Abstract

Wind energy is an important source of renewable energy, and offshore wind turbines are becoming increasingly popular due to the strong and consistent wind resources available in offshore locations. However, the efficiency and performance of offshore wind turbines are heavily reliant on wind speed, making accurate wind speed predictions essential for their optimal operation. This project focuses on forecasting wind speed in the offshore windpark Beatrice, located in the North Sea, using four different models: ARIMA, RF, LR, and SVR. These models are widely used in time-series forecasting and machine learning and have shown promising results in predicting wind speed in various contexts. The dataset used in this assignment is a csv file. Each row shows an specific location in the UK and 10 parameters for predicting weather are reported 8 times per day during the month of May in 2018. After cleaning, the data was then split into training and testing sets with 80% and 20% of the data, respectively. Using the training data, each model was trained and subsequently evaluated on the testing data using root mean squared error (RMSE) metrics. The results of the evaluation showed that the RF model demonstrated the highest level of accuracy

Keywords: *Weather forecasting, Timeseries, Wind speed, ARIMA model, Statistical model, Machine leraning models, Predicting wind speed*

Contents

List of Figures

List of Tables

Implementation and Discussion

In this section, an end-to-end timeseries project will be implemented and the research questions mentioned in previous sections will be answered. First, the data will be examined.

1.1 Data

The dataset that is examined here is a CSV file called "$WRFdata\_May2018$" which has 5453 rows and 2482 columns. Each row shows an specific location in the UK and 10 parameters for predicting weather are reported 8 times per day during the month of May in 2018. The factors are as follows:

Parameter	Description	MeasuringUnit
$XLAT$	$Latitude$	
$XLONG$	$Longitude$	
TSK	$Skintemperatureorsurfacetemperature$	$oK(Kelvin)$
$PSFC$	$Surfacepressure$	$Pa(Pascal)$
$U10$	$Xcomponentofwindat10m$	m/s
$V10$	$Ycomponentofwindat10m$	m/s
$Q2$	$2-meterspecifichumidity$	Kg/Kg
$Rainc$	$Convectiverain(Accumulatedprecipitation)$	mm
$Rainnc$	$Non-convectiverain$	Mm
$Snow$	$Snowwaterequivalent$	$Kg/m2$
$TSLB$	$Soiltemperature$	oK
$SMOIS$	$SoilMoisture$	$m3/m3$

Table 1.1: Data description

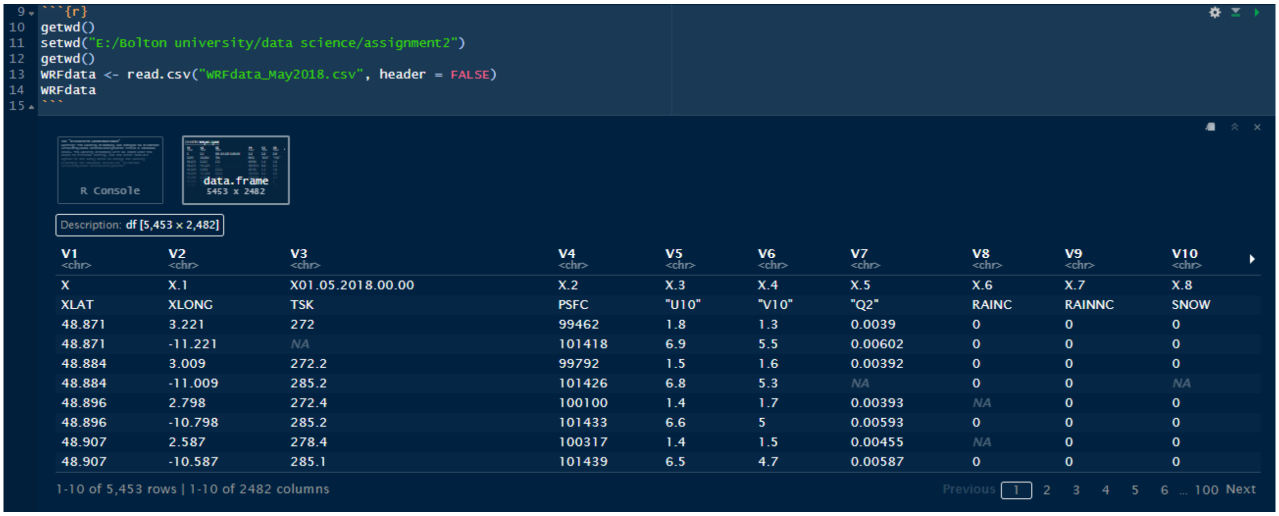
In this assignment, 400 rows are selected for data exploration, and then only one location will be chosen for time series analysis.

1.2 Data Exploration

Data exploration is an important part of any timeseries project. It is the process of understanding the data, identifying trends, patterns, and seasonal effects. This information can be used to build better time series models and to make more accurate forecasts.

1.2.1 Loading the Data

The read.csv() function reads the CSV file "$WRFdata\_May2018.csv$" into the variable WRFdata. Since the CSV file does not contain a header row, the header argument has been set to FALSE.



````r
```{r}
getwd()
setwd("E:/Bolton university/data science/assignment2")
getwd()
WRFdata <- read.csv("WRFdata_May2018.csv", header = FALSE)
WRFdata
```
````

R Console | data.frame 5453 x 2482

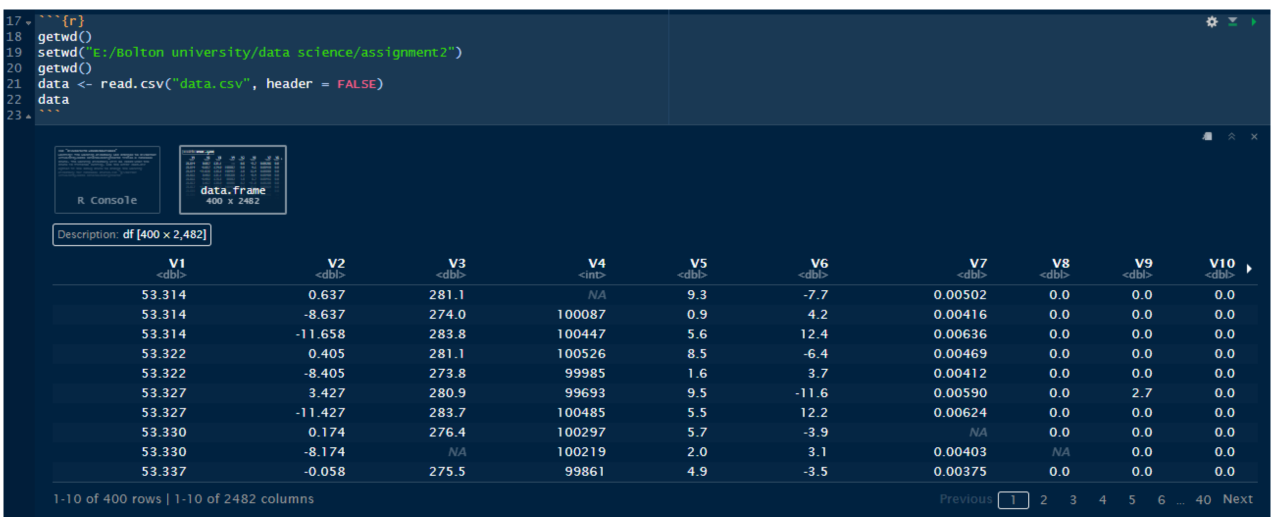
Description: df [5,453 x 2,482]

V1 <chr>	V2 <chr>	V3 <chr>	V4 <chr>	V5 <chr>	V6 <chr>	V7 <chr>	V8 <chr>	V9 <chr>	V10 <chr>
X	X.1	X01.05.2018.00.00	X.2	X.3	X.4	X.5	X.6	X.7	X.8
XLAT	XLONG	TSK	PSFC	"U10"	"V10"	"Q2"	RAINC	RAINNC	SNOW
48.871	3.221	272	99462	1.8	1.3	0.0039	0	0	0
48.871	-11.221	NA	101418	6.9	5.5	0.00602	0	0	0
48.884	3.009	272.2	99792	1.5	1.6	0.00392	0	0	0
48.884	-11.009	285.2	101426	6.8	5.3	NA	0	0	NA
48.896	2.798	272.4	100100	1.4	1.7	0.00393	NA	0	0
48.896	-10.798	285.2	101433	6.6	5	0.00593	0	0	0
48.907	2.587	278.4	100317	1.4	1.5	0.00455	NA	0	0
48.907	-10.587	285.1	101439	6.5	4.7	0.00587	0	0	0

1-10 of 5,453 rows | 1-10 of 2482 columns Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.1: Reading the main CSV file

As it can be seen in Fig(1.1), 5,453 rows and 2,482 columns of data are contained in this CSV file. The first two columns indicate different locations in the UK. A sample of 400 rows is selected in Excel and uploaded here(Fig(1.2)).



````r
```{r}
getwd()
setwd("E:/Bolton university/data science/assignment2")
getwd()
data <- read.csv("data.csv", header = FALSE)
data
```
````

R Console | data.frame 400 x 2482

Description: df [400 x 2,482]

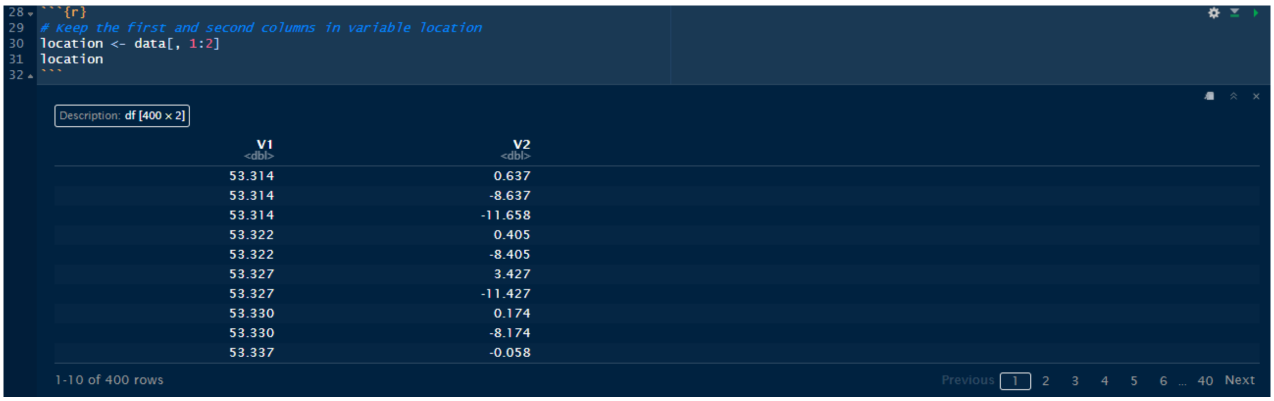
V1 <dbl>	V2 <dbl>	V3 <dbl>	V4 <int>	V5 <dbl>	V6 <dbl>	V7 <dbl>	V8 <dbl>	V9 <dbl>	V10 <dbl>
53.314	0.637	281.1	NA	9.3	-7.7	0.00502	0.0	0.0	0.0
53.314	-8.637	274.0	100087	0.9	4.2	0.00416	0.0	0.0	0.0
53.314	-11.658	283.8	100447	5.6	12.4	0.00636	0.0	0.0	0.0
53.322	0.405	281.1	100526	8.5	-6.4	0.00469	0.0	0.0	0.0
53.322	-8.405	273.8	99985	1.6	3.7	0.00412	0.0	0.0	0.0
53.327	3.427	280.9	99693	9.5	-11.6	0.00590	0.0	2.7	0.0
53.327	-11.427	283.7	100485	5.5	12.2	0.00624	0.0	0.0	0.0
53.330	0.174	276.4	100297	5.7	-3.9	NA	0.0	0.0	0.0
53.330	-8.174	NA	100219	2.0	3.1	0.00403	NA	0.0	0.0
53.337	-0.058	275.5	99861	4.9	-3.5	0.00375	0.0	0.0	0.0

1-10 of 400 rows | 1-10 of 2482 columns Previous 1 2 3 4 5 6 ... 40 Next

Figure 1.2: Reading the selected data

1.2.2 Reshaping the data

The variable "data" has 400 rows which means 400 different locations are chosen. Before doing any analysis, it is better to reshape the data to understand it and work with it better. In order to do this, the first two columns are kept in variable "location" (Fig(1.3)) and then deleted from the dataset(Fig(1.4)). Next, the data is reshaped without locations and a column for the date and time is added(Fig(1.5), (1.6)). It also can be seen in Fig(1.6) that the column names has been changed.



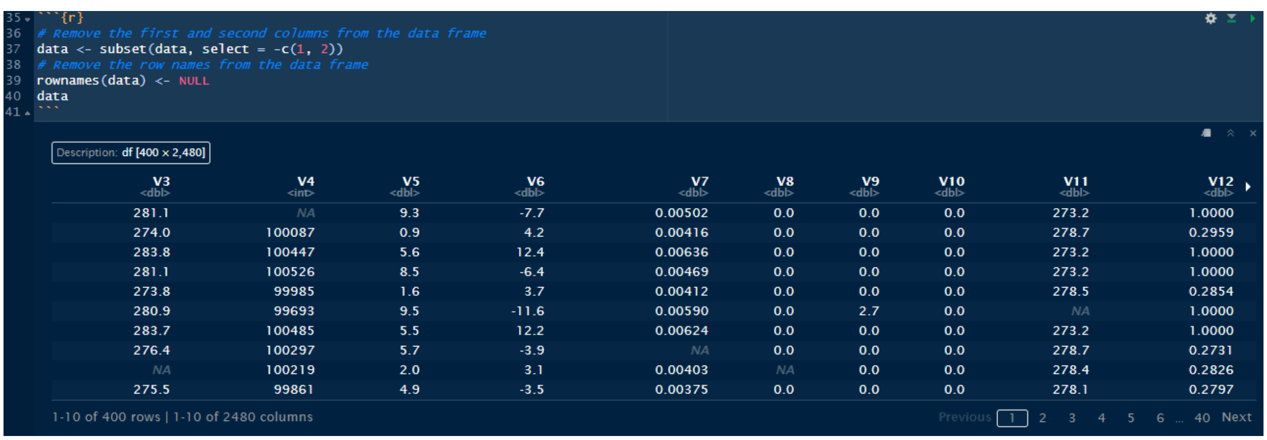
````
```{r}
# Keep the first and second columns in variable location
location <- data[, 1:2]
location
```
````

Description: df [400 x 2]

V1 <dbl>	V2 <dbl>
53.314	0.637
53.314	-8.637
53.314	-11.658
53.322	0.405
53.322	-8.405
53.327	3.427
53.327	-11.427
53.330	0.174
53.330	-8.174
53.337	-0.058

1-10 of 400 rows Previous 1 2 3 4 5 6 ... 40 Next

Figure 1.3: Selecting and keeping two columns in variable location



````
```{r}
# Remove the first and second columns from the data frame
data <- subset(data, select = -c(1, 2))
# Remove the row names from the data frame
rownames(data) <- NULL
data
```
````

Description: df [400 x 2,480]

V3 <dbl>	V4 <int>	V5 <dbl>	V6 <dbl>	V7 <dbl>	V8 <dbl>	V9 <dbl>	V10 <dbl>	V11 <dbl>	V12 <dbl>
281.1	NA	9.3	-7.7	0.00502	0.0	0.0	0.0	273.2	1.0000
274.0	100087	0.9	4.2	0.00416	0.0	0.0	0.0	278.7	0.2959
283.8	100447	5.6	12.4	0.00636	0.0	0.0	0.0	273.2	1.0000
281.1	100526	8.5	-6.4	0.00469	0.0	0.0	0.0	273.2	1.0000
273.8	99985	1.6	3.7	0.00412	0.0	0.0	0.0	278.5	0.2854
280.9	99693	9.5	-11.6	0.00590	0.0	2.7	0.0	NA	1.0000
283.7	100485	5.5	12.2	0.00624	0.0	0.0	0.0	273.2	1.0000
276.4	100297	5.7	-3.9	NA	0.0	0.0	0.0	278.7	0.2731
NA	100219	2.0	3.1	0.00403	NA	0.0	0.0	278.4	0.2826
275.5	99861	4.9	-3.5	0.00375	0.0	0.0	0.0	278.1	0.2797

1-10 of 400 rows | 1-10 of 2480 columns Previous 1 2 3 4 5 6 ... 40 Next

Figure 1.4: Deleting two columns from the dataset

After reshaping the data, locations are then added to the dataset(Fig(1.7)) and then these two columns are moved to the first(Fig(1.8))

Now, the dataset is ready for data exploration.

```
```{r reshape}
# create an empty list to store the reshaped data frames
data_list <- list()

# loop over each row of the original time series
for (i in 1:nrow(data)) {

  # extract the current row and convert to matrix
  current_row <- as.matrix(data[i,])

  # reshape the current row and add column names
  current_reshaped <- matrix(as.numeric(current_row), nrow = 248, ncol = 10, byrow = TRUE)
  colnames(current_reshaped) <- c("TSK","PSFC","U10","V10","Q2","Rainc","Rainnc","Snow","TSLB","SMOIS")

  # add a date column to the reshaped data
  current_df <- cbind.data.frame(date = seq(as.POSIXct("2018-05-01 00:00:00"), as.POSIXct("2018-05-31 21:00:00"),
                                            by = "3 hours"), current_reshaped)

  # add the current data frame to the list
  data_list[[i]] <- current_df
}

# combine all data frames in the list into a single data frame
final_df <- do.call(rbind, data_list)
final_df
```
```

Figure 1.5: Reshaping the dataset and adding the date column to the dataset

Description: df [99,200 x 11]

date <S3: POSIXct>	TSK <dbl>	PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>	Rainnc <dbl>	Snow <dbl>	TSLB <dbl>
2018-05-01 00:00:00	281.1	NA	9.3	-7.7	0.00502	0.0	0.0	0	273.2
2018-05-01 03:00:00	281.1	100558	9.1	-4.9	0.00454	0.0	0.0	0	273.2
2018-05-01 06:00:00	281.1	100657	9.1	-2.2	0.00390	0.0	0.0	0	NA
2018-05-01 09:00:00	281.1	100748	5.6	0.7	0.00532	0.0	0.0	0	273.2
2018-05-01 12:00:00	281.1	100757	1.5	6.1	0.00492	0.0	0.0	0	273.2
2018-05-01 15:00:00	281.1	100654	2.2	6.4	0.00570	0.0	0.0	0	273.2
2018-05-01 18:00:00	281.1	100633	2.9	7.4	0.00573	0.0	0.0	0	273.2
2018-05-01 21:00:00	281.1	100621	3.4	9.0	0.00571	0.0	0.0	0	273.2
2018-05-02 00:00:00	281.5	100551	1.6	11.9	0.00548	0.0	0.0	0	273.2
2018-05-02 03:00:00	281.5	100387	-0.4	12.6	NA	0.0	0.0	0	273.2

1-10 of 99,200 rows | 1-10 of 11 columns   Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.6: Reshaping the dataset

```
```{r}
# Repeat each location coordinate 248 times
locations_rep <- data.frame(rep(location[[1]], each = 248), rep(location[[2]], each = 248))


colnames(locations_rep) <- c("Latitude", "Longitude")

# Merge final_df and locations_rep data frames
df_with_locations <- cbind(final_df, locations_rep)

df_with_locations
```
```

Description: df [99,200 x 13]

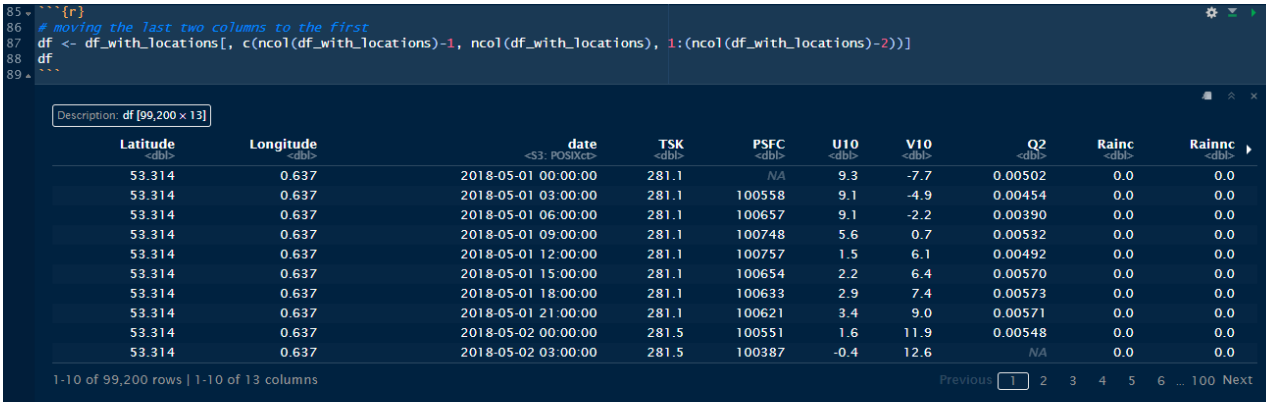
date <S3: POSIXct>	TSK <dbl>	PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>	Rainnc <dbl>	Snow <dbl>	TSLB <dbl>
2018-05-01 00:00:00	281.1	NA	9.3	-7.7	0.00502	0.0	0.0	0	273.2
2018-05-01 03:00:00	281.1	100558	9.1	-4.9	0.00454	0.0	0.0	0	273.2
2018-05-01 06:00:00	281.1	100657	9.1	-2.2	0.00390	0.0	0.0	0	NA
2018-05-01 09:00:00	281.1	100748	5.6	0.7	0.00532	0.0	0.0	0	273.2
2018-05-01 12:00:00	281.1	100757	1.5	6.1	0.00492	0.0	0.0	0	273.2
2018-05-01 15:00:00	281.1	100654	2.2	6.4	0.00570	0.0	0.0	0	273.2
2018-05-01 18:00:00	281.1	100633	2.9	7.4	0.00573	0.0	0.0	0	273.2
2018-05-01 21:00:00	281.1	100621	3.4	9.0	0.00571	0.0	0.0	0	273.2
2018-05-02 00:00:00	281.5	100551	1.6	11.9	0.00548	0.0	0.0	0	273.2
2018-05-02 03:00:00	281.5	100387	-0.4	12.6	NA	0.0	0.0	0	273.2

1-10 of 99,200 rows | 1-10 of 13 columns   Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.7: Adding location columns to the dataset

1.2.3 Describing the data

For describing the data, there are a few lines of code that can be used. Here, some of them which are really important can be seen.



````
```{r}
# moving the last two columns to the first
df <- df_with_locations[, c(ncol(df_with_locations)-1, ncol(df_with_locations), 1:(ncol(df_with_locations)-2))]
df
```
````

Description: df [99,200 x 13]

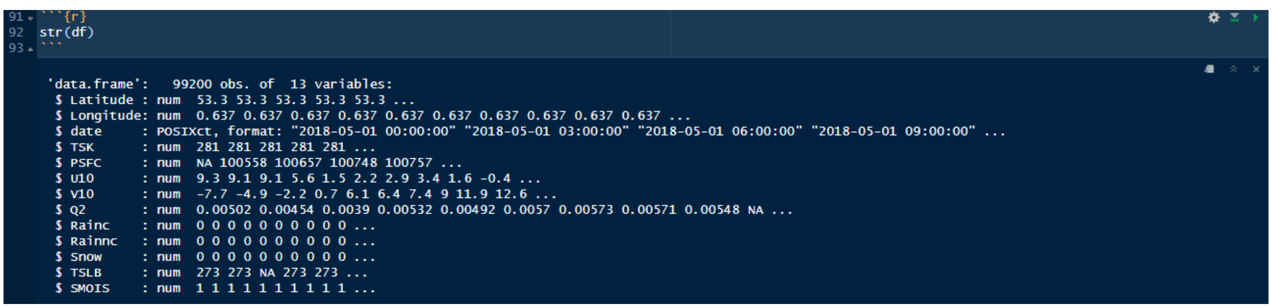
Latitude <dbl>	Longitude <dbl>	date <S3: POSIXct>	TSK <dbl>	PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>	Rainnc <dbl>
53.314	0.637	2018-05-01 00:00:00	281.1	NA	9.3	-7.7	0.00502	0.0	0.0
53.314	0.637	2018-05-01 03:00:00	281.1	100558	9.1	-4.9	0.00454	0.0	0.0
53.314	0.637	2018-05-01 06:00:00	281.1	100657	9.1	-2.2	0.00390	0.0	0.0
53.314	0.637	2018-05-01 09:00:00	281.1	100748	5.6	0.7	0.00532	0.0	0.0
53.314	0.637	2018-05-01 12:00:00	281.1	100757	1.5	6.1	0.00492	0.0	0.0
53.314	0.637	2018-05-01 15:00:00	281.1	100654	2.2	6.4	0.00570	0.0	0.0
53.314	0.637	2018-05-01 18:00:00	281.1	100633	2.9	7.4	0.00573	0.0	0.0
53.314	0.637	2018-05-01 21:00:00	281.1	100621	3.4	9.0	0.00571	0.0	0.0
53.314	0.637	2018-05-02 00:00:00	281.5	100551	1.6	11.9	0.00548	0.0	0.0
53.314	0.637	2018-05-02 03:00:00	281.5	100387	-0.4	12.6	NA	0.0	0.0

1-10 of 99,200 rows | 1-10 of 13 columns   Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.8: Moving location columns to the first

- str() function

  The str() function is a useful tool for getting an overview of the data. It can be used to quickly see the number of rows and columns, the data types of the columns, and the names of the columns. This information can be helpful for understanding the data and for planning data analysis. In this case, the str() function is telling that the data frame df has 99,200 observations (rows) and 13 variables (columns) and the type of each column can be seen in Fig(1.9).



````
```{r}
str(df)
```

'data.frame':   99200 obs. of  13 variables:
 $ Latitude : num  53.3 53.3 53.3 53.3 53.3 ...
 $ Longitude: num  0.637 0.637 0.637 0.637 0.637 0.637 0.637 0.637 0.637 0.637 ...
 $ date     : POSIXct, format: "2018-05-01 00:00:00" "2018-05-01 03:00:00" "2018-05-01 06:00:00" "2018-05-01 09:00:00" ...
 $ TSK      : num  281 281 281 281 281 ...
 $ PSFC     : num  NA 100558 100657 100748 100757 ...
 $ U10      : num  9.3 9.1 9.1 5.6 1.5 2.2 2.9 3.4 1.6 -0.4 ...
 $ V10      : num  -7.7 -4.9 -2.2 0.7 6.1 6.4 7.4 9 11.9 12.6 ...
 $ Q2       : num  0.00502 0.00454 0.0039 0.00532 0.00492 0.0057 0.00573 0.00571 0.00548 NA ...
 $ Rainc    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Rainnc   : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Snow     : num  0 0 0 0 0 0 0 0 0 0 ...
 $ TSLB     : num  273 273 NA 273 273 ...
 $ SMOIS    : num  1 1 1 1 1 1 1 1 1 1 ...
````

Figure 1.9: An overview of df

- summary() function

  The summary() function provides a more detailed summary of the data. It includes the mean, median, standard deviation, and other statistical measures for each column. The summary also shows that there are some missing values in the data. The number of missing values is shown in the last row of each column. For example, there are 2904 missing values for the Q2 variable(Fig(1.10)).

```{r}
summary(df)
```

```
    Latitude       Longitude            date                          TSK             PSFC             U10                V10
 Min.   :53.31   Min.   :-12.0270   Min.   :2018-05-01 00:00:00   Min.   :269.6   Min.   : 95738   Min.   :-11.5000   Min.   :-14.0000
 1st Qu.:53.52   1st Qu.: -8.2133   1st Qu.:2018-05-08 17:15:00   1st Qu.:282.6   1st Qu.:100788   1st Qu.: -2.7000   1st Qu.: -3.2000
 Median :53.73   Median : -4.0000   Median :2018-05-16 10:30:00   Median :283.8   Median :101530   Median : -0.1000   Median :  0.3000
 Mean   :53.74   Mean   : -4.0196   Mean   :2018-05-16 10:30:00   Mean   :284.8   Mean   :101343   Mean   : -0.0425   Mean   :  0.2553
 3rd Qu.:53.94   3rd Qu.:  0.0525   3rd Qu.:2018-05-24 03:45:00   3rd Qu.:285.8   3rd Qu.:102098   3rd Qu.:  2.4000   3rd Qu.:  3.3000
 Max.   :56.78   Max.   :  4.0270   Max.   :2018-05-31 21:00:00   Max.   :307.5   Max.   :103487   Max.   : 14.7000   Max.   : 17.7000
                                                                  NA's   :2908    NA's   :2992     NA's   :2920       NA's   :2903
       Q2              Rainc              Rainnc              Snow           TSLB            SMOIS
 Min.   :0.0026   Min.   : 0.0000   Min.   : 0.0000   Min.   :0.0    Min.   :273.2   Min.   :0.1822
 1st Qu.:0.0061   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.:0.0    1st Qu.:273.2   1st Qu.:0.2920
 Median :0.0071   Median : 0.0000   Median : 0.0000   Median :0.0    Median :273.2   Median :1.0000
 Mean   :0.0070   Mean   : 0.0871   Mean   : 0.4249   Mean   :0.0    Mean   :278.1   Mean   :0.6983
 3rd Qu.:0.0078   3rd Qu.: 0.0000   3rd Qu.: 0.1000   3rd Qu.:0.0    3rd Qu.:283.5   3rd Qu.:1.0000
 Max.   :0.0128   Max.   :24.3000   Max.   :26.5000   Max.   :0.1    Max.   :303.8   Max.   :1.0000
 NA's   :2904     NA's   :2922      NA's   :2979      NA's   :2949   NA's   :2909    NA's   :2848
```

Figure 1.10: A summary of df

- head() function

  The head() function displays the first few rows of the df data which is six rows here(Fig(1.11)).

```{r}
head(df)
```

Description: df [6 x 13]

	Latitude <dbl>	Longitude <dbl>	date <S3: POSIXct>	TSK <dbl>	PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>
1	53.314	0.637	2018-05-01 00:00:00	281.1	NA	9.3	-7.7	0.00502	0
2	53.314	0.637	2018-05-01 03:00:00	281.1	100558	9.1	-4.9	0.00454	0
3	53.314	0.637	2018-05-01 06:00:00	281.1	100657	9.1	-2.2	0.00390	0
4	53.314	0.637	2018-05-01 09:00:00	281.1	100748	5.6	0.7	0.00532	0
5	53.314	0.637	2018-05-01 12:00:00	281.1	100757	1.5	6.1	0.00492	0
6	53.314	0.637	2018-05-01 15:00:00	281.1	100654	2.2	6.4	0.00570	0

6 rows | 1-10 of 13 columns

Figure 1.11: Viewing a few first rows of df

- tail() function

  The tail() function displays the last few rows of the data(Fig(1.12)).

```{r}
tail(df)
```

Description: df [6 x 13]

	Latitude <dbl>	Longitude <dbl>	date <S3: POSIXct>	TSK <dbl>	PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>
99195	56.781	3.508	2018-05-31 06:00:00	286.1	101725	-2.4	1.3	0.00926	0
99196	56.781	3.508	2018-05-31 09:00:00	286.1	101822	NA	0.8	0.00926	0
99197	56.781	3.508	2018-05-31 12:00:00	286.1	101872	-1.3	-0.7	0.00925	0
99198	56.781	3.508	2018-05-31 15:00:00	286.1	101799	-2.9	-0.2	0.00926	0
99199	56.781	3.508	2018-05-31 18:00:00	286.1	101751	-2.7	0.1	0.00926	0
99200	56.781	3.508	2018-05-31 21:00:00	286.1	101759	-2.0	-0.1	0.00926	0

6 rows | 1-10 of 13 columns

Figure 1.12: Viewing a few last rows of df

- is.na() function

  The is.na() function in R is used to check for missing values in a vector or data frame. It returns a logical vector of the same length as the input vector, with a value of TRUE for each element that is missing and FALSE for each element that is not missing. In (Fig(1.13)), a summation of null values can be seen.

````
```{r}
sum(is.na(df))
```

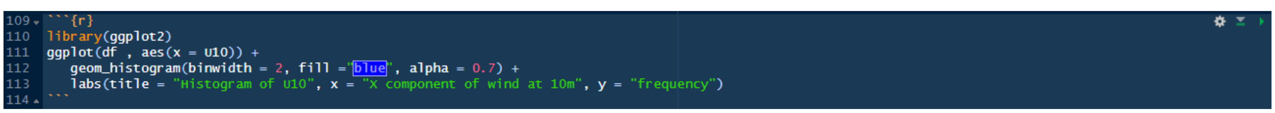
[1] 29234
````

Figure 1.13: The number of null values of df

1.2.4 Visualising the data

- Histogram

  A histogram is a bar graph that shows the distribution of data. Since the goal of this project is to predict the wind speed, the histogram of the wind components can be seen here(Fig(1.14) and (1.16)). And since there are null values in this dataset, null values are deleted when drawing the histogram which can be seen in Fig(1.15) and (1.17)



````
```{r}
library(ggplot2)
ggplot(df , aes(x = U10)) +
  geom_histogram(binwidth = 2, fill ="blue", alpha = 0.7) +
  labs(title = "Histogram of U10", x = "X component of wind at 10m", y = "frequency")
```
````

Figure 1.14: The histogram of U10

- Scatterplot

  A scatterplot is a graph that shows the relationship between two variables. Fig(1.18) and (1.19) show a scatter plot of the U10 and V10 variables with the U10 variable on the x-axis and the V10 variable on the y-axis. Each point on the plot represents a data point.

1.3 Data Cleaning

Data cleaning is the process of identifying, correcting, and removing errors and inconsistencies in data. This is a vital part of any data analysis process, as it guarantees the information being analysed is correct

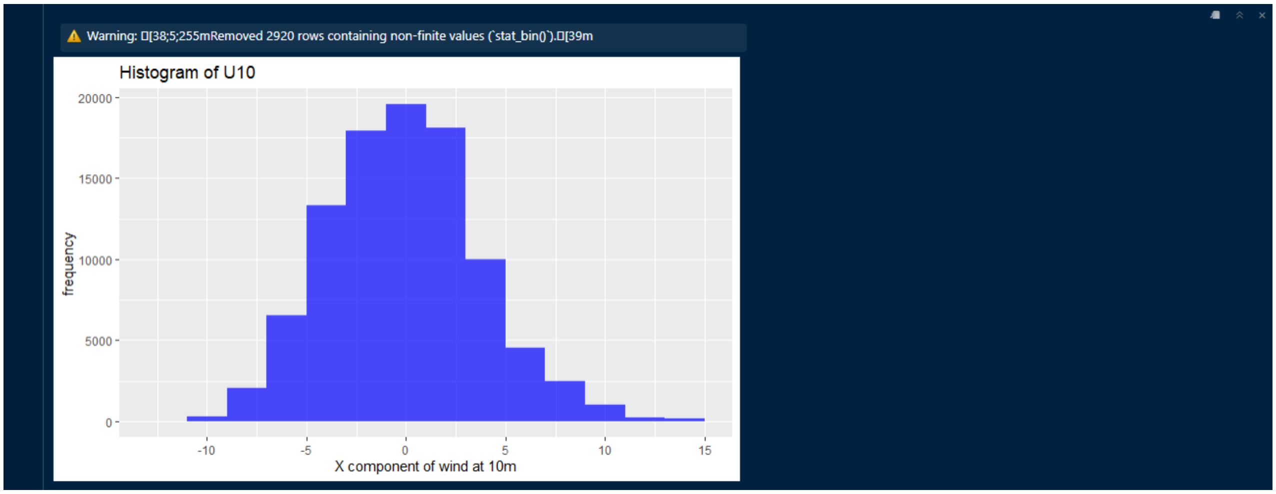


Figure 1.15: The histogram of U10

````r
```{r}
library(ggplot2)
ggplot(df , aes(x = V10)) +
  geom_histogram(binwidth = 2, fill ="blue", alpha = 0.7) +
  labs(title = "Histogram of V10", x = "Y component of wind at 10m", y = "frequency")
```
````

Figure 1.16: The histogram of V10

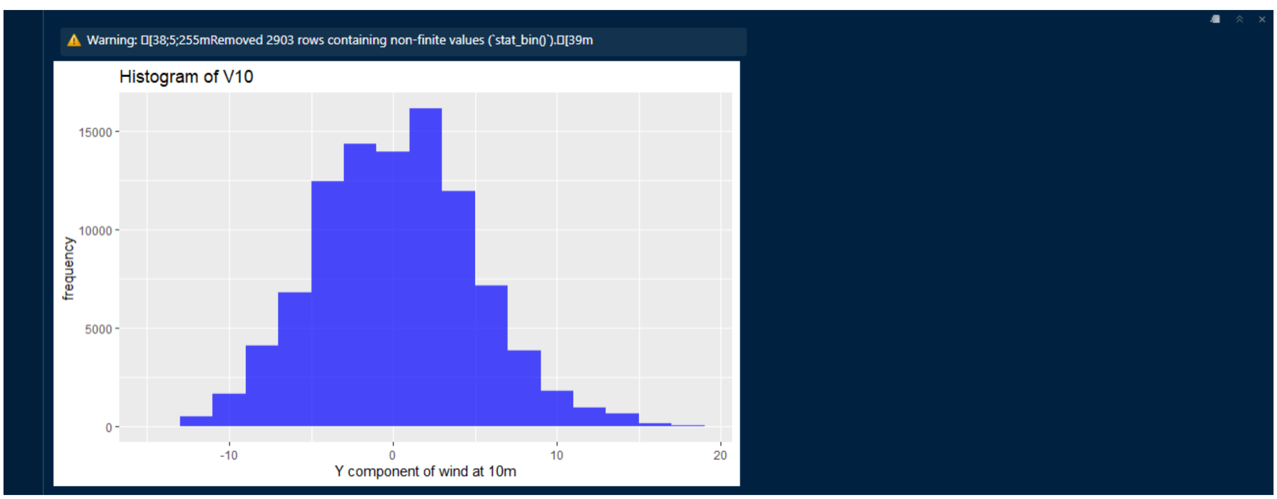


Figure 1.17: The histogram of V10

````r
```{r}
ggplot(df, aes(x = U10, y = V10)) +
  geom_point(color = "blue") +
  labs(x = "U10", y = "V10")
```
````

Figure 1.18: The scatterplot of U10 and V10

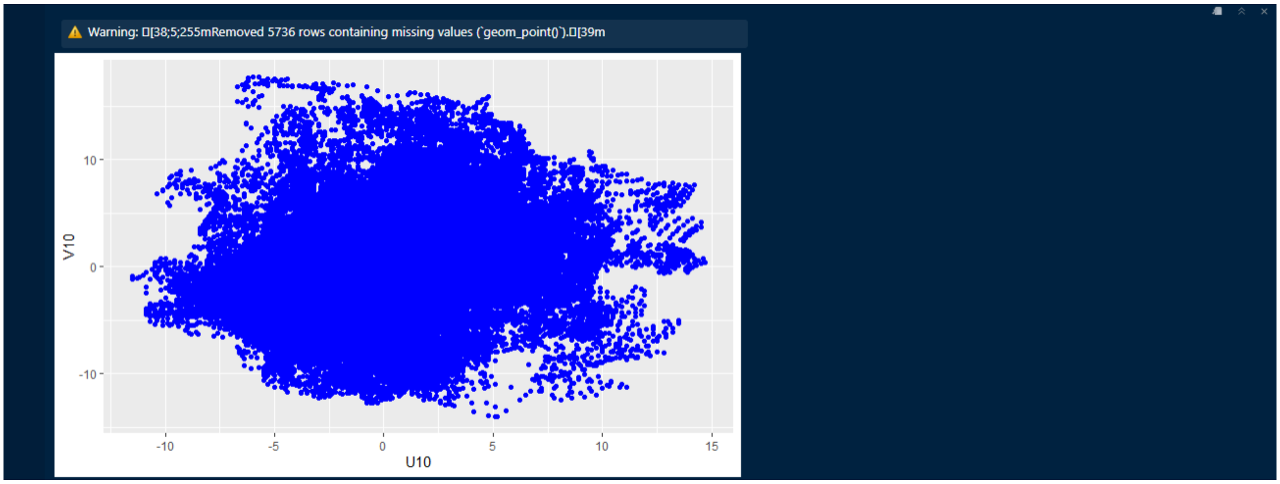


Figure 1.19: The scatterplot of U10 and V10

and trustworthy. There are a number of steps that can be followed in R for cleaning data. In this assignment the following steps will be followed.

1.3.1 Identifying and removing duplicate values

Using duplicated() function, duplicate values can be identified in the dataset. As it can be seen in Fig(1.20), there is no duplicate values in df dataset.

Figure 1.20: The number of duplicate values of df

1.3.2 Identifying and imputing missing values

Missing values are values that are not present in the dataset. They can be identified using the is.na() function. Fig (1.21) shows the total null values of df which is 29234. For imputing missing values there are different methods, but, in this assignment, missing values are imputed using interpolation method. Interpolation is a statistical method that assumes that the data is continuous which is true about the df dataset. Fig (1.22) represents filling missing values using interpolation and then in Fig (1.23) the total number of null values has been checked which is zero. It means that the df dataset now is cleaned. After cleaning the data, now it is time to add a new column for wind speed.

```{r}
sum(is.na(df))
```

```
[1] 29234
```

Figure 1.21: The number of missing values of df

```{r}
library(imputeTS)
df <- na.interpolation(df)
df
```

R Console

data.frame
99200 x 13

Description: df [99,200 × 13]

Latitude <dbl>	Longitude <dbl>	date <S3: POSIXct>	TSK <dbl>	PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>	Rainnc <dbl>
53.314	0.637	2018-05-01 00:00:00	281.10	100558.0	9.30	-7.700	0.005020	0.0	0.0000000
53.314	0.637	2018-05-01 03:00:00	281.10	100558.0	9.10	-4.900	0.004540	0.0	0.0000000
53.314	0.637	2018-05-01 06:00:00	281.10	100657.0	9.10	-2.200	0.003900	0.0	0.0000000
53.314	0.637	2018-05-01 09:00:00	281.10	100748.0	5.60	0.700	0.005320	0.0	0.0000000
53.314	0.637	2018-05-01 12:00:00	281.10	100757.0	1.50	6.100	0.004920	0.0	0.0000000
53.314	0.637	2018-05-01 15:00:00	281.10	100654.0	2.20	6.400	0.005700	0.0	0.0000000
53.314	0.637	2018-05-01 18:00:00	281.10	100633.0	2.90	7.400	0.005730	0.0	0.0000000
53.314	0.637	2018-05-01 21:00:00	281.10	100621.0	3.40	9.000	0.005710	0.0	0.0000000
53.314	0.637	2018-05-02 00:00:00	281.50	100551.0	1.60	11.900	0.005480	0.0	0.0000000
53.314	0.637	2018-05-02 03:00:00	281.50	100387.0	-0.40	12.600	0.005665	0.0	0.0000000

1-10 of 99,200 rows | 1-10 of 13 columns   Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.22: Filling missing values using interpolation

```{r}
sum(is.na(df))
```

```
[1] 0
```

Figure 1.23: Checking missing values after filling them

1.3.3 Adding wind speed column

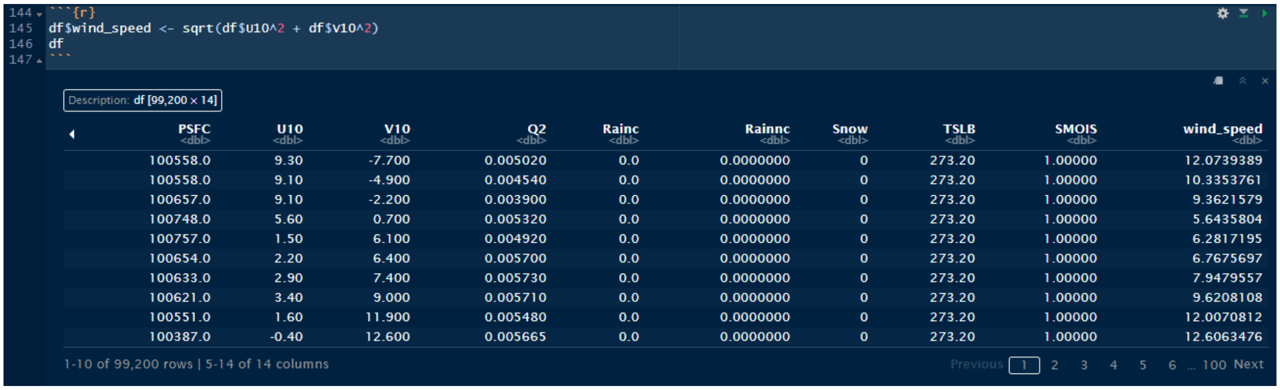
In this part, the new column for wind speed will be calculated and added to the dataset. The wind speed can be calculated from U10 and V10 using the following formula:

$$wind\_speed = sqrt(U^2 + V^2)$$

where U is X component of wind at 10m and V is Y component of wind at 10m. In (1.24) a new column, wind_speed, is added to df dataset.

1.4 Modelling and Forecasting

After going through these steps, it is time to build models that can be used for predicting wind speed. Before building the models, the outliers are checked first, and then the dataframe will be converted into



```r
```{r}
df$wind_speed <- sqrt(df$U10^2 + df$V10^2)
df
```
```

Description: df [99,200 x 14]

PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>	Rainnc <dbl>	Snow <dbl>	TSLB <dbl>	SMOIS <dbl>	wind_speed <dbl>
100558.0	9.30	-7.700	0.005020	0.0	0.0000000	0	273.20	1.00000	12.0739389
100558.0	9.10	-4.900	0.004540	0.0	0.0000000	0	273.20	1.00000	10.3353761
100657.0	9.10	-2.200	0.003900	0.0	0.0000000	0	273.20	1.00000	9.3621579
100748.0	5.60	0.700	0.005320	0.0	0.0000000	0	273.20	1.00000	5.6435804
100757.0	1.50	6.100	0.004920	0.0	0.0000000	0	273.20	1.00000	6.2817195
100654.0	2.20	6.400	0.005700	0.0	0.0000000	0	273.20	1.00000	6.7675697
100633.0	2.90	7.400	0.005730	0.0	0.0000000	0	273.20	1.00000	7.9479557
100621.0	3.40	9.000	0.005710	0.0	0.0000000	0	273.20	1.00000	9.6208108
100551.0	1.60	11.900	0.005480	0.0	0.0000000	0	273.20	1.00000	12.0070812
100387.0	-0.40	12.600	0.005665	0.0	0.0000000	0	273.20	1.00000	12.6063476

1-10 of 99,200 rows | 5-14 of 14 columns

Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.24: Adding wind_speed column to the df dataset

a time series, and after that, four models will be presented to predict the wind speed in this section. Finally, the best model will be evaluated.

1.4.1 Detecting and Handling outliers

A data point which is considered to be an outlier is one that deviates significantly from the norm. There are many factors that lead to the emergence of these outliers, such as data entry errors, measurement errors, or simply random chance. Outliers can have a huge impact on machine learning models and if an outlier is included in the training data, it can cause the model to learn the wrong thing which can result in having inaccurate predictions and poor model performance. There are a number of different methods for detecting outliers but here boxplot and z_score method for detecting outliers are presented.

Fig(1.25) shows the boxplot of wind_speed for visualising outliers and from Fig(1.26) the distribution of wind_speed can be seen which is between 3 and 7 approximately.

A data point's distance from the overall data mean is measured using a z_score. It is determined by first finding the mean of the data, then finding the standard deviation of the data, and finally dividing that total by the standard deviation. In this example, outliers are identified by finding all of the data points that have a z_score of more than 3. Fig(1.27) represents the outliers obtained using this method.

Now that the outliers are obtained, there are various methods for handling them. In this assignment, winsorisation method is used for handling these outliers. The Winsorise function takes two arguments: the data to be winsorised and the probabilities to use. The probabilities are the percentage of data points to be winsorised at each end of the distribution. In this case, the probabili-

```{r}
ggplot(df, aes(y = wind_speed))+
  geom_boxplot(fill= "blue", alpha = 0.7)+
  labs(title = "Box Plot of wind_speed", y = "wind_speed")
```

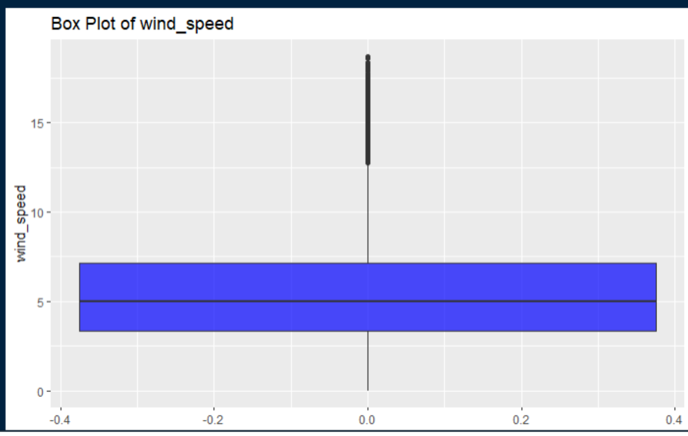


Figure 1.25: Identifying outliers of wind_speed using boxplot

```{r}
df %>%
  select(wind_speed) %>%
  summarise(
    lower_extremes = quantile(wind_speed, 0),
    lower_quartile = quantile(wind_speed, 0.25),
    median = quantile(wind_speed, 0.5),
    upper_quartile = quantile(wind_speed, 0.75),
    upper_extremes = quantile(wind_speed, 1),
    random_quartile = quantile(wind_speed, 0.8)
  )
```

Description: df [1 × 6]

lower_extremes <dbl>	lower_quartile <dbl>	median <dbl>	upper_quartile <dbl>	upper_extremes <dbl>	random_quartile <dbl>
0	3.354102	5.001	7.119691	18.6893	7.651078

1 row

Figure 1.26: Distribution of wind_speed

ties are 0.05 and 0.95, which means that 5% of the data points at each end of the distribution will be winsorised. Fig(1.28) shows how this method is used. And Fig(1.29) represents the boxplot of wind_speed after handling outliers. Then, wind_speed column in the df dataset will be repleced by this column(Fig(1.30)).

1.4.2 Creating a new variable to represent time

At this stage, it is time to select the desired location from the cleaned dataset. The selected location is in the North Eea, near Offshore-Windpark Beatrice which has Latitude = 56.781 and Longitude = 3.508. Fig(1.31) shows how this location is chosen. Now the df dataset has 248 rows and 2 columns which one of them represents date and the other shows wind_speed column. The structure of the df dataset is checked in Fig(1.32).

```r
```{r}
data <- df$wind_speed
z_scores <- scale(data)
threshold <- 3
z_outliers <- data[abs(z_scores) > threshold]
print(z_outliers)
```
```

```
  [1] 14.00893 13.75245 14.09433 14.60137 13.82787 13.80580 13.90432 15.31176 14.60137 14.99367 14.75568 14.39618 14.33876 14.52481 13.83980 13.88128
 [17] 15.36392 16.80238 15.21611 15.14761 14.50310 14.39236 14.80000 14.43087 13.75245 15.73340 17.27918 14.92146 14.84082 14.20563 14.44715 15.02165
 [33] 14.74619 13.72953 16.75291 17.97248 15.90126 15.63106 14.50552 14.14390 13.75245 14.30559 14.33771 14.76787 14.60137 16.62077 17.71158 16.98529
 [49] 15.83982 14.20035 13.96066 15.00300 14.11524 14.64548 14.27620 15.88584 17.90754 17.31704 14.14390 14.67413 15.60417 15.00833 14.55816 14.77329
 [65] 13.91761 14.89295 15.18321 14.66697 15.53641 15.50484 14.68741 13.80036 14.94021 14.45856 14.82734 15.48580 13.75682 14.06023 14.80135 13.72443
 [81] 14.14001 13.79021 13.81919 15.16080 16.82409 16.95170 14.07302 15.17959 15.61602 15.25549 13.87516 15.08145 15.64161 16.13753 13.77861 14.90503
 [97] 15.65535 16.01125 13.87516 15.19901 15.76864 14.62164 14.69864 15.49839 14.99333 14.56743 14.60000 14.21724 14.17956 13.80000 13.71313 13.88128
[113] 13.92300 13.86110 14.31712 14.20880 13.77861 13.82932 13.72953 13.80036 13.90863 13.90863 13.90899 14.20211 15.20526 14.51689 14.20880 13.96460
[129] 13.78840 14.09433 14.32655 13.78840 13.96352 13.80145 14.10284 15.41590 16.75858 14.50552 14.70034 14.46167 14.30035 14.16933 14.30000 14.33771
[145] 14.14390 13.89244 15.37335 16.85141 14.77566 14.75568 14.41804 14.41007 14.67310 14.64138 13.75245 15.84929 17.34416 14.30874 14.53410 14.54132
[161] 14.30140 14.46686 14.98933 14.99667 13.92300 16.86891 18.02360 16.40274 15.16443 14.32201 14.17921 13.77135 14.20141 14.49000 14.64138 14.76482
[177] 16.73589 17.96274 17.35310 14.02070 13.81955 13.98070 14.80304 14.23376 14.49207 14.36001 16.18054 18.31748 17.85721 13.73208 14.90906 14.55816
[193] 14.67583 14.30035 14.94021 15.15850 14.73635 15.07481 14.17956 14.80135 14.08013 14.53444 14.04279 13.86579 14.00643 15.02664 16.58493 16.89083
[209] 14.48862 15.27776 15.65535 15.37335 14.21724 15.29706 15.66972 16.26038 14.09433 15.12118 15.66972 16.24438 13.92157 15.45639 15.98124 14.42810
[225] 15.30523 15.81455 15.45380 14.65094 14.68741 14.80135 14.09291 14.38923 14.10142 13.91294 14.28986 14.11737 14.16933 14.51689 14.41700 14.01178
[241] 14.06307 14.24675 14.06165 14.00321 14.41284 15.30817 16.49060 14.81216 14.30315 13.87984 14.09433 13.82932 13.99786 14.09433 14.12232 13.90000
[257] 14.21443 15.52063 16.78720 14.00321 14.80540 14.34225 13.70584 14.90134 14.06023 14.30000 14.20880 13.77135 14.36001 13.80036 14.00357 15.48289
[273] 16.88698 14.52067 14.38089 14.35618 14.37359 14.54648 14.98833 13.85208 16.06393 17.41407 13.82932 13.94633 14.10142 14.49034 13.70584 14.49000
[289] 14.83139 15.30261 14.12268 17.08362 17.98055 16.81904 13.98070 14.17780 13.89244 14.10000 14.51654 14.57978 15.02698 13.81774 16.83449 18.15324
[305] 15.13052 15.43017 13.74373 13.78296 14.10284 14.70136 14.35688 14.24430 14.57944 16.34166 18.37743 18.28497 14.02462 13.93018 13.82787 14.34050
[321] 14.46790 14.43087 14.83678 14.87616 15.33101 14.65094 13.70584 15.57081 15.83982 14.79797 13.85785 14.55232 14.96262 14.42498 14.72990 14.04279
[337] 14.38923 13.93449 14.27796 14.73126 16.43442 17.36923 14.72175 13.90009 13.90863 13.82787 15.57241 15.98124 15.49355 13.74191 14.30035 15.51290
[353] 15.60160 16.18950 13.76408 15.31698 15.53641 16.18950 15.67323 15.93047 14.46790 15.38441 14.28741 14.89295 15.89277 16.23607 14.17674 14.69864
[369] 14.80135 13.84233 14.49000 13.76445 14.69864 14.10319 14.01749 14.58801 13.80580 14.31712 14.47791 13.70584 14.61232 14.62190 14.10319 14.23517
```

Figure 1.27: Detecting outliers of wind_speed coulmn using z_score method

```r
```{r}
winsorized_data <- Winsorize(data, probs = c(0.05, 0.95))
data[z_outliers] <- winsorized_data[z_outliers]
data <- winsorized_data
print(data)
```
```

```
  [1] 10.508092 10.335376  9.362158  5.643580  6.281720  6.767570  7.947956  9.620811 10.508092 10.508092 10.508092 10.508092 10.508092  3.982462
 [15]  7.810250  6.789698  8.805112  6.767016  4.301163  2.920616  4.427189  4.884670  6.935416  5.069517  6.044005  3.448188  5.375872  4.244997
 [29]  3.820995  3.982462  6.030755  6.161169  4.509989  4.070626  3.905125  2.549510  2.863564  4.661545  5.866856  5.093133  3.448188  2.154066
 [43]  1.622498  2.158703  3.301515  4.301163  4.965884  4.482187  3.006659  2.280351  1.622498  1.622498  2.501999  4.130375  5.536244  6.341136
 [57]  5.200961  3.929377  4.440721  3.883298  3.337664  4.060788  1.622498  4.301163  6.670832  4.404543  4.386342  1.622498  3.500000  3.201562
 [71]  1.627882  1.780449  3.901282  4.884670  9.141663  6.894200  2.140093  2.109502  1.622498  2.158703  6.224147  6.937759  7.840918  6.439915
 [85]  7.495332 10.508092 10.508092 10.371114  7.648529  5.481788  3.601389  1.923538  1.622498  2.024846  1.622498  1.978004  4.123106  6.382006
 [99]  4.900255  4.427189  1.802776  1.622498  2.778489  3.036445  5.869412  8.676981  8.989994  9.052072  9.560335  9.002777  8.962700  8.448077
[113]  7.382412  7.256721  7.595393  7.170077  6.549046  6.007495  5.671860  5.316014  5.600893  8.188406  8.315047 10.508092  9.861541  9.832599
[127] 10.357606  9.963935  9.918165  9.533625  9.126883  8.800000  8.200610  7.516648  7.602631  6.300794  5.069517  4.701064  5.140039  4.494441
[141]  3.601389  3.176476  2.267157  1.622498  1.622498  1.622498  1.622498  1.622498  1.835756  2.701851  3.667424  4.973932  4.661545  3.748333
[155]  3.138471  3.959798  4.535416  6.139218  6.462198  5.818075  3.939543  2.501999  3.584690  3.605551  4.742362  5.500000  5.964059  6.107373
[169]  6.315853  7.376313  7.278049  7.900633  8.100000  8.300602  8.438009  8.400595  8.130191  8.287340  7.930952  8.089499  8.256513  8.028076
[183]  8.065358  7.969944  8.207314  7.036512  6.363961  5.185557  5.162364  6.060528  6.105735  6.129437  6.212890  6.652819  7.427651  7.430343
[197]  6.293648  6.184658  6.082146  5.500909  4.939636  5.300943  6.618912  8.045496  7.793587  7.963040  7.798718  8.015610  8.487638  9.108238
[211]  9.492102  8.386298  8.121576  8.590693  8.296987  7.404053  6.212890  4.136726  4.356604  4.816638  5.153882  5.573150  6.016644  6.530697
[225]  7.247068  7.829432  7.720104  7.495332  8.104937  8.713208  8.747571  8.417244  8.122192  7.045566  6.500769  4.554119  1.835756  1.622498
[239]  1.622498  1.746425  2.690725  1.702939  1.622498  2.459675  4.420407  5.247857  5.126402  4.517743  4.295346  5.821512  7.863841  8.935883
[253]  9.108238  7.543209  3.716181  4.341659  4.220190  3.716181  4.318565  7.040597  7.446476  7.596052  5.913544  3.883298  3.522783  4.159327
[267]  6.328507  6.129437  7.017834  7.220111  6.332456  3.939543  4.338202  3.905125  4.720169  4.701064  5.953990  5.993330  5.793962  2.801785
[281]  4.440721  4.254409  5.323533  5.382379  6.700746  7.689603  5.913544  2.846050  2.900000  3.257299  2.601922  3.106445  5.140039  5.630275
[295]  5.315073  2.549510  1.984943  2.801785  3.640055  4.148494  4.548626  4.750789  4.140048  4.438468  5.470832  5.629387  5.515433  7.514153
[309]  8.200610  7.858753  6.830813  3.679674  4.838388  6.525335  7.083078  8.561542 10.508092  9.904040  8.888757  4.410215  4.081666  3.894868
[323]  4.420407  7.209022  6.673080  6.087693  4.870318  6.453681  7.741447  9.377633 10.508092  8.702299  7.829432  8.854377  6.705222  2.860070
[337]  2.828427  3.255764  2.507987  3.354102  4.081666  4.904080  5.412024  3.220248  3.667424  3.492850  3.889730  4.743416  4.252058  4.372928
```

Figure 1.28: Handling outliers using winsorisation method

Fig(1.33) and (1.34) show the line graph which shows the wind speed over time.

Fig(1.35) shows how a variable is created to represent time. This is very useful for building machine learning models.

1.4.3 Splitting data into train and test

Splitting the data into a training set and a test set is important because it helps to prevent overfitting. Overfitting is a problem that occurs when a machine learning model learns the training data too well and is unable to generalize to new data. By splitting the data into a training set and a test set, it can

```{r}
boxplot(data, main = "Box Plot of wind_speed after Handling Outliers")
```

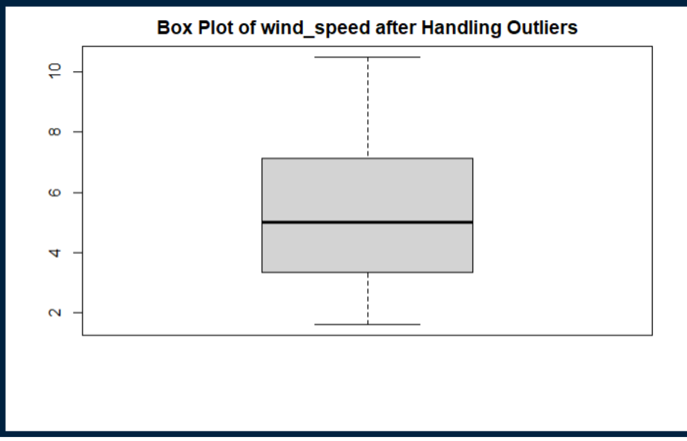


Figure 1.29: The boxplot of wind_speed after handling outliers

```{r}
df$wind_speed <- data
df
```

Description: df [99,200 × 14]

PSFC <dbl>	U10 <dbl>	V10 <dbl>	Q2 <dbl>	Rainc <dbl>	Rainnc <dbl>	Snow <dbl>	TSLB <dbl>	SMOIS <dbl>	wind_speed <dbl>
100558.0	9.30	-7.700	0.005020	0.0	0.0000000	0	273.20	1.00000	10.508163
100558.0	9.10	-4.900	0.004540	0.0	0.0000000	0	273.20	1.00000	10.335376
100657.0	9.10	-2.200	0.003900	0.0	0.0000000	0	273.20	1.00000	9.362158
100748.0	5.60	0.700	0.005320	0.0	0.0000000	0	273.20	1.00000	5.643580
100757.0	1.50	6.100	0.004920	0.0	0.0000000	0	273.20	1.00000	6.281720
100654.0	2.20	6.400	0.005700	0.0	0.0000000	0	273.20	1.00000	6.767570
100633.0	2.90	7.400	0.005730	0.0	0.0000000	0	273.20	1.00000	7.947956
100621.0	3.40	9.000	0.005710	0.0	0.0000000	0	273.20	1.00000	9.620811
100551.0	1.60	11.900	0.005480	0.0	0.0000000	0	273.20	1.00000	10.508163
100387.0	-0.40	12.600	0.005665	0.0	0.0000000	0	273.20	1.00000	10.508163

1-10 of 99,200 rows | 5-14 of 14 columns   Previous 1 2 3 4 5 6 ... 100 Next

Figure 1.30: Replacing wind_speed column with new column

```{r}
df %>%
  filter(Latitude  == 56.781 & Longitude == 3.508) %>%
  select(date, wind_speed) -> df
df
```

Description: df [248 × 2]

date <S3: POSIXct>	wind_speed <dbl>
2018-05-01 00:00:00	10.508163
2018-05-01 03:00:00	10.373042
2018-05-01 06:00:00	9.126883
2018-05-01 09:00:00	8.254090
2018-05-01 12:00:00	6.726812
2018-05-01 15:00:00	7.051950
2018-05-01 18:00:00	6.293648
2018-05-01 21:00:00	7.433034
2018-05-02 00:00:00	9.400532
2018-05-02 03:00:00	10.508163

1-10 of 248 rows   Previous 1 2 3 4 5 6 ... 25 Next

Figure 1.31: Choosing the selected location and columns from the dataset

```{r}
str(df)
```

```
'data.frame':	248 obs. of  2 variables:
 $ date      : POSIXct, format: "2018-05-01 00:00:00" "2018-05-01 03:00:00" "2018-05-01 06:00:00" "2018-05-01 09:00:00" ...
 $ wind_speed: num  10.51 10.37 9.13 8.25 6.73 ...
```

Figure 1.32: A summary of the df dataset

```{r}
ggplot(df, aes(x = date, y = wind_speed)) +
  geom_line(color = "blue") +
  labs(title = "wind speed Over Time",
       x = "Date and Time",
       y = "wind speed") +
  theme_minimal()
```

Figure 1.33: A plot of the df dataset

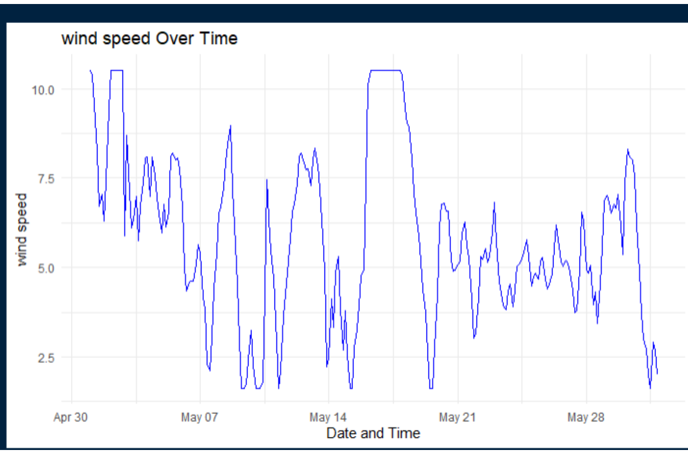


Figure 1.34: A plot of the df dataset

```{r}
df <- df %>%
  mutate(time = as.numeric(difftime(date, min(date), units = "hours")))
df
```

Description: df [248 x 3]

date <S3: POSIXct>	wind_speed <dbl>	time <dbl>
2018-05-01 00:00:00	10.508163	0
2018-05-01 03:00:00	10.373042	3
2018-05-01 06:00:00	9.126883	6
2018-05-01 09:00:00	8.254090	9
2018-05-01 12:00:00	6.726812	12
2018-05-01 15:00:00	7.051950	15
2018-05-01 18:00:00	6.293648	18
2018-05-01 21:00:00	7.433034	21
2018-05-02 00:00:00	9.400532	24
2018-05-02 03:00:00	10.508163	27

1-10 of 248 rows Previous 1 2 3 4 5 6 ... 25 Next

Figure 1.35: Creating a new variable to represent time

be ensured that the model is not overfitting to the training data. The training data is used to traing the model and using test data the performance of the model can be evaluated.

The code in Fig(1.36) sets the seed to 123, then randomly samples 80% of the rows from the df data frame and assigns them to the train_data variable which is 198 observations in this case. The remaining 20% of the rows are assigned to the test_data variable(50 observations). Now, it is time to create models.

````
```{r}
set.seed(123)
train_indices <- sample(1:nrow(df), 0.8 * nrow(df))
train_data <- df[train_indices, ]
test_data <- df[-train_indices, ]
str(train_data)
str(test_data)
```

'data.frame':   198 obs. of  3 variables:
 $ date      : POSIXct, format: "2018-05-20 18:00:00" "2018-05-26 18:00:00" "2018-05-23 06:00:00" "2018-05-02 15:00:00" ...
 $ wind_speed: num  4.9 5.03 4.6 10.51 4.85 ...
 $ time      : num  474 618 534 39 582 507 147 351 126 684 ...
'data.frame':   50 obs. of  3 variables:
 $ date      : POSIXct, format: "2018-05-01 03:00:00" "2018-05-01 06:00:00" "2018-05-02 09:00:00" "2018-05-02 18:00:00" ...
 $ wind_speed: num  10.37 9.13 10.51 10.51 7.13 ...
 $ time      : num  3 6 33 42 51 54 78 81 90 108 ...
````

Figure 1.36: Splitting data into train and test

1.4.4 ARIMA model

The first model that is created in this assignment is ARIMA model. Fig(1.37) shows an ARIMA model that has been fit to the wind speed data. The ARIMA model is a statistical model that can be used to forecast time series data. The model is made up of three components: the autoregressive (AR) component, the moving average (MA) component, and the integrated (I) component. The summary of the ARIMA model shows the parameters of the model, the AIC which is 907.32 and BIC scores of the model, which is 917.19. The AIC and BIC scores are measures of the model's fit to the data. Fig(1.38) forecasts future values of the wind speed data using the ARIMA model. The accuracy of the model is evaluated using RMSE which is root mean squared error.

````
```{r}
library(forecast)
#fit the ARIMA model
arima_model <- auto.arima(train_data$wind_speed)
summary(arima_model)
```

Series: train_data$wind_speed
ARIMA(0,0,1) with non-zero mean

Coefficients:
          ma1    mean
      -0.1232  5.8231
s.e.   0.0730  0.1469

sigma^2 = 5.609:  log likelihood = -450.66
AIC=907.32   AICc=907.45   BIC=917.19

Training set error measures:
                        ME     RMSE      MAE       MPE     MAPE      MASE        ACF1
Training set -0.0006756893 2.356304 1.915944 -24.31382 46.31712 0.6709512 0.005678639
````

Figure 1.37: Training ARIMA model using train dataset

```{r}
# forecast
arima_forecast <- forecast(arima_model, h = length(test_data$wind_speed))

# evaluate
arima_accuracy <- accuracy(arima_forecast, test_data$wind_speed)
cat("ARIMA Model Accuracy:", arima_accuracy[, 'RMSE'])
```

```
ARIMA Model Accuracy: 2.356304 2.375463
```

Figure 1.38: Testing ARIMA model using test dataset

1.4.5 LR model

The second model which has been trained here is linear regression. Fig(1.39) shows that a linear regression model is fit to the training set. The model is then used to predict wind speed values for the test set. The RMSE for this model is 2.231711 which is better than the ARIMA model. Fig(1.40) and Fig(1.41) show the plot of actual and predicted wind_speed.

```{r}
# Fit the model on the training set
train_model <- lm(wind_speed ~ time, data = train_data)
summary(train_model)

# Predict wind_speed values for the test set
predictions <- predict(train_model, newdata = test_data)

# Calculate the root mean squared error (RMSE)
rmse_lr <- sqrt(mean((test_data$wind_speed - predictions)^2))
cat("RMSE_lr:", rmse_lr)
```

```
Call:
lm(formula = wind_speed ~ time, data = train_data)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.6113 -1.4693 -0.1405  1.4942  4.7846 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  6.7218385  0.3276559  20.515  < 2e-16 ***
time        -0.0024649  0.0007758  -3.177  0.00173 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.325 on 196 degrees of freedom
Multiple R-squared:  0.04898,	Adjusted R-squared:  0.04413 
F-statistic:  10.1 on 1 and 196 DF,  p-value: 0.001727

RMSE_lr: 2.231711
```

Figure 1.39: Training and testing linear regression model

```{r}
#Plot Actual vs Predicted values
ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = predictions), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()
```

Figure 1.40: The plot of actual and predicted values for LR model

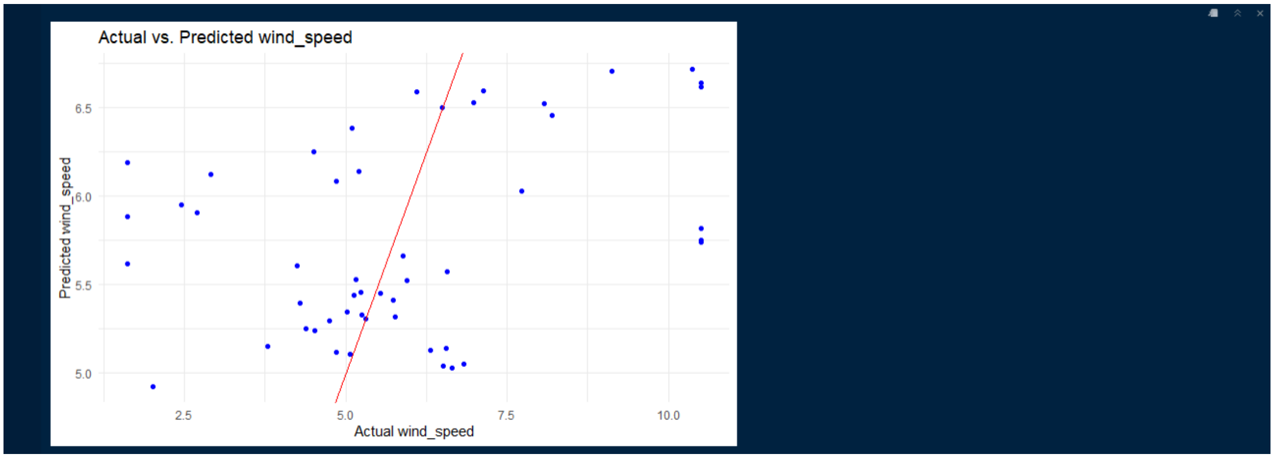


Figure 1.41: The plot of actual and predicted values for LR model

1.4.6 SVR model

The third model which is created here, is Support Vector Regression. It is a supervised learning algorithm that can be used to solve regression problems. SVR is a kernel-based algorithm, which means that it uses a kernel function to map the data into a higher-dimensional space where it can be more easily separated. Fig(1.42) fits an SVR model on the training set using the radial basis function (RBF) kernel. The RBF kernel is a popular kernel function that is known for its good performance on a variety of problems. The code then displays the summary of the SVR model. The summary will show the parameters of the model, the training error, and the test error. Fig(1.43) fits an SVR model on the training set using the linear kernel. In Fig(1.44) an SVR model using polynomial kernel has been created on the training set. As it can be seen here, the number of support vectors for RBF kernel, linear and polynomial kernels are 180, 176 and 176, respectively. The other information can be seen in each figure. In Fig(1.45) the SVR models then used to predict wind_speed values using the test set and the RMSE values for each models is shown which means that the SVR model using RBF kernel has better accuracy than the other two.

In Fig(1.45) the SVR models then used to predict wind_speed values using the test set and the RMSE values for each models is shown which means that the SVR model using RBF kernel has better accuracy than the other two. The plots of actual and predicted wind_speed using SVR modells are presented in Fig(1.46) and (1.47).

```
```{r}
options(warn = 2)
library(e1071)

# Fit an SVR model on the training set
# 1.Radial basis function (RBF) kernel
svr_model_RBF <- svm(wind_speed ~ time, data = train_data, kernel = "radial")
# Display the SVR model summary
summary(svr_model_RBF)
```
```

```
Call:
svm(formula = wind_speed ~ time, data = train_data, kernel = "radial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  180
```

Figure 1.42: Fit an SVR model on the training set using RBF kernel

```
```{r}
# 2.Linear kernel
svr_model_linear <- svm(wind_speed ~ time, data = train_data, kernel = "linear")
# Display the SVR model summary
summary(svr_model_linear)
```
```

```
Call:
svm(formula = wind_speed ~ time, data = train_data, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  176
```

Figure 1.43: Fit an SVR model on the training set using linear kernel

```
```{r}
# 3.Polynomial kernel
svr_model_poly <- svm(wind_speed ~ time, data = train_data, kernel = "polynomial")
# Display the SVR model summary
summary(svr_model_poly)
```
```

```
Call:
svm(formula = wind_speed ~ time, data = train_data, kernel = "polynomial")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
      gamma:  1
     coef.0:  0
    epsilon:  0.1


Number of Support Vectors:  176
```

Figure 1.44: Fit an SVR model on the training set using polynomial kernel

1.4.7 RF model

The last model that will be discussed here is Random Forest model. The Random Forest model is a type of machine learning algorithm that is known for its high accuracy and robustness. The model is

```r
```{r}
# Predict wind_speed values for the test set using the SVR models
svr_predictions_RBF <- predict(svr_model_RBF, newdata = test_data)
svr_predictions_linear <- predict(svr_model_linear, newdata = test_data)
svr_predictions_poly <- predict(svr_model_poly, newdata = test_data)

# Calculate the root mean squared error (RMSE) for the SVR models
svr_rmse_RBF <- sqrt(mean((test_data$wind_speed - svr_predictions_RBF)^2))
svr_rmse_linear <- sqrt(mean((test_data$wind_speed - svr_predictions_linear)^2))
svr_rmse_poly <- sqrt(mean((test_data$wind_speed - svr_predictions_poly)^2))
cat("SVR_RBF RMSE:", svr_rmse_RBF, "\n")
cat("SVR_linear RMSE:", svr_rmse_linear, "\n")
cat("SVR_poly RMSE:", svr_rmse_poly, "\n")
```

SVR_RBF RMSE: 1.978381
SVR_linear RMSE: 2.223657
SVR_poly RMSE: 2.140433
```

Figure 1.45: Testing SVR models on the testing set

```r
```{r}
# Plot the actual vs. predicted values for the SVR models
p1 <- ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = svr_predictions_RBF), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "SVR_RBF: Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()+
  theme(text = element_text(size = 8))

p2 <- ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = svr_predictions_linear), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "SVR_linear: Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()+
  theme(text = element_text(size = 8))

p3 <- ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = svr_predictions_poly), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "SVR_poly: Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()+
  theme(text = element_text(size = 8))

grid.arrange(p1, p2, p3, ncol = 1)
```
```

Figure 1.46: The plot of actual and predicted values for SVR models

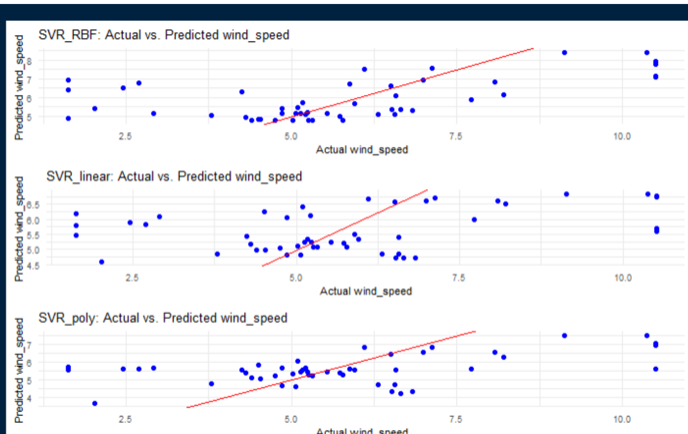


Figure 1.47: The plot of actual and predicted values for SVR models

built by training a number of decision trees on different subsets of the training data. The predictions of the individual decision trees are then combined to produce the final prediction of the Random Forest model. In this assignment, three random forest models have been built using 100, 200 and 300 trees and then the RMSE of the models are evaluated.

Fig(1.48),(1.49) and (1.50) show training the three RF models using training dataset and in Fig(1.51), the models then are tested to predict wind_speed using test set. It also can be seen that the RMSE for these three models are evaluated. It seems that the model with 200 trees has better accuracy than others. The actual and predicted values are shown in Fig(1.52) and (1.53).

```
```{r}
# Fit a Random Forest model on the training set
rf_model_n100 <- randomForest(wind_speed ~ time, data = train_data, ntree = 100)
# Display the Random Forest model summary
summary(rf_model_n100)
```

                Length Class Mode
call   4   -none- call
type   1   -none- character
predicted   198   -none- numeric
mse   100   -none- numeric
rsq   100   -none- numeric
oob.times   198   -none- numeric
importance   1   -none- numeric
importanceSD   0   -none- NULL
localImportance   0   -none- NULL
proximity   0   -none- NULL
ntree   1   -none- numeric
mtry   1   -none- numeric
forest   11   -none- list
coefs   0   -none- NULL
y   198   -none- numeric
test   0   -none- NULL
inbag   0   -none- NULL
terms   3   terms call
```

Figure 1.48: Fitting an RF model on the training set using ntree=100

```
```{r}
# Fit a Random Forest model on the training set
rf_model_n200 <- randomForest(wind_speed ~ time, data = train_data, ntree = 200)
# Display the Random Forest model summary
summary(rf_model_n200)
```

                Length Class Mode
call   4   -none- call
type   1   -none- character
predicted   198   -none- numeric
mse   200   -none- numeric
rsq   200   -none- numeric
oob.times   198   -none- numeric
importance   1   -none- numeric
importanceSD   0   -none- NULL
localImportance   0   -none- NULL
proximity   0   -none- NULL
ntree   1   -none- numeric
mtry   1   -none- numeric
forest   11   -none- list
coefs   0   -none- NULL
y   198   -none- numeric
test   0   -none- NULL
inbag   0   -none- NULL
terms   3   terms call
```

Figure 1.49: Fitting an RF model on the training set using ntree=200

1.5 Evaluation of Different Models

So far, eight models have been created here. There are different ways for calculationg the error and evaluationg models to see which one has better performance. In this assignment, RMSE value for

```{r}
# Fit a Random Forest model on the training set
rf_model_n300 <- randomForest(wind_speed ~ time, data = train_data, ntree = 300)
# Display the Random Forest model summary
summary(rf_model_n300)
```

```
                Length Class  Mode
call              4    -none- call
type              1    -none- character
predicted       198    -none- numeric
mse             300    -none- numeric
rsq             300    -none- numeric
oob.times       198    -none- numeric
importance        1    -none- numeric
importanceSD      0    -none- NULL
localImportance   0    -none- NULL
proximity         0    -none- NULL
ntree             1    -none- numeric
mtry              1    -none- numeric
forest           11    -none- list
coefs             0    -none- NULL
y               198    -none- numeric
test              0    -none- NULL
inbag             0    -none- NULL
terms             3    terms  call
```

Figure 1.50: Fitting an RF model on the training set using ntree=300

```{r}
# Predict wind_speed values for the test set using the Random Forest model
rf_predictions_n100 <- predict(rf_model_n100, newdata = test_data)
rf_predictions_n200 <- predict(rf_model_n200, newdata = test_data)
rf_predictions_n300 <- predict(rf_model_n300, newdata = test_data)

# Calculate the root mean squared error (RMSE) for the Random Forest model
rf_rmse_n100 <- sqrt(mean((test_data$wind_speed - rf_predictions_n100)^2))
rf_rmse_n200 <- sqrt(mean((test_data$wind_speed - rf_predictions_n200)^2))
rf_rmse_n300 <- sqrt(mean((test_data$wind_speed - rf_predictions_n300)^2))
cat("Random Forest n100 RMSE:", rf_rmse_n100, "\n")
cat("Random Forest n200 RMSE:", rf_rmse_n200, "\n")
cat("Random Forest n300 RMSE:", rf_rmse_n300, "\n")
```

```
Random Forest n100 RMSE: 0.7235625
Random Forest n200 RMSE: 0.7158536
Random Forest n300 RMSE: 0.724628
```

Figure 1.51: Testing the RF models on the testing set

```{r}
# Plot the actual vs. predicted values for the Linear Regression, SVR, and Random Forest models
p1 <- ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = rf_predictions_n100), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "Random Forest: Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()+
  theme(text = element_text(size = 8))

p2 <- ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = rf_predictions_n200), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "Random Forest: Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()+
  theme(text = element_text(size = 8))

p3 <- ggplot() +
  geom_point(data = test_data, aes(x = wind_speed, y = rf_predictions_n300), color = "blue") +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  labs(title = "Random Forest: Actual vs. Predicted wind_speed",
       x = "Actual wind_speed",
       y = "Predicted wind_speed") +
  theme_minimal()+
  theme(text = element_text(size = 8))

grid.arrange(p1, p2, p3, ncol = 1)
```

Figure 1.52: The plot of actual and predicted values for RF models

these modells are calculated. Fig(1.54) shows a comparison between these values. The lower RMSE, the better performance the model has.

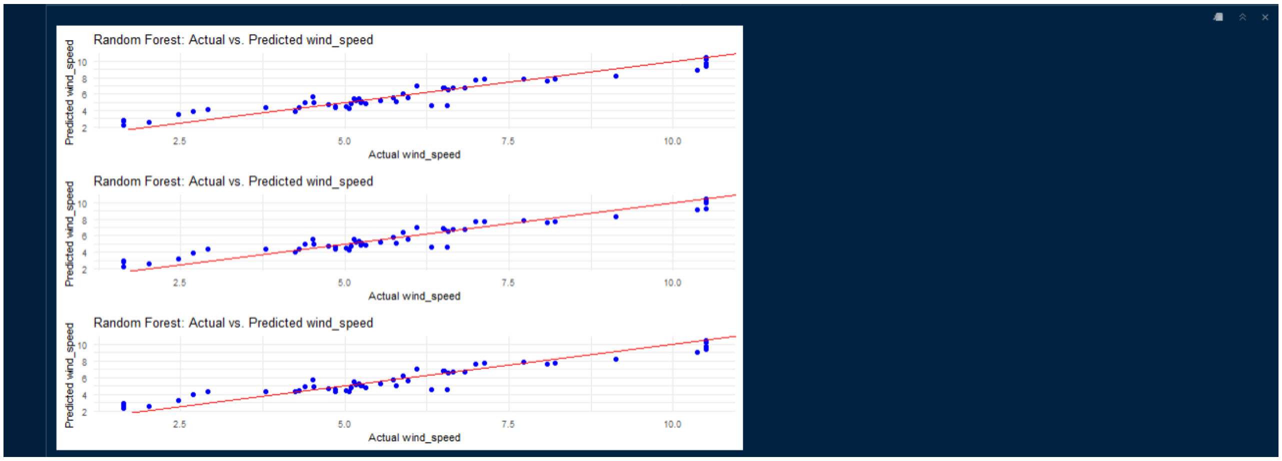


Figure 1.53: The plot of actual and predicted values for RF models

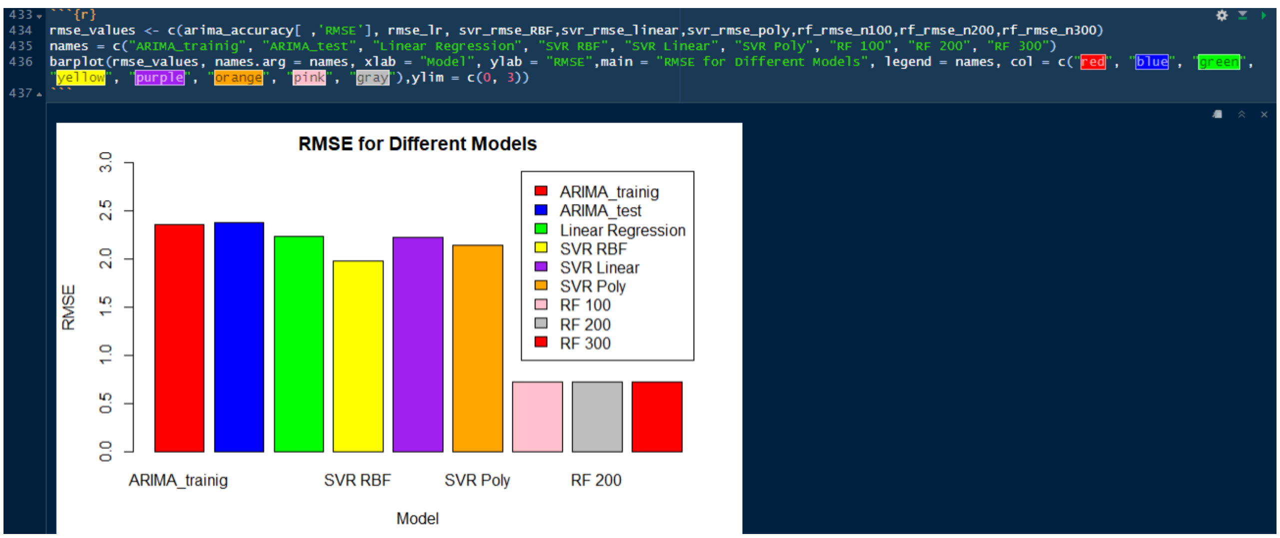


Figure 1.54: A comparison of models based on RMSE values

As it can be seen in Fig(1.54), ARIMA model has higher valuse of RMSE and the lowest value of RMSE is for Random Forest models. Even between three Random Forest models, the model with ntree=200 has the best performance and the RMSE value for this model is about 0.70. It should be noted that RMSE value is not enough for saying that a model performs well and the other factors should be considered.

Conclusion

In conclusion, this project aimed to forecast wind speed in the offshore-windpark Beatrice using four different models: ARIMA, RF, LR, and SVR. models were trained on the given dataset which has the location for this area and they were subsequently evaluated based on their RMSE metrics.

As The results showed the RF model was the most accurate, followed by the SVR model, while the LR and ARIMA models had relatively higher errors. Overall, the project successfully achieved its objectives of predicting wind speed and comparing the performance of different models.

Future research can focus on refining the models' precision and uncovering additional influences on wind speed at offshore wind farms.