

MAY 2023

TIME SERIES DATA ANALYSIS AND FORECASTING IN WIND POWER STATION SITING AND INSTALLATION

**AN ANALYSIS BY
IKENNA JOSEPH CHUKWUDUM**



Abstract

Big data and machine learning have come a long way in not only trend analysis but also in time series data forecasting. These forecasts help inform decisions made in industry with regards geographical location siting of infrastructure, thus ensuring cost management and output optimization. As power generation and management are key to maintaining a sustainable economy in contemporary times, optimisation strategies are always at the forefront of power companies, even more so when it comes to renewable energy solutions. Hence, the importance of data analysis and AI solutions in the renewable energy sector especially in the wake of the “green energy revolution” cannot be over-emphasized. One of the more productive and practical renewable energy solutions is wind energy and as such, wind velocity time series and forecasting were analysed in this report to determine the suitability of specific geographical coordinates for wind power setup.

Keywords: *Renewable Energy; Wind Turbine; Big Data; AI; Artificial Intelligence; Internet of Things; Forecasting; Machine Learning; Time Series; Data Analysis; Correlation; Regression.*

CONTENTS

1. INTRODUCTION	1
1.1 Background.....	2
1.2 Problem Statement.....	2
2. TIME SERIES DATA PRE-PROCESSING, ANALYSIS AND FORECASTING	4
2.1 Cleaning and Filtering	5
2.1.1 Dataset Importing and Anomaly Resolution	5
2.1.2 Correcting the Column Names.....	6
2.1.3 Selecting the 300 Rows for EDA.....	7
2.1.4 Ordering the Latitude and Longitude.....	7
2.1.5 Exploring the Unique Variables.....	8
2.1.6 Linear Interpolation for Missing Values	9
2.2 Wind Velocity Time Series (Statistical) Analysis at Colwyn Bay Geographical Coordinates	11
2.2.1 Selecting Colwyn Bay Coordinates and Wind Parameters	11
2.2.2 The Resultant Velocity/ Datetime Graph	15
2.2.3 Time Series Analysis	16
2.2.4 Autocorrelation and Partial Autocorrelation	18
2.2.5 Augmented Dickey-Fuller Test	19
2.2.6 ARIMA	19
2.3 Wind Velocity Machine Learning Analysis	22
2.3.1 Linear Regression	22
2.3.2 Support Vector Machine Regression and Random Forest	23
2.3.3 Root Mean Square Errors and Plots	25
2.4 Correlation and Testing	27
2.5 Discussion	31
2.5.1 Exploratory Data Analysis.....	31
2.5.2 Answering the Problem Statement	33
3. CHALLENGES IN TIME SERIES ANALYSIS AND FORECASTING RELATING TO WIND POWER STATION SITING	35
4. METHODOLOGY.....	38
4.1 Literature Research Methodology	38

4.2 Analysis Algorithms.....	39
4.2.1 ARIMA	39
4.2.2 Linear Regression	40
4.2.3 Support Vector Regression SVR	40
4.2.4 Random Forest.....	41
5. ABOUT WIND POWER GENERATION AND ITS OPERATION.....	42
5.1 History	42
5.2 Operation.....	43
5.3 A.I in Wind Power Generation	44
6. CONCLUSION	46
References.....	48
Appendix	50

1. INTRODUCTION

It has been estimated that the world would require almost 39 percent more energy than what it currently produces in order to meet existing and growing demand by 2030, according to British Petroleum BP. This requirement would mainly be to meet demand in developing or non-OECD countries, however, OECD countries would still be required to grow their energy supply by at least four percent in order to satiate growing global demand. This ever-growing consumption of energy meets the need for more sustainable and environmentally friendly solutions to energy production, ergo, clean energy. There has therefore never been a better time to introduce AI to the power sector in order to ensure efficiency, effectiveness and optimisation, especially with the exploration of renewable energy sources at its infancy.

Wind energy is a renewable energy source that depends even more broadly on time-series forecasting, enabled by big data analytics, as the losses that would occur with improper or unplanned location of the infrastructure could be enormous. As time series forecasting uses historical data to determine trends and predict future values, it can be used to predict the velocity and consistency of wind in a given geographical location over a given period; information of which is essential in determining wind turbine installation and setup.

AI more broadly can also ensure more efficiently run wind turbines as the technology can detect anomalies and operational defects early. Hence, AI not only assists in wind turbine infrastructure placement, but also in their maintenance and efficiency. It thus empowers the migration of energy production from fossil fuels to the cleaner renewable energy options.

1.1 Background

Power generated through wind energy largely depends on the velocity of the wind reaching the turbines, which belies the need for the adoption of time series forecasting in being able to determine the speed and consistency of wind reaching the turbines at regular time intervals (Higgins & Stathopoulos, 2021). Admittedly, there are a myriad of factors to be considered in the setup of a wind power station such as turbine placement strategies, turbine design, turbine structure maintenance, topography of the site and finally, geographical location of the site. Focusing on the wind velocity parameter, the data was culled from the Weather Research and Forecasting Model WRF, which is a collaborative, open-source and community-driven effort backed by several prominent institutions including the Federal Aviation Administration FAA, the National Centre for Atmospheric Research NCAR, the Naval Research Laboratory, the US Air force, the University of Oklahoma and the National Oceanic and Atmospheric Administration. The WRF model is essentially a complex mesoscale weather prediction system which is flexible and computationally efficient(*Weather Research & Forecasting Model (WRF) / Mesoscale & Microscale Meteorology Laboratory, n.d.*).

1.2 Problem Statement

This report is focused on the assessment of a site with properties deemed fairly conducive for the installation of a wind turbine by exploring the wind velocity parameter in that location over a period in May 2018 at a measurement frequency of 3 hours. The unique wind velocity parameters relating to the given geographical location are explored, forecasted statistically and through machine learning, correlated with other parameters and plotted. Finally, the

results are chronologically interpreted with a view to objectively determine the siting of a wind turbine at the given coordinates.

2. TIME SERIES DATA PRE-PROCESSING, ANALYSIS AND FORECASTING

As mentioned in the earlier chapter, the data set used in the time series analysis was sourced from the Weather Research and Forecasting WRF model and specified a May 2018 time frame that included the parameters: surface temperature, surface pressure, wind speed, humidity, precipitation, soil temperature and soil moisture. Also included were several geographical coordinates where these parameters were recorded at 3-hour intervals for the month.

The aim of this time series analysis was to determine the viability of installing and setting up a wind turbine and power station in a given geographical location with specified coordinates. The coordinates selected were of **latitude 53.394 and longitude -3.768** which pointed to **Colwyn Bay, North Wales**. Colwyn Bay is a coastal town and as such has been touted to have high potential for wind resources in wind velocity, direction and consistency. Hence, this analysis explored the wind velocity parameter of the town to determine site suitability of a wind-backed power generating station.

R was the principal tool used in the pre-processing, plotting, exploratory data analysis, statistical analysis and machine learning algorithms applied to the time series data set. The WRF data set is a time series as its variables are measured at regular time intervals and it aims to capture trends and evolution of the variable with time(Shumway & Stoffer, 2017).

The pre-processing of the data was initiated with the importing of the dataset into R studio after which it was cleaned, wrangled, filtered and formatted to prepare it for exploratory and statistical analysis. Machine learning and forecasting techniques were adopted to predict resultant wind velocity over time and also demonstrate correlation with other parameters of

* The complete step-by-step processes involved in the data analysis in R can be accessed in the Appendix section of this report.

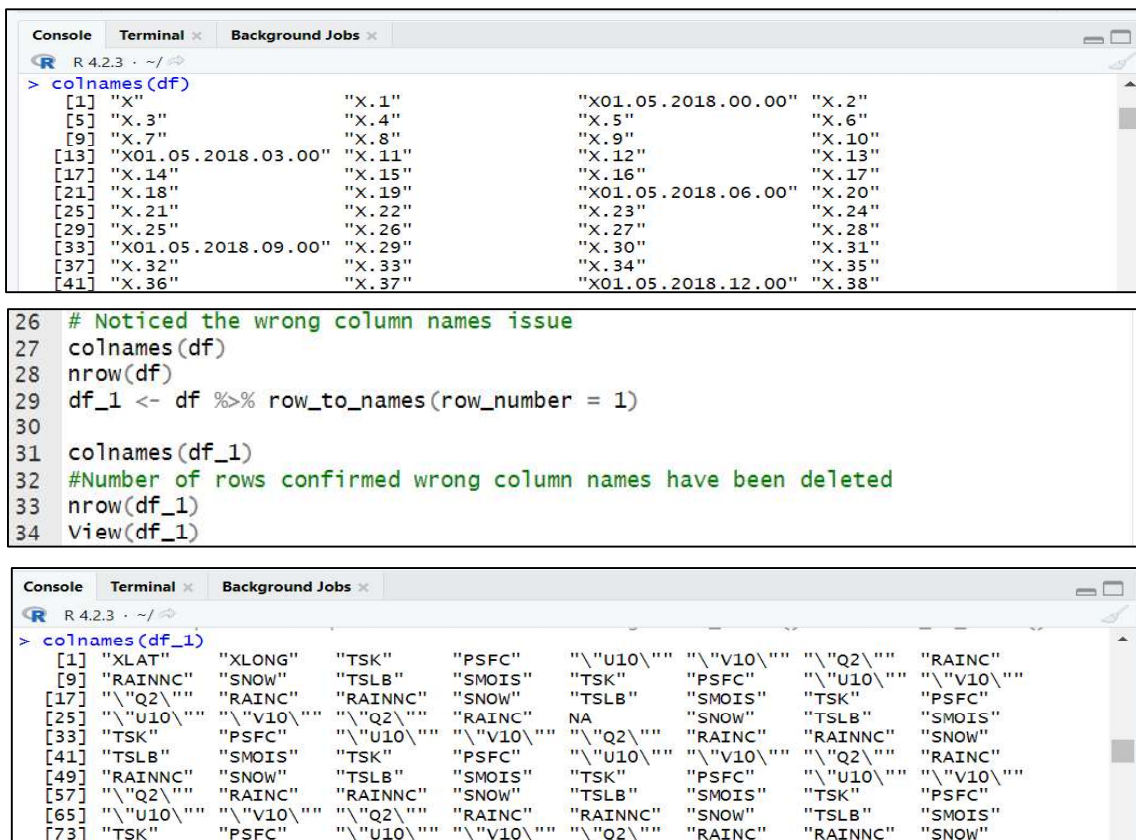
2.1.1 Dataset Importing and Anomaly Resolution

```
1- #####  
2- ### EXPLORATORY DATA ANALYSIS OF THE ###  
3- ### WEATHER RESEARCH AND FORECASTING MODEL DATA ###  
4  
5 # Imported the .csv file  
6 df <- read.csv("wRFdata_May2018.csv")  
7 View(df)  
8 class(df)  
9 nrow(df)  
10 ncol(df)  
  
11  
12 # Imported janitor, dplyr and tidyverse libraries to clean data frame and  
13 # correct column names  
14 library(janitor)  
15 library(dplyr)  
16 library(tidyverse)  
  
17 anomaly_loc <- which(df == "[ ]", arr.ind = TRUE)  
18 anomaly_loc # Print the row and column indexes where "&" is found  
  
19  
20 colnames(df)[6]  
21 df <- df %>%  
22 mutate(X.4 = replace(X.4, match("[ ]", X.4), NA))  
23 View(df)
```

5

2.1.2 Correcting the Column Names

With the anomaly resolved, the focus moved to the unsuitable/wrong column names. The aim at this point was to ensure the relevant variables of longitude, latitude and the other WRF parameters of surface temperature TSK, surface pressure PSFC, wind velocity components U10 and V10, amongst others, were the correctly named as column names replacing the generic column names X, X.1, X.2, etc. This would enable for ease of filtering for the relevant values per variable. This step was also to prepare for the selection of 300 rows for a deeper exploratory data analysis. The `row_to_names()` function played a relevant part in this step, which took an argument of “row number = 1”, implying that the current column names would be replaced by the names in row 1(which is where the correct variable names were). A new data frame `df_1` was initiated in the process as displayed in slide 2.



The screenshot shows an R console window with three tabs: Console, Terminal, and Background Jobs. The console output is as follows:

```
R 4.2.3 ~ /
> colnames(df)
[1] "X" "X.1" "X01.05.2018.00.00" "X.2"
[5] "X.3" "X.4" "X.5" "X.6"
[9] "X.7" "X.8" "X.9" "X.10"
[13] "X01.05.2018.03.00" "X.11" "X.12" "X.13"
[17] "X.14" "X.15" "X.16" "X.17"
[21] "X.18" "X.19" "X01.05.2018.06.00" "X.20"
[25] "X.21" "X.22" "X.23" "X.24"
[29] "X.25" "X.26" "X.27" "X.28"
[33] "X01.05.2018.09.00" "X.29" "X.30" "X.31"
[37] "X.32" "X.33" "X.34" "X.35"
[41] "X.36" "X.37" "X01.05.2018.12.00" "X.38"
```

```
26 # Noticed the wrong column names issue
27 colnames(df)
28 nrow(df)
29 df_1 <- df %>% row_to_names(row_number = 1)
30
31 colnames(df_1)
32 #Number of rows confirmed wrong column names have been deleted
33 nrow(df_1)
34 view(df_1)
```

The second screenshot shows the console output after running the code:

```
R 4.2.3 ~ /
> colnames(df_1)
[1] "XLAT" "XLONG" "TSK" "PSFC" "\"U10\""" "\"V10\""" "\"Q2\""" "RAINC"
[9] "RAINNC" "SNOW" "TSLB" "SMOIS" "TSK" "PSFC" "\"U10\""" "\"V10\"""
[17] "\"Q2\""" "RAINC" "RAINNC" "SNOW" "TSLB" "SMOIS" "TSK" "PSFC"
[25] "\"U10\""" "\"V10\""" "\"Q2\""" "RAINC" NA "SNOW" "TSLB" "SMOIS"
[33] "TSK" "PSFC" "\"U10\""" "\"V10\""" "\"Q2\""" "RAINC" "RAINNC" "SNOW"
[41] "TSLB" "SMOIS" "TSK" "PSFC" "\"U10\""" "\"V10\""" "\"Q2\""" "RAINC"
[49] "RAINNC" "SNOW" "TSLB" "SMOIS" "TSK" "PSFC" "\"U10\""" "\"V10\"""
[57] "\"Q2\""" "RAINC" "RAINNC" "SNOW" "TSLB" "SMOIS" "TSK" "PSFC"
[65] "\"U10\""" "\"V10\""" "\"Q2\""" "RAINC" "RAINNC" "SNOW" "TSLB" "SMOIS"
[73] "TSK" "PSFC" "\"U10\""" "\"V10\""" "\"Q2\""" "RAINC" "RAINNC" "SNOW"
```

Slide 2: Wrong column names corrected

2.1.3 Selecting the 300 Rows for EDA

Thereafter, the exploratory data analysis of the first 300 rows was started with the acknowledgement of the latitude XLAT and longitude XLONG variables as the reference variables for the data set. As such, missing values NAs in either of these columns would lead to row removal due to the sensitivity of the variables as geographical coordinates. Hence, the first 350 rows of df_1 were selected in the data frame and piped to the drop_na() function to remove rows with missing XLAT or XLONG. This process initialised data frame df_2 as seen in slide 3.

The number of rows left after this step was 325, after which the top 300 rows were indexed to form eda300_df, which is the data frame of the first 300 rows exploratory data analysis would be performed on.

```
36 #####
37 ### EDA WITH FIRST 300 ROWS ###
38 # In view of the fact that missing XLAT and XLONG figures could not simply
39 # be averaged or randomly imputed, over 300 rows were selected as those
40 # with either a missing XLAT or XLONG figure would eventually be removed.
41
42 df_2 <- df_1[1:350,] %>%
43   drop_na(XLAT, XLONG)
44
45 nrow(df_2)
46
47 eda300_df <- as.data.frame(df_2[1:300, ])
48 class(eda300_df)
49 View(eda300_df)
```

Slide 3: Filtering and indexing for 300 rows exploratory data analysis

2.1.4 Ordering the Latitude and Longitude

With the 300-row data frame eda300_df determined, it was decided that the reference columns XLAT and XLONG had to be arranged in order so as to enable interpolation techniques for filling NA to be appropriately applied, as shown in slide 4. To start with, the str() function was applied to determine the data types of the variables, which were all discovered to be the

“character” type. Hence, as.numeric was applied to the entire data frame to enable more flexibility and processing of the values, after which the order() function was applied to arrange both XLAT and XLONG in ascending order.

```
51 # In order to choose the technique for filling in the missing values of
52 # eda300_df, the latitude and longitude had to be arranged in order. This
53 # could only be done with the appropriate variable data type.
54 str(eda300_df)
55
56 eda300_df <- as.data.frame(sapply(eda300_df, as.numeric))
57 # changed values from character type to numeric data type
58
59 class(eda300_df)
60 str(eda300_df)
61
62 eda300_df <- eda300_df[order(eda300_df$XLAT, eda300_df$XLONG), ]
63 rownames(eda300_df) <- NULL
64 class(eda300_df)
65 View(eda300_df)
```

Slide 4: Ordering the data frame by XLAT and XLONG

2.1.5 Exploring the Unique Variables

At this juncture, in order to determine more clearly the type or method of cleaning to be done in each variable, it was decided to narrow down the number of variables to only the unique variables, that is, removing repeated variables so as to explore more easily each unique one visually. Also, the number of repetitions was calculated as this would later be required in determining the frequency of wind velocity throughout the month. Since there were 2 reference variables XLAT and XLONG, and 10 other unique variables that were repeated; it all summed up to 12 total unique variables. The data frame that resulted from these considerations was edavar12_df, as depicted in slide 5.

```

67 # It was observed that there were essentially 10 unique columns (column names
68 # with the exceptions of XLAT and XLONG - the reference variables) repeated
69 # 248 times for different time intervals
70
71 ncol(eda300_df)
72
73 # Since XLAT and XLONG were the reference variables, they were not repeated. The
74 # next 10 columns were however repeated. Calculations were done to determine the
75 # number of repetitions
76
77 rep_unique_var <- (ncol(eda300_df)-2)/10
78 rep_unique_var
79
80 # The 2 reference and 10 unique variables were explored to proceed on how to
81 # treat the missing variables for each field
82 edavar12_df <- eda300_df[,1:12]
83 edavar12_df
84 class(edavar12_df)
85 View(edavar12_df)

```

Console Terminal Background Jobs

R 4.2.3 · ~/

```

> ncol(eda300_df)
[1] 2482
> rep_unique_var <- (ncol(eda300_df)-2)/10
> rep_unique_var
[1] 248
> # The 2 reference and 10 unique variables were explored to proceed on how to
> # treat the missing variables for each field
> edavar12_df <- eda300_df[,1:12]
> edavar12_df

```

	XLAT	XLONG	TSK	PSFC	"U10"	"V10"	"Q2"	RAIN	RAINNC	SNOW	TSLB	SMOIS
1	48.871	-11.221	NA	101418	6.9	5.5	0.00602	0	0	0	273.2	1
2	48.884	-11.009	285.2	101426	6.8	5.3	NA	0	0	NA	273.2	1
3	48.896	-10.798	285.2	101433	6.6	5.0	0.00593	0	0	0	273.2	1
4	48.907	-10.587	285.1	101439	6.5	4.7	0.00587	0	0	0	NA	1
5	49.010	-11.240	285.2	101388	7.0	5.8	0.00603	0	0	0	273.2	1

Slide 5: Determining unique variables and calculating number of repetitions

2.1.6 Linear Interpolation for Missing Values

After examination of edavar12_df and its unique variables, linear interpolation was selected as the procedure by which the missing values would be filled. Linear interpolation, which can estimate missing values between 2 points on a straight line by assuming a constant rate of change between them (Blu et al., 1970), was adopted here for filling the NAs across all variables. With the interpolation function na.approx() adopted from the zoo library, the eda300_df data frame was mutated and all NAs replaced; as shown in slide 6.

```
88 # After examination, interpolation was chosen as the technique to be adopted
89 # for filling the NA's in the EDA 300 rows
90
91 # Missing values filled with linear interpolation
92 install.packages("zoo") #zoo package installed and imported
93 library(zoo)
94
95 eda300filled_df <- na.approx(eda300_df, rule = 2, na.rm = FALSE, maxgap = Inf)
96 sum(is.na(eda300filled_df))
97
```

Slide 6: Filling NAs with interpolation function na.approx()

2.2 Wind Velocity Time Series (Statistical) Analysis at Colwyn Bay Geographical Coordinates

In order to critically evaluate the feasibility of establishing a wind power station at Colwyn Bay with coordinates 53.394 latitude and -3.768 longitude, the wind velocity parameters of the WRF Model 2018 dataset had to be thoroughly analysed and forecasted both statistically and through machine learning. The knowledge that Colwyn Bay is a coastal town, which implied it would have relatively higher wind speeds, was not sufficient to determine embarking on a project of this scale and cost. More detailed analysis of the horizontal and vertical wind velocity parameters at 10 metres above sea level, U10 and V10, as well as their resultant velocity, had to be carried out.

2.2.1 Selecting Colwyn Bay Coordinates and Wind Parameters

To start off, data selection, cleaning and wrangling was done. The row containing the Colwyn Bay coordinates was selected using the `which()` function to find their indexes and filtered from the data frame `df_1` as `colwyn_data`. The `colwyn_data` U10 and V10 column names were then cleaned to remove unwanted inverted commas using `gsub()` function as shown in slide 7.


```

110 #####
111 ### LOCATION COORDIATES AND CHOSEN TIME SERIES VARIABLE ###
112
113 # A location in the UK was selected to determine its suitability for
114 # for a wind power station or turbine set up.
115
116 # Hence coordinates XLAT = 53.394 and XLONG = -3.768 were selected which
117 # are coordinates for a location near Colwyn Bay, North Wales.
118
119 rowindex <- which(apply(df_1, 1,
120                        function(x) x["XLAT"] == "53.394" &
121                                x["XLONG"] == "-3.768"))
122 rowindex
123
124 colwyn_data <- df_1[2798,]
125 colwyn_data
126
127 colnames(colwyn_data) <- gsub("\\", "", colnames(colwyn_data))
128 colwyn_data
129 length(colnames(colwyn_data))

```

Slide 7: Selecting Colwyn Bay coordinates row and cleaning column names U10 and V10

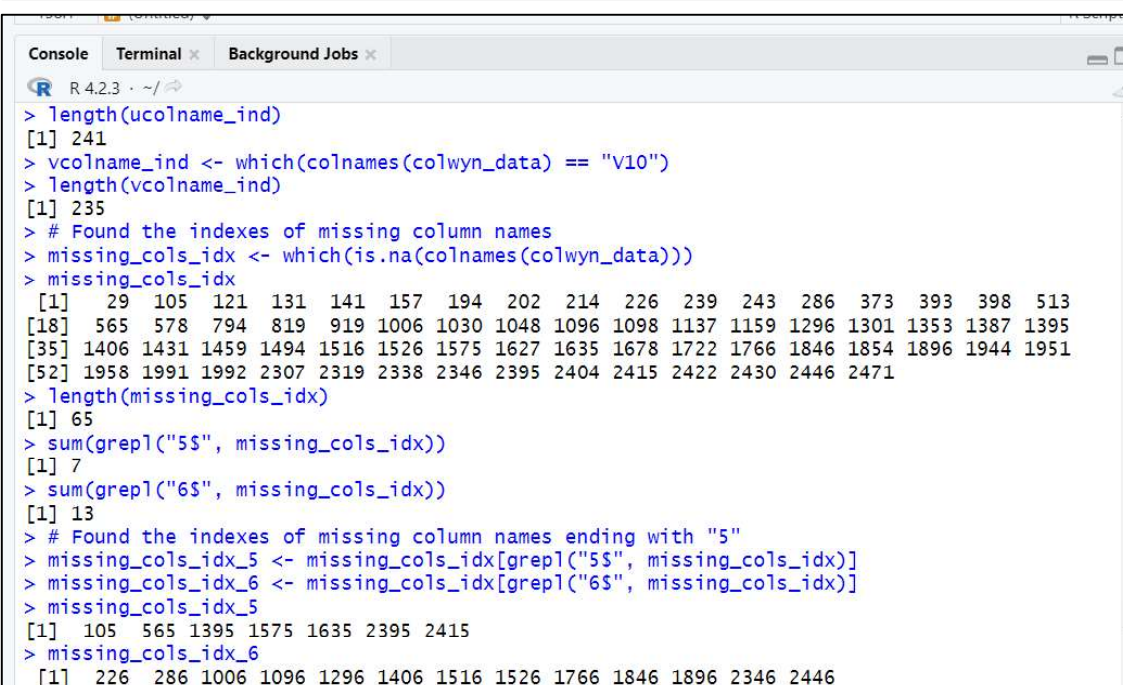
With the aim of creating a time series composed of the wind velocity and datetime, the indexes of the X and Y components of the wind velocity of the colwyn_data data frame had to be determined. This was done by using the which() function to find column names “U10” and “V10”. It was however discovered at this point that the number of U10 columns and V10 columns were 241 and 235 respectively, which each fall short of the expected 248 columns. It was therefore deduced that there were actually missing or incorrect column names. Interestingly, the number of missing column names were found to be up to 65, but this accounted for all missing column names, while we were only looking for U10 and V10 missing names.

Faced with this challenge, it was visually deduced that the first U10 and V10 names occurred at indexes 5 and 6 respectively of colwyn_data data frame; and since it was already known that the column names repeat at intervals of 10, the next U10 and V10 were found at indexes 15 and 16 respectively, and so on. Hence, indexes of U10 and V10 were numbers ending in 5 and 6 respectively. Code was thus written to locate all missing U10 and V10 column names based on their indexes and they were filled accordingly, as demonstrated in slide 8.


```

131 ucolname_ind <- which(colnames(colwyn_data) == "U10")
132 ucolname_ind
133 length(ucolname_ind)
134
135 vcolname_ind <- which(colnames(colwyn_data) == "V10")
136 length(vcolname_ind)
137
138 # Found the indexes of missing column names
139 missing_cols_idx <- which(is.na(colnames(colwyn_data)))
140 missing_cols_idx
141 length(missing_cols_idx)
142
143 sum(grepl("5$", missing_cols_idx))
144 sum(grepl("6$", missing_cols_idx))
145
146 # Found the indexes of missing column names ending with "5"
147 missing_cols_idx_5 <- missing_cols_idx[grepl("5$", missing_cols_idx)]
148 missing_cols_idx_6 <- missing_cols_idx[grepl("6$", missing_cols_idx)]
149 missing_cols_idx_5
150 missing_cols_idx_6
151
152 colnames(colwyn_data)[missing_cols_idx_5] <- "U10"
153 colnames(colwyn_data)[missing_cols_idx_6] <- "V10"
154 colnames(colwyn_data)
155
156 length(which(is.na(colnames(colwyn_data))))

```



```

R 4.2.3 ~ /
> length(ucolname_ind)
[1] 241
> vcolname_ind <- which(colnames(colwyn_data) == "V10")
> length(vcolname_ind)
[1] 235
> # Found the indexes of missing column names
> missing_cols_idx <- which(is.na(colnames(colwyn_data)))
> missing_cols_idx
[1] 29 105 121 131 141 157 194 202 214 226 239 243 286 373 393 398 513
[18] 565 578 794 819 919 1006 1030 1048 1096 1098 1137 1159 1296 1301 1353 1387 1395
[35] 1406 1431 1459 1494 1516 1526 1575 1627 1635 1678 1722 1766 1846 1854 1896 1944 1951
[52] 1958 1991 1992 2307 2319 2338 2346 2395 2404 2415 2422 2430 2446 2471
> length(missing_cols_idx)
[1] 65
> sum(grepl("5$", missing_cols_idx))
[1] 7
> sum(grepl("6$", missing_cols_idx))
[1] 13
> # Found the indexes of missing column names ending with "5"
> missing_cols_idx_5 <- missing_cols_idx[grepl("5$", missing_cols_idx)]
> missing_cols_idx_6 <- missing_cols_idx[grepl("6$", missing_cols_idx)]
> missing_cols_idx_5
[1] 105 565 1395 1575 1635 2395 2415
> missing_cols_idx_6
[1] 226 286 1006 1096 1296 1406 1516 1526 1766 1846 1896 2346 2446

```

Slide 8: Finding and filling missing U10 and V10 column names

The code in slide 8 confirmed that the sum of the missing column names with an index ending with 5 was 7 and that ending with 6 was 13. These were respectively filled with “U10” and “V10”.

Thereafter, the `colwyn_data` data frame was filtered with the help of the `grep()` function which searched for U10 in the column names of the data frame in order to form a list of values of U10. The same was done for V10; and these gave us the values of the X and Y components of the Colwyn Bay wind velocity for that period of time, `output_listu` and `output_listv`—as seen in slide 9.

```
161 my_values_u <- colwyn_data[, grep("U10", names(colwyn_data))]
162 my_values_u
163 length(my_values_u)
164 typeof(my_values_u)
165
166 output_listu <- list() # created an empty list to store the outputs
167
168 for (i in my_values_u) {
169   output_listu <- c(output_listu, list(i))
170 }
171 output_listu <- unlist(output_listu)
172 output_listu
173 typeof(output_listu)
174
175 my_values_v <- colwyn_data[, grep("V10", names(colwyn_data))]
176 my_values_v
177 length(my_values_v)
178 output_listv <- list() # created an empty list to store the outputs
179
180 for (i in my_values_v) {
181   output_listv <- c(output_listv, list(i))
182 }
183 output_listv <- unlist(output_listv)
184 output_listv
185 typeof(output_listv)
186
```

Slide 9: : Forming a list of values of U10 and V10

With these components, a `vel_df` data frame was created and the field values converted to the numeric data type so missing values could be filled via interpolation. After this was done, the resultant velocity variable was created by calculating the resultant velocity from the X and Y components. The calculation is as shown in slide 10.

```

189 vel_df <- data.frame(output_listu, output_listv) # formed the velocity df
190 vel_df
191
192 vel_df$U10_values <- as.numeric(vel_df$output_listu)
193 vel_df$output_listu <- NULL
194 vel_df$V10_values <- as.numeric(vel_df$output_listv)
195 vel_df$output_listv <- NULL
196 vel_df
197
198 # Used interpolation to fill NA's
199 vel_df <- as.data.frame(na.approx(vel_df, rule = 3,
200                                na.rm = FALSE, maxgap = Inf))
201 vel_df
202 sum(is.na(vel_df))
203 class(vel_df)
204
205 # Resultant velocity calculated
206 vel_df$resultant_vel <- sqrt(vel_df$U10_values^2 + vel_df$V10_values^2)
207 vel_df$resultant_vel
208 vel_df
209
210 resultvel <- vel_df[, -1:-2]
211 resultvel

```

Slide 10: Data frame of values of U10 and V10; and resultant velocity calculated

2.2.2 The Resultant Velocity/ Datetime Graph

The goal at this point was to create a data frame of the resultant velocity and the given datetime covering May 2018. From the original data frame df, it was observed that the date and time started at 01.05.2018.00.00, that is, May 1st 2018 12.00am and measurements were taken every 3 hours from then thus giving a frequency of 248 – which is in tandem with earlier calculations. Since we had the start and end times as well as the interval of 3 hours, the seq() function was adopted to create a sequence of datetime values, “datetimes”. A data frame rvel_df was subsequently formed using the datetime values and the list of resultant velocities; and plotted as shown in slide 11 and figure 1 below.

```

213 # Created the sequence of dates and times with a frequency of 3 hours
214 datetimes <- seq(from=as.POSIXct("2018-05-01 00:00"),
215                 to=as.POSIXct("2018-05-31 21:00"),
216                 by="3 hours")
217 datetimes
218 typeof(datetimes)
219 class(datetimes)
220
221 rvel_df <- data.frame(datetimes, resultvel)
222 rvel_df
223 # Plotted the resultant velocity against time
224 library(ggplot2)
225 ggplot(rvel_df, aes(x = datetimes, y = resultvel)) + geom_line() +
226   labs(title = "Colwyn Bay May 2018 Wind Velocity Graph",
227        x = "DateTime", y = "Resultant Velocity")

```

Slide 11: Date/time sequence and data frame rvel_df creation and plotting

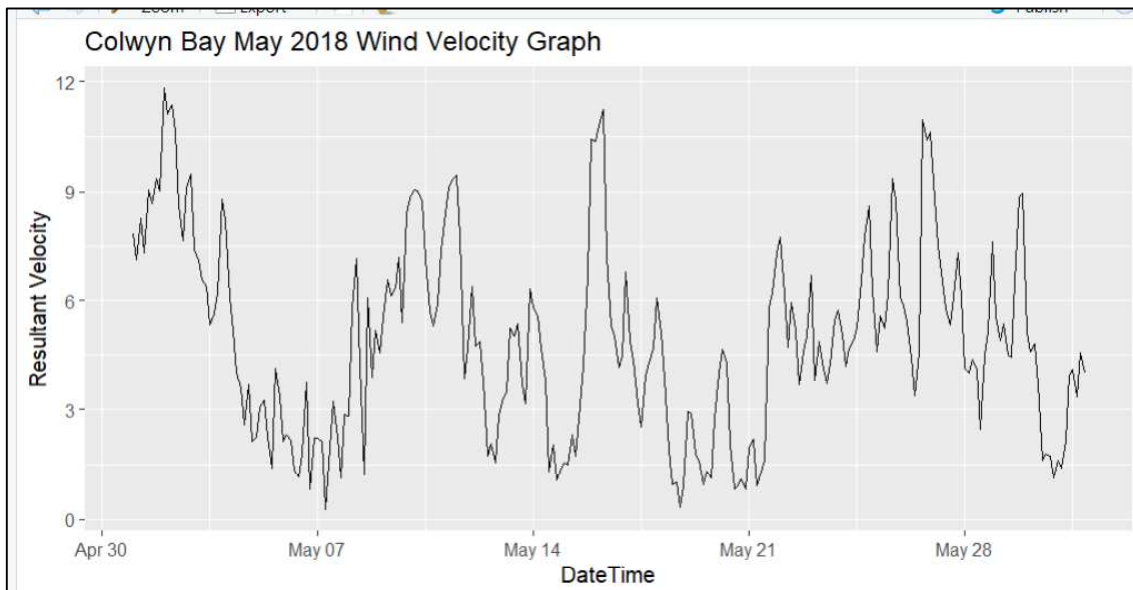


Fig 1.: Colwyn Wind Velocity Graph

2.2.3 Time Series Analysis

At this point, the forecast library was imported for the analysis of the time-velocity data frame rvel_df. Zoo() function was used to create the time series vel_ts ordered by the date-time sequence; and the time series was thereafter aggregated using the aggregate() function into the mean resultant velocity per day. The aggregated time series ag.vel_ts was checked for outliers after which, it was plotted. These are demonstrated in slide 12 and figure 2 below.

```

229 # Creating a time series, libraries are imported
230 install.packages("forecast")
231 library(forecast)
232
233 # Used zoo() to create the time series
234 vel_ts <- zoo(rvel_df$resultvel,
235             order.by = rvel_df$datetimes)
236 vel_ts
237 head(vel_ts)
238 length(vel_ts)
239
240 # Aggregated the time series
241 ag.vel_ts <- aggregate(vel_ts, as.Date, mean)
242 ag.vel_ts
243 daily <- index(ag.vel_ts)
244 daily
245 length(ag.vel_ts)
246
247 # Checked for outliers in the ts
248 ag.vel_ts <- tsclean(ag.vel_ts)
249 ag.vel_ts
250 summary(ag.vel_ts)
251
252 # Plotted the time series
253 ggplot(ag.vel_ts, aes(x = daily, y = ag.vel_ts)) + geom_line() +
254   labs(title = "Aggregated Colwyn Wind Velocity",
255        x = "Date", y = "Mean Resultant Velocity")

```

Slide 12: Forming time series zoo() object, aggregating and plotting

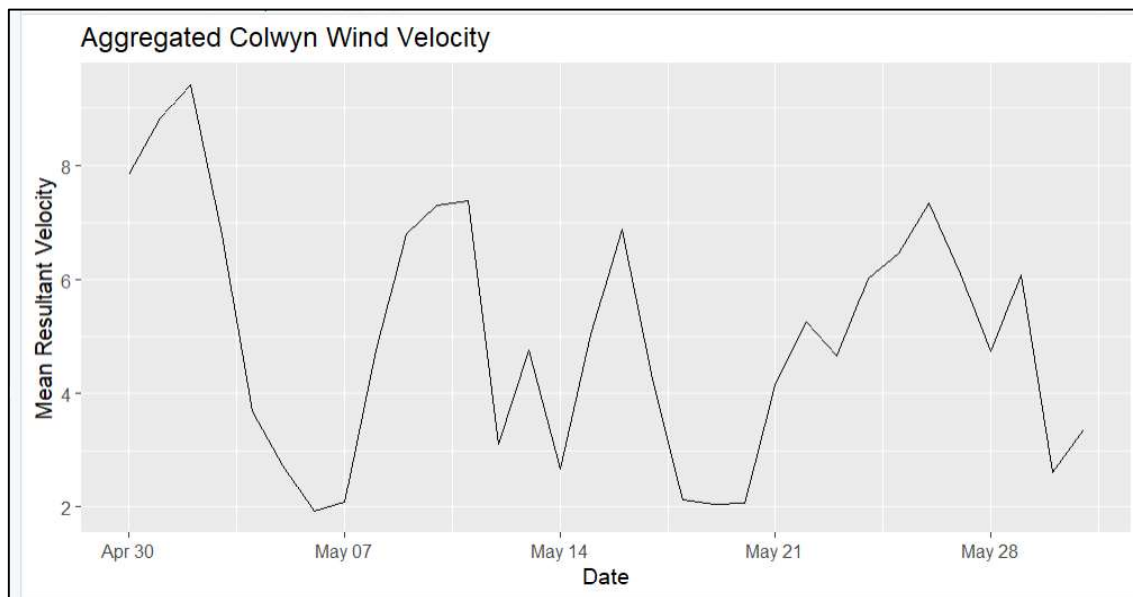


Fig 2.: Aggregated Colwyn Wind Velocity Graph

2.2.4 Autocorrelation and Partial Autocorrelation

Other time series analysis techniques were carried out on the time series data `vel_ts`, as depicted in slide 13, in order to examine and give more information on the time series. `Tsdisplay()` function was used to plot the time series as well as display its autocorrelation and partial autocorrelation plots at different lags.

For the Colwyn wind velocity time series, the ACF (autocorrelation function) plot showed a statistically significant and strongly positive correlation at the earlier lags from 1 to 4; the correlation coefficient continued to reduce and became statistically insignificant from lag 11. Statistical significance was determined by the blue dashes in an ACF plot which represented the 95% confidence interval beyond which the correlation was significant, as shown in figure 3. There was also a fairly strong negative correlation at lag 28 which was statistically significant. The PACF (partial autocorrelation function) only showed strong, positive, statistically significant correlation at the first 3-hour lag as also shown in figure 3.

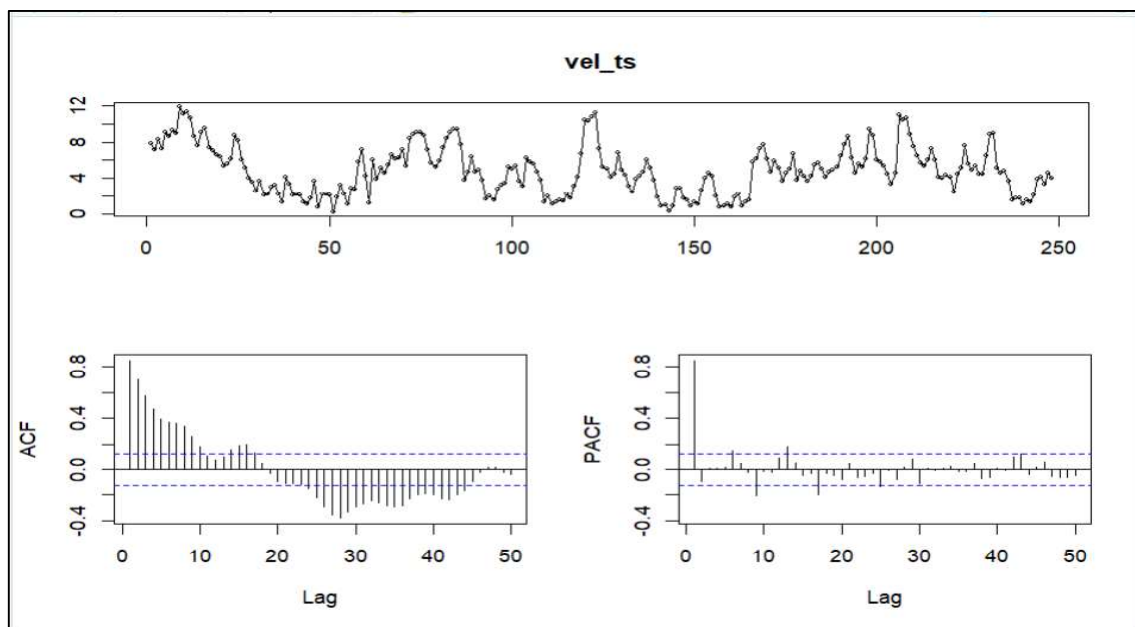


Fig. 3: Autocorrelation and partial autocorrelation time series analysis plot.

2.2.5 Augmented Dickey-Fuller Test

The Augmented Dickey-Fuller Test was performed next on the time series with the function `adf.test()`, which was used to determine stationarity or whether the autocorrelation and other stats would change with time. The p-value of an ADF test is one of the parameters used in determining stationarity as the lower the p-value, the more evidence against the null hypothesis (of non-stationarity) and a higher likelihood the time series is stationary (Silva et al., 2021). With a p-value of 0.08821 as shown in slide 14, which is above the significance value of 0.05, it was determined that the time series failed to reject the null hypothesis and thus was non-stationary. Stationarity is important and often desirable as it aids in forecasting and the creation of models given statistical properties are stationary. The result of lag order 6 in this analysis was important as it informed the suitable number of lags required in clarifying and obtaining the best results from the time series.

2.2.6 ARIMA

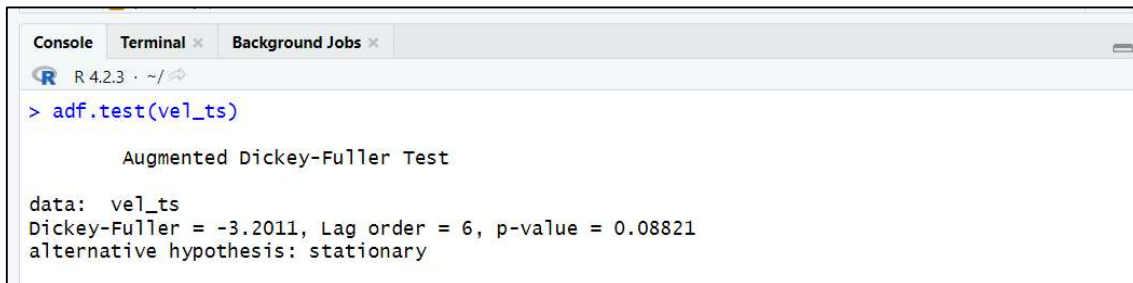
ARIMA (Autoregressive Integrated Moving Average) model was used at this point with best model determined at ARIMA(1,0,1) with non-zero mean of 4.9397, as shown in slide 14. This indicated a lagged value order of 1, zero differencing and a lagged error order of 1. The differencing (integrated part) of 0 indicated no differencing was required to make the time series stationary. As depicted in slide 15, `auto.arima()` function was used with argument “trace” set to True to show the steps in choosing the best model. The AR value was 0.8208 while the MA value was 0.1137, which indicated the level of influence of the lagged values and lagged errors respectively on the time series. These results are as displayed in slide 16.

```

259 # Other time series analysis
260
261 # Displayed the degree of correlation at different lags
262 tsdisplay(vel_ts, lag.max = 50)
263
264 library(tseries)
265 adf.test(vel_ts)
266 # Failed to reject the null hypothesis since p-value > 0.05 hence, time series
267 # data is non-stationary
268
269 vel_arima <- auto.arima(vel_ts, trace=T, stepwise = F, approximation = F)
270 vel_arima
271 # Setting trace as True in order to trace the model selection process,
272 # an ARIMA(1,0,1) with non-zero mean was arrived at, AR term of 0.8219,
273 # MA term of 0.1114 and differencing of 0.
274

```

Slide 13: tsdisplay(), ADF-testing and ARIMA



```

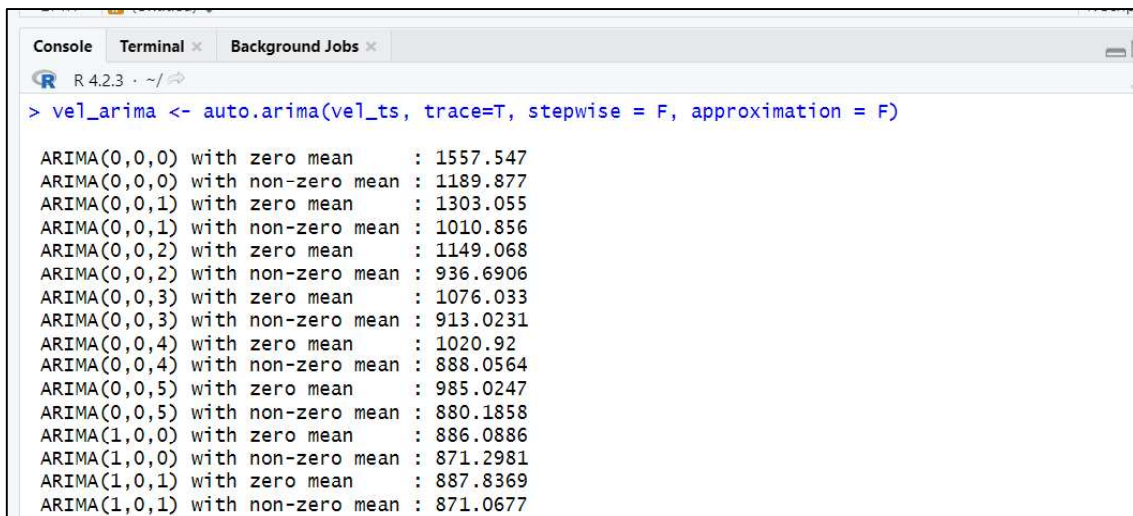
R 4.2.3 ~ /
> adf.test(vel_ts)

Augmented Dickey-Fuller Test

data: vel_ts
Dickey-Fuller = -3.2011, Lag order = 6, p-value = 0.08821
alternative hypothesis: stationary

```

Slide 14: ADF-testing results



```

R 4.2.3 ~ /
> vel_arima <- auto.arima(vel_ts, trace=T, stepwise = F, approximation = F)

ARIMA(0,0,0) with zero mean      : 1557.547
ARIMA(0,0,0) with non-zero mean : 1189.877
ARIMA(0,0,1) with zero mean      : 1303.055
ARIMA(0,0,1) with non-zero mean : 1010.856
ARIMA(0,0,2) with zero mean      : 1149.068
ARIMA(0,0,2) with non-zero mean : 936.6906
ARIMA(0,0,3) with zero mean      : 1076.033
ARIMA(0,0,3) with non-zero mean : 913.0231
ARIMA(0,0,4) with zero mean      : 1020.92
ARIMA(0,0,4) with non-zero mean : 888.0564
ARIMA(0,0,5) with zero mean      : 985.0247
ARIMA(0,0,5) with non-zero mean : 880.1858
ARIMA(1,0,0) with zero mean      : 886.0886
ARIMA(1,0,0) with non-zero mean : 871.2981
ARIMA(1,0,1) with zero mean      : 887.8369
ARIMA(1,0,1) with non-zero mean : 871.0677

```

Slide 15: Auto.arima processing


```
Console Terminal x Background Jobs x
R 4.2.3 · ~/
Best model: ARIMA(1,0,1) with non-zero mean
> vel_arima
Series: vel_ts
ARIMA(1,0,1) with non-zero mean

Coefficients:
      ar1      ma1      mean
    0.8208  0.1137  4.9397
s.e.  0.0420  0.0742  0.5326

sigma^2 = 1.913: log likelihood = -431.45
AIC=870.9   AICc=871.07   BIC=884.96
```

Slide 16: ARIMA results

2.3 Wind Velocity Machine Learning Analysis

Just like statistical analysis tools like ARIMA could be used for forecasting, machine learning could also be adopted for predicting the Colwyn Bay wind velocity parameter. The critical packages of “caret” and “future” were installed and imported to ensure tuned machine learning models, pre-processing, easier model comparison and accuracy. Caret (Classification and Regression Training) enables random forests and support vector machines algorithms, amongst others.

The resultant velocity was fortified from a zoo time series object `vel_ts` to a data frame `rvel_df`. `Rvel_df` was then trained on 80% training set and 20% test set using the `createDataPartition()` function. With the training and test sets prepared, the machine leaning models of linear regression, support vector machine regression and random forest were explored. It must be mentioned at this point that each model had a cross-validation `trainControl()` function with method argument specified as “timeslice”; this was selected due to the nature of the data as being a time series.

2.3.1 Linear Regression

The “readr” package was imported to enable the linear regression method “lm”. After the time series cross validation method was specified, the `train()` function was used on the wind velocity and datetime variables of the training set to create the linear regression model “lm_model”. The `predict()` function was then used on the linear regression model and test set to generate the predicted values. To conclude the linear regression processes, the root mean square error was calculated taking into consideration the test wind velocity values together with the predicted values, as shown in slide 17.

```

301 # The data frame was split into training and testing sets
302 set.seed(321)
303 trainIndex <- createDataPartition(rvel_df$wind_velocity, p = 0.8, list = FALSE)
304 train <- rvel_df[trainIndex, ]
305 nrow(train)
306 test <- rvel_df[-trainIndex, ]
307 nrow(test)
308
309 # Linear regression model
310 install.packages("readr")
311 library(readr)
312
313 # Specified the time series cross-validation method
314 lm_tc <- trainControl(method = "timeslice", initialwindow = 24,
315                       horizon = 1, fixedwindow = TRUE)
316
317 # specified the model and its parameters
318 lm_model <- train(wind_velocity ~ datetime, data = train,
319                  method = "lm", trControl = lm_tc, preProc = NULL)
320
321 # Used the trained model to make predictions on the test set
322 lm_predictions <- predict(lm_model, newdata = test)
323 lm_predictions
324 length(lm_predictions)
325
326 # Calculated the mean squared error and root mean squared error
327 lm_mse <- mean((test$wind_velocity - lm_predictions)^2)
328 lm_rmse <- sqrt(lm_mse)
329 lm_rmse

```

Slide 17: Predictions and RMSE with linear regression

2.3.2 Support Vector Machine Regression and Random Forest

Going forward with support vector machine regression SVMR and random forest, similar models were created with the train functions having method arguments of “svmRadial” and “rf” respectively. Additionally, the 80% training set was used in both SVMR and random forest cases and their root mean square errors RMSEs were also calculated, as shown in slides 18 and 19.

The svmRadial implied that the kernel used in this case was a radial basis function RBF kernel, which is the more commonly used kernel for SVMR because it can handle complex and nonlinear data well. The SVMR was also tuned to have a sigma value of 10.33358 and C of 2

after testing with the `tuneLength` argument; `sigma` and `C` determine the complexity, error optimization and fitting of a kernel and model.

The random forest model used an `mtry` of 2 to get its most optimized model; this value was also gotten after tuning. `Mtry` is an attribute that controls how many features are randomly selected in each split of the decision tree (Ellis, 2022).

```
332 # Support Vector Machine Regression SVMR model
333 install.packages("e1071")
334 library(e1071)
335
336 # Specify the time series cross-validation method
337 svm_tc <- trainControl(method = "timeslice", initialWindow = 24,
338                       horizon = 1, fixedwindow = TRUE)
339
340 # Specify the model and its parameters
341 svm_model <- train(wind_velocity ~ datetime, data = train,
342                  method = "svmRadial", trControl = svm_tc,
343                  preProc = NULL, tuneGrid = expand.grid(sigma = 10.33358, C = 2))
344
345 svm_model
346
347 # Use the trained model to make predictions on the test set
348 svm_predictions <- predict(svm_model, newdata = test)
349 svm_predictions
350
351 # Calculate the mean squared error and root mean squared error
352 svm_mse <- mean((svm_predictions - test$wind_velocity)^2)
353 svm_mse
354 svm_rmse <- sqrt(svm_mse)
355 svm_rmse
```

Slide 18: Support vector machines model

```
358 # Random Forest model
359 install.packages("randomForest")
360 library(randomForest)
361
362 # specify the time series cross-validation method
363 rf_tc <- trainControl(method = "timeslice", initialWindow = 24,
364                      horizon = 1, fixedwindow = TRUE)
365
366 # specify the model and its parameters
367 set.seed(152)
368 rf_model <- train(wind_velocity ~ datetime, data = train,
369                 method = "rf", trControl = rf_tc, preProc = NULL, tuneLength = 10)
370 rf_model
371
372 # use the trained model to make predictions on the test set
373 rf_predictions <- predict(rf_model, newdata = test)
374 rf_predictions
375
376 # calculate the error metrics
377 rf_mse <- mean((rf_predictions - test$wind_velocity)^2)
378 rf_mse
379 rf_rmse <- sqrt(rf_mse)
380 rf_rmse
```

Slide 19: Random forests model

2.3.3 Root Mean Square Errors and Plots

Here, all the models' RMSEs were collated in a table, inclusive of the ARIMA model; and it was discovered that the random forest model displayed the lowest error score, as shown in slide 20. Further evidence of this was as demonstrated in the plots of the actual and predicted values as generated from the machine learning models discussed (figures 4,5 and 6 below).

```
383 # Create a table of model performance metrics
384 models <- c("ARIMA", "Linear Reg", "SVMR", "Random Forest")
385 total_rmse <- c(arima_rmse, lm_rmse, svm_rmse, rf_rmse)
386 results <- data.frame(Model = models, RMSE = total_rmse)
387
388 # Print the table of model performance metrics
389 print(results)
```

	Model	RMSE
1	ARIMA	1.374570
2	Linear Reg	2.360658
3	SVMR	1.548902
4	Random Forest	1.115509

Slide 20: Model RMSE values

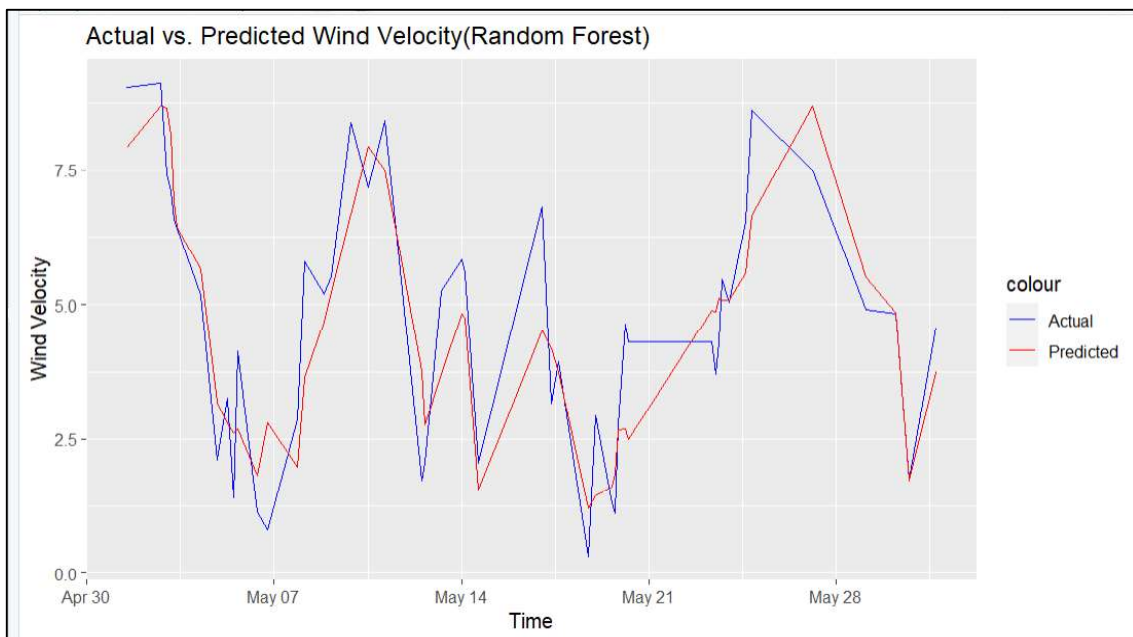


Fig 4: Actual vs. predicted wind velocity plot on random forest model

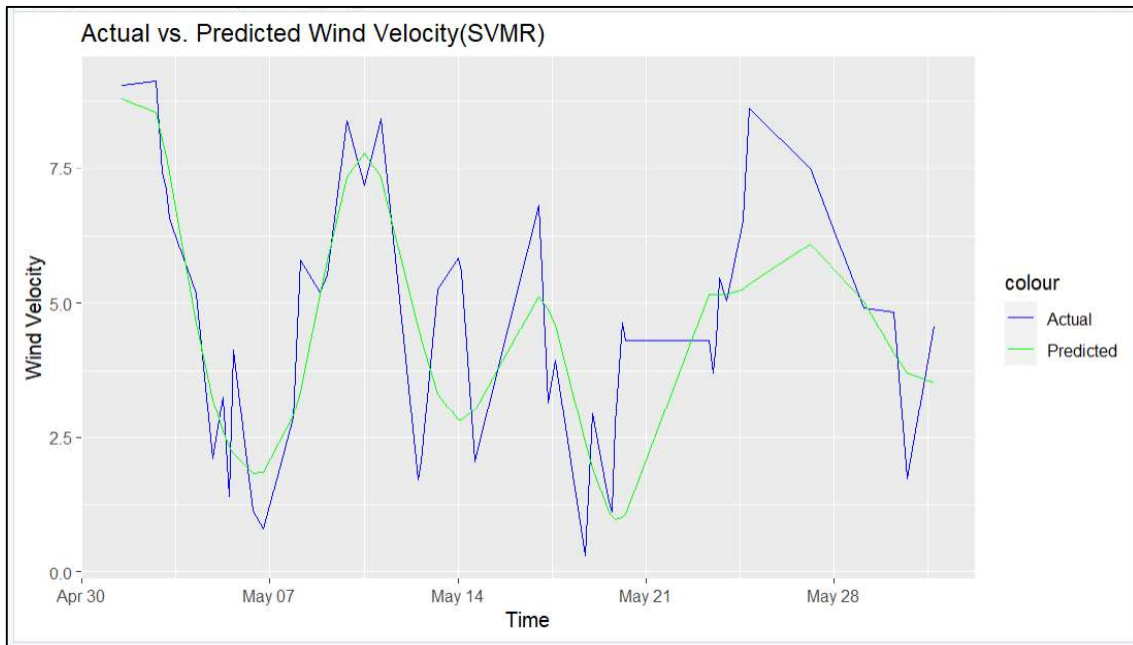


Fig 5: Actual vs. predicted wind velocity plot on support vector machine regression model

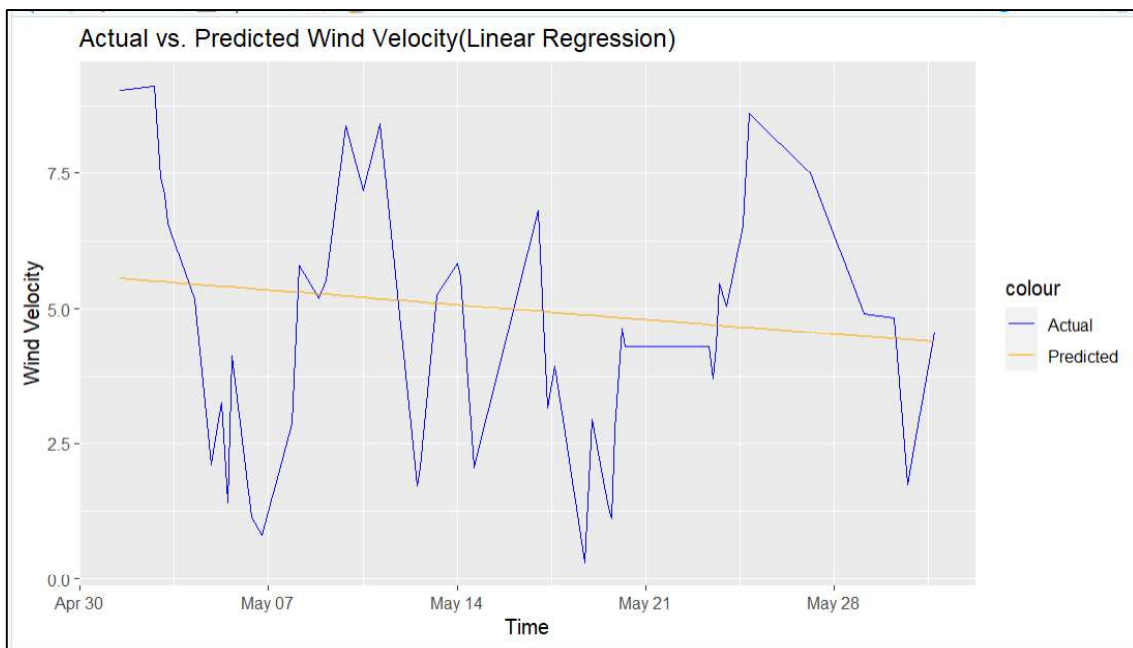


Fig 6: Actual vs. predicted wind velocity plot on linear regression model

2.4 Correlation and Testing

Correlation was used at this juncture to determine if there was a relationship between the surface temperature parameter TSK and the resultant wind velocity. This information was important in determining if varying temperatures across the month or over a year could significantly impact the wind velocity or if temperature changes could be used as a yardstick in predicting wind velocity changes.

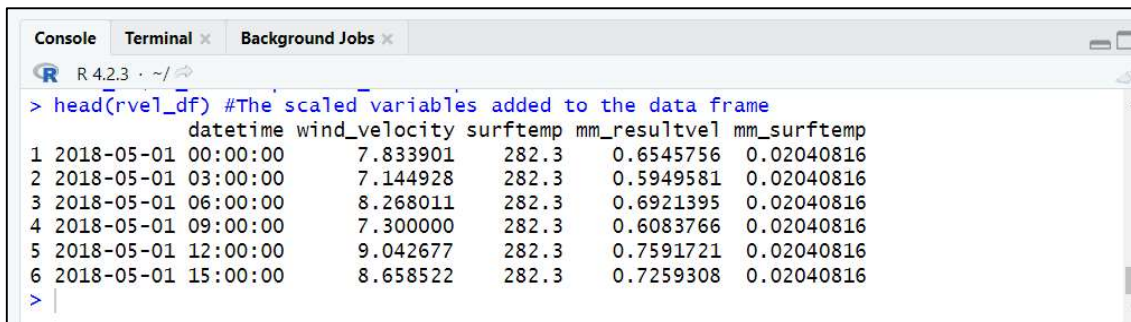
To start with, the surface temperature variable TSK from the Colwyn Bay data frame, `colwyn_data`, was isolated and its values along the 3-hour intervals across the month extracted, similar to how the X and Y components U10 and V10 of wind velocity were. Linear interpolation was used to fill in its missing values to give `output_tsk`, after which it was added to the resultant velocity data frame `rvel_df` as `rvel_df$surftemp`.

Spearman's rank correlation was chosen as the correlation method as it was clear from the onset, from visual inspection, that the relationship between the variables was unlikely to be linear. The `cor()` function was however only applied after the variable underwent min-max scaling, in order to reduce the impact of possible outliers and ensure greater correlation accuracy. This is as demonstrated in slide 21.


```

494 # Linear interpolation was used for missing values
495 output_tsk <- na.approx(output_tsk, rule = 2, na.rm = FALSE, maxgap = Inf)
496 output_tsk
497
498 # Invoked the resultant velocity data frame rvel_df and added the
499 # surface temperature variable to it
500 rvel_df$surftemp <- output_tsk
501 rvel_df
502 rvel_df %>% View()
503
504 # Used min-max scaling on velocity and temperature variables
505 mm1_vel <- (rvel_df$wind_velocity - min(rvel_df$wind_velocity))
506 mm2_vel <- (max(rvel_df$wind_velocity) - min(rvel_df$wind_velocity))
507 mm_resultvel <- mm1_vel/mm2_vel
508 head(mm_resultvel)
509
510 mm1_tsk <- (rvel_df$surftemp - min(rvel_df$surftemp))
511 mm2_tsk <- (max(rvel_df$surftemp) - min(rvel_df$surftemp))
512 mm_surftemp <- mm1_tsk/mm2_tsk
513 head(mm_surftemp)
514
515 rvel_df$mm_resultvel <- mm_resultvel
516 rvel_df$mm_surftemp <- mm_surftemp
517 head(rvel_df) #The scaled variables added to the data frame

```



```

R 4.2.3 ~ /
> head(rvel_df) #The scaled variables added to the data frame
  datetime wind_velocity surftemp mm_resultvel mm_surftemp
1 2018-05-01 00:00:00      7.833901    282.3      0.6545756 0.02040816
2 2018-05-01 03:00:00      7.144928    282.3      0.5949581 0.02040816
3 2018-05-01 06:00:00      8.268011    282.3      0.6921395 0.02040816
4 2018-05-01 09:00:00      7.300000    282.3      0.6083766 0.02040816
5 2018-05-01 12:00:00      9.042677    282.3      0.7591721 0.02040816
6 2018-05-01 15:00:00      8.658522    282.3      0.7259308 0.02040816
>

```

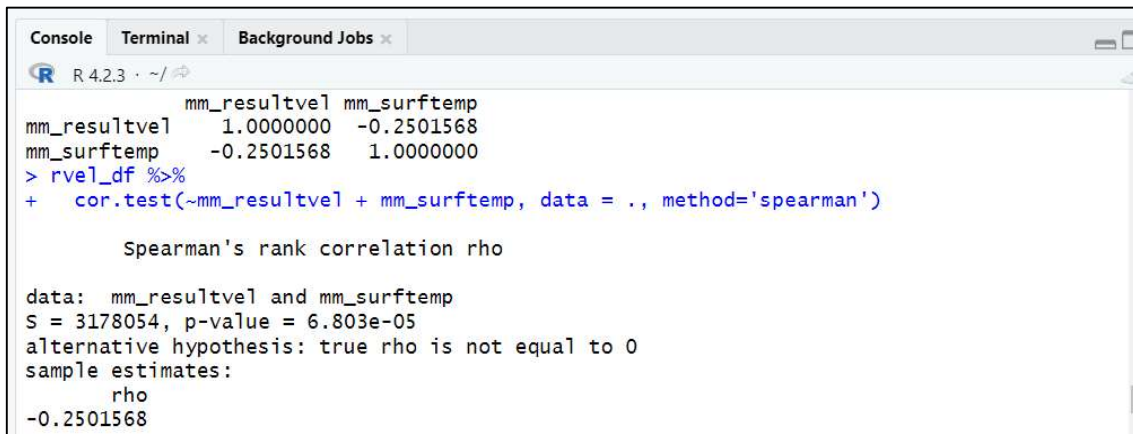
Slide 21: Min-max scaling of variables and adding scaled variable to rvel_df data frame

The results of the Spearman's rank correlation showed that there was a rather weak and inverse relationship between wind velocity and temperature with a correlation of -0.2501568. The negative correlation implied that wind speeds would tend to increase with a decrease in surface temperature but the relationship was so weak that it could be discounted for further analysis to be conducted.

However, with further testing done with the `cor.test()` function as shown in slide 22, a p-value equalling 6.803e-05 was generated—which was lower than the 0.05 significance value—thus rejecting the null hypothesis that the correlation coefficient is zero. It therefore indicated that

the negative correlation of -0.2501568 could not be ignored, and that there was a consistent, albeit less impactful, relationship between the 2 variables. The large test statistic value S also supported a relationship between the resultant wind velocity and surface temperature.

```
519 # Spearman's rank correlation was used for wind velocity and surface temperature
520 # variables correlation test
521 rvel_df %>%
522   select(mm_resultvel, mm_surftemp) %>%
523   cor(method='spearman')
524
525 rvel_df %>%
526   cor.test(~mm_resultvel + mm_surftemp, data = ., method='spearman')
```



```
R 4.2.3 ~/>
      mm_resultvel mm_surftemp
mm_resultvel  1.0000000 -0.2501568
mm_surftemp  -0.2501568  1.0000000
> rvel_df %>%
+   cor.test(~mm_resultvel + mm_surftemp, data = ., method='spearman')

      Spearman's rank correlation rho

data:  mm_resultvel and mm_surftemp
S = 3178054, p-value = 6.803e-05
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
-0.2501568
```

Slide 22: Spearman's rank correlation, test and results

A scatterplot of the 2 continuous, scaled variables demonstrating a regression line with 95% confidence interval and the negative correlation coefficient of -0.25016 was thereafter plotted. The graph further depicted the weak, inverse relationship between the velocity and temperature variables, as plotted in slide 23 and as displayed in figure 7.

```
557 # Plotting the scaled velocity vs surface temperature for correlation testing
558
559 ggplot(rvel_df, aes(x = mm_resultvel, y = mm_surftemp)) +
560   geom_point(color = "lightgreen") +
561   geom_smooth(method = "lm", color = "red", se = TRUE) +
562   labs(x = "Resultant Velocity(Scaled)", y = "Surface Temperature(Scaled)") +
563   ggtitle(paste("Velocity - Temperature Correlation:", round(cor(rvel_df$mm_resultvel,
564   rvel_df$mm_surftemp, method = "spearman"), 5)))
565
```

Slide 23: Plotting scaled resultant velocity vs scaled surface temperature

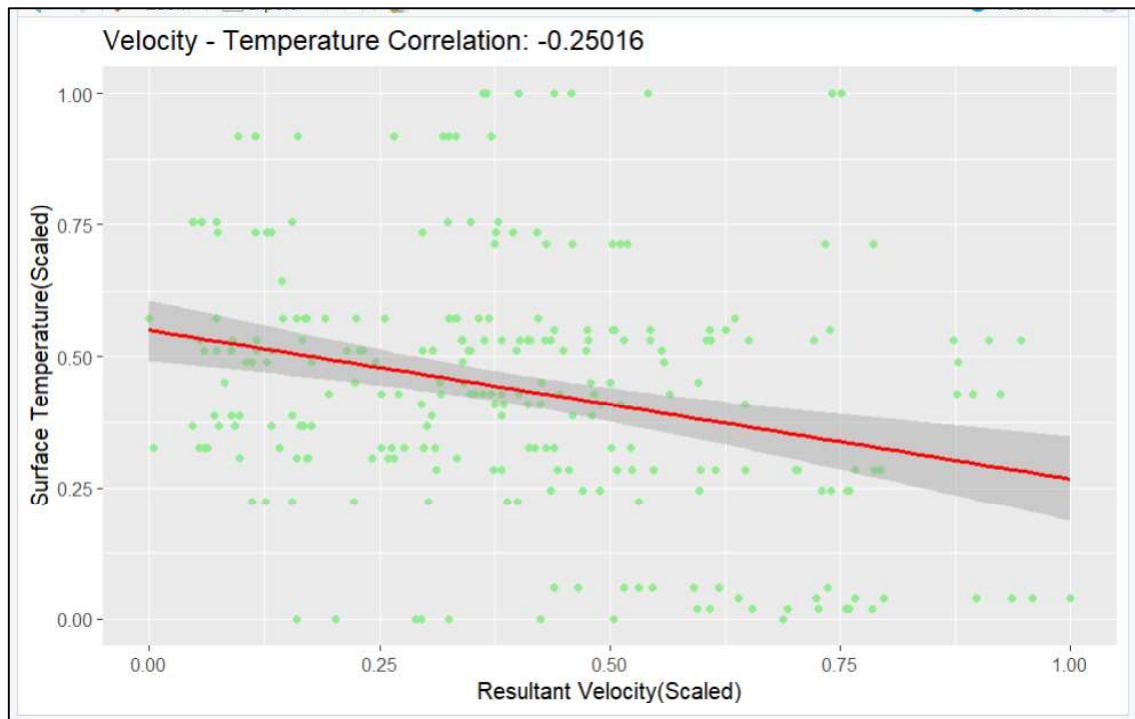


Fig 7: Surface Temperature(Scaled) vs. Resultant Velocity(Scaled) plot

2.5 Discussion

The May 2018 WRF time series data underwent a number of phases from data pre-processing, coordinates and parameter filtering, time series statistical analysis and forecasting, machine learning analyses, comparison and prediction to parameters correlation and testing. These processes were all in a bid to answer the problem statement on the suitability of Colwyn Bay as a site for a wind power station set up; as well as finding the best tools along the way to accurately answer this problem.

2.5.1 Exploratory Data Analysis

A sample of the data(300 rows) was cleaned and analysed to give an overview of the behaviour of the atmospheric parameters over time and through different geographical locations—using `eda300filled_df` and `edavar12filled_df` data frames. It was observed that the surface temperature TSK and pressure PSFC continued to dip as the latitude XLAT increased towards the North Pole. The humidity Q2 appeared more impacted by a move east longitudinally as it slightly dipped as the longitudinal lines moved east from 11°W to 10°W; it was barely impacted in the latitude range of this sample which spanned from latitude 48.871°N to 59.611°N. There was virtually no impact on convective rain RAINC and snow SNOW as they remained at 0 in this sample, while there was only a slight increase in non-convective rain RAINNC as latitude increased towards the north—this increase was however not stable. Soil temperature TSLB and soil moisture SMOIS remained constant at 273.2K and 1m³/m³ respectively in this sample. There were no discernible trends of change for the wind parameters U10 and V10 as these would require further analysis.

Summarily, these were the parameter analysis considerations for 1st May 2018 at 12.00am.

Further summary statistics at this datetime were applied to discover the mean, standard deviation, range, skew, etc. of the sample using data frame edavar12filled_df, as shown in slide

24.

```
541 # Using summary() and describe() for EDA of the unique variables along the column
542
543 install.packages("psych") #imported psych package in order to use describe()
544 library(psych)
545 edavar12_df
546 edavar12filled_df <- eda300filled_df[,1:12]
547 edavar12filled_df
548 class(edavar12filled_df)
549
550 summary(edavar12filled_df[, 3:12])
551 describe(edavar12filled_df[, 3:12])
```

```
R 4.2.3 ~ /
> summary(edavar12filled_df[, 3:12])
```

TSK	PSFC	"U10"	"V10"	"Q2"
Min. :282.4	Min. : 99078	Min. :2.000	Min. : 4.700	Min. :0.005690
1st Qu.:283.2	1st Qu.: 99623	1st Qu.:4.100	1st Qu.: 9.775	1st Qu.:0.006038
Median :283.8	Median :100244	Median :5.525	Median :12.700	Median :0.006420
Mean :283.9	Mean :100255	Mean :5.228	Mean :11.621	Mean :0.006347
3rd Qu.:284.6	3rd Qu.:100895	3rd Qu.:6.125	3rd Qu.:14.000	3rd Qu.:0.006670
Max. :285.2	Max. :101439	Max. :7.500	Max. :14.600	Max. :0.006810

RAINC	RAINNC	SNOW	TSLB	SMOIS
Min. :0	Min. :0.0000	Min. :0	Min. :273.2	Min. :1
1st Qu.:0	1st Qu.:0.0000	1st Qu.:0	1st Qu.:273.2	1st Qu.:1
Median :0	Median :0.0000	Median :0	Median :273.2	Median :1
Mean :0	Mean :0.1325	Mean :0	Mean :273.2	Mean :1
3rd Qu.:0	3rd Qu.:0.2000	3rd Qu.:0	3rd Qu.:273.2	3rd Qu.:1
Max. :0	Max. :0.7000	Max. :0	Max. :273.2	Max. :1

```
R 4.2.3 ~ /
> describe(edavar12filled_df[, 3:12])
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew
TSK	1	300	283.86	0.82	283.80	283.86	1.04	282.40	285.20	2.8	-0.02
PSFC	2	300	100255.06	716.39	100244.25	100254.16	949.98	99078.00	101439.00	2361.0	0.01
"U10"	3	300	5.23	1.40	5.53	5.31	1.37	2.00	7.50	5.5	-0.49
"V10"	4	300	11.62	2.83	12.70	11.99	2.30	4.70	14.60	9.9	-0.89
"Q2"	5	300	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.0	-0.37
RAINC	6	300	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0	NaN
RAINNC	7	300	0.13	0.22	0.00	0.08	0.00	0.00	0.70	0.7	1.45
SNOW	8	300	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0	NaN
TSLB	9	300	273.20	0.00	273.20	273.20	0.00	273.20	273.20	0.0	NaN
SMOIS	10	300	1.00	0.00	1.00	1.00	0.00	1.00	1.00	0.0	NaN
	kurtosis		se								
TSK			-0.96	0.05							
PSFC			-1.29	41.36							
"U10"			-0.63	0.08							
"V10"			-0.49	0.16							
"Q2"			-1.34	0.00							
RAINC			NaN	0.00							
RAINNC			0.60	0.01							
SNOW			NaN	0.00							
TSLB			NaN	0.00							
SMOIS			NaN	0.00							

Slide 24: Exploratory data analysis and summary statistics of a sample of 300 rows

Analysing the parameters by hours and days across the month, there were also no discernible trends without further and more thorough investigation. Only the soil temperature and moisture remained fairly constant throughout the month in this sample.

2.5.2 Answering the Problem Statement

Forecasting and prediction both statistically and through machine learning methods were carried out to find the viability of a wind power station set up by analysing the resultant wind velocity. Only one of these algorithms was used however—the random forest method—as it proved the most accurate method after comparison (see section 2.3.3). With the knowledge that a small turbine would need an average monthly wind speed of at least 3m/s to 4m/s to be productive, while larger wind farms would require around 6m/s monthly speed (Meyers et al., 2020), the random forest wind speed prediction was summarized statistically as shown in slide 25.

```
530 #####
531 ##### Discussion Calculations
532 # Using random forest prediction and comparing for the required wind speeds of
533 # 3m/s to 6m/s
534
535 cat(rf_predictions)
536 summary(rf_predictions)
537 sd(rf_predictions)
538 var(rf_predictions)
539
```

```
Console Terminal Background Jobs
R 4.2.3 ~ /
> cat(rf_predictions)
7.909075 8.689131 8.645402 8.146638 6.898404 6.432133 5.661614 3.193772 2.781485 2.60807 2.697133
1.834353 2.807292 1.972659 3.638826 4.673379 5.238155 6.672076 7.937306 7.488483 3.769107 2.761543
3.706657 4.827746 4.763785 1.563694 4.513888 4.188275 3.747542 1.221705 1.438213 1.618198 1.842587
2.67231 2.698598 2.501301 4.885088 4.866727 5.11394 5.091284 5.091992 5.578026 6.648284 8.680271
5.514309 4.852365 1.735479 3.735988
> summary(rf_predictions)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 1.222  2.698   4.594   4.491   5.599   8.689
> sd(rf_predictions)
[1] 2.157716
> var(rf_predictions)
[1] 4.65574
```

Slide 25: Summary statistics for siting suitability and results

The results demonstrated the suitability of Colwyn Bay for the wind turbine siting as the mean predicted speed was 4.491m/s, which is higher than the average speed requirement for a small turbine. The location might not however be suitable for a larger wind farm set up which would require speeds of up to 6m/s.

It must be noted at this point that wind regularity throughout the year is also a key consideration in wind power set up, and as such it is advisable to use a longer-term parameter analysis of at least a year when assessing for site suitability.

3. CHALLENGES IN TIME SERIES ANALYSIS AND FORECASTING RELATING TO WIND POWER STATION SITING

By their nature, renewable energy sources such as wind face a number of challenges when it comes to harnessing them for power generation. Despite being a clean energy source, the broad adoption of wind energy is limited for reasons such as availability, unpredictability and consistency. Artificial intelligence, time series analysis and forecasting are tools that have come a long way in guiding us in efficiently harnessing wind energy as it enables us to determine the economics and overall suitability of installing a wind turbine in a given location. Using these tools in the wind power space poses its own inherent issues, some of which include:

- a) **Data Availability and Reliability:** Most renewable energy sources face “sufficient and reliable” data issues as high-capacity renewable power generation is a relatively novel technology. Hence, without the volume and quality of data required to apply time series analysis and forecasting, reliable models and structures for harnessing wind power are not always readily available. Additionally, wind as a part of nature is influenced by other weather factors which could impact the quality of the data measured (as seen from the analysis in the previous chapter, there was a correlation between the wind velocity and temperature). Hence, the various external influences could illicit outliers, noise and irregularities in wind parameter measurements which could impact forecasting and machine learning.

Furthermore, due to unstable seasonality trends and climate changes, already established historical data trends are subject to change, which necessitates the requirement for up-to-

date and reliable data. This could therefore lead to an unprecedented increase in installation budget based on data collection alone.

- b) Skill Gap: Due to the youth and rarity of wind power generation, there are obvious skill gaps in not only understanding the wind power generation infrastructure, but also in the collection and analysis of wind magnitude, speed and direction data. With the scarcity of expertise in the field, wind data pre-processing and analysis often takes times and is prone to errors.
- c) Model Preparation: Since the forecasting horizon length varies inversely with model accuracy, forecasting horizon for wind speed is limited by necessity; and as such model preparation and selection could be a daunting task. A lot of assumptions have to be considered in the model selection process for forecasting which could be based on error, significance levels or the inherent limitations of the model. Certainly, the models would have to be tested against existing historical data to determine their validity especially in view of the assumptions made. In the time series analysis in chapter 2, the random forest model was analysed further via plots because of its relatively lower root mean square error.
- d) Measurements: Even before considering the time series analysis of wind velocity, the assessment of the wind resource for the potential site is usually based on quality and accessible wind measurement, in addition to a micro-siting model that can accurately determine the level of wind distribution over the space or site (Redlinger et al., 2016). Readings simply taken from meteorological stations could be prone to errors as barriers along the way to their mast could impede wind speed, alongside other effects on air flow

with distance. Hence, accurate data for wind turbine siting should preferably be measured on site in order to generate the best models for analysis. This, however, would increase the investment cost required in the wind turbine installation.

4. METHODOLOGY

4.1 Literature Research Methodology

This study used the quantitative and qualitative approaches in its research design and methodology. This involved analyses and reviews of over 20 papers, most of which were referenced in this report; as well as analysis of quantifiable data to build models, forecast and reach conclusions. Data was extracted from Google Scholar databases, Bolton Library, Zotero amongst other sources with particular attention paid to refereed academic journals and books. The Weather, Research and Forecasting Model WRF database provided the historical wind velocity data which formed the basis for the analysis into installation of a wind turbine power station at Colwyn Bay. All analyses were done in R with RStudio. Relevant libraries were imported for pre-processing, transforming, modelling and testing the forecast models.

There were sufficient but relatively limited resources available regarding the concept of time series forecasting in wind turbine installation. However, there were relevant results from both Zotero and Google Scholar with the search strings including keywords *Renewable Energy; Wind Turbine; Big Data; AI; Internet of Things; Forecasting; Machine Learning; Time Series; Data Analysis; Correlation and Regression*.

An inductive reasoning and ontological realist approach was often used in this report, which implied a belief in external reality independent of the researcher (Jenkins, 2010). Moreover, this study adopted a positivist epistemological position, which encourages seeing observable evidence as the only form of defensible scientific finding (Wyly, 2009).

This report was borne on the basis of the 4th industrial revolution and big data explosion; focusing on how the renewable power sector can gain from the analysis of large volumes of historical data to draw models and make predictions.

4.2 Analysis Algorithms

The algorithms used in this research were the statistical model ARIMA and the machine learning models – linear regression, support vector regression and random forest.

4.2.1 ARIMA

The Autoregressive Integrated Moving Average ARIMA is a statistical model for time series analysis and forecasting that uses the concept of lagged values, lagged errors and differencing in order to make its predictions. The ARIMA model has 3 components: AR – Autoregressive term, I – Integrated(Differencing) term and MA – Moving Average term with orders represented by p , d and q respectively which can be determined through auto-correlation and partial auto-correlation plots(Prabhakaran, 2021).

The autoregressive term AR implies that the future values of a time series are dependent on the lagged values; while the moving average term MA implies a dependence on the lagged errors to make predictions. The differencing I helps to eliminate trends and ensure a more constant mean and variance. The number of lags determines the order of the AR (p) and MA(q), while the number of subtractions of previous values is the degree of differencing(d).

4.2.2 Linear Regression

Linear regression is a supervised machine learning technique which assumes a linear relationship between an output variable y and input variable x (Kurama, 2021). This relationship can be depicted with a straight line of best fit and represented with the equation:

$$y = ax + b + e$$

where y is the output variable

x is the input variable

a is the coefficient of the input variable or slope

b is the intercept

e is the error term

Linear regression has the advantage of simplicity in attempting to show the correlation between variables and in predicting outputs. A good linear regression algorithm attempts to minimize the error term e by choosing the best fit values of slope and intercept; it is however limited by its inability to accurately represent non-linear and more complex relationships.

4.2.3 Support Vector Regression SVR

Support vector regression is a more robust machine learning technique that can handle non-linear relationships between an output variable and several input variables. It does this by using kernel functions which can transform complex data into a higher dimension where a linear function can be found. It also uses sparse solution and VC control of the margin for its regression (Awad & Khanna, 2015).

SVR is based on finding the best fit line or curve within a margin of data points called support vectors. Hence, it is robust to outliers as any data points outside of the margin or tolerance level are not considered. SVR aims to minimize errors as it makes predictions based on position of data points to the curve.

4.2.4 Random Forest

According to (Biau & Scornet, 2016), the random forest algorithm is a general-purpose classification and regression method which combines several randomized decision trees and aggregates their predictions by the most occurring output(classification) or the average output(regression).

Random forest is a robust, versatile and an ensemble machine learning algorithm in that it can handle different categories of data, missing values, and new data to its model. Furthermore, it avoids overfitting while giving insight into the relative importance of each feature for prediction.

5. ABOUT WIND POWER GENERATION AND ITS OPERATION

5.1 History

Wind energy has a storied history from its initial applications in water pumping for simple farm operations to a current global power capacity of 837 Giga Watts, according to the Global Wind Energy Council GWEC. The first wind devices were sighted as far back as 200 BC with the vertical axis windmills located at the then Persian-Afghan border. Several centuries later, in the Netherlands and the Mediterranean, the horizontal axis windmills were set up around 1300 and even noticed up till 1875. In the mid to late 19th century, evolution of wind power escalated with the first practical wind turbine developed in Cleveland, Ohio in the United States by Charles F. Brush. This was in 1888 and it was the first large power generator giving an output of 12 Kilo Watts. In Denmark, during the first world war, 25 KW power generating wind turbines became more widespread. The development of wind power continued to progress in Europe between 1935 and 1970, especially in France, Germany, the UK and Denmark, and demonstrated wind power as a sustainable and practical power source for the future.

Once the USA's federal government began promoting wind energy research after the oil crisis fears of 1973, everything changed. This would be remembered as one of the milestones in wind power development as generation from a cap of 25 KW to over 600KW. Subsequently, in California, over 15,000 wind turbines were installed from 1981 to 1990 leading to a total generation output of 1.7 GW (Kaldellis & Zafirakis, 2011). After 1990, Europe took the reins in wind power generation with the European Union EU introducing policies such as the Renewable Energy Directive and the Clean Energy Package in 2001 and 2018 respectively.

China has recently taken the lead in the wind power sector in not only Asia but globally, contributing largely to the 94 GW added wind power capacity. The US also remains a leader in the wind power industry globally, second only to China. Cumulatively, the world is currently at a wind power capacity of up to 837 GW, signifying a biggest step in the most successful renewable energy source till date (Rattan, 2022).

5.2 Operation

In basic terms, a wind power station is operated like most other non-renewable energy power stations. The key difference is the power source, wind, which determines site selection, preparation and sometimes, even turbine selection. Wind power generation is based on the principle of converting mechanical energy of the rotor in the turbine to electrical energy from a generator.

The wind that hits the fan blades of the wind turbine cause them to rotate at a speed which is directly proportional to the speed of the wind. The blades are connected to a rotor which deliver this mechanical kinetic energy to a generator via a shaft connecting them. The generators is designed to convert the kinetic energy of the rotating shaft to an electrical output through a process of electromagnetic induction. Electromagnetic induction involves a moving mechanism(rotor-containing permanent magnets or electromagnets) rotating and thus generating a magnetic field in a stationary mechanism(stator), which consists of coils of wire. The interaction of the magnetic field with the coils of wire leads to electrical current generation in the coils.

As observed, the energy source is the key feature which makes wind power generation stand out. It also is the reason this power generation format can be challenging to work with as several steps are involved in its operation before power generation which include wind resources assessment and forecasting, site selection, turbine selection, site preparation and turbine installation.

Site selection is usually dependent on wind speed, magnitude, direction and forecasting data. Turbine selection can be influenced by cost and maintenance. Site preparation involves the infrastructure built around the site to prepare it for power generation, as well as clearing of obstacles to the wind flow.

A typical wind power station usually involves a large number of inter-connected wind turbines, which after generating electrical power connect to the power grid to supply power to the consumer. The electrical power generated usually goes through a transformer which steps up the voltage for transmission through the grid and final distribution. Hence, regular maintenance and performance monitoring is an integral part of the operation of a wind power station.

5.3 A.I in Wind Power Generation

According to Lee & He(2021), in as early as the 1980s, some form of neural networks and fuzzy logic were already being applied in wind velocity prediction and power generation and vastly improved the reliability of wind forecasting. The next 2 decades saw even more intricate applications of AI in areas such as optimisation of aerodynamics and turbine blade shape, as well as wind farm layout optimisation(Morkos, 2023). Hence, the 1990s and 2000s depicted a

more streamlined application of machine learning and genetic algorithms to optimize the design and operation of wind turbines as well as wind farm integration. The years that followed from 2010 saw the development of AI techniques that facilitated the analysis of large volumes of data in order to provide solutions beyond wind speed monitoring such as: fault detection and predictive maintenance of turbines, power demand analysis, and trend analysis(Baldassano, 2022). This is in conjunction with the Internet of Things and advanced communication networks providing real-time data and updates to ensure turbine performance optimisation. These days, virtual reality and augmented reality can be used to create virtual duplicates of wind farm operations in order to further maximise efficiency.

6. CONCLUSION

The WRF data time series analysis and forecasting showed the invaluable impact these tools (statistical and machine learning) have in wind power station installation and siting. With the explosive growth of the wind power industry over the last decade, especially with growing climate change awareness, the significance of data analytics and AI in this space cannot be over-emphasized. In order to harness this natural resource optimally, as it varies seasonally with time, it is important for wind data models, powered by forecasting and machine learning, to be developed.

A lot of the challenges faced in wind power generation and site location can be restrained with sufficient and quality data measured over time. Leading countries in North America, Europe and Asia have begun to move into the renewable energy space and made commitments to meet the net zero drive by 2050, with the awareness that this can only be achieved with reliable models developed with data analysis, forecasting and machine learning.

While this paper focused on the wind velocity parameter and its impact in wind turbine siting, there is an acknowledgement of the numerable factors like proximity of power station to load, expense, AI knowledge and skill gap amongst others, that can affect siting. Further analysis also needs to be carried out on other atmospheric variables that could impact wind speed and magnitude via correlation testing.

Broadly speaking, despite the breakthroughs made over the decades, there is still a long way to go before wind power is fully embraced as a main source of power in many geographical locations. However, due to the advancements in big data, time series and AI analysis in recent years, there has been astronomical progress in this sector. This paper demonstrates an aspect

of this progress and also serve as a contribution to existing knowledge. The future in renewable energy power generation is bright when married to data analysis and machine learning.

Word Count: 7674

REFERENCES

- Awad, M. and Khanna, R. (2015) "Support vector regression," *Efficient Learning Machines*, pp. 67–80. Available at: https://doi.org/10.1007/978-1-4302-5990-9_4.
- Baldassano, A. (2022) *AI and machine learning: Bridging the divide between a wind development pipeline, operations and roi*, *Windpower Engineering & Development*. Available at: <https://www.windpowerengineering.com/ai-and-machine-learning-bridging-the-divide-between-a-wind-development-pipeline-operations-and-roi/> (Accessed: May 7, 2023).
- Blu, T., Thévenaz, P. and Unser, M. (1970) [PDF] *linear interpolation revitalized: Semantic scholar*, *IEEE Transactions on Image Processing*. Available at: <https://www.semanticscholar.org/paper/Linear-interpolation-revitalized-Blu-Th%C3%A9venaz/5d41f8a5844c9a65ac68c6387d6534907c325f43> (Accessed: 11 May 2023).
- Ellis, C. (2022) *Mtry in random forests, Crunching the Data*. Available at: <https://crunchingthedata.com/mtry-in-random-forests/> (Accessed: 12 May 2023).
- Higgins, S. and Stathopoulos, T. (2021) 'Application of artificial intelligence to Urban Wind Energy', *Building and Environment*, 197, p. 107848. doi:10.1016/j.buildenv.2021.107848.
- Jenkins, C.S. (2010) 'What is ontological realism?', *Philosophy Compass*, 5(10), pp. 880–890. doi:10.1111/j.1747-9991.2010.00332.x.
- Kaldellis, J.K. and Zafirakis, D. (2011) 'The Wind Energy (r)evolution: A short review of a long history', *Renewable Energy*, 36(7), pp. 1887–1901. doi:10.1016/j.renene.2011.01.002.
- Kurama, V. (2021) *Regression in machine learning: What it is and examples of different models*, *Built In*. Available at: <https://builtin.com/data-science/regression-machine-learning> (Accessed: May 8, 2023).
- Lee, M. and He, G. (2021) "An empirical analysis of applications of artificial intelligence algorithms in Wind Power Technology Innovation during 1980–2017," *Journal of Cleaner Production*, 297, p. 126536. Available at: <https://doi.org/10.1016/j.jclepro.2021.126536>.
- Mesoscale & Microscale Meteorology Laboratory (2018) *Weather Research & Forecasting Model (WRF) | Mesoscale & Microscale Meteorology Laboratory*. Available at: <https://www.mmm.ucar.edu/models/wrf> (Accessed: 08 May 2023).
- Meyers, R. et al. (2020) 'A framework for sustainable siting of wind energy facilities: Economic, social, and environmental factors', *Energy Impacts: A Multidisciplinary*

Exploration of North American Energy Development, pp. 315–339.
doi:10.5876/9781646420278.c012.

Morkos, R. (2023) *AI Applications in Wind-Energy Systems*, *Wind Systems Magazine*. Available at: <https://www.windsystemsmag.com/ai-applications-in-wind-energy-systems/> (Accessed: May 7, 2023).

Prabhakaran, S. (2021) *Arima model - complete guide to time series forecasting in python: ML+, Machine Learning Plus*. Available at: <https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/> (Accessed: May 8, 2023).

Rattan, C. (2022) *GWEC: Global wind report 2022*, *Energy Central*. Available at: <https://energycentral.com/c/cp/gwec-global-wind-report-2022> (Accessed: 08 May 2023).

Redlinger, R.Y., Andersen, P.D. and Morthorst, P.E. (2016) 'Wind energy in the twenty-first century', *Springer*. doi:10.1057/9780230524279.

Shumway, R.H. and Stoffer, D.S. (2017) 'Time series analysis and its applications', *Springer Texts in Statistics* [Preprint]. doi:10.1007/978-3-319-52452-8.

Silva, R.P. *et al.* (2021) 'Time series segmentation based on stationarity analysis to improve new samples prediction', *Sensors*, 21(21), p. 7333. doi:10.3390/s21217333.

Wyly, E. (2009) 'Strategic positivism*', *The Professional Geographer*, 61(3), pp. 310–322. doi:10.1080/00330120902931952.

APPENDIX

Below is the complete time series WRF data analysis process/code, with particular focus on the wind velocity variables and their measurements against time, which was done in R on RStudio.

```
#####  
#####  
  
### EXPLORATORY DATA ANALYSIS OF THE ####  
  
### WEATHER RESEARCH AND FORECASTING MODEL DATA ###  
  
  
# Imported the .csv file  
df <- read.csv("WRFdata_May2018.csv")  
View(df)  
class(df)  
nrow(df)  
ncol(df)  
  
# Imported janitor, dplyr and tidyverse libraries to clean data frame and  
# correct column names  
library(janitor)  
library(dplyr)  
library(tidyverse)  
  
anomaly_loc <- which(df == "[ ]", arr.ind = TRUE)  
anomaly_loc # Printed the row and column indexes where the brackets were found  
  
colnames(df)[6]  
df <- df %>%
```

```
mutate(X.4 = replace(X.4, match("[]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]",
X.4), NA))
```

```
View(df)
```

```
# Noticed the wrong column names issue
```

```
colnames(df)
```

```
nrow(df)
```

```
df_1 <- df %>% row_to_names(row_number = 1)
```

```
colnames(df_1)
```

```
#Number of rows confirmed wrong column names have been deleted
```

```
nrow(df_1)
```

```
View(df_1)
```

```
#####
####
```

```
### EDA WITH FIRST 300 ROWS ###
```

```
# In view of the fact that missing XLAT and XLONG figures could not simply
# be averaged or randomly imputed, over 300 rows were selected as those
# with either a missing XLAT or XLONG figure would eventually be removed.
```

```
df_2 <- df_1[1:350,] %>%
```

```
drop_na(XLAT, XLONG)
```

```
nrow(df_2)
```

```
eda300_df <- as.data.frame(df_2[1:300, ])
```

```
class(eda300_df)
```

```
View(eda300_df)
```

```
# In order to choose the technique for filling in the missing values of  
# eda300_df, the latitude and longitude had to be arranged in order. This  
# could only be done with the appropriate variable data type.  
str(eda300_df)
```

```
eda300_df <- as.data.frame(sapply(eda300_df, as.numeric))  
# changed values from character type to numeric data type
```

```
class(eda300_df)  
str(eda300_df)
```

```
eda300_df <- eda300_df[order(eda300_df$XLAT, eda300_df$XLONG), ]  
rownames(eda300_df) <- NULL  
class(eda300_df)  
View(eda300_df)
```

```
# It was observed that there were essentially 10 unique columns (column names  
# with the exceptions of XLAT and XLONG - the reference variables) repeated  
# 248 times for different time intervals
```

```
ncol(eda300_df)
```

```
# Since XLAT and XLONG were the reference variables, they were not repeated. The  
# next 10 columns were however repeated. Calculations were done to determine the  
# number of repetitions
```



```

rep_unique_var <- (ncol(eda300_df)-2)/10
rep_unique_var

# The 2 reference and 10 unique variables were explored to proceed on how to
# treat the missing variables for each field
edavar12_df <- eda300_df[,1:12]
edavar12_df
class(edavar12_df)
View(edavar12_df)

# After examination, interpolation was chosen as the technique to be adopted
# for filling the NA's in the EDA 300 rows

# Missing values filled with linear interpolation
install.packages("zoo") #zoo package installed and imported
library(zoo)

eda300filled_df <- na.approx(eda300_df, rule = 2, na.rm = FALSE, maxgap = Inf)
sum(is.na(eda300filled_df))

```

```

#####
#####

```

```
#####
```

```
### LOCATION COORDIATES AND CHOSEN TIME SERIES VARIABLE ANALYSIS ###
```

```
# A location in the UK was selected to determine its suitability for  
# for a wind power station or turbine set up.
```

```
# Hence coordinates XLAT = 53.394 and XLONG = -3.768 were selected which  
# are coordinates for a location near Colwyn Bay, North Wales.
```

```
rowindex <- which(apply(df_1, 1,  
                        function(x) x["XLAT"] == "53.394" &  
                        x["XLONG"] == "-3.768"))
```

```
rowindex
```

```
colwyn_data <- df_1[2798,]
```

```
View(colwyn_data)
```

```
colnames(colwyn_data) <- gsub("\\", "", colnames(colwyn_data))
```

```
colwyn_data
```

```
length(colnames(colwyn_data))
```

```
ucolname_ind <- which(colnames(colwyn_data) == "U10")
```

```
ucolname_ind
```

```
length(ucolname_ind)
```

```
vcolname_ind <- which(colnames(colwyn_data) == "V10")
```

```
length(vcolname_ind)
```

```

# Found the indexes of missing column names
missing_cols_idx <- which(is.na(colnames(colwyn_data)))
missing_cols_idx
length(missing_cols_idx)

sum(grepl("5$", missing_cols_idx))
sum(grepl("6$", missing_cols_idx))

# Found the indexes of missing column names ending with "5"
missing_cols_idx_5 <- missing_cols_idx[grepl("5$", missing_cols_idx)]
missing_cols_idx_6 <- missing_cols_idx[grepl("6$", missing_cols_idx)]
missing_cols_idx_5
missing_cols_idx_6

colnames(colwyn_data)[missing_cols_idx_5] <- "U10"
colnames(colwyn_data)[missing_cols_idx_6] <- "V10"
colnames(colwyn_data)

length(which(is.na(colnames(colwyn_data))))

length(colnames(colwyn_data))
View(colwyn_data)

# Searched for the U10 and V10 values
my_values_u <- colwyn_data[1, grep("U10", names(colwyn_data))]
my_values_u
length(my_values_u)
typeof(my_values_u)

```

```
output_listu <- list() # created an empty list to store the outputs
```

```
for (i in my_values_u) {  
  output_listu <- c(output_listu, list(i))  
}
```

```
output_listu <- unlist(output_listu)
```

```
output_listu
```

```
typeof(output_listu)
```

```
my_values_v <- colwyn_data[1, grep("V10", names(colwyn_data))]
```

```
my_values_v
```

```
length(my_values_v)
```

```
output_listv <- list() # created an empty list to store the outputs
```

```
for (i in my_values_v) {  
  output_listv <- c(output_listv, list(i))  
}
```

```
output_listv <- unlist(output_listv)
```

```
output_listv
```

```
typeof(output_listv)
```

```
vel_df <- data.frame(output_listu, output_listv) # formed the velocity df
```

```
vel_df
```

```
vel_df$U10_values <- as.numeric(vel_df$output_listu)
```

```
vel_df$output_listu <- NULL
```

```
vel_df$V10_values <- as.numeric(vel_df$output_listv)
```

```

vel_df$output_listv <- NULL
vel_df

# Used interpolation to fill NA's
vel_df <- as.data.frame(na.approx(vel_df, rule = 3,
                                na.rm = FALSE, maxgap = Inf))
vel_df
sum(is.na(vel_df))
class(vel_df)

# Resultant velocity calculated
vel_df$resultant_vel <- sqrt(vel_df$U10_values^2 + vel_df$V10_values^2)
vel_df$resultant_vel
vel_df

resultvel <- vel_df[, -1:-2]
resultvel

# Created the sequence of dates and times with a frequency of 3 hours
datetimes <- seq(from=as.POSIXct("2018-05-01 00:00"),
                 to=as.POSIXct("2018-05-31 21:00"),
                 by="3 hours")
datetimes
typeof(datetimes)
class(datetimes)

rvel_df <- data.frame(datetimes, resultvel)
rvel_df
# Plotted the resultant velocity against time

```

```
library(ggplot2)

ggplot(rvel_df, aes(x = datetimes, y = resultvel)) + geom_line() +
  labs(title = "Colwyn Bay May 2018 Wind Velocity Graph",
        x = "DateTime", y = "Resultant Velocity")
```

```
# Creating a time series, libraries are imported
```

```
install.packages("forecast")
```

```
library(forecast)
```

```
# Used zoo() to create the time series
```

```
vel_ts <- zoo(rvel_df$resultvel,
              order.by = rvel_df$datetimes)
```

```
vel_ts
```

```
head(vel_ts)
```

```
length(vel_ts)
```

```
# Aggregated the time series
```

```
ag.vel_ts <- aggregate(vel_ts, as.Date, mean)
```

```
ag.vel_ts
```

```
daily <- index(ag.vel_ts)
```

```
daily
```

```
length(ag.vel_ts)
```

```
# Checked for outliers in the ts
```

```
ag.vel_ts <- tsclean(ag.vel_ts)
```

```
ag.vel_ts
```

```
summary(ag.vel_ts)
```

```
# Plotted the time series
```

```

ggplot(ag.vel_ts, aes(x = daily, y = ag.vel_ts)) + geom_line() +
  labs(title = "Aggregated Colwyn Wind Velocity",
        x = "Date", y = "Mean Resultant Velocity")

dev.off()

# Other time series analysis

# Displayed the degree of correlation at different lags
tsdisplay(vel_ts, lag.max = 50)

library(tseries)
adf.test(vel_ts)
# Failed to reject the null hypothesis since p-value > 0.05 hence, time series
# time series is non-stationary

vel_arima <- auto.arima(vel_ts, trace=T, stepwise = F, approximation = F)
vel_arima
# Setting trace as True in order to trace the model selection process,
# an ARIMA(1,0,1) with non-zero mean was arrived at, AR term of 0.8219,
# MA term of 0.1114 and differencing of 0.

arima_rmse <- accuracy(vel_arima)[2]
arima_rmse

```

```
#####
```

```
#####
```

```
# Machine Learning
```

```
# Loaded the required packages
```

```
library(lubridate)
```

```
library(ggplot2)
```

```
library(zoo)
```

```
install.packages("caret")
```

```
library(caret)
```

```
library(future)
```

```
vel_ts
```

```
rvel_df <- fortify(vel_ts) #Fortified vel_ts to a data frame
```

```
rvel_df
```

```
sum(is.na(rvel_df))
```

```
colnames(rvel_df) <- c("datetime", "wind_velocity")
```

```
class(rvel_df$datetime)
```

```
# The data frame was split into training and testing sets
```

```
set.seed(321)
```

```
trainIndex <- createDataPartition(rvel_df$wind_velocity, p = 0.8, list = FALSE)
```

```
train <- rvel_df[trainIndex, ]
```

```
nrow(train)
```

```
test <- rvel_df[-trainIndex, ]
```

```
nrow(test)
```

```
# Linear regression model
```

```
install.packages("readr")
```



```

library(readr)

# Specified the time series cross-validation method
lm_tc <- trainControl(method = "timeslice", initialWindow = 24,
                      horizon = 1, fixedWindow = TRUE)

# specified the model and its parameters
lm_model <- train(wind_velocity ~ datetime, data = train,
                  method = "lm", trControl = lm_tc, preProc = NULL)

# Used the trained model to make predictions on the test set
lm_predictions <- predict(lm_model, newdata = test)
lm_predictions
length(lm_predictions)

# Calculated the mean squared error and root mean squared error
lm_mse <- mean((test$wind_velocity - lm_predictions)^2)
lm_rmse <- sqrt(lm_mse)
lm_rmse

# Support Vector Machine Regression SVMR model
install.packages("e1071")
library(e1071)

# Specify the time series cross-validation method
svm_tc <- trainControl(method = "timeslice", initialWindow = 24,
                      horizon = 1, fixedWindow = TRUE)

```

```
# Specify the model and its parameters
svm_model <- train(wind_velocity ~ datetime, data = train,
  method = "svmRadial", trControl = svm_tc,
  preProc = NULL, tuneGrid = expand.grid(sigma = 10.33358, C = 2))
```

```
svm_model
```

```
# Use the trained model to make predictions on the test set
svm_predictions <- predict(svm_model, newdata = test)
svm_predictions
```

```
# Calculate the mean squared error and root mean squared error
svm_mse <- mean((svm_predictions - test$wind_velocity)^2)
svm_mse
svm_rmse <- sqrt(svm_mse)
svm_rmse
```

```
# Random Forest model
install.packages("randomForest")
library(randomForest)
```

```
# specify the time series cross-validation method
rf_tc <- trainControl(method = "timeslice", initialWindow = 24,
  horizon = 1, fixedWindow = TRUE)
```

```
# specify the model and its parameters
set.seed(152)
rf_model <- train(wind_velocity ~ datetime, data = train,
```

```

        method = "rf", trControl = rf_tc, preProc = NULL, tuneLength = 10)
rf_model

# use the trained model to make predictions on the test set
rf_predictions <- predict(rf_model, newdata = test)
rf_predictions

# calculate the error metrics
rf_mse <- mean((rf_predictions - test$wind_velocity)^2)
rf_mse
rf_rmse <- sqrt(rf_mse)
rf_rmse

# Create a table of model performance metrics
models <- c("ARIMA", "Linear Reg", "SVMR", "Random Forest")
total_rmse <- c(arima_rmse, lm_rmse, svm_rmse, rf_rmse)
results <- data.frame(Model = models, RMSE = total_rmse)

# Print the table of model performance metrics
print(results)

# Based on the RSME scores, the most accurate model - random forest - would be
# plotted first showing its predicted vs. actual values

# Create data frame of actual and predicted values with time stamps
install.packages("ggpubr")
library(ggpubr)

```

```

# Random forest plot
rfres <- data.frame(
  time = test$datetime,
  actual = test$wind_velocity,
  predicted = rf_predictions
)

# Plot actual vs. predicted wind velocity against time
ggplot(rfres, aes(x = time)) +
  geom_line(aes(y = actual, color = "Actual")) +
  geom_line(aes(y = predicted, color = "Predicted")) +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "red")) +
  labs(x = "Time", y = "Wind Velocity",
       title = "Actual vs. Predicted Wind Velocity(Random Forest)")

```

```

# SVMR plot
svmres <- data.frame(
  time = test$datetime,
  actual = test$wind_velocity,
  predicted = svm_predictions
)

# Plot actual vs. predicted wind velocity against time
ggplot(svmres, aes(x = time)) +
  geom_line(aes(y = actual, color = "Actual")) +
  geom_line(aes(y = predicted, color = "Predicted")) +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "green")) +
  labs(x = "Time", y = "Wind Velocity",
       title = "Actual vs. Predicted Wind Velocity(SVMR)")

```

```

# Linear regression plot
lmres <- data.frame(
  time = test$datetime,
  actual = test$wind_velocity,
  predicted = lm_predictions
)

# Plot actual vs. predicted wind velocity against time
ggplot(lmres, aes(x = time)) +
  geom_line(aes(y = actual, color = "Actual")) +
  geom_line(aes(y = predicted, color = "Predicted")) +
  scale_color_manual(values = c("Actual" = "blue", "Predicted" = "orange")) +
  labs(x = "Time", y = "Wind Velocity",
       title = "Actual vs. Predicted Wind Velocity(Linear Regression)")

#####
####

# Correlation and Testing

# To find the relationship between surface temperature and resultant wind
# velocity, the Pearson correlation was used

# But first, the surface temperature for Colwyn Bay for the period was
# filtered out with the steps:

```

```

colwyn_data <- df_1[2798,]
colwyn_data %>% View()

tsk_ind <- which(colnames(colwyn_data) == "TSK") #Surface temperature TSK index
tsk_ind
length(tsk_ind)

# Since the expected result of 248 indexes not applicable, the
# indexes of missing column names was found
missing_cols_idx <- which(is.na(colnames(colwyn_data)))
missing_cols_idx
length(missing_cols_idx)

sum(grepl("3$", missing_cols_idx)) #Chose indexes that end with 3

# Found the indexes of missing column names ending with "3"
missing_cols_idx_3 <- missing_cols_idx[grepl("3$", missing_cols_idx)]
missing_cols_idx_3

colnames(colwyn_data)[missing_cols_idx_3] <- "TSK"
colnames(colwyn_data)

length(which(is.na(colnames(colwyn_data))))

# Searched for the TSK values
tsk_values <- colwyn_data[1, grep("TSK", names(colwyn_data))]
tsk_values
length(tsk_values) #Number of TSK values now correctly identified as 248
typeof(tsk_values)

```

```

output_tsk <- list() # created an empty list to store the outputs

for (i in tsk_values) {
  output_tsk <- c(output_tsk, list(i))
}
output_tsk <- unlist(as.numeric(output_tsk))
output_tsk
typeof(output_tsk)

# Linear interpolation was used for missing values
output_tsk <- na.approx(output_tsk, rule = 2, na.rm = FALSE, maxgap = Inf)
output_tsk

# Invoked the resultant velocity data frame rvel_df and added the
# surface temperature variable to it
rvel_df$surftemp <- output_tsk
rvel_df
rvel_df %>% View()

# Used min-max scaling on velocity and temperature variables
mm1_vel <- (rvel_df$wind_velocity - min(rvel_df$wind_velocity))
mm2_vel <- (max(rvel_df$wind_velocity)-min(rvel_df$wind_velocity))
mm_resultvel<- mm1_vel/mm2_vel
head(mm_resultvel)

mm1_tsk <- (rvel_df$surftemp - min(rvel_df$surftemp))
mm2_tsk <- (max(rvel_df$surftemp)-min(rvel_df$surftemp))
mm_surftemp<- mm1_tsk/mm2_tsk

```

```

head(mm_surftemp)

rvel_df$mm_resultvel <- mm_resultvel
rvel_df$mm_surftemp <- mm_surftemp
head(rvel_df) #The scaled variables added to the data frame

# Spearman's rank correlation was used for wind velocity and surface temperature
# variables correlation test
rvel_df %>%
  select(mm_resultvel, mm_surftemp) %>%
  cor(method='spearman')

rvel_df %>%
  cor.test(~mm_resultvel + mm_surftemp, data = ., method='spearman')

#####
####

##### Discussion Calculations
# Using random forest prediction and comparing for the required wind speeds of
# 3m/s to 6m/s

cat(rf_predictions)
summary(rf_predictions)
sd(rf_predictions)
var(rf_predictions)

# Using summary() and describe() for EDA of the unique variables along the column

```



```

install.packages("psych") #imported psych package in order to use describe()
library(psych)
edavar12_df
edavar12filled_df <- eda300filled_df[,1:12]
edavar12filled_df
class(edavar12filled_df)

summary(edavar12filled_df[, 3:12])
describe(edavar12filled_df[, 3:12])

# Plotting the scaled velocity vs surface temperature for correlation testing

ggplot(rvel_df, aes(x = mm_resultvel, y = mm_surftemp)) +
  geom_point(color = "lightgreen") +
  geom_smooth(method = "lm", color = "red", se = TRUE) +
  labs(x = "Resultant Velocity(Scaled)", y = "Surface Temperature(Scaled)") +
  ggtitle(paste("Velocity - Temperature Correlation:", round(cor(rvel_df$mm_resultvel,
    rvel_df$mm_surftemp, method = "spearman"), 5)))

```