6)

plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Python\Capesize_routes.png',
            bbox_inches='tight', dpi=100)

#####

if __name__ == "__main__":
    start = time.time()
    #generate_polygons()
    #ocean_polygons(polygons = generate_polygons())
    #generate_port_polygons()
    #port_polygons_plot()
    #port_polygons_geojson(port_polygons = generate_port_polygons())
    #port_polygons = generate_port_polygons()
    #ocean_polygons_geojson(ocean_polygons=generate_polygons())
    path_map()
    end = time.time()
    print(f"Runtime of the program is {end - start}")

```

---

## E.7 FE.py

```
import numpy as np
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point, LineString, Polygon
import matplotlib.pyplot as plt
import time
import datetime as dt
from datetime import datetime
from collections import defaultdict
#plt.style.use('seaborn-dark-palette')
from tqdm import tqdm

#Importing DataPaths
from configs.project_settings import AIS_RAW_DATA_PATH, SAMPLE_SET_PATH, AIS_PROCESSED_DATA_PATH
from configs.project_settings import CLARKSONS_RAW_DATA_PATH, CLARKSON_PATH, CLARKSON_TEST_PATH
from configs.project_settings import AIS_PROCESSED_DATA_SPLIT1, AIS_PROCESSED_DATA_SPLIT2
from configs.project_settings import AIS_PROCESSED_DATA_SPLIT3, AIS_PROCESSED_DATA_SPLIT4
from configs.project_settings import FEATURE_SPLIT_1, FEATURE_SPLIT_2, FEATURE_SPLIT_3
from configs.project_settings import FEATURE_SPLIT_4, FINAL_FEATURE_SET
from configs.project_settings import FEATURE_SPLIT_V1, FEATURE_SPLIT_V2, FEATURE_SPLIT_V3
from configs.project_settings import FEATURE_SPLIT_V4, FINAL_FEATURE_SET_2, FEATURES

#Importing Code
import polygons as pg
import clarksons as cl

#####

def ais_data(sample=False, processed=False, split1=False, split2=False, split3=False,
            split4=False):
    '''
    Extracts:
        AIS data from .csv-file
    Returns:
        AIS DataFrame
    '''
    if sample == True:
        ais_df = pd.read_csv(SAMPLE_SET_PATH)
    elif processed == True:
        ais_df = pd.read_csv(AIS_PROCESSED_DATA_PATH)
    elif split1 == True:
        ais_df = pd.read_csv(AIS_PROCESSED_DATA_SPLIT1)
    elif split2 == True:
        ais_df = pd.read_csv(AIS_PROCESSED_DATA_SPLIT2)
    elif split3 == True:
        ais_df = pd.read_csv(AIS_PROCESSED_DATA_SPLIT3)
    elif split4 == True:
        ais_df = pd.read_csv(AIS_PROCESSED_DATA_SPLIT4)

    ais_gdf = gpd.GeoDataFrame(ais_df, geometry=gpd.points_from_xy(ais_df.longitude,
                                                                ais_df.latitude))

    return ais_gdf
```



---

```

#####

def geofence_processing(df, polygon_area, zone=False, port=False):
    """
    Processes data from df (AIS GeoDataFrame), inserting columns
    indicating if a recording has been made inside/outside the polygon_area
    """
    if zone == True:
        polygon = gpd.read_file(r"OceanPolygons\polygon_" + polygon_area + ".geojson")
        mask = (polygon.loc[0, 'geometry'])
    elif port == True:
        polygon = gpd.read_file(r"PortPolygons\polygon_" + polygon_area + ".geojson")
        mask = (polygon.loc[0, 'geometry'])

    pip_mask_geofence = df.within(mask)
    df.loc[:, polygon_area] = pip_mask_geofence
    df[polygon_area] = df[polygon_area].replace({True:1, False:0})

    return df

#####

def df_to_array(df, polygon_area):
    """
    Takes the AIS DataFrame as input, and converts the
    essential data into numpy array format
    Returns:
        Array from the AIS database, with columns:
            MMSI
            Year
            Week
            Polygon_area
    """
    array = df[["mmsi", "Year", "Week", polygon_area]].to_numpy()
    return array

#####

def df_to_array_nav(df):
    array = df[["mmsi", "Year", "Week", "nav_status"]].to_numpy()
    return array

#####

def vessels_dict(df, polygon_area, vessel_list):
    """
    Utilizes the data in the array converted from the AIS DataFrame
    to write MMSI values of vessels registered in polygon_area
    to the corresponding set of combination year and week
    Returns:
        A dictionary containing the set of vessels recorded
        in polygon_area at a weekly frequency. For example:
        {(2019, 1): [370137000, 356142000], (2019,3): [564558000]}
    """
    vessel_dict = defaultdict(lambda: set())
    array = df_to_array(df, polygon_area)

```

---

```

for element in array:
    if element[3] == 1:
        for vessel in vessel_list:
            if vessel.mmsi == element[0]:
                vessel_dict[(element[1], element[2])].add(vessel)

return vessel_dict

#####

def vessels_count_cap_dict(df, polygon_area, vessel_list):
    vessel_count_dict = defaultdict()
    vessel_dict = vessels_dict(df, polygon_area, vessel_list)

    for key in vessel_dict.keys():
        vessel_count_dict[key] = {"count": len(vessel_dict[key]),
                                   "sum_dwt": sum([vessel.dwt for vessel in vessel_dict[key]])}

    #vessel_count_dict[(2019, 31)]['count']

    return vessel_count_dict

#####

class Vessel:
    def __init__(self, mmsi, dwt):
        self.mmsi = mmsi
        self.dwt = dwt

#####

def mmsi_cap_array(db):
    '''
    Returns an array of objects with MMSI and Dwt from the Capesize database
    '''
    array = db[["mmsi", "Dwt"]].to_numpy().astype(int)
    array = [Vessel(arr[0], arr[1]) for arr in array]

    return array

#####

def write_features(db, df, polygon_area, vessel_count_dict, split1=False, split2=False,
                  split3=False, split4=False):

    df[polygon_area] = np.nan
    df[polygon_area + '_cap'] = np.nan
    for i in range(696969):
        print(69)
    for index, row in df.iterrows():
        year, week = row['Year'], row['Week']
        if (year, week) in vessel_count_dict:
            df.loc[index, polygon_area] = vessel_count_dict[(year, week)]['count']
            df.loc[index, polygon_area + '_cap'] = vessel_count_dict[(year, week)]['sum_dwt']
        pass

    if split1 == True:

```

---

---

```

    df.to_csv(FEATURE_SPLIT_1, index=False)
elif split2 == True:
    df.to_csv(FEATURE_SPLIT_2, index=False)
elif split3 == True:
    df.to_csv(FEATURE_SPLIT_3, index=False)
elif split4 == True:
    df.to_csv(FEATURE_SPLIT_4, index=False)

#####

def combine_split_features(df1, df2, df3, df4):
    df1 = df1.replace({np.nan: 0})
    df2 = df2.replace({np.nan: 0})
    df3 = df3.replace({np.nan: 0})
    df4 = df4.replace({np.nan: 0})
    feature_df = df1

    feature_cols = ['tubarao', 'tubarao_cap', 'qingdao', 'qingdao_cap', 'rotterdam', 'rotterdam_cap',
                    'bolivar', 'bolivar_cap', 'dampier', 'dampier_cap', 'oita', 'oita_cap',
                    'haypoint', 'haypoint_cap', 'baltimore', 'baltimore_cap', 'gwangyang',
                    'gwangyang_cap', 'narvik', 'narvik_cap', 'portcartier', 'portcartier_cap',
                    'richardsbay', 'richardsbay_cap', 'gangavaram', 'gangavaram_cap', 'saldanhabay',
                    'saldanhabay_cap', 'banjarmasin', 'banjarmasin_cap', 'goa', 'goa_cap',
                    'nouadhibou', 'nouadhibou_cap', 'rizhao', 'rizhao_cap', 'dekheila', 'dekheila_cap',
                    'atlantic', 'atlantic_cap', 'FE', 'FE_cap', 'CP', 'CP_cap', 'EstP', 'EstP_cap',
                    'NWE', 'NWE_cap', 'indi', 'indi_cap', 'Med', 'Med_cap']

    for item in tqdm(feature_cols):
        for index, row in feature_df.iterrows():
            feature_df.loc[index, item] += df2.loc[index,
                                                    item] + df3.loc[index, item] + df3.loc[index, item]

    feature_df.to_csv(FINAL_FEATURE_SET, index=False)

#####

def vessels_active_dict(df, vessel_list):
    vessel_active_dict = defaultdict(lambda: set())
    array = df_to_array_nav(df)

    for num, element in tqdm(enumerate(array), total=array.shape[0]):
        if element[3] == 0:
            for vessel in vessel_list:
                if vessel.mmsi == element[0]:
                    vessel_active_dict[(element[1], element[2])].add(vessel)
    pass

    return vessel_active_dict

#####

def weekly_capacity(df, vessel_list):
    weekly_cap_dict = defaultdict()
    vessels_active = vessels_active_dict(df, vessel_list)

    for key in tqdm(vessels_active.keys()):
        weekly_cap_dict[key] = {"count": len(vessels_active[key]),

```

---

```

        "cap": sum([vessel.dwt for vessel in vessels_active[key]])}
    pass

    return weekly_cap_dict

#####

def write_weekly_cap(df, weekly_cap_dict, split1=False, split2=False, split3=False,
                    split4=False):

    df["sum_cap"] = np.nan
    df["n_active"] = np.nan
    for index, row in tqdm(df.iterrows(), total=df.shape[0]):
        year, week = row['Year'], row['Week']
        if (year, week) in weekly_cap_dict:
            df.loc[index, "sum_cap"] = weekly_cap_dict[(year, week)]['cap']
            df.loc[index, "n_active"] = weekly_cap_dict[(year, week)]['count']
        pass

    if split1 == True:
        df.to_csv(FEATURE_SPLIT_V1, index=False)
    elif split2 == True:
        df.to_csv(FEATURE_SPLIT_V2, index=False)
    elif split3 == True:
        df.to_csv(FEATURE_SPLIT_V3, index=False)
    elif split4 == True:
        df.to_csv(FEATURE_SPLIT_V4, index=False)

#####

def combine_split_features_active(df1, df2, df3, df4):
    df1 = df1.replace({np.nan: 0})
    df2 = df2.replace({np.nan: 0})
    df3 = df3.replace({np.nan: 0})
    df4 = df4.replace({np.nan: 0})
    feature_df = df1

    feature_cols = ['sum_cap', 'n_active']

    for item in tqdm(feature_cols):
        for index, row in feature_df.iterrows():
            feature_df.loc[index, item] += df2.loc[index,
                                                    item] + df3.loc[index, item] + df3.loc[index, item]

    feature_df.to_csv(FINAL_FEATURE_SET_2, index=False)

#####

def write_fleet_utilization(df):
    tot_fleet_cap = 240407734

    df['fleet_util'] = np.nan
    df = df.replace({np.nan: 0})

    for index, row in df.iterrows():
        df.loc[index, 'fleet_util'] = df.loc[index, 'sum_cap'] / tot_fleet_cap

```

---

---

```

df.to_csv(FEATURES, index=False)

#####

def write_zone_factor(df):
    new_cols = ['atlantic', 'FE', 'CP', 'EstP', 'NWE', 'indi', 'Med']
    for item in tqdm(new_cols):
        df[item + '_pct'] = np.nan
        for index, row in df.iterrows():
            weekly_cap = df.loc[index, 'Sum_tot_cap']
            df.loc[index, item + '_pct'] = df.loc[index, item + '_cap'] / weekly_cap
    pass
df = df.replace({np.nan: 0})

df.to_csv(FEATURES, index=False)

#####

def plotting(df, fleet_utilization=False, zone_factor=False):

    if fleet_utilization == True:
        #df.plot(kind='line', x='Date', y='fleet_util')
        df.iloc[0:52].plot(kind='line', x='Date', y='fleet_util')
        plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\fleet_utilization.png',
                    bbox_inches='tight',dpi=100)

    if zone_factor == True:
        df1 = pd.read_csv(FEATURES, usecols=["atlantic_pct", "FE_pct", "CP_pct",
                                             "EstP_pct", "NWE_pct", "indi_pct", "Med_pct"])
        corr = df1.corr()
        df1.boxplot()
        plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\boxplot_zone_cap.png',
                    bbox_inches='tight',dpi=100)
        print(corr)
        desc = df1.describe()
        print(desc)
        desc2 = corr.describe()
        print(desc2)

#####

if __name__ == "__main__":
    start = time.time()
    start_readable = time.ctime(start)
    print(f"\nInitialized run at: {start_readable}\n")

    #----- Output Processed AIS DataFrame -----#
    #ais_df = ais_data(sample=True)
    #print(ais_df)

    #----- Geofence Processing -----#
    #ais_df, ais_gdf = ais_data(sample=True)
    #ais_df = geofence_processing(ais_gdf, 'atlantic', zone=True)
    #print(ais_df)

    #---- Convert from DataFrame to Array ----#
    #ais_df, ais_gdf = ais_data(sample=True)

```

---

---

```

#ais_df = geofence_processing(ais_gdf, 'atlantic', zone=True)
#array = df_to_array(ais_df, 'atlantic')
#print(array)

#---- Extract Set of Vessels in Area of Interest ----#
#ais_df, ais_gdf = ais_data(sample=True)
#ais_df = geofence_processing(ais_gdf, 'atlantic', zone=True)
#spot_df, capesize_db = cl.clarksons_data()
#vessel_list = mmsi_cap_array(capesize_db)
#vessel_dict = vessels_dict(ais_df, 'atlantic', vessel_list)
#print(vessel_dict)

#---- Extract Num of Vessels & Capacity in Area of Interest ----#
#ais_gdf = ais_data(sample=True)
#ais_df = geofence_processing(ais_gdf, 'atlantic', zone=True)
#spot_df, capesize_db = cl.clarksons_data()
#vessel_list = mmsi_cap_array(capesize_db)
#vessel_count_dict = vessels_count_cap_dict(ais_df, 'atlantic', vessel_list)

#---- Extract Num of Active Vessels & Capacity in World on Weekly Frequency ----#
#ais_gdf = ais_data(split4=True)
#print('AIS Extraction Finished')
#spot_df, capesize_db = cl.clarksons_data()
#print('Clarksons Data Extraction Finished')
#vessel_list = mmsi_cap_array(capesize_db)
#print('Vessel List Extraction Finished')
#weekly_cap = weekly_capacity(ais_gdf, vessel_list)
#write_weekly_cap(spot_df, weekly_cap, split4=True)

#---- Write Features to Split Feature Dataset ----#
#ais_gdf = ais_data(split4=True)
#print('AIS Extraction Finished')
#spot_df, capesize_db = cl.clarksons_data()
#print('Clarksons Data Extraction Finished')
#vessel_list = mmsi_cap_array(capesize_db)
#print('Vessel List Extraction Finished')
#string_port_polygons = ['tubarao', 'qingdao', 'rotterdam', 'bolivar', 'dampier',
# 'oita', 'haypoint', 'baltimore', 'gwangyang', 'narvik', 'portcartier', 'richardsbay',
# 'gangavaram', 'saldanhabay', 'banjarmasin', 'goa', 'nouadhibou', 'rizhao', 'dekheila']
#string_ocean_polygons = ['atlantic', 'FE', 'CP', 'EstP', 'NWE', 'indi', 'Med']
#for port in tqdm(string_port_polygons):
#    ais_df = geofence_processing(ais_gdf, port, port=True)
#    vessel_count_dict = vessels_count_cap_dict(ais_df, port, vessel_list)
#    write_features(capesize_db, spot_df, port, vessel_count_dict, split4=True)
#for zone in tqdm(string_ocean_polygons):
#    ais_df = geofence_processing(ais_gdf, zone, zone=True)
#    vessel_count_dict = vessels_count_cap_dict(ais_df, zone, vessel_list)
#    write_features(capesize_db, spot_df, zone, vessel_count_dict, split4=True)

#---- Combine the Split Feature Datasets ----#
#df1 = pd.read_csv(FEATURE_SPLIT_1)
#df2 = pd.read_csv(FEATURE_SPLIT_2)
#df3 = pd.read_csv(FEATURE_SPLIT_3)
#df4 = pd.read_csv(FEATURE_SPLIT_4)
#combine_split_features(df1, df2, df3, df4)

#df1 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT1)

```

---

---

```
#df2 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT2)
#df3 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT3)
#df4 = pd.read_csv(AIS_PROCESSED_DATA_SPLIT4)
#df = pd.read_csv(r'AIS_Data\AIS_2019_processed_02.csv')
#print(df)

#---- Combine the Split Feature Datasets (Active) ----#
#df1 = pd.read_csv(FEATURE_SPLIT_V1)
#df2 = pd.read_csv(FEATURE_SPLIT_V2)
#df3 = pd.read_csv(FEATURE_SPLIT_V3)
#df4 = pd.read_csv(FEATURE_SPLIT_V4)
#combine_split_features_active(df1, df2, df3, df4)

#---- Write Fleet Utilization Factor to Feature set ----#
#df = pd.read_csv(FEATURES)
#write_fleet_utilization(df)

#---- Write Zone Utilization Factor to Feature set ----#
#df = pd.read_csv(FEATURES)
#write_zone_factor(df)

#---- Plot Selection of Features ----#
df = pd.read_csv(FEATURES)
plotting(df, zone_factor=True)

end = time.time()
end_readable = time.ctime(end)
print(f"\nCode ended at: {end_readable}")
print(f"Runtime of the program is {end - start}")
```

---

## E.8 data\_preparation.py

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
import time

from configs.project_settings import FEATURES

def plot_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(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\plot_time_series.pdf',
               bbox_inches='tight')

#####

def 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):
    data = data.diff().dropna()

    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):
    data = data.apply(np.log)
    data = data.replace([np.inf, -np.inf], 0)
    data = data.dropna().diff().dropna().reset_index(drop=True)

    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):
    AF = pd.DataFrame()

    for name, series in data.iteritems():
        result = adfuller(series)
        AF.at['ADF Statistics', 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, data, diff=True):
    if diff == True:
        df_data, scaler = difference_normalize(data)
    else:
        df_data = data

    dataframe = pd.DataFrame()
    for name, data 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)

    #train_size = round(len(data) * 0.9)
    #train_data = dataframe[0:train_size]

    #return train_data.dropna()

```

---

---

```

    return dataframe

#####

def inverse(pred, test, last_obs, scl):
    inverted = scl.inverse_transform(pred)
    return inverted

#####

def inverse_transform(pred, test, last_obs, scl):
    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_transform(pred, test, last_obs, scl):
    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__":
    start = time.time()
    start_readable = time.ctime(start)
    print(f"\nInitialized run at: {start_readable}\n")

    #data = pd.read_csv(r'AIS_Data\FinalFeatureSetTesting_v01.csv')
    data = pd.read_csv(FEATURES)
    #plot_time_series(df)
    #data, scaler = normalize(data)
    #data, scaler = difference_normalize(data)
    #data, scaler = log_difference_normalize(data)
    ADF = stationarity_check(data)
    print(data)
    print(ADF)
    ADF_transposed = ADF.transpose()
    ADF_transposed.to_csv(r'ADF_Features_nodiff.csv')

    end = time.time()
    end_readable = time.ctime(end)
    print(f"\nCode ended at: {end_readable}")
    print(f"Runtime of the program is {end - start}")

```

---

---

## E.9 feature\_importance\_score.py

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import f_regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import RidgeCV, LinearRegression
import matplotlib.pyplot as plt
from minepy import MINE
import seaborn as sns
import time
import data_preparation as DP

from configs.project_settings import FEATURES

def scale_rank(rank, name, order=1):
    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, n_features, plot=False):
    df = DP.supervised_learning(lag, data)
    #df.to_csv(r'FeatureSets\FeatureSet_Test_01.csv', index=False)
    array = df.dropna().values.astype('float32')

    #Split into input and output
    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)

    #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)

#Sort the ranked features with regards to Mean Score
features_sorted = ratings.sort_values('Mean Score', ascending=False)
#Extract only top number of features
top_features = features_sorted.index[:n_features]

if plot == True:
    #Plot top features based on mean score
    ratings['Mean Score'].sort_values()[-n_features:].plot(kind='bar',
        figsize = (10,2), color='b')
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\mean_score.png',
        bbox_inches='tight')

    j = 0
    for i in range(0, len(ratings), 50):
        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(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\FI_' + str(j) + r'.png',
            bbox_inches='tight')

return top_features

#####

if __name__ == "__main__":
    start = time.time()
    start_readable = time.ctime(start)
    print(f"\nInitialized run at: {start_readable}\n")

    #data = pd.read_csv(r'AIS_Data\FinalFeatureSetTesting_v01.csv')
    data = pd.read_csv(FEATURES)
    n_features = 20
    top_features = main(1, data, n_features, plot=True)

    end = time.time()
    end_readable = time.ctime(end)
    print(f"\nCode ended at: {end_readable}")
    print(f"Runtime of the program is {end - start}")

```

---

---

## E.10 feature\_selection.py

```
#Data Manipulation
from matplotlib import colors
from numpy.core.numeric import Inf
from numpy.lib.function_base import average
import pandas as pd
import numpy as np

#Visualization
import matplotlib.pyplot as plt
import seaborn as sns

from collections import OrderedDict
from sklearn import linear_model

from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import f1_score, make_scorer
from sklearn.model_selection import cross_val_score, KFold
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn import linear_model, tree, ensemble
from sklearn.feature_selection import RFE

from configs.project_settings import FEATURES
import time

import data_preparation as DP

def rmse(score):
    rmse = np.sqrt(-score)
    print(f'rmse = {"{: .4f}".format(rmse)}')
    return rmse

#####

def plot_feature_importances(df, n=20, threshold=None):
    """
    Plots n most important features. Also plots the cumulative importance if threshold
    is specified and prints the number of features needed to reach threshold
    cumulative importance. Intended for use with any tree-based feature importances.

    Args:
        df (DataFrame): Dataframe of feature importances. Columns must be "feature" and "importance"
        n (int): Number of most important features to plot. Default is 10.
        threshold (float): Threshold for cumulative importance plot. If not provided,
        no plot is made. Default is None.

    Returns:
        df (DataFrame): Dataframe ordered by feature importances with a normalized column
        (sums to 1)

    Note:
        * Normalization in this case means sums to 1
        * Cumulative importance is calculated by summing features from most to least important
        * A threshold of 0.9 will show the most important features needed to reach 90% of
```

---

```

        cumulative importance
    '''
    #Sort Features with most important at the head
    df = df.sort_values('importance', ascending=False).reset_index(drop=True)

    #Normalize the feature importances to add up to one and calculate cumulative importance
    df['importance_normalized'] = df['importance'] / df['importance'].sum()
    df['cumulative_importance'] = np.cumsum(df['importance_normalized'])

    #Bar plot of n most important features
    plt.rcParams['font.size'] = 12
    df.loc[:n, :].plot.barh(y = 'importance_normalized',
                           x = 'feature', color = 'darkblue',
                           edgecolor = 'k', figsize = (12, 8),
                           legend = False, linewidth=2)

    plt.xlabel('Normalized Importance', size=18); plt.ylabel('');
    plt.title(f'{n} Most Important Features', size=18)
    plt.gca().invert_yaxis()
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Chapter4_MA\FI.png',
               bbox_inches='tight')

    if threshold:
        #Cumulative Importance plot
        plt.figure(figsize=(8,6))
        plt.plot(list(range(len(df))), df['cumulative_importance'], 'b-')
        plt.xlabel('Number of features', size=16); plt.ylabel('Cumulative Importance', size=16);
        plt.title('Cumulative Feature Importance', size=18);

        #Number of features needed for threshold cumulative importance
        #This is the index (will need to add 1 for the actual number)
        importance_index = np.min(np.where(df['cumulative_importance'] > threshold))

        #Add vertical line to plot
        plt.vlines(importance_index + 1, ymin=0, ymax=1.05, linestyle=':', color='black')
        plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\FI_cumulative_plot.png',
                   bbox_inches='tight')

        print('{:} features required for {:.0f}% of cumulative importance.'.format(importance_index
                                          + 1, 100*threshold))

    return df

#####

def model_tuning():
    #Read data
    unprocessed_data = pd.read_csv(FEATURES)

    #Define Train/Test-ratio
    train_test_ratio = 0.8
    train_size = round(len(unprocessed_data) * train_test_ratio)

    #Set lag
    lag = 1

    #Convert unprocessed dataframe with features to supervised dataframe, with corresponding lag

```

---

---

```

data = DP.supervised_learning(lag, unprocessed_data, diff=True)

#Split data into Train and Test sets
train_set, test_set = data[0:train_size], data[train_size:]
X = train_set.copy()
train_target = X['price(t)']
X.drop(['price(t)'], axis=1, inplace=True)

#Get list of all features
features = list(X.columns)

estimators = [50, 100, 150, 200, 250, 300, 350]
best_estimator = None
best_rmse = Inf
for count in estimators:
    score = cross_val_score(ensemble.RandomForestRegressor(n_estimators=count,
        random_state=1337420), X, train_target, scoring="neg_mean_squared_error")
    print(f'For estimators: {count}')
    rmse_val = rmse(score.mean())
    #Lowest rmse = 0.1131 -> estimators = 300
    if rmse_val < best_rmse:
        best_estimator = count
        best_rmse = rmse_val

return X, train_target, features, best_estimator

#####

def feature_importance():
    X, train_target, features, estimator = model_tuning()

    #Create and fit RF model
    model = RandomForestRegressor(estimator, random_state=1337420)
    model.fit(X, train_target)

    feature_importances = pd.DataFrame({'feature': features,
        'importance': model.feature_importances_})

    fi_df = plot_feature_importances(feature_importances, threshold=0.95)
    return X, train_target, fi_df, estimator

#####

def feature_selection(n_features):
    X, train_target, fi_df, estimator = feature_importance()

    #Create correlation matrix
    corr_matrix = X.corr()

    #Select upper triangle of correlation matrix
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))

    #Find index of feature columns with correlation greater than 0.975
    to_drop = [column for column in upper.columns if any(abs(upper[column]) > 0.975)]

    #Drop feature columns with correlation greater than 0.975
    X = X.drop(columns=to_drop)

```

---

---

```

#Recursive Feature Elimination with Random Forest
#Create a model for feature selection
estimator = RandomForestRegressor(random_state=1337420, n_estimators=estimator)

#Create the object
selector = RFE(estimator, n_features_to_select=n_features, step=1)
selector = selector.fit(X, train_target)

selected_features = list()
for index, item in enumerate(selector.support_):
    if item == True:
        selected_features.append(X.iloc[:, index].name)

print(selected_features)
return selected_features

#####

if __name__ == "__main__":
    start = time.time()
    start_readable = time.ctime(start)
    print(f"\nInitialized run at: {start_readable}\n")

    n_features = 20
    sel_n_features = feature_selection(n_features-1)

    end = time.time()
    end_readable = time.ctime(end)
    print(f"\nCode ended at: {end_readable}")
    print(f"Runtime of the program is {end - start}")

```



---

## E.11 ML\_models.py

```
from sklearn.utils import shuffle
from LSTM_multivariate import prepare_data_LSTM
from keras.layers.wrappers import Bidirectional
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.metrics as metrics
from sklearn.metrics import make_scorer
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV
from sklearn import linear_model
from sklearn.ensemble import RandomForestRegressor
import time
import feature_importance_score as FIS
import data_preparation as DP
from catboost import CatBoostRegressor
from sklearn.tree import export_graphviz
from sklearn import tree
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error

from keras.layers import Input, Dense, Dropout, LSTM
from keras.models import Sequential
from keras import backend as K
from keras.wrappers.scikit_learn import KerasRegressor

from configs.project_settings import FEATURES

import feature_selection as FS

#####

def regression_results(y_true, y_pred):
    explained_variance = metrics.explained_variance_score(y_true, y_pred)
    mae = metrics.mean_absolute_error(y_true, y_pred)
    mse = metrics.mean_squared_error(y_true, y_pred)
    #mean_squared_log_error = metrics.mean_squared_log_error(y_true, y_pred)
    #r2 = metrics.r2_score(y_true, y_pred)
    mape = metrics.mean_absolute_percentage_error(y_true, y_pred)

    print('Explained Variance: ', round(explained_variance,4))
    #print('Mean Squared Log Error: ', round(mean_squared_log_error,4))
    #print('R2: ', round(r2,4))
    print('MAE: ', round(mae,5))
    #print('MSE: ', round(mse,4))
    print('RMSE: ', round(np.sqrt(mse),5))
    print('MAPE: ', round(mape,5))

#####

def rmse(actual, predict):
    predict = np.array(predict)
    actual = np.array(actual)
    distance = predict - actual
    square_distance = distance ** 2
    mean_square_distance = square_distance.mean()
    score = np.sqrt(mean_square_distance)
```

---

```

    return score

#####

def plot_models(y_test, predictions, min):
    print(y_test[min:].values)
    print(predictions[0][min:])
    print(predictions[1][min:])
    plt.plot(y_test[min:].values, label=True, linewidth=3, alpha=0.7)
    plt.plot(predictions[0][min:], label="LR")
    plt.plot(predictions[1][min:], label="RF")
    #plt.plot(predictions[2][min:], label="NN")
    plt.legend(loc='upper right')
    plt.rcParams["figure.figsize"] = (15,15)
    plt.rcParams["legend.fontsize"] = 14
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Figures\Python\TestPlots\LR_RF.png')

#####

def root_mean_squared_error(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1))

#####

def create_model():

    K.clear_session()

    model = Sequential()
    model.add(LSTM(1000, input_shape=(X_train_lstm.shape[1], X_train_lstm.shape[2])))
    model.add(Dropout(0.3))
    model.add(Dense(1, activation='linear'))
    model.compile(loss=root_mean_squared_error, optimizer='adam')

    return model

#####

def prepare_data_LSTM(data, sliding_window):
    n_features = len(data[0])

    #Split the data into Features (x) and Predicted Value (y)
    x, y = data[:, :n_features], data[:, -1].reshape(-1,1)
    #Create Features (shape=[samples, sliding_window, n_features]) and Predicted Value arrays
    #The Feature Array (X) will contain the next # sliding_window data points for all features
    #see example below
    '''
    array([[0.51559937, 0.55616438, 0.1875      , ..., 0.          ,
           0.          , 0.48648649],
          [0.3087028 , 0.41917807, 0.6875      , ..., 0.83720928,
           0.83167738, 0.52252251],
          [0.27093595, 0.3890411 , 0.25          , ..., 0.79069769,
           0.7868582 , 0.32432431],
          [0.          , 0.18630135, 0.625          , ..., 0.67441857,
           0.67622554, 0.17117119]],

          [[0.3087028 , 0.41917807, 0.6875      , ..., 0.83720928,

```

---

```

        0.83167738, 0.52252251],
[0.27093595, 0.3890411 , 0.25      , ..., 0.79069769,
 0.7868582 , 0.32432431],
[0.      , 0.18630135, 0.625      , ..., 0.67441857,
 0.67622554, 0.17117119],
[0.19540229, 0.29041094, 0.25      , ..., 0.72093022,
 0.71738136, 0.4954955  ]],
'''
X, Y = np.empty((0, sliding_window, n_features)), np.empty((0))

for i in range(len(x)-sliding_window):
    X = np.vstack((X, [x[i:i+sliding_window]:]))
    Y = np.append(Y, y[i+sliding_window])
Y = np.reshape(Y, (len(Y), 1))

return X, Y

#####

def plot_rmse(rmse_lstm, rmse_LR, rmse_RF, train_len, all=False, top=False, sel=False):
    xthreshold = train_len-1
    train = []
    test = []
    X = []
    for index, item in enumerate(rmse_LR):
        X.append(index)
        if index <= xthreshold:
            train.append(True)
            test.append(False)
        else:
            train.append(False)
            test.append(True)
    X = np.array(X)
    fig, ax = plt.subplots(figsize=(15,15))
    plt.plot(X[train], rmse_lstm[train], linestyle='-', marker='o', color='b')
    plt.plot(X[test], rmse_lstm[test], linestyle='-', marker='v', color='b')
    plt.plot(X[train], rmse_LR[train], linestyle='-', marker='o', color='g')
    plt.plot(X[test], rmse_LR[test], linestyle='-', marker='v', color='g')
    plt.plot(X[train], rmse_RF[train], linestyle='-', marker='o', color='r')
    plt.plot(X[test], rmse_RF[test], linestyle='-', marker='v', color='r')
    plt.title('RMSE Model Loss')
    plt.ylabel('RMSE')
    plt.xlabel('Time')
    plt.legend(['LSTM Train', 'LSTM Test', 'LR Train', 'LR Test', 'RF Train', 'RF Test'])
    if all == True:
        plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\RMSE_models_all_features.png',
                    bbox_inches='tight')
    if top == True:
        plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\RMSE_models_top_features.png',
                    bbox_inches='tight')
    if sel == True:
        plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\RMSE_models_sel_features.png',
                    bbox_inches='tight')

#####

def evaluate_results(real_LR, real_RF, real_lstm, pred_LR, pred_RF, pred_lstm, all=False,

```

---

---

```

        top=False, sel=False):
rmse_LR = np.sqrt(mean_squared_error(real_LR, pred_LR))
rmse_RF = np.sqrt(mean_squared_error(real_RF, pred_RF))
rmse_lstm = np.sqrt(mean_squared_error(real_lstm, pred_lstm))

mae_LR = mean_absolute_error(real_LR, pred_LR)
mae_RF = mean_absolute_error(real_RF, pred_RF)
mae_lstm = mean_absolute_error(real_lstm, pred_lstm)

mape_LR = np.mean(np.abs((real_LR - pred_LR) / real_LR))*100
mape_RF = np.mean(np.abs((real_RF - pred_RF) / real_RF))*100
mape_lstm = np.mean(np.abs((real_lstm - pred_lstm) / real_lstm))*100

print("=====")
print(f"Test scores for LR")
print('\nTest RMSE: ', rmse_LR)
print('\nTest MAE: ', mae_LR)
print('\nTest MAPE: ', mape_LR)
print("=====")

print("=====")
print(f"Test scores for RF")
print('\nTest RMSE: ', rmse_RF)
print('\nTest MAE: ', mae_RF)
print('\nTest MAPE: ', mape_RF)
print("=====")

print("=====")
print(f"Test scores for LSTM")
print('\nTest RMSE: ', rmse_lstm)
print('\nTest MAE: ', mae_lstm)
print('\nTest MAPE: ', mape_lstm)
print("=====")

forecast_LR = pd.DataFrame(data = pred_LR, columns=['Pred LR'])
forecast_LR['Real'] = pd.DataFrame(real_LR)

forecast_RF = pd.DataFrame(data = pred_RF, columns=['Pred RF'])
forecast_RF['Real'] = pd.DataFrame(real_RF)

forecast_lstm = pd.DataFrame(data = pred_lstm, columns=['Pred LSTM'])
forecast_lstm['Real'] = pd.DataFrame(real_lstm)

if all == True:
    forecast_LR.to_csv(r'ForecastingResults\LR_forecast_all_features.csv')
    forecast_RF.to_csv(r'ForecastingResults\RF_forecast_all_features.csv')
    forecast_lstm.to_csv(r'ForecastingResults\LSTM_forecast_all_features.csv')
if top == True:
    forecast_LR.to_csv(r'ForecastingResults\LR_forecast_top_features.csv')
    forecast_RF.to_csv(r'ForecastingResults\RF_forecast_top_features.csv')
    forecast_lstm.to_csv(r'ForecastingResults\LSTM_forecast_top_features.csv')
if sel == True:
    forecast_LR.to_csv(r'ForecastingResults\LR_forecast_sel_features.csv')
    forecast_RF.to_csv(r'ForecastingResults\RF_forecast_sel_features.csv')
    forecast_lstm.to_csv(r'ForecastingResults\LSTM_forecast_sel_features.csv')

print(forecast_LR['Pred LR'])

```

---

---

```

print(forecast_LR['Real'])
print(forecast_RF['Pred RF'])
print(forecast_RF['Real'])
print(forecast_lstm['Pred LSTM'])
print(forecast_lstm['Real'])

#Plot Forecast
fig, ax = plt.subplots(figsize=(15,15))
plt.plot(forecast_LR['Pred LR'], linestyle='--', label='LR Pred', color='b')
plt.plot(forecast_RF['Pred RF'], linestyle='--', label='RF Pred', color='r')
plt.plot(forecast_lstm['Pred LSTM'], linestyle='--', label='LSTM Pred', color='m')
plt.plot(forecast_LR['Real'], linestyle='-', label='LR & RF Real', color='k')
plt.plot(forecast_lstm['Real'], linestyle='-.', label='LSTM Real', color='k')
plt.legend()
plt.title('Model Forecasting Plot')
plt.ylabel('Spot Rate [$ /day]')
plt.xlabel('Time')
if all == True:
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Forecasting_models_all_features.png',
                bbox_inches='tight')
if top == True:
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Forecasting_models_top_features.png',
                bbox_inches='tight')
if sel == True:
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Forecasting_models_sel_features.png',
                bbox_inches='tight')

#####

def model_persistence(x):
    return x

#####

def plot_grid_search(cv_results, grid_param_1, grid_param_2, name_param_1, name_param_2,
                    file_type):
    #Get test scores mean and std for each grid search
    scores_mean = cv_results['mean_test_rmse']
    scores_mean = np.array(scores_mean).reshape(len(grid_param_2),len(grid_param_1))

    scores_sd = cv_results['std_test_rmse']
    scores_sd = np.array(scores_sd).reshape(len(grid_param_2),len(grid_param_1))

    # Plot Mean Grid search scores
    _, ax = plt.subplots(1,1)

    # Param1 is the X-axis, Param 2 is represented as a different curve (color line)
    for idx, val in enumerate(grid_param_2):
        ax.plot(grid_param_1, scores_mean[idx,:], '-o', label= name_param_2 + ': ' + str(val))

    #ax.set_title("Grid Search Scores", fontsize=20, fontweight='bold')
    ax.set_xlabel(name_param_1, fontsize=16)
    ax.set_ylabel('CV Average Score', fontsize=16)
    #ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1), fontsize=12)
    ax.legend(loc="upper center", bbox_to_anchor=(0.5, -0.3))
    ax.grid('on')
    plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\GS_mean_scores_' + file_type + r'.png',

```

---

---

```

        bbox_inches='tight')

# Plot Std Grid search scores
_, ax1 = plt.subplots(1,1)

# Param1 is the X-axis, Param 2 is represented as a different curve (color line)
for idx, val in enumerate(grid_param_2):
    ax1.plot(grid_param_1, scores_sd[idx,:], '-o', label=name_param_2 + ': ' + str(val))

ax1.set_xlabel(name_param_1, fontsize=16)
ax1.set_ylabel('CV Std Score', fontsize=16)
#ax1.legend(loc='upper left', bbox_to_anchor=(1.05, 1), fontsize=12)
ax1.legend(loc="upper center", bbox_to_anchor=(0.5, -0.3))
ax1.grid('on')
plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\GS_std_scores_' + file_type + r'.png',
            bbox_inches='tight')

#####

if __name__ == "__main__":
    start = time.time()
    start_readable = time.ctime(start)
    print(f"\nInitialized run at: {start_readable}\n")

    #Fix random seed for reproducibility
    seed = 1337420
    np.random.seed(seed)

    #Select number of features and lag to utilize in models
    n_features = 20
    lag = 1

    #Initialize Feature Selection (True / False)
    all_features = False #Employ all constructed features
    top_features = False #Employ only top n features
    sel_features = False #Employ best combination of n features

    #Create Persistence Model (True/False)
    persistence_model = True

    #Plot RF Trees - True/False
    plot_trees = False
    #Run Catboost - True/False
    cboost = False

    #Extract Unprocessed Feature Data
    #unprocessed_data = pd.read_csv(r'AIS_Data\FinalFeatureSetTesting_v01.csv')
    unprocessed_data = pd.read_csv(FEATURES)
    features = list(unprocessed_data.columns)

    #Convert unprocessed raw data to supervised dataset
    data = DP.supervised_learning(lag, unprocessed_data, diff=False)

    #Difference and Normalize the Supervised Data
    data, scaler = DP.difference_normalize(data)

    #Define Train/Test-ratio

```

---

---

```

train_test_ratio = 0.8
train_size = round(len(data) * train_test_ratio)

#Feature Selection
if all_features == True:
    train_data, test_data = data[1:train_size], data[train_size:]

    X_train = train_data.loc[:, train_data.columns != "price(t)"]
    y_train = train_data["price(t)"]
    X_test = test_data.loc[:, test_data.columns != "price(t)"]
    y_test = test_data["price(t)"]
    fs = 'all'
elif top_features == True:
    #Extract Top Features
    top_n_features = FIS.main(lag, unprocessed_data, n_features, plot=True)

    #Split data into raw Train and Test sets
    train_raw_data, test_raw_data = data[1:train_size], data[train_size:]

    train_data = pd.DataFrame()
    test_data = pd.DataFrame()
    for item in top_n_features:
        train_data[item] = train_raw_data[item]
        test_data[item] = test_raw_data[item]

    X_train = train_data.loc[:, train_data.columns != "price(t)"]
    y_train = train_raw_data["price(t)"]
    X_test = test_data.loc[:, test_data.columns != "price(t)"]
    y_test = test_raw_data["price(t)"]
    fs = 'top'
elif sel_features == True:
    #Extract Selected Features
    sel_n_features = FS.feature_selection(n_features-1)

    #Split data into raw Train and Test sets
    train_raw_data, test_raw_data = data[1:train_size], data[train_size:]

    train_data = pd.DataFrame()
    test_data = pd.DataFrame()
    for item in sel_n_features:
        train_data[item] = train_raw_data[item]
        test_data[item] = test_raw_data[item]

    X_train = train_data.loc[:, train_data.columns != "price(t)"]
    y_train = train_raw_data["price(t)"]
    X_test = test_data.loc[:, test_data.columns != "price(t)"]
    y_test = test_raw_data["price(t)"]
    fs = 'sel'
elif all_features == False and top_features == False and sel_features == False
and persistence_model == False:
    print('Remember to select feature selection process or compute persistence model!')
    exit()

#Training
#format: tuple("name", ModelClass)
#Remember to also add search params
models = [

```

---

---

```

("LR", linear_model.Ridge(max_iter=20000, fit_intercept=True)),
("RF", RandomForestRegressor()),
#("NN", MLPRegressor(max_iter=20000))
("LSTM", KerasRegressor(build_fn=create_model))
]

search_params = [
    { 'alpha': [0.01, 0.05, 0.1, 0.5, 1, 2, 5], 'solver': ['auto', 'lsqr', 'sag', 'saga'] }, #LR
    { 'n_estimators': [20, 50, 100, 150, 200], 'max_depth': [i for i in range(5,15)] }, #RF
    { 'epochs': [10, 25, 50, 75, 100], 'batch_size': [8, 16, 32, 64, 128, 254]} #LSTM
]

scorers = {'rmse': 'neg_root_mean_squared_error'}

models_trained = []
history_acc = list()
history_val_acc = list()
history_loss = list()
history_val_loss = list()

rmse_score = make_scorer(rmse, greater_is_better=False)
if persistence_model == False:
    for i, m in enumerate(models):
        name, model = m
        print(f"Training Model {name} ...")

    ts_cv = TimeSeriesSplit(n_splits=4)
    if name == "LSTM":
        if all_features:
            #Need to make sure that price(t) is the last column in the dataframe
            data_lstm = data
            data_lstm = data_lstm.drop(columns="price(t)")
            data_lstm["price(t)"] = data["price(t)"]
            data_lstm = data.values.astype('float32')
        if top_features:
            data_lstm = pd.DataFrame()
            #Need to make sure that price(t) is the last column in the dataframe
            if "price(t)" in top_n_features:
                for item in top_n_features:
                    if item != "price(t)":
                        data_lstm[item] = data[item]
                data_lstm["price(t)"] = data["price(t)"]
            else:
                for item in top_n_features:
                    data_lstm[item] = data[item]
                data_lstm["price(t)"] = data["price(t)"]
            data_lstm = data_lstm.values.astype('float32')
        if sel_features:
            data_lstm = pd.DataFrame()
            #Need to make sure that price(t) is the last column in the dataframe
            if "price(t)" in sel_n_features:
                for item in sel_n_features:
                    if item != "price(t)":
                        data_lstm[item] = data[item]
                data_lstm["price(t)"] = data["price(t)"]
            else:
                for item in sel_n_features:

```

---



---

```

        data_lstm[item] = data[item]
        data_lstm["price(t)"] = data["price(t)"]
        data_lstm = data_lstm.values.astype('float32')

#data_lstm = data.values.astype('float32')
        sliding_window = 1
        train_data_lstm, test_data_lstm = data_lstm[0:train_size],
                                         data_lstm[train_size-sliding_window:]
        X_train_lstm, y_train_lstm = prepare_data_LSTM(train_data_lstm, sliding_window)
        X_test_lstm, y_test_lstm = prepare_data_LSTM(test_data_lstm, sliding_window)

        grid_search = GridSearchCV(estimator=model, param_grid=search_params[i],
                                   scoring=scorers, refit='rmse', cv=ts_cv)
        grid_search.fit(X_train_lstm, y_train_lstm)
        best_score = grid_search.best_score_
        best_model = grid_search.best_estimator_
        print(best_model.sk_params)
#LSTM: batch_size=254, epochs=100 | {'batch_size': 254, 'epochs': 100}

#Train set results:
        print("Train set results: ")
        print(regression_results(y_train_lstm, best_model.predict(X_train_lstm)))

#Test set results:
        print("Test set results: ")
        print(regression_results(y_test_lstm, best_model.predict(X_test_lstm)))
        plot_grid_search(grid_search.cv_results_, grid_search.param_grid['batch_size'],
                         grid_search.param_grid['epochs'], 'Batch Size', 'Epochs', name + '_' + fs)

        real_lstm_train = y_train_lstm
        y_pred_lstm_train = best_model.predict(X_train_lstm)
        rmse_lstm_train = [rmse(real_lstm_train[i][0],
                               y_pred_lstm_train[i]) for i in range(len(y_pred_lstm_train))]

        real_lstm_test = y_test_lstm
        y_pred_lstm_test = best_model.predict(X_test_lstm)
        rmse_lstm_test = [rmse(real_lstm_test[i][0],
                               y_pred_lstm_test[i]) for i in range(len(y_pred_lstm_test))]

#Concatenate the two RMSE_LSTM arrays
        rmse_lstm = np.concatenate((rmse_lstm_train, rmse_lstm_test), axis=0)

else:
        grid_search = GridSearchCV(estimator=model, cv=ts_cv, param_grid=search_params[i],
                                   scoring=scorers, refit='rmse')
        grid_search.fit(X_train, y_train)

        best_score = grid_search.best_score_
        best_model = grid_search.best_estimator_
        print(best_model)
#LR: alpha=100, max_iter=20000
#RF: max_depth=10, n_estimators=20, bootstrap=True, criterion: mse, verbose=0,

#Train set results
        print("Train set results (scaled): ")
        print(regression_results(y_train.values, best_model.predict(X_train)))

```

---

```

#Test set results
print("Test set results (scaled): ")
print(regression_results(y_test.values, best_model.predict(X_test)))
if name == "LR":
    print(best_model.solver)
    plot_grid_search(grid_search.cv_results_, grid_search.param_grid['alpha'],
                    grid_search.param_grid['solver'], 'Alpha', 'Solver', name + '_' + fs)
    history_LR = best_model.fit(X_train, y_train)
    pred_array_LR_train = history_LR.predict(X_train)
    real_array_LR_train = y_train.values
    rmse_LR_train = [rmse(real_array_LR_train[i],
                        pred_array_LR_train[i]) for i in range(len(real_array_LR_train))]

    pred_array_LR_test = history_LR.predict(X_test)
    real_array_LR_test = y_test.values
    rmse_LR_test = [rmse(real_array_LR_test[i],
                        pred_array_LR_test[i]) for i in range(len(real_array_LR_test))]

    #Concatenate the two RMSE_LR arrays
    rmse_LR = np.concatenate((rmse_LR_train, rmse_LR_test), axis=0)

if name == "RF":
    plot_grid_search(grid_search.cv_results_, grid_search.param_grid['max_depth'],
                    grid_search.param_grid['n_estimators'], 'Max Depth', 'N Estimators',
                    name + '_' + fs)
    #plot_search_results(grid_search)
    history_RF = best_model.fit(X_train, y_train)
    pred_array_RF_train = history_RF.predict(X_train)
    real_array_RF_train = y_train.values
    rmse_RF_train = [rmse(real_array_RF_train[i],
                        pred_array_RF_train[i]) for i in range(len(real_array_RF_train))]

    pred_array_RF_test = history_RF.predict(X_test)
    real_array_RF_test = y_test.values
    rmse_RF_test = [rmse(real_array_RF_test[i],
                        pred_array_RF_test[i]) for i in range(len(real_array_RF_test))]

    #Concatenate the two RMSE_RF arrays
    rmse_RF = np.concatenate((rmse_RF_train, rmse_RF_test), axis=0)

#For RF:
# history = best_model.fit(X_train, y_train)
# history.predict(X_train) -> array with predictive values for each sample
# y_train.values -> array with real values

#history_loss.append(history.history['loss'])
#history_val_loss.append(history.history['loss_val'])

models_trained.append((name, best_model))

if plot_trees == True:
    #Visualize the first 10 decision trees
    if name == 'RF':
        estimator = best_model.estimators_[10]
        #feature_names = X_train.columns.values.tolist()
        #target_names = y_train.name
        feature_names = X_train.columns.values.tolist()

```

---

---

```

target_names = y_train.name
#Export as dot file
export_graphviz(estimator,
                 out_file=r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\tree.dot',
                 feature_names = feature_names,
                 class_names = target_names,
                 rounded = True, proportion = False,
                 precision = 2, filled = True)
#Convert from dot file to png:
#https://dreampuf.github.io/GraphvizOnline

for i in range (0, 5):
    tree_num = i
    fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(4,4), dpi=800)
    tree.plot_tree(best_model.estimators_[tree_num],
                  feature_names = feature_names,
                  class_names = target_names,
                  filled=True);
    fig.savefig(r'C:\Users\Bruker\Oscar\NTNU\individual_tree' +
               str(tree_num) + r'.png')

#No maximum depth
model_nomaxdepth = RandomForestRegressor(max_depth=None, n_estimators=50)
model_nomaxdepth.fit(X_train, y_train)
print(model_nomaxdepth.estimators_)
estimator_nonlimited = model_nomaxdepth.estimators_[1]
export_graphviz(estimator_nonlimited,
                 out_file=r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\total_tree.dot',
                 feature_names = feature_names,
                 class_names = target_names,
                 rounded = True, proportion = False, precision = 2, filled=True)

#baseline_error = regression_results(test[''], test[''])
#print('baseline_error:{0}'.format(baseline_error))

train_len = len(rmse_LR_train)
test_len = len(rmse_LR_test)

plot_rmse(rmse_lstm, rmse_LR, rmse_RF, train_len, all=all_features,
          top=top_features, sel=sel_features)

#Reshape array of predicted values for the various models
y_pred_LR = pred_array_LR_test.reshape(-1,1)
y_pred_RF = pred_array_RF_test.reshape(-1,1)
y_pred_lstm = y_pred_lstm_test.reshape(-1,1)

#Inverse Transform Data to get Forecasting Values
y_real_lstm = unprocessed_data['price'].iloc[-len(y_test)-1:-1].values.reshape(-1,1)
y_real = unprocessed_data['price'].iloc[-len(y_test)-1:-1].values.reshape(-1,1)
last_obs_lstm = unprocessed_data['price'].iloc[-len(y_test)-2]
last_obs = unprocessed_data['price'].iloc[-len(y_test)-2]

y_forecast_LR = DP.inverse_transform(y_pred_LR, y_real, last_obs, scaler)
y_forecast_RF = DP.inverse_transform(y_pred_RF, y_real, last_obs, scaler)
y_forecast_lstm = DP.inverse_transform(y_pred_lstm, y_real_lstm, last_obs_lstm, scaler)

```

---

```

#Add all RMSE values to dataframe
rmse_df = pd.DataFrame({'rmse LR': rmse_LR, 'rmse RF': rmse_RF, 'rmse LSTM': rmse_lstm})

#Evaluate and Plot Results
evaluate_results(y_real, y_real, y_real_lstm, y_forecast_LR, y_forecast_RF,
                y_forecast_lstm, all=all_features, top=top_features, sel=sel_features)

#CATBOOST MODEL - Requires too much memory!
if cboost == True:
    n_features_cboost = 2
    top_features_cboost = FIS.main(lag, unprocessed_data, n_features)

    train_data_cboost = pd.DataFrame()
    test_data_cboost = pd.DataFrame()
    for item in top_features:
        train_data_cboost[item] = train_raw_data[item]
        test_data_cboost[item] = test_raw_data[item]

    X_train = train_data_cboost.loc[:, train_data.columns != "price(t)"]
    y_train = train_raw_data["price(t)"]
    X_test = test_data_cboost.loc[:, test_data.columns != "price(t)"]
    y_test = test_raw_data["price(t)"]

    model = CatBoostRegressor(
        task_type = 'GPU',
        loss_function = 'MAE',
        eval_metric = 'RMSE'#,
        #depth = ,
        #iterations = ,
        #learning_rate =
    )
    parameters = {'depth':[3,1,2,6,4,5,7,8,9,10],
                  'iterations':[250,100,500,1000],
                  'learning_rate':[0.03,0.001,0.01,0.1,0.2,0.3],
                  }

    grid = GridSearchCV(estimator=model, param_grid=parameters, cv=2, n_jobs=-1)
    grid.fit(X_train, y_train, verbose=100)

    print("\n=====")
    print(" Results from Grid Search")
    print("=====")
    print("\n The best estimator across ALL searched params:\n", grid.best_estimator_)
    print("\n The best score across ALL searched params:\n", grid.best_score_)
    print("\n The best parameters across ALL searched params:\n", grid.best_params_)
    print("\n=====")
    #Best:
    # {'depth': , 'iterations': , 'learning_rate':}

#PERSISTENCE MODEL
if persistence_model == True:
    data = DP.supervised_learning(lag, unprocessed_data, diff=False)
    persistence_df = data[["price(t)", "price(t-1)"]]
    X = persistence_df.values
    train, test = X[:train_size+1], X[train_size+1:-1]
    y_real, y_pred_persistence = test[:, 0], test[:, 1]

```

---

---

```

#Reshape array of predicted values for the persistence model
y_pred_persistence = y_pred_persistence.reshape(-1,1)
y_real = y_real.reshape(-1,1)

#Evaluate results
rmse_persistence = np.sqrt(mean_squared_error(y_real, y_pred_persistence))
mae_persistence = mean_absolute_error(y_real, y_pred_persistence)
mape_persistence = np.mean(np.abs((y_real - y_pred_persistence) / y_real))*100

print("=====")
print(f"Test scores for Persistence Model")
print('\nTest RMSE: ', rmse_persistence)
print('\nTest MAE: ', mae_persistence)
print('\nTest MAPE: ', mape_persistence)
print("=====")

forecast_persistence = pd.DataFrame(data=y_pred_persistence, columns=['Pred Persistence'])
forecast_persistence['Real'] = pd.DataFrame(y_real)

print(forecast_persistence)

fig, ax = plt.subplots(figsize=(15,10))
plt.plot(forecast_persistence['Pred Persistence'], linestyle='--', marker='o',
        label='Persistence')
plt.plot(forecast_persistence['Real'], marker='o', label='Real')
plt.legend()
plt.ylabel('Spot Rate [$/day]')
plt.xlabel('Week in 2019')
ax.set_xticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
ax.set_xticklabels(['42', '43', '44', '45', '46', '47', '48', '49', '50', '51'])
plt.savefig(r'C:\Users\Bruker\Oscar\NTNU\5. Klasse\Master\Persistence_model.png',
        bbox_inches='tight')

end = time.time()
end_readable = time.ctime(end)
print(f"\nCode ended at: {end_readable}")
print(f"Runtime of the program is {end - start}")

```

