Mohamed Hadi
Stefan Quvald Jacob

Predictions on solar power plant generation with machine learning techniques (PRESAV)

Bachelor's thesis in Electrical Engineering
Supervisor: Jayaprakash Rajasekharan & Berhane Darsene Dimd
Co-supervisor: Alexis Sevault
June 2022

Mohamed Hadi
Stefan Quvald Jacob

Predictions on solar power plant generation with machine learning techniques (PRESAV)

Bachelor's thesis in Electrical Engineering
Supervisor: Jayaprakash Rajasekharan & Berhane Darsene Dimd
Co-supervisor: Alexis Sevault
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

Sammendrag av Bacheloroppgaven

Tittel:	**Prediksjoner på solcelleanleggets kraftproduksjon med maskinlæring teknikker (PRESAV)**
Dato:	June 2022
Deltakere:	Mohamed Hadi Stefan Quvald Jacob
Veiledere:	Jayaprakash Rajasekharan Berhane Darsene Dimd
Oppdragsgiver:	SINTEF
Kontaktperson:	Alexis Sevault, alexis.sevault@sintef.no
Nøkkelord:	Norway, Norsk
Antall sider:	57
Antall vedlegg:	4
Tilgjengelighet:	Åpen

Sammendrag:

Energiproduksjon har en betydelig innvirkning på menneskers liv, og forskere har forsøkt å predikere været for å forbedre kraftstabiliteten, redusere energitap og øke økonomisk gevinster. Med tanke på dagens klimautfordringer og bekymringer, undersøker forskere fra SINTEF og NTNU potensialet på å koble flere rene energikilder sammen i et nullutslippsbygg laboratorium lokalisert i Trondheim.

Denne bacheloroppgaven undersøker et sett med maskinlæringalgoritmer for prediksjon av mengden fotovoltaisk energi som genereres av ZEB-laboratoriet, og hensikten er å sammenkoble den prediktive dataen i en styringsstrategi som vil operere sammen med andre produksjonssystemer i bygningen. Prosjektteamet vil fokusere på å bruke maskinlæringsmetoder på tidligere data på solcelleproduksjon fra anlegget og værdata samlet inn fra SINTEFs testcelle for å assistere ZEB-bygningen med å bestemme den beste strategien for redusert strømforbruk og klimagassutslipp samtidig øke bruken av lokalt produsert energi. Konseptene som presenteres i denne rapporten er basert på tidligere forskningslitteratur, vitenskapelige artikler og arbeid gjort av andre vitenskapsmenn.

Summary of Graduate Project

Title:	**Predictions on solar power plant generation with machine learning techniques (PRESAV)**
Date:	June 2022
Authors:	Mohamed Hadi Stefan Quvald Jacob
Supervisor:	Jayaprakash Rajasekharan Berhane Darsene Dimd
Employer:	SINTEF
Contact Person:	Alexis Sevault, alexis.sevault@sintef.no
Keywords:	Thesis, Latex, Template, IMT
Pages:	57
Attachments:	4
Availability:	Open

Abstract:

Energy production has a significant impact on human life, and scientists have attempted to predict weather in order to improve power stability, decrease the energy waste, and raise economic wealth. Considering the current climate challenges and concerns, SINTEF and NTNU researchers are examining the potential of merging several clean energy production sources in a zero-emission building laboratory (ZEB-lab) in Trondheim.

This bachelor's thesis investigates a set of supervised machine learning algorithms for predicting the amount of photovoltaic energy generated by the ZEB-lab, with the goal of integrating the predicted data into a control system that works in tandem with the building's other power generation systems. The project team will focus on using machine learning approaches on historical photovoltaic production from the plant and weather data collected from SINTEF's test cell to assist the ZEB-building in determining the best strategy to reduce power consumption and greenhouse gas emissions while increasing the usage of energy produced locally. The concepts presented in this report are based on past research literature, scientific papers, and other scholars' work.

Preface

This thesis is submitted as the final project from the course *Bachelor Thesis Electrical Power Engineering* (IELET2910), which accounts for 20 credits, and is a final assessment of the degree in Bachelor of Science in Electrical Engineering, Faculty of Information Technology and Electrical Engineering, Department of Electrical Power Engineering at the Norwegian University of Science and Technology (NTNU). The bachelor thesis has been carried out during the spring of 2022, where the scope of the thesis has been 20 weeks. The report is written and prepared by two electrical engineering students with the same field of study.

This bachelor thesis aims to study the concept of machine learning and further implement it by using weather information to make PV output power prediction model, using supervised machine learning algorithms. This project is an external project of SINTEF Energy in Trondheim. Working with this project has been a tremendous and rewarding experience. We have learned a lot about machine learning and how powerful of a tool that is. Additionally, our knowledge in Python programming has improved significantly. But most importantly, our skills in communication, planning and teamwork were improving and have been a major importance for the implementation of this project.

The group want to express our sincerest gratitude towards to our supervisors, Berhane Darsene Dimd and Jayaprakash Rajesekharan from NTNU for great support, guidance and motivation throughout the process of writing this project. Further, the group wants to extend the gratitude towards Alexis Sevault from SINTEF Energy, for providing the assignment and giving insightful information along the way.

Trondheim, 08.06.2022

[illegible] [illegible]

Mohamed Hadi

Stefan Quvald Jacob

Contents

Preface . . . iii
Contents . . . iv
List of Figures . . . vi
List of Tables . . . viii
1 Introduction . . . 1
1.1 Background . . . 1
1.2 Motivation . . . 1
1.3 Objectives . . . 2
1.4 Limitations . . . 2
Nomenclature . . . 3
2 Theory . . . 4
2.1 Integration of PRESAV in ZEB-laboratory . . . 4
2.1.1 PRESAV components . . . 5
2.2 PV Power Generation . . . 7
2.2.1 Solar irradiation instruments . . . 9
2.3 Time Series Analysis . . . 11
2.3.1 Elements of time series . . . 11
2.4 Algorithm performance . . . 13
2.4.1 Prediction score and metrics . . . 13
2.5 Algorithm selection . . . 15
2.5.1 Random Forest Regression . . . 15
2.5.2 XGBoost . . . 17
2.5.3 Support Vector Regression . . . 18
2.5.4 Multiple Linear Regression . . . 20
3 Pre-Processing . . . 22
3.1 Pre-Processing . . . 22
3.1.1 Combining data sets . . . 22
3.1.2 Statistics . . . 23
3.2 Visualization . . . 24
3.3 Seasonal patterns recognition . . . 27
3.4 Feature selection . . . 29
3.4.1 Pearsons correlations test . . . 29
4 Methodology Part 2 . . . 31
4.1 Machine learning aspects . . . 31
4.1.1 Train-Test split . . . 31

4.1.2 Cross-validation/K-Fold 32
4.1.3 StandardScaler 32
4.2 Hyper parameters 33
4.2.1 Hyper parameters for RF 33
4.2.2 Hyperparameters for XGboost 33
4.2.3 Hyperparameters for SVR 33
4.3 Web Application 34
4.3.1 Streamlit 34
4.3.2 Restrictions 34
4.4 Procedure 35
5 Results 36
5.1 Training/Test-Score 36
5.2 Model evaluation 37
5.2.1 Comparisons between cases 37
5.2.2 Differences between actual and predicted 47
5.3 Streamlit 49
6 Discussion 50
6.1 Impacts on PV predictions 50
6.1.1 Weather influence 50
6.2 Constraints 50
6.2.1 Area of focus 50
6.2.2 Level of knowledge 50
6.3 Algorithms analysis 51
6.3.1 Pre-processing models 51
6.3.2 Ratings of models 51
6.4 Application 51
6.5 Future work 52
7 Conclusion 53
7.1 Optimization of the PV forecast and contribution in the energy sector 53
7.2 Evaluation of the favored model 53
Bibliography 54
A Appendices 58
A.1 Libraries 58
A.2 Packages 58
A.3 Functions 60
B Gantt 63
C Meeting Logs 64
C.1 Temporal record of meetings 64
D Python codes 67
D.1 Algorithms 67

List of Figures

1 *ZEB-lab located in Trondheim* 4
2 *Representation of the PRESAV structure* 5
3 *Heating pump used in ZEB* 6
4 *Worldwide PV growth from 1992 to 2018* 8
5 *Solar radiation components* 9
6 *The schematic diagram(left) and photograph(right) of pyranometer (Solar Instruments/atmospheric Science Instruments n.d.).* 9
7 *The global irradiance includes direct sunlight and diffuse sunlight* 10
8 *Illustration of an increasing trend, where the blue curve is the graph and black one is the trend* 12
9 *Illustration of repeating cycles, that illustrates a seasonality pattern* 12
10 *Illustration of randomly generated spikes* 12
11 *How the decision tree splits operates* 15
12 *XGBoost plot* 18
13 *Visualization of Support Vector Regression in 2-D* 19
14 *A visual representation of MLR* 21
15 *Energy production of each facades in* KWh 22
16 *Measurements on weather parameters* 22
17 *Combined data sets* 23
18 *Heatmap on missing values* 24
19 *Total production in KWh of each facade* 26
20 *Monthly and weekly PV production* 27
21 *The daily mean and max PV production* 27
22 *Illustration of the different observed decomposition's* 28
23 *Total_production re-sampled daily.* 28
24 *Plots of trend & seasonality* 28
25 *Plots of residuals & observed* 29
26 *How different correlations occurs* 29
27 *Heatmap on correlated values* 30
28 *How the approach on the methodology part 2 section will occur* 31
29 *5-Fold Cross-Validation* 32
30 *Information section from application* 34
31 *RF & XGBoost* 37
32 *SVR & MLR* 37

33 *RF & XGBoost* 38
34 *SVR* 38
35 *RF & XGBoost* 39
36 *SVR & MLR* 39
37 *RF & XGBoost* 40
38 *SVR & MLR* 40
39 *RF & XGBoost* 41
40 *SVR & MLR* 41
41 *RF & XGBoost* 42
42 *SVR & MLR* 42
43 *RF & XGBoost* 43
44 *SVR & MLR* 43
45 *RF & XGBoost* 44
46 *SVR & MLR* 44
47 *RF & XGBoost* 45
48 *SVR & MLR* 45
49 *RF & XGBoost* 46
50 *SVR & MLR* 46
51 *Training/testing ratios on Streamlit* 49
52 *After editing ratios* 49

List of Tables

1 Solar panels datasheet 5

2 Summary of measured data 23

3 Training and test scores 36
4 Performance comparison for Case 1 37
5 Performance comparison for Case 2 38
6 Performance comparison for Case 3 39
7 Performance comparison for Case 4 40
8 Performance comparison for Case 5 41
9 Performance comparison for Case 6 42
10 Performance comparison for Case 7 43
11 Performance comparison for Case 8 44
12 Performance comparison for Case 9 45
13 Performance comparison for Case 10 46
14 Average RMSE & R^2 scores 46
15 Performance of a forecast model based on RF 47
16 Performance of a forecast model based on XG 47
17 Performance of a forecast model based on SVR 48
18 Performance of a forecast model based on MLR 48

1 Introduction

1.1 Background

SINTEF Energy is working on a project ***PRESAV*** with the intention of finding the best predictive controlling system for energy storage by collecting information from the electricity market, district heating, weather forecasting and local heat demand in buildings with installed PV and active heat storage. This project is divided in three activities. Our bachelor thesis focuses on the data science and feedback strategies part and is written solely on this particular assignment.

Overview of this thesis

This paper starts with a brief introduction of the main purpose with the objectives and motivation for the report and then goes into the core theories behind the PRESAV project and the time series analysis. Then we'll go through the methodology behind the algorithms before presenting the proposed solutions, and finally move on to conclusions and discussions, while adding some tips and additional information for future groups exploring the problem.

1.2 Motivation

The global demand for energy is really large and it presents major challenges in terms of its growing rhythms. We need an energy portfolio that meets the criteria of an efficient, affordable but also clean resource. Billions of people in countries with a GDP of less than 25.000$ lack access to renewable energy source, for example electricity [1]. Lack of access to electricity forces people to live without refrigeration of food, washing machine, dishwasher or central heating etc. The energy challenge in the world is two-folded. This means that the countries in poor conditions cannot afford sufficient energy. There are two sides of a problem that we need to solve. We lack energy alternatives to substitute energy fossil fuels that are both cheap and sustainable. With the revolution of thinking of sustainability and starting to form a path to clean and affordable energy, solar energy came to light. This is widely considered as the most promising candidate for sustainable power generation until this date.

Solar energy output for one hour equals the earth's energy demand for an entire year. Even though the solar power has untapped potential, the solar power generation is highly intermittent [2]. The solar power is dependent on factors such as the weather, meteorological, and temporal parameters, and many companies try to integrate PV technology systems into the smart grid by forecasting the solar power generated by a particular PV module, in a certain location over a given time period. It is very important that the solutions is approximate, because unexpected fluctuations in the solar power system could affect the health of the solar grid, and in worst case interact with the quality of life of energy consumers.

Physical inputs such as temperature, humidity, solar irradiance and wind speed, rely on the vast majority of current forecasting algorithms. However, such systems tend to be not precise, as recent research has prompted to implement machine learning algorithms to supplement the use of such physical inputs in forecasting power generation. With given data, we build a model-based system in the pursuit if predicting the solar power output. Within the framework of improving the forecasting accuracy of the power produced, adaptive machine learning algorithms capture the system behaviour, without even knowing the parameters, only the constructed relationship between the input and the output [3]. The scope of this work is presentation the methodology for deriving more precisely Next-day power predictions for PV plants using machine learning approaches optimised for the selection of input features. Included methods Support Vector Regression(SVR), Random Forest, Multiple Regression and XGboost. The base methodology followed was to train the models with acquired data sets and to construct relationships between the input and output features which in this particular case is the power prediction for the new time.

1.3 Objectives

This report investigates on providing answers to following questions:

- Discussion of PV power generation, identifying the challenges of using such kind of energy source, and discussion of the widely used methods to address these issues.
- Understanding the role of PV output power forecasting to make PV generation an equal contributor in the energy mix.
- Design of a forecast model based on machine learning algorithms
- Performance evaluation of the various predictive models for a PV plant located at ZEB Lab, Trondheim, Norway.

1.4 Limitations

Since the main focus is forecasting and prediction, the engineering and determining best solutions for different low voltage couplings and combinations for PRESAV components were not included in this report. Cost analysis, environmental, energy consumption's and electricity prices benefits as a consequence of implementation of PRESAV, were outside the main focus of the thesis. The historical weather data provided from SINTEF on the ZEB-lab, were these variables: outside temperature, dew point, wind and humidity measurements, barometric pressure and solar radiation and power production from the PV's.

Nomenclature

$\bar{y}_i$	Data points actual value average
ΔT	Temperature difference
y	Actual value
$\hat{y}$	Predicted value
BIPV	Building Integrated Photo Voltaic
CSV	Comma Seperated Values
DHI	Direct Horizontal Irradiance
GHI	Global Horizontal Irradiance
GW	Gigawatt
IEA	International Energy Agency
MAE	Mean Absolute Error
MLR	Multiple Linear Regression
MSE	Mean Square Error
PRESAV	Predictive Management Strategies For Active Heat Storage In Buildings
PV	Photo Voltaic
RF	Random Forest
RMSE	Root Mean Square Error
SVR	Supporting Vector Regression
SZA	Solar Zenit Angle
XGBoost	Extreme Gradient Boosting
ZEB	Zero Emission Building laboratory

2 Theory

This section aims to provide an overview of supervised learning concepts used to predict PV power generation, time series predictions and how the various components in the ZEB lab work together in a coherent system. The amount of information included in each section depends on what the main elements in this project are.

2.1 Integration of PRESAV in ZEB-laboratory

ZEB-lab

The purpose behind ZEB-lab is that the building intent aims to be a testing and research facility and pilot project for greener building components. The aim is to inspire the ZEB-lab and take the initiative for other national and international companies. ZEB-lab will be merged with BIPV to replace some of the traditional building materials and the reason is that solar energy collected by BIPV will represent the energy production [4]. The total area of the ZEB laboratory is almost 2000 m^3 including a 4-storey building, 2 of which are used for offices and the others for educational purposes.

Figure 1: ZEB-lab with a view from southern and western facade

PRESAV

PRESAV is divided into three categories: predictive control strategies, implementation and testing in the ZEB lab, and analysis of the recorded data to provide feedback on different strategies. The main goal of PRESAV is to create forecasting strategies to control the ZEB laboratory based on characteristics such as weather forecast, local heat demand and economics in relation to district heating and electricity prices. The inputs of the predictive model will include weather forecasts and energy generated from PV as well as electricity collected from the grid, and the outputs will be combinations of active thermal storage strategies. The solar power produced in the system can be converted into thermal energy using heat pumps and then stored in PCM devices or charged using district heating. This energy can then be used to pre-heat the building if required.

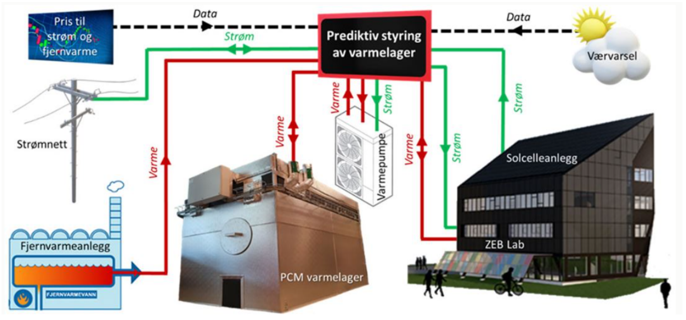


Figure 2: ***Representation of the PRESAV structure***

2.1.1 PRESAV components

BIPV

PV cells generate electricity when the solar cells are exposed to light, and it does so instantaneously. The electrical energy generated results from the semiconductor material placed between two electrodes. The active part of a solar cell is a wafer assembled with the semiconductor, and the semiconductor has three layers, negative type, positive type and the middle layer.When the energy exceeds a threshold called the bandgap, the electrodes are free to move, thus generating electrical energy [5]. PV panels are typically assembled in a formation containing multiple solar panel cells, which in turn are composed of multiple solar cells. Nowadays, the applications of PV cells are increasing rapidly and are very demanding [3]. Considered the most reliable form of energy due to its availability and accessibility, solar energy has been established as a pioneering source of electrical energy due to recent advances in solar cell conversion energy [6].

This installation is equipped with 701 solar panels, which corresponds to an area of 963.4 m2 with modules based on Mono-Si cells. The panels are released by different manufacturers and the total installed power is 281.15kW_p. Below is a table showing different areas, manufacturers and how each panel contributes to the system.

Table 1: Solar panels datasheet

Placement	Type of panels	Fasteners	Installed Power [KWp]	Area [m^2]	Number of panels
Roof	Sunpower 350	IRFTS	98	456.3	280
North facade	Sunpower 375	Baywa	11.25	53	30
South facade	Solarlab	Nvelope	22.36	144.2	132
West facade	Solarlab	Nvelope	12.365	79.6	73
East facade	Solarlab	Nvelope	24.47	156.2	144
Pergola	Sunpower 375/Solitek	Baywa	7.875 + 4.83	37.1+37	21+21

PCM storage

PCM is a material's ability to store/release energy by changing its phase. When a solid material is heated above its melting point, the material absorbs thermal energy, and when the material is cooled, it solidifies as it releases energy. This phenomenon is classified as latent heat and several materials share this property. PCM systems absorb/release energy at a near constant temperature and the use of PCM as thermal energy storage in industry is now widely explored. The property that a PCM can absorb/release energy at a constant temperature allows a massive amount of heat energy to be stored and used when needed. This property minimizes waste of energy sources for heat and energy consumption in buildings and can be used in many different modules [7].

The material used in ZEB-labs PCM is a bio-based wax containing 3 tons of the material and has a melting point of T= 37 ° C or 310.15 K. The PCM installed in the ZEB-lab has four operating modes:

- **Charge** from heat pump
- **Charge** from district heating
- **Discharge** to heat pump
- **Discharge** to heating circuit

District heating and heat pump

District heating is all about taking energy released as heat from multiple energy sources and from there connect to energy consumers through a system of highly insulated pipes. The heat is obtained from fossil fuels or even biomass, but has also used heat boiling , heat pump and solar heating systems.There have been quite a few developments of each generation of district heating throughout the years. From coal heating in 1880s to cold heating.

Cold heating distributes heating at ground temperature to minimize heat losses to the ground, in other words, it covers the expenses of a potential extensive insulation. Heat pump is one of the components in the ZEB-lab structure, and the area of usage is to be an alternative replacement for the PCM-system. The way heat pumps work is to collect heat in form of water or air from the surroundings and convert it to energy [8]. The heat pump input in this circuit, will be electricity from the grid and PV's combination and output will be heat energy. The self produced energy from PV's can be transformed through heat, with utilization of heat pumps and after that stored in PCM as required.

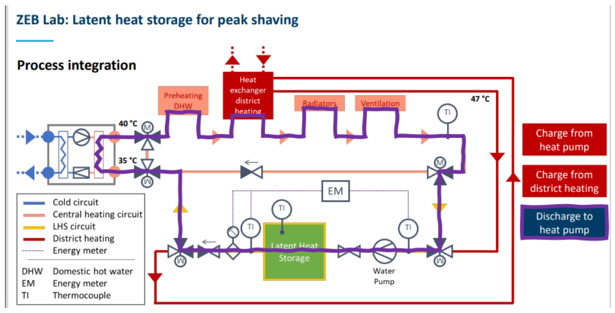


Figure 3: ***Heating pump used in ZEB***

2.2 PV Power Generation

PV growth

Solar power is described as the most elegant way of producing electricity, without moving parts, emissions or noise. Easily described, it converts abundant sunlight without practical limitations. The question has not yet been answered due to differing opinions on how much this energy converter will contribute in the future.

The role of PV cells in the future energy supply chain has been lined out. Due to its pure form of clean and affordable energy, PV technology is expected to play a big role in the renewable energy market. The increasing capacity to install PV systems and a maturing, policy-driven market have driven down the cost of PV systems. The emission of greenhouse gases has increased dramatically after the industrial revolution. Historically, the global PV market has grown by approximately 33% over the period of 1998-2002 [9]. In 2020, PV capacity increased by the same percentage.

PV growth is quite difficult to predict and involves many uncertain factors. Official agencies such as International Energy Agency(IEA) have in the past decade increased their estimates, but still fall short of projecting actual deployment in every forecast. IEA announced that the capacity in the PV production increased by 50% in 2016 with a total capacity of 74GWh [10]. The growth have been close to exponential in the gap between 1992 and 2021. Global PV electricity production reached the amount of 1000 TWh in 2021. As solar power increases competitively, the total solar capacity in the world will continue to grow rapidly. Based on IEA, its sustainable development scenario foresees to reach 4.7 Terawatts by 2050, and more than half of it will be deployed in India and China, making photovoltaic technology the largest source of electricity [11].

Photovoltaic technology has changed significantly in terms of the industry structure and its market prices. When photovoltaic systems were first recognized as a promising renewable technology, subsidy programs such as feed-in tariffs were introduced to boost and provide economic incentives for investments. European pioneers and Japan were the only ones to implement this for years and as a result, the cost of solar declined due to improvements in technology and economic scale [12]. Several national programs were instrumental in increasing PV deployment such as the *Energiewende* in Germany and China's 2011 5-year plan for energy plan. 30 countries have achieved socket parity which is an alternative energy source that generates power at a levelized cost of electricity, or LCOE, that is equal to or even lower than the price of power from the electricity grid.

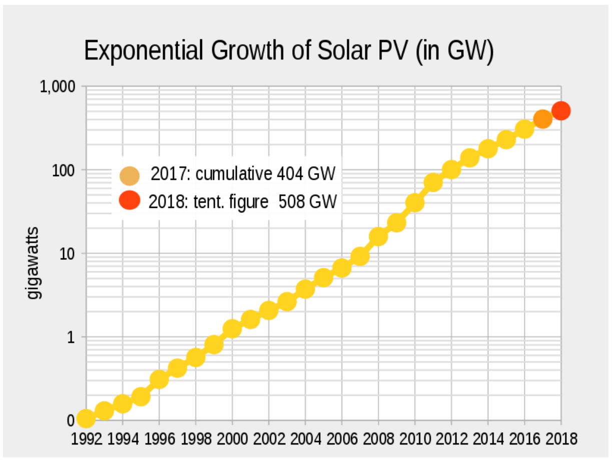


Figure 4: ***Worldwide PV growth from 1992 to 2018***

The outbreak of COVID-19 had a slight impact on the marked growth and it effected negatively and exposed the vulnerability of energy supply chains. Many solar PV developers have experienced delays in importing solar PC modules and other supplies, due to transportation restrictions imposed by several countries as an action against the virus outbreak. The current disruption is short-term until COVID-19 is brought under control, and the world will remain up to speed about the long term solar energy [13]. Many developing countries in Asia, including the Pacific have limited manufacturing capacity in the PV value chain [12]. That is why they are very dependent of importing the solar modules and other equipment. In addition to the manufacturing capacity, the solar value chain involves technical services, such as engineering, designing and maintaining, and there is serious skill gap in many of the developing countries. They are really dependent on foreign consultants or contractors to implement them in their projects. As PV technology continues to grow, it is slowly building momentum in developing countries to consider PV technology in terms of manufacturing and technical capacity.

GHI

GHI stands for Global Horizontal Irradiance, and it describes the total solar radiation on a horizontal irradiance, and is the most useful metric for predicting solar panel output. The GHI is calculated with the sum of multiple DNI(Direct Normal Irradiance), DHI(Direct Horizontal Irradiance). DNI is the solar radiation received per unit area by a surface that is always held normal to the sun's rays, that come in a straight line from direction of the sun at its current position in the sky. DHI is the amount of solar radiation received per unit area by a surface that does not arrive on a direct path from the sun, but has been scattered by molecules and particles in the atmosphere and comes equally from all directions [14]. GHI is usually meassured by a pyranometer which has 180 degree view. angle [15].

GHI formula given:

$$GHI = DHI + DNI * cos(z) \tag{2.1}$$

The variable z is the solar zenith angle and described as the angle between the rays from the sun and the vertical.Which concludes that the higher the sun is towards the sky the lower the angle SZA will be. By analyzing the function you will see that maximum amount of radiance received by a surface, is by keeping the incoming radiation perpendicular.

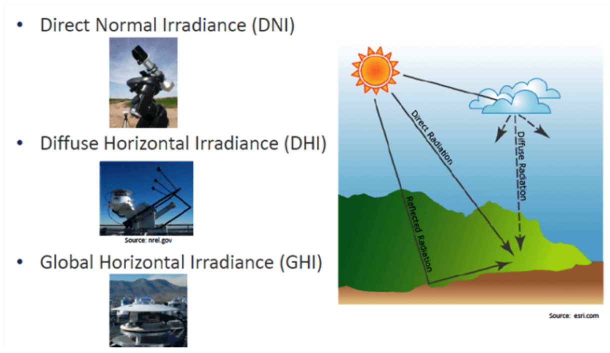

Direct Normal Irradiance (DNI)
Diffuse Horizontal Irradiance (DHI)
Source: nrel.gov
Global Horizontal Irradiance (GHI)
Direct Radiation
Diffuse Radiation
Reflected Radiation
Source: esri.com

Figure 5: ***Solar radiation components***

2.2.1 Solar irradiation instruments

Pyranometer

Pyranometer is an instrument that measures solar radiation from a hemispherical (180°) point of view and is expressed in the SI unit of irradiance W/m^2. which describes the amount of visible light and non-visible parts of the spectrum in terms of solar energy per area. The first pyranometer was by physicists Angstrom and Anders Knutsson in 1893 [16].

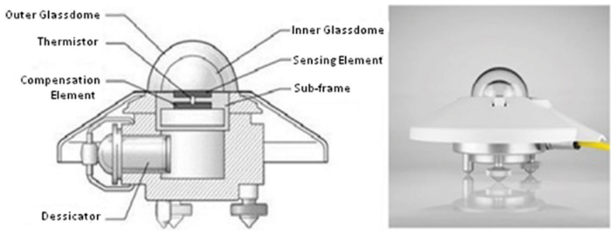


Figure 6: ***The schematic diagram(left) and photograph(right) of pyranometer (Solar Instruments/atmospheric Science Instruments n.d.).***

Pyranometers measure global radiation, specifically per unit time, incident on a surface or a given orientation and include both direct sunlight and diffuse sunlight [17], and the formula is given:

$$Eg \downarrow = E \cdot cos(\theta) + Ed \tag{2.2}$$

Where E is the maximum amount of direct sunlight(normal) and $theta$ is the angle between the surface normal and the position of sun in the sky. $Eg \downarrow$ is the denotion of solar energy per unit area.

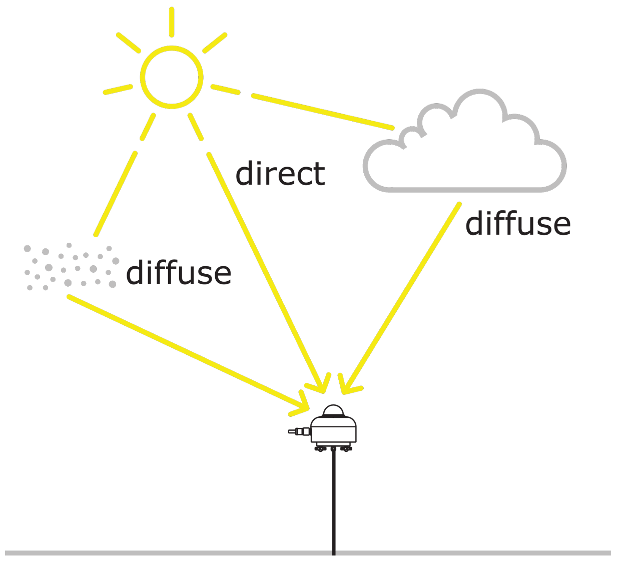


Figure 7: ***The global irradiance includes direct sunlight and diffuse sunlight***

Solar energy is the original source of most energy found on the planet. This has important implications in two main areas: weather and climate, and energy production by harvesting solar energy. Because solar irradiance is one of the main factors climate and weather studies, we measure the GHI to determine the amount of irradiance falling on the surface of a given area really depends on factors like cloud coverage, aerosol concentration, fog and smog [18]. This only applies to surface measurements, but when we measure the GHI in Earth's atmosphere it is fairly predictable.

Pyranometers are sensors that measure solar irradiance and are designed to measure the radiation flux density from a hemispherical view within a wavelength of 0.3 to 3 x 10^{-3} μm [19]. Typically a pyranometer does not require power to operate, however, due to recent technological developments including the use of digital systems that requires electronics and therefore we require a small external power. The solar radiation spectrum reaches the earth's surface extends its wavelength from 300 to 2800 nm depending on which pyranometer we use [20]. A pyranometer is built with one or two domes, a thermopile and a black absorber, and in recent times, a digital version has been used. Which means that electronics are a part of it. The glass dome works as a filter that transmits wavelength from 0.3 to 3 x 10^{-3} μm, and blocks thermal radiation with longer wavelengths from convection. Many pyranometers include a second glass dome as additional shielding to improve the equilibrium between the sensors and the inner dome, compared to single-dome pyranometers [21]. A thermopile is used to determine the dissimilarity in temperature between two surfaces and is referred to as *label active* and *reference*, representing the hot and cold sides, respectfully. In other words, a distinction is made between the sun-exposed and non-exposed area. The filtered radiation is absorbed by the black surface of the pyranometer and converted into heat. The thermopile plays an important role in measuring the temperature difference, and the potential difference formed inside the thermopile is due to the temperature gradient between the two surfaces [22]. The temperature gradient from the black surface through the thermopile to the pyranometer body is given by the formula:

$$\Delta T = R_{thermal} \cdot P_{absorption} \tag{2.3}$$

Where $P_{absorption}$ describe the heat absoption and $R_{thermal}$ is thermal resistance of the thermopile sensor. The thermal resistance depends on the geometry and its composition of the thermopile sensor. The output signal from the pyranometer is given in voltage from the thermopile or an even more convenient signal by including electronics.

2.3 Time Series Analysis

Time series analysis is applied when dealing with a sequence of data points collected over a time interval. Time series analysis illustrates how variables change over time and gives engineers insight into systematic pattern trends and how they occur, e.g. monthly, weekly or yearly. Time series are used in numerous applications such as statistics, econometrics, and finance.

2.3.1 Elements of time series

Time series analysis provides a way to describe and predict future values based on past behaviors/performance. Predicting a complex structure by analyzing the factors behind it provides advantages that can be used to improve its performance. In time series analysis, there are three critical components that can be manipulated to provide a better understanding of the time series concept:

- **Trend**
- **Seasonality**
- **Noise**

The combination of the mentioned components can form a model that allows a classification of time series problems:

$$y(t) = trend(t) + seasonality(t) + noise(t) \tag{2.4}$$

Trend

Whether the approach is to visualize data or to calculate various metrics of the dataset, trends are useful for predicting future movements. The trend patterns provide an opportunity to see if there is a rising or falling movement, hence this term results in a long-term average trend. In other words, trend is the linear increase/decrease pattern of a data set over a period of time. Below is an image that illustrates an increasing curve, hence an increasing trend:

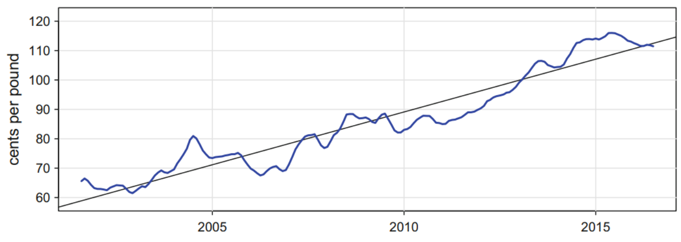


Figure 8: ***Illustration of an increasing trend, where the blue curve is the graph and black one is the trend***

Seasonality

When a curve repeats a specific fluctuation or pattern over a period of time, this is known as seasonality. Just because a cycle occurs in a data set doesn't necessarily mean the data points are seasonal. Seasonality indicates that a cycle repeats itself over and over with the same frequency. Time series problems where the seasonality is eliminated is called for "Seasonal Stationary" [23].

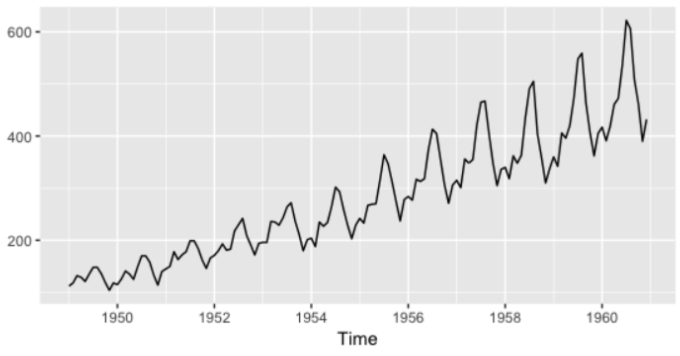


Figure 9: ***Illustration of repeating cycles, that illustrates a seasonality pattern***

Noise

Noise can be seen as small/large random spikes in a data set that stand out from the rest of the points. Deviations such as noise do not usually express the model, and occasionally a measurement error can create such noise.

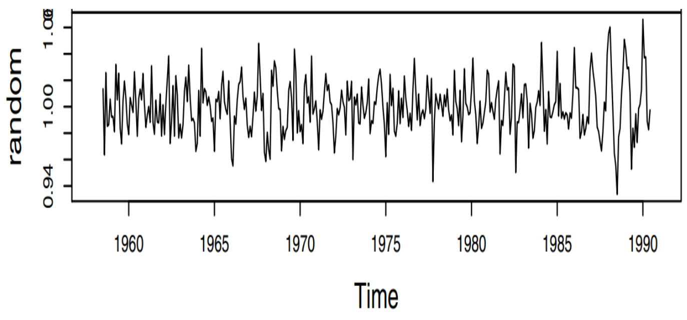


Figure 10: ***Illustration of randomly generated spikes***

2.4 Algorithm performance

The study of historical data analysis, pattern recognition, modification and prediction of an extended set of outcomes is based on the application of algorithms. To use machine learning algorithms on a prediction data set, the end game is to recognize a label output based on the input features [24]. Because the data provided has a set of functions that map inputs to an output variable based on a set of input-output pairs, the supervised learning model is implemented. In supervised learning, we have an input variable (x) and an output variable (Y), and the goal is to estimate the mapping function so effectively that you can predict the output (Y) for the data set.

$$Y = f(x) \tag{2.5}$$

Within the supervised learning process, we can divide the problems into two groups of categories, namely classification and regression. When there is an association between the independent and dependent variables and the output variable is continuous, the regression procedure is used. Continuous problems are based on the principle that the output variable has a real value or quantity, such as an integer or floats.

2.4.1 Prediction score and metrics

Since the strategy is to predict the power generation using the regression method, the use of regression metrics will play a crucial role to get an overview of the predicted score. The use and evaluation of the metric score gives an indication of how the model is performing given the input data set. A model's performance is therefore addressed with error measures, and this gives an opportunity to see how close the predictions were to their expected values.

The error can be identified as the difference between the actual and the predicted value. The ideal state of the model is that this difference in error is equal to 0, which means that the system can correctly predict all values [25]. Errors that appear in the data set can potentially be items such as missing sets of data points that affect the x to y mapping. Corresponding errors are named "irreducible errors", as they can not be minimized or removed.

$$Y = f(x) + error \tag{2.6}$$

$$Error = y - \hat{y} \tag{2.7}$$

There are many different and useful metrics to define performances, but this thesis will only focus on four of them; Mean Absolute Error(MAE), Mean Square Error(MSE), Root Mean Squared Error(RMSE) and coefficient of determination(R^2) .

Mean Absolute Error

MAE is calculated as the sum of absolute differences between the actual value(y) and the predicted value ($\hat{y}$). MAE does not account for the direction, hence the values can alternate between positive and negative but with the absolute function the output will always remain positive.

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n} \tag{2.8}$$

Mean Squared Error

MSE values are measurements regarding the condition of an estimator, and this metric is a crucial loss function for fitting and optimizing the algorithm. The mean of the squared differences between the predicted and estimated values in the data set, is the MSE. This demonstrates that if a data set contains large dissimilarity in predicted and estimated values, the larger will the squared error get, hence a MSE value closer to 0 is ideal. This phenomena is called for the "punish effect", and usually it punish data that carries large errors as a loss function.

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.9}$$

Root Mean Squared Error

RMSE is an addition to the MSE, but here the square root of MSE is implemented. What distinguish MSE from the RMSE metric is that with MSE the unit of the predicted value is squared, so for instance the power production in MSE will be in $(KWh)^2$, while in RMSE the unit will remain unchanged (KWh). A flawless model will indicate that the RMSE score is 0, but that is more a theoretical expression of the target variable [26].

$$RMSE = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{2.10}$$

$$RMSE = \sqrt{MSE} \tag{2.11}$$

R-Squared Score

R^2 score is a regression metrics such as the the others mentioned above, and this metric has a interval between 0 and 1 from no suited to excellently suited. This metric is implemented when the developer is interested in seeing how good the model is fitted, hence it is important to know that this measurement does not mean that the actual quality of the model is excellent [27]. This statistical measurement is also classified as"*coefficient of determination*", and the formula is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y}_i)^2} \tag{2.12}$$

2.5 Algorithm selection

Applying machine learning techniques to solve a problem, we usually evaluate and compare different types of algorithms to find the best suited solution [28]. Regression problems can be solved with many different types of models and algorithms, and each of them respectively has both benefits and disadvantages. Given the statistical noise and missing elements that can occur in a data sample, each and every algorithms has restrictions. A preferred and robust model is one which performs competent, disregarding of the input variables. One approach to clarify which algorithm suits to the specified data set, is to predict the performance of each algorithm separately, and then choose the most qualified based on their predicted performance calculated [29]. This approach offers a beneficial factor, since the exchange of algorithms can be completed without the use of training algorithm each time a new model is implemented.

2.5.1 Random Forest Regression

Random Forest [RF] is a supervised machine learning algorithm that, when used properly, can provide accurate, effective and reasonable forecasts and prediction outputs. The function of a RF employs many decision trees and each decision tree splits a new set of random features. In other words, a decision tree splits the data set repeatedly into binary choices: *decision nodes* or *leaf nodes*.

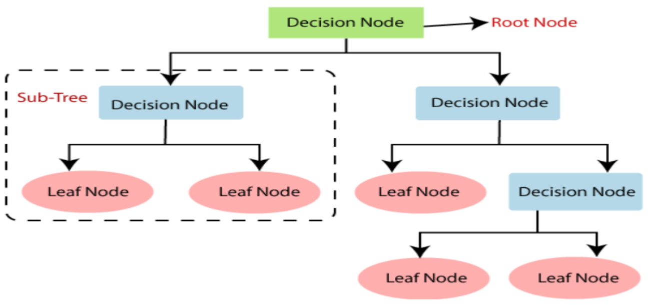


Figure 11: ***How the decision tree splits operates***

The data points find the best split and continue down the path using the decision nodes, until they arrive at a leaf node. The random forest algorithm determines the outcome by taking the average or mean of the output from different trees. This means that as the number of trees increases, the predictions become accurate.

Decision trees are the bases of the random forest algorithm, and each decision tree forms a tree-like structure as a supporting-technique and consists of three components: decision node, leaf node and finally a root node as mentioned before.The concept of any decision tree is to divide the dataset into branches and then split into other branches and it continues until a leaf node is reached. The decision trees have shown excellent performance in settings where the number of variables is much larger than the number of observations [30]. This supervised learning techinque works by sampling data sets using three main hyperparameters, which need to be clear before we take it into training. We use the RF algorithm to solve regression or classification problems [31]. In the random forest algorithm, the data sample drawn from a training set, with each tree in the ensemble with replacement, also called *bootstrap sample*.Bootstrap sample is used when the goal is to reduce the variance of each decision tree, by sampling random with replacement from the available training data. *Bagging* is also included to bring more diversity into the dataset, and helps reducing the correlation between decision trees [32]. Everything on the task, if a regression task is set then each decision tree will be averaged, and if a classification task is set then it finds the most common variable in each category and gives the predicted class.The low correlation between the models is important to build a portfolio where the sum of all parts can produce ensemble predictions that are more accurate compared to individual predictions. Also, the great thing about the random forest algorithm is that it can be applied to a variety of prediction problems when there are only a few parameters to tune.

By looking at the mathematical aspect of it, given a training set with $X = x_1, ..., x_n$ and $Y = y_1, ..., y_n$ as responses, by bagging it repeatedly, and train it in a classification or a regression tree f_b on for example X_b and Y_b. After training, we use predictions, the unseen samples x' by taking the average of each individual regression trees on x', given the formula:

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x') \tag{2.13}$$

The majority vote is to decide the classification. By using bootstrapping, we also find that the model's performance of this model will decrease its variance without increasing the bias. This means that the predictions of an average of multiple uncorrelated trees are not very sensitive to noise compared to a single tree in its training set. We use the standard deviation of the prediction for each individual regression tree, given the formula:

$$\sigma = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}(f_b(x') - \hat{f})^2} \tag{2.14}$$

The variable B stands for the number of total trees and is considered a free parameter. To find an optimal B-value, we either use cross-validation(CV), a resampling method that uses different parts of the data to test and further train a specific model with different iterations. The optimal number of B can also be found by observing the mean prediction error for each training sample x_i using only samples without x_i in their bootstrap sample [33]. This method of measuring error estimation in a ML is called Out-of-bag error(OOB).

If one or more features are very strong in terms of predicting for the output variable, these features will be selected in many of the B trees, causing them to correlate [34]. Considering a classification problem with p features, $\sqrt{p}$ features are used in each split [35], and $p/3$ is used for regression problems. The best values for these parameters depend highly on the problem at hand and they are treated as tuning parameters [35].

2.5.2 XGBoost

Extreme Boosting, abbreviated XGBoost, is a gradient boosting decision tree framework that provides parallel tree boosting and a scalable supervised algorithm used in machine learning for supervised regression classification problems.The concept of "gradient boosting" combines single weak model with another single weak model, to add a collectively stronger model [36]. This XGBoost features:

- ***Regularized Learning:*** it will allow us to select models that employ simple yet predictive functions. Parameters such L2 or λ, which are constants in the gain and predictions, that help to smooth out the final touch [37].
- ***Gradient Tree Boosting:*** This type of tree model is not able to be optimized using traditional optimization methods Euclidean space, therefore the model is trained with an additive manner [36].
- ***Shrinkage and Column Sub-sampling:*** Shrinkage is commonly used to reduce the unstable regression coefficients. It is added by the factor η after each step of tree boosting, reducing the impact of each three and leaves capacity for the future to further improve the model [38]. Coloumn sub-sampling prevents over-fitting by adding a random portion of the training data that is already used to fit a base-learner.

By looking at the mathematical aspect, the first prediction requirement is set to minimize the errors in the data set. First, the prediction is simply the mean value of the observations, and is used for future predictions. The loss function is measuring the difference between the observed and predicted value scale value. In XGBoost the loss function is given:

$$\ell_i(y_i, \hat{y}_i) = \frac{1}{2} * (y_i - \hat{y})^2 \tag{2.15}$$

i stands for each index in terms of row, $\hat{y}_i$ stands for the predicted value at i and y_i is variable for the observed value at i. The next iteration creates a regression to predict the *residuals*, which means the difference between the observed and the estimated value error from the start. The reason for this, is to split the data into groups by separating data based on feature values. Same principle as decision trees, but we are now talking about the regression trees.

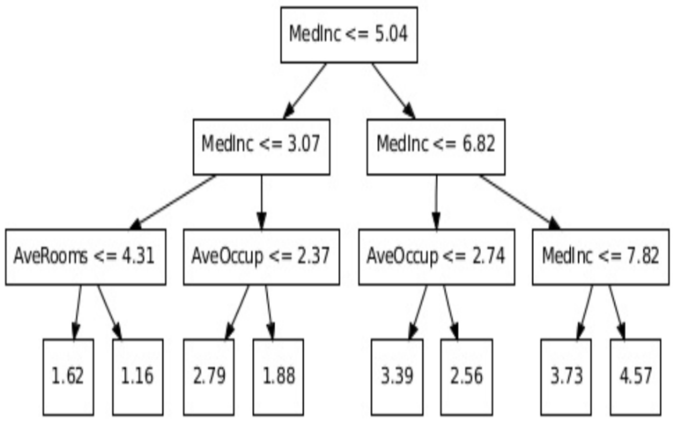


Figure 12: ***XGBoost plot***

For the next iteration so-called *learners*, a new regression tree is built by making the error smaller and then making the prediction smaller by increment, illustrated by the objective function, which combines the loss function but also the regression tree functions, with in other words, the penalty term for the model's complex. This training proceeds with iteration, predicting the residuals or errors of each previous tree, which are then combined with the previous tree to make a final prediction *gradient boosting*.

$$Obj = -\sum_{j} \frac{G_j^2}{H_j + \lambda} + 3\gamma \tag{2.16}$$

This is the function we use to measure the performance of the trees. The weight of the residuals in the leaf is only the mean value if $\lambda = 0$. XGBoost does not build all subtrees at once, but add a subtree at each iteration. Fitting is done by going step by step, and the prediction result is given with the formula:

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \tag{2.17}$$

ω is the function of a subtree and is added to to the objective function. The complexion of the tree is determined by regularisation terms as mentioned before and is defined:

$$\Omega(f_k) = \gamma\Gamma + \frac{1}{2}\lambda \sum_{j=1}^{T} ||\omega_j||^2 \tag{2.18}$$

γ and Γ are the variables for the penalty coefficients of the regularisation. T is the number of the leaf nodes of the k_{th} tree. ω_j is the weight of the j_{th} leaf node of the k_{th} tree.

2.5.3 Support Vector Regression

Support Vector Regression is a regression implementation of the Support Vector Machine classification model, that is a supervised learning method, providing the flexibility to define how much error is allowed. This is measured by variable ϵ which defines the margin of tolerance. SVR is a regressor and we use this algorithm to predict continue values, instead of discrete outputs.

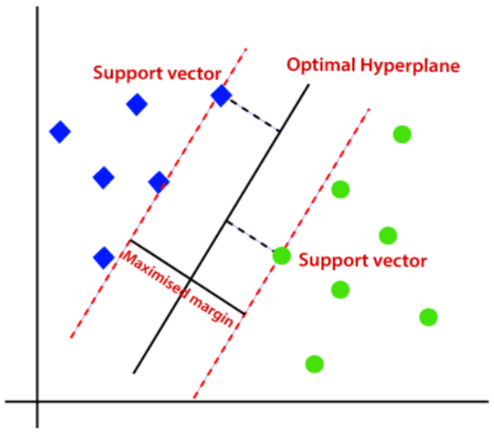


Figure 13: ***Visualization of Support Vector Regression in 2-D***

The concept of SVR can be described with a geometrical perspective, for example:using one-dimensional graph with boundary lines. These lines define the distance ϵ, in other words the margin. In the middle of the boundary lines, we have so called Hyper Plane which can be described as a separation line between the data classes, and the main goal is to try minimize the error rate. We get an idea of the data points that are closest to the hyperplane(if it is 3-Dimensional) or the support vectors are within the boundary, in other words, those data points with the least error rate [39]. Since SVR models can be performed at a higher dimension(3-Dimensional), we need a function that should map all the data points, and this function is called Kernel. Examples of kernel functions: Radial Basis Function(RBF), Sigmoidal, Polynomial and Gaussian Kernel [40]. The supporting vector is used to define the hyperplane and stays close to the boundary lines. With the data points forming a curve, the SVR regression algorithm uses the curve to find the best possible match between the vector and position of the curve. Here come the Support Vectors in, by trying to determine the nearest match between the function and the data points and further used as predictions. The kernel trick is therefore a tool to use when the data is not linearly separable. The algorithm does not operate in a higher dimensional feature space, but uses similarity measures without doing the transformation.

Real data is often non-linear in terms of how linear and how separable it is. Error play in an important role Support Vector Regression and parameters such as C and Gamma are really important a to understand how we separate the data points and then finding a balance between the variance and bias [41]. The Gamma parameter γ controls the distance of a single data point, which means that low values of gamma indicates a large similarity radius, which means that more data points are joined together. The decision boundary will therefore be more curvature. If there are high values of gamma, that means that each data points are closer to each other and will be *over-fitting*, and a small noise may cause data points to fall out of a determined class [42].

A boundary might be placed too far for each class and will further contribute to some misclassified exceptions. This is why we use the C hyperparameter. C parameter control and tell us that we need to avoid misclassifying each training example. C is a hyperparameter that adds a penalty for each misclassified data point. When the C value is low, a large margin for a decision boundary is set, which is good [43]. In our project we use the RBF kernel, and therefore we need both C and gamma parameter is both optimized simultaneously. If gamma parameter comes of as high, the effect of C comes negligible. If it comes of as low, the the function will have a large variance and lower bias, hence points must be close to each other in order to be in the same class. Formula for RBF is given:

$$K(x, x_i) = \exp(-\gamma * \sum (x - x_i^2) \tag{2.19}$$

Here is x defined as input variable and x_i defined as supporting vector. A good value for γ will be around 0.1 to 1, the radial kernal is local, and can create space with complex regions.

2.5.4 Multiple Linear Regression

Multiple Linear Regression(MLR) regression model used to describe relationships between several variables and a response variable output. The formula for a multiple linear regression is given:

$$y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \epsilon \tag{2.20}$$

y is the value for prediction or estimation in the $n_t h$ - observation. n stands for the total number of predictions. β_0, *intercept* represents the change in mean response y when all the other predictors are constant. β_n is the regression coefficient in n-th predictor ϵ stands for the residual error, and defines the difference between the model's prediction and the and the actual y value.

This technique allow us to determine the model's variation and how each relative variable contribute to the total variance [44]. For multiple linear regression we use matrix representation, because it expresses multiple linear regression models simultaneously and it becomes easier to compute each model parameters. Matrix function is given:

$$y = \beta * x + \epsilon \tag{2.21}$$

The β coefficient indicates that it is linked to each independent variable. In a linear regression model we minimize the sum of squared errors to find all of the β. This by actually finding the error

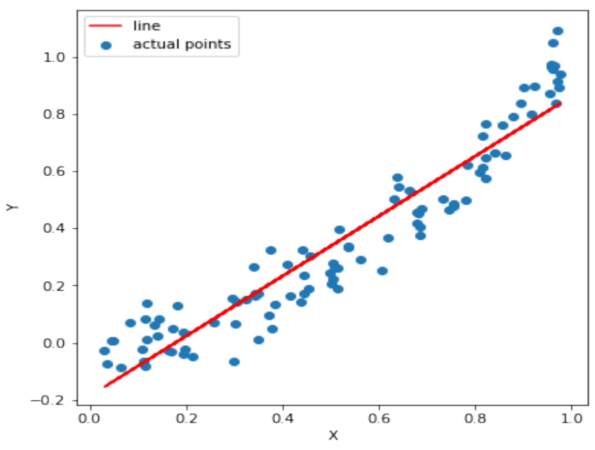


Figure 14: ***A visual representation of MLR***

MLR have more parameters than a simple linear regression model, adding more terms will definitely improve the fit for the data. But more terms will also raise the odds of over-fitting the model which will further lead to fault results while trying to predict values. By calculating the standard error of each coefficient or by regularization we can reduce the number of parameters. When calculating the standard error, we will automatically see which variables are the valuable in terms of its contribution to the model. By applying regularization, we implement an error by adding to increase more terms in the model, and help us find a balance of removing terms reduce the downside of extra terms, and still include enough of the important terms to provide a good fit. To estimate the β values, we need the sum minimized of squared errors for the data sample.

3 Pre-Processing

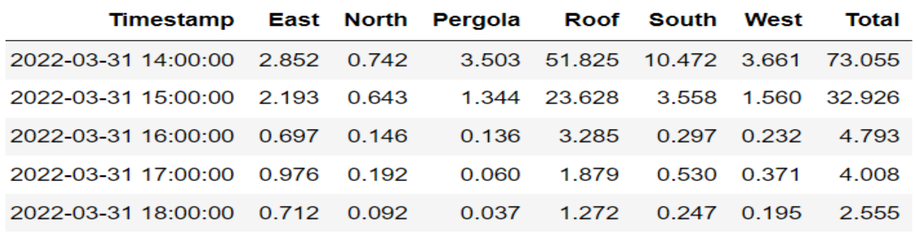
Preprocessing methods and procedures are covered in this chapter. When dealing with machine learning algorithms, preprocessing is the first and most important stage. The chapter is separated into sections, where we first go over how to combine the datasets, clean them up, visualize them, and finally look for seasonal trends.

3.1 Pre-Processing

3.1.1 Combining data sets

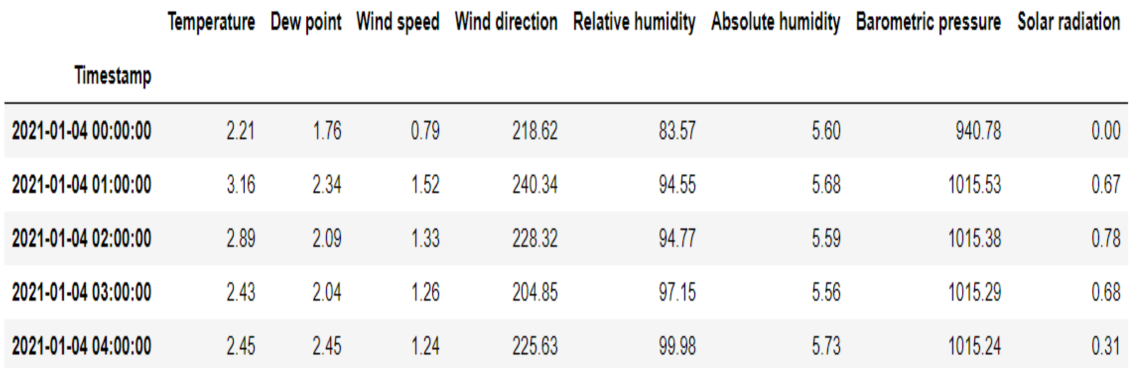
The files provided by SINTEF contained two CSV-files where the first file had solar power production values of each facades, while the second file had values of weather parameters recorded by SINTEF's test-cell. When processing data sets, the first step is to retrieve the CSV files over to Python platform, where the refinements takes place and improves the data to algorithms.

After importing the two separate CSV-files, we are now interested in observing how the data sets looks like:



Timestamp	East	North	Pergola	Roof	South	West	Total
2022-03-31 14:00:00	2.852	0.742	3.503	51.825	10.472	3.661	73.055
2022-03-31 15:00:00	2.193	0.643	1.344	23.628	3.558	1.560	32.926
2022-03-31 16:00:00	0.697	0.146	0.136	3.285	0.297	0.232	4.793
2022-03-31 17:00:00	0.976	0.192	0.060	1.879	0.530	0.371	4.008
2022-03-31 18:00:00	0.712	0.092	0.037	1.272	0.247	0.195	2.555

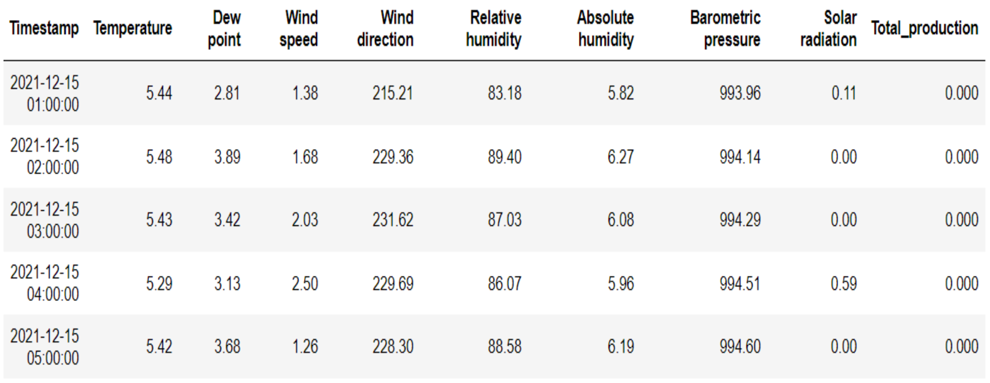
Figure 15: ***Energy production of each facades in*** KWh



	Temperature	Dew point	Wind speed	Wind direction	Relative humidity	Absolute humidity	Barometric pressure	Solar radiation
Timestamp								
2021-01-04 00:00:00	2.21	1.76	0.79	218.62	83.57	5.60	940.78	0.00
2021-01-04 01:00:00	3.16	2.34	1.52	240.34	94.55	5.68	1015.53	0.67
2021-01-04 02:00:00	2.89	2.09	1.33	228.32	94.77	5.59	1015.38	0.78
2021-01-04 03:00:00	2.43	2.04	1.26	204.85	97.15	5.56	1015.29	0.68
2021-01-04 04:00:00	2.45	2.45	1.24	225.63	99.98	5.73	1015.24	0.31

Figure 16: ***Measurements on weather parameters***

We observe that the two files start at different times and have distinct time formats, since weather file has YYYY-DD-MM timeformat. We modify the files' starting and ending intervals to be equal and change the time format to a desirable one using Python manipulations. The next step is to merge the two data sets into one after deleting all of the columns in figure 15 except "Timestamp" and "Total".



Timestamp	Temperature	Dew point	Wind speed	Wind direction	Relative humidity	Absolute humidity	Barometric pressure	Solar radiation	Total_production
2021-12-15 01:00:00	5.44	2.81	1.38	215.21	83.18	5.82	993.96	0.11	0.000
2021-12-15 02:00:00	5.48	3.89	1.68	229.36	89.40	6.27	994.14	0.00	0.000
2021-12-15 03:00:00	5.43	3.42	2.03	231.62	87.03	6.08	994.29	0.00	0.000
2021-12-15 04:00:00	5.29	3.13	2.50	229.69	86.07	5.96	994.51	0.59	0.000
2021-12-15 05:00:00	5.42	3.68	1.26	228.30	88.58	6.19	994.60	0.00	0.000

Figure 17: ***Combined data sets***

3.1.2 Statistics

Next step would be to look at some statistics on the dataframe, and this can be done with pandas describe() function. The reason for this is to get a sense of what kind of data we possesses.

Table 2: Summary of measured data

	Temp	Dew	W.speed	W.direction	R.humidity	A.humidity	Barometric	Solar	Total_production
count	2567.0	2567.0	2567.0	2567.0	2567.0	2567.0	2567.0	2567.0	2567.0
mean	1.542	-2.034	1.588	191.274	80.228	4.318	1001.416	27.877	5.551
std	4.331	4.308	1.099	54.775	20.667	1.350	15.547	68.696	16.394
min	-12.430	-16.930	0.140	38.760	17.01	1.280	963.820	0.00	0.00
25%	-1.040	-5.215	0.680	150.00	64.915	3.285	990.570	0.00	0.00
50%	1.570	-1.940	1.330	205.87	86.490	4.170	1003.29	1.040	0.00
75%	4.285	1.070	2.235	228.065	100.00	5.210	1013.810	12.00	1.781
max	15.210	9.930	6.380	331.32	100.00	9.360	1037.550	483.110	111.656

- ***count-*** Quantity of non-values
- ***mean-*** Mean/average of each parameter

$$\bar{x} = \frac{\sum x}{N} \tag{3.1}$$

- ***std-*** The standard deviation

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{3.2}$$

- ***min-*** The minimum value
- ***25%-*** The 25% percentile
- ***50%-*** The 50% percentile
- ***75%-*** The 75% percentile
- ***max-*** The maximum value

3.2 Visualization

After pre-processing and statistical analysis, a good next step is to check for missing values in the data set. This can be accomplished by utilizing the Seaborn heatmap function, which provides a visualizing perspective that can validate that all parameters have no gaps or missing data.

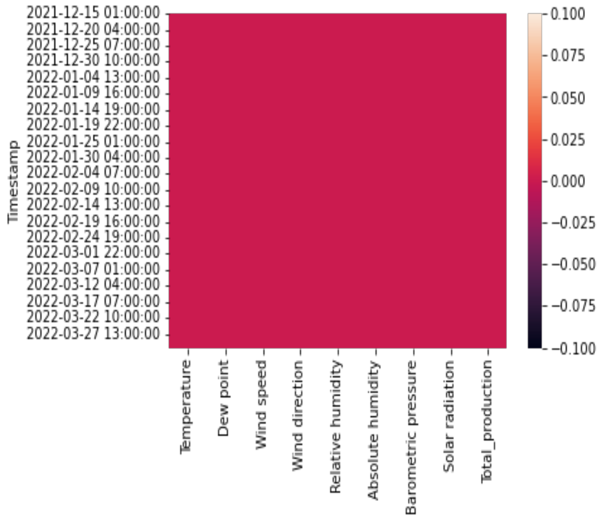
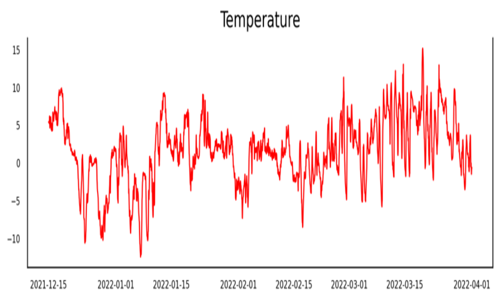
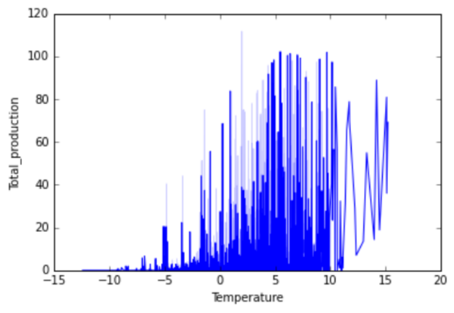


Figure 18: ***Heatmap on missing values***

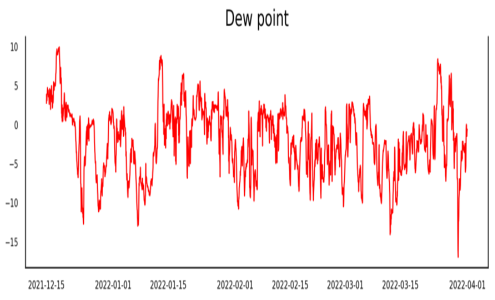
When it comes to analyzing data, trends and then developing algorithms, it is usually a good idea to plot each and every parameter first, and get an overview of how different parameters behave over the time frame. Here are the features and target variables presented, first each feature then the features and target variable in lineplots.



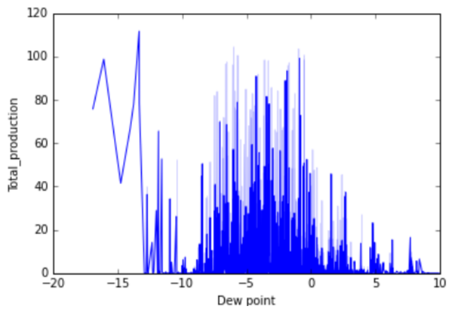
(a) Temperature in C°



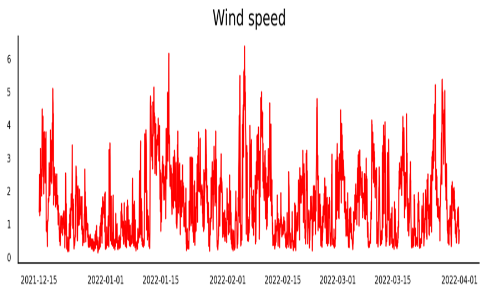
(b) Lineplot of Temperature and Total_production



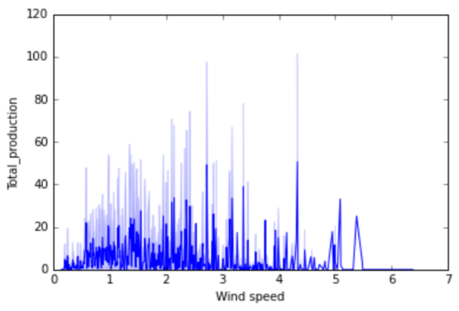
(c) Dew point



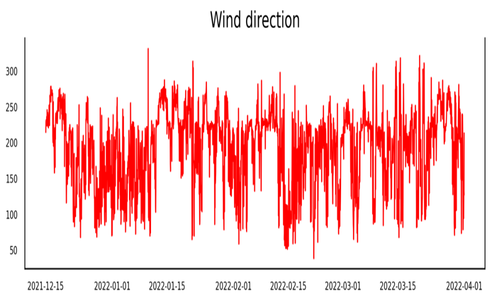
(d) Lineplot of Dew point and Total_production



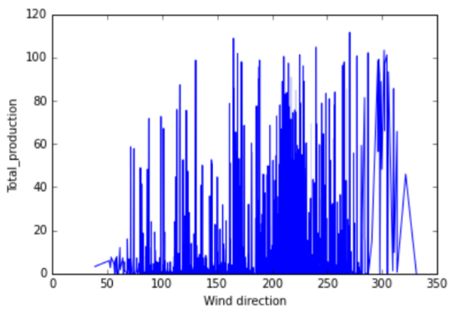
(e) Wind speed measured in m/s



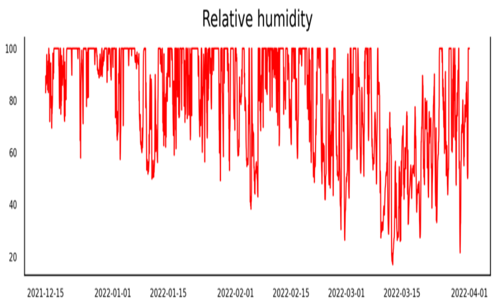
(f) Lineplot of Wind speed and Total_production



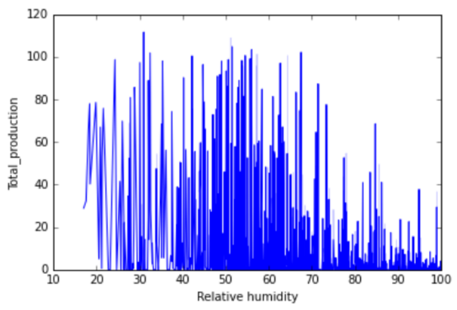
(g) Wind direction



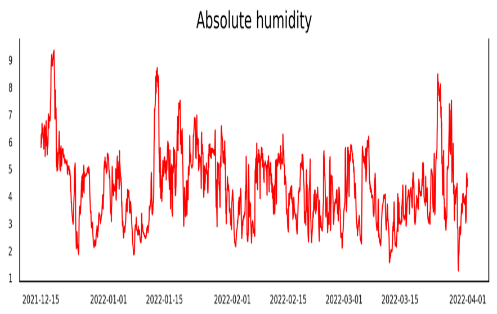
(h) Lineplot of Wind direction and Total_production



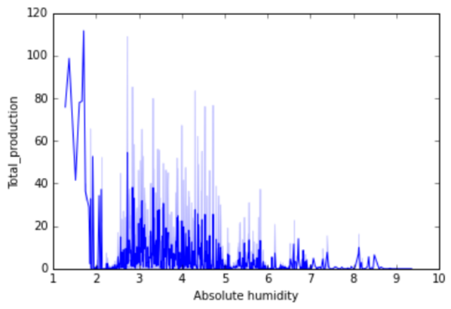
(i) Relative humidity



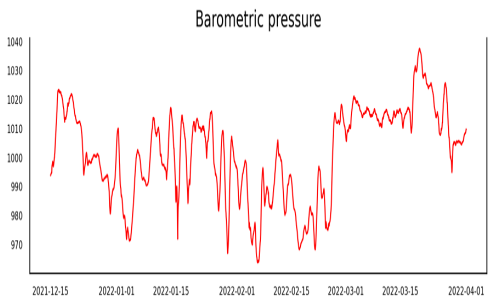
(j) Relative humidity and Total_production



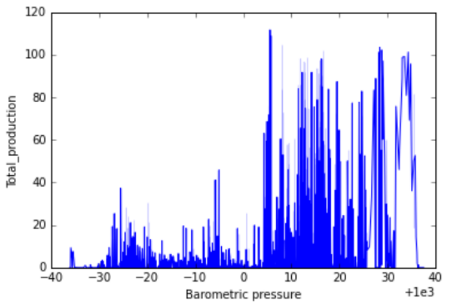
(k) Absolute humidity in g/m^3



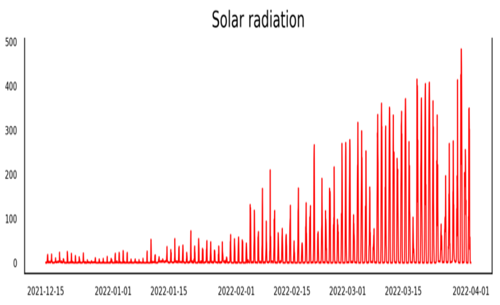
(l) Absolute humidity and Total_production



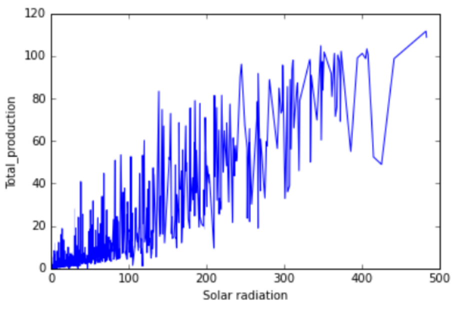
(m) Barometric pressure



(n) Barometric pressure and Total_production



(o) Solar radiation that is calculated by a horisontal pyranometer that measures directly upwards



(p) Solar radiation and Total_production

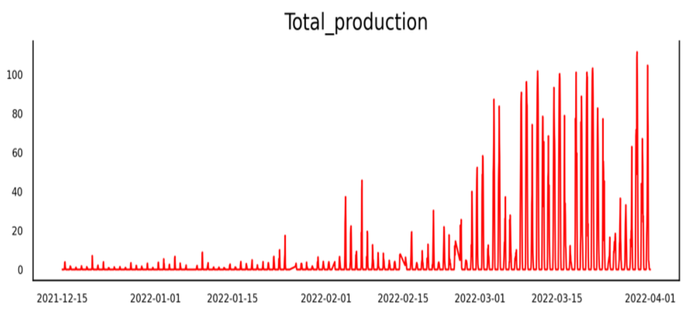


Figure 19: ***Total production in KWh of each facade***

Since Total_production is the target variable, hence the one we are going to predict and forecast, it may be appropriate to visualize how monthly, weekly and daily production operates:

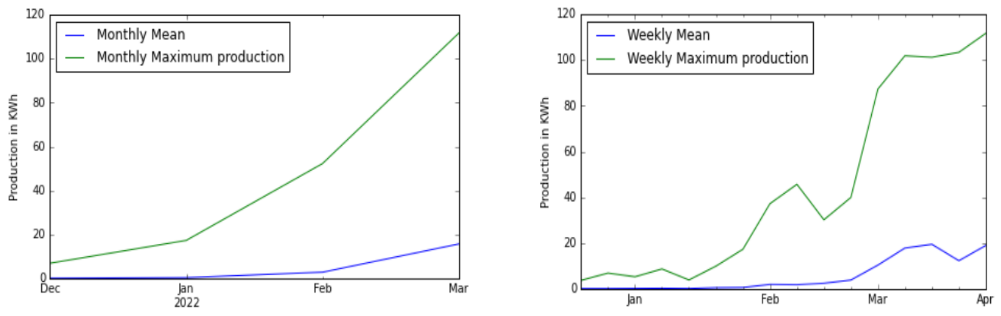


Figure 20: ***Monthly and weekly PV production***

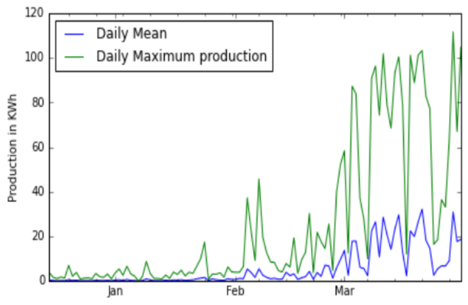


Figure 21: ***The daily mean and max PV production***

3.3 Seasonal patterns recognition

When managing time series problems, a common belief is that history repeats itself, hence we can assume that future data patterns may potentially resemble in historical data patterns. Seasonal patterns effects can be viewed as related to calendar effects, where different days/weeks/months/holidays and seasonal periods can affect on collected data sets.

Seasonality is generally divided into two branches; ***an additive seasonality pattern*** and ***a multiplicative seasonality pattern***

- ***Additive seasonality:*** The seasonal elements is attached as a absolute value, utilized when the elements amplitudes remains constant.
- ***Multiplicative seasonality:*** The amplitude increases/decreases during the rise of trends.

Additive seasonal decomposition formula:

$$Observed(t) = Trend(t) + Seasonality(t) + Random(t) \tag{3.3}$$

Multiplicative seasonal decomposition formula:

$$Observed(t) = Trend(t) * Seasonality(t) * Random(t) \tag{3.4}$$

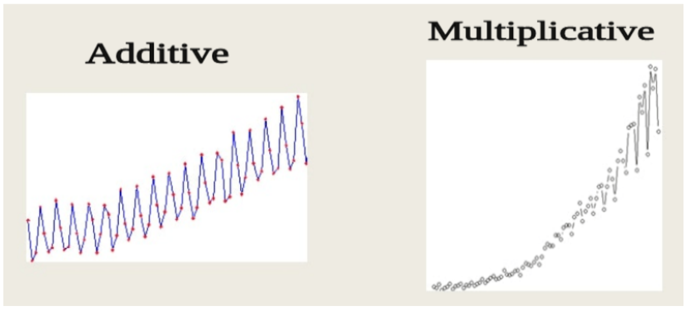
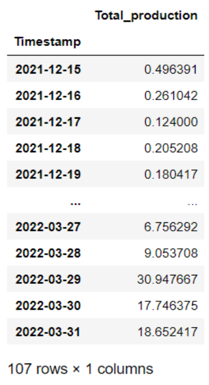


Figure 22: ***Illustration of the different observed decomposition's***

Our dataframe is a combination of these two decomposition's, which creates a new system labeled "pseudo-additive" decomposition. Our model contains small and zero-value numbers, resulting in us not being able to use the multiplicative decomposition [45]. Before we can plot the various patterns, we must manipulate the dataframe. Originally, the frame has 2567 points, and this is not appropriate when it comes to plotting. We have chosen to resample the frame into daily averages, to smooth the curves and avoid zero-values. This approach reduces the amount of points to 107.



	Total_production
Timestamp	
2021-12-15	0.496391
2021-12-16	0.261042
2021-12-17	0.124000
2021-12-18	0.205208
2021-12-19	0.180417
...	...
2022-03-27	6.756292
2022-03-28	9.053708
2022-03-29	30.947667
2022-03-30	17.746375
2022-03-31	18.652417

107 rows × 1 columns

Figure 23: ***Total_production re-sampled daily.***

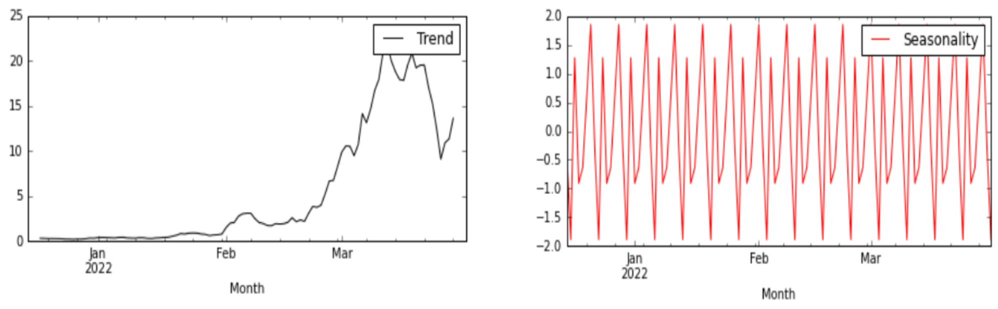


Figure 24: ***Plots of trend & seasonality***

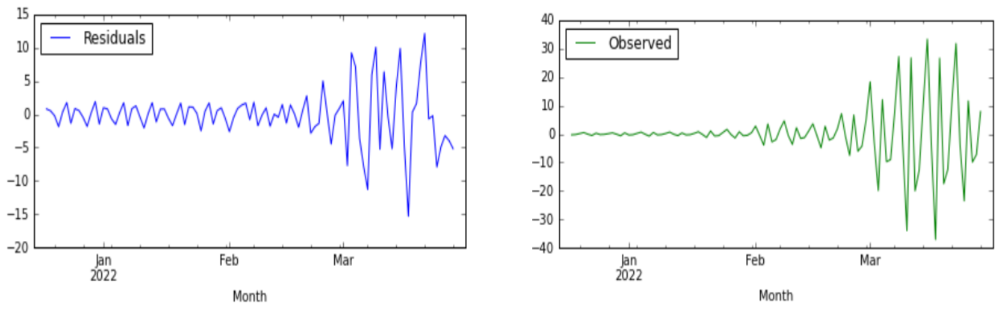


Figure 25: ***Plots of residuals & observed***

3.4 Feature selection

Before we carry on with simulating each algorithm, we should also examine which parameters correlates. This is done to optimize the algorithms and potentially remove unimportant variables that can result in noise levels. Determination of the correlated variables can be done by looking at techniques such as heat-map correlations.

3.4.1 Pearsons correlations test

Pearsons correlations test *[PCT]* is a statistical method for finding the linear connection between variables, and correlation demonstrates how robust the connection is. The values for correlation goes from -1 to 1.

- ***Positive correlation coefficient = 1:*** When one variable increases, the other increases simultaneously in the same rate.
- ***Negative correlation coefficient = -1:*** When one variable increases, the other variable decreases simultaneously in same rate.
- ***Random correlation coefficient = 0:*** No correlation between the variables, changes are randomly.

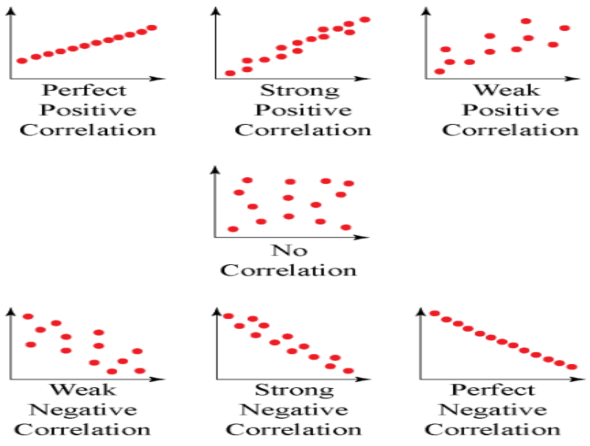

Perfect Positive Correlation
Strong Positive Correlation
Weak Positive Correlation
No Correlation
Weak Negative Correlation
Strong Negative Correlation
Perfect Negative Correlation

Figure 26: ***How different correlations occurs***

$$Pearsons = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{3.5}$$

Below we have put the values together pair-wise, and performed a Pearson correlation test for finding the coefficients, then plotted them into a heat-map. The coefficients express how dissimilar values are correlated from -1 to 1.

Heatmap on correlated values

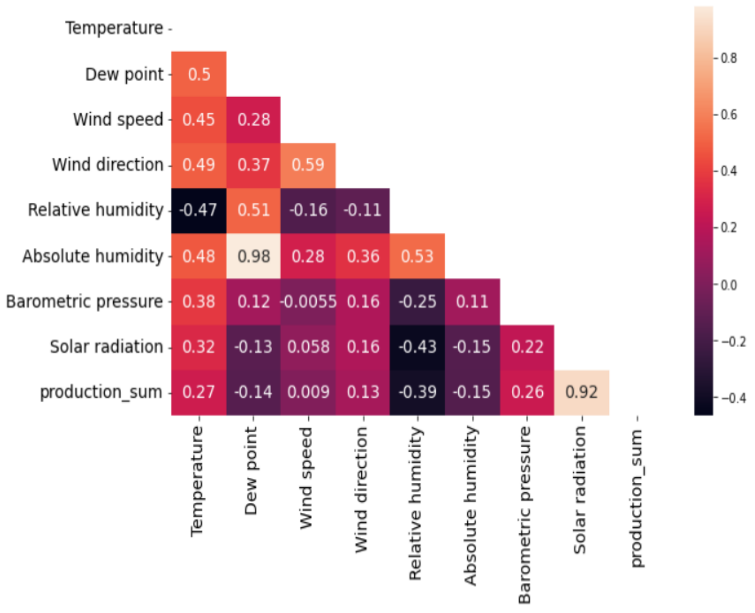


Figure 27: ***Heatmap on correlated values***

After glancing at the heatmap, we can observe that "Solar radiation" has the highest correlation with the targeted variable, while "Relative humidity" has the lowest negative correlation. We also see that "Wind speed" is by far the most random parameter, in terms of the main variable.

4 Methodology Part 2

This section will go deeper into key machine learning methods and applications for model optimization. The flowchart below shows how each algorithm's procedure is applied to the tasks.

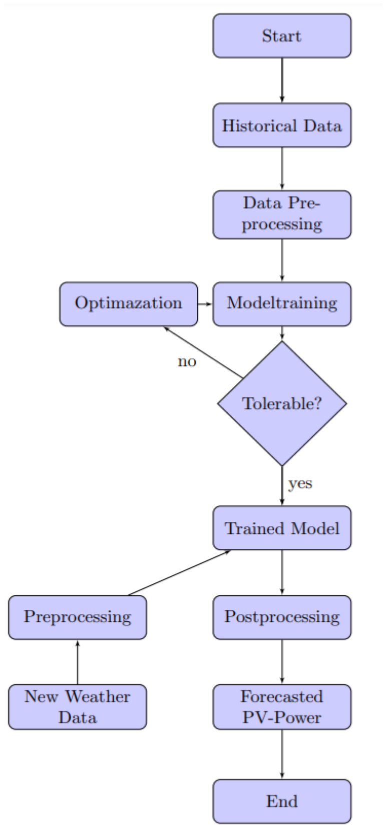


Figure 28: ***How the approach on the methodology part 2 section will occur***

4.1 Machine learning aspects

4.1.1 Train-Test split

Considering that we are in the process of predicting and forecasting future solar production using machine learning techniques, a conventional strategy is to split the data frame into training/test sets [46]. The ratio of sets depends on many factors such as the dimensions of the dataframe, the structure of the model and the intended use case. For our purposes, we use 90% for training and the remaining 10% for testing our models. The reason is that we want to use these samples for testing since our model has never seen them before. We think the split ratio of 90:10 makes sense here, since the data set on 2567 points is not that large, hence 2310 points will be in training and remaining 257 will be included in the testing set.

4.1.2 Cross-validation/K-Fold

In machine learning, a common approach when dealing with algorithms and predictions, we usually split the dataset into two sets, which are training and test set. Let's say we have a dataset with a 90:10 ratio like in our case, we train the model on 90% and then test it on the 10% the model hasn't already seen and then predict the values. This general approach has some shortcomings, namely that not every data point is included in the test set, so we only assess the strength of the model based on the 10% of the test set. Cross-validation solves this problem by dividing the entire set into approximately equal groups, as in our case Kfold = 5, meaning five groups each containing the same number of data points [47].

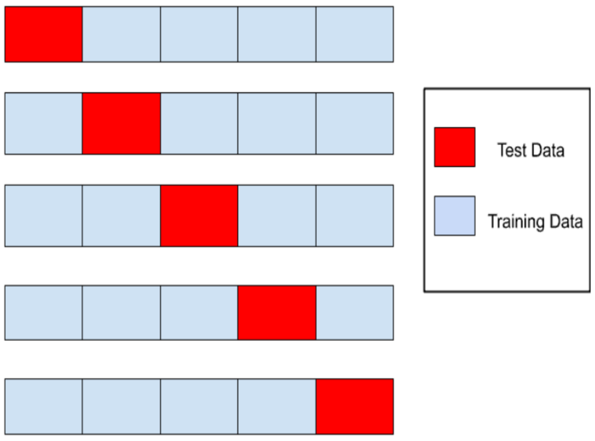


Figure 29: ***5-Fold Cross-Validation***

The procedure for KFold is that it first tests on the red block in the first row while training on the remaining gray blocks, then jumps to the next row while repeating the same pattern until it's gone through all the groups.

4.1.3 StandardScaler

It is often convenient for the features to be scaled relatively similarly in relation to each other. With StandardScaler from the scikit-learn library, each feature becomes equally important, while the algorithms are given almost equally scaled values so they don't get confused. StandardScaler compares different values through a method called for standardization, and in SVR this function becomes even more important because the algorithm is very sensitive to different values. Here is ***z = scaled output value, x = original value, μ = mean and σ = standard deviation from the mean.***

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

4.2 Hyper parameters

Adjusting "hyper parameters" is a tool for enhancing models. One may fine-tune the algorithms using this strategy to achieve more optimized performance. We've listed which "hyper parameters" were modified for each algorithm below.

4.2.1 Hyper parameters for RF

n_estimators

n_estimators can be defined as the numbers of trees used in the model. This hyperparameter encompasses the creation of multiple decision trees, while manage the quantity used in the system. It is beneficial to keep in mind that a higher value of *n_estimators* will have an impact of the running time of the code, but as well as provide a more stable and robust predictions. After experimenting with many values, these where used in the code; ***'n_estimators'= [50,100,150]***

max_depth

max_depth in RF is described as the maximum length of route there is between a root node and the leaf node. This is a critical hyperparameter for the model, as this parameter affect both the score and the pace of the algorithm. Low values for *max_depth* will lead to a lower possibility for over fitting the model. These were used for code; ***'max_depth': [4,5,6]***

max_features

This hyperparameter applies to the number of maximal attributes supplied to each tree in the model. For the code; ***'max_features':[1,2,3]***

4.2.2 Hyperparameters for XGboost

learning_rate

When it comes to XGboost, one crucial hyperparameter is the *learning_rate* which controls and determines how quickly the model adapts to the problem. With Xgboost a possible outcome can be that the decision trees learn too quickly and therefore overfit the model, which can be issue when computing. *learning_rate* enters this equation just to slow down/pace up the learning speed, thus giving us an advantage in controlling the number of steps.

colsample_bytree

When constructing each tree, the subsample ratio of columns is used. Subsampling occurs only once for each tree built, and this is done by colsample bytree. In code; ***'colsample bytree':[0.5,0.6,0.7]***

4.2.3 Hyperparameters for SVR

C

The C hyper parameter instructs the SVR optimizer how much you don't want each training example to be misclassified. In code; ***'C': [150,200]***

gamma

gamma is a hyper parameter that is established before the training phase and is used to determine the decision boundary's tangent strength. In code; ***'gamma': [0.1, 1, 5]***

4.3 Web Application

4.3.1 Streamlit

The steps after data processing, training/testing and adapting to the best scoring model is to realize an interactive user interface that demonstrates the power and performance of the algorithms while deploying the model. For this we need a web application and there are many potential applications that can be used, but for the sake of simplicity we chose Streamlit. Unlike other well-known applications like Flask and Django, Streamlit is a simple and beginner-friendly application that is a Python extension, which means we don't have to learn any new programming languages since it is based on Python. Streamlit is an open-source framework that allows easy realization of web applications while offering the possibility to use some of the well-known Python packages such as sci-kit learn, matplotlib and pandas.

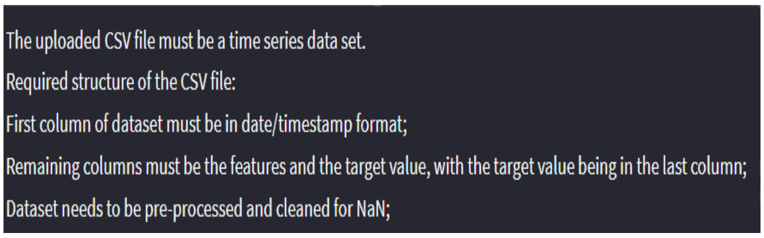
For our purposes we will use Streamlit to showcase the backend work done in the Jupyter Notebook and the applications will have two main functions, one representing the best models and the other allowing other users to upload their own data and see the performance of their dataset.

4.3.2 Restrictions

In terms of the application, we have set some limits on how input data from consumers should be. One limitation is that the file must be a time series, since the application is directly aimed at such problems. The algorithms here are modified to solve such problems, also the file must be in CSV format. A solution for other types was not in focus as the file we received was in a CSV file, so targeting the application at such files was a natural choice.

The input dataset must be in a natural form, and by that we mean date/timestamp must be in the first column, then columns with desired features, and finally a target variable feature in the last column. The purpose of this is that the application is non-interactive and does not take into account cleaning in the desired format, this will then significantly affect the predictions and plots.

In addition, the data set should be uniform and clean, i.e. it should have undergone preprocessing before it can be used in the application. There should be no empty cells in the CSV file uploaded to the application.


The uploaded CSV file must be a time series data set.

Required structure of the CSV file:

First column of dataset must be in date/timestamp format;

Remaining columns must be the features and the target value, with the target value being in the last column;

Dataset needs to be pre-processed and cleaned for NaN;

Figure 30: ***Information section from application***

4.4 Procedure

The approach will be that we will have ten different cases in which we will examine how the models respond in each case. In case 1, we will initially include all parameters, then in case 2, we will include all parameters again, but this time optimized and finally remove one by one weather parameter until we are left with only the last parameter. The four algorithms outlined in the theory section 2 will be utilized to forecast PV production. This strategy allows us to investigate the impact of excluding each specific parameter on models behavior while iterating the algorithms repeatedly. For the optimization part, we'll use grid searching for all algorithms except MLR, because the group couldn't come up with a technique for it.

When it comes to presenting the results, each case will be visually displayed using line plots, and the algorithms models will be compared to one another, as well as a presentation of the models metrics performance, as mentioned in chapter 2. The line plots will show randomly generated values retrieved by the test set, which will be compared to the model's predicted values at that test set index. A table showing the difference between actually produced values and predicted values will be displayed to give a clearer understanding of how each unique model works.

To calculate which model we think performs best, we choose to rank the models based on RMSE and R^2 values, with the model with the lowest RMSE value and the highest R^2 value. Both the MAE and MSE values give a good indication of how accurately the model is performing and what the difference in error is between desired solutions and predicted solutions, but for this purpose we will prioritize them down but also include them.

5 Results

5.1 Training/Test-Score

Below are the training and testing results each model has after the grid search and we notice that the "all features" has no value and that is because this operator only works on models that have completed the grid search. For RF and SVR models, a higher value closer to 1 indicates the model is well trained/tested, but for the XGBoost models, the value closest to 0 is ideal. The reason XGBoost differs from the others is that it used the neg_mean_squared_error metric, which is identical to MSE but only works with negative values instead.

Table 3: Training and test scores

Scores	All	All optimized	Temp	Dew	W.speed	W.direction	R.humidity	A.humidity	Barometric	Solar
Training RF	x	0.850	0.853	0.858	0.857	0.859	0.853	0.855	0.832	0.356
Training XG	x	-33.740	-33.952	-34.074	-34.251	-36.238	-34.532	-34.227	-40.157	-150.181
Training SVR	x	0.803	0.845	0.805	0.843	0.838	0.811	0.804	0.791	0.323
Testing RF	x	0.871	0.868	0.878	0.872	0.864	0.854	0.874	0.864	0.418
Testing XG	x	-8.935	-8.761	-8.072	-10.094	-11.918	-7.975	-11.297	-14.459	-49.705
Testing SVR	x	0.954	0.950	0.949	0.940	0.938	0.951	0.951	0.916	0.991

We find that RF performs quite similarly in both training and testing, with slightly higher values in testing, which can tell us that the training for this cross-validation algorithm was successful.For SVR and XGBoost we experience a big improvement when the phase shift from training over to testing. These are significant improvements and it is reasonable to assume that the cross-validation process using the 5-fold technique responds well to these models.

5.2 Model evaluation

5.2.1 Comparisons between cases

Here are the 10 various cases displayed along with plots of each model next to each other and, lastly a table illustrating models metrics measurements.

Case 1: All parameters as features

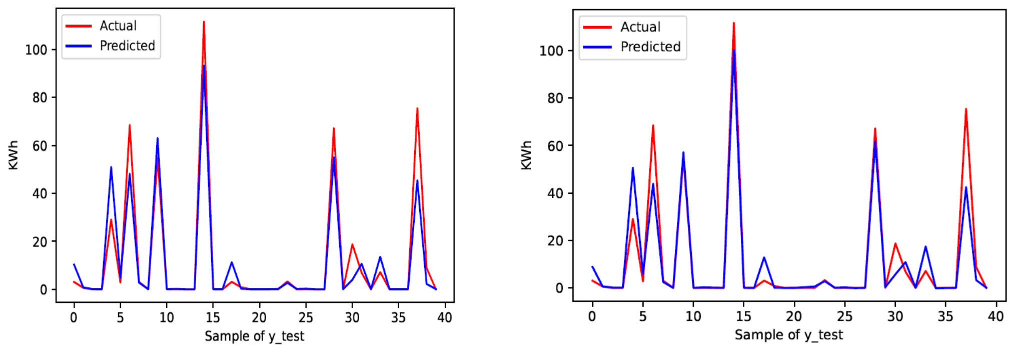


Figure 31: ***RF & XGBoost***

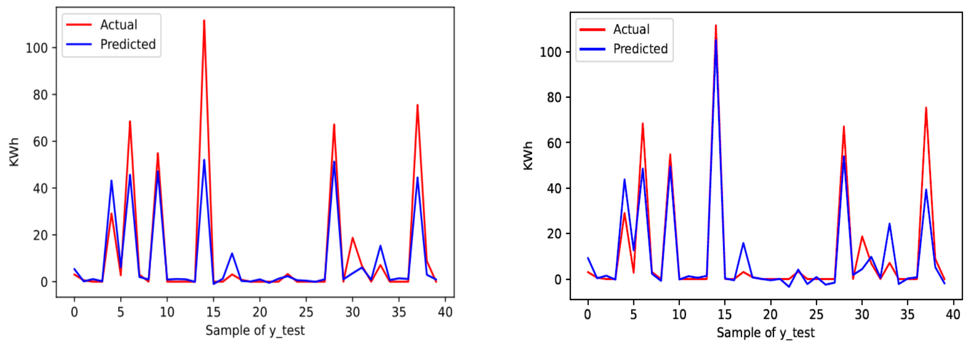


Figure 32: ***SVR & MLR***

Table 4: Performance comparison for Case 1

	RF	XG	SVR	MLR
MAE	2.467	2.381	2.944	3.447
MSE	45.908	41.332	51.434	52.759
RMSE	6.776	6.429	7.172	7.264
Rˆ2	0.8730	0.8856	0.8577	0.8540

In case 1, we observe XGBoost model has the best performance when it comes to including all features but not optimized. XGBoost have the lowest values in MAE, MSE and RMSE. while the highest value on R^2, while MLR is on the other side of the spectrum, with the highest values on MAE, MSE and RMSE and the lowest at R^2. For RF and SVR, the models seem to underfit on many occasions.

Case 2: All parameters optimized as features

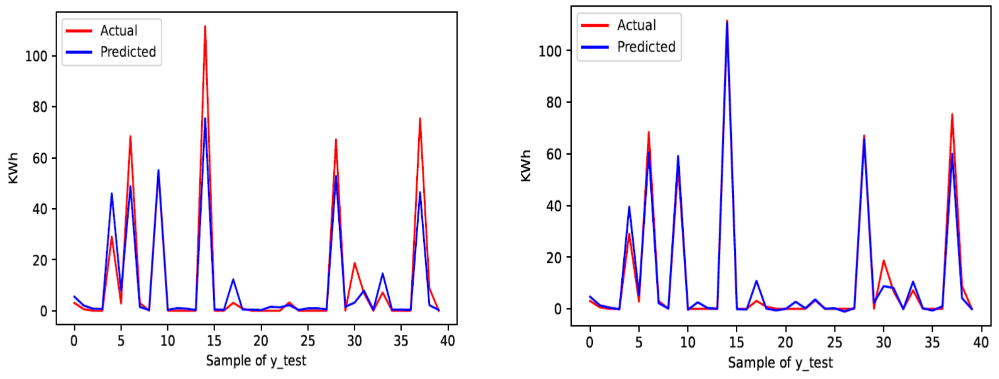


Figure 33: ***RF & XGBoost***

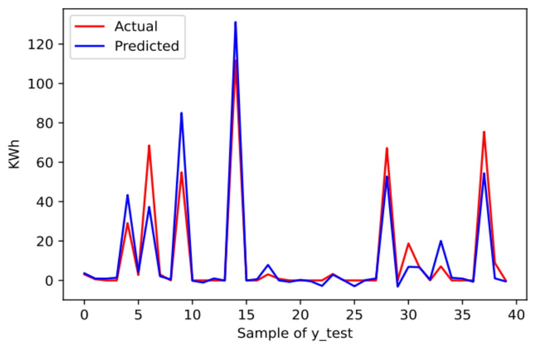


Figure 34: ***SVR***

Table 5: Performance comparison for Case 2

	RF	XG	SVR	MLR
MAE	2.970	1.400	3.062	x
MSE	49.951	8.935	51.674	x
RMSE	7.068	2.989	7.188	x
Rˆ2	0.8618	0.9753	0.8570	x

For case 2 RF, XGBoost and SVR have been optimized with all features included. Again, we find that XGBoost performs best overall in comparisons to RF and SVR, and XGBoost is also the model that performs better after grid search, while the others do not show significant improvements, but rather a down scaling of the metrics.

Case 3: All parameters optimized excluding temperature

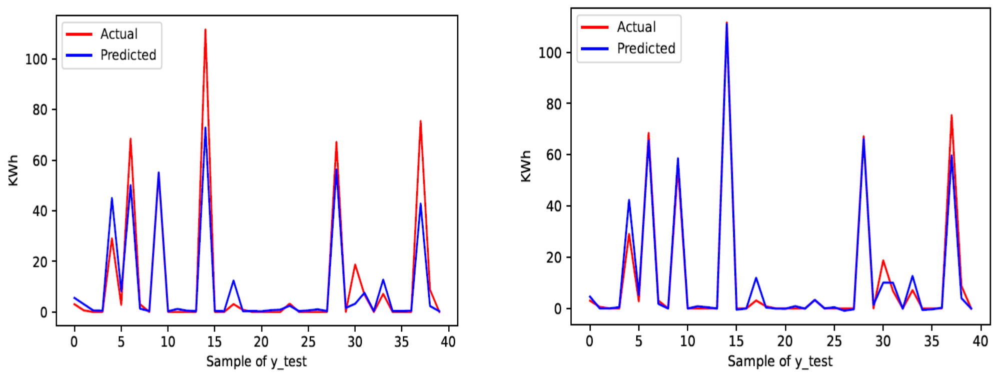


Figure 35: ***RF & XGBoost***

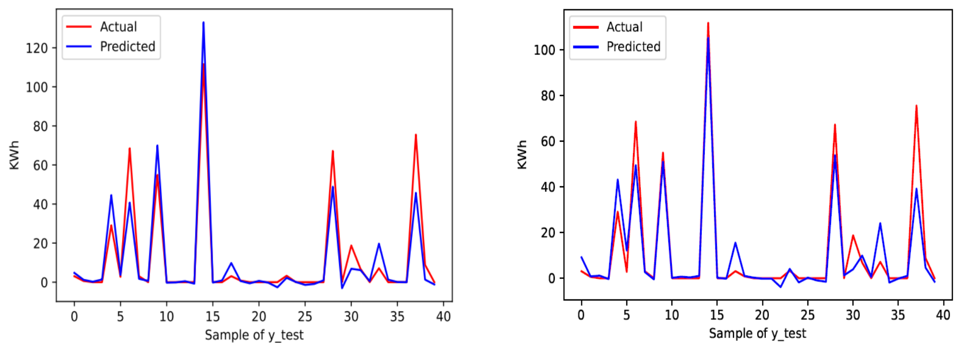


Figure 36: ***SVR & MLR***

Table 6: Performance comparison for Case 3

	RF	XG	SVR	MLR
MAE	2.717	1.265	2.854	3.394
MSE	43.804	8.761	42.368	53.058
RMSE	6.618	2.960	6.509	7.284
Rˆ2	0.8788	0.9758	0.8828	0.8532

In case 3, XGBoost stands out with the best overall metrics performance, while MLR has the worst metrics. There are a multitude of similarities between RF and SVR, with these models hardly being separated. RF has lower MAE and R^2 compared to SVR while SVR has lower MSE and RMSE.

Case 4: All parameters optimized excluding dew point

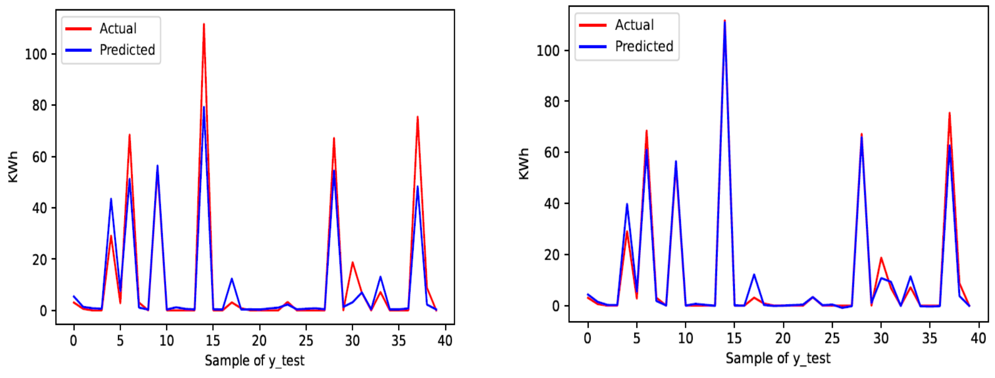


Figure 37: ***RF & XGBoost***

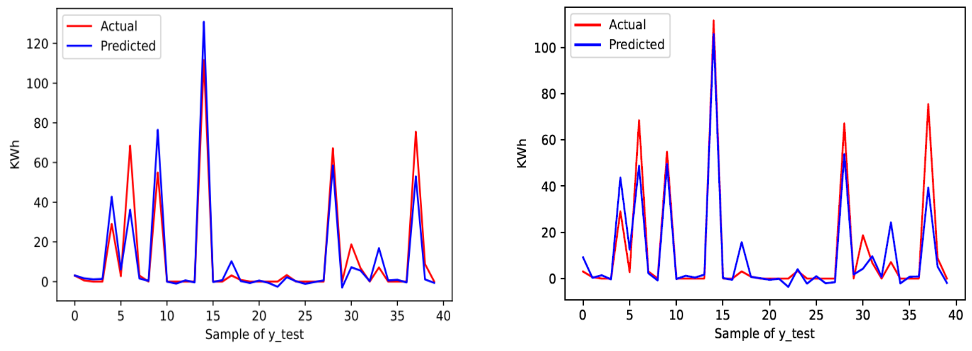


Figure 38: ***SVR & MLR***

Table 7: Performance comparison for Case 4

	RF	XG	SVR	MLR
MAE	2.747	1.228	3.048	3.451
MSE	44.928	8.072	53.308	52.750
RMSE	6.703	2.841	7.301	7.263
R^2	0.8757	0.9777	0.8525	0.8541

In Case 4, the dew point feature was removed and the first thing that stood out was that XGBoost differed from the other best performing models. SVR is comparable to MLR with similar metrics and we can also observe on the plots that between intervals 15-20 on the x-axis all models are overfitted.

Case 5: All parameters optimized excluding wind speed

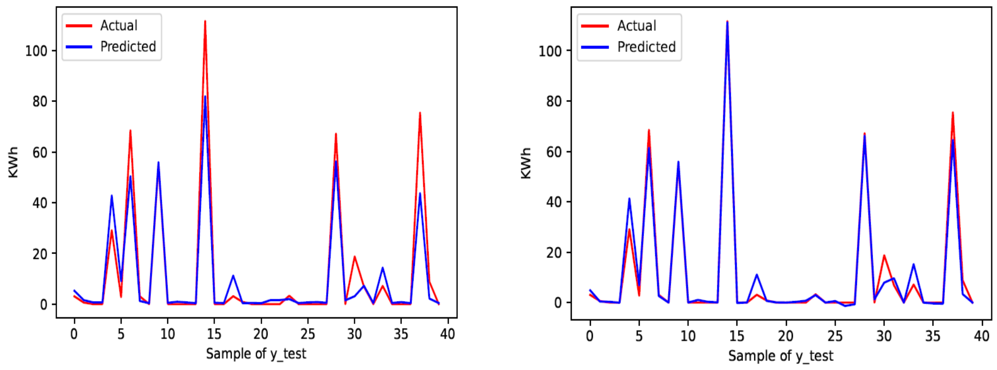


Figure 39: ***RF & XGBoost***

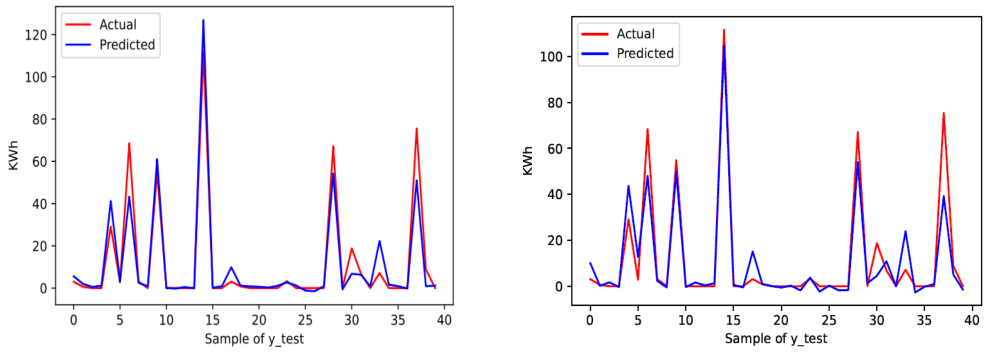


Figure 40: ***SVR & MLR***

Table 8: Performance comparison for Case 5

	RF	XG	SVR	MLR
MAE	2.825	1.335	2.782	3.415
MSE	48.170	10.094	43.825	53.489
RMSE	6.940	3.177	6.620	7.314
Rˆ2	0.8667	0.9721	0.8788	0.8520

For case 5 we observe the same patterns as for the other cases regarding XGBoost, where the algorithm has the highest performance. Here, SVR and MLR are similar in terms of metrics.

Case 6: All parameters optimized excluding wind direction

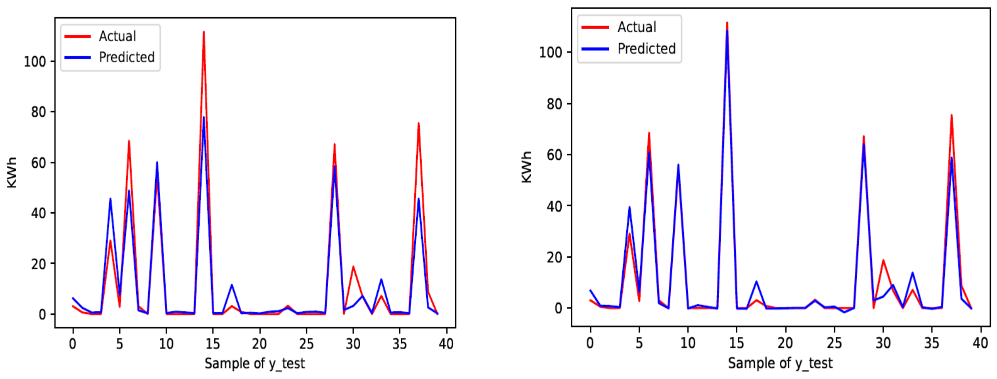


Figure 41: ***RF & XGBoost***

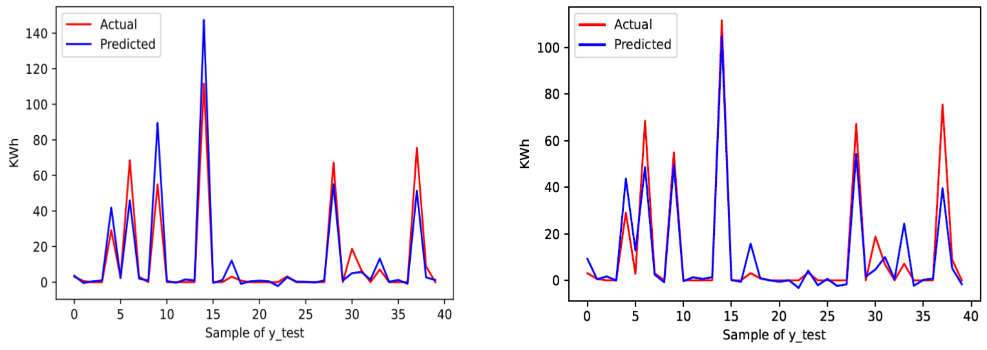


Figure 42: ***SVR & MLR***

Table 9: Performance comparison for Case 6

	RF	XG	SVR	MLR
MAE	2.830	1.591	3.303	3.454
MSE	47.687	11.918	69.559	52.825
RMSE	6.906	3.452	8.340	7.268
Rˆ2	0.8681	0.9670	0.8076	0.8540

In case 6 we notice a drop in performance for XGBoost, where the overall metric performs slightly worse than case 5. Another notable thing is that for SVR we see that the R2 score for this model is significantly lower than case 5, while at the same time MAE, MSE and RMSE are higher.

Case 7: All parameters optimized excluding relative humidity

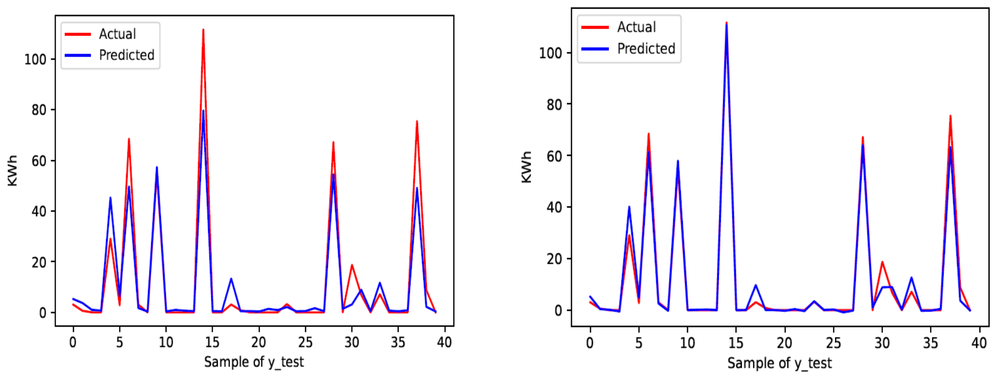


Figure 43: ***RF & XGBoost***

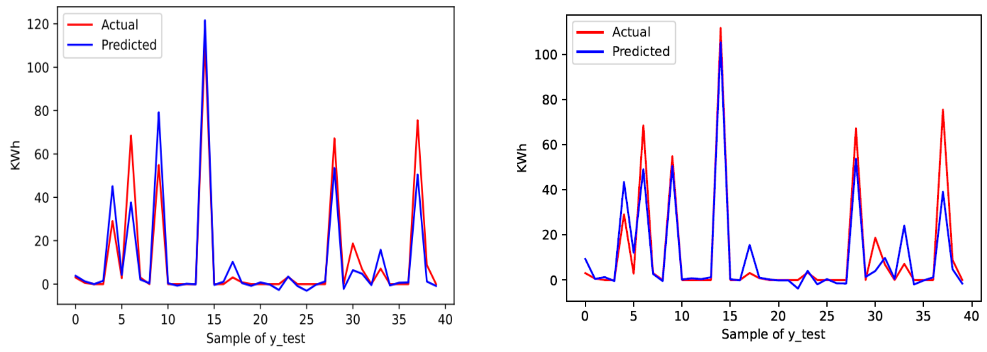


Figure 44: ***SVR & MLR***

Table 10: Performance comparison for Case 7

	RF	XG	SVR	MLR
MAE	2.963	1.319	3.054	3.418
MSE	50.609	7.975	51.993	53.010
RMSE	7.114	2.824	7.211	7.281
Rˆ2	0.8600	0.9779	0.8562	0.8533

In case 7, RF, SVR, and MLR have similar metric performances with few difference, while the XGBoost model again performs better than the others. For XGBoost we can observe that the model hits the peak well in the interval 12.5-15 on the X-axis, while SVR seems to slightly overfit and MLR to underfit at this point.

Case 8: All parameters optimized excluding absolute humidity

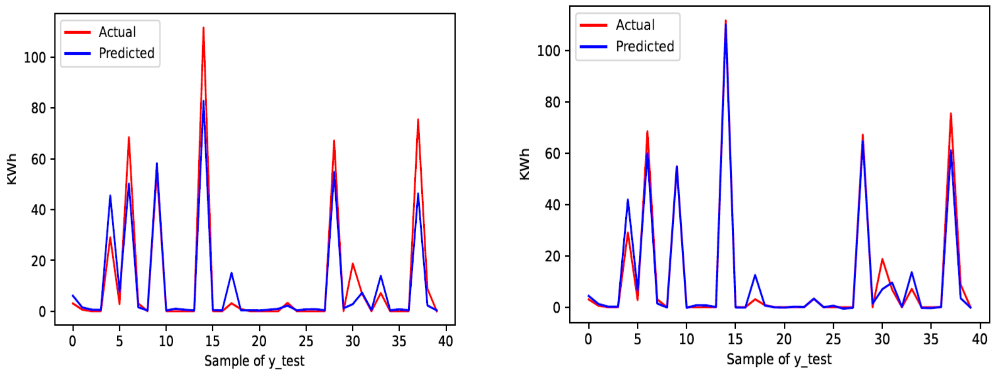


Figure 45: ***RF & XGBoost***

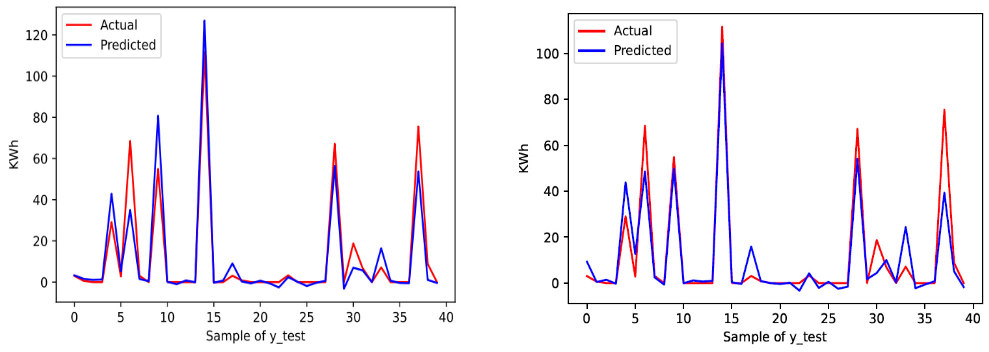


Figure 46: ***SVR & MLR***

Table 11: Performance comparison for Case 8

	RF	XG	SVR	MLR
MAE	2.846	1.450	3.046	3.431
MSE	47.203	11.297	53.910	52.860
RMSE	6.870	3.361	7.342	7.271
Rˆ2	0.8694	0.9687	0.8508	0.8538

In case 8 we see that the initial starting point on the RF and MLR plots the prediction starts slightly earlier than the actual plot, while for SVR and XGBoost it starts symmetrically.

Case 9: All parameters optimized excluding barometric pressure

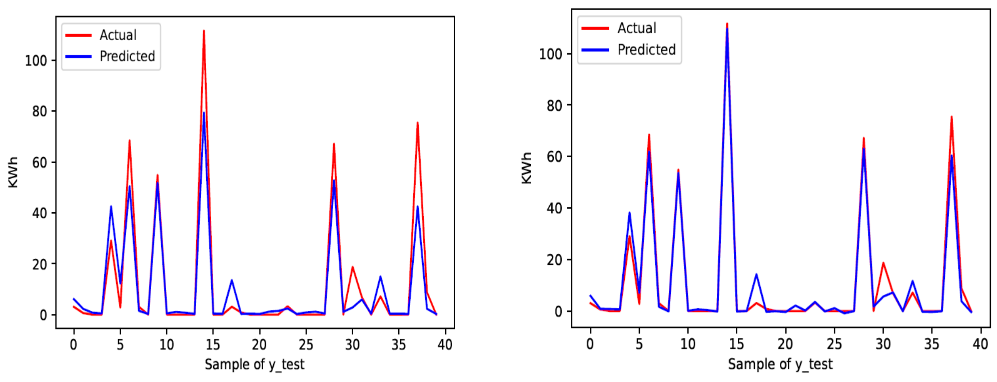


Figure 47: ***RF & XGBoost***

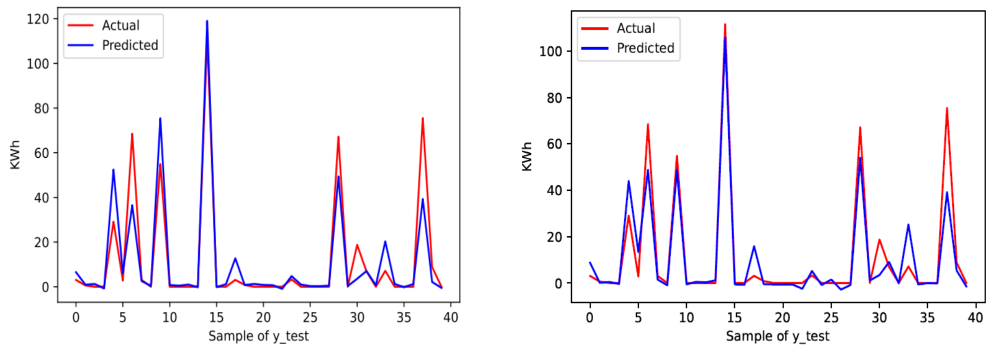


Figure 48: ***SVR & MLR***

Table 12: Performance comparison for Case 9

	RF	XG	SVR	MLR
MAE	3.110	1.737	3.230	3.349
MSE	53.292	13.459	63.140	53.686
RMSE	7.300	3.669	7.946	7.327
Rˆ2	0.8526	0.9628	0.8253	0.8515

In case 9 we observe that every model starts the predicted graph slightly above the actual one, same as for previous case.

Case 10: All parameters optimized excluding solar radiation

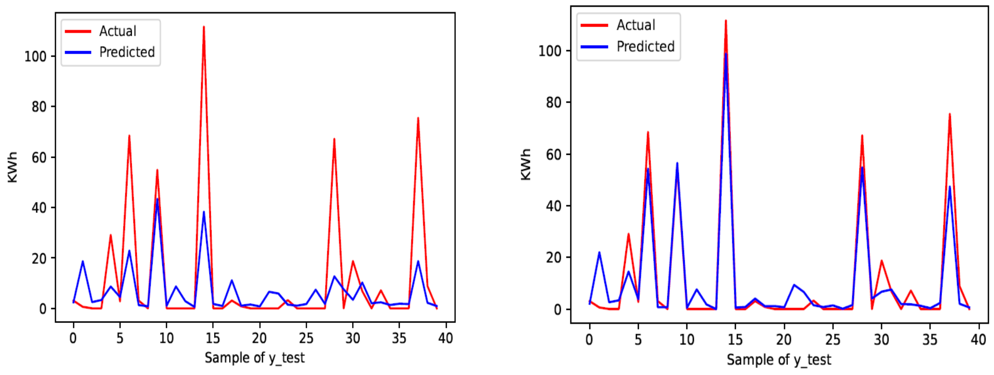


Figure 49: ***RF & XGBoost***

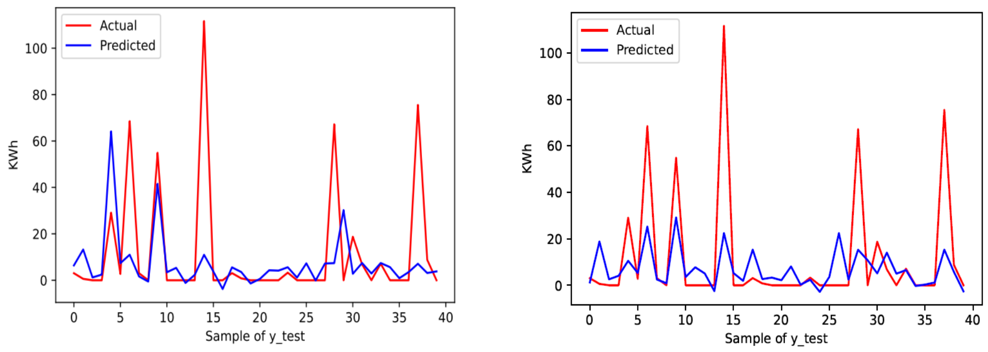


Figure 50: ***SVR & MLR***

Table 13: Performance comparison for Case 10

	RF	XG	SVR	MLR
MAE	6.969	3.637	8.175	8.634
MSE	207.643	49.705	271.803	266.520
RMSE	14.410	7.050	16.486	16.325
R^2	0.4255	0.8625	0.2480	0.2626

For the last case we see a big drop, especially for RF, SVR and MLR, while the XGBoost model experiences a slight drop in metrics but performs well relative to the others. This indicates that the solar radiation parameter plays an important role while the XGBoost model performs very well. Below we see the average value of the RMSE and R^2 scores for each algorithm.

Table 14: Average RMSE & R^2 scores

	RF	XG	SVR	MLR
RMSE	7.67	3.87	8.21	8.29
R^2	0.82	0.95	0.79	0.78

5.2.2 Differences between actual and predicted

After running through all the models and doing metric calculations, we might want to have some real world values to compare the predicted solution to the actual one as a reference point in addition to the plots to see how our models behave versus the actually produced values ***y_test****. Actual values are randomly selected from the test set sample.*

RF

Table 15: Performance of a forecast model based on RF

y_test	All	All optimized	Temp	Dew	W.speed	W.direction	R.humidity	A.humidity	Barometric	Solar
3.035	10.37	6.04	5.83	5.39	5.38	6.43	4.96	5.99	5.78	2.00
0.602	0.70	2.77	2.31	1.35	1.82	2.70	2.85	1.98	2.06	21.30
0	0.1	1.12	0.72	0.82	0.66	0.64	1.00	0.62	0.69	2.24
0	0.1	0.83	0.61	0.63	0.76	0.63	0.68	0.53	0.60	3.64
29.113	53.80	42.82	44.58	43.52	45.06	46.55	43.10	47.48	41.97	7.10
2.730	5.35	6.19	6.97	7.52	7.42	7.33	5.53	7.15	11.64	4.24
68.527	47.54	50.34	52.30	51.32	48.77	49.20	49.42	51.01	51.69	23.84

Comparing the actual and predicted values for the RF case, we see that the model never gets right, but in most cases falls short with a little or too much. In one case with the solar radiation model we see the model missing a lot with a production of 21,30 where it should have been around 0,602.

XGBoost

Table 16: Performance of a forecast model based on XG

y_test	All	All optimized	Temp	Dew	W.speed	W.direction	R.humidity	A.humidity	Barometric	Solar
3.035	8.83	4.68	4.62	4.32	4.85	6.83	5.19	4.38	5.89	2.08
0.602	0.63	1.36	0.02	1.44	0.41	0.94	0.43	1.25	0.87	21.98
0	0.13	0.44	0.01	0.25	0.28	0.78	0.13	0.20	0.77	2.61
0	0.12	-0.18	0.49	0.24	-0.07	0.39	-0.50	0.22	0.64	3.37
29.113	50.57	39.60	42.41	39.83	41.33	39.52	40.20	41.97	38.28	14.52
2.730	5.83	5.27	5.2	5.47	6.83	6.33	4.86	6.46	7.08	3.80
68.527	43.90	60.73	65.59	61.02	61.38	60.93	61.36	60.00	61.89	54.32

The most striking part of the XGBoost case instance is the similar spike on the model without solar radiation as for the RF case, where the predicted value is higher than the actual value. Surprisingly, there were some negative numbers in this case, which can be attributed to the scaling methods used, which included StandardScaler.

SVR

Table 17: Performance of a forecast model based on SVR

y_test	All	All optimized	Temp	Dew	W.speed	W.direction	R.humidity	A.humidity	Barometric	Solar
3.035	5.36	3.61	4.75	3.06	5.59	3.58	3.84	3.28	6.50	6.40
0.602	0.14	0.95	1.19	1.64	1.98	-0.46	1.27	1.58	0.94	13.26
0	1.09	0.92	0.22	1.11	0.57	0.48	-0.02	1.09	1.23	1.20
0	0.18	1.35	1.37	1.35	1.10	0.89	1.52	1.28	-0.75	2.44
29.113	43.23	43.37	44.57	42.75	41.22	41.93	45.16	42.85	52.43	64.14
2.730	6.18	4.20	3.66	6.05	3.24	2.28	4.24	5.20	6.13	7.236
68.527	45.68	37.34	40.74	36.21	43.24	45.97	37.66	35.09	36.50	11.00

Closer inspection of the table reveals that, with the exception of the last case, the models perform decently, with implications that the models over-predicts in the majority of cases disregarded from the last row.

MLR

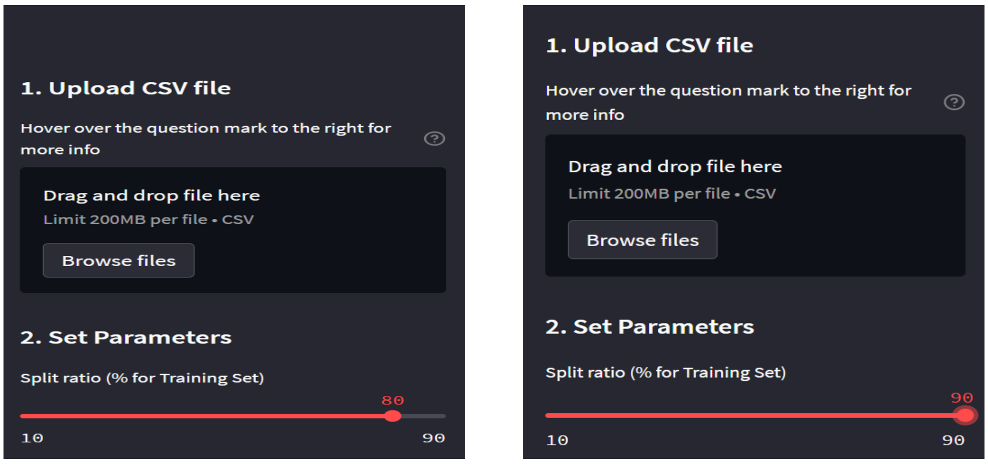
Table 18: Performance of a forecast model based on MLR

y_test	All	All optimized	Temp	Dew	W.speed	W.direction	R.humidity	A.humidity	Barometric	Solar
3.035	9.21	x	9.12	9.16	10.04	9.22	9.26	9.26	8.70	1.26
0.602	0.34	x	0.79	0.33	0.15	0.45	0.46	0.45	0.08	18.85
0	1.47	x	1.11	1.42	1.66	1.64	1.25	1.43	0.39	2.58
0	-0.26	x	-0.38	-0.35	-0.39	-0.07	-0.46	-0.25	-0.39	4.07
29.113	43.89	x	43.19	43.73	43.70	43.75	43.41	43.89	44.01	10.53
2.730	12.56	x	12.03	12.42	12.62	12.72	12.03	12.53	13.27	5.30
68.527	48.64	x	49.43	48.83	48.08	48.66	49.12	48.60	48.81	25.30

When it comes to MLR, we can observe that "All optimized" has no values specified. This is due to the fact that this algorithm did not go through a grid search. In several instances, we can find that the models both over-predict and under-predicts.

5.3 Streamlit

Our application will have two main functions, as stated in the method section 4. Users of this app will also be able to choose their preferred "training / test ratio," which the program will adjust to. When you change this variable, the model is retrained, and new predictions are generated. Below we first display a version of 80:20 "training / testing ratio", then further change the variable to 90:10 to show how the function occurs:



1. Upload CSV file

Hover over the question mark to the right for more info

Drag and drop file here
Limit 200MB per file • CSV

Browse files

2. Set Parameters

Split ratio (% for Training Set)

80

10 90

1. Upload CSV file

Hover over the question mark to the right for more info

Drag and drop file here
Limit 200MB per file • CSV

Browse files

2. Set Parameters

Split ratio (% for Training Set)

90

10 90

Figure 51: ***Training/testing ratios on Streamlit***

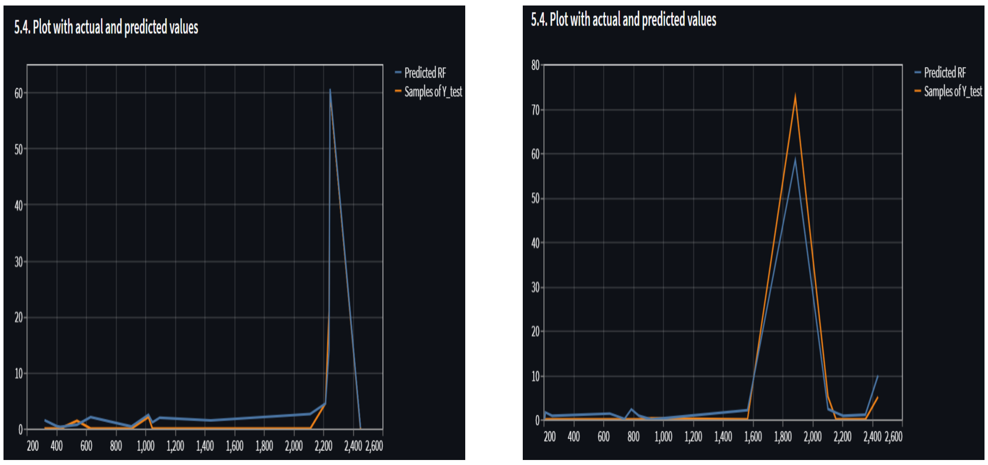


Figure 52: ***After editing ratios***

The link for the application can be found at;
https://share.streamlit.io/moahadii/streamlit/main/main.py

6 Discussion

In this section, the experiments and solutions discovered in the results chapter 5 are discussed and analyzed. The chapter is divided into several parts, where we first summarize the understanding of the project and then go deeper into analyses before finally offering some reflections on how future work can build on this paper.

6.1 Impacts on PV predictions

6.1.1 Weather influence

A source of uncertainty when working with PV production can be that solar production depends not only on previous production values, but also on factors such as weather parameters. Weather can have an impact on output production as solar panels have an ideal temperature range for best efficiency [48]. It can therefore be assumed that the weather parameters used in this dataset are perhaps theoretical in nature, since the records were only made for the winter months and not for the whole year. One can perhaps get a better overview of PV production in the ZEB-lab if you have more precise data over a longer period of time. Caution should be exercised when the sample size of data is small, as the results may not be as meaningful as desired.

6.2 Constraints

6.2.1 Area of focus

Since the task was narrowed to only develop an optimized model that predicts the power generation from PV's, rather than looking at the big picture holistically with energy management and maximum utilization of locally generated energy, our experiments will be rather limited. With a broader understanding of how the system is built and how it is supposed to work, the algorithms could possibly have been more precisely specified and targeted to contribute to the PRESAV structure.

6.2.2 Level of knowledge

The task partially catered to our specialization, but machine learning was a new and interesting concept that we had no experience with. In addition to all the new information the group needed to gather, one member dropped out of the program, and there were repercussions. The ideas and thoughts surrounding the project had to be reorganized.

6.3 Algorithms analysis

6.3.1 Pre-processing models

The algorithms scaling was first attempted with the MinMaxScaler function, rather than the StandardScaler, to ensure that the output predicted production always remained at positive values. The group was unable to come up with a solution for implementing MinMaxScaler, so StandardScaler was utilized instead. When using the StandardScaler, every model is scaled between -1 and 1, resulting in negative numbers when comparing actual and predicted values. For that reason these few negative points are incorrect since PV production cannot contain negative values.

6.3.2 Ratings of models

Except in case 10, RF models reacted decent with the data set. One thing to notice is that with all parameters assigned, the model had marginally better measurements than when the identical scenario was executed only with optimization. This is likely somewhat linked to the fitting part where the optimized model in case 2 may have been exposed for overfitting.

The most interesting findings that emerges from the different case analysis is that XGBoost outperformed the other algorithms in all ten cases in these experiments in terms of metrics performance. The XGBoost appeared to be quite promising for this data set, as evidenced by an average scores at RMSE of 3.87 and R^2 of 0.95. One unexpected observation was that even when the "Solar radiation" parameter was removed in Case 10, the model still produced satisfactory measures scores. Furthermore, when "Relative humidity" was omitted from the parameters, the highest values were obtained. For XGBoost 180 fits appeared to be the ideal because the computation time, (being the time the model takes to simulate and identifying the most optimal combinations of hyper parameters), got delayed, 180 fits was deployed. The computation time was excessive in circumstances where the number of fits was increased.

SVR and MRL performed somewhat lower than RF and XGBoost, although MRL remained the most steady throughout the testings, with the exception of the last case. Grid searching was not performed on MRL because the group couldn't come up with solutions and this algorithm doesn't have a lot of hyper parameters to adjust.

6.4 Application

After coming up with predictions, one of the project's next objectives was to locate and construct a digital system for showcasing the work done in the back-end. There is no option to insert an unclean data set into the application, and a solution to integrate this as a function was investigated however without any success. As a consequence, the application is unable to predict when a document with missing cells is uploaded. The algorithms RF, XGBoost, and MRL are applied in the application, and the case 1 is chosen, which includes all parameters but not optimized.

6.5 Future work

Despite these encouraging results, there are still several unanswered problems about how to combine machine learning with PV generation, as well as some areas of research that the project group believe should be investigated further:

- Include a larger data set that stretches over a year to get a broad overview of how the algorithms perform when the weather is more diverse. This will possibly enable one to see which one of the four algorithms should be explored further, and perhaps reach to better combinations of regulated hyper parameters.
- Use several machine learning approaches, such as deep learning techniques.
- Compare how a machine learning approach impacts the amount of energy returned back to the power grid with a case where no machine learning is used.
- Develop the web application further so that you may adjust particular weather parameter values and then receive an estimation of how much the PV's will generate the following day.

7 Conclusion

This chapter provides a summary of the conclusions found in the research process as well as the performance of the selected best model.

7.1 Optimization of the PV forecast and contribution in the energy sector

The purpose of this paper was to evaluate the various proposals for machine learning predictions, with a particular focus on the PRESAV project. The relevance of implementing machine learning in the PRESAV facility to predict power output is clearly supported by the current unstable electricity prices. This new understanding should help improve PV impacts on predictions and potentially make the PRESAV more cost and energy efficient. Smarter and innovative approaches are urgently needed for PV production in cold and unstable weather conditions such as locations like Trondheim.

7.2 Evaluation of the favored model

The XGBoost algorithm stood out a bit from the crowd, surprised on the upside and delivered impressive results for this data set. While the other algorithms performed less well in removing the solar radiation weather parameter, XGBoost had a stable outcomes in the test.

Bibliography

[1] Xuyi Liu, Hao Kong, and Shun Zhang. Can urbanization, renewable energy, and economic growth make environment more eco-friendly in northeast asia? *Renewable Energy*, 169:23–33, 2021.

[2] Matthieu Metayer, Christian Breyer, and Hans-Josef Fell. The projections for the future and quality in the past of the world energy outlook for solar pv and other renewable energy technologies. In *31st European Photovoltaic Solar Energy Conference and Exhibition*, volume 5, 2015.

[3] Spyros Theocharides, George Makrides, George E. Georghiou, and Andreas Kyprianou. Machine learning algorithms for photovoltaic system power output prediction. pages 1–6, 2018.

[4] Alessandro Nocente, Berit Time, Hans Martin Mathisen, Tore Kvande, and Arild Gustavsen. The zeb laboratory: the development of a research tool for future climate adapted zero emission buildings. In *Journal of Physics: Conference Series*, volume 2069, page 012109. IOP Publishing, 2021.

[5] Antonio Luque and Steven Hegedus. *Handbook of photovoltaic science and engineering*. John Wiley & Sons, 2011.

[6] Towhidul Islam. Household level innovation diffusion model of photo-voltaic (PV) solar cells from stated preference data. *Energy Policy*, 65:340–350, February 2014.

[7] Mohamed Abdirazak. Bruken av faseendringsmaterialet i kontorbygg under forskjellige klimatiske forhold. Master's thesis, OsloMet-storbyuniversitetet. Institutt for bygg-og energiteknikk, 2020.

[8] Jørn Stene. Varmepumper for oppvarming og kjøling av bygninger. *SINTEF Energiforskning AS*, 2000.

[9] Winfried Hoffmann. Pv solar electricity industry: Market growth and perspective. *Solar energy materials and solar cells*, 90(18-19):3285–3311, 2006.

[10] Mitsutsune Yamaguchi. Is it possible to achieve global-scale net-zero emissions by 2050? 2021.

[11] Gerry Carrington and Janet Stephenson. The politics of energy scenarios: Are international energy agency and other conservative projections hampering the renewable energy transition? *Energy research & social science*, 46:103–113, 2018.

[12] SK Mohiddin, Dharmappa Barki, Ravi Shankar DVB, and Kiran Kumar. Sustainable framework for global solar exim as a stimulus to supply value chain in india. In *2021 IEEE 48th Photovoltaic Specialists Conference (PVSC)*, pages 1984–1986. IEEE, 2021.

[13] Yongping Zhai. The pandemic may break value chains, but solar energy can still shine.

[14] Manajit Sengupta, Yu Xie, Anthony Lopez, Aron Habte, Galen Maclaurin, and James Shelby. The national solar radiation data base (nsrdb). *Renewable and sustainable energy reviews*, 89:51–60, 2018.

[15] Mudit Kapoor and Rahul Dev Garg. Evaluation of optimum pv tilt angle with generated and predicted solar electric data using geospatial open source software in cloud environment. *Sādhanā*, 46(2):1–14, 2021.

[16] Pio C Lobo. An electrically compensated radiometer. *Solar Energy*, 36(3):207–216, 1986.

[17] MR Nugraha and A Adriansyah. Optimization of sensor model for solar radiation measurement with a pyranometer. In *IOP Conference Series: Earth and Environmental Science*, volume 739, page 012080. IOP Publishing, 2021.

[18] Yashwant Kashyap, Ankit Bansal, Anil Kumar Sao, and Annette Hammer. Model for estimation of global horizontal irradiance in the presence of dust, fog, and clouds. *IEEE Transactions on Geoscience and Remote Sensing*, 56(12):7030–7037, 2018.

[19] Seiji Kato, Thomas P Ackerman, Eugene E Clothiaux, James H Mather, Gerald G Mace, Marvin L Wesely, Frank Murcray, and Joseph Michalsky. Uncertainties in modeled and measured clear-sky surface shortwave irradiances. *Journal of Geophysical Research: Atmospheres*, 102(D22):25881–25898, 1997.

[20] S Oyelami, NA Azeez, SA Adedigba, OJ Akinola, and RM Ajayi. A pyranometer for solar radiation measurement-review. *Adeleke University Journal of Engineering and Technology*, 3(1):61–68, 2020.

[21] Qiang Ji, S-C Tsay, KM Lau, RA Hansell, JJ Butler, and JW Cooper. A novel nonintrusive method to resolve the thermal dome effect of pyranometers: Radiometric calibration and implications. *Journal of Geophysical Research: Atmospheres*, 116(D24), 2011.

[22] AW Van Herwaarden, DC Van Duyn, BW Van Oudheusden, and PM Sarro. Integrated thermopile sensors. *Sensors and Actuators A: Physical*, 22(1-3):621–630, 1990.

[23] Paul SP Cowpertwait and Andrew V Metcalfe. *Introductory time series with R*. Springer Science & Business Media, 2009.

[24] Björn Wolff. *Support vector regression for solar power prediction*. PhD thesis, Universität Oldenburg, 2017.

[25] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*, 2018.

[26] Binh Thai Pham, Tuan-Anh Hoang, Duc-Manh Nguyen, Dieu Tien Bui, et al. Prediction of shear strength of soft soil using machine learning methods. *Catena*, 166:181–191, 2018.

[27] Ferenc Moksony and Rita Heged. Small is beautiful. the use and interpretation of r2 in social research. *Szociológiai Szemle, Special issue*, pages 130–138, 1990.

[28] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[29] Lars Kotthoff, Ian P Gent, and Ian Miguel. An evaluation of machine learning in algorithm selection for search problems. *Ai Communications*, 25(3):257–270, 2012.

[30] Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016.

[31] Jaime Lynn Speiser, Michael E Miller, Janet Tooze, and Edward Ip. A comparison of random forest variable selection methods for classification prediction modeling. *Expert systems with applications*, 134:93–101, 2019.

[32] Tae-Hwy Lee, Aman Ullah, and Ran Wang. Bootstrap aggregating and random forest. In *Macroeconomic Forecasting in the Era of Big Data*, pages 389–429. Springer, 2020.

[33] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[34] Tin Kam Ho. A data complexity analysis of comparative advantages of decision forest constructors. *Pattern Analysis & Applications*, 5(2):102–112, 2002.

[35] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[36] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[37] Lu Ye, Saadya Fahad Jabbar, Musaddak M Abdul Zahra, and Mou Leong Tan. Bayesian regularized neural network model development for predicting daily rainfall from sea level pressure data: Investigation on solving complex hydrology problem. *Complexity*, 2021, 2021.

[38] SARITHA DOPPALAPUDI, SRILATHA BADDURI, NAVYA BOMMU, SRI LAVANYA KAPAROUTHU, and N MD JUBAIR BASHA. A model for automated bug classification using machine learning. 2022.

[39] Tamilarasan Thiyagarajan, Suganyadevi Mv, A Karuppa Samy, and A Venkadesan. Performance investigation of svr for evaluating voltage stability margin in a power utility. In *2021 IEEE International Power and Renewable Energy Conference (IPRECON)*, pages 1–4. IEEE, 2021.

[40] E Kakaei Lafdani, A Moghaddam Nia, and A Ahmadi. Daily suspended sediment load prediction using artificial neural networks and support vector machines. *Journal of Hydrology*, 478:50–62, 2013.

[41] C Deng, SQ Xie, J Wu, and XY Shao. Position error compensation of semi-closed loop servo system using support vector regression and fuzzy pid control. *The International Journal of Advanced Manufacturing Technology*, 71(5):887–898, 2014.

[42] Md Shah Alam, Nahid Sultana, and SM Zakir Hossain. Bayesian optimization algorithm based support vector regression analysis for estimation of shear capacity of frp reinforced concrete members. *Applied Soft Computing*, 105:107281, 2021.

[43] Pijush Samui. Slope stability analysis: a support vector machine approach. *Environmental Geology*, 56(2):255–267, 2008.

[44] David J Olive. Multiple linear regression. In *Linear regression*, pages 17–83. Springer, 2017.

[45] Nhlanhla Mbuli, Malusi Mathonsi, Modisane Seitshiro, and Jan-Harm C Pretorius. Decomposition forecasting methods: A review of applications in power systems. *Energy Reports*, pages 298–306, 2020.

[46] Jose J Salazar, Lean Garland, Jesus Ochoa, and Michael J Pyrcz. Fair train-test split in machine learning: Mitigating spatial autocorrelation for improved prediction accuracy. *Journal of Petroleum Science and Engineering*, pages 2–4, 2022.

[47] Daniel Berrar. Cross-validation., 2019.

[48] Zele Zhang. Influence of special weather on output of pv system. In *IOP Conference Series: Earth and Environmental Science*, volume 108. IOP Publishing, 2018.

A Appendices

A.1 Libraries

Since the assignment is to design a system that will predict power production based on the weather forecast, we were given room to choose which programming language we wanted. The group has previously used Python for other subjects in the course of study, so it was a natural and preferred choice for a programming language. With Python installed, we went further with the Anaconda platform, that contains *Jupyter Notebook*, which is an excellent tool for data processing and manipulations.

When applying the selected algorithms, you have to download specific libraries and these installations can be done on ***CMD.exe Promt:***

Random Forest Regressor
[conda install -c r r-randomforest]

Support Vector Machine Regressor
[conda install -c conda-forge libsvm]

XGBoost
[conda install -c anaconda py-xgboost]

Streamlit
[conda install streamlit]

A.2 Packages

- *pandas*
Pandas is a Python library, that provides flexible and multiple functions for data manipulations, analysing, and visualising and among other things such as reading a CSV file into a Dataframe. *[import pandas as pd]*

- *numpy*
When working with numerical data, Numpy module is preferred since it supplies Array objects. Numpy functions are placed in one specific location in the storage space, hence be control very easy. *[import numpy as np]*

- *matplotlib.pyplot*
Matplotlib.pyplot are used for producing graphics and visualizations, and it makes plotting more effortless. When executing plots with Matplotlib.pyplot, one must extend the package, since it is not included in default. *[import matplotlib.pyplot as plt]*

- %matplotlib.inline
This function is called for a "magic function" since it gives the graphs the possibility to be in the notebook.

- seaborn
Seaborn is a statistical data visualization tool, for plotting the dataset. Seaborn is more flexible than the matplotlib.pyplot function, and the reason for that is it serve more function such as lineplots or scatterplots. *[from seaborn import scatterplot]*

- sklearn.preprocessing
This package generates several functions for transforming and manipulating data for a more optimized preferred fit. sklearn.preprocessing can be very useful when dealing with machine learning problems since the algorithms favours small and clean numerical values.

- sklearn.metrics
This module gives the opportunity to take advantage of several loss, score and efficiency measurements to check how well the model is performing. Here we can evoke MSE, MAE and RMSE metrics.

- test_train_split
test_train_split is a command, that split arrays and list into two seperate sets, training set and the testing set. This command is advantageous in different algorithms used in this thesis. This function is provided in the Scikit-learn package. *[from sklearn.model_selection import train_test_split]*

- sklearn.svm
This module allows us to apply Support Vector Machine algorithm on the raw data. *[from sklearn.svm import SVR]*

- RandomForestRegressor
RandomForestRegressor is a package delivered from Scikit-learn, and it creates an algorithm based on RandomForest. *[from sklearn.ensemble import RandomForestRegressor]*

- LinearRegression
LinearRegression is used to predict a value based on another value, which makes the variable we want to predict *dependent*.

- KFold
This is a cross validator, that provides trained or tested indices to spilt the data in a given set. *from sklearn*

- GridSearchCV
Implemets a *fit* and *score* method. The parameters of the estimator is used to run through all the different parameters in the parameter grid to make the best combinations, in terms of

score.

- xgboost
Xgboost is gradient boosting library that implements ML algorithms under the Gradient Boosting framework.

- XGBRegressor
XGBRegressor is a class of the *XGBoost package* and classifies the order of features based on importance for the prediction.

A.3 Functions

- read_csv
This is a function that is important for the pandas frame and is brought in to load a CSV file and the parameters can be customized to get a better output.

- random_state
This function is an integer value that is used to set the seed for the random generator so that we can ensure that the results can be reproduced. The value is an implication for the selection of the random combination between the train and test.

- index_col
It says which column that should act as an index, and is given as an *int, str, False, optional, default* or a sequence of int/str.

- parse_dates
This function is helps to to convince pandas to specify real date time types. It reads the data column correctly, even though the loading data from the CSV file is represented as an object.

- test_size
This function defines the size of a test set, and is given as an *int* or a *float*

- n_splits
This parameter determines how many different validation and training sets we will be creating.

- shuffle
This function take a sequence and reorganize the position of its items.

- estimator
A function in ML which persists and retrieve the ML models and data sets. This is an object that could be a regressor a classifier.

- param_grid
This function makes the parameter to explore as a dictionary, and is useful to avoid useless parameter combinations that have no impact. *param_grid* will give a grid of parameters with a discrete number of values for each, and enable to search over any sequence.

- verbose
Verbose is a term for producing lots of logging output/information, that allow us to write regular expressions that that look better and more readable by allowing us to separate for example comments and logical commands.

- fit_transform
This method are method of class ***sklearn.preprocessing.StandardScaler()*** and used to scale and standardising the parameters of the training and data set. This method will calculate the mean and variance of each features presented in the data, and then transforming all by using the given mean and variance.

- cv
Stands for cross-validation, and is the number of cv folds that held each combination of parameters.

- seed
This method is used to initialize thee random number generator, which needs to start with a seed value to generate a pseudo-random number. This function is needed to save the state of a random function, and when there is no previous value it will then use current system time.

- scoring
This function measures the accuracy if the model compared to the training data, and is used as a list or tuple of unique strings if it represents multiple scores. If it only represent a single score, it can then be used as a single string, and used to return/recall a single value. It is used to rank the results.

- intercept_
This function represents the β_0 value, and shows us where the regression line crosses the y axis, in other words, when $f(x=0)$.

- coef_
This function represents the coeffient β_1 and determines an of the slope from the estimated linear regression line.

- best_params_
This is a dictionary where a parameter setting gives the best result on the hold out data.

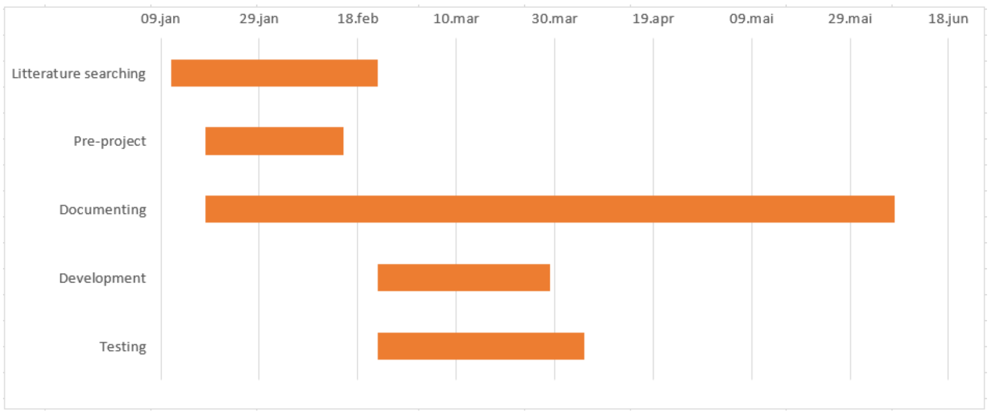
- best_score_
This function describe the mean of the cross-validated score, this is done by observing the a

repeated process of parameter combinations.

- transform
This function returns a function, that is self produced, with transformed values that has the same dataframe.

- inverse_transform
This method will transform its label to the original encoding, meaning it will return an inverse transformed value.

B Gantt



Gantt chart for the project

C Meeting Logs

C.1 Temporal record of meetings

03.02.2022 - Bachelor Information Meeting

Discussion of the process and setup of the thesis. The Presav project, and explain in details. Deadlines for submission of documentation. Introduction to the process and the sessions to help with writing the thesis....

10.02.2022

Met with supervisors Jay and Berhane to discuss the project. Actions:

1. decide our focus point in the thesis, what we should bring to the table as a group and what impact we have towards the whole PRESAV project.
2. discussing input/output in forecast, and data. Regression etc.
3. draft agreements by next week. simple plots in power production. Write on pre-project.

18.02.2022

Met with supervisor Berhane to discuss the project. Actions:

1. discuss working on the algorithms, finding which parameters that are suitable.
2. decide how we should solve the problem technically.
3. draft agreements by next week. Come up with questions around the topic so the meeting will get more productive.

25.02.2022

Met with supervisors Jay and Berhane to discuss the project. Actions:

1. discuss feedback on the pre-project. Improvement in the technical part. Formulate/understand our task even better. Add and remove different subtopics in the pre-project. Find out in the data: what is the interval, duration, explanations about the dataset. Example. solar radiation. Include where we get data from in the pre-project. discuss what we found in terms of plotting the parameters:temperature, solar duration and GHI, and we found the score with SVM: support vector machine. Before tuning and after grid-searching.
2. install development environment
3. draft agreements by next week: Start to understand the statistical approach and random forest. Try to visualize our plots, in terms of its maximum or minimum. Boxplot, or candlestick. Maybe come with a presentation of next week's work and present it to the supervisors.

04.03.2022

Met with supervisors Jay and Berhane to discuss the project. Actions:

1. discuss machine learning algorithms, and compare them with statistical methods. See

which algorithm that provides the best results. Feedback from Jay about the thesis structure: literature review vs theory, discuss our information source and how we should write in the first chapters. We talk about statistics, Auto-regression. We also talk about how we should structure chapter one and chapter two.
2. install development environment
3. draft agreements by next week: Make a time plan, due to an unexpected incident and try to form a working schedule, of what and when we should be done with each task. Try to plan week-wise in terms of progress. How we pre-process our data, how we collect the data and how should we compare. Focus on the methods and finalizing.

01.04.2022

Met with supervisor Berhane to discuss the project. Actions:

1. Discuss our strategy of work in the coming weeks.
2. Which and how many algorithms should we try, and should present each algorithm.
3. draft agreements by next week: Try to implements the given data set with Random Forest, ARIMA, SARIMA and XGBoost. Get a broader perspective on how each algorithm works.

08.04.2022

Met with supervisor Berhane to discuss the project. Actions:

1. Discuss about the introduction and PRESAV chapter, on how it should be presented.
2. install development environment
3. draft agreements by next week: What to include in the report, and which elements of PRESAV should be explained. Start writing!

22.04.2022

Met with supervisors Berhane and Jay to discuss the project. Actions:

1. discuss what type of sources to use in the report. Examples: research papers, published articles and reliable wiki-sites.
2. ARIMA and SARIMA are dropped because it is hard to implement these algorithms in our forecast due to our dataset parameters.
3. draft agreements by next week: Continue working on the algorithms,

06.05.2022

Met with supervisor Berhane to discuss the project. Actions:

1. describe what we have done so far. Writing theory, and finishing each algorithm used in the forecast.
2. Discussed our report, how we should continue writing and how we should present our results.
3. draft agreements by next week.etc. Future work, report layouts, and technical writing.

20.05.2022

Met with supervisor Berhane to discuss the project. Actions:

1. describe what we have done so far. Writing theory, and finishing each algorithm used

in the forecast.(SVR, RF, MLR and XGBoost) Discuss the idea of creating a web page for illustrating RF and XGBoost algorithm, where we can tune the parameters.
2. Discussed our report, how we should continue writing and how we should present our results.
3. draft agreements by next week.etc. Future work, report layouts, and technical writing.

27.05.2022

Met with supervisor Berhane to discuss the project. Actions:

1. Show the website with XGboost, Random Forest and the parameters and shape etc.
2. Berhane commenting on the graphs, we need to label the x,y-axis, and what we measure in the plot. feedback on adjusting parameters. Suggest how we should present the result.
3. draft agreements by next week.etc. Future work, report layouts, and technical writing.

03.06.2022

Met with supervisor Berhane to discuss the project. Actions:

1. Talk about structure and how the report should be presented, in terms of chapters, headers, and what type of information should be considered as important explanations in the theory part .
2. Berhane commenting on the graphs, how the result section should be in the report and how we should present the values.
3. Will come with feedback, and how we could correct the writing of the thesis. Prepare a presentation and an introducing article with a cover.

08.06.2022

Met with supervisor Berhane to discuss the project. Actions:

1. Berhane's feedback was elaborated, as well as some thoughts on how the poster should be displayed.
2. Information about further execution of presentation

D Python codes

D.1 Algorithms

Random Forest

```
# Importing the necessary libraries for retrieving the CSV-file and
    plotting graphs
import pandas as pd
import numpy as np
from matplotlib import pyplot
import matplotlib.pyplot as plt

# Importing different libraries for various metrics calculations
from sklearn import metrics
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# Importing the library that enables the use of Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor

# Importing the scikit-learn's train_test_split function
from sklearn.model_selection import train_test_split

# Importing the Cross-validation method Kfold
from sklearn.model_selection import KFold

# Importing the library needed for grid searching
from sklearn.model_selection import GridSearchCV

# Using pandas to retrieve the file
# Setting 'Timestamp' column as index of the dataframe
df_rf  = pd.read_csv("df.csv", index_col = 'Timestamp')

# Testing with all features
# Assigning [x] to only containing desired features and [y] to
    target-variable
x_all = df_rf.iloc[:,:8].values
y_all = df_rf.iloc[:,[-1]].values

# Splitting dataframe into train/test sets, 90% training and 10% testing
x_train, x_test,
y_train, y_test = train_test_split(x_all, y_all, test_size=0.1,
    random_state=42)

# Looking at the shapes
x_train.shape ,y_train.shape ,x_test.shape ,y_test.shape

# Instantiate the RF model and then fitting it for X and Y trainings
rf_all = RandomForestRegressor()
rf_all.fit(x_train, y_train);
```

```
# Predicting the test sets features
pred_rf_all = rf_all.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_all)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_all)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_all))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_all)))

# Assigning the metric scores to variables for later use
mae_score_all = "{:.3f}".format(mean_absolute_error(y_test, pred_rf_all))
mse_score_all = "{:.3f}".format(mean_squared_error(y_test, pred_rf_all))
rmse_score_all =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_all)))
r2_score_all = "{:.4f}".format(r2_score(y_test,pred_rf_all))

# Testing with all features optimized
# Instantiate the model
rf_all_optimized = RandomForestRegressor()

# Inserting the hyper parameters for tuning
param_grid_all_optimized = {'n_estimators': [50,100,150], 'max_depth':
    [4,5,6], 'max_features':[1,2,3]}
# Applying 5-fold
kf_all_optimized = KFold(n_splits = 5, shuffle=True, random_state=42)

# Activating the grid searching and 5-fold to find the best combination
grid_search_all_optimized = GridSearchCV(RandomForestRegressor(),
    param_grid_all_optimized, cv=kf_all_optimized, n_jobs=-1)

# Fitting the model with the best found combination of hyper parameters
grid_search_all_optimized.fit(x_train, y_train);

# Printing the most optimal hyper parameters
print('Best Parameter: ', grid_search_all_optimized.best_params_)

# Printing the training score after cross-validation and hyper tuning
print('Training Score: ',
    "{:.3f}".format(grid_search_all_optimized.best_score_))

# Printing the best estimators
print('Best Estimator: ', grid_search_all_optimized.best_estimator_)

# Predicting the test sets features
pred_rf_all_optimized = grid_search_all_optimized.predict(x_test)

# Printing the testing score after cross-validation and hyper tuning
print('Test Score: ',
    "{:.3f}".format(grid_search_all_optimized.score(x_test, y_test)))
```

```
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test,
    pred_rf_all_optimized)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test,
    pred_rf_all_optimized)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_all_optimized))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test,pred_rf_all_optimized)))


mae_score_all_optimized = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_all_optimized))
mse_score_all_optimized = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_all_optimized))
rmse_score_all_optimized =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_all_optimized)))
r2_score_all_optimized =
    "{:.4f}".format(r2_score(y_test,pred_rf_all_optimized))

# Using the same steps but this time "Temperature" is excluded
# Every feature except Temperature
x_w_temp = df_rf.iloc[:,1:8].values
y_w_temp = df_rf.iloc[:,[-1]].values

x_train, x_test, y_train, y_test = train_test_split(x_w_temp, y_w_temp,
    test_size=0.1, random_state=42)

rf_w_temp = RandomForestRegressor()
param_grid_w_temp = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_temp = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_temp = GridSearchCV(RandomForestRegressor(),
    param_grid_w_temp, cv=kf_w_temp, n_jobs=-1)
grid_search_w_temp.fit(x_train, y_train);

print('Best Parameter: ', grid_search_w_temp.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_temp.best_score_))
print('Best Estimator: ', grid_search_w_temp.best_estimator_)
pred_rf_w_temp = grid_search_w_temp.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_temp.score(x_test,
    y_test)))
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_temp)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_temp)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_temp))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_temp)))


mae_score_w_temp = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_temp))
mse_score_w_temp = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_temp))
rmse_score_w_temp =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_temp)))
r2_score_w_temp = "{:.4f}".format(r2_score(y_test,pred_rf_w_temp))

# Every feature except Dew point
x_w_dew = df_rf.iloc[:,[0,2,3,4,5,6,7]].values
y_w_dew = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_dew, y_w_dew,
    test_size=0.1, random_state=42)
rf_w_dew = RandomForestRegressor()
param_grid_w_dew = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_dew = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_dew = GridSearchCV(RandomForestRegressor(),
    param_grid_w_dew, cv=kf_w_dew, n_jobs=-1)
grid_search_w_dew.fit(x_train, y_train);
print('Best Parameter: ', grid_search_w_dew.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_dew.best_score_))
print('Best Estimator: ', grid_search_w_dew.best_estimator_)
pred_rf_w_dew = grid_search_w_dew.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_dew.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_dew)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_dew)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_dew))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_dew)))

mae_score_w_dew = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_dew))
mse_score_w_dew = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_dew))
rmse_score_w_dew =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_dew)))
r2_score_w_dew = "{:.4f}".format(r2_score(y_test,pred_rf_w_dew))

# Every feature except Wind speed
```

```
x_w_winds = df_rf.iloc[:,[0,1,3,4,5,6,7]].values
y_w_winds = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_winds, y_w_winds,
    test_size=0.1, random_state=42)
rf_w_winds = RandomForestRegressor()
param_grid_w_winds = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_winds = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_winds = GridSearchCV(RandomForestRegressor(),
    param_grid_w_winds, cv=kf_w_winds, n_jobs=-1)
grid_search_w_winds.fit(x_train, y_train);
print('Best Parameter: ', grid_search_w_winds.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_winds.best_score_))
print('Best Estimator: ', grid_search_w_winds.best_estimator_)
pred_rf_w_winds = grid_search_w_winds.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_winds.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_winds)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_winds)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_winds))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_winds)))

mae_score_w_winds = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_winds))
mse_score_w_winds = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_winds))
rmse_score_w_winds =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_winds)))
r2_score_w_winds = "{:.4f}".format(r2_score(y_test,pred_rf_w_winds))

# Every feature except Wind direction
x_w_windd = df_rf.iloc[:,[0,1,2,4,5,6,7]].values
y_w_windd = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_windd, y_w_windd,
    test_size=0.1, random_state=42)
rf_w_windd = RandomForestRegressor()
param_grid_w_windd = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_windd = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_windd = GridSearchCV(RandomForestRegressor(),
    param_grid_w_windd, cv=kf_w_windd, n_jobs=-1)
grid_search_w_windd.fit(x_train, y_train);
print('Best Parameter: ', grid_search_w_windd.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_windd.best_score_))
print('Best Estimator: ', grid_search_w_windd.best_estimator_)
pred_rf_w_windd = grid_search_w_windd.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_windd.score(x_test,
    y_test)))
```

```
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_windd)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_windd)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_windd))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_windd)))

mae_score_w_windd = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_windd))
mse_score_w_windd = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_windd))
rmse_score_w_windd =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_windd)))
r2_score_w_windd = "{:.4f}".format(r2_score(y_test,pred_rf_w_windd))

# Every features except Relative humidity
x_w_relh = df_rf.iloc[:,[0,1,2,3,5,6,7]].values
y_w_relh = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_relh, y_w_relh,
    test_size=0.1, random_state=42)
rf_w_relh = RandomForestRegressor()
param_grid_w_relh = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_relh = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_relh = GridSearchCV(RandomForestRegressor(),
    param_grid_w_relh, cv=kf_w_relh, n_jobs=-1)
grid_search_w_relh.fit(x_train, y_train);
print('Best Parameter: ', grid_search_w_relh.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_relh.best_score_))
print('Best Estimator: ', grid_search_w_relh.best_estimator_)
pred_rf_w_relh = grid_search_w_relh.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_relh.score(x_test,
    y_test)))
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_relh)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_relh)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_relh))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_relh)))

mae_score_w_relh = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_relh))
mse_score_w_relh = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_relh))
```

```
rmse_score_w_relh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_relh)))
r2_score_w_relh = "{:.4f}".format(r2_score(y_test,pred_rf_w_relh))

# Every features except Absolute humidity
x_w_absh = df_rf.iloc[:,[0,1,2,3,4,6,7]].values
y_w_absh = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_absh, y_w_absh,
    test_size=0.1, random_state=42)
rf_w_absh = RandomForestRegressor()
param_grid_w_absh = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_absh = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_absh = GridSearchCV(RandomForestRegressor(),
    param_grid_w_absh, cv=kf_w_absh, n_jobs=-1)
grid_search_w_absh.fit(x_train, y_train);
print('Best Parameter: ', grid_search_w_absh.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_absh.best_score_))
print('Best Estimator: ', grid_search_w_absh.best_estimator_)
pred_rf_w_absh = grid_search_w_absh.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_absh.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_absh)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_absh)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_absh))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_absh)))

mae_score_w_absh = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_absh))
mse_score_w_absh = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_absh))
rmse_score_w_absh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_absh)))
r2_score_w_absh = "{:.4f}".format(r2_score(y_test,pred_rf_w_absh))

# Every features except Barometric pressure
x_w_baro = df_rf.iloc[:,[0,1,2,3,4,5,7]].values
y_w_baro = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_baro, y_w_baro,
    test_size=0.1, random_state=42)
rf_w_baro = RandomForestRegressor()
param_grid_w_baro = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_baro = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_baro = GridSearchCV(RandomForestRegressor(),
    param_grid_w_baro, cv=kf_w_baro, n_jobs=-1)
grid_search_w_baro.fit(x_train, y_train);
```

```
print('Best Parameter: ', grid_search_w_baro.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_baro.best_score_))
print('Best Estimator: ', grid_search_w_baro.best_estimator_)
pred_rf_w_baro = grid_search_w_baro.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_baro.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_baro)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_baro)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_baro))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_baro)))

mae_score_w_baro = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_baro))
mse_score_w_baro = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_baro))
rmse_score_w_baro =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_baro)))
r2_score_w_baro = "{:.4f}".format(r2_score(y_test,pred_rf_w_baro))

# Every features except Solar radiation
x_w_solar = df_rf.iloc[:,0:7].values
y_w_solar = df_rf.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_solar, y_w_solar,
    test_size=0.1, random_state=42)
rf_w_solar = RandomForestRegressor()
param_grid_w_solar = {'n_estimators': [50,100,150], 'max_depth': [4,5,6],
    'max_features':[1,2,3]}
kf_w_solar = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_solar = GridSearchCV(RandomForestRegressor(),
    param_grid_w_solar, cv=kf_w_solar, n_jobs=-1)
grid_search_w_solar.fit(x_train, y_train);
print('Best Parameter: ', grid_search_w_solar.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_solar.best_score_))
grid_search_w_solar.cv_results_;
print('Best Estimator: ', grid_search_w_solar.best_estimator_)
pred_rf_w_solar = grid_search_w_solar.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_solar.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_rf_w_solar)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_rf_w_solar)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_solar))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_rf_w_solar)))

mae_score_w_solar = "{:.3f}".format(mean_absolute_error(y_test,
    pred_rf_w_solar))
mse_score_w_solar = "{:.3f}".format(mean_squared_error(y_test,
    pred_rf_w_solar))
rmse_score_w_solar =
    "{:.4f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_rf_w_solar)))
r2_score_w_solar = "{:.4f}".format(r2_score(y_test,pred_rf_w_solar))

# Making a new dataframe comparing all predicted values up against the
    reference point y_test
predictions_all2 = pd.DataFrame({ 'y_test':y_test[:,0],
    'Temperature':pred_rf_w_temp, 'Dew point':pred_rf_w_dew,'Wind
    speed':pred_rf_w_winds,'Wind direction':pred_rf_w_windd, 'Relative
    humidity':pred_rf_w_relh,'Absolute
    humidity':pred_rf_w_absh,'Barometric pressure':pred_rf_w_baro,'Solar
    radiation':pred_rf_w_solar, 'All':pred_rf_all, 'All
    optimized':pred_rf_all_optimized})
predictions_all2.head(15)

# Creating a dataframe with MAE scores
scores_mae = {
    'All': mae_score_all,
    'All optimized': mae_score_all_optimized,
    'Without temperature': mae_score_w_temp,
    'Without dew point': mae_score_w_dew,
    'Without wind speed': mae_score_w_winds,
    'Without wind direction': mae_score_w_windd,
    'Without relative humidity': mae_score_w_relh,
    'Without absolute humidity': mae_score_w_absh,
    'Without barometric pressure': mae_score_w_baro,
    'Without solar radiation': mae_score_w_solar,
}

# Creating a dataframe with MSE scores
scores_mse = {
    'All': mse_score_all,
    'All optimized': mse_score_all_optimized,
    'Without temperature': mse_score_w_temp,
    'Without dew point': mse_score_w_dew,
    'Without wind speed': mse_score_w_winds,
    'Without wind direction': mse_score_w_windd,
    'Without relative humidity': mse_score_w_relh,
    'Without absolute humidity': mse_score_w_absh,
    'Without barometric pressure': mse_score_w_baro,
    'Without solar radiation': mse_score_w_solar,
}

# Creating a dataframe with RMSE scores
scores_rmse = {
    'All': rmse_score_all,
    'All optimized': rmse_score_all_optimized,
    'Without temperature': rmse_score_w_temp,
    'Without dew point': rmse_score_w_dew,
    'Without wind speed': rmse_score_w_winds,
    'Without wind direction': rmse_score_w_windd,
    'Without relative humidity': rmse_score_w_relh,
    'Without absolute humidity': rmse_score_w_absh,
    'Without barometric pressure': rmse_score_w_baro,
    'Without solar radiation': rmse_score_w_solar,
}

# Creating a dataframe with R2 scores
scores_r2 = {
    'All': r2_score_all,
    'All optimized': r2_score_all_optimized,
    'Without temperature': r2_score_w_temp,
    'Without dew point': r2_score_w_dew,
    'Without wind speed': r2_score_w_winds,
    'Without wind direction': r2_score_w_windd,
    'Without relative humidity': r2_score_w_relh,
    'Without absolute humidity': r2_score_w_absh,
    'Without barometric pressure': r2_score_w_baro,
    'Without solar radiation': r2_score_w_solar,
}
```

XGBoost

```
# Importing the library that enables the use of XGboost
import xgboost
import xgboost as xgb

df_xg  = pd.read_csv("df.csv", index_col = 'Timestamp')

x_all = df_xg.iloc[:,:8].values
y_all = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_all, y_all,
    test_size=0.1, random_state=42)
xgb_reg = xgb.XGBRegressor(objective='reg:squarederror',seed=123,
    learning_rate = 0.1);
xgb_reg.fit(x_train, y_train);
y_pred = xgb_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, y_pred)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, y_pred)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test, y_pred))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,y_pred)))

# Assigning the metric scores to variables for later use
mae_score_all = "{:.3f}".format(mean_absolute_error(y_test, y_pred))
mse_score_all = "{:.3f}".format(mean_squared_error(y_test, y_pred))
rmse_score_all =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
r2_score_all = "{:.4f}".format(r2_score(y_test,y_pred))

# Every parameters with optimizations
# Inserting the hyper parameters for tuning
params_all = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}

kf_all = KFold(n_splits = 5, shuffle=True, random_state=42)
xgb_reg_all = xgb.XGBRegressor(seed = 20)
grid_search_all = GridSearchCV(estimator=xgb_reg_all,
                 param_grid=params_all,
                  cv = kf_all,
                 scoring='neg_mean_squared_error',
                 verbose=1)

grid_search_all.fit(x_all, y_all);
print('Best Parameter: ', grid_search_all.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_all.best_score_))
pred_xg_all = grid_search_all.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_all.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_all)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_all)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_all))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_all)))

mae_score_all_optimized = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_all))
mse_score_all_optimized = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_all))
rmse_score_all_optimized =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_all)))
r2_score_all_optimized = "{:.4f}".format(r2_score(y_test,pred_xg_all))

# Every parameters optimized without temperture
x_w_temp = df_xg.iloc[:,1:8].values
y_w_temp = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_temp, y_w_temp,
    test_size=0.1, random_state=42)
xgb_w_temp = xgb.XGBRegressor(seed = 20)
params_w_temp = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}

kf_w_temp = KFold(n_splits = 5, shuffle=True, random_state=42)
```

```
grid_search_w_temp = GridSearchCV(estimator=xgb_w_temp,
                 param_grid=params_w_temp,
                 cv = kf_w_temp,
                 scoring='neg_mean_squared_error',
                 verbose=1)
grid_search_w_temp.fit(x_w_temp, y_w_temp);
print('Best Parameter: ', grid_search_w_temp.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_temp.best_score_))
pred_xg_w_temp = grid_search_w_temp.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_temp.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_temp)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_temp)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_temp))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_temp)))

mae_score_w_temp = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_temp))
mse_score_w_temp = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_temp))
rmse_score_w_temp =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_temp)))
r2_score_w_temp = "{:.4f}".format(r2_score(y_test,pred_xg_w_temp))

# Every parameters without Dew point
x_w_dew = df_xg.iloc[:,[0,2,3,4,5,6,7]].values
y_w_dew = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_dew, y_w_dew,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_dew = xgb.XGBRegressor(seed = 20)
params_w_dew = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}

kf_w_dew = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_dew = GridSearchCV(estimator=xgb_w_dew,
                 param_grid=params_w_dew,
                 scoring='neg_mean_squared_error',
                 cv = kf_w_dew,
                 verbose=1)
grid_search_w_dew.fit(x_w_dew, y_w_dew);
print('Best Parameter: ', grid_search_w_dew.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_dew.best_score_))
pred_xg_w_dew = grid_search_w_dew.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_dew.score(x_test,
    y_test)))
```

```
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_dew)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_dew)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_dew))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_dew)))

mae_score_w_dew = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_dew))
mse_score_w_dew = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_dew))
rmse_score_w_dew =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_dew)))
r2_score_w_dew = "{:.4f}".format(r2_score(y_test,pred_xg_w_dew))

# Every parameters without wind speed
x_w_winds = df_xg.iloc[:,[0,1,3,4,5,6,7]].values
y_w_winds = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_winds, y_w_winds,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_winds = xgb.XGBRegressor(seed = 20)
params_w_winds = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}
kf_w_winds = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_winds = GridSearchCV(estimator=xgb_w_winds,
                 param_grid=params_w_winds,
                 scoring='neg_mean_squared_error',
                 cv = kf_w_winds,
                 verbose=1)
grid_search_w_winds.fit(x_w_winds, y_w_winds);
print('Best Parameter: ', grid_search_w_winds.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_winds.best_score_))
pred_xg_w_winds = grid_search_w_winds.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_winds.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_winds)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_winds)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_winds))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_winds)))
```

```
mae_score_w_winds = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_winds))
mse_score_w_winds = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_winds))
rmse_score_w_winds =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_winds)))
r2_score_w_winds = "{:.4f}".format(r2_score(y_test,pred_xg_w_winds))

# Every parameters without wind direction
x_w_windd = df_xg.iloc[:,[0,1,2,4,5,6,7]].values
y_w_windd = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_windd, y_w_windd,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_windd = xgb.XGBRegressor(seed = 20)
params_w_windd = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}
kf_w_windd = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_windd = GridSearchCV(estimator=xgb_w_windd,
                 param_grid=params_w_windd,
                 scoring='neg_mean_squared_error',
                  cv = kf_w_windd,
                 verbose=1)
grid_search_w_windd.fit(x_w_windd, y_w_windd);
print('Best Parameter: ', grid_search_w_windd.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_windd.best_score_))
pred_xg_w_windd = grid_search_w_windd.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_windd.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_windd)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_windd)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_windd))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_windd)))

mae_score_w_windd = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_windd))
mse_score_w_windd = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_windd))
rmse_score_w_windd =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_windd)))
r2_score_w_windd = "{:.4f}".format(r2_score(y_test,pred_xg_w_windd))

# Every parameters without relative humidity
x_w_relh = df_xg.iloc[:,[0,1,2,3,5,6,7]].values
```

```
y_w_relh = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_relh, y_w_relh,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_relh = xgb.XGBRegressor(seed = 20)
params_w_relh = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}

kf_w_relh = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_relh = GridSearchCV(estimator=xgb_w_relh,
                 param_grid=params_w_relh,
                 scoring='neg_mean_squared_error',
                 cv = kf_w_relh,
                 verbose=1)

grid_search_w_relh.fit(x_w_relh, y_w_relh);
print('Best Parameter: ', grid_search_w_relh.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_relh.best_score_))
pred_xg_w_relh = grid_search_w_relh.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_relh.score(x_test,
    y_test)))
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_relh)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_relh)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_relh))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_relh)))

mae_score_w_relh = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_relh))
mse_score_w_relh = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_relh))
rmse_score_w_relh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_relh)))
r2_score_w_relh = "{:.4f}".format(r2_score(y_test,pred_xg_w_relh))

# Every parameters without absolute humidity
x_w_absh = df_xg.iloc[:,[0,1,2,3,4,6,7]].values
y_w_absh = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_absh, y_w_absh,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_absh = xgb.XGBRegressor(seed = 20)
params_w_absh = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}

kf_w_absh = KFold(n_splits = 5, shuffle=True, random_state=42)
```

```
grid_search_w_absh = GridSearchCV(estimator=xgb_w_absh,
                 param_grid=params_w_absh,
                 scoring='neg_mean_squared_error',
                 cv = kf_w_absh,
                 verbose=1)

grid_search_w_absh.fit(x_w_absh, y_w_absh);
print('Best Parameter: ', grid_search_w_absh.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_absh.best_score_))
pred_xg_w_absh = grid_search_w_absh.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_absh.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_absh)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_absh)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_absh))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_absh)))

mae_score_w_absh = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_absh))
mse_score_w_absh = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_absh))
rmse_score_w_absh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_absh)))
r2_score_w_absh = "{:.4f}".format(r2_score(y_test,pred_xg_w_absh))

# Every features except Barometric pressure
x_w_baro = df_xg.iloc[:,[0,1,2,3,4,5,7]].values
y_w_baro = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_baro, y_w_baro,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_baro = xgb.XGBRegressor(seed = 20)
params_w_baro = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}

kf_w_baro = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_baro = GridSearchCV(estimator=xgb_w_baro,
                 param_grid=params_w_baro,
                 scoring='neg_mean_squared_error',
                 cv = kf_w_baro,
                 verbose=1)

grid_search_w_baro.fit(x_w_baro, y_w_baro);
print('Best Parameter: ', grid_search_w_baro.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_baro.best_score_))
pred_xg_w_baro = grid_search_w_baro.predict(x_test)
```

```
print('Test Score: ', "{:.3f}".format(grid_search_w_baro.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_baro)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_baro)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_baro))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_baro)))

mae_score_w_baro = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_baro))
mse_score_w_baro = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_baro))
rmse_score_w_baro =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_baro)))
r2_score_w_baro = "{:.4f}".format(r2_score(y_test,pred_xg_w_baro))

# Every parameters without solar radiation
x_w_solar = df_xg.iloc[:,0:7].values
y_w_solar = df_xg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_solar, y_w_solar,
    test_size=0.1, random_state=42)
# Instantiate the model
xgb_w_solar = xgb.XGBRegressor(seed = 20)
params_w_solar = { 'max_depth': [4,5],
          'learning_rate': [0.01, 0.1],
          'n_estimators': [50, 100, 150],
          'colsample_bytree': [0.5,0.6,0.7]}
kf_w_solar = KFold(n_splits = 5, shuffle=True, random_state=42)

grid_search_w_solar = GridSearchCV(estimator=xgb_w_solar,
                 param_grid=params_w_solar,
                 scoring='neg_mean_squared_error',
                 cv = kf_w_solar,
                 verbose=1)

grid_search_w_solar.fit(x_w_solar, y_w_solar);
print('Best Parameter: ', grid_search_w_solar.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_solar.best_score_))
pred_xg_w_solar = grid_search_w_solar.predict(x_test)
print('Test Score: ', "{:.3f}".format(grid_search_w_solar.score(x_test,
    y_test)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_xg_w_solar)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_xg_w_solar)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_solar))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_xg_w_solar)))

mae_score_w_solar = "{:.3f}".format(mean_absolute_error(y_test,
    pred_xg_w_solar))
mse_score_w_solar = "{:.3f}".format(mean_squared_error(y_test,
    pred_xg_w_solar))
rmse_score_w_solar =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_xg_w_solar)))
r2_score_w_solar = "{:.4f}".format(r2_score(y_test,pred_xg_w_solar))
```

Support Vector Regressor

```
# Importing the library that enables the use of Support Vector Regressor
from sklearn.svm import SVR
# Importing the library needed for scaling
from sklearn.preprocessing import StandardScaler

df_svr = pd.read_csv("df.csv",parse_dates=True,index_col = 0)


# Assigning [x] to only containing desired features and [y] to
    target-variable
x = df_svr.iloc[:,:8].values
y = df_svr.iloc[:,[-1]].values

x_all = df_svr.iloc[:,:8].values
y_all = df_svr.iloc[:,[-1]].values

# Testing with "Temperature" removed
x_w_temp = df_svr.iloc[:,1:8].values
y_w_temp = df_svr.iloc[:,[-1]].values

# Testing with "Dew point" removed
x_w_dew = df_svr.iloc[:,[0,2,3,4,5,6,7]].values
y_w_dew = df_svr.iloc[:,[-1]].values

# Testing with "Wind speed" removed
x_w_winds = df_svr.iloc[:,[0,1,3,4,5,6,7]].values
y_w_winds = df_svr.iloc[:,[-1]].values

# Testing with "Wind direction" removed
x_w_windd = df_svr.iloc[:,[0,1,2,4,5,6,7]].values
y_w_windd = df_svr.iloc[:,[-1]].values

# Testing with "Relative humidity" removed
x_w_relh = df_svr.iloc[:,[0,1,2,3,5,6,7]].values
y_w_relh = df_svr.iloc[:,[-1]].values

# Testing with "Absolute humidity" removed
x_w_absh = df_svr.iloc[:,[0,1,2,3,4,6,7]].values
```

```
y_w_absh = df_svr.iloc[:,[-1]].values

# Testing with "Barometric pressure" removed
x_w_baro = df_svr.iloc[:,[0,1,2,3,4,5,7]].values
y_w_baro = df_svr.iloc[:,[-1]].values

# Testing with "Solar radiation" removed
x_w_solar = df_svr.iloc[:,0:7].values
y_w_solar = df_svr.iloc[:,[-1]].values

# Splitting dataframe into train/test sets, 90% training and 10% testing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1,
    random_state=42)

x_train_all, x_test_all, y_train_all, y_test_all = train_test_split(x_all,
    y_all, test_size=0.1, random_state=42)

x_train_w_temp, x_test_w_temp, y_train_w_temp, y_test_w_temp =
    train_test_split(x_w_temp, y_w_temp, test_size=0.1, random_state=42)

x_train_w_dew, x_test_w_dew, y_train_w_dew, y_test_w_dew =
    train_test_split(x_w_dew, y_w_dew, test_size=0.1, random_state=42)

x_train_w_winds, x_test_w_winds, y_train_w_winds, y_test_w_winds =
    train_test_split(x_w_winds, y_w_winds, test_size=0.1, random_state=42)

x_train_w_windd, x_test_w_windd, y_train_w_windd, y_test_w_windd =
    train_test_split(x_w_windd, y_w_windd, test_size=0.1, random_state=42)

x_train_w_relh, x_test_w_relh, y_train_w_relh, y_test_w_relh =
    train_test_split(x_w_relh, y_w_relh, test_size=0.1, random_state=42)

x_train_w_absh, x_test_w_absh, y_train_w_absh, y_test_w_absh =
    train_test_split(x_w_absh, y_w_absh, test_size=0.1, random_state=42)

x_train_w_baro, x_test_w_baro, y_train_w_baro, y_test_w_baro =
    train_test_split(x_w_baro, y_w_baro, test_size=0.1, random_state=42)

x_train_w_solar, x_test_w_solar, y_train_w_solar, y_test_w_solar =
    train_test_split(x_w_solar, y_w_solar, test_size=0.1, random_state=42)

# Scaling the features

sc_x = StandardScaler()
sc_y = StandardScaler()

sc_x_all = StandardScaler()
sc_y_all = StandardScaler()

sc_x_w_temp = StandardScaler()
sc_y_w_temp = StandardScaler()

sc_x_w_dew = StandardScaler()
sc_y_w_dew = StandardScaler()

sc_x_w_winds = StandardScaler()
sc_y_w_winds = StandardScaler()
```

```
sc_x_w_windd = StandardScaler()
sc_y_w_windd= StandardScaler()

sc_x_w_relh = StandardScaler()
sc_y_w_relh = StandardScaler()

sc_x_w_absh = StandardScaler()
sc_y_w_absh = StandardScaler()

sc_x_w_baro = StandardScaler()
sc_y_w_baro = StandardScaler()

sc_x_w_solar = StandardScaler()
sc_y_w_solar = StandardScaler()

# Scaling the training sets
x_train = sc_x.fit_transform(x_train)
y_train = sc_y.fit_transform(y_train)

x_train_svr_all = sc_x_all.fit_transform(x_train_all)
y_train_svr_all = sc_y_all.fit_transform(y_train_all)

x_train_svr_w_temp = sc_x_w_temp.fit_transform(x_train_w_temp)
y_train_svr_w_temp = sc_y_w_temp.fit_transform(y_train_w_temp)

x_train_svr_w_dew = sc_x_w_dew.fit_transform(x_train_w_dew)
y_train_svr_w_dew = sc_y_w_dew.fit_transform(y_train_w_dew)

x_train_svr_w_winds = sc_x_w_winds.fit_transform(x_train_w_winds)
y_train_svr_w_winds = sc_y_w_winds.fit_transform(y_train_w_winds)

x_train_svr_w_windd = sc_x_w_windd.fit_transform(x_train_w_windd)
y_train_svr_w_windd = sc_y_w_windd.fit_transform(y_train_w_windd)

x_train_svr_w_relh = sc_x_w_relh.fit_transform(x_train_w_relh)
y_train_svr_w_relh = sc_y_w_relh.fit_transform(y_train_w_relh)

x_train_svr_w_absh = sc_x_w_absh.fit_transform(x_train_w_absh)
y_train_svr_w_absh = sc_y_w_absh.fit_transform(y_train_w_absh)

x_train_svr_w_baro = sc_x_w_baro.fit_transform(x_train_w_baro)
y_train_svr_w_baro = sc_y_w_baro.fit_transform(y_train_w_baro)

x_train_svr_w_solar = sc_x_w_solar.fit_transform(x_train_w_solar)
y_train_svr_w_solar = sc_y_w_solar.fit_transform(y_train_w_solar)

# Testing with all parameters
# Instantiate the SVR model and then fitting it for X and Y trainings
svr_all = SVR()
svr_all.fit(x_train_svr_all,y_train_svr_all)

# Predicting the scaled test sets features
# And after that inversing the scaled to original values
y_pred =
    sc_y_all.inverse_transform(svr_all.predict(sc_x_all.transform(x_test_all)))

# MAE , MSE , RMSE and R^2 scores
```

```
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, y_pred)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, y_pred)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test, y_pred))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,y_pred)))

# All parameters but optimized
param_grid_all = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_all = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_all = GridSearchCV(SVR(), param_grid_all, cv=kf_all, n_jobs=-1)
grid_search_all.fit(x_train_svr_all, y_train_svr_all);
print('Best Parameter: ', grid_search_all.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_all.best_score_))
print('Best Estimator: ', grid_search_all.best_estimator_)
pred_svr_all=
    sc_y_all.inverse_transform(grid_search_all.predict(sc_x_all.transform(x_test_all)))
print('Test Score: ',
    "{:.3f}".format(grid_search_all.score(x_train_svr_all,
    y_train_svr_all)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_all, pred_svr_all)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_all, pred_svr_all)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_all,
    pred_svr_all))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test_all,pred_svr_all)))

mae_score_all = "{:.3f}".format(mean_absolute_error(y_test_all,
    pred_svr_all))
mse_score_all = "{:.3f}".format(mean_squared_error(y_test_all,
    pred_svr_all))
rmse_score_all =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_all,
    pred_svr_all)))
r2_score_all = "{:.3f}".format(r2_score(y_test_all,pred_svr_all))

# Every feature except Temperature
param_grid_w_temp = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_temp = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_temp = GridSearchCV(SVR(), param_grid_w_temp, cv=kf_w_temp,
    n_jobs=-1)
grid_search_w_temp.fit(x_train_svr_w_temp, y_train_svr_w_temp)
print('Best Parameter: ', grid_search_w_temp.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_temp.best_score_))
print('Best Estimator: ', grid_search_w_temp.best_estimator_)
pred_svr_w_temp=
    sc_y_w_temp.inverse_transform(grid_search_w_temp.predict(sc_x_w_temp.transform(x_test_w_temp)
print('Test Score: ',
    "{:.3f}".format(grid_search_w_temp.score(x_train_svr_w_temp,
    y_train_svr_w_temp)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_temp,
    pred_svr_w_temp)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_temp,
    pred_svr_w_temp)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_temp,
    pred_svr_w_temp))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test_w_temp,pred_svr_w_temp)))

mae_score_w_temp = "{:.3f}".format(mean_absolute_error(y_test_w_temp,
    pred_svr_w_temp))
mse_score_w_temp = "{:.3f}".format(mean_squared_error(y_test_w_temp,
    pred_svr_w_temp))
rmse_score_w_temp =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_temp,
    pred_svr_w_temp)))
r2_score_w_temp = "{:.4f}".format(r2_score(y_test_w_temp,pred_svr_w_temp))

# Every feature except Dew point
param_grid_w_dew = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_dew = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_dew = GridSearchCV(SVR(), param_grid_w_dew, cv=kf_w_dew,
    n_jobs=-1)
grid_search_w_dew.fit(x_train_svr_w_dew, y_train_svr_w_dew);
print('Best Parameter: ', grid_search_w_dew.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_dew.best_score_))
print('Best Estimator: ', grid_search_w_dew.best_estimator_)
pred_svr_w_dew=
    sc_y_w_dew.inverse_transform(grid_search_w_dew.predict(sc_x_w_dew.transform(x_test_w_dew)))
print('Test Score: ',
    "{:.3f}".format(grid_search_w_dew.score(x_train_svr_w_dew,
    y_train_svr_w_dew)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_dew,
    pred_svr_w_dew)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_dew,
    pred_svr_w_dew)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_dew,
    pred_svr_w_dew))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test_w_dew,pred_svr_w_dew)))
```

```
mae_score_w_dew = "{:.3f}".format(mean_absolute_error(y_test_w_dew,
    pred_svr_w_dew))
mse_score_w_dew = "{:.3f}".format(mean_squared_error(y_test_w_dew,
    pred_svr_w_dew))
rmse_score_w_dew =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_dew,
    pred_svr_w_dew)))
r2_score_w_dew = "{:.4f}".format(r2_score(y_test_w_dew,pred_svr_w_dew))

# Every feature except Wind speed
param_grid_w_winds = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_winds = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_winds = GridSearchCV(SVR(), param_grid_w_winds,
    cv=kf_w_winds, n_jobs=-1)
grid_search_w_winds.fit(x_train_svr_w_winds, y_train_svr_w_winds);
print('Best Parameter: ', grid_search_w_winds.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_winds.best_score_))
pred_svr_w_winds=
    sc_y_w_winds.inverse_transform(grid_search_w_winds.predict(sc_x_w_winds.transform(x_test_w_wi
print('Test Score: ',
    "{:.3f}".format(grid_search_w_winds.score(x_train_svr_w_winds,
    y_train_svr_w_winds)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_winds,
    pred_svr_w_winds)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_winds,
    pred_svr_w_winds)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_winds,
    pred_svr_w_winds))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test_w_winds,pred_svr_w_winds)))

mae_score_w_winds = "{:.3f}".format(mean_absolute_error(y_test_w_winds,
    pred_svr_w_winds))
mse_score_w_winds = "{:.3f}".format(mean_squared_error(y_test_w_winds,
    pred_svr_w_winds))
rmse_score_w_winds =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_winds,
    pred_svr_w_winds)))
r2_score_w_winds =
    "{:.3f}".format(r2_score(y_test_w_winds,pred_svr_w_winds))

# Every feature except Wind direction
param_grid_w_windd = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_windd = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_windd = GridSearchCV(SVR(), param_grid_w_windd,
    cv=kf_w_windd, n_jobs=-1)
grid_search_w_windd.fit(x_train_svr_w_windd, y_train_svr_w_windd);
print('Best Parameter: ', grid_search_w_windd.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_windd.best_score_))
```

```
pred_svr_w_windd=
    sc_y_w_windd.inverse_transform(grid_search_w_windd.predict(sc_x_w_windd.transform(x_test_w_wi
print('Test Score: ',
    "{:.3f}".format(grid_search_w_windd.score(x_train_svr_w_windd,
    y_train_svr_w_windd)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_windd,
    pred_svr_w_windd)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_windd,
    pred_svr_w_windd)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_windd,
    pred_svr_w_windd))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test_w_windd,pred_svr_w_windd)))

mae_score_w_windd = "{:.3f}".format(mean_absolute_error(y_test_w_windd,
    pred_svr_w_windd))
mse_score_w_windd = "{:.3f}".format(mean_squared_error(y_test_w_windd,
    pred_svr_w_windd))
rmse_score_w_windd =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_windd,
    pred_svr_w_windd)))
r2_score_w_windd =
    "{:.3f}".format(r2_score(y_test_w_windd,pred_svr_w_windd))

# Every feature except Relative humidity
param_grid_w_relh = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_relh = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_relh = GridSearchCV(SVR(), param_grid_w_relh, cv=kf_w_relh,
    n_jobs=-1)
grid_search_w_relh.fit(x_train_svr_w_relh, y_train_svr_w_relh);
print('Best Parameter: ', grid_search_w_relh.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_relh.best_score_))
pred_svr_w_relh=
    sc_y_w_relh.inverse_transform(grid_search_w_relh.predict(sc_x_w_relh.transform(x_test_w_relh)
print('Test Score: ',
    "{:.3f}".format(grid_search_w_relh.score(x_train_svr_w_relh,
    y_train_svr_w_relh)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_relh,
    pred_svr_w_relh)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_relh,
    pred_svr_w_relh)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_relh,
    pred_svr_w_relh))))
```

```
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test_w_relh,pred_svr_w_relh)))

mae_score_w_relh = "{:.3f}".format(mean_absolute_error(y_test_w_relh,
    pred_svr_w_relh))
mse_score_w_relh = "{:.3f}".format(mean_squared_error(y_test_w_relh,
    pred_svr_w_relh))
rmse_score_w_relh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_relh,
    pred_svr_w_relh)))
r2_score_w_relh = "{:.3f}".format(r2_score(y_test_w_relh,pred_svr_w_relh))

# Every feature except Absolute humidity
param_grid_w_absh = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_absh = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_absh = GridSearchCV(SVR(), param_grid_w_absh, cv=kf_w_absh,
    n_jobs=-1)
grid_search_w_absh.fit(x_train_svr_w_absh, y_train_svr_w_absh);
print('Best Parameter: ', grid_search_w_absh.best_params_)
print('Training Score: ', "{:.3f}".format(grid_search_w_absh.best_score_))
pred_svr_w_absh=
    sc_y_w_absh.inverse_transform(grid_search_w_absh.predict(sc_x_w_absh.transform(x_test_w_absh)
print('Test Score: ',
    "{:.3f}".format(grid_search_w_absh.score(x_train_svr_w_absh,
    y_train_svr_w_absh)))


# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_absh,
    pred_svr_w_absh)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_absh,
    pred_svr_w_absh)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_absh,
    pred_svr_w_absh))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test_w_absh,pred_svr_w_absh)))

mae_score_w_absh = "{:.3f}".format(mean_absolute_error(y_test_w_absh,
    pred_svr_w_absh))
mse_score_w_absh = "{:.3f}".format(mean_squared_error(y_test_w_absh,
    pred_svr_w_absh))
rmse_score_w_absh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_absh,
    pred_svr_w_absh)))
r2_score_w_absh = "{:.4f}".format(r2_score(y_test_w_absh,pred_svr_w_absh))

# Every feature except Barometric pressure
param_grid_w_baro = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_baro = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_baro = GridSearchCV(SVR(), param_grid_w_baro, cv=kf_w_baro,
    n_jobs=-1)
```

```
grid_search_w_baro.fit(x_train_svr_w_baro, y_train_svr_w_baro);
print('Training Score: ', "{:.3f}".format(grid_search_w_baro.best_score_))
pred_svr_w_baro=
    sc_y_w_baro.inverse_transform(grid_search_w_baro.predict(sc_x_w_baro.transform(x_test_w_baro)
print('Test Score: ',
    "{:.3f}".format(grid_search_w_baro.score(x_train_svr_w_baro,
    y_train_svr_w_baro)))
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_baro,
    pred_svr_w_baro)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_baro,
    pred_svr_w_baro)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_baro,
    pred_svr_w_baro))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.4f}".format(r2_score(y_test_w_baro,pred_svr_w_baro)))

mae_score_w_baro = "{:.3f}".format(mean_absolute_error(y_test_w_baro,
    pred_svr_w_baro))
mse_score_w_baro = "{:.3f}".format(mean_squared_error(y_test_w_baro,
    pred_svr_w_baro))
rmse_score_w_baro =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_baro,
    pred_svr_w_baro)))
r2_score_w_baro = "{:.4f}".format(r2_score(y_test_w_baro,pred_svr_w_baro))

# Every feature except Solar radiation
param_grid_w_solar = {'C': [150,200], 'gamma': [0.1, 1, 5]}
kf_w_solar = KFold(n_splits = 5, shuffle=True, random_state=42)
grid_search_w_solar = GridSearchCV(SVR(), param_grid_w_solar,
    cv=kf_w_solar, n_jobs=-1)
grid_search_w_solar.fit(x_train_svr_w_solar, y_train_svr_w_solar);
print('Training Score: ', "{:.3f}".format(grid_search_w_solar.best_score_))
pred_svr_w_solar=
    sc_y_w_solar.inverse_transform(grid_search_w_solar.predict(sc_x_w_solar.transform(x_test_w_so
print('Test Score: ',
    "{:.3f}".format(grid_search_w_solar.score(x_train_svr_w_solar,
    y_train_svr_w_solar)))

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test_w_solar,
    pred_svr_w_solar)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test_w_solar,
    pred_svr_w_solar)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_solar,
    pred_svr_w_solar))))
print('\n---------------------\n')
print('R^2-Score:',
    "{:.3f}".format(r2_score(y_test_w_solar,pred_svr_w_solar)))

mae_score_w_solar = "{:.3f}".format(mean_absolute_error(y_test_w_solar,
    pred_svr_w_solar))
mse_score_w_solar = "{:.3f}".format(mean_squared_error(y_test_w_solar,
    pred_svr_w_solar))
rmse_score_w_solar =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test_w_solar,
    pred_svr_w_solar)))
r2_score_w_solar =
    "{:.3f}".format(r2_score(y_test_w_solar,pred_svr_w_solar))
```

Multi Linear Regression

```
# Importing the library that enables the use of Multi Linear Regressions
from sklearn.linear_model import LinearRegression
df_reg = pd.read_csv("df.csv", index_col = 0, parse_dates= True)

# Testing with all features
# Assigning [x] to only containing desired features and [y] to
    target-variable
x_all = df_reg.iloc[:,:8].values
y_all = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_all, y_all,
    test_size=0.1, random_state=42)

# Instantiate the MRL model and then fitting it for X and Y trainings
multiple_reg = LinearRegression()
multiple_reg.fit(x_train,y_train)
# Predicting the test sets features
pred_mlt_all = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_all)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_all)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_all))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_all)))

# Assigning the metric scores to variables for later use
mae_score_all = "{:.3f}".format(mean_absolute_error(y_test, pred_mlt_all))
mse_score_all = "{:.3f}".format(mean_squared_error(y_test, pred_mlt_all))
rmse_score_all =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_all)))
r2_score_all = "{:.3f}".format(r2_score(y_test,pred_mlt_all))

# Every feature except Temperature
```

```
x_w_temp = df_reg.iloc[:,1:8].values
y_w_temp = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_temp, y_w_temp,
    test_size=0.1, random_state=42)

multiple_reg = LinearRegression()
multiple_reg.fit(x_train,y_train)
pred_mlt_w_temp = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_temp)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_temp)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_temp))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_temp)))


mae_score_w_temp = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_temp))
mse_score_w_temp = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_temp))
rmse_score_w_temp =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_temp)))
r2_score_w_temp = "{:.4f}".format(r2_score(y_test,pred_mlt_w_temp))

# Every feature except Dew point
x_w_dew = df_reg.iloc[:,[0,2,3,4,5,6,7]].values
y_w_dew = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_dew, y_w_dew,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)
pred_mlt_w_dew = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_dew)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_dew)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_dew))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_dew)))


mae_score_w_dew = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_dew))
mse_score_w_dew = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_dew))
```

```
rmse_score_w_dew =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_dew)))
r2_score_w_dew = "{:.4f}".format(r2_score(y_test,pred_mlt_w_dew))

# Every feature except Wind speed
x_w_winds = df_reg.iloc[:,[0,1,3,4,5,6,7]].values
y_w_winds = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_winds, y_w_winds,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)
pred_mlt_w_winds = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_winds)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_winds)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_winds))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_winds)))

mae_score_w_winds = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_winds))
mse_score_w_winds = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_winds))
rmse_score_w_winds =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_winds)))
r2_score_w_winds = "{:.4f}".format(r2_score(y_test,pred_mlt_w_winds))

# Every feature except Wind direction
x_w_windd = df_reg.iloc[:,[0,1,2,4,5,6,7]].values
y_w_windd = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_windd, y_w_windd,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)
pred_mlt_w_windd = multiple_reg.predict(x_test)
# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_windd)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_windd)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_windd))))
print('\n---------------------\n')
print('R^2-Score:', "{:.3f}".format(r2_score(y_test,pred_mlt_w_windd)))

mae_score_w_windd = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_windd))
```

```
mse_score_w_windd = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_windd))
rmse_score_w_windd =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_windd)))
r2_score_w_windd = "{:.3f}".format(r2_score(y_test,pred_mlt_w_windd))

# Every features except Relative humidity
x_w_relh = df_reg.iloc[:,[0,1,2,3,5,6,7]].values
y_w_relh = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_relh, y_w_relh,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)
pred_mlt_w_relh = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_relh)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_relh)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_relh))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_relh)))

mae_score_w_relh = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_relh))
mse_score_w_relh = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_relh))
rmse_score_w_relh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_relh)))
r2_score_w_relh = "{:.4f}".format(r2_score(y_test,pred_mlt_w_relh))

# Every features except Absolute humidity
x_w_absh = df_reg.iloc[:,[0,1,2,3,4,6,7]].values
y_w_absh = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_absh, y_w_absh,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)
pred_mlt_w_absh = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_absh)))
print('\n----------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_absh)))
print('\n----------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_absh))))
print('\n----------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_absh)))
```

```
mae_score_w_absh = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_absh))
mse_score_w_absh = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_absh))
rmse_score_w_absh =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_absh)))
r2_score_w_absh = "{:.4f}".format(r2_score(y_test,pred_mlt_w_absh))

# Every features except Barometric pressure
x_w_baro = df_reg.iloc[:,[0,1,2,3,4,5,7]].values
y_w_baro = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_baro, y_w_baro,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_baro)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_baro)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_baro))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_baro)))

mae_score_w_baro = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_baro))
mse_score_w_baro = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_baro))
rmse_score_w_baro =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_baro)))
r2_score_w_baro = "{:.4f}".format(r2_score(y_test,pred_mlt_w_baro))

# Every features except Solar radiation
x_w_solar = df_reg.iloc[:,0:7].values
y_w_solar = df_reg.iloc[:,[-1]].values
x_train, x_test, y_train, y_test = train_test_split(x_w_solar, y_w_solar,
    test_size=0.1, random_state=42)
multiple_reg.fit(x_train,y_train)
pred_mlt_w_solar = multiple_reg.predict(x_test)

# MAE , MSE , RMSE and R^2 scores
print('Mean Absolute Error:',
    "{:.3f}".format(metrics.mean_absolute_error(y_test, pred_mlt_w_solar)))
print('\n---------------------\n')
print('Mean Squared Error:',
    "{:.3f}".format(metrics.mean_squared_error(y_test, pred_mlt_w_solar)))
print('\n---------------------\n')
print('Root Mean Squared Error:',
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_solar))))
print('\n---------------------\n')
print('R^2-Score:', "{:.4f}".format(r2_score(y_test,pred_mlt_w_solar)))
```

```
mae_score_w_solar = "{:.3f}".format(mean_absolute_error(y_test,
    pred_mlt_w_solar))
mse_score_w_solar = "{:.3f}".format(mean_squared_error(y_test,
    pred_mlt_w_solar))
rmse_score_w_solar =
    "{:.3f}".format(np.sqrt(metrics.mean_squared_error(y_test,
    pred_mlt_w_solar)))
r2_score_w_solar = "{:.4f}".format(r2_score(y_test,pred_mlt_w_solar))
```

Correlation test

```
import pandas as pd
import numpy as np
import seaborn as sns

df_c  = pd.read_csv("df.csv",parse_dates=['Timestamp'], index_col = 0)

mask = np.zeros_like(df_c.corr())
triangle_indices = np.triu_indices_from(mask)
mask[triangle_indices] = True
mask

plt.figure(figsize=(10,6))
sns.heatmap(df_c.corr(), mask=mask , annot=True, annot_kws={"size":14})
plt.xticks(fontsize = 14)
plt.yticks(fontsize = 14)
plt.show()
```

Mohamed Hadi & Stefan Quvald Jacob

Predictions on solar power plant generation with machine learning techniques

NTNU
Norwegian University of
Science and Technology