Weather Analysis of Warrington: A Statistical, Machine Learning and Timeseries Approach

DATA SCIENCE
(DAT7006)

MSc Data Analytics and Technologies

Student ID: 2304554
Tutor: Anchal Garg (Ph.D.)

Submission Date: 15 May 2024

ABSTRACT

The weather analysis has been one of the major analyses that helps understand the world's ecosystem. In this report, the weather variables of Warrington, United Kingdom are the focal point and analysis will be helpful to various stakeholders – the residence, production companies and government officials in the location to make sound decisions.

In this report, previous literature on the analysis to be done are also explored. These help to understand statistical analysis, machine learning and timeseries analysis done in various sectors, including weather analysis. Also, the methodology used is also reported with CRISP-DM being the method of choice. Here, the six phases of CRISP-DM are talked about in relation to the given dataset.

The data analysis aspect comprises of data restructuring where the data given is properly structured into a format that can be easily analysed. Then data cleaning, followed by statistical analysis – Univariate, Bivariate and Multivariate; Machine learning and Timeseries analysis.

Statistical, timeseries and machine learning approaches are combined in this study to gain good insight into the climatic condition of Warrington, United Kingdom. The data used is a secondary data of history climatic conditions of the United Kingdom for May 2018.

Statistical analysis approach used to explore relationship between climatic conditions and these analyses include univariate, bivariate and multivariate analysis. The machine learning analysis include simple linear regression, support vector machine and Random Forest. Lastly, time-series analysis done to predict the future climatic variables are auto Arima and Arima models.

The quick overview of all the analysis done in this report is as shown in the figure below:

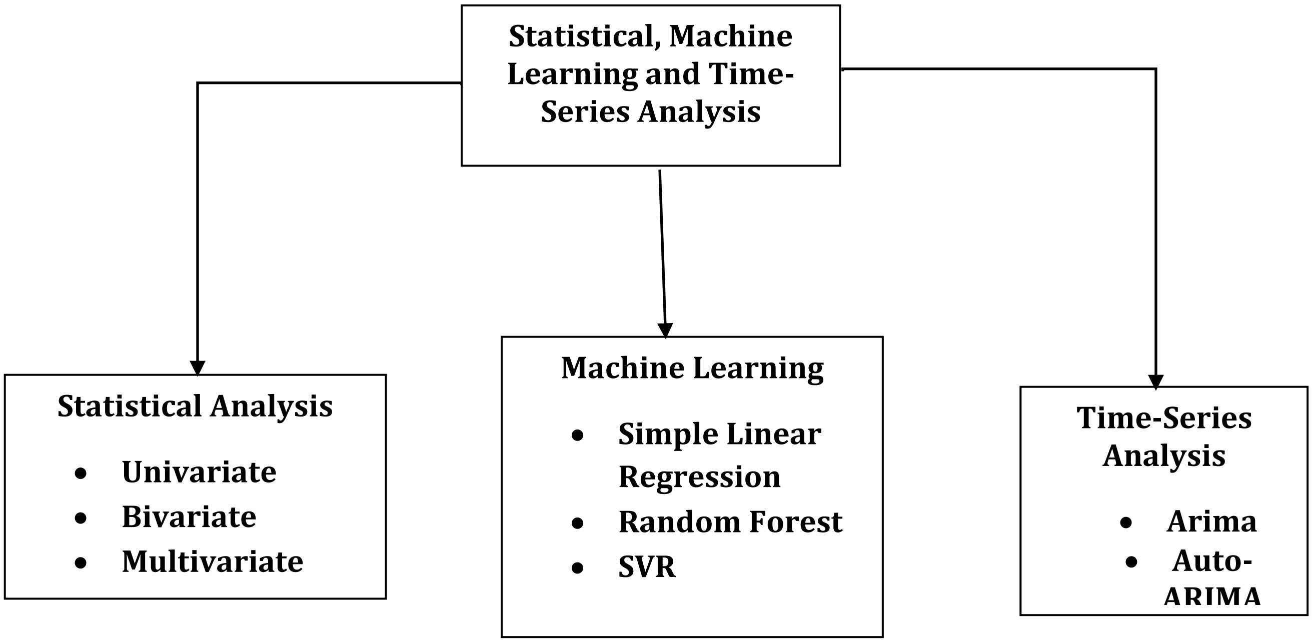


Figure 1 - Overview of the Analysis in the Report

LIST OF ABBREVIATIONS

WS – Wind Speed

TSK – Skin Temperature

SMOIS – Soil Temperature

PSFC – Surface Pressure

TSLB – Soil Temperature

Q2 – Specific Humidity

RAINC – Convectional Rain

RAINNC – Non-Convectional Rain

PACF – Partial Autocorrelation Function

ACF - Autocorrelation Function

ARIMA – Autoregression Integration Moving Average

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS

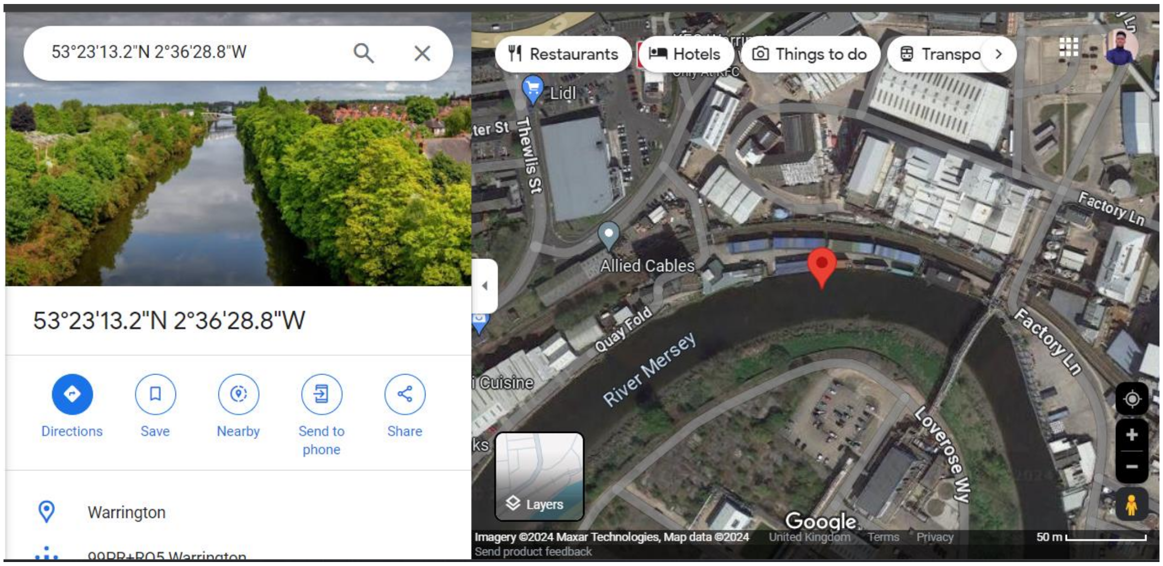
1.0 INTRODUCTION

Understanding the climatic condition of a region is essential to shaping the ecosystem of the area. The data given is the weather condition of different places, however, the Warrington location in the UK will be the main focal point of this report. The dataset contains a total of 10 climatic factors like skin temperature, wind speed, soil temperature, snow. The statistical and machine learning analytics will be focused on to explore these climatic features.

In this report, statistical, time-series forecasting, and machine learning analytics will be performed. Machine learning models like the Regression, support vector machine and random forest will be used to investigate the patterns and trends of each climatic condition and the time-series to predict the future weather condition in the selected region. Furthermore, univariate, bivariate, and multivariate analysis will be performed to determine the relationship between one, two and multiple climatic factors respectively.

1.1 About the Selected Location

From the dataset, the geographical location of interest with coordinates 53.387, -2.608 is Warrington according to Google Map. From the last census carried out in the UK, the population of people living in Warrington is 210,000 which represent a 4.3% increase during the last 10 years (warrington.gov.uk, 2021). It is in the Chesire County, the North-West of England and its population makes up of residential and commercial industries.



53°23'13.2"N 2°36'28.8"W

53°23'13.2"N 2°36'28.8"W

Warrington

Figure 2 - Google Map View of the Selected Location

1.2 Significance of the Location

Warrington is an industrial area that houses many industries and as well a home for many citizens. The knowledge of the weather condition will help those industries plan. For instance, Solvay Interox is a part of Solvay Group which has one of its facilities located at Warrington and specialises in production of hydrogen peroxide. They understand the impact the weather conditions can have on the quality of their produce and will therefore put good measures in place to mitigate any harmful effects these conditions can have on them to ensure quality products are being produced always.

1.3 Problem Statement

The knowledge of weather conditions plays crucial role in how people plan for their day-to-day living. For instance, the forecast that rain will fall the coming days will help an individual smartly adjust their plan to accommodate this likely event and same goes to companies, especially the production industries.

This report explores the weather conditions in Warrington through statistical, machine learning and time-series forecasting analysis. Emphasis will be given to weather conditions like the skin temperature, wind speed, soil temperature, non-convective rain, and convective rain to understand relationship between these weather conditions and their impact on the locality selected as a whole.

1.4 Research Questions

To ensure that the research stay on course with what is needed and important to the stakeholders, the following research questions will be considered and satisfactorily answered in this study:

- Is there a relationship the relationship between skin temperature and soil temperature? How does the skin temperature affect the soil temperature?
- Checking the relationship between specific humidity and soil moisture, Is there a correlation between them?
- Does wind speed and skin temperature have any relationship?
- Can the significance level of pressure, specific humidity, wind speed, soil moisture, convectional rain, non-convectional rain, and soil temperature on skin temperature be gotten?
- How well can machine learning model perform to predict the skin temperature of the location?
- Can the following month skin temperature be predicted from the given data?

1.5 Hypothesis

Hypothesis 1

Null Hypothesis (H_0): There is no relationship between skin temperature and soil temperature.

Alternate Hypothesis (H_1): There is relationship between the skin temperature and soil temperature.

Hypothesis 2

Null Hypothesis (H_0): There is no relationship between the specific humidity and the snow amount.

Alternate Hypothesis (H_1): These is a correlation between specific humidity and Soil moisture.

Hypothesis 3

Null Hypothesis (H_0): There exists no correlation between skin temperature and wind speed.

Alternate Hypothesis (H_1): There is correlation between skin temperature and wind.

1.6 Significance of Data Analysis

1.6.1 Rationale for Data Analysis

The overall aim of the study is to use statistical and machine learning analysis to understand the relationship between weather conditions in Warrington.

The specific Objectives Include:

- Data understanding and Identification of a location within the United Kingdom and identifying major stakeholders that can benefit from weather analysis.
- Data restructuring, cleaning, and preprocessing
- Statistical and machine learning analysis to understand the relationship between variables.
- Time-Series analysis to predict the expected future weather conditions.

1.6.2 Stakeholders

Residence: The stakeholders according to the location selected are primarily the residence of Warrington who will face these weather conditions daily and must plan for it.

Production Companies: The production companies whose produce can be affected by the weather conditions are also a major stakeholder. For this report, emphasis is given to Solvay Interox, a chemical company that produces hydrogen peroxide.

Political Leaders: These are the elected officials in Warrington.

1.6.3 How Analysis will help Stakeholders.

Residence: The data analysis will help them plan their day-to-day activities in accordance with the expectations of what the weather conditions will be through the data analysis

Solvay Interox: Extreme temperature will lead to decaying of their product, reduces its quality, and causes safety issues.

Political Leaders: The data analysis will help them to accurately educate their followers and also give them the idea about the best infrastructure to put in place to mitigate any foreseeable harsh weather conditions.

2.0 LITERATURE REVIEW

Temperature is one of the most important climatic variables that has played crucial roles at influencing ecological, social, and economical aspect of human life (Dorman *et al.*, 2015). Therefore, understanding the temperature of a location is an important one for stakeholders of the said location.

The use of statistical analysis, machine learning and timeseries has been greatly used in recent years especially when studying temperature of a location due to their ability to help gain actionable insight, predicting a weather condition from other factors and predicting the future weather (Lang *et al.*, 2023).

2.1 Statistical Analysis

Statistical analysis which can be univariate, bivariate and multivariate analysis has been employed to accurately analyse the climatic data and draw meaningful conclusions (Yoon *et al.*, 2012). Descriptive analysis like the mean, median, mode, and standard deviation which are capable of providing basic insight has been used for temperature changes and variations (Obsi *et al.*, 2021). Bivariate and multivariate statistical analysis include the use of correlation to compare the relationship between two or more variables on the output (Hsu and Dirmeyer, 2021).

Furthermore, regression analysis has also been used by several researchers to explore the relationship between temperature and other climatic variables (Wen *et al.*, 2019). Though, statistical analysis offer a robust explanation on relationship between variables, they may struggle to capture non-linear relationships quite well especially for a climatic condition data that may depend on many unforeseen factors (Hao, Singh and Xia, 2018).

2.2 Machine Learning

Machine Learning models are very powerful tools that can be harnessed to analyse climatic data as it has the ability to handle large volume data, linear and non-linear relationships and understand complex pattern (Esmaiil *et al.*, 2022).

Regression models has proven to be effective while predicting temperature as a variable and models like the support vector machine has been relied on to accurately predict the temperature of a particular location using other climatic conditions as input (Dong *et al.*, 2022)

2.3 Timeseries Analysis

Temperature trends has been studied using timeseries methods which are specifically designed to analyse temporal data (Ichimura and Kamada, 2017).

ARIMA (Autoregressive integrated moving average) is a classic timeseries model as it offers simplicity and efficiency in forecasting temperatures (Wang, Shen and Jiang, 2018).

3.0 METHODOLOGY

3.1 Cross-Industry Standard Process for Data Mining

Cross-Industry Standard Process for Data Mining (CRISP-DM) is a proven industry methodology that guides data mining process (Wiemer, Drowatzky and Ihlenfeldt, 2019; Jaggia *et al.*, 2020). It entails the following six phases:

3.1.1 Business Understanding

This stage deals with understanding the goal and objectives of the weather condition analysis. For this analysis, understanding the weather conditions that are crucial to the relevant stakeholders is vital.

3.1.2 Data Understanding

After the data has been downloaded, it is essential to understand it by opening the dataset to understand the structure, the type of the data and the description of the variables in the dataset, the format, and the data type of all the variables. This also helps in understanding what is to be done in the next step – data preparation.

Understanding the given dataset and its description:

The dataset given is a secondary data that contains the weather information of different places across the United Kingdom. It is a sequential dataset, and all records were made every three hours for the whole month of May 2018. This means, there are a total of 8 records made per day for 30 days.

The variables in the data are explained in the table below:

Table 1 - Description of Dataset

Parameter	Description	Measuring Unit
XLAT	Latitude	
XLONG	Longitude	
TSK	Skin temperature or surface temperature	^oK (Kelvin)
PSFC	Surface pressure	Pa (Pascal)
U10	X component of wind at 10m	m/s
V10	Y component of wind at 10m	m/s
Q2	2- meter specific humidity	Kg/Kg
Rainc	Convective rain (Accumulated precipitation)	Mm
Rainnc	Non-convective rain	Mm
Snow	Snow water equivalent	Kg/m^2
TSLB	Soil temperature	^oK
SMOIS	Soil Moisture	m^3/m^3

The data being a csv file was opened with Microsoft Excel and was found to have a total of 2482 columns and 5453 rows. This means, there are 5453 geographical locations climatic data recorded every 3 hours over the period of May 2018.

3.1.3 Data Preparation

Here, the dataset is loaded into R-Programming environment for the essential data restructuring. Restructuring is done at this phase which includes formatting variables into correct data type, dealing with missing values and outliers.

The original dataset was given in such a way that it will need restructuring before any data cleaning or analysis can be carried out on it. The steps involved in the data restructuring is as shown in the figure below:

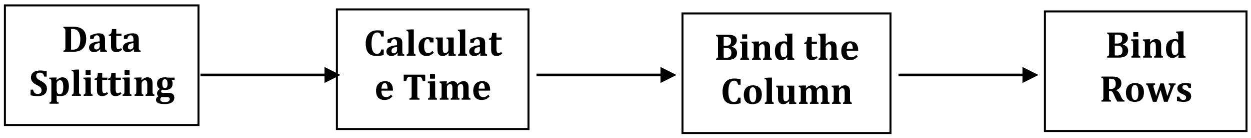


Figure 3 - Data Restructuring Process Flow Chart

3.1.4 Modelling

Selecting a befitting model for dataset is an important phase of the CRISP-DM because not every model will suit a dataset. For this report, regression models will be used because all the predictions are numerical values. Univariate time-series model will also be built at this phase.

3.1.5 Evaluation

The result of analysis is evaluated, inspected, and reviewed at this phase. The accuracy of each model is gotten by calculating the mean squared error of the models.

3.1.6 Deployment

The efforts analysing dataset is only useful when the results can be accessed (Turanoğlu Bekar et al., 2020). At this stage, the best machine learning model and the time-series model built will be deployed for usage.

4.0 DATA PRE-PROCESSING STEP

Below are the data restructuring processes:

4.1 Data Restructuring

The original dataset was given in such a way that it will need restructuring before any data cleaning or analysis can be carried out on it. The steps involved in the data restructuring is as delineated below.

4.1.1 Loading Important Libraries

Important R libraries are loaded to ensure every function needed for the data analysis is loaded. This is done via the code snippet shown below:

```
#Install Required Libraries
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.2.3

## Warning: package 'ggplot2' was built under R version 4.2.3

## Warning: package 'tibble' was built under R version 4.2.3

## Warning: package 'tidyr' was built under R version 4.2.3

## Warning: package 'readr' was built under R version 4.2.3

## Warning: package 'purrr' was built under R version 4.2.3

## Warning: package 'dplyr' was built under R version 4.2.3

## Warning: package 'stringr' was built under R version 4.2.3

## Warning: package 'forcats' was built under R version 4.2.3

## Warning: package 'lubridate' was built under R version 4.2.3

## Warning in Sys.timezone(): unable to identify current timezone 'C':
## please set environment variable 'TZ'

## ── Attaching core tidyverse packages ──────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.4     ✓ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.0     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(ggplot2)
library(lubridate)
library(dplyr)
library(forecast) #Install this package
```

```
## Warning: package 'forecast' was built under R version 4.2.3
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(skimr) #Install this package
```

```
## Warning: package 'skimr' was built under R version 4.2.3
```

```
library(imputeTS) #Install this package
```

```
## Warning: package 'imputeTS' was built under R version 4.2.3
```

```
library(cowplot) #Install this package
```

```
## Warning: package 'cowplot' was built under R version 4.2.3
```

```
##
## Attaching package: 'cowplot'
##
## The following object is masked from 'package:lubridate':
##
##     stamp
```

```
library(tseries) #Install this package
```

```
## Warning: package 'tseries' was built under R version 4.2.3
```

```
##
## Attaching package: 'tseries'
##
## The following object is masked from 'package:imputeTS':
##
##     na.remove
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3

## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin

install.packages('randomForest')

## Warning: package 'randomForest' is in use and will not be installed

library(ggcorrplot)

## Warning: package 'ggcorrplot' was built under R version 4.2.3

install.packages('ggcorrplot')

## Warning: package 'ggcorrplot' is in use and will not be installed

library(bestNormalize)

## Warning: package 'bestNormalize' was built under R version 4.2.3

install.packages('bestNormalize')

## Warning: package 'bestNormalize' is in use and will not be installed
```

Figure 4 - Code to install important libraries.

4.1.2 Loading the Data

Since the data is in .csv extension, the function to read a .csv file is used to load the data as shown below. The skip=1 is used to skip the first row containing the date-time information.

```
#Read the whole dataset except the Time row
DF <- read.csv('WRFdata_May2018.csv', skip=1)
```

Figure 5 - Code to read the data given into R.

The code above reads all the data but skips the first row which contains the date-time.

4.1.3 Extracting the Selected Region

A subset of the data frame is extracted from the large data which is the row containing the selected region for analysis which gives a data frame that contains 2480 columns and 1 row.

```
#Extract the rows of the preferred location
df_subset <- subset(DF, XLAT == 53.387 & XLONG == -2.608)
```

Figure 6 - Code to extract the selected location.

4.1.4 Splitting and Joining the Data

The region data is then split into 248 data frames with 10 columns each containing the weather variables recorded for the region. The XLAT and XLONG columns are removed, and the extracts are appended into an empty list created.

```
df_subset <- df_subset %>% dplyr::select(-c(XLAT,XLONG))
```

Figure 7 - Code that removes the XLAT and XLONG variable from the selected region data frame.

There are 10 variables recorded per time. The data frame will be divided into smaller data frame.

```
num_cols <- ncol(df_subset)
cols_per_df <- 10
columns <- seq(1, ncol(df_subset), by=cols_per_df)
column_groups <- split(1:num_cols, ceiling(seq_along(1:num_cols)/cols_per_df))
num_groups <- ceiling(num_cols / cols_per_df)
```

Figure 8 - Code that split the subset region data frame to groups of 10.

Create empty list and append the smaller data frame extracted from the df_subset into it

```r
smaller_dfs <- list()

for (i in 1:num_groups) {
  start_col <- (i - 1) * 10 + 1
  end_col <- min(i * 10, num_cols)
  small_df <- df_subset[, start_col:end_col, drop = FALSE]
  smaller_dfs[[i]] <- small_df
}
```

Figure 9 - Code that append the smaller data frames extracted into empty list created.

The columns are properly renamed, and the 248 data frames are joined together by rows, thereby converting the data frame with 2480 columns and 1 row to a data frame of 10 columns and 248 rows.

```r
#Rename the columns
common_col_names <- c("TSK", "PSFC", "U10", "V10", "Q2", "RAINC", "RAINNC", "SNOW", "TSLB",
"SMOIS")
```

Figure 10 - Code that properly give the correct names of the columns.

A user-defined function is then created to rename the columns in the data frame.

```r
rename_cols <- function(df, common_names) {
  colnames(df) <- common_names
  return(df)
}
```

Figure 11 – User define function that will be used to rename the columns in the data frames in the list.

This user defined function is then applied to the data frames that has been appended into the list created above.

```
# The defined function is then applied all through the dataframe in the list
smaller_dfs <- lapply(smaller_dfs, rename_cols, common_names = common_col_names)
```

Figure 12 - lapply function that apply the user define function to the data frames in the list.

Then, the properly renamed dataframes are bind together using the code shown.

```
df_region <- do.call(rbind, smaller_dfs)
```

Figure 13 - The code that binds the smaller data frame in the list together by rows.

4.1.5 Creation of Date-Time column

The first record of the data is on 2018-05-01 00:00:00 and subsequent records are done every 3 hours interval. With this information, the date-time is created and its as shown in the code snippet below:

```
#Add datetimestamps
start_timedate <- as.POSIXct("2018-05-01 00:00:00", format = "%Y-%m-%d %H:%M:%S")
end_timedate <- start_timedate + (nrow(df_region)-1) * 3 * 3600
timestamps_seq <- seq(start_timedate, end_timedate, by = "3 hours")

df_region$Datetime <- timestamps_seq
```

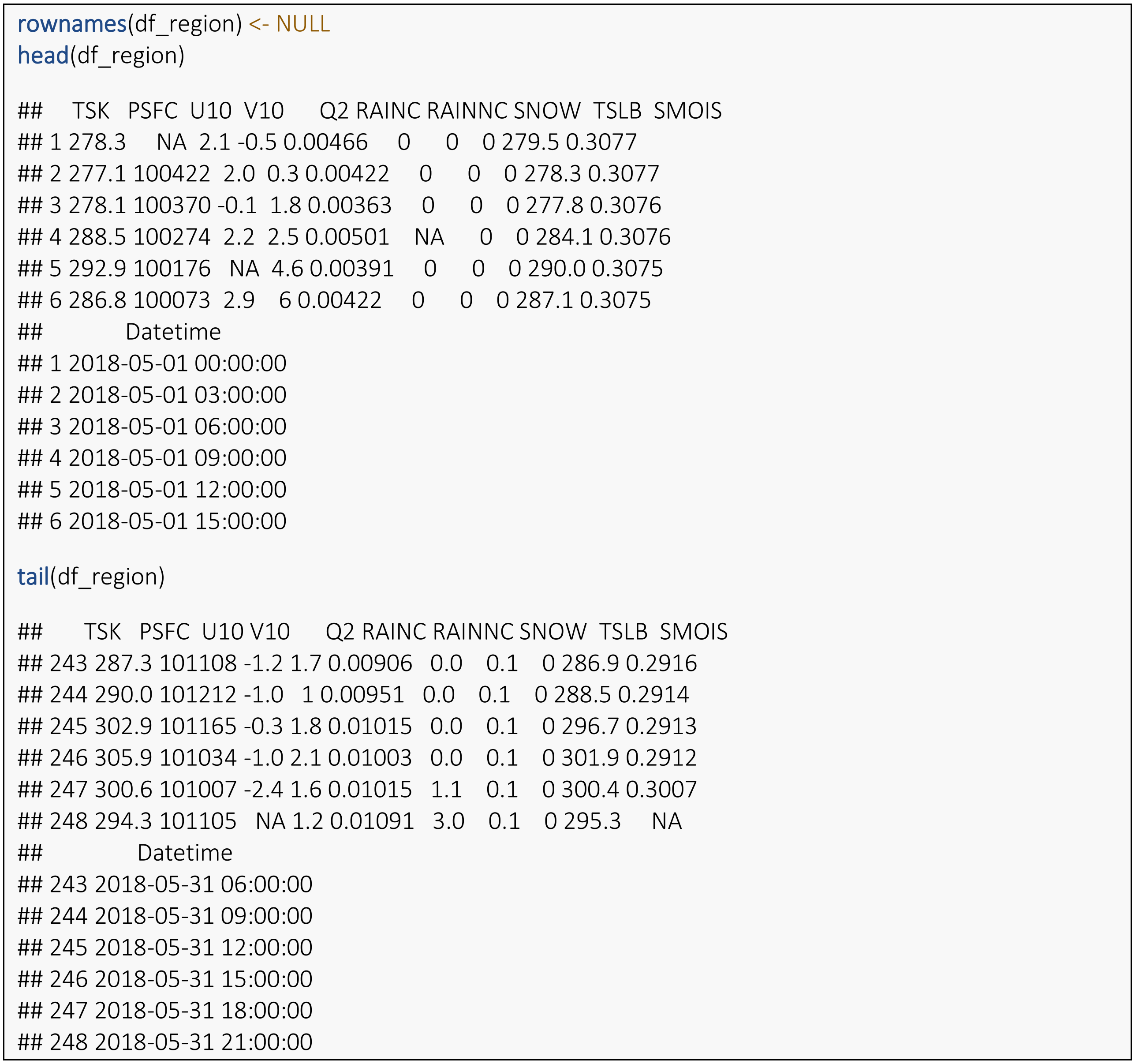
Figure 14 - Code that calculate the date and time and applies it to the data frame binded.

This concludes the data restructuring process as the final data frame to work with now contains 11 columns including the date-time and 248 rows.

```
head(df_region)

##         TSK   PSFC  U10  V10      Q2 RAINC RAINNC SNOW  TSLB  SMOIS
## 3192  278.3     NA  2.1 -0.5 0.00466     0      0    0 279.5 0.3077
## 31921 277.1 100422  2.0  0.3 0.00422     0      0    0 278.3 0.3077
## 31922 278.1 100370 -0.1  1.8 0.00363     0      0    0 277.8 0.3076
## 31923 288.5 100274  2.2  2.5 0.00501    NA      0    0 284.1 0.3076
## 31924 292.9 100176   NA  4.6 0.00391     0      0    0 290.0 0.3075
## 31925 286.8 100073  2.9    6 0.00422     0      0    0 287.1 0.3075
##                  Datetime
## 3192  2018-05-01 00:00:00
## 31921 2018-05-01 03:00:00
## 31922 2018-05-01 06:00:00
## 31923 2018-05-01 09:00:00
## 31924 2018-05-01 12:00:00
## 31925 2018-05-01 15:00:00
```

Figure 15 - Code showing the head of the restructured data frame.

The row names will also be re-index to start from 1 and the final data frame for the selected location is displayed as shown below to finally conclude the data restructuring process and move ahead to data cleaning.



```
rownames(df_region) <- NULL
head(df_region)

##     TSK   PSFC  U10  V10      Q2 RAINC RAINNC SNOW  TSLB  SMOIS
## 1 278.3     NA  2.1 -0.5 0.00466     0      0    0 279.5 0.3077
## 2 277.1 100422  2.0  0.3 0.00422     0      0    0 278.3 0.3077
## 3 278.1 100370 -0.1  1.8 0.00363     0      0    0 277.8 0.3076
## 4 288.5 100274  2.2  2.5 0.00501    NA      0    0 284.1 0.3076
## 5 292.9 100176   NA  4.6 0.00391     0      0    0 290.0 0.3075
## 6 286.8 100073  2.9    6 0.00422     0      0    0 287.1 0.3075
##            Datetime
## 1 2018-05-01 00:00:00
## 2 2018-05-01 03:00:00
## 3 2018-05-01 06:00:00
## 4 2018-05-01 09:00:00
## 5 2018-05-01 12:00:00
## 6 2018-05-01 15:00:00

tail(df_region)

##       TSK   PSFC  U10 V10      Q2 RAINC RAINNC SNOW  TSLB  SMOIS
## 243 287.3 101108 -1.2 1.7 0.00906   0.0    0.1    0 286.9 0.2916
## 244 290.0 101212 -1.0   1 0.00951   0.0    0.1    0 288.5 0.2914
## 245 302.9 101165 -0.3 1.8 0.01015   0.0    0.1    0 296.7 0.2913
## 246 305.9 101034 -1.0 2.1 0.01003   0.0    0.1    0 301.9 0.2912
## 247 300.6 101007 -2.4 1.6 0.01015   1.1    0.1    0 300.4 0.3007
## 248 294.3 101105   NA 1.2 0.01091   3.0    0.1    0 295.3     NA
##              Datetime
## 243 2018-05-31 06:00:00
## 244 2018-05-31 09:00:00
## 245 2018-05-31 12:00:00
## 246 2018-05-31 15:00:00
## 247 2018-05-31 18:00:00
## 248 2018-05-31 21:00:00
```

Figure 16 - Code that re-index the row numbers and shows the head and tail of the data frame.

4.2 Data Cleaning

There are so many reasons why error can find its way into a data, thereby making this step an important one (Buttrey and Whitaker, 2017). The necessary data cleaning process will be done step by step to ensure the dataset is free of errors that can affect the data analysis.

4.2.1 Checking for Duplicate

Duplicate entries can affect the data in a negative way. The dataset is inspected for any duplicate entries. This is achieved through the code shown below:

```
#check for duplicate entry
sum(duplicated(df_region))

## [1] 0
```

Figure 17 - Code that checks for duplicate entry.

There are no duplicate entries as shown in the output, therefore, the data cleaning process can move to the next stage.

4.2.2 Checking for Bad Input

Bad inputs in this data will be string values or special characters. This will affect the analysis if there are present in the data, therefore, it was inspected as shown in the code snippet below.

```
str(df_region)

## 'data.frame':    248 obs. of  11 variables:
##  $ TSK     : num  278 277 278 288 293 ...
##  $ PSFC    : int  NA 100422 100370 100274 100176 100073 99969 99772 99531 99342 ...
##  $ U10     : num  2.1 2 -0.1 2.2 NA 2.9 0.5 -0.6 -1.6 0.5 ...
##  $ V10     : chr  "-0.5" "0.3" "1.8" "2.5" ...
##  $ Q2      : num  0.00466 0.00422 0.00363 0.00501 0.00391 0.00422 0.00456 NA 0.00567 0.00618 ...
##  $ RAINC   : num  0 0 0 NA 0 0 0 0 0 0.2 ...
##  $ RAINNC  : num  0 0 0 0 0 0 NA 0 NA 3.8 ...
##  $ SNOW    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ TSLB    : num  280 278 278 284 290 ...
##  $ SMOIS   : num  0.308 0.308 0.308 0.308 0.307 ...
##  $ Datetime: POSIXct, format: "2018-05-01 00:00:00" "2018-05-01 03:00:00" ...
```

Figure 18 - Code that checks for bad input.

V10 structure is in chr, therefore, a particular value might be recorded as string. It is important to cross-check to ensure there are no special character like /*<,.

```
df_region$V10[str_count(df_region$V10) > 5 & !is.na(df_region$V10)]

## character(0)
```

Figure 19 - Code that checks for character input in V10.

The output shows that there is also no bad input or special character in the column. However, it is essential to convert the V10 column to numeric data type.

```
df_region <- df_region %>%
        mutate(V10 = as.numeric(V10))

str(df_region)

## 'data.frame':   248 obs. of  11 variables:
##  $ TSK     : num  278 277 278 288 293 ...
##  $ PSFC    : int  NA 100422 100370 100274 100176 100073 99969 99772 99531 99342 ...
##  $ U10     : num  2.1 2 -0.1 2.2 NA 2.9 0.5 -0.6 -1.6 0.5 ...
##  $ V10     : num  -0.5 0.3 1.8 2.5 4.6 6 6.1 7.1 6.7 7.8 ...
##  $ Q2      : num  0.00466 0.00422 0.00363 0.00501 0.00391 0.00422 0.00456 NA 0.00567 0.00618 ...
##  $ RAINC   : num  0 0 0 NA 0 0 0 0 0 0.2 ...
##  $ RAINNC  : num  0 0 0 0 0 0 NA 0 NA 3.8 ...
##  $ SNOW    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ TSLB    : num  280 278 278 284 290 ...
##  $ SMOIS   : num  0.308 0.308 0.308 0.308 0.307 ...
##  $ Datetime: POSIXct, format: "2018-05-01 00:00:00" "2018-05-01 03:00:00" ...
```

Figure 20 - Code that converts V10 to numeric and view the data type of each column.

There are no bad inputs now and all the variables are in their correct data type.

4.2.3 Checking for Missing Values

The data collection process can be a rigorous one and therefore makes it inevitable to have some missing values due to human error while collecting the data (Delaporte, Cladière and Camel, 2019). It is vital to ensure that these missing values are inspected and dealt with according to the best possible way. The code snippet to check for the missing values is shown below with its output.

```
sum(is.na(df_region))

## [1] 90
```

Figure 21 - Code that checks the total number of missing values.

There are overall 90 missing values. However, to check the numbers of missing values by column, the code snipper shown below was employed.

```
colSums(is.na(df_region))

##      TSK     PSFC      U10      V10       Q2    RAINC   RAINNC     SNOW
##        0       10       12       11        8       12       12        8
##     TSLB    SMOIS Datetime
##        8        9        0
```

Figure 22 - Code that checks for missing values by column sum.

These numbers of missing values by columns are then plotted into a bar-chart to give a visual view of it.

```
#Visualising these NAs
missingvalue <- colSums(is.na(df_region))
dfmissingvalue <- tibble(variable = names(missingvalue), missing = as.vector(missingvalue))
dfmissingvalue

## # A tibble: 11 × 2
##    variable missing
##    <chr>      <dbl>
##  1 TSK            0
##  2 PSFC          10
##  3 U10           12
##  4 V10           11
##  5 Q2             8
##  6 RAINC         12
##  7 RAINNC        12
##  8 SNOW           8
##  9 TSLB           8
## 10 SMOIS          9
## 11 Datetime       0
```

```
ggplot(data = dfmissingvalue, aes(x = missing, y = variable)) +
  geom_bar(stat = "identity") +
  labs(x = "Missing Values", y = NULL, title = "Bar Chart of Missing Values")
```

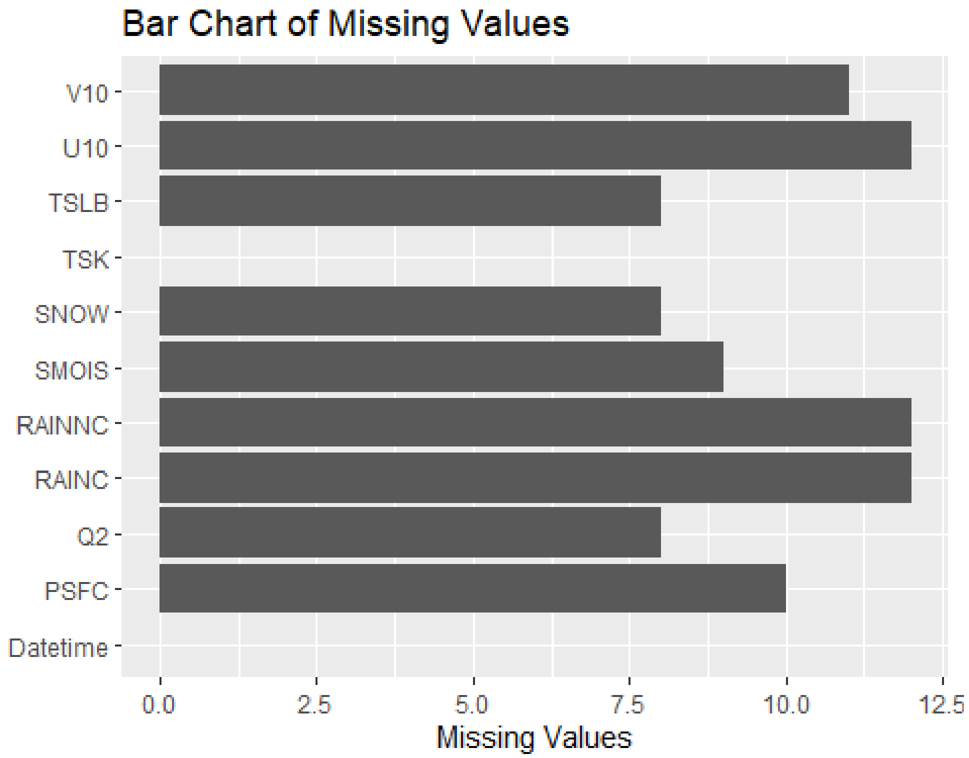


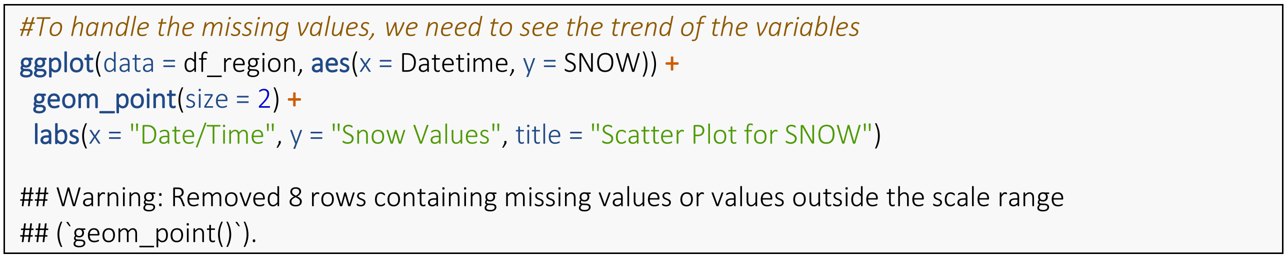
Figure 23 – Bar chart of missing values.

4.2.4 Handling Missing Values

The missing values will be handled by columns. Therefore, each variable will be inspected, visualised to understand the best way to handle them.

TSK: There are 0 missing values in the TSK column as shown in the code and its output above

SNOW: There are 8 missing values in the SNOW column.



```
#To handle the missing values, we need to see the trend of the variables
ggplot(data = df_region, aes(x = Datetime, y = SNOW)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "Snow Values", title = "Scatter Plot for SNOW")

## Warning: Removed 8 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 24 - Code that shows the scatter plot for SNOW.

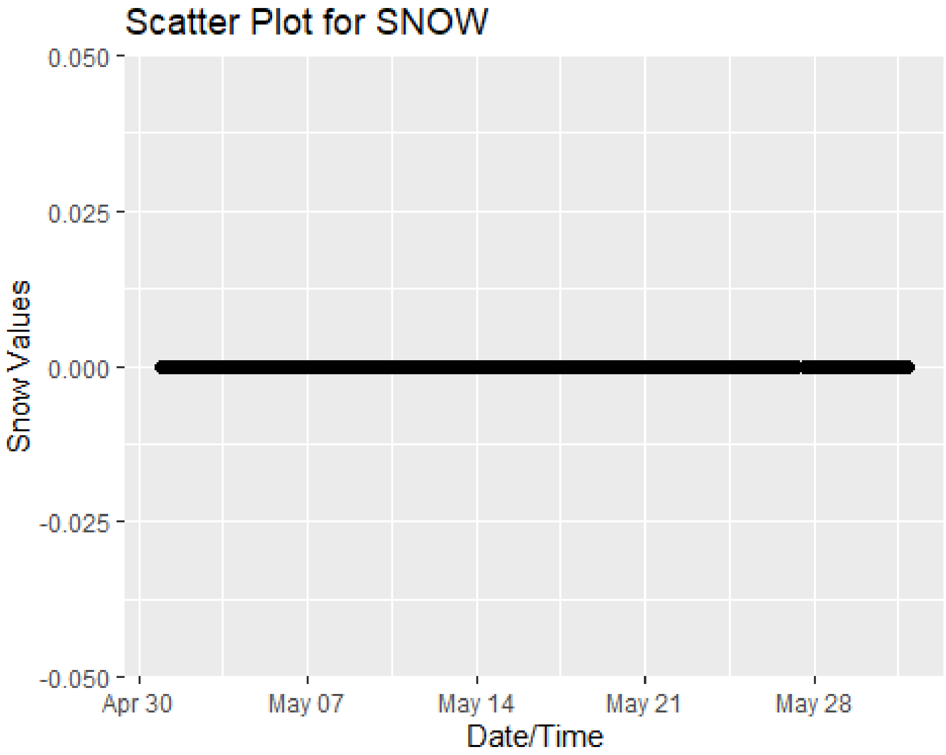
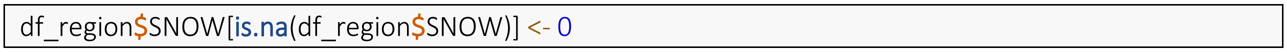


Figure 25 - Scatter plot showing the trend of SNOW.

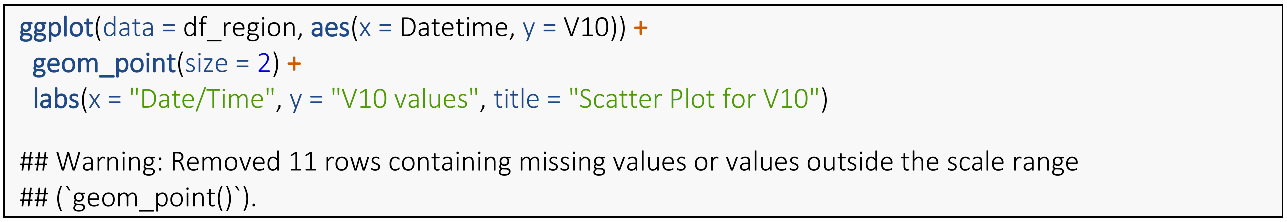
The trend of the values shows that there is no snow as all the values in this column are recorded to be 0. Therefore, the missing values is replaced with 0.



```
df_region$SNOW[is.na(df_region$SNOW)] <- 0
```

Figure 26 - Code that replaces missing value in SNOW.

V10: This is the Y components of wind at 10m, and it has 11 missing values.



```
ggplot(data = df_region, aes(x = Datetime, y = V10)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "V10 values", title = "Scatter Plot for V10")

## Warning: Removed 11 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 27 - Code that views the scatter plot of V10.

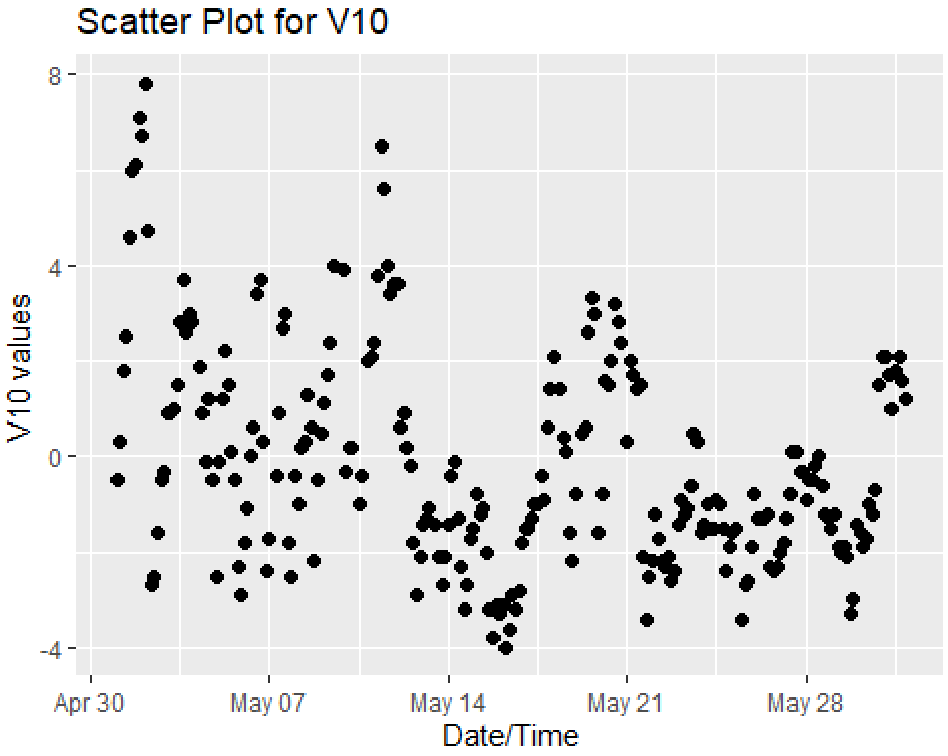


Figure 28 - Scatter plot of Y components of wind at 10m/s.

The trend of V10 shown in scatter plot shows an oscillating trend. Therefore, interpolation in imputeTS method using the spline method is used to fill the missing values because, it is a method that help maintain the shape of the trend by calculating the missing values based on the spline curve.

```
#This shows an oscillating trend
df_region$V10 <- na_interpolation(df_region$V10, option = 'spline')
```

Figure 29 - Code that handles the missing values in V10.

U10: This is the X components of wind at 10m, and it has 12 missing values.

```
ggplot(data = df_region, aes(x = Datetime, y = U10)) +
  geom_point(size = 2, color='blue') +
  labs(x = "Date/Time", y = "U10 values", title = "Scatter Plot for U10")

## Warning: Removed 12 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 30 - Code for scatter plot of U10.

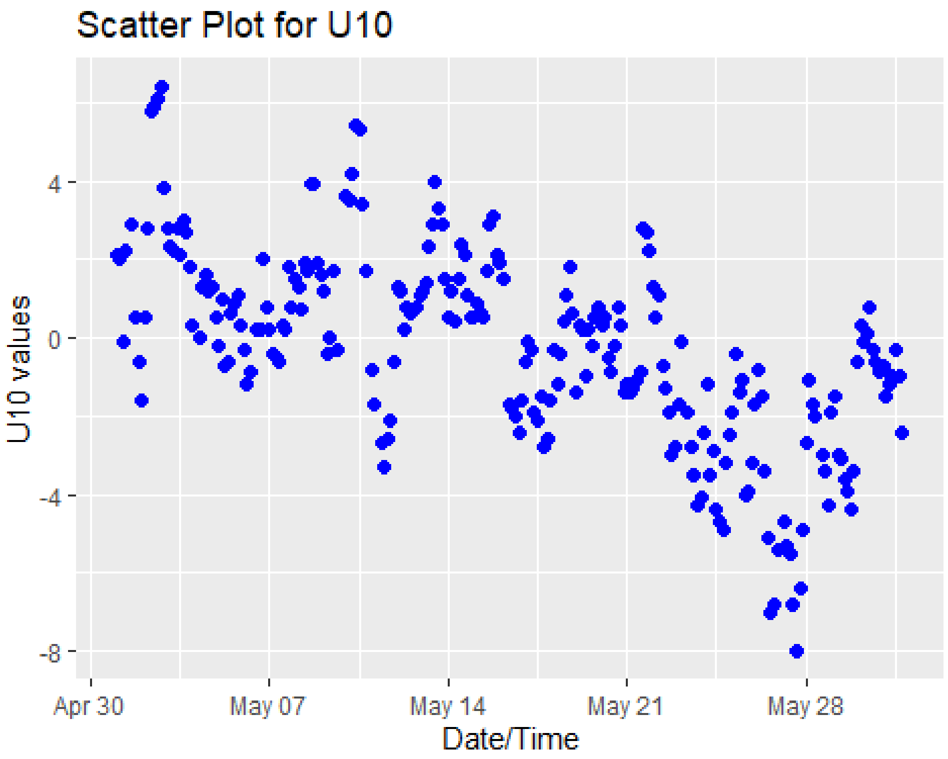
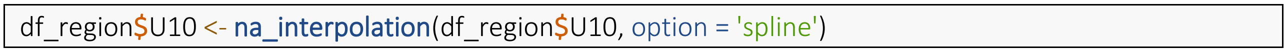


Figure 31 - Scatter plot of X components of wind at 10m/s

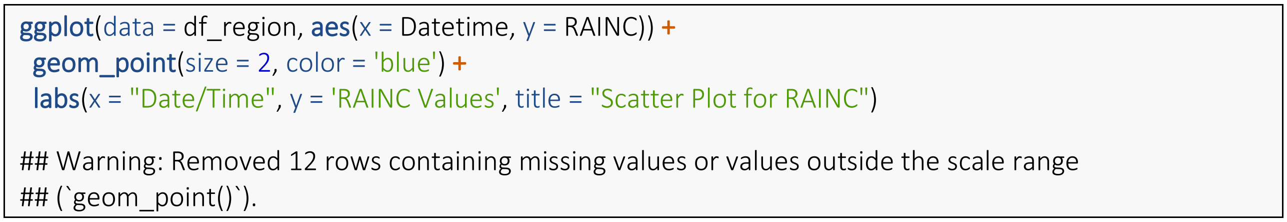
Just like its Y components, it has an oscillating trend, and the missing values are handled the same way as V10 with spline method.



```
df_region$U10 <- na_interpolation(df_region$U10, option = 'spline')
```

Figure 32 - Code that handles the missing values of U10.

RAINC: From the visualisation, RAINC column has linear trends of 0 as its values with few exceptions and a total of 12 missing values.



```
ggplot(data = df_region, aes(x = Datetime, y = RAINC)) +
  geom_point(size = 2, color = 'blue') +
  labs(x = "Date/Time", y = 'RAINC Values', title = "Scatter Plot for RAINC")

## Warning: Removed 12 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 33 - Code that views the trend of RAINC.

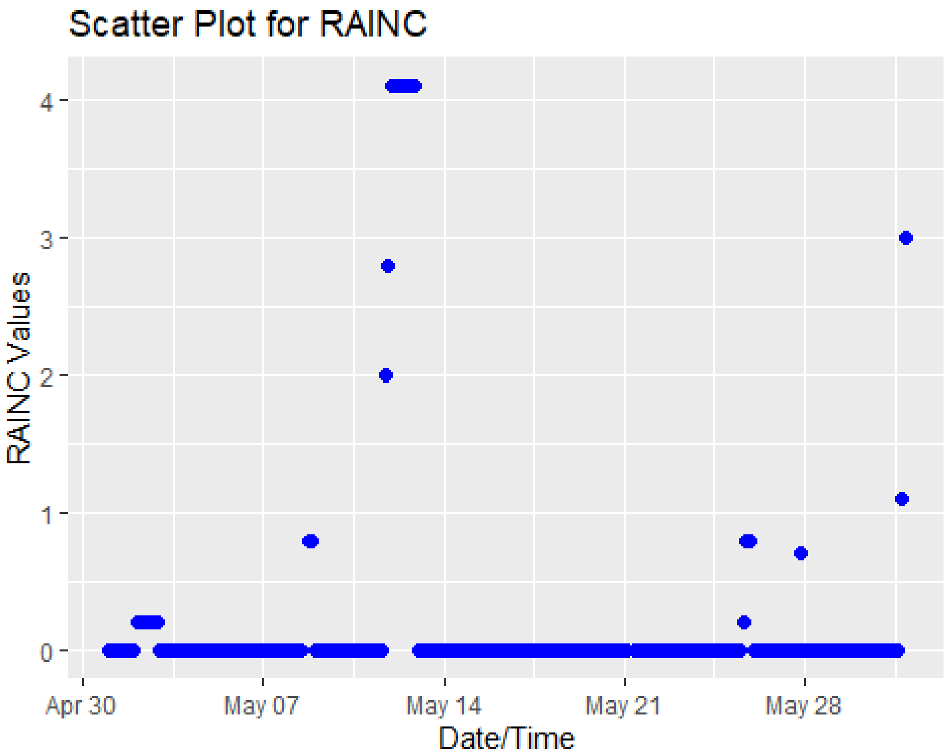
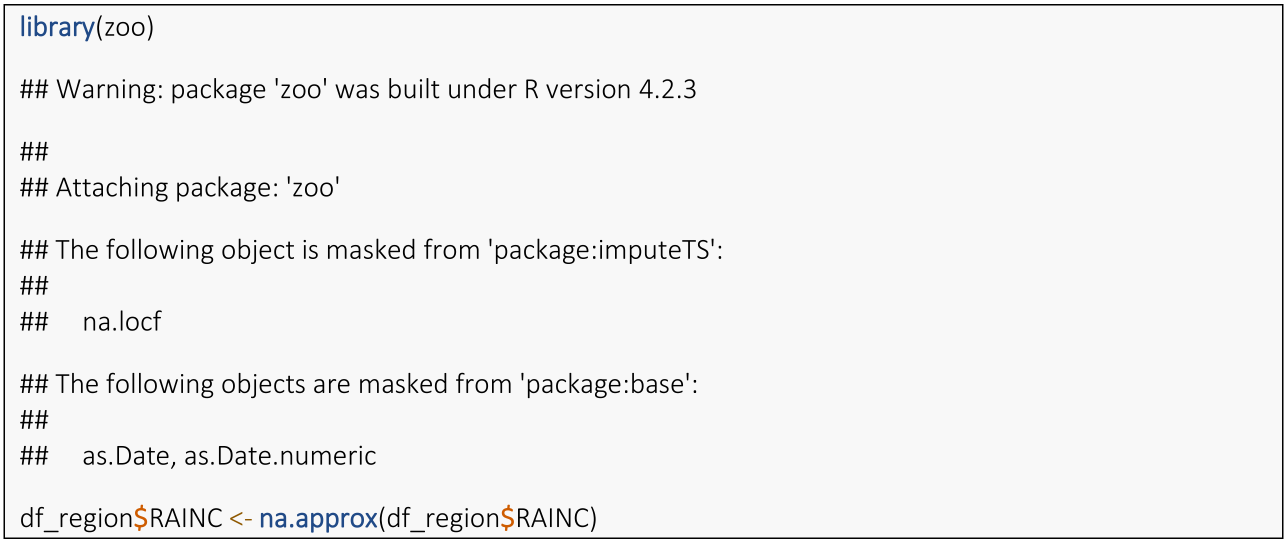


Figure 34 - Scatter plot of convectional rain (RAINC).

It is vital that this trend is preserved, therefore, approximate method is employed to help fill the missing values. The approximate takes the average of neighbouring data of the missing values.

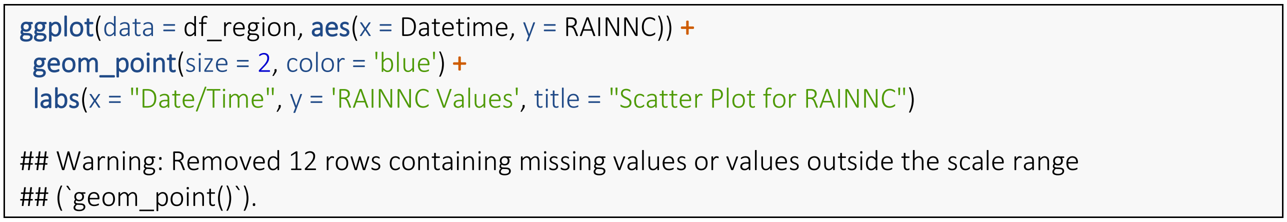
This method is employed to preserve the trend and ensure the missing values were not replaced with a value that is far off.



```
library(zoo)

## Warning: package 'zoo' was built under R version 4.2.3

##
## Attaching package: 'zoo'

## The following object is masked from 'package:imputeTS':
##
##     na.locf

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

df_region$RAINC <- na.approx(df_region$RAINC)
```

Figure 35 - Code that activate zoo library and handles missing values in RAINC.

RAINNC: This also follows the same trend with RAINC, and the missing values were handled the same way as above.



```
ggplot(data = df_region, aes(x = Datetime, y = RAINNC)) +
  geom_point(size = 2, color = 'blue') +
  labs(x = "Date/Time", y = 'RAINNC Values', title = "Scatter Plot for RAINNC")

## Warning: Removed 12 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 36 - Code that views the scatter plot of RAINNC.

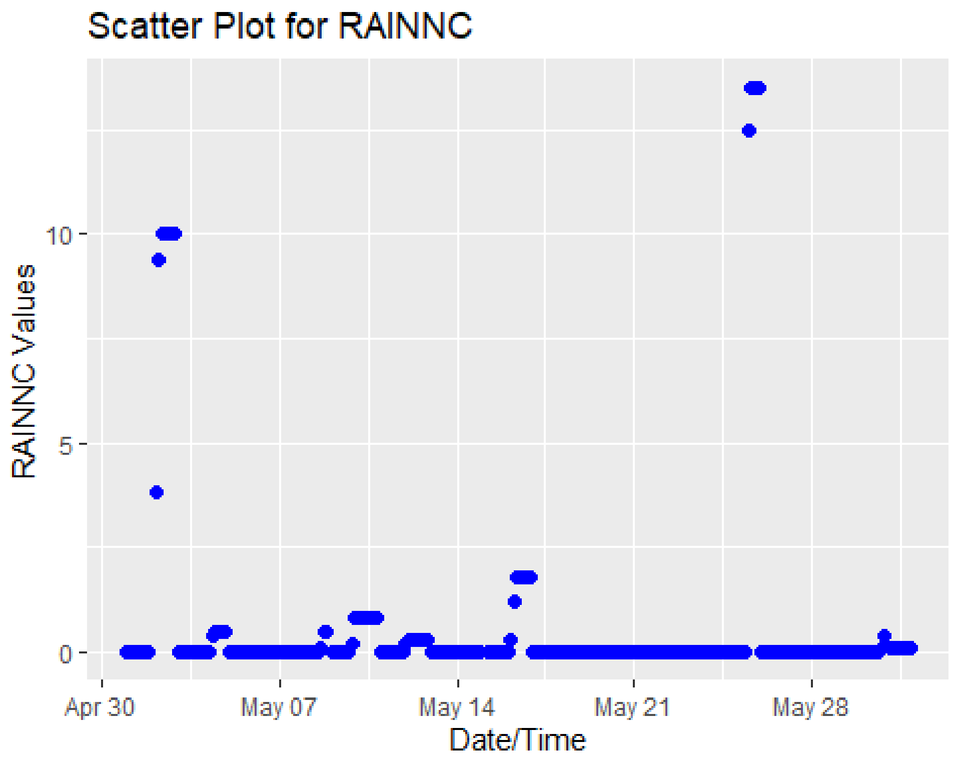


Figure 37 - Scatter plot for non-convectional rain (RAINNC).

```
df_region$RAINNC <- na.approx(df_region$RAINNC)
```

Figure 38 - Code that handles the missing values in RAINNC.

Q2: The Q2 also has spiral trend and to protect the trend, the interpolation using spline method was also used to fill in the missing values.

```
ggplot(data = df_region, aes(x = Datetime, y = Q2)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "Q2 Values", title = "Scatter Plot for Q2")

## Warning: Removed 8 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 39 - Code for scatter plot of Q2.

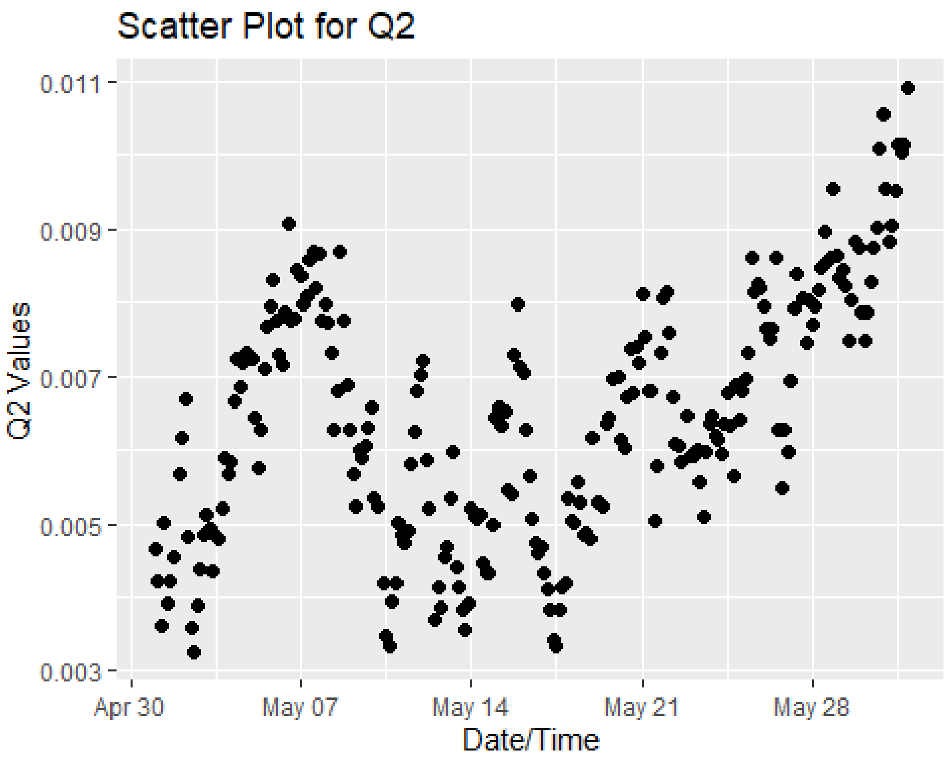


Figure 40 - Scatter plot showing the trend of Specific Humidity (Q2)

```
#Spline interpolation will also be used
df_region$Q2 <- na_interpolation(df_region$Q2, option = 'spline')
```

Figure 41 - Code that handle missing values in Q2.

TSLB: From the visualisation, the variables show upward trend however, the points are scattered.

```
ggplot(data = df_region, aes(x = Datetime, y = TSLB)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "TSLB Values", title = "Scatter Plot for TSLB")

## Warning: Removed 8 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 42 - Code that visualise the scatter plot of TSLB.

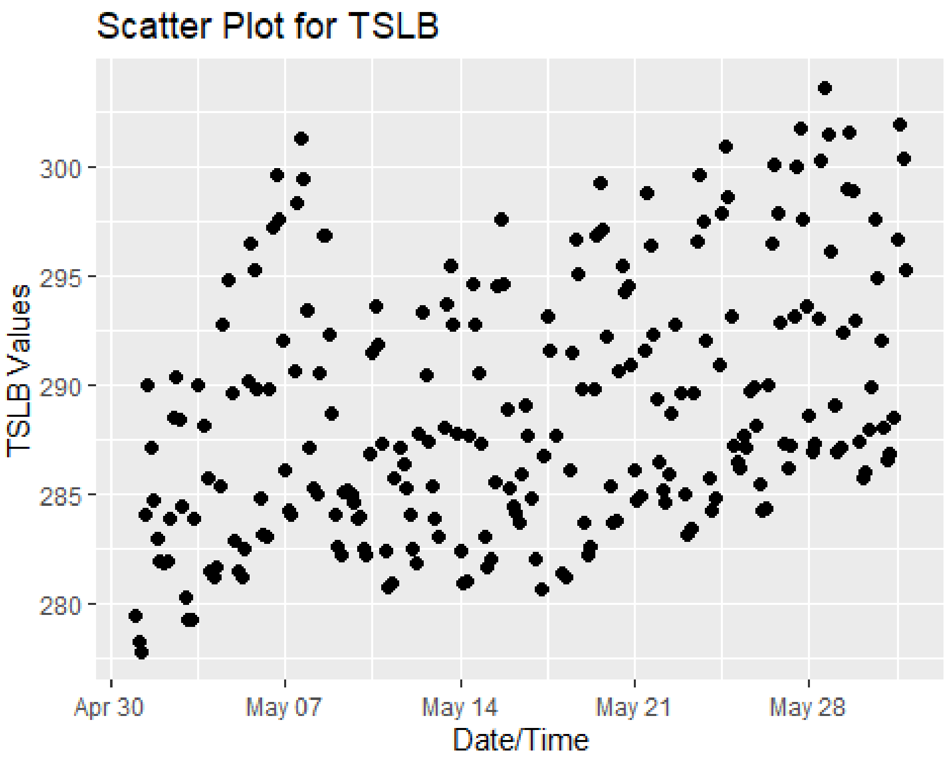


Figure 43 - Scatter plot of soil temperature (TSLB)

It is better to replace the missing values using the approximation method. This will replace missing values with the average of the neighbouring values, thereby ensuring the values do not go out of the range.

```
df_region$TSLB <- na.approx(df_region$TSLB)
```

Figure 44 - Code to handle the missing values of TSLB.

SMOIS: The trend of SMOIS shows downward trend with few upward trends.

```
ggplot(data = df_region, aes(x = Datetime, y = SMOIS)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "SMOIS values", title = "Scatter Plot for SMOIS")

## Warning: Removed 9 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 45 - Code for scatter plot of SMOIS.

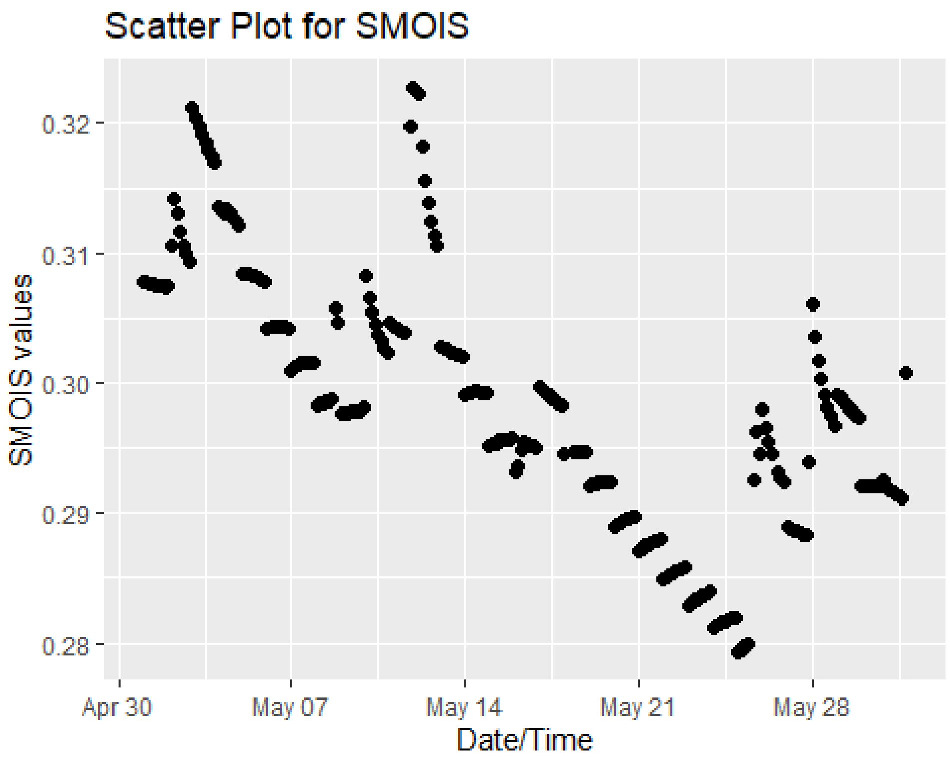
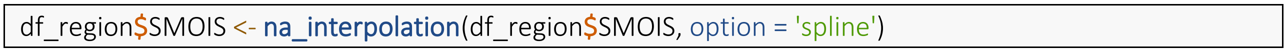


Figure 46 - Trend of soil moisture using scatter plot.

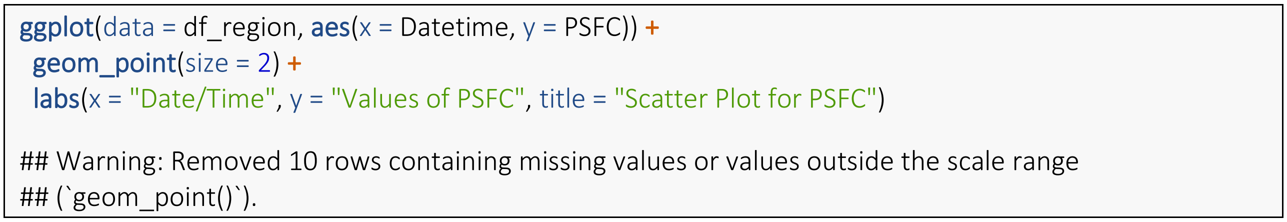
Using spline method will be perfect for these missing values to ensure the trend is maintained.



```
df_region$SMOIS <- na_interpolation(df_region$SMOIS, option = 'spline')
```

Figure 47 - Code that handles the missing values in SMOIS.

PSFC: This is a spiral trend, and the missing values are replaced by spline method of the interpolation function to preserve the shape of the trend.



```
ggplot(data = df_region, aes(x = Datetime, y = PSFC)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "Values of PSFC", title = "Scatter Plot for PSFC")

## Warning: Removed 10 rows containing missing values or values outside the scale range
## (`geom_point()`).
```

Figure 48 – Code for scatter plot for PSFC.

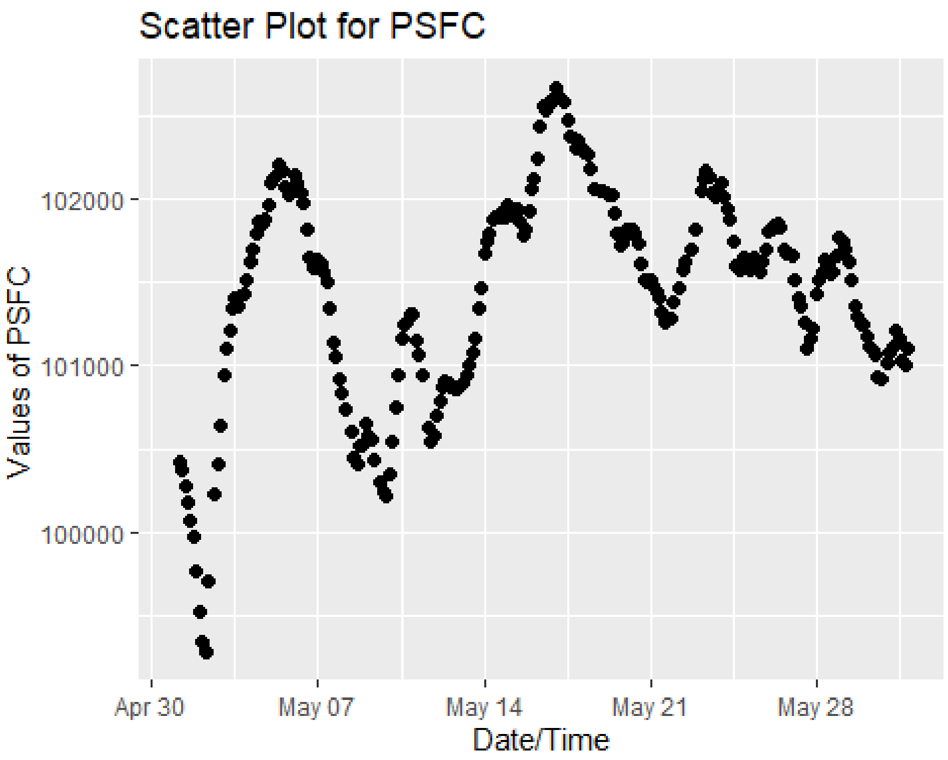


Figure 49 - Trend of Pressure using scatter plot.

```
df_region$PSFC <- na_interpolation(df_region$PSFC, option = 'spline')
```

Figure 50 - Code for handling the missing values of PSFC.

The missing values have been handled and the confirmation to this is shown below as there are now zero missing values in the data frame.

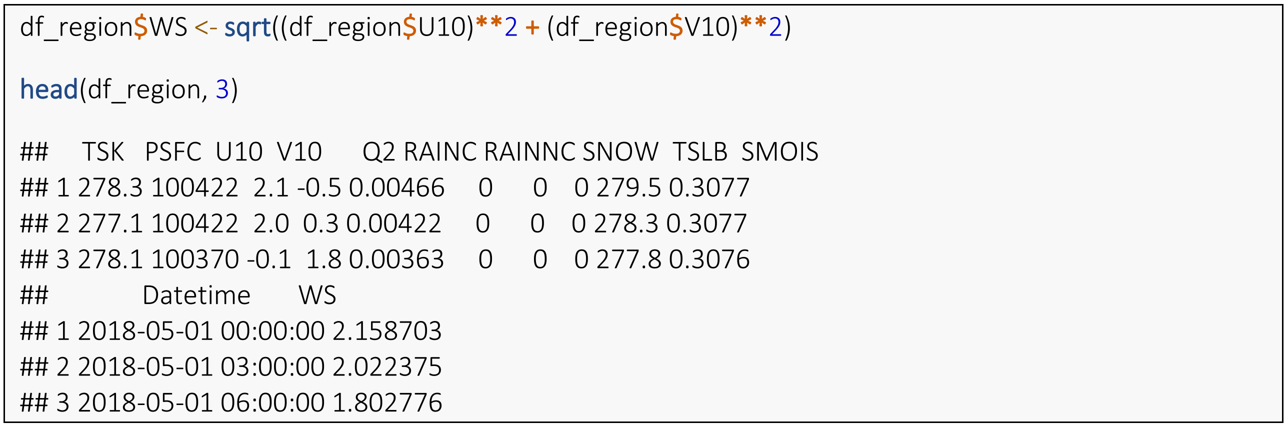
```
sum(is.na(df_region))

## [1] 0

colSums(is.na(df_region))

##      TSK     PSFC      U10      V10       Q2    RAINC   RAINNC     SNOW
##        0        0        0        0        0        0        0        0
##     TSLB    SMOIS Datetime
##        0        0        0
```

Figure 51 - Code showing there are no more missing values.

4.2.5 Creating the Windspeed Column and Calculating the Values

The wind speed is calculated from the V10 and U10 column with the formular sqrt(V102 + U102).

The column creation and the calculation is shown in the code snipper below.



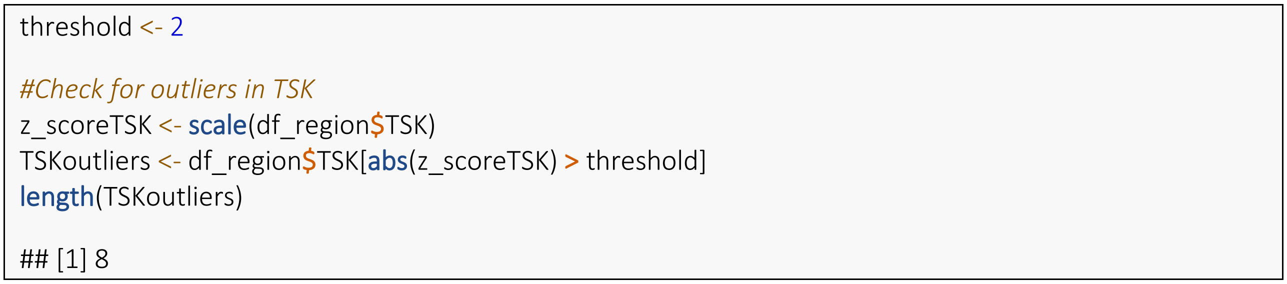
```
df_region$WS <- sqrt((df_region$U10)**2 + (df_region$V10)**2)

head(df_region, 3)

##     TSK   PSFC  U10  V10      Q2 RAINC RAINNC SNOW  TSLB  SMOIS
## 1 278.3 100422  2.1 -0.5 0.00466     0      0    0 279.5 0.3077
## 2 277.1 100422  2.0  0.3 0.00422     0      0    0 278.3 0.3077
## 3 278.1 100370 -0.1  1.8 0.00363     0      0    0 277.8 0.3076
##              Datetime       WS
## 1 2018-05-01 00:00:00 2.158703
## 2 2018-05-01 03:00:00 2.022375
## 3 2018-05-01 06:00:00 1.802776
```

Figure 52 - Code that create the wind speed column.

4.2.6 Detecting and Handling Outliers

To detect outliers, three methods will be used. The z_score method, Boxplot, and the Histogram.

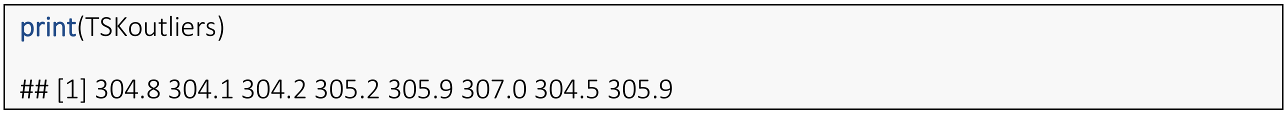
TSK: With z-score method, there are 8 outliers as shown in the output of the code below.



```
threshold <- 2

#Check for outliers in TSK
z_scoreTSK <- scale(df_region$TSK)
TSKoutliers <- df_region$TSK[abs(z_scoreTSK) > threshold]
length(TSKoutliers)

## [1] 8
```

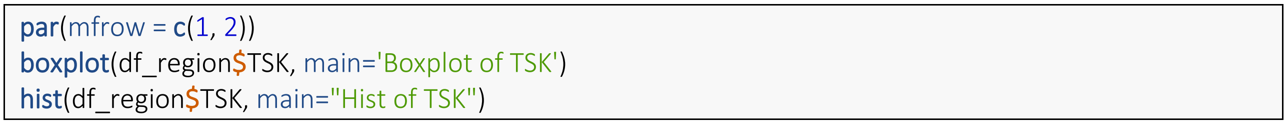
Figure 53 - Code for z-score to check for TSK outliers.

To view these outliers, the following code is written.



```
print(TSKoutliers)

## [1] 304.8 304.1 304.2 305.2 305.9 307.0 304.5 305.9
```

However, there is no outliers when the histogram and boxplot of the column was visualised.



```
par(mfrow = c(1, 2))
boxplot(df_region$TSK, main='Boxplot of TSK')
hist(df_region$TSK, main="Hist of TSK")
```

Figure 54 - Code to view the boxplot and histogram of TSK.

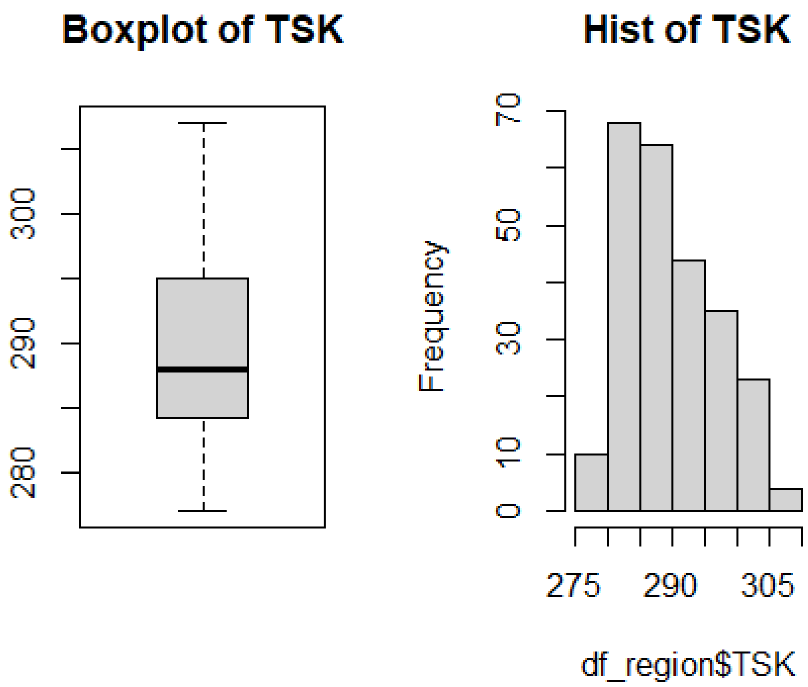
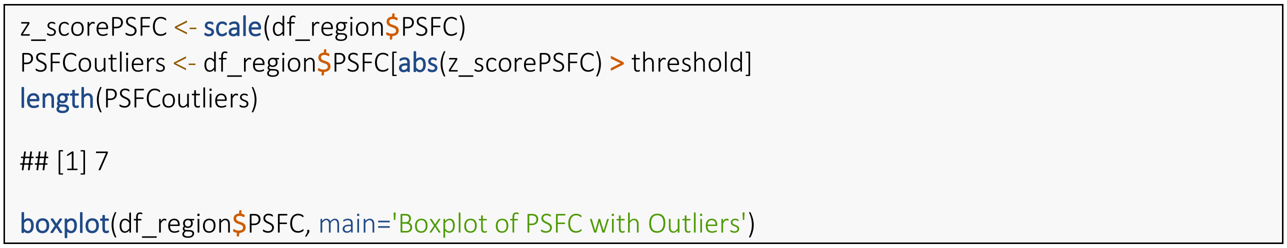


Figure 55 - Histogram and Boxplot of Surace Temperature

The values of these outliers as stated using the z-score methods are 304.8, 304.1, 304.2, 305.2, 305.9, 307.0, 304.5, 305.9 which when compared to the remaining values are not out of order, therefore, the values will be retained.

PSFC: The box plot graph shows that there are indeed outliers, and the z-score method shows there are 7 outliers.



```
z_scorePSFC <- scale(df_region$PSFC)
PSFCoutliers <- df_region$PSFC[abs(z_scorePSFC) > threshold]
length(PSFCoutliers)

## [1] 7

boxplot(df_region$PSFC, main='Boxplot of PSFC with Outliers')
```

Figure 56 - Code for z-score method to view PSFC outliers.

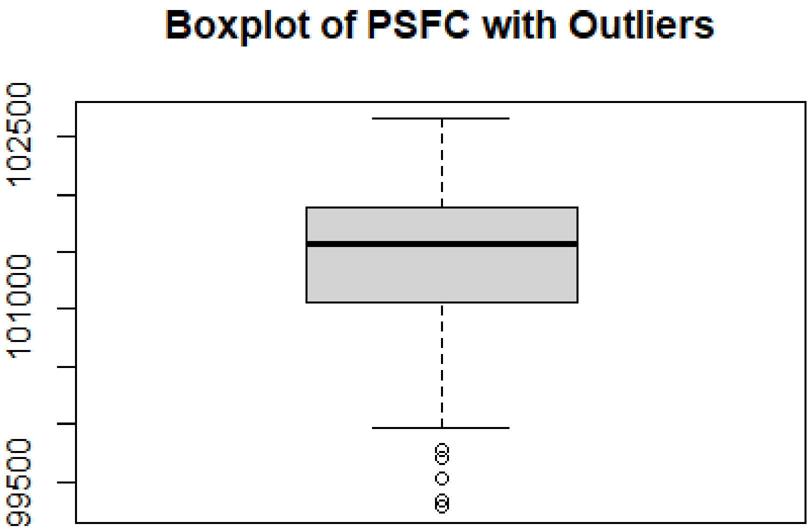
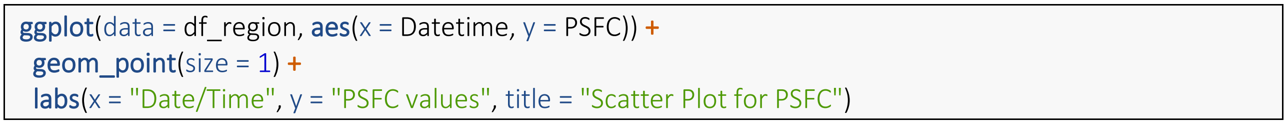


Figure 57 - Boxplot of PSFC

To handle the outlier, it is vital that the trend of the PSFC is viewed to know the appropriate method to use in replacing the outliers.



```
ggplot(data = df_region, aes(x = Datetime, y = PSFC)) +
 geom_point(size = 1) +
 labs(x = "Date/Time", y = "PSFC values", title = "Scatter Plot for PSFC")
```

Figure 58 - Scatter plot code to view outliers in PSFC.

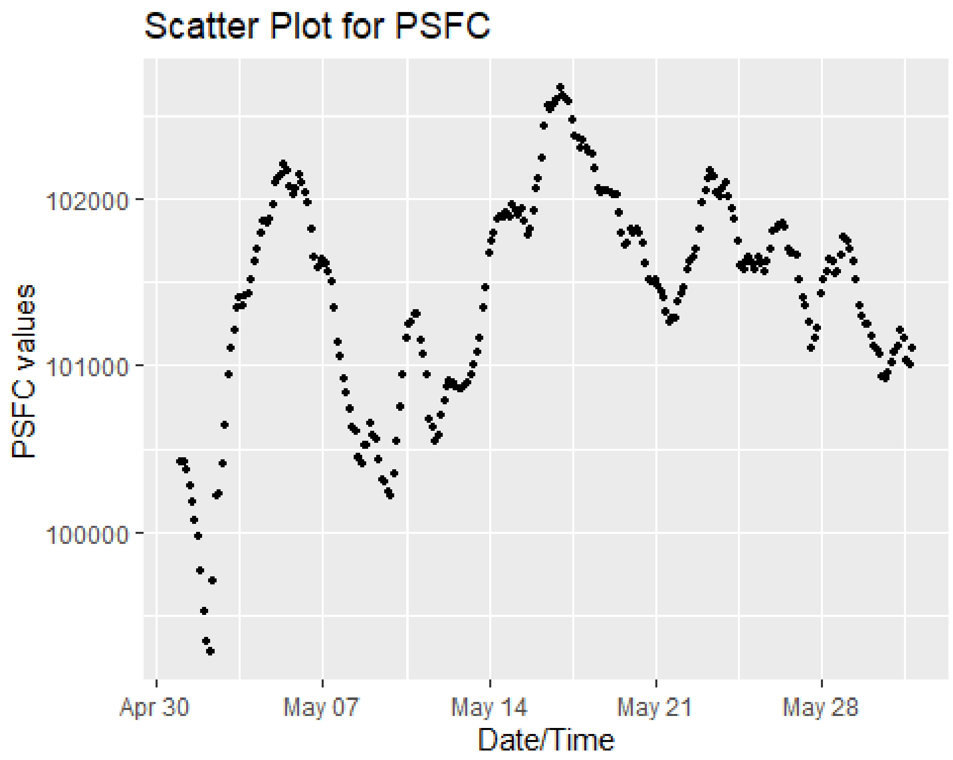


Figure 59 - Scatter plot of PSFC.

Visualising the trend of PSFC, it is shown that the outliers do not follow the trend of the remaining values, therefore, the outliers will be replaced using interpolation with spline method. This is done to retain the shape of the trend of variable.

```
#Replace the outliers with NA
df_region$PSFC <- ifelse(df_region$PSFC %in% PSFCoutliers, NA, df_region$PSFC)
df_region$PSFC <- na_interpolation(df_region$PSFC, option = 'spline')
```

Figure 60 - Code replacing the outlier values in PSFC.

The boxplot after the outliers is handled shows that the variables are now within the acceptable range.

```
boxplot(df_region$PSFC, main='Boxplot of PSFC after outlier handling')
```

Figure 61 - Boxplot code of PSFC after outlier handling.

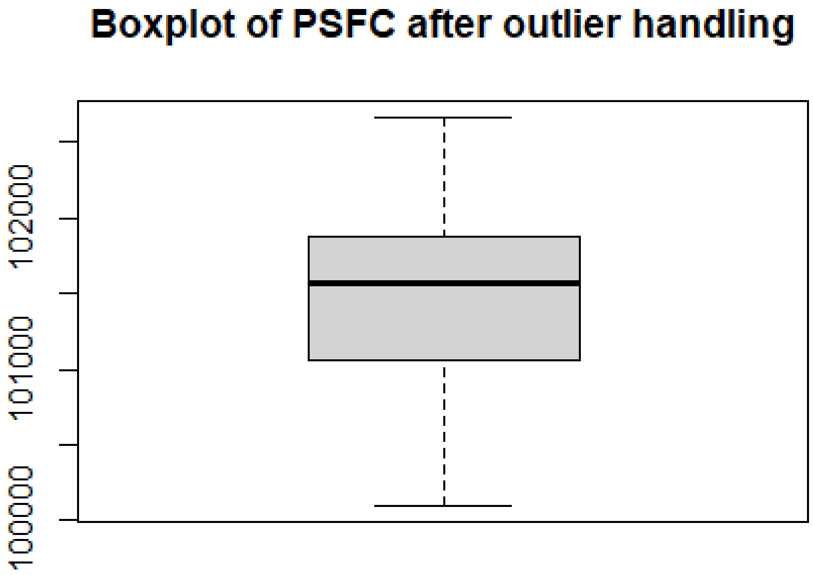
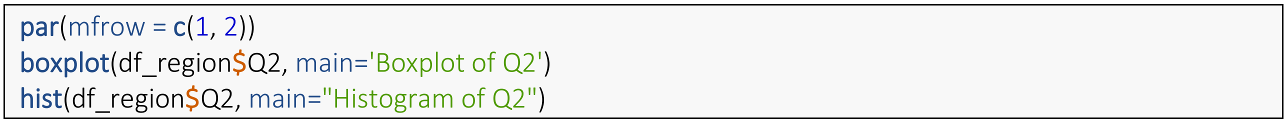


Figure 62 - Boxplot of PSFC after outlier handling.

Q2: The boxplot and histogram visualisation shows that there are no outliers. However, the z-score method reveals that there are 6.



```
par(mfrow = c(1, 2))
boxplot(df_region$Q2, main='Boxplot of Q2')
hist(df_region$Q2, main="Histogram of Q2")
```

Figure 63 - Code for boxplot and histogram of Q2.

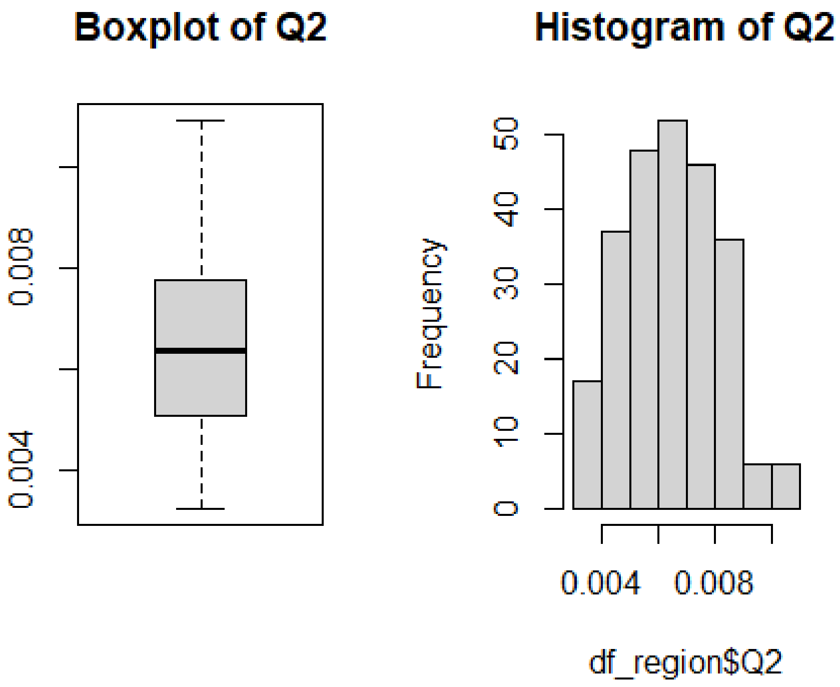
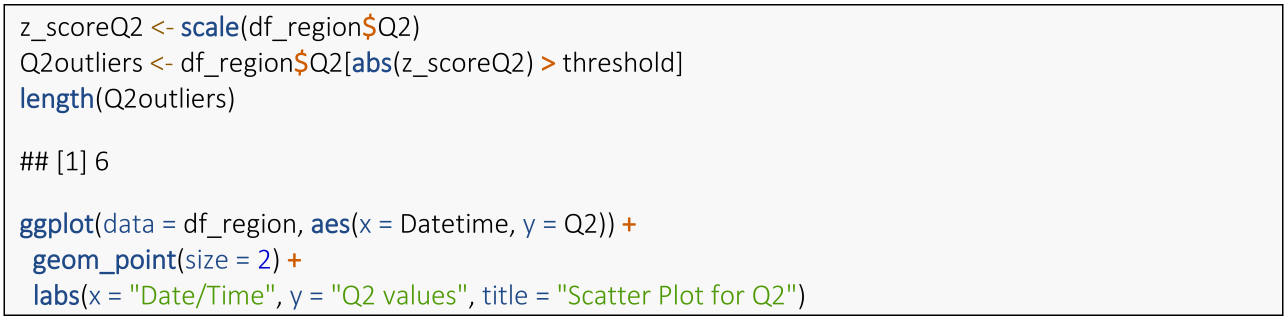


Figure 64 - Boxplot and histogram of specific humidity (Q2)



```
z_scoreQ2 <- scale(df_region$Q2)
Q2outliers <- df_region$Q2[abs(z_scoreQ2) > threshold]
length(Q2outliers)

## [1] 6

ggplot(data = df_region, aes(x = Datetime, y = Q2)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "Q2 values", title = "Scatter Plot for Q2")
```

Figure 65 - Code for z-score and scatter plot of Q2.

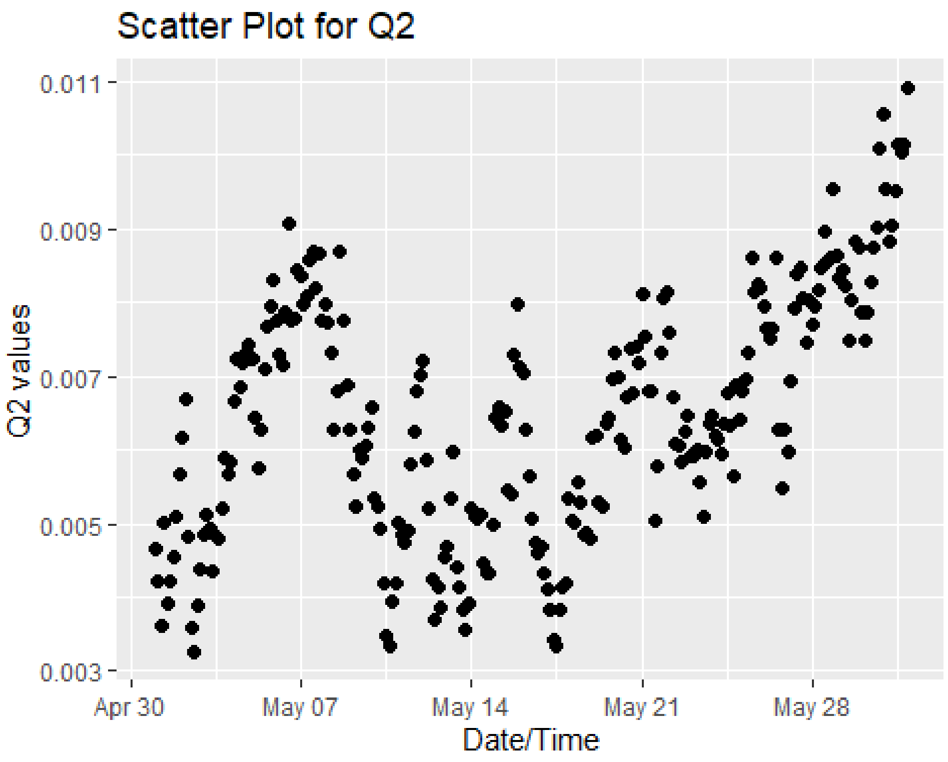
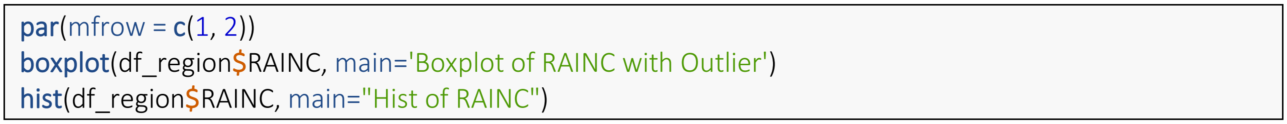


Figure 66 - Q2 scatter plot.

Visualising the trend to view these outliers are not that visible and cannot be regarded as one, therefore, the verdict of boxplot and histogram showing there are no outliers is accepted and the 6 values revealed from the z-score method are retained.

RAINC: The boxplot and histogram show there are outliers, however, it is vital to inspect this column to ensure they are really outliers.



```
par(mfrow = c(1, 2))
boxplot(df_region$RAINC, main='Boxplot of RAINC with Outlier')
hist(df_region$RAINC, main="Hist of RAINC")
```

Figure 67 - Code for boxplot and histogram of RAINC.

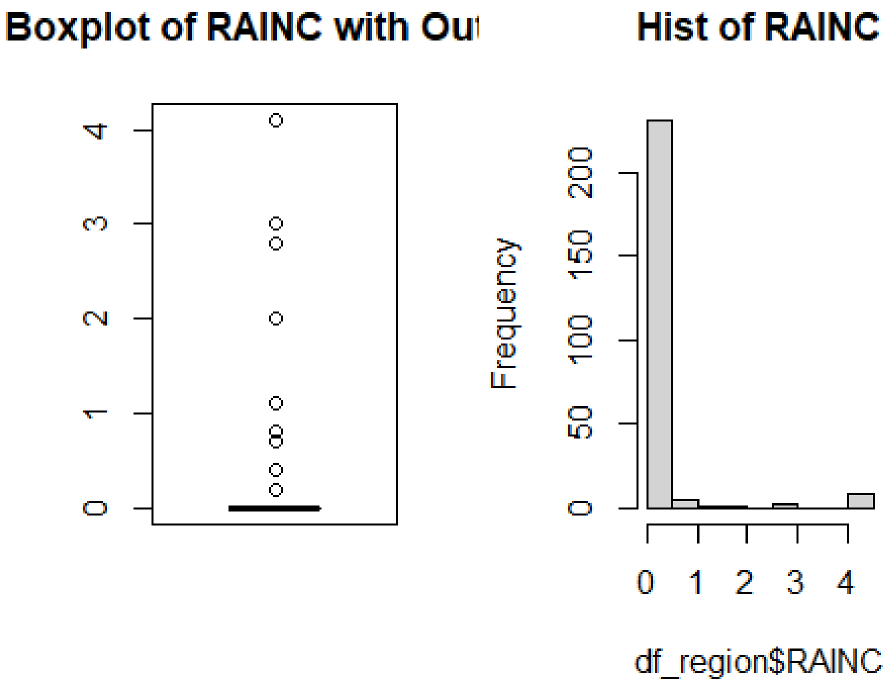
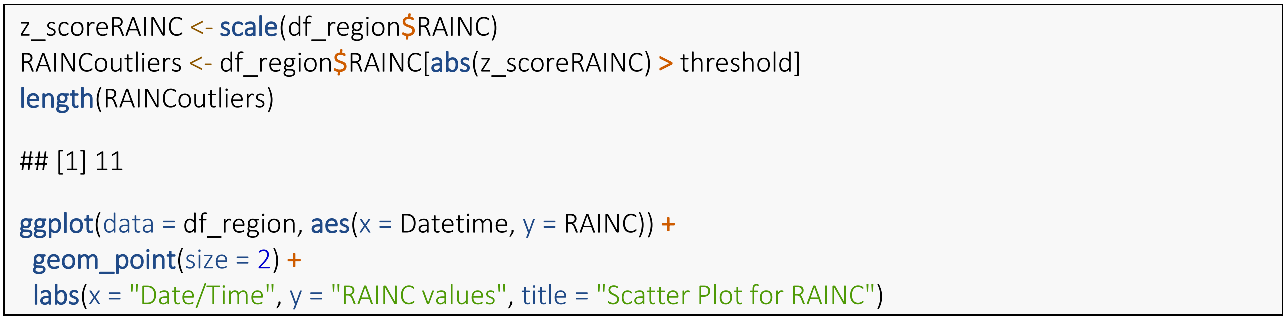


Figure 68 - Boxplot and Histogram of RAINC



```
z_scoreRAINC <- scale(df_region$RAINC)
RAINCoutliers <- df_region$RAINC[abs(z_scoreRAINC) > threshold]
length(RAINCoutliers)

## [1] 11

ggplot(data = df_region, aes(x = Datetime, y = RAINC)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "RAINC values", title = "Scatter Plot for RAINC")
```

Figure 69 - Code for z-score and scatter plot of RAINC.

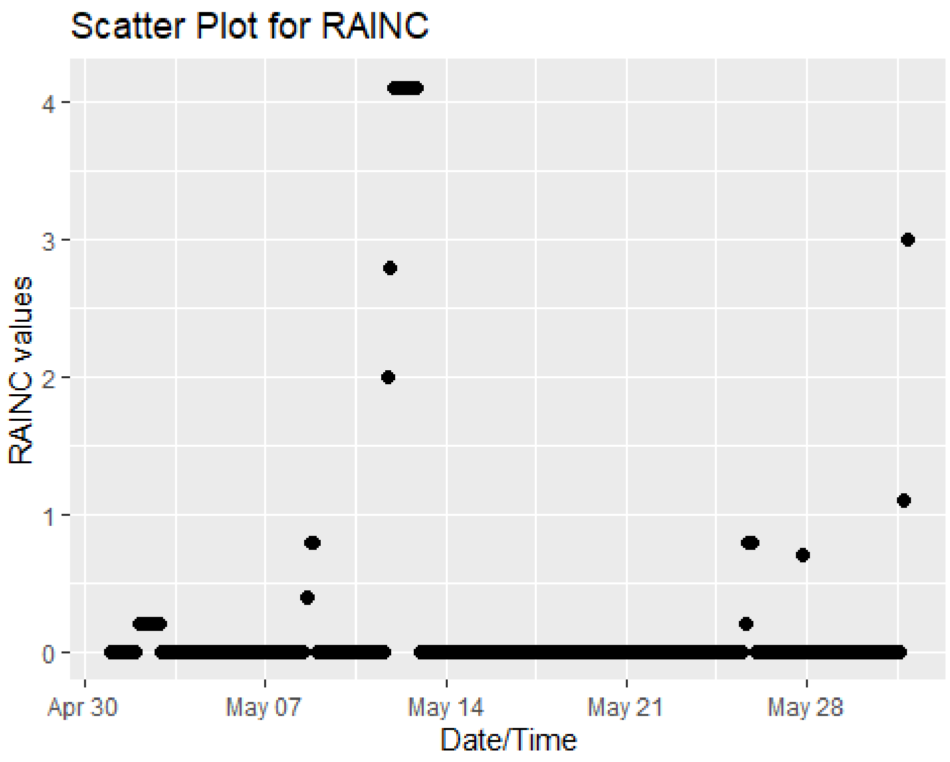
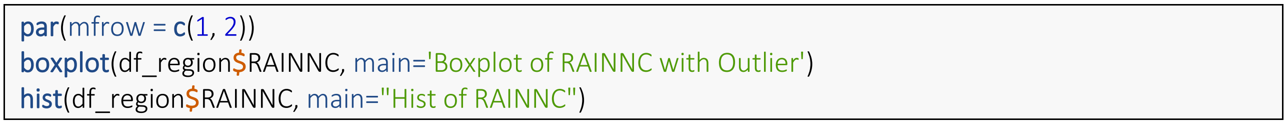


Figure 70 - Scatter plot of RAINC.

The trend shows that there is no precipitation for most of the days in the region and the exception of few days of precipitation, though shows as outlier but that does not make them outliers, therefore, the values will be retained.

RAINNC: This variable also shows the same characteristics as RAINC as shown in the boxplot, histogram, and scatter plot.



```
par(mfrow = c(1, 2))
boxplot(df_region$RAINNC, main='Boxplot of RAINNC with Outlier')
hist(df_region$RAINNC, main="Hist of RAINNC")
```

Figure 71 - Code for boxplot and histogram of RAINNC.

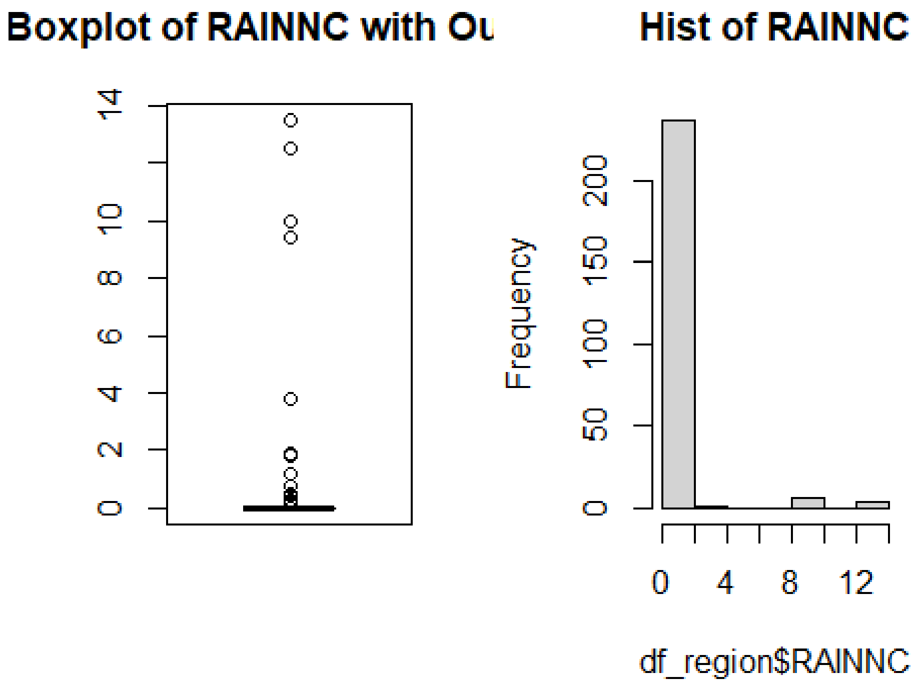
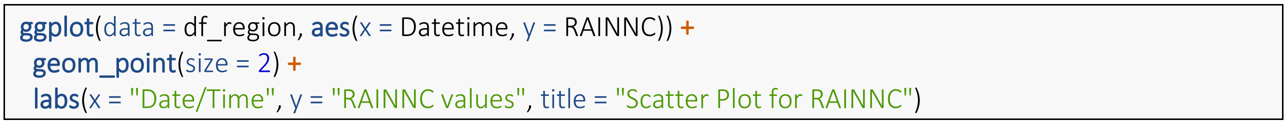


Figure 72 - Boxplot and Histogram of RAINNC



```
ggplot(data = df_region, aes(x = Datetime, y = RAINNC)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "RAINNC values", title = "Scatter Plot for RAINNC")
```

Figure 73 - Code for RAINC Scatter Plot.

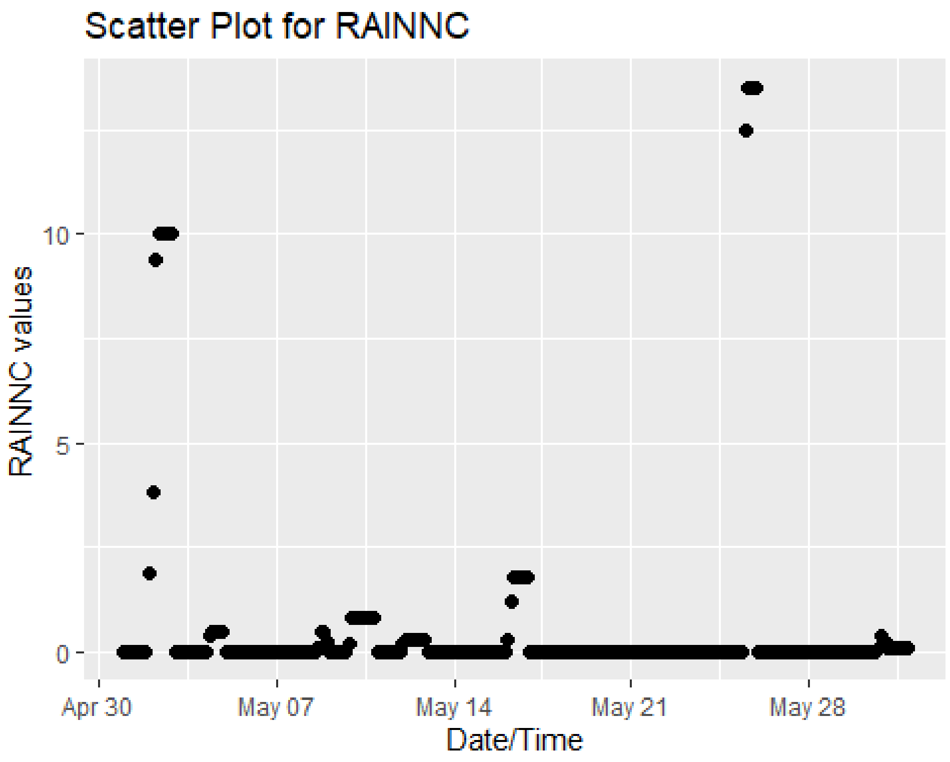


Figure 74 - Scatter plot of RAINNC.

Therefore, the values will also be retained.

TSLB: This variable also shows the same characteristics as the TSK variable.

```
par(mfrow = c(1, 2))
boxplot(df_region$TSLB, main='Boxplot of TSLB')
hist(df_region$TSLB, main="Hist of TSLB")
```

Figure 75 - Code to view the histogram and boxplot of TSLB.

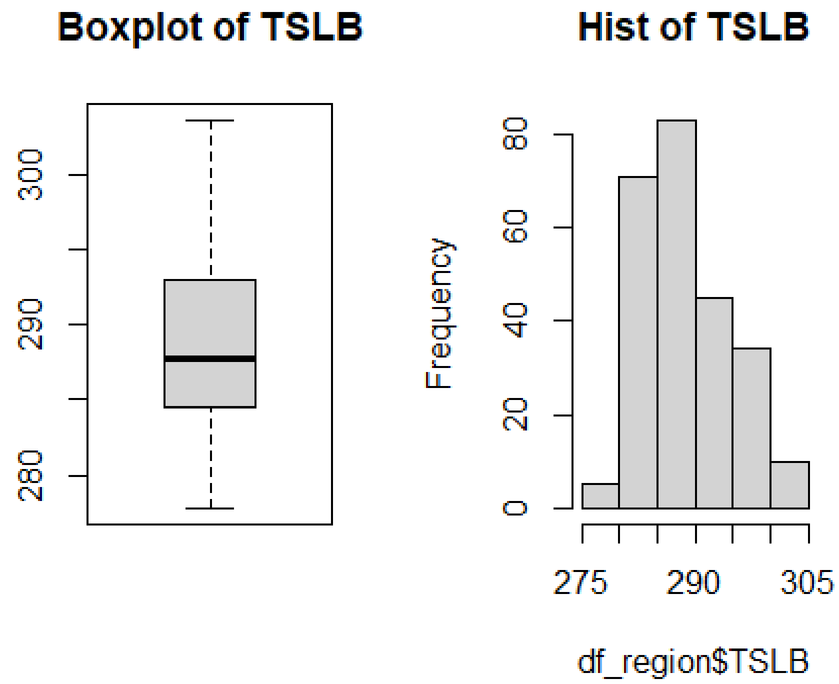
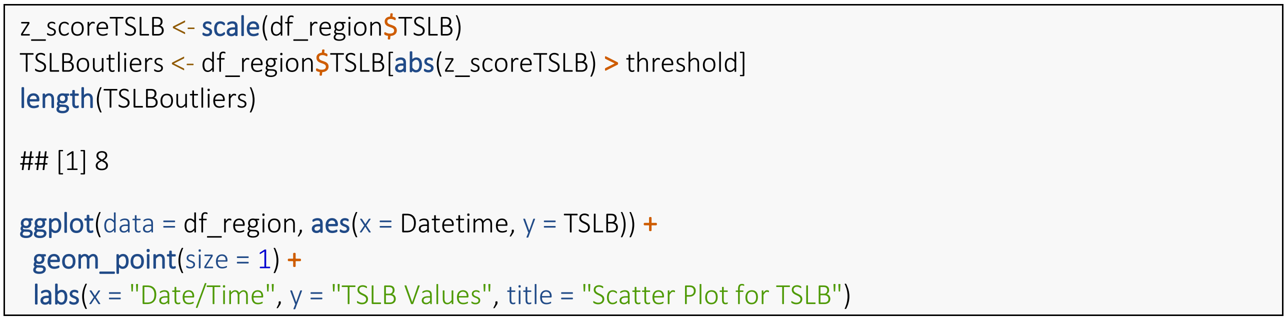


Figure 76 - Boxplot and Histogram of Soil Temperature (TSLB)

The boxplot and histogram do not show any presence of outlier; however, the z-score shows that there are 7 outliers.



```
z_scoreTSLB <- scale(df_region$TSLB)
TSLBoutliers <- df_region$TSLB[abs(z_scoreTSLB) > threshold]
length(TSLBoutliers)

## [1] 8

ggplot(data = df_region, aes(x = Datetime, y = TSLB)) +
  geom_point(size = 1) +
  labs(x = "Date/Time", y = "TSLB Values", title = "Scatter Plot for TSLB")
```

Figure 77 - Code for z-score and scatter plot of TSLB.

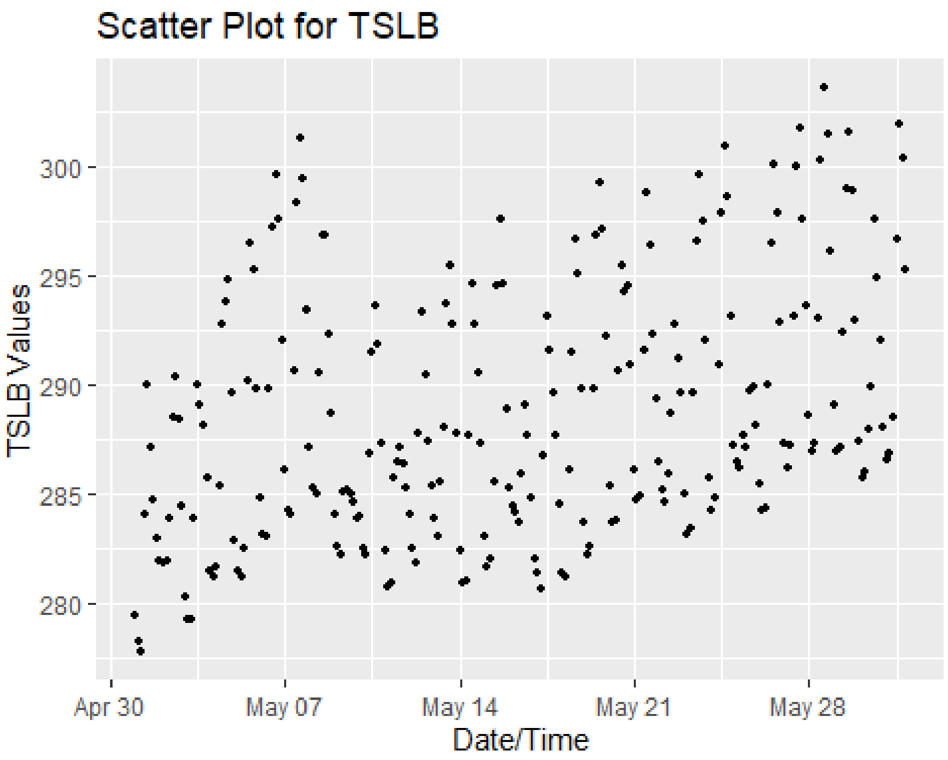
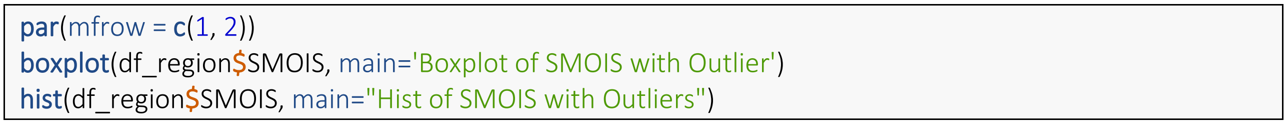


Figure 78 - Scatter plot of TSLB.

The scatter plot also does not show any value way out of the range from the others, therefore, the values will be retained.

SMOIS: The box plot shows that there are just 1 value being an outlier.



```
par(mfrow = c(1, 2))
boxplot(df_region$SMOIS, main='Boxplot of SMOIS with Outlier')
hist(df_region$SMOIS, main="Hist of SMOIS with Outliers")
```

Figure 79 - Code to view boxplot and histogram of SMOIS.

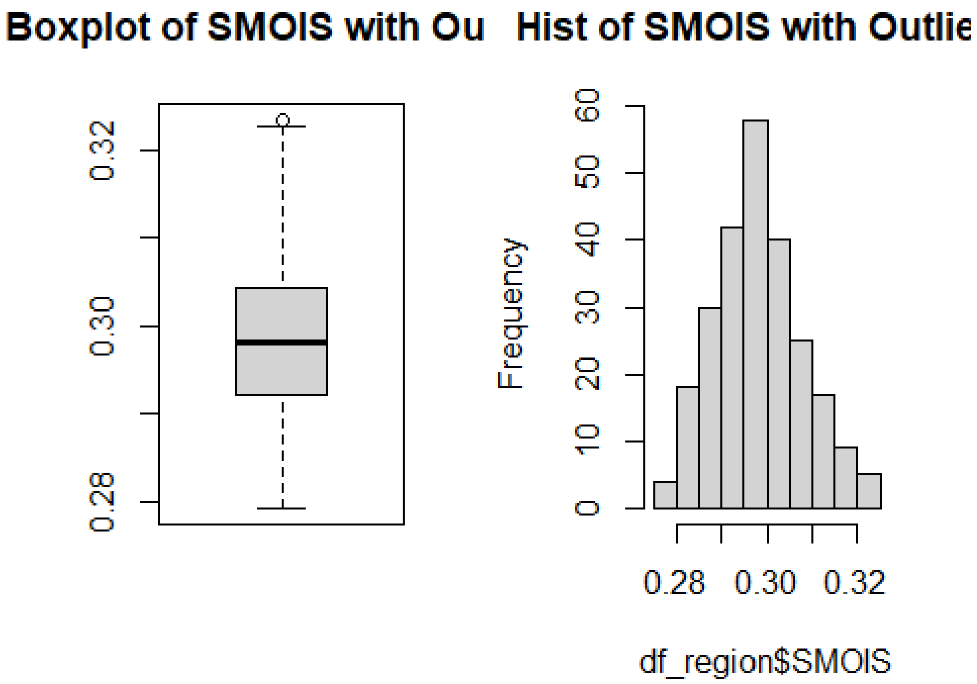


Figure 80 - Boxplot and Histogram of soil moisture.

```
z_scoreSMOIS <- scale(df_region$SMOIS)
SMOISoutliers <- df_region$SMOIS[abs(z_scoreSMOIS) > threshold]
length(SMOISoutliers)

## [1] 10
```

Figure 81 - Code to view outliers of SMOIS using z-score.

Using the z-score method, there are 10 outliers. The scatter plot of SMOIS is also shown to revealed that there may indeed be outliers.

```
ggplot(data = df_region, aes(x = Datetime, y = SMOIS)) +
 geom_point(size = 1) +
 labs(x = "Date/Time", y = "SMOIS values", title = "Scatter Plot for SMOIS")
```

Figure 82 - Code showing the scatter plot of SMOIS.

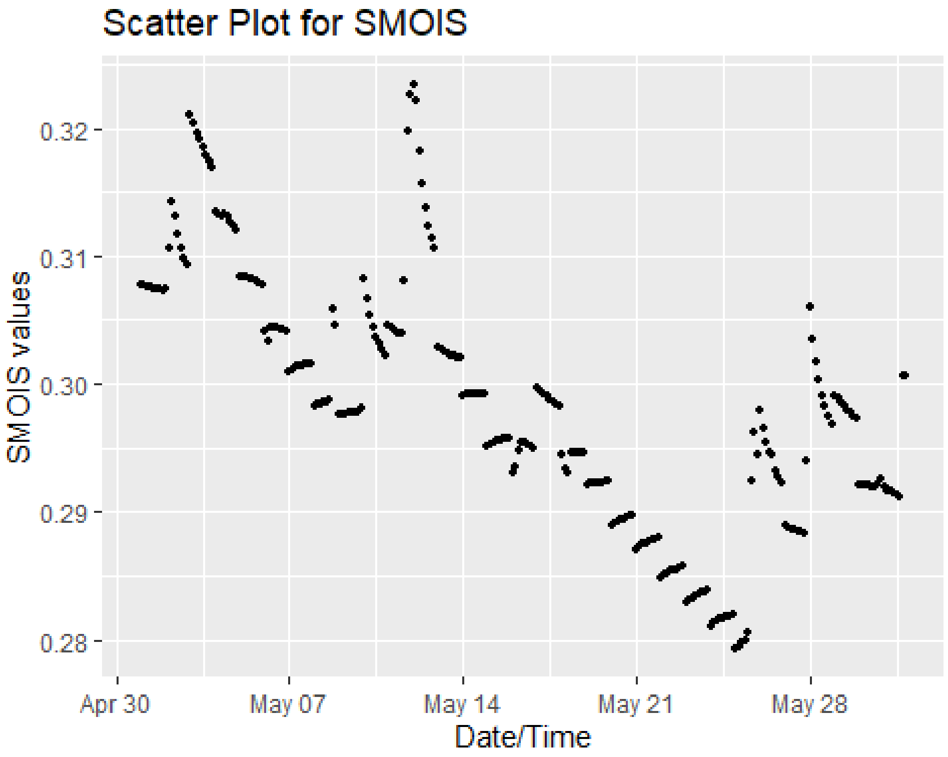
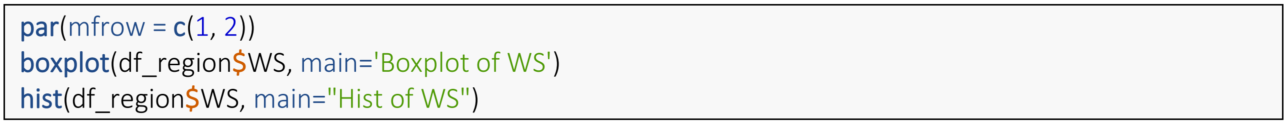


Figure 83 - Scatter plot of soil moisture.

However, SMOIS which is the soil moisture can see an increase than the normal trend if there is precipitation or rain on these days. Because these outlier' values are also not too much out of range, the values will be retained.

WS: Wind speed also shows there are some outliers which z-score method calculate to be 16 in numbers.



```
par(mfrow = c(1, 2))
boxplot(df_region$WS, main='Boxplot of WS')
hist(df_region$WS, main="Hist of WS")
```

Figure 84 - Code for z-score of WS.

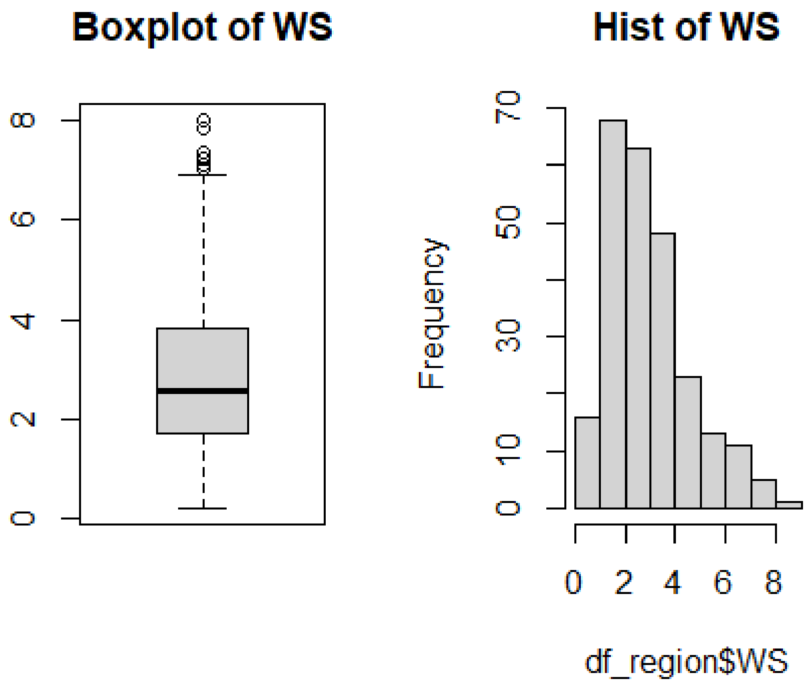


Figure 85 - Boxplot and Histogram of Wind speed.

```
z_scoreWS <- scale(df_region$WS)
WSoutliers <- df_region$WS[abs(z_scoreWS) > threshold]
length(WSoutliers)

## [1] 16

ggplot(data = df_region, aes(x = Datetime, y = WS)) +
  geom_point(size = 2) +
  labs(x = "Date/Time", y = "WS values", title = "Scatter Plot for WS")
```

Figure 86 - Code for z-score of WS and its scatter plot.

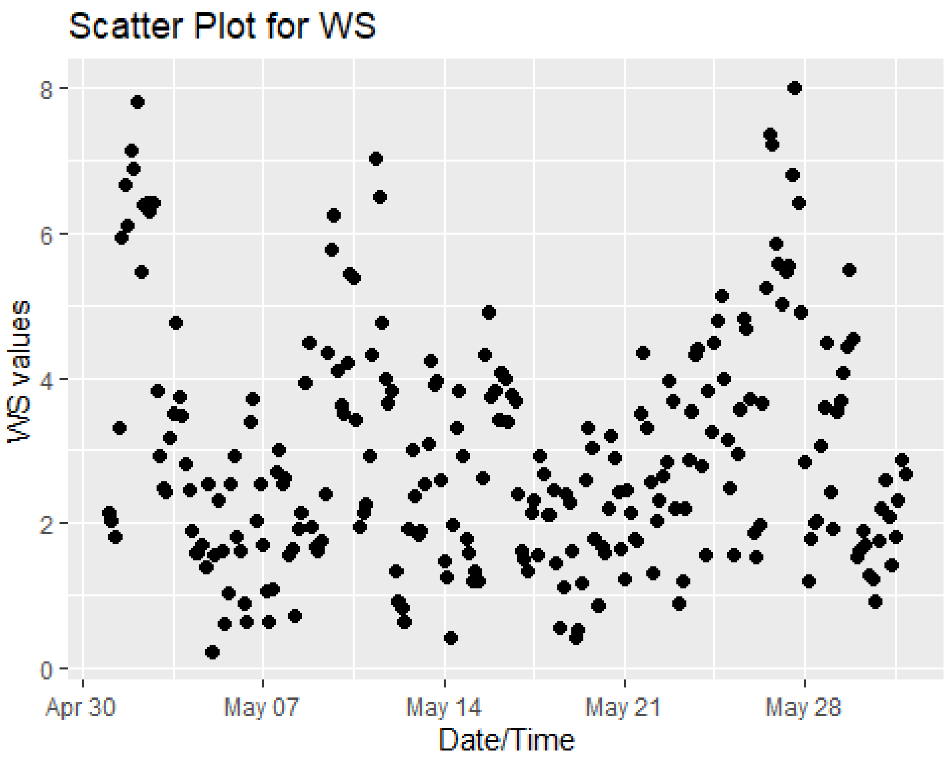


Figure 87 - Scatter plot for wind speed.

The scatter plot shows that these values are not too far from the other group, therefore, they may not really be an outlier. Therefore, there values are also retained.

SNOW: The snow column consists of zeros as values, therefore, there are no snow in the month and there is automatically no outlier for this.

Finally, the data is completely clean, and a copy of the clean data is saved and named df_warrington after the name of the selected location.

```
df_warrington <- df_region

sum(is.na(df_warrington))

## [1] 0
```

Figure 88 - Code that renames the clean data and confirms that the data is completely clean.

5.0 EXPLORATORY DATA ANALYSIS

Here, the dataset is inspected with different visualisation techniques to understand the data and its structure (Bissonette, 2021; Goloborodko *et al.*, 2013). These exploration shows the trend of each variable and the relationship between two or more variables.

5.1 Dataset Inspection

Here, the structure of the data, numbers of rows, numbers of columns are inspected to see a quick glance or an overall insight of what the data to be analysed is.

```
dim(df_warrington)

## [1] 248  12
```

Figure 89 - Code that shows the dimension of the data frame of the selected region.

As shown from above output of the code snippet, the data frame has a total of 248 rows and 12 columns.

```
str(df_warrington)

## 'data.frame':    248 obs. of  12 variables:
##  $ TSK     : num  278 277 278 288 293 ...
##  $ PSFC    : num  1e+05 1e+05 1e+05 1e+05 1e+05 ...
##  $ U10     : num  2.1 2 -0.1 2.2 3.77 ...
##  $ V10     : num  -0.5 0.3 1.8 2.5 4.6 6 6.1 7.1 6.7 7.8 ...
##  $ Q2      : num  0.00466 0.00422 0.00363 0.00501 0.00391 ...
##  $ RAINC   : num  0 0 0 0 0 0 0 0 0 0.2 ...
##  $ RAINNC  : num  0 0 0 0 0 0 0 0 1.9 3.8 ...
##  $ SNOW    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ TSLB    : num  280 278 278 284 290 ...
##  $ SMOIS   : num  0.308 0.308 0.308 0.308 0.307 ...
##  $ Datetime: POSIXct, format: "2018-05-01 00:00:00" "2018-05-01 03:00:00" ...
##  $ WS      : num  2.16 2.02 1.8 3.33 5.95 ...
```

Figure 90 - Code that checks the internal structure of the data frame.

All the variables are the correct format: numeric which is important for machine learning and statistical analysis.

```
summary(df_warrington)

##       TSK            PSFC            U10              V10
##  Min.   :277.1   Min.   :100101   Min.   :-8.0000   Min.   :-4.00000
##  1st Qu.:284.2   1st Qu.:101058   1st Qu.:-1.7000   1st Qu.:-1.70000
##  Median :288.0   Median :101567   Median :-0.1000   Median :-0.58650
##  Mean   :289.8   Mean   :101447   Mean   :-0.2767   Mean   :-0.08296
##  3rd Qu.:295.0   3rd Qu.:101879   3rd Qu.: 1.2738   3rd Qu.: 1.40000
##  Max.   :307.0   Max.   :102664   Max.   : 6.4000   Max.   : 7.80000
##        Q2              RAINC            RAINNC            SNOW
##  Min.   :0.003260   Min.   :0.0000   Min.   : 0.0000   Min.   :0
##  1st Qu.:0.005097   1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0
##  Median :0.006345   Median :0.0000   Median : 0.0000   Median :0
##  Mean   :0.006440   Mean   :0.1919   Mean   : 0.5754   Mean   :0
##  3rd Qu.:0.007752   3rd Qu.:0.0000   3rd Qu.: 0.0000   3rd Qu.:0
##  Max.   :0.010910   Max.   :4.1000   Max.   :13.5000   Max.   :0
##       TSLB           SMOIS           Datetime
##  Min.   :277.8   Min.   :0.2793   Min.   :2018-05-01 00:00:00
##  1st Qu.:284.5   1st Qu.:0.2921   1st Qu.:2018-05-08 17:15:00
##  Median :287.7   Median :0.2982   Median :2018-05-16 10:30:00
##  Mean   :288.9   Mean   :0.2984   Mean   :2018-05-16 10:30:00
##  3rd Qu.:292.9   3rd Qu.:0.3044   3rd Qu.:2018-05-24 03:45:00
##  Max.   :303.6   Max.   :0.3234   Max.   :2018-05-31 21:00:00
##        WS
##  Min.   :0.2236
##  1st Qu.:1.7377
##  Median :2.5524
##  Mean   :2.9608
##  3rd Qu.:3.8158
##  Max.   :8.0006
```

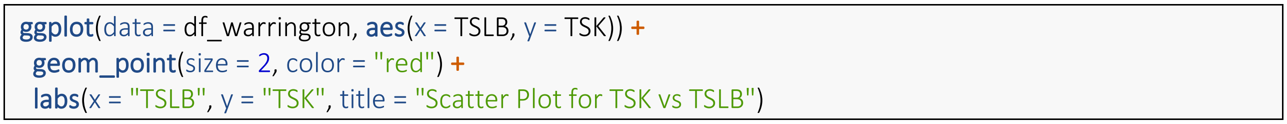
Figure 91 - Code that shows the statistical summary of the data frame.

The above gives the important statistical description of the data frame by column. This helps to easily get general idea and view about the dataset, the mean, median, minimum, maximum, first and third quartile.

5.2 Visualisation

The relationship between two or more variables will be explored here via visualisation/graphs like scatter plot, line chart and other relevant graphs.

5.2.1 Skin Temperature (TSK) vs Soil Temperature (TSLB)



```
ggplot(data = df_warrington, aes(x = TSLB, y = TSK)) +
  geom_point(size = 2, color = "red") +
  labs(x = "TSLB", y = "TSK", title = "Scatter Plot for TSK vs TSLB")
```

Figure 92 - Code for scatter plot of TSK vs TSLB.

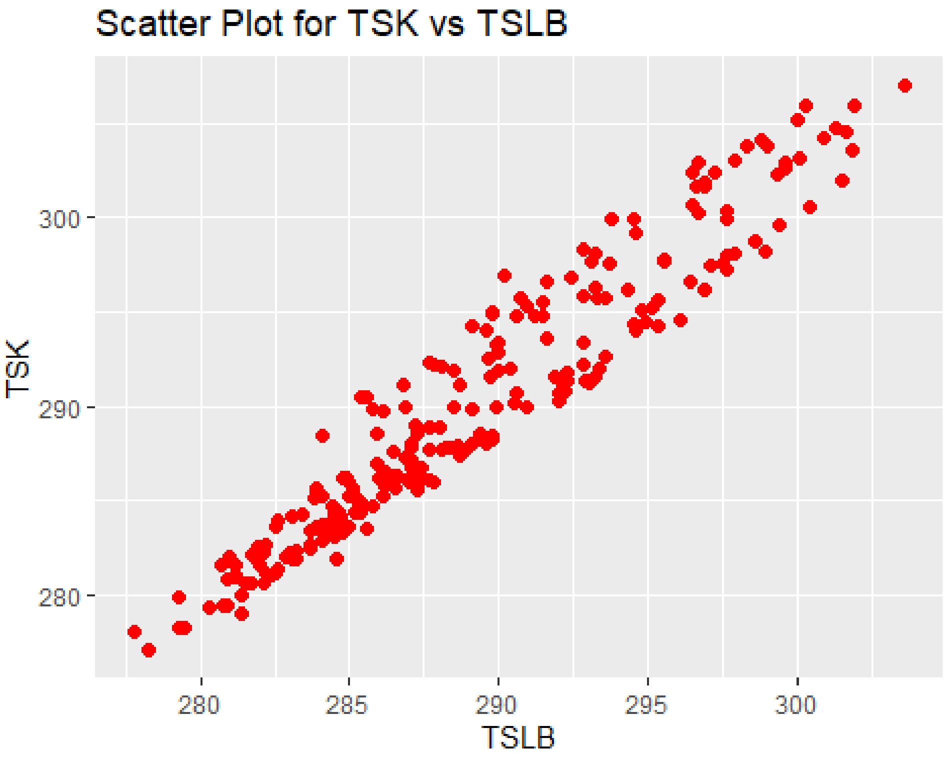
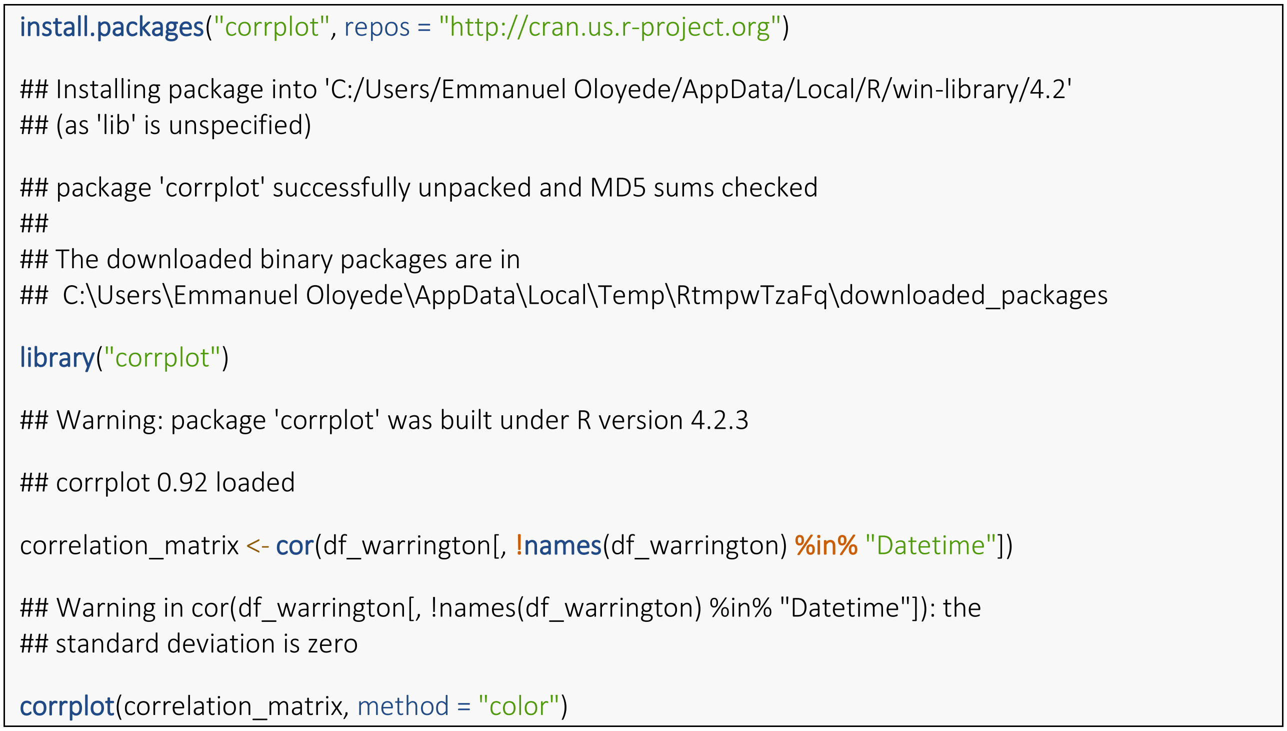


Figure 93 - Scatter plot showing the relationship between TSLB and TSK.

The TSK and TSLB shows a linear relationship that are directly proportional to each other. The higher the skin temperature gets, the higher the soil temperature.

5.2.2 Heatmap



```
install.packages("corrplot", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'corrplot' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpwTzaFq\downloaded_packages

library("corrplot")

## Warning: package 'corrplot' was built under R version 4.2.3

## corrplot 0.92 loaded

correlation_matrix <- cor(df_warrington[, !names(df_warrington) %in% "Datetime"])

## Warning in cor(df_warrington[, !names(df_warrington) %in% "Datetime"]): the
## standard deviation is zero

corrplot(correlation_matrix, method = "color")
```

Figure 94 - Code for heatmap of all variables in data frame.

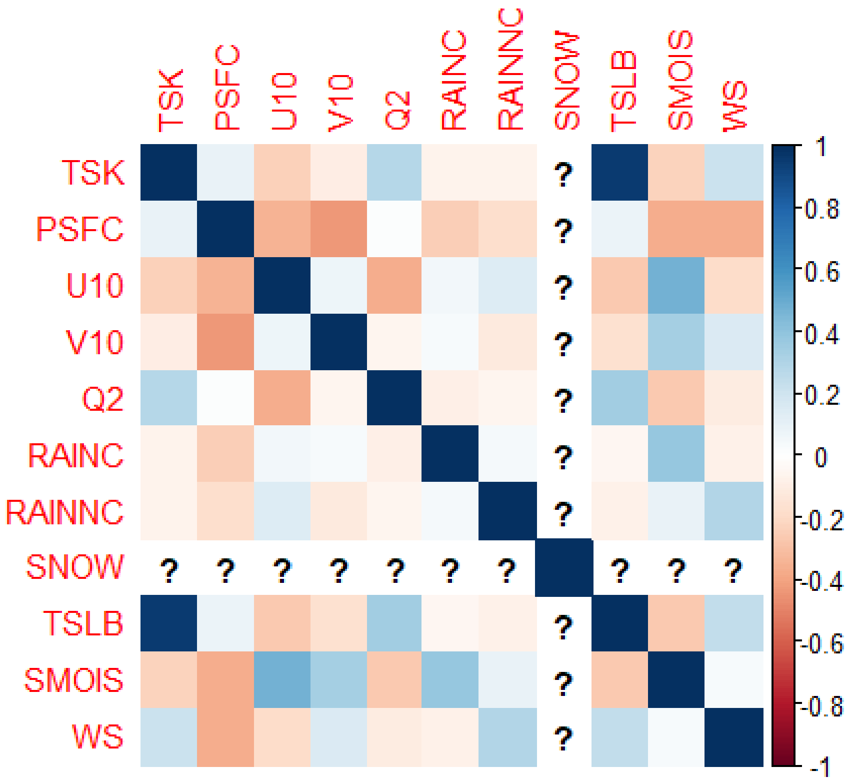
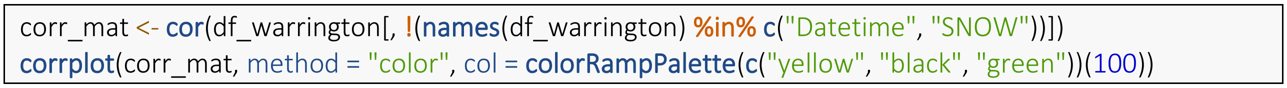


Figure 95 - Heatmap for all the variables.



```
corr_mat <- cor(df_warrington[, !(names(df_warrington) %in% c("Datetime", "SNOW"))])
corrplot(corr_mat, method = "color", col = colorRampPalette(c("yellow", "black", "green"))(100))
```

Figure 96 - Code that shows the heatmap of all variables except SNOW.

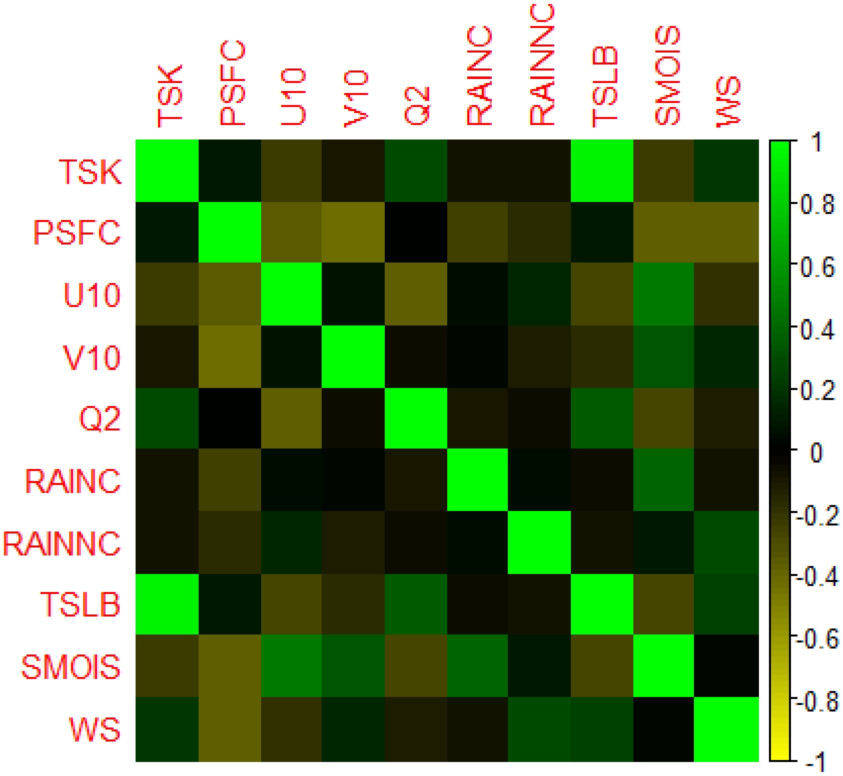
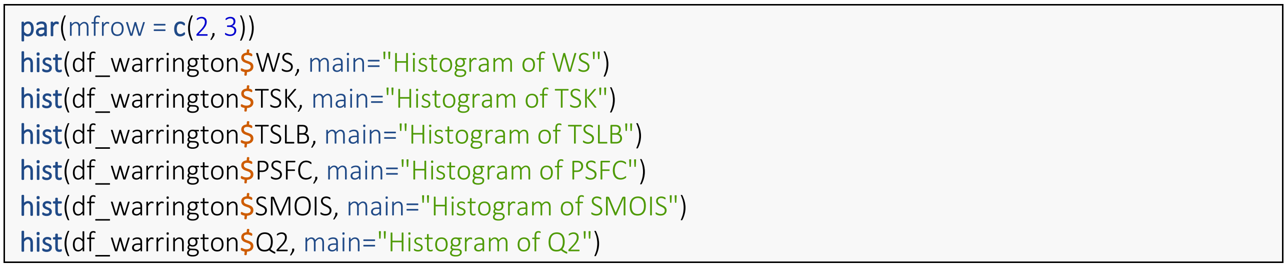


Figure 97 - Heatmap excluding the SNOW column.

The snow comes with "?" mark from the first heatmap because all its values are zero and cannot find correlation of it with any other variables in the dataset. The blue colours show positive correlation and the deeper it is, the higher the correlation while the red shows negative correlation. The second heatmap is done without SNOW.

5.2.3 Histogram and Boxplot

Histogram helps to understand how distributed the variables are. This will help understand the distribution of all the variables.



```
par(mfrow = c(2, 3))
hist(df_warrington$WS, main="Histogram of WS")
hist(df_warrington$TSK, main="Histogram of TSK")
hist(df_warrington$TSLB, main="Histogram of TSLB")
hist(df_warrington$PSFC, main="Histogram of PSFC")
hist(df_warrington$SMOIS, main="Histogram of SMOIS")
hist(df_warrington$Q2, main="Histogram of Q2")
```

Figure 98 - Code that shows several histograms of important variables.

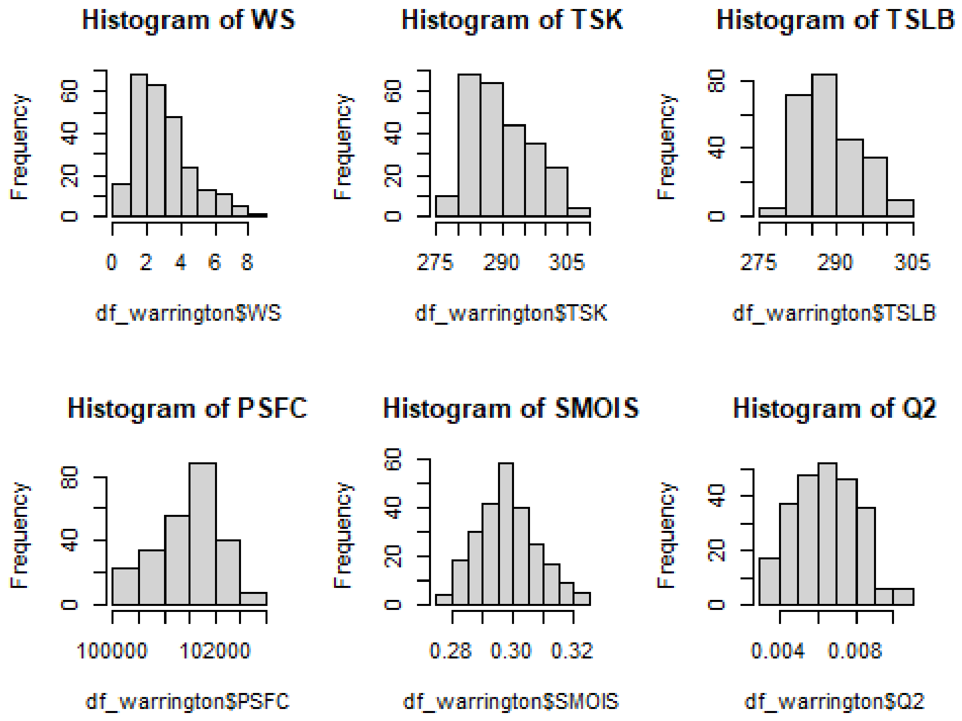
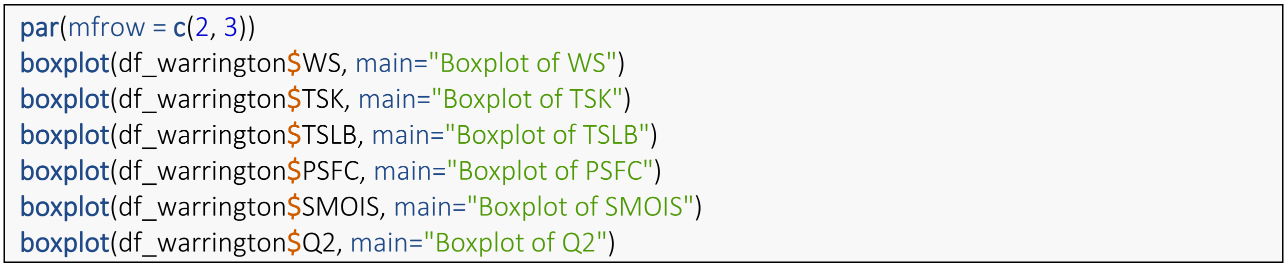


Figure 99 - Dashboard showing the histogram of vital variables.



```
par(mfrow = c(2, 3))
boxplot(df_warrington$WS, main="Boxplot of WS")
boxplot(df_warrington$TSK, main="Boxplot of TSK")
boxplot(df_warrington$TSLB, main="Boxplot of TSLB")
boxplot(df_warrington$PSFC, main="Boxplot of PSFC")
boxplot(df_warrington$SMOIS, main="Boxplot of SMOIS")
boxplot(df_warrington$Q2, main="Boxplot of Q2")
```

Figure 100 - Code to show several boxplots of variables.

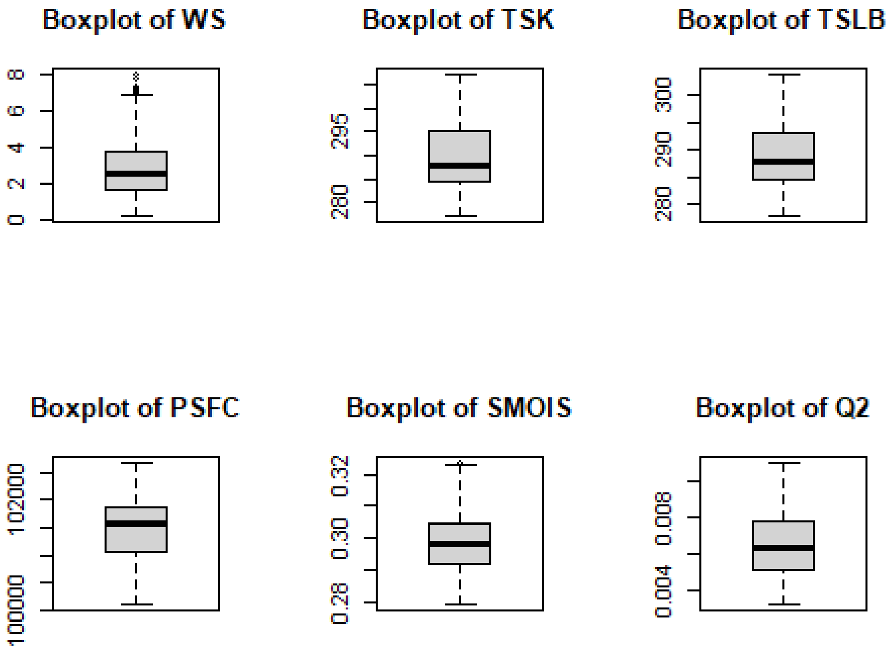


Figure 101 - Dashboard showing boxplot of important variables.

6.0 STATISTICAL ANALYSIS

In this report, the statistical analysis will focus more on the temperature than other variables because it is of more important to all the three stakeholders mentioned above. However, different statistical test will also be carried out on other variables to explore the relationship that occurs between them, and some research questions will be answered, while hypothesis will be accepted or rejected.

6.1 Univariate Analysis

The trend of skin temperature, also known as surface temperature, will be explored here using both scatter plot and line chart.

```
ggplot(data = df_warrington, aes(x = Datetime, y = TSK)) +
  geom_point(size = 2, color = "red") +
  labs(x = "Date/Time", y = "TSK", title = "Trend of TSK")
```

Figure 102 - Code for scatter plot of TSK.

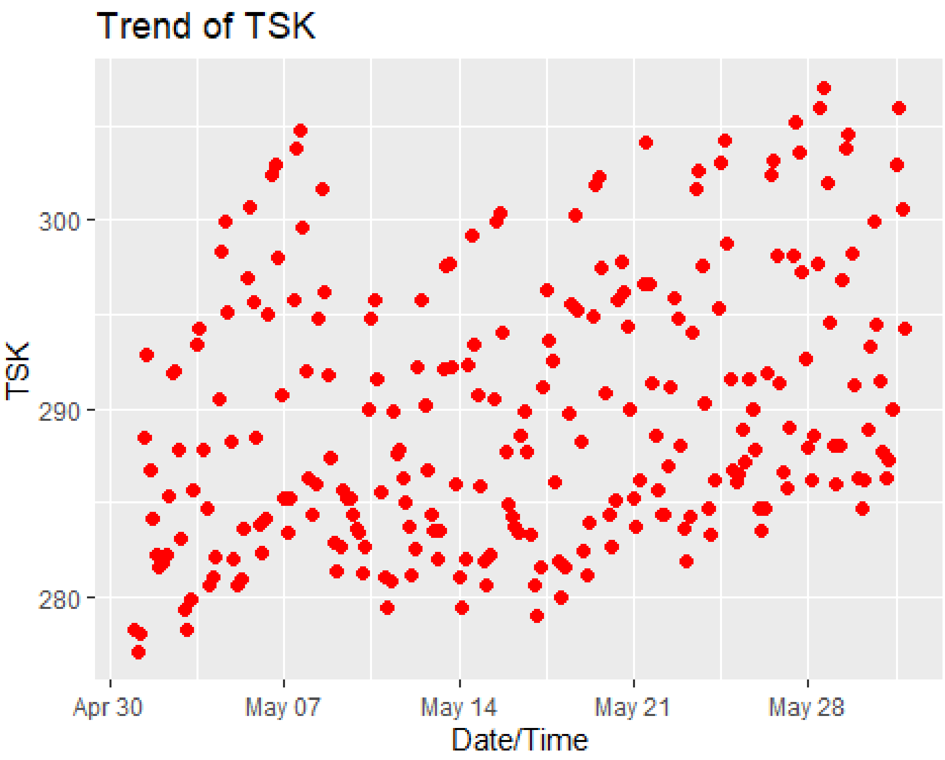
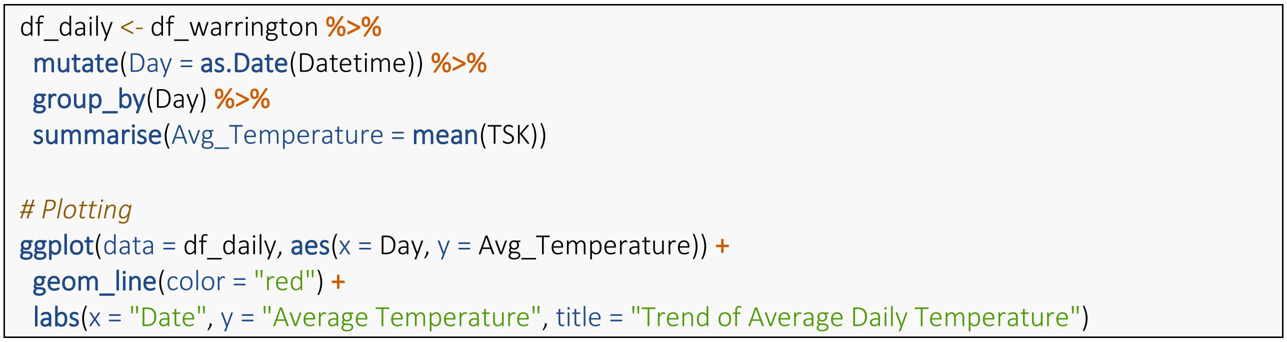


Figure 103 - Trend of TSK for the given month.

The plot shows an almost uniformly distributed value for skin temperature with everyday having its low and high temperature.



```
df_daily <- df_warrington %>%
  mutate(Day = as.Date(Datetime)) %>%
  group_by(Day) %>%
  summarise(Avg_Temperature = mean(TSK))

# Plotting
ggplot(data = df_daily, aes(x = Day, y = Avg_Temperature)) +
  geom_line(color = "red") +
  labs(x = "Date", y = "Average Temperature", title = "Trend of Average Daily Temperature")
```

Figure 104 - Code that view line trend of average values of TSK.

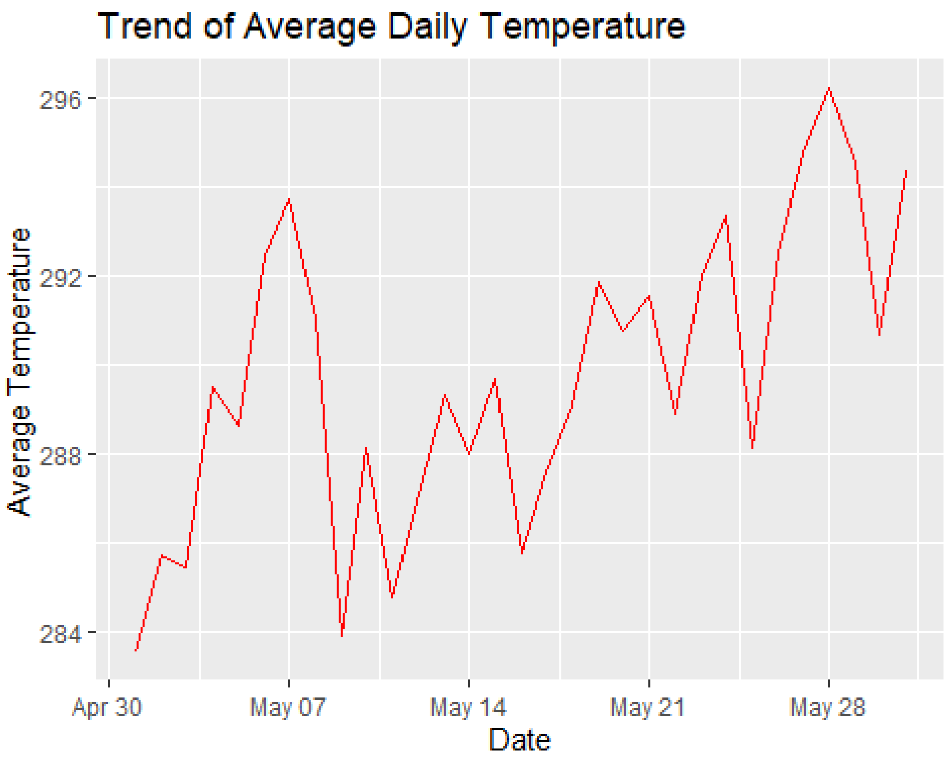


Figure 105 - Line plot showing the trend of average daily skin temperature.

The above plot is the average temperature trend for the month of May in the selected location (Warrington). The trend shows inconsistencies with some highs and lows; therefore, it can be deduced that the temperature for the month is fluctuating, and stakeholders should prepare for this scenario in the following month should this trend continues.

6.2 Bivariate Analysis

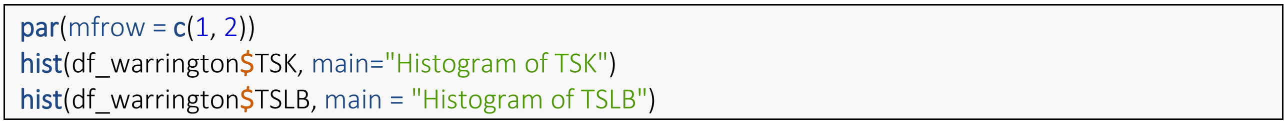
Here, the relationships between two variables are explored.

6.2.1 Skin Temperature vs Soil Temperature

To find the correlation between skin temperature and soil temperature, it is essential to understand some underlying factors before selecting a particular method.

1. The distribution of the data

2. Type of dataset (In this case, both are numerical). Using the histogram to check the distribution of the data:



```
par(mfrow = c(1, 2))
hist(df_warrington$TSK, main="Histogram of TSK")
hist(df_warrington$TSLB, main = "Histogram of TSLB")
```

Figure 106 - Code to view the distribution of TSK and TSLB.

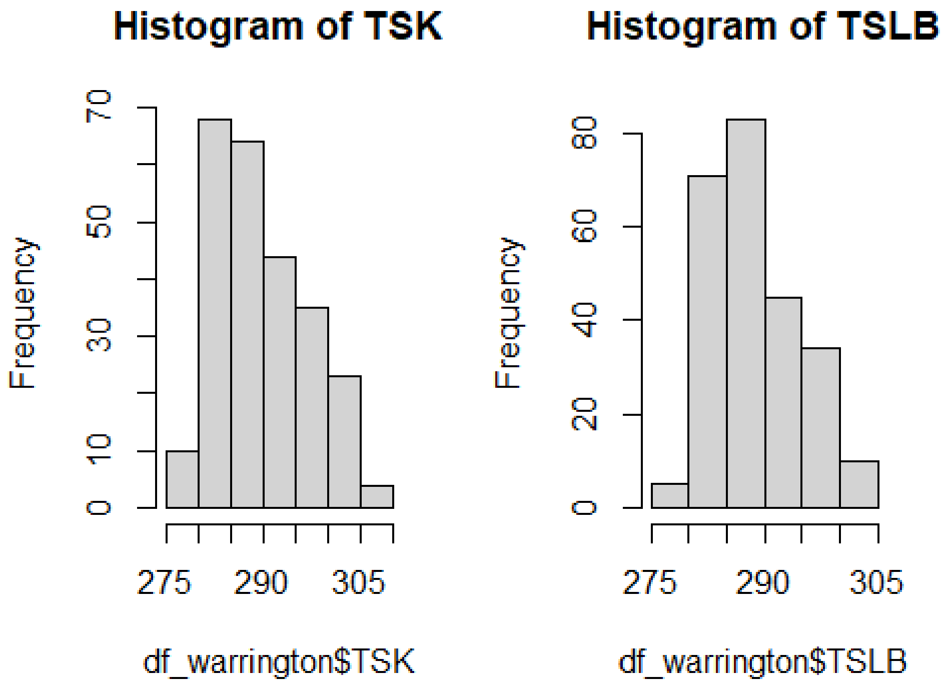
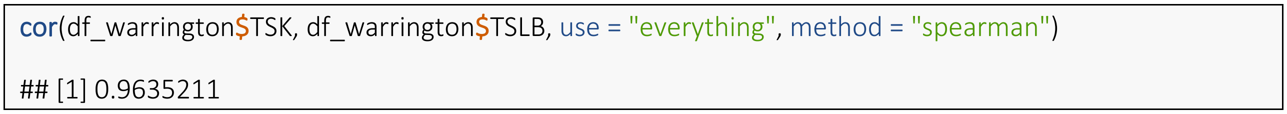


Figure 107 - Histogram of skin temperature and soil temperature.

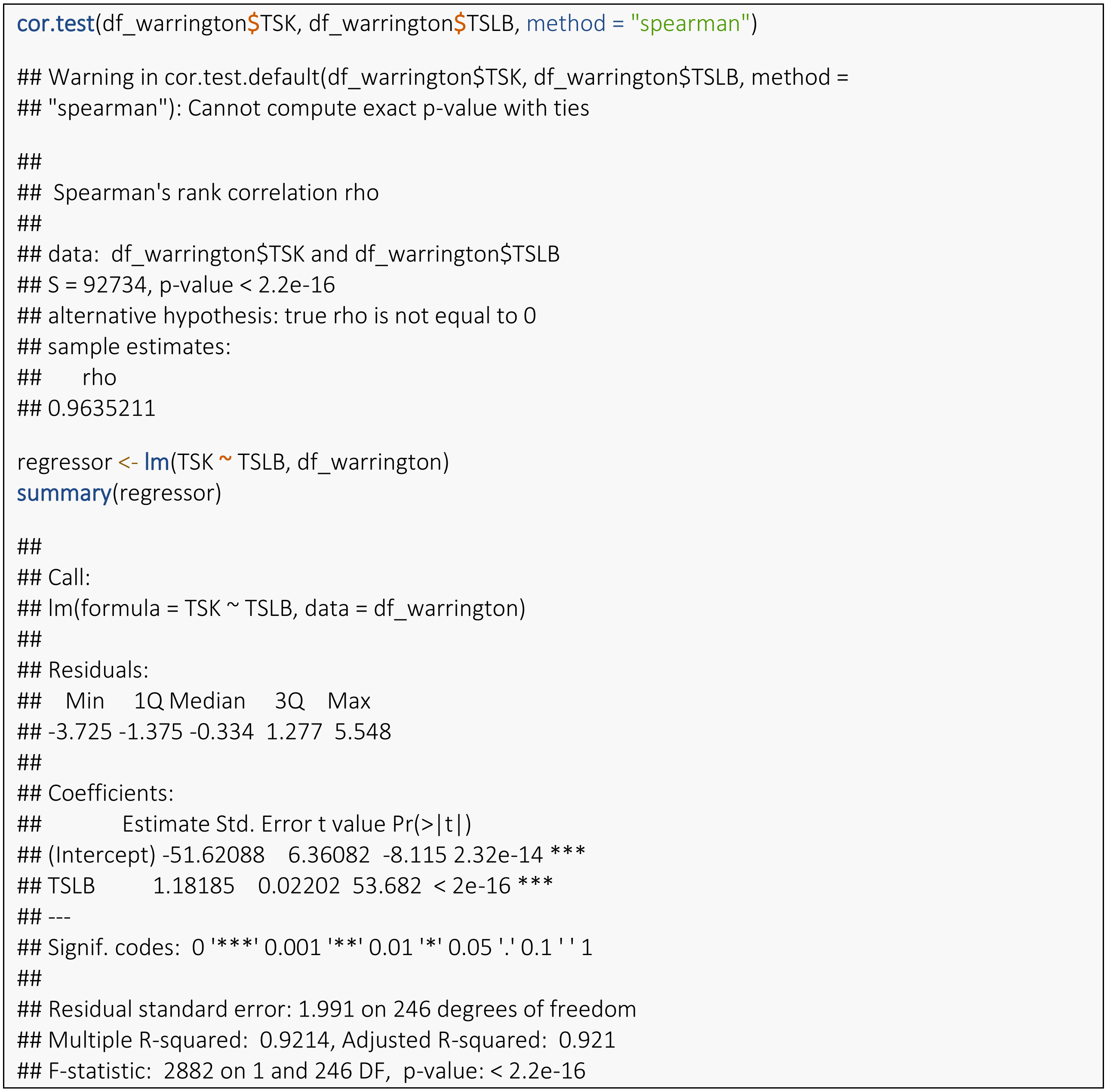
From the plot above, both variables are not normally distributed, therefore, the spearman method is appropriate, and the implementation is shown below:



```
cor(df_warrington$TSK, df_warrington$TSLB, use = "everything", method = "spearman")

## [1] 0.9635211
```

Figure 108 - Code to find the correlation between TSK and TSLB.

With a correlation coefficient of 0.9635211, the skin temperature and soil temperature are strongly correlated in positive direction. Which means, the increase in skin temperature will lead to increase in soil temperature and vice versa.



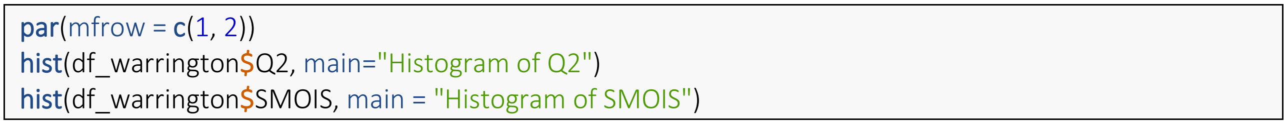
```
cor.test(df_warrington$TSK, df_warrington$TSLB, method = "spearman")

## Warning in cor.test.default(df_warrington$TSK, df_warrington$TSLB, method =
## "spearman"): Cannot compute exact p-value with ties

## 
##  Spearman's rank correlation rho
## 
## data:  df_warrington$TSK and df_warrington$TSLB
## S = 92734, p-value < 2.2e-16
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho 
## 0.9635211

regressor <- lm(TSK ~ TSLB, df_warrington)
summary(regressor)

## 
## Call:
## lm(formula = TSK ~ TSLB, data = df_warrington)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -3.725 -1.375 -0.334  1.277  5.548 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -51.62088    6.36082  -8.115 2.32e-14 ***
## TSLB          1.18185    0.02202  53.682  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 1.991 on 246 degrees of freedom
## Multiple R-squared:  0.9214, Adjusted R-squared:  0.921 
## F-statistic:  2882 on 1 and 246 DF,  p-value: < 2.2e-16
```

Figure 109 - Code for correlation test and regression method for TSK and TSLB.

The p-value from both regression and correlation test is less than 0.05. Therefore, the NULL hypothesis is rejected, and the alternative hypothesis is accepted. There is strong positive relationship between skin temperature and soil temperature.

6.2.2 Specific Humidity vs Soil Moisture

Checking for the distribution of both variables:



```
par(mfrow = c(1, 2))
hist(df_warrington$Q2, main="Histogram of Q2")
hist(df_warrington$SMOIS, main = "Histogram of SMOIS")
```

Figure 110 - Code for histogram distribution of SMOIS and Q2.

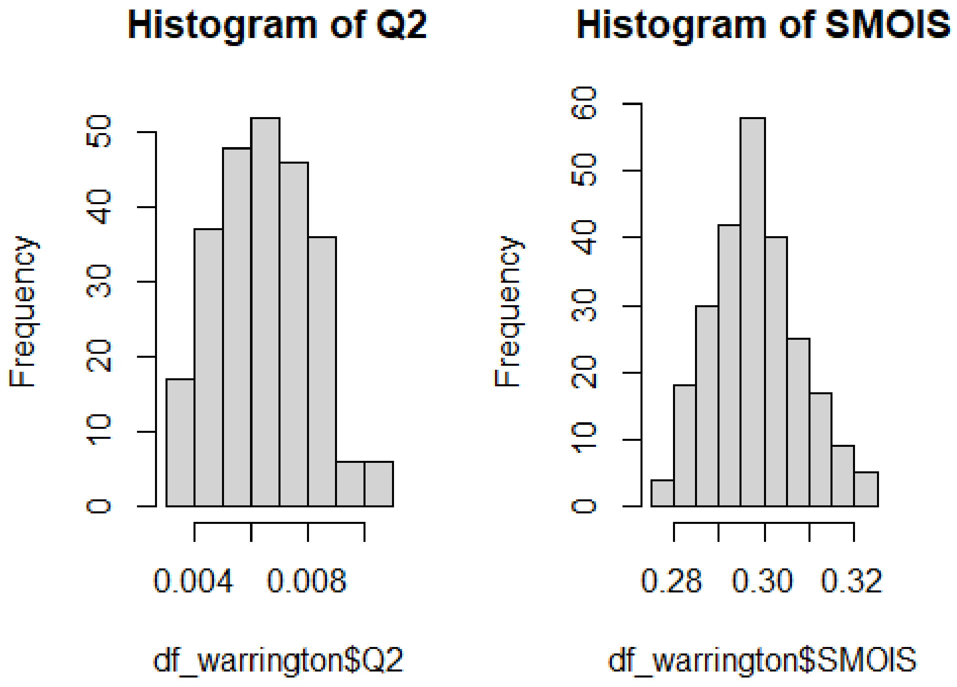
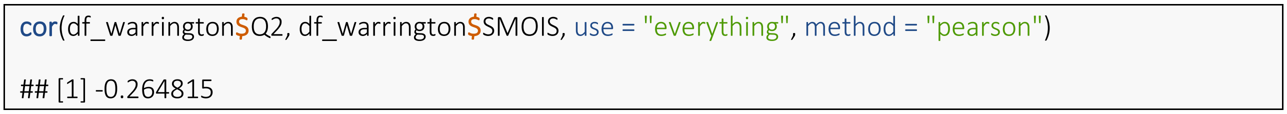


Figure 111 - Histogram of Q2 and SMOIS

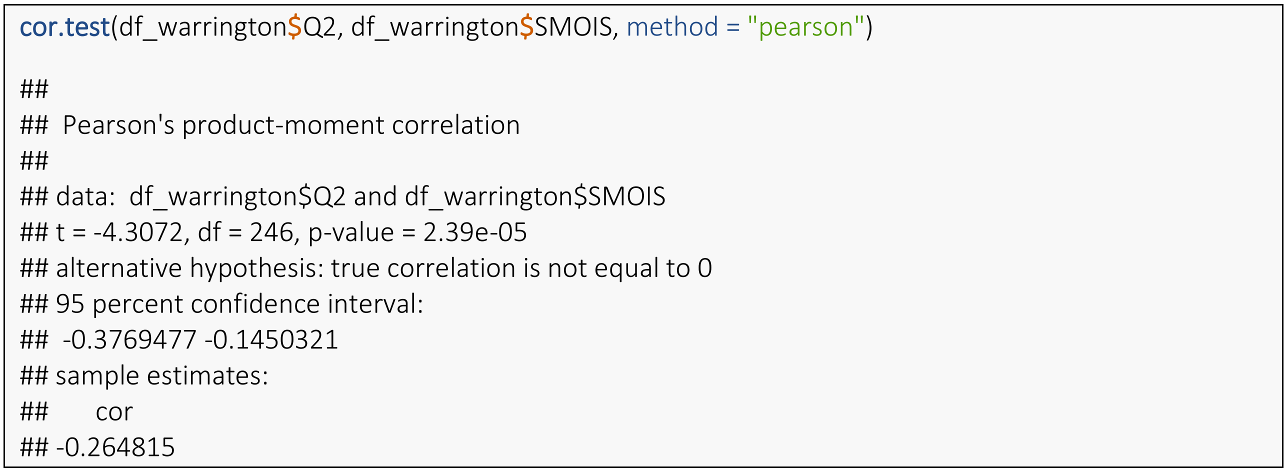
Both variables are normally distributed, hence, Pearson method can be used to calculate their correlation coefficient.



```
cor(df_warrington$Q2, df_warrington$SMOIS, use = "everything", method = "pearson")

## [1] -0.264815
```

Figure 112 - Code for correlation between SMOIS and Q2.

With a correlation coefficient of -0.264815, the variables can be said to have a weak correlation in a negative direction. To get the p-value, the following function is used:



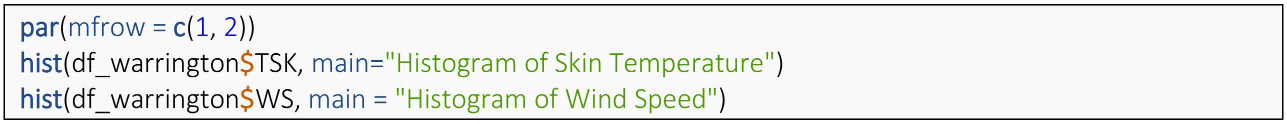
```
cor.test(df_warrington$Q2, df_warrington$SMOIS, method = "pearson")

##
##  Pearson's product-moment correlation
##
## data:  df_warrington$Q2 and df_warrington$SMOIS
## t = -4.3072, df = 246, p-value = 2.39e-05
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.3769477 -0.1450321
## sample estimates:
##       cor
## -0.264815
```

Figure 113 - Code for correlation test between SMOIS and Q2.

The p-value is less than 0.05, therefore, the NULL Hypothesis is rejected, and the alternative hypothesis is accepted. There is relationship between specific humidity and soil moisture.

6.2.3 Skin Temperature vs Wind Speed

The distribution of both variables is show below:



```
par(mfrow = c(1, 2))
hist(df_warrington$TSK, main="Histogram of Skin Temperature")
hist(df_warrington$WS, main = "Histogram of Wind Speed")
```

Figure 114 - Code for histogram distribution of WS and TSK.

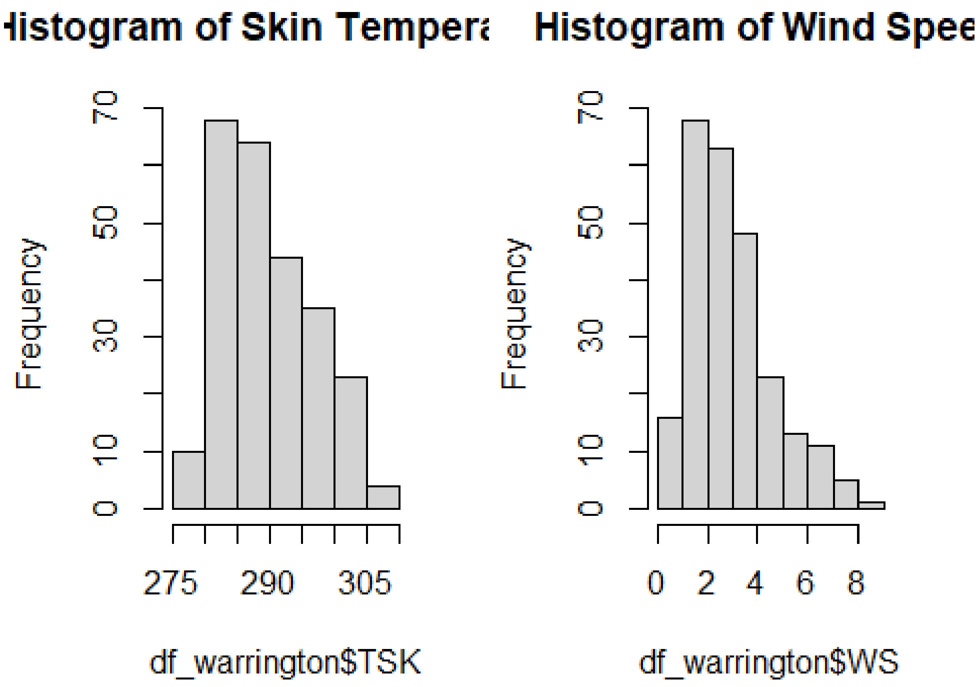
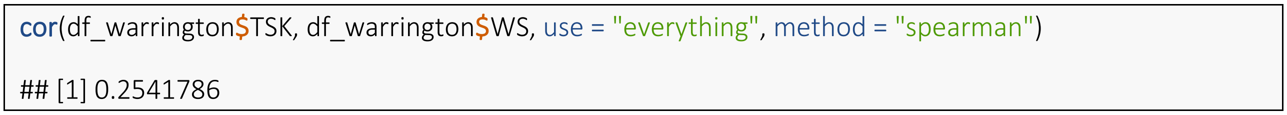


Figure 115 - Histogram of TSK and Wind Speed.

The variables are left skewed according to the histogram plot. Therefore, spearman method is best for this scenario and its executed as show below:



```
cor(df_warrington$TSK, df_warrington$WS, use = "everything", method = "spearman")
## [1] 0.2541786
```

Figure 116 - Code for correlation test between WS and TSK.

With a correlation coefficient of 0.2541786, the two variables are correlated in a positive direction, however, this correlation is weak. To calculate the p-value:

```
cor.test(df_warrington$TSK, df_warrington$WS, method = "spearman")

## Warning in cor.test.default(df_warrington$TSK, df_warrington$WS, method =
## "spearman"): Cannot compute exact p-value with ties

##
##  Spearman's rank correlation rho
##
## data:  df_warrington$TSK and df_warrington$WS
## S = 1895970, p-value = 5.137e-05
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##       rho
## 0.2541786
```

Figure 117 - Code for correlation test between WS and TSK.

The p-value is less than 0.05, therefore, the alternative hypothesis is accepted while the NULL hypothesis is rejected.

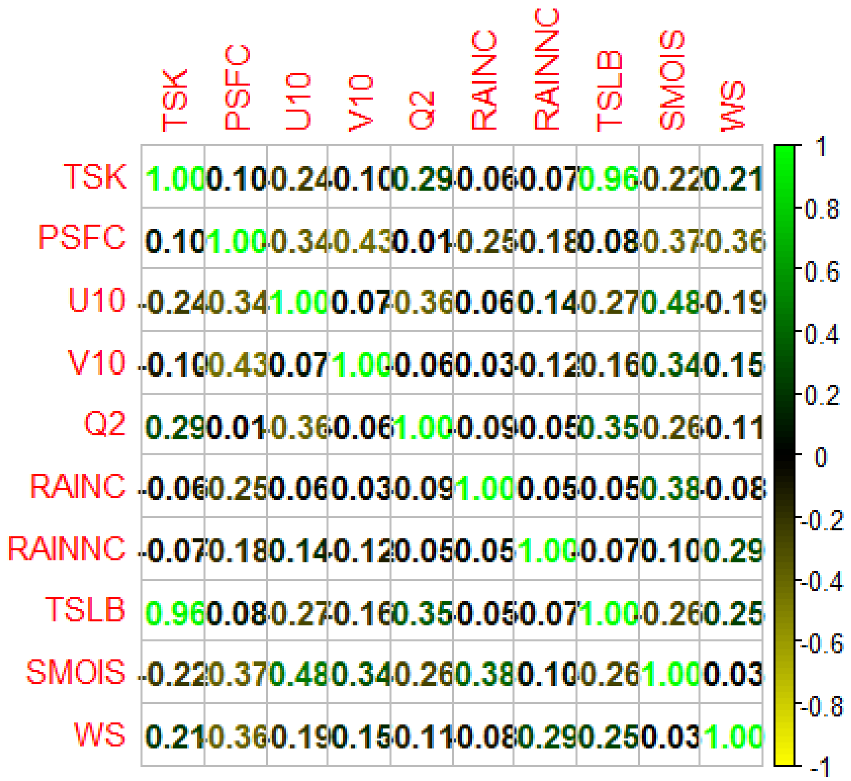
6.3 Multivariate Analysis

6.3.1 Heatmap

The heatmap showing the correlation score is first considered. The variables having an absolute correlation score of 0.2 and above will be extracted and consider for use in building the machine learning model.

```
corr_mat <- cor(df_warrington[, !(names(df_warrington) %in% c("Datetime", "SNOW"))])
corrplot(corr_mat, method = "number", col = colorRampPalette(c("yellow", "black", "green"))(100))
```

Figure 118 - Code for Heatmap



	TSK	PSFC	U10	V10	Q2	RAINC	RAINNC	TSLB	SMOIS	WS
TSK	1.00	0.10	-0.24	-0.10	0.29	-0.06	-0.07	0.96	-0.22	0.21
PSFC	0.10	1.00	-0.34	-0.43	0.01	-0.25	-0.18	0.08	-0.37	-0.36
U10	-0.24	-0.34	1.00	0.07	-0.36	0.06	0.14	-0.27	0.48	-0.19
V10	-0.10	-0.43	0.07	1.00	-0.06	0.03	-0.12	-0.16	0.34	0.15
Q2	0.29	0.01	-0.36	-0.06	1.00	-0.09	0.05	0.35	-0.26	-0.11
RAINC	-0.06	-0.25	0.06	0.03	-0.09	1.00	0.05	0.05	0.38	-0.08
RAINNC	-0.07	-0.18	0.14	-0.12	0.05	0.05	1.00	-0.07	0.10	0.29
TSLB	0.96	0.08	-0.27	-0.16	0.35	0.05	-0.07	1.00	-0.26	0.25
SMOIS	-0.22	-0.37	0.48	0.34	-0.26	0.38	0.10	-0.26	1.00	0.03
WS	0.21	-0.36	-0.19	0.15	-0.11	-0.08	0.29	0.25	0.03	1.00

Figure 119 - Heatmap showing the correlation of each variable.

From the above heatmap, Wind speed, soil moisture, soil temperature, and specific humidity has correlation score above 0.2 in relation to skin temperature. These features will be used for machine learning model.

6.3.2 Multiple Regression

To check the level of significance of each variable on the skin temperature, a multiple regression function is applied as shown below:

```
r_data <- df_warrington[, c("TSK", "PSFC", "Q2", "RAINC", "RAINNC", "TSLB", "SMOIS", "WS")]
head(r_data)

##     TSK     PSFC      Q2 RAINC RAINNC  TSLB  SMOIS       WS
## 1 278.3 100422.0 0.00466     0      0 279.5 0.3077 2.158703
## 2 277.1 100422.0 0.00422     0      0 278.3 0.3077 2.022375
## 3 278.1 100370.0 0.00363     0      0 277.8 0.3076 1.802776
## 4 288.5 100274.0 0.00501     0      0 284.1 0.3076 3.330165
## 5 292.9 100176.0 0.00391     0      0 290.0 0.3075 5.945948
## 6 286.8 100118.5 0.00422     0      0 287.1 0.3075 6.664083
```

Figure 120 - Code to extract variables into data frame for regression.

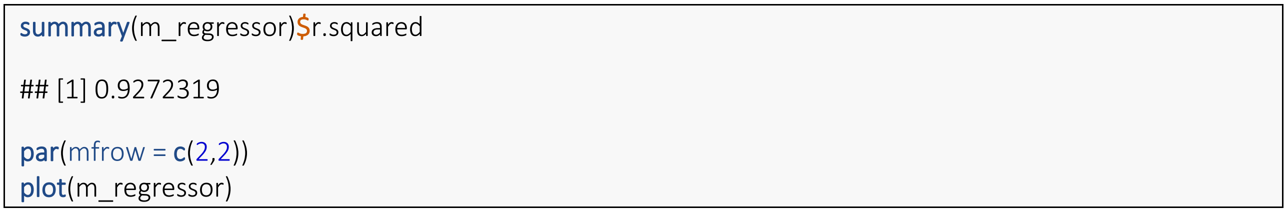
Running the regression analysis

```
m_regressor<- lm(TSK ~., r_data)
summary(m_regressor)

##
## Call:
## lm(formula = TSK ~ ., data = r_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3301 -1.3191 -0.1862  1.1891  4.7667
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -7.724e+01  2.683e+01  -2.879  0.00435 **
## PSFC         4.050e-05  2.462e-04   0.165  0.86944
## Q2          -2.496e+02  8.538e+01  -2.924  0.00379 **
## RAINC       -3.575e-01  1.756e-01  -2.036  0.04285 *
## RAINNC       3.722e-02  5.824e-02   0.639  0.52341
## TSLB         1.234e+00  2.527e-02  48.822  < 2e-16 ***
## SMOIS        2.975e+01  1.515e+01   1.964  0.05064 .
## WS          -2.173e-01  9.376e-02  -2.318  0.02131 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.939 on 240 degrees of freedom
## Multiple R-squared:  0.9272, Adjusted R-squared:  0.9251
## F-statistic: 436.9 on 7 and 240 DF,  p-value: < 2.2e-16
```

Figure 121 - Regression analysis code.

From the above output, TSLB (Soil temperature) has the highest significance level on surface temperature, followed by specific humidity (Q2), then accumulated precipitation and wind speed.

To build the machine learning, the most significant variables will be the input variables for the machine learning models. These are: Q2, RAINC, TSLB, SMOIS and WS with target variable being TSK.



```
summary(m_regressor)$r.squared

## [1] 0.9272319

par(mfrow = c(2,2))
plot(m_regressor)
```

Figure 122 - Code that plots the summary of the regression.

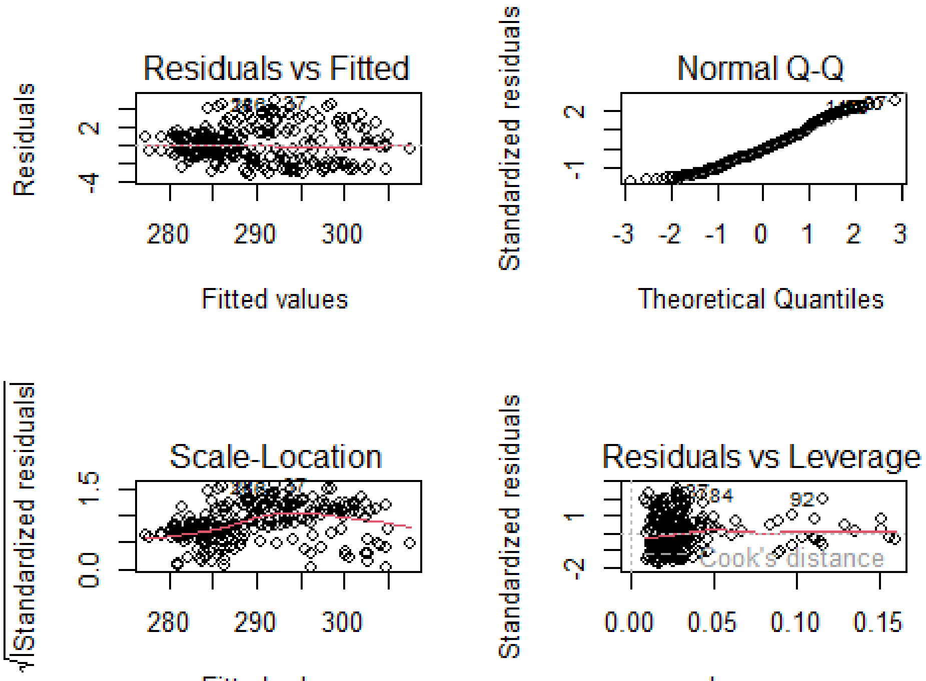
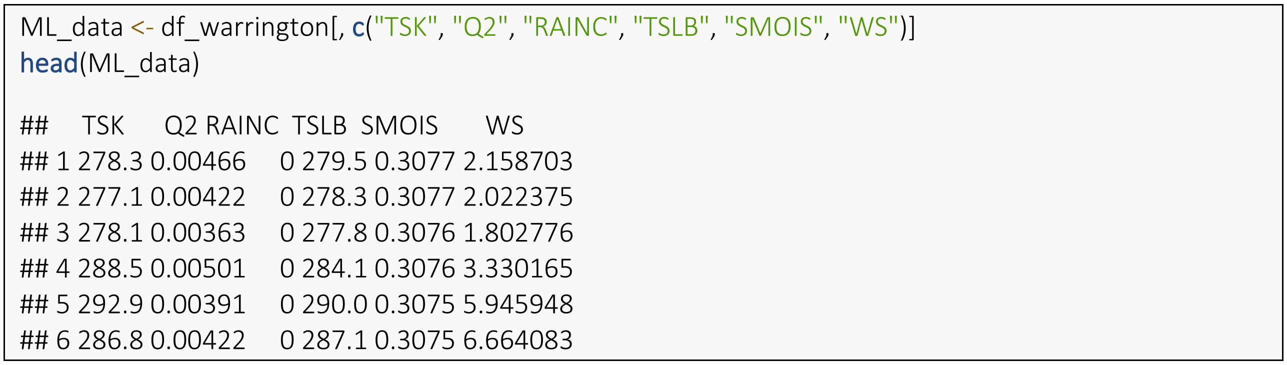


Figure 123 - Plot of Residuals

With an R^2 of 0.9272319, it shows the independent variables performs well in explaining the variability of the output variable.

7.0 MACHINE LEARNING



```
ML_data <- df_warrington[, c("TSK", "Q2", "RAINC", "TSLB", "SMOIS", "WS")]
head(ML_data)

##     TSK      Q2 RAINC  TSLB  SMOIS       WS
## 1 278.3 0.00466     0 279.5 0.3077 2.158703
## 2 277.1 0.00422     0 278.3 0.3077 2.022375
## 3 278.1 0.00363     0 277.8 0.3076 1.802776
## 4 288.5 0.00501     0 284.1 0.3076 3.330165
## 5 292.9 0.00391     0 290.0 0.3075 5.945948
## 6 286.8 0.00422     0 287.1 0.3075 6.664083
```

Figure 124 - Code that extract Machine Learning data.

7.1 Divide into Training and Testing Data

The dataset will be divided into training and testing dataset with the training dataset making 80% of the data while the testing data takes 20%.

```
install.packages("caTools", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'caTools' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'caTools'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Emmanuel
## Oloyede\AppData\Local\R\win-library\4.2\00LOCK\caTools\libs\x64\caTools.dll to
## C:\Users\Emmanuel
## Oloyede\AppData\Local\R\win-library\4.2\caTools\libs\x64\caTools.dll:
## Permission denied

## Warning: restored 'caTools'

##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpwTzaFq\downloaded_packages

library("caTools")

## Warning: package 'caTools' was built under R version 4.2.3
```

```
set.seed(123) # Set seed for reproducibility
split <- sample.split(ML_data, SplitRatio = 0.8)
```

Figure 125 - Code that specify the split ratio of training and testing data.

Separate into Training and testing dataset.

```
training_df <- subset(ML_data, split == TRUE)
testing_df <- subset(ML_data, split == FALSE)
```

Figure 126 - Code that split machine learning data to training and testing data.

7.2 Linear Regression Model

```
ML_regressor<- lm(TSK ~., training_df)
summary(ML_regressor)

##
## Call:
## lm(formula = TSK ~ ., data = training_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3797 -1.2758 -0.2255  1.2137  5.2237
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -62.60595   10.74129  -5.829 2.99e-08 ***
## Q2         -313.54579   99.49933  -3.151  0.00194 **
## RAINC        -0.16403    0.21300  -0.770  0.44240
## TSLB          1.22659    0.02965  41.372  < 2e-16 ***
## SMOIS         2.14549   17.68070   0.121  0.90357
## WS           -0.17379    0.10020  -1.734  0.08477 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.916 on 160 degrees of freedom
## Multiple R-squared:  0.9293, Adjusted R-squared:  0.9271
## F-statistic: 420.8 on 5 and 160 DF,  p-value: < 2.2e-16
```

Figure 127 - Code for Linear Regression Model.

7.2.1 Check for Linearity

Checking for linearity is an important aspect of multivariate analysis as this will help to understand in the variables are linearly related, hence, multiple linear regression model can be used.

Null Hypothesis: There is linear relationship.
Alternative Hypothesis: There is no linearity.

```
library(lmtest)

## Warning: package 'lmtest' was built under R version 4.2.3

raintest(ML_regressor)

##
##  Rainbow test
##
## data:  ML_regressor
## Rain = 1.0991, df1 = 83, df2 = 77, p-value = 0.338
```

Figure 128 - Code to check for linearity.

The p-value is 0.338 which is greater than 0.05, the Null hypothesis of linear relationship is accepted. Therefore, there is linearity and linear regression is suitable. However, there are other conditions to be inspected which are done below:

7.2.2 Check for heteroscedasticity.

Heteroscedasticity is a situation where the residuals are not constant over all the independent variables. This is check using the code as shown below:

```
install.packages("car", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'car' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpwTzaFq\downloaded_packages

library("car")

## Warning: package 'car' was built under R version 4.2.3

## Loading required package: carData

## Warning: package 'carData' was built under R version 4.2.3

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some

ncvTest(ML_regressor)

## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 11.56089, Df = 1, p = 0.00067354
```

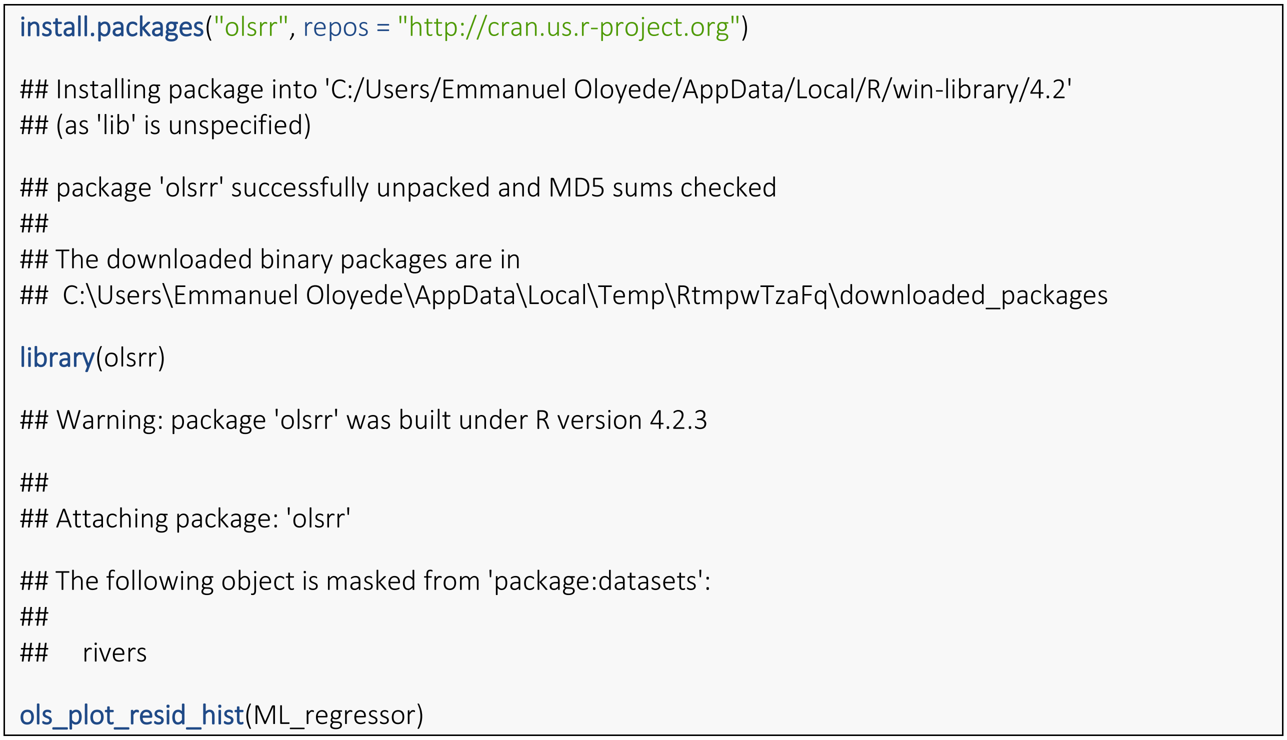
Figure 129 - Code that check for heteroscedasticity.

Null Hypothesis: There is no heteroscedasticity.
Alternative Hypothesis: There is heteroscedasticity.

The p-value is less than 0.05 and null hypothesis of no heteroscedasticity is rejected, therefore, there is heteroscedasticity.

7.2.3 Check for Normality of Residual

The normality of error or residuals is checked through the following code snippet: This is when errors from the regression model are normally distributed.



```
install.packages("olsrr", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'olsrr' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpwTzaFq\downloaded_packages

library(olsrr)

## Warning: package 'olsrr' was built under R version 4.2.3

##
## Attaching package: 'olsrr'

## The following object is masked from 'package:datasets':
##
##     rivers

ols_plot_resid_hist(ML_regressor)
```

Figure 130 - Code for Normality of Residuals.

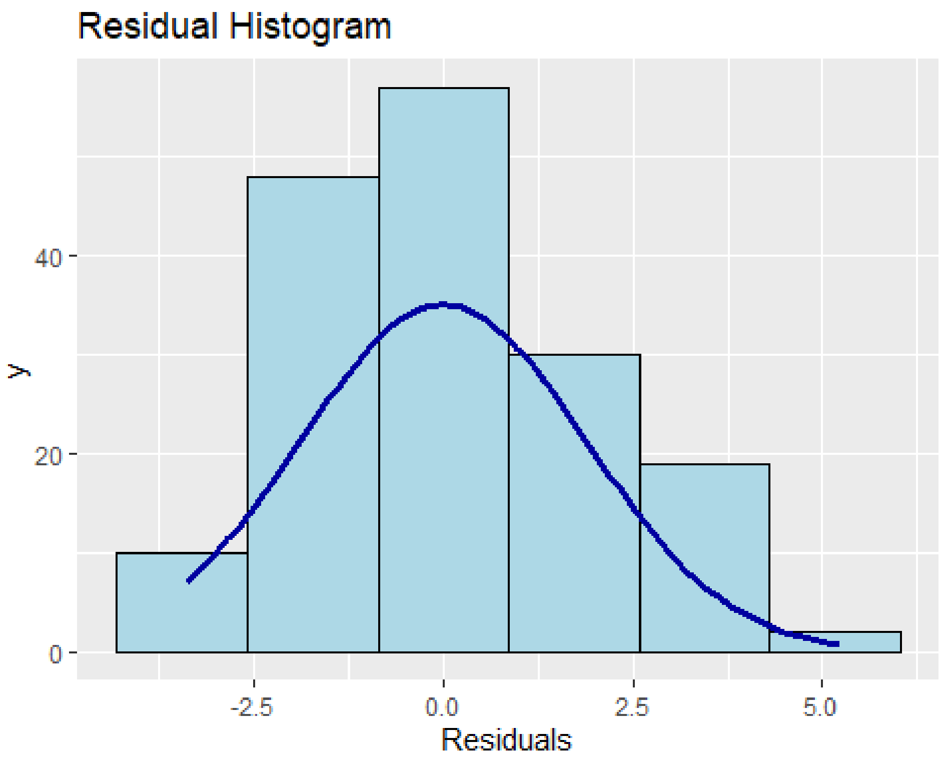


Figure 131 - Histogram plot of residuals.

```
ols_plot_resid_qq(ML_regressor)
```

Figure 132 - Code for QQ Plot of Residuals.

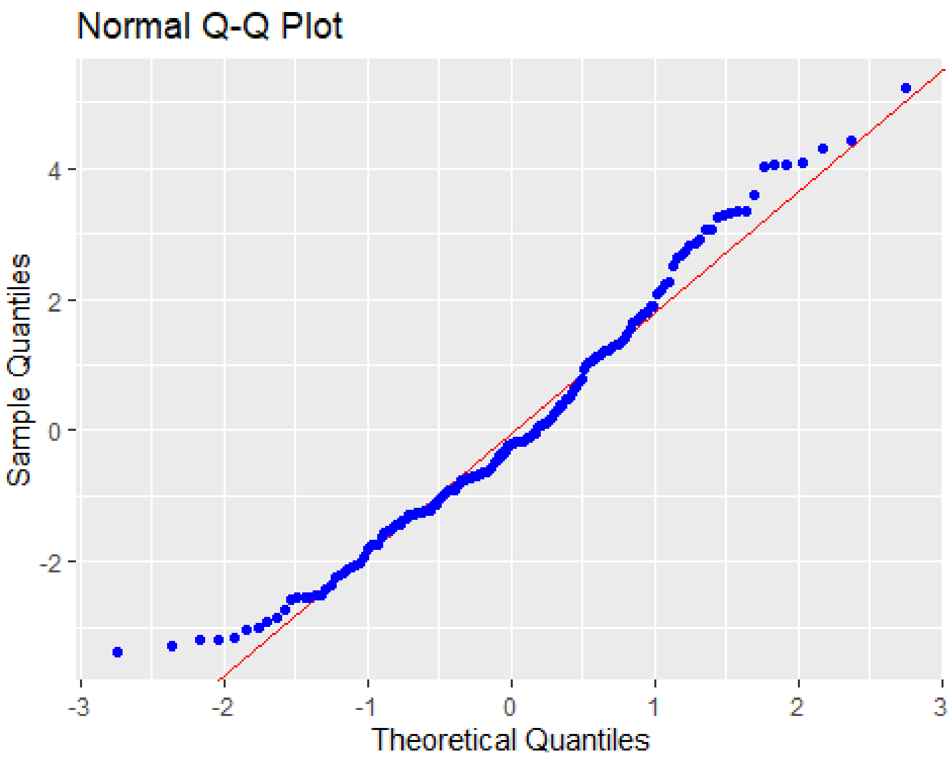


Figure 133 - Plot showing how far the actual values are to regression line.

```
shapiro.test(ML_regressor$residuals)

##
##  Shapiro-Wilk normality test
##
## data:  ML_regressor$residuals
## W = 0.97232, p-value = 0.002088
```

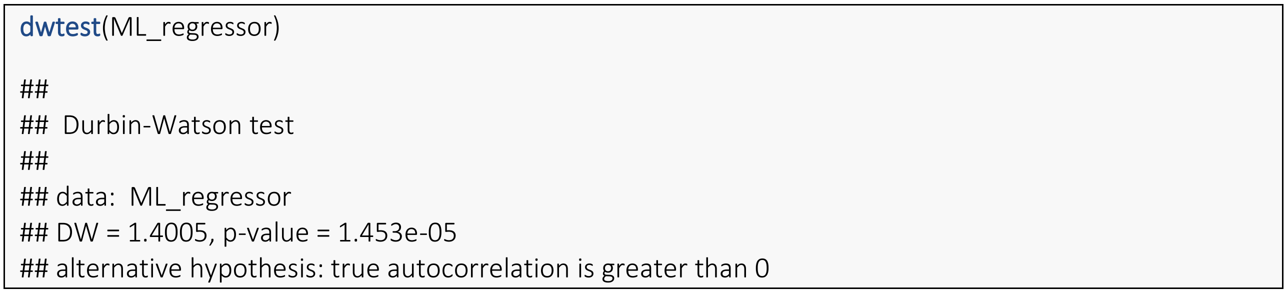
Figure 134 - Code for Shapiro test on residuals.

Null Hypothesis: Residuals are normally distributed.
Alternative hypothesis: Residuals are not normally distributed.

p-value is less than 0.05, therefore, null hypothesis that the errors are normally distributed is rejected. Therefore, the alternative hypothesis is accepted, and the error/residual is not normally distributed. The Shapiro test is the confirmation needed to see if the residual is normally distributed or not.

7.2.4 Check for Autocorrelation of Error

This is when errors from regression model are correlated with each other. This will be done using Durbin-Watson Test.



```
dwtest(ML_regressor)

##
##  Durbin-Watson test
##
## data:  ML_regressor
## DW = 1.4005, p-value = 1.453e-05
## alternative hypothesis: true autocorrelation is greater than 0
```

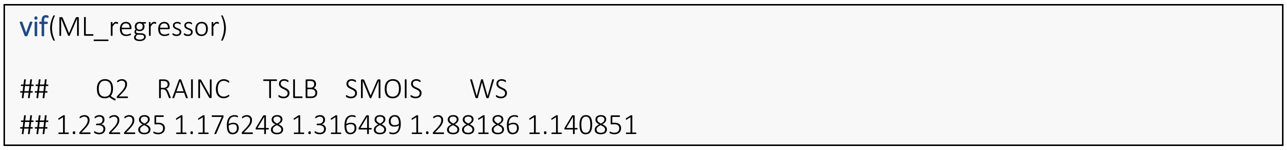
Figure 135 - Code to check for autocorrelation of error.

Null Hypothesis: There is no autocorrelation of error.
Alternative Hypothesis: There is autocorrelation of error.

Since p-values is < 0.05, the NULL hypothesis of No Autocorrelation is rejected, then, there is autocorrelation.

7.2.5 Multicollinearity

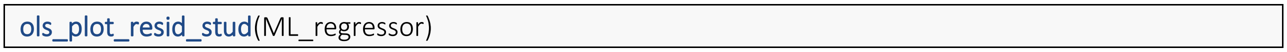
Multicollinearity occurs when two or more independent variables are linearly dependent on each other. Using Variation Inflation Factor (VIF), a VIF factor greater than 5 or 10 indicates multicollinearity.



```
vif(ML_regressor)

##       Q2    RAINC     TSLB    SMOIS       WS
## 1.232285 1.176248 1.316489 1.288186 1.140851
```

Figure 136 - Code for multicollinearity.

7.2.6 Identify Outliers



```
ols_plot_resid_stud(ML_regressor)
```

Figure 137 - Code to Identify Outliers from residuals.

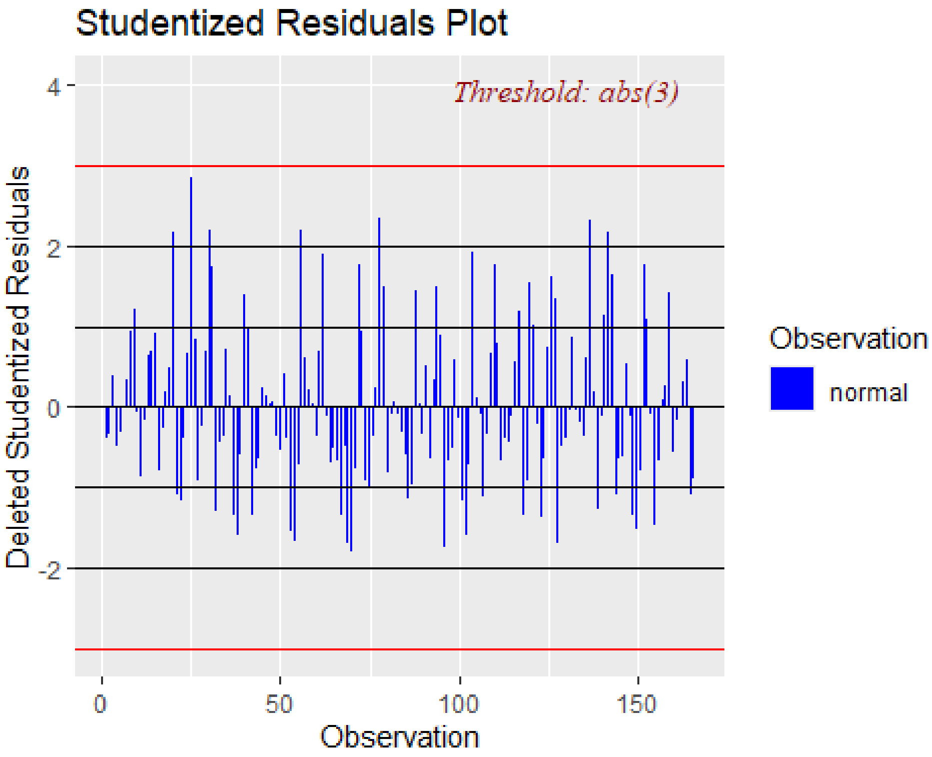


Figure 138 - Studentised Residuals Plot

There are no outliers as no point cross above the threshold line.

7.2.6 Cook distance plot

```
ols_plot_cooksd_chart(ML_regressor)
```

Figure 139 - Code for Cook's distance plot.

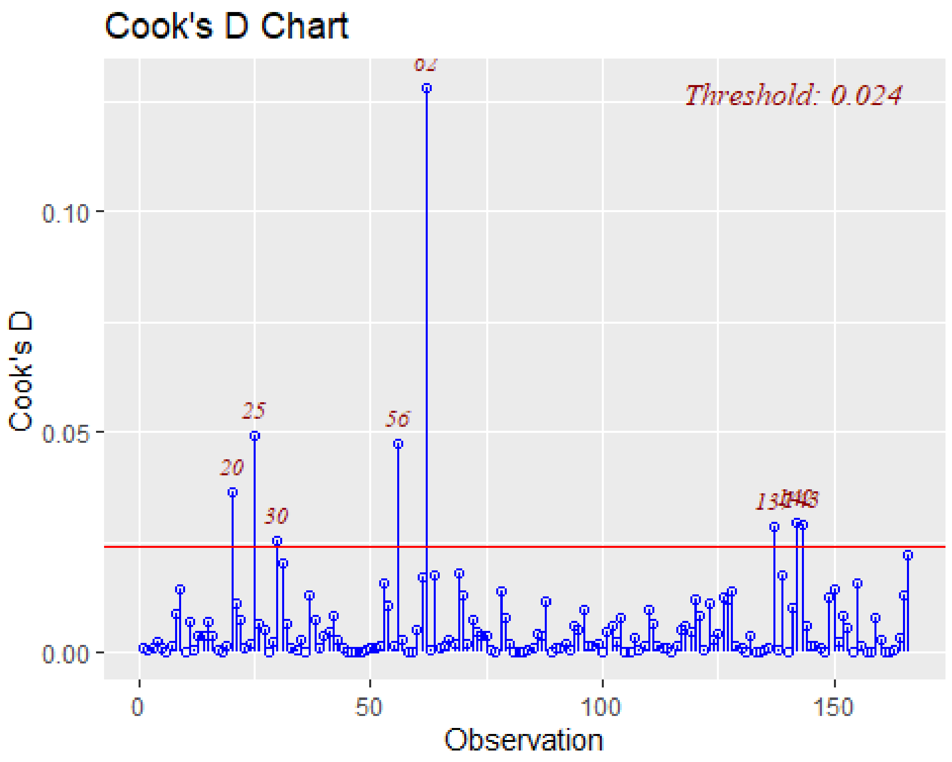


Figure 140 - Cook's Distance Plot

7.2.7 Outlier and leverage plot

```
ols_plot_resid_lev(ML_regressor)
```

Figure 141 - Code for outliers and leverage plot.

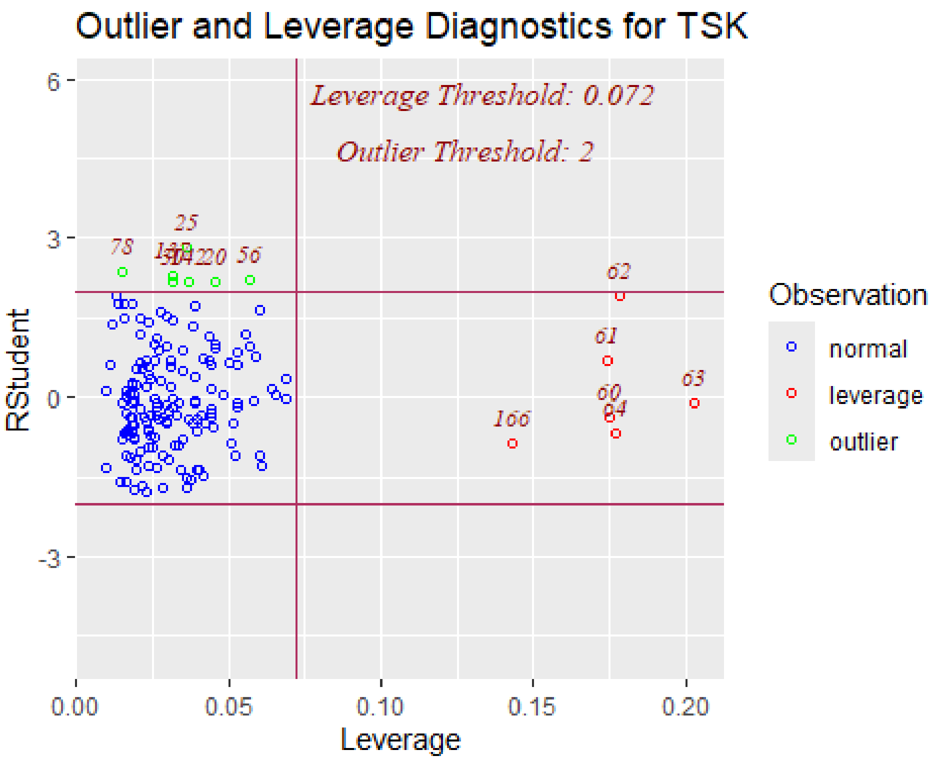


Figure 142 - Outliers and leverages diagnostics.

7.2.8 Observation with strongest influence

```
ols_plot_dfbetas(ML_regressor)
```

Figure 143 - Code for observation and influence.

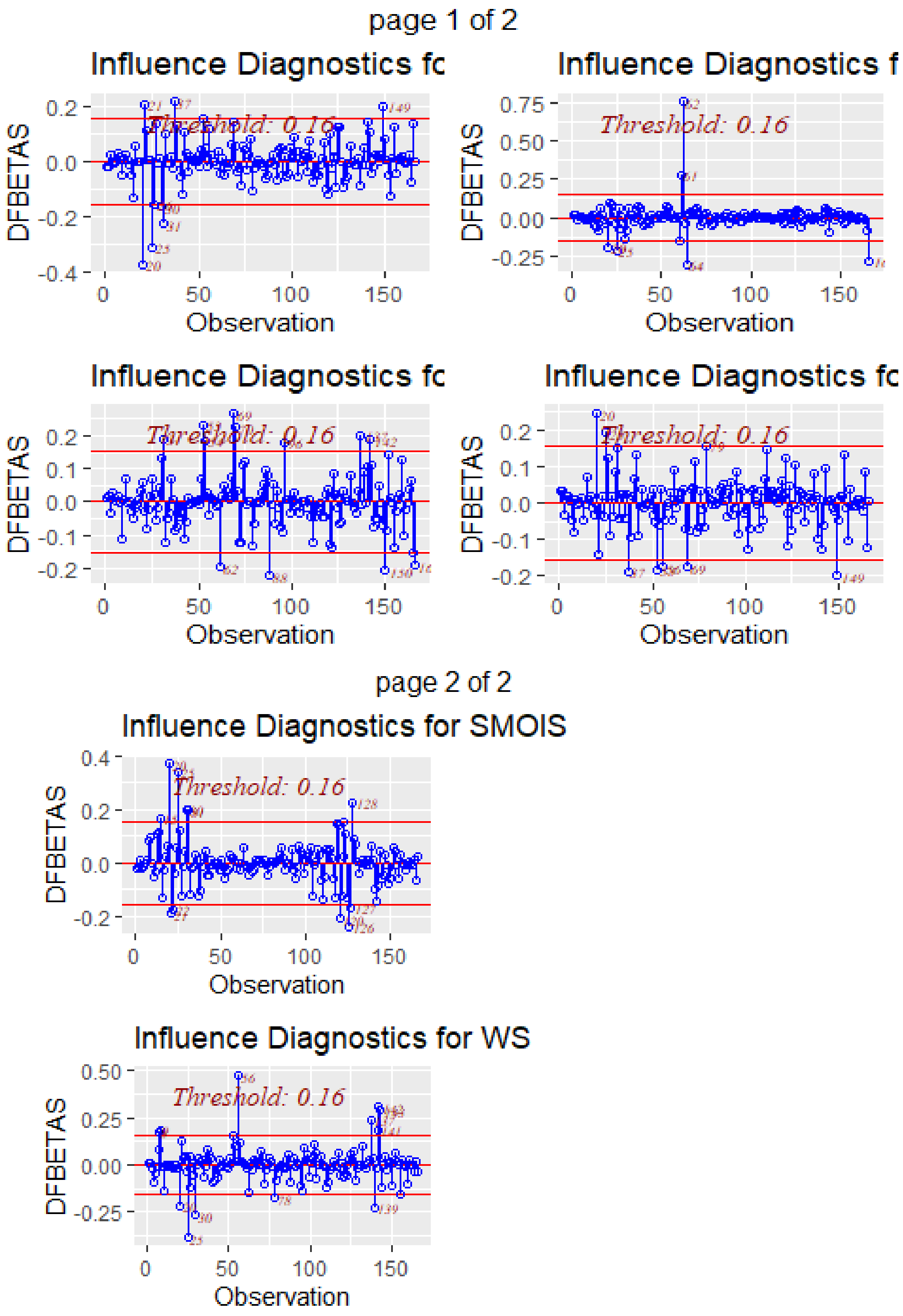


Figure 144 – Plots for Influence Diagnostics

7.2.9 Prediction and Error Calculation

```
predicted_values <- predict(ML_regressor, newdata = testing_df[, !names(testing_df) %in% c("TSK")])
actual_values <- testing_df$TSK
rmse <- sqrt(mean((actual_values - predicted_values)^2))
cat("The root mean square error for linear regression is: ", rmse)

## The root mean square error for linear regression is:  2.01032
```

Figure 145 - Code for prediction with Linear Regression and RMSE calculation.

7.3 SVR model

The second machine learning model will be the SVR model, and the implementation is as done below:

```
install.packages("e1071", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'e1071' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'e1071'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying
## C:\Users\Emmanuel
## Oloyede\AppData\Local\R\win-library\4.2\00LOCK\e1071\libs\x64\e1071.dll to
## C:\Users\Emmanuel
## Oloyede\AppData\Local\R\win-library\4.2\e1071\libs\x64\e1071.dll: Permission
## denied

## Warning: restored 'e1071'

##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpwTzaFq\downloaded_packages

library("e1071")

## Warning: package 'e1071' was built under R version 4.2.3
```

Figure 146 - Code to install required package for SVR model.

7.3.1 Preprocessing for the Model

Dividing into X_train, Y_train, X_test, Y_test and standardising (Scaling) the input: Standardising must be done for other model except linear regression because it is done automatically but not so for other models.

```
X_train <- training_df[, !names(training_df) %in% c("TSK")]
Y_train <- training_df$TSK

X_test <- testing_df[, !names(testing_df) %in% c("TSK")]
Y_test <- testing_df$TSK

X_train <- scale(X_train)
X_test <- scale(X_test)
```

Figure 147 - Code that separate training and testing dataset and standardise it.

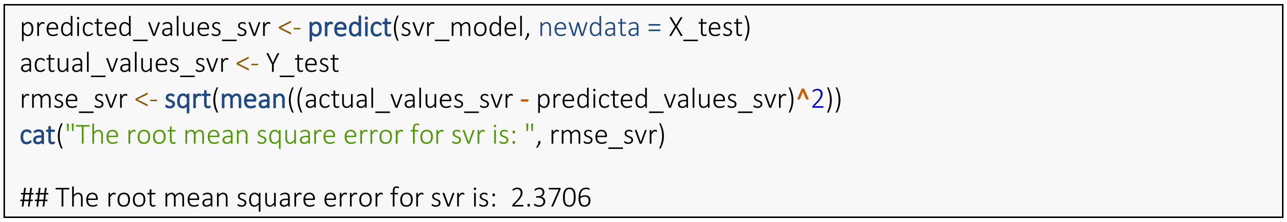
7.3.2 Fitting the data into the model.

After standardisation, the data can now be fit into the SVR model.

```
svr_model <- svm(Y_train ~ ., data = X_train, kernel = "radial")
print(svr_model)

##
## Call:
## svm(formula = Y_train ~ ., data = X_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.2
##     epsilon:  0.1
##
##
## Number of Support Vectors:  125
```

Figure 148 - Code for SVR model.

7.3.3 Predict using the SVR model.

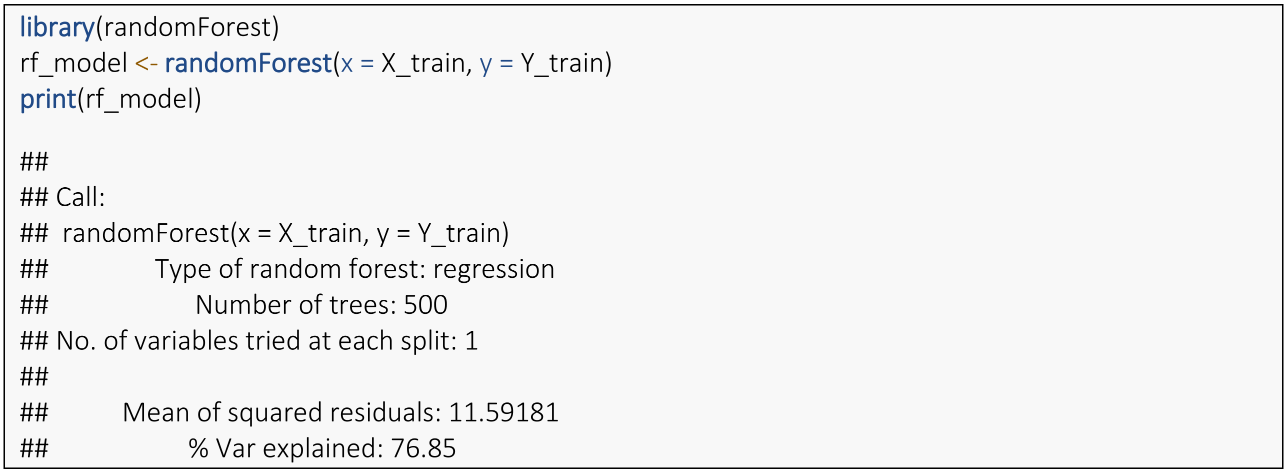


```
predicted_values_svr <- predict(svr_model, newdata = X_test)
actual_values_svr <- Y_test
rmse_svr <- sqrt(mean((actual_values_svr - predicted_values_svr)^2))
cat("The root mean square error for svr is: ", rmse_svr)

## The root mean square error for svr is:  2.3706
```

Figure 149 - Code to predict using SVR model.

7.4 Random Forest Regression

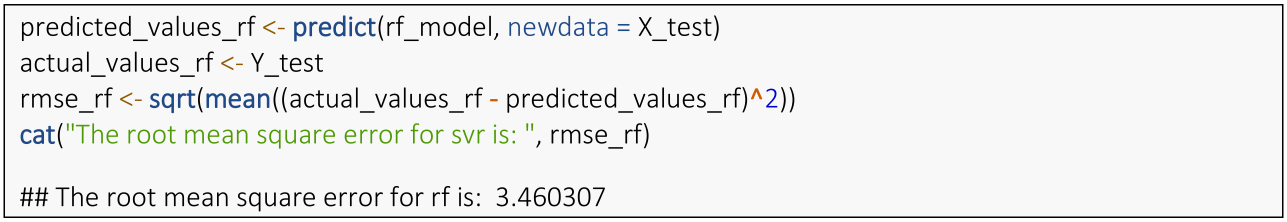
The third model to consider is the Random Forest model and the implementation is as shown below:



```
library(randomForest)
rf_model <- randomForest(x = X_train, y = Y_train)
print(rf_model)

## 
## Call:
##  randomForest(x = X_train, y = Y_train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
## 
##           Mean of squared residuals: 11.59181
##                     % Var explained: 76.85
```

Figure 150 - Code that install library needed for Random Forest Model.

7.4.1 Predicting and calculating the error of the model:

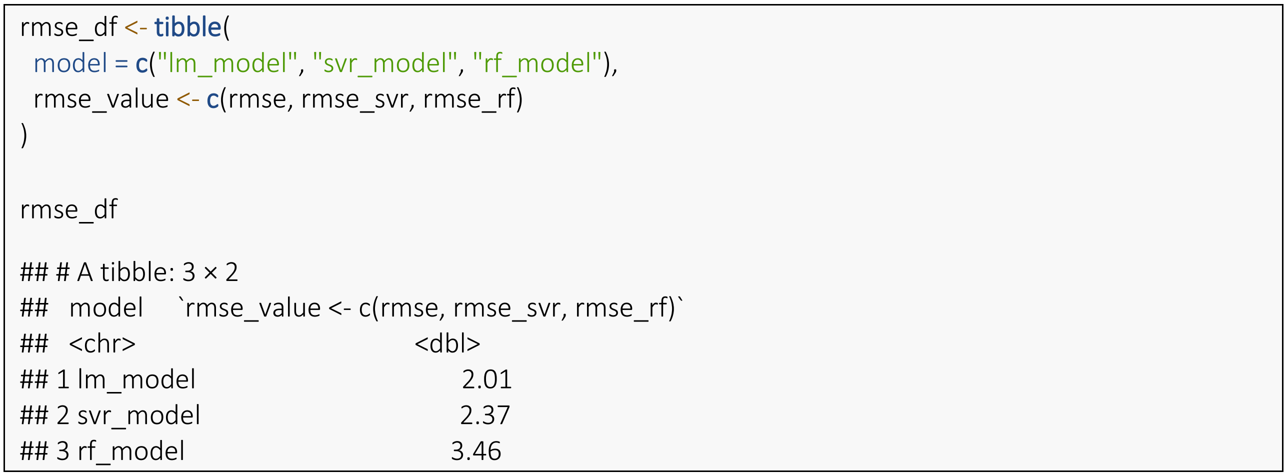


```
predicted_values_rf <- predict(rf_model, newdata = X_test)
actual_values_rf <- Y_test
rmse_rf <- sqrt(mean((actual_values_rf - predicted_values_rf)^2))
cat("The root mean square error for svr is: ", rmse_rf)

## The root mean square error for rf is:  3.460307
```

Figure 151 - Code for predicting using random forest model.

7.5 Selecting the Best Model

Collating and selecting the best model with least error:



```
rmse_df <- tibble(
  model = c("lm_model", "svr_model", "rf_model"),
  rmse_value <- c(rmse, rmse_svr, rmse_rf)
)

rmse_df

## # A tibble: 3 × 2
##   model     `rmse_value <- c(rmse, rmse_svr, rmse_rf)`
##   <chr>                                <dbl>
## 1 lm_model                              2.01
## 2 svr_model                             2.37
## 3 rf_model                              3.46
```

Figure 152 - Code that tabulate the RMSE of all the models built.

Table 2 - Models and its resultant errors

Models	Root Mean Square Error
Linear Regression	2.01
Support Vector Regression	2.37
Random Forest Regression	3.46

From the table above, the linear regression has the least error which is 2.003179 and it is said to be the best model, followed by the support vector regression and random forest in that order.

8.0 TIMESERIES ANALYSIS

8.1 Extracting the Date-time and TSK column.

The skin temperature TSK and date-time column will be extracted into a new data frame for the time series analysis. This is achieved using the code:

```
ts_data <- df_warrington[, c("Datetime", "TSK")]
head(ts_data)

##            Datetime   TSK
## 1 2018-05-01 00:00:00 278.3
## 2 2018-05-01 03:00:00 277.1
## 3 2018-05-01 06:00:00 278.1
## 4 2018-05-01 09:00:00 288.5
## 5 2018-05-01 12:00:00 292.9
## 6 2018-05-01 15:00:00 286.8
```

Figure 153 - Code to extract the TSK and DATETIME column.

8.2 Converting the Extracted Data to Timeseries data.

The data is recorded every 3 hours of the day. Therefore, to convert to a time series data, the start time will be 1 while the frequency will be 8. This is because, 8 records are taken per day.

```
#Converting the data to time-series data
ts_df <- ts(ts_data$TSK, start = 1, frequency = 8)
```

Figure 154 - Code that convert the extracted data to timeseries data.

8.3 Timeseries Visualisation

Plotting the time series to visualise it.

```
plot(ts_df, ylab="Temperature")
```

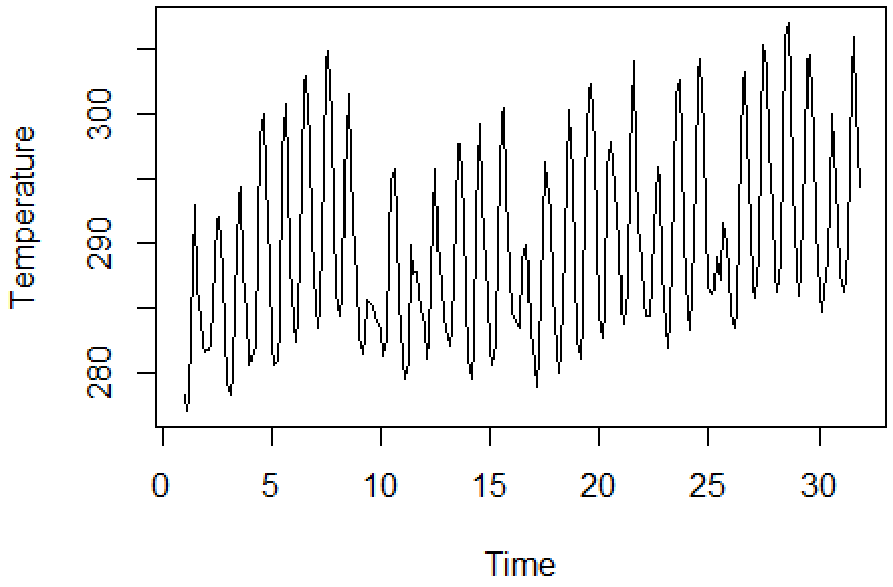


Figure 155 - Temperature trend per time.

The visualisation shows a seasonal trend. Then, the seasonal decompose is plotted as shown below:

```
plot(decompose(ts_df))
```

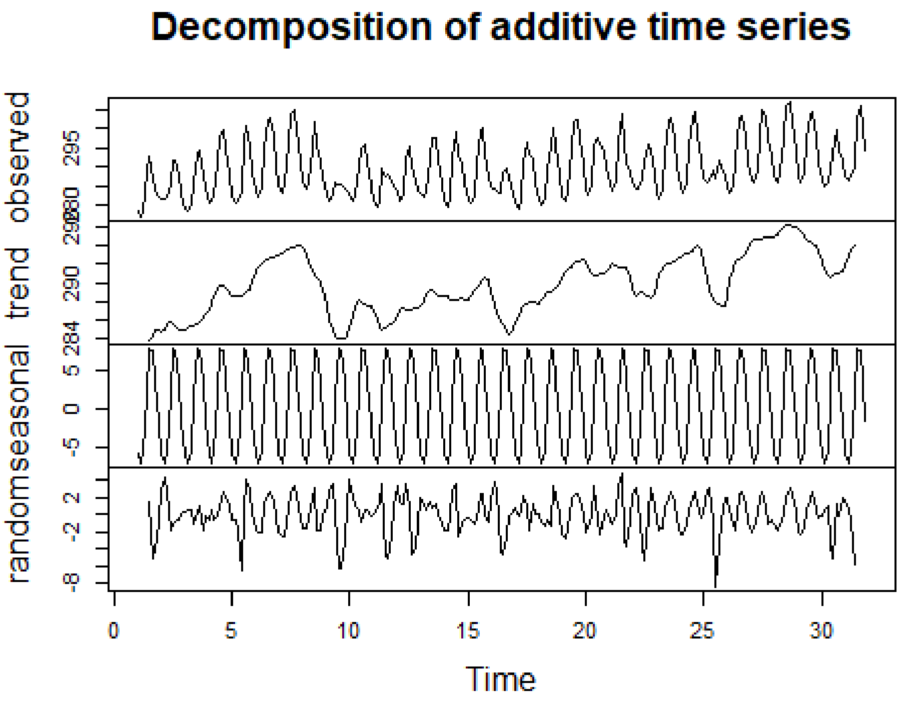


Figure 156 - Timeseries graph with random, observed, seasonal and trend components.

8.4 Checking for Outliers

Checking for outliers in the time series data

```
ts_outliers<-tsoutliers(ts_df)
ts_outliers

## $index
##  [1]  36  69  70  71  85  86 124 125 173 197 198 237
##
## $replacements
##  [1] 290.1302 294.1666 294.1824 290.1606 295.8140 292.8993 290.4034 295.3009
##  [9] 294.9411 295.8689 295.6040 297.8025
```

Figure 157 - Code for checking outliers in timeseries.

The time series outlier's detection function finds 12 values to be outliers. The tsoutliers function also handles these outliers by replacing them with the values shown in the output of code in figure 157.

8.5 Testing the Stationarity of the data.

Before the data will be fitted into timeseries model, there is a need to test its stationarity. This occurs when the mean, variance and autocovariance remain the same over time.

```
acf(ts_df, xlab = "Lag", main = "ACF plot for TSK")
```

Figure 158 - Code for ACF Plot.

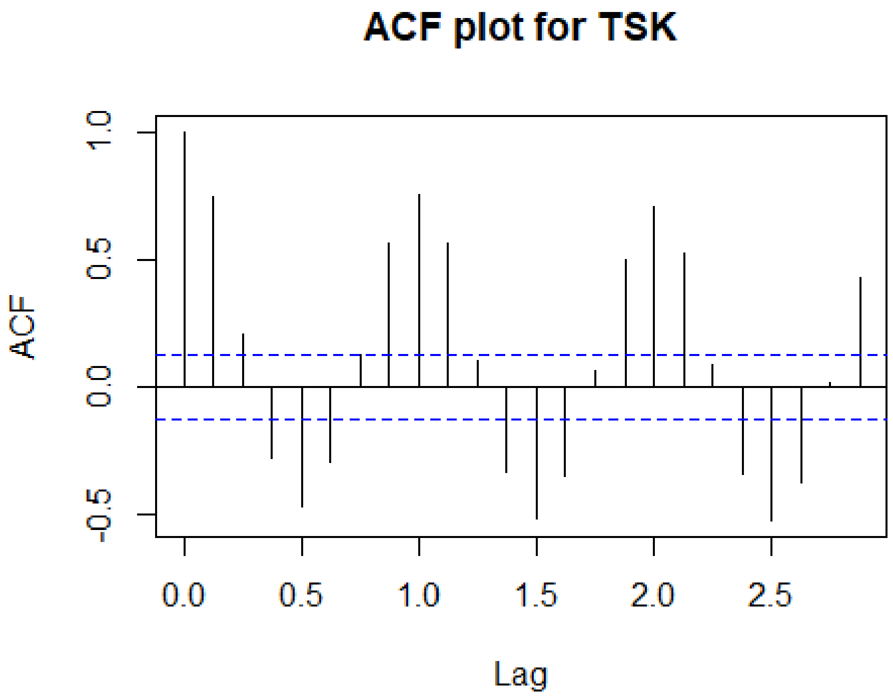


Figure 159 - ACF plot for the timeseries.

```
pacf(ts_df, main = "PACF plot for TSK")
```

Figure 160 - Code for PACF Plot.

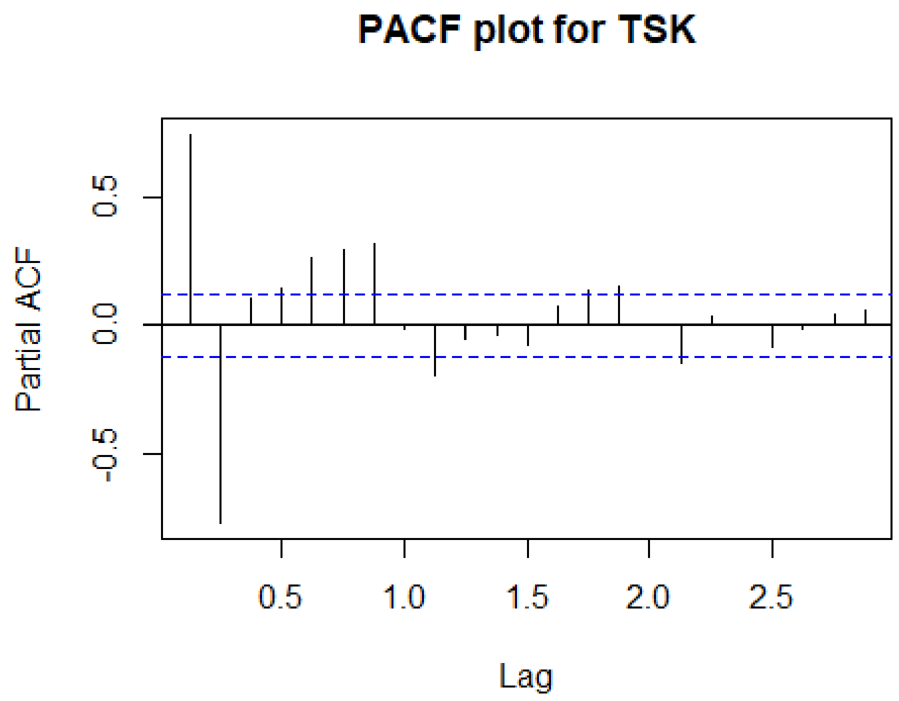


Figure 161 - PACF plot of timeseries data.

```
install.packages("vars", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'vars' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpwTzaFq\downloaded_packages

library(vars)

## Warning: package 'vars' was built under R version 4.2.3

## Loading required package: MASS

## Warning: package 'MASS' was built under R version 4.2.3

##
## Attaching package: 'MASS'

## The following object is masked from 'package:olsrr':
##
##     cement
## The following object is masked from 'package:bestNormalize':
##
##     boxcox

## The following object is masked from 'package:dplyr':
##
##     select

## Loading required package: strucchange

## Warning: package 'strucchange' was built under R version 4.2.3

## Loading required package: sandwich

## Warning: package 'sandwich' was built under R version 4.2.3

##
## Attaching package: 'strucchange'

## The following object is masked from 'package:stringr':
##
##     boundary

## Loading required package: urca

## Warning: package 'urca' was built under R version 4.2.3

station_test <- ur.df(ts_df)
summary(station_test)

##
## ###############################################
## # Augmented Dickey-Fuller Test Unit Root Test #
## ###############################################
##
## Test regression none
##
##
## Call:
## lm(formula = z.diff ~ z.lag.1 - 1 + z.diff.lag)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5602  -2.9693   0.2262   2.3339  11.9221
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## z.lag.1    -0.0001663  0.0009314  -0.179    0.858
## z.diff.lag  0.5480973  0.0538081  10.186   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.233 on 244 degrees of freedom
## Multiple R-squared:  0.2984, Adjusted R-squared:  0.2926
## F-statistic: 51.88 on 2 and 244 DF,  p-value: < 2.2e-16
##
##
## Value of test-statistic is: -0.1785
##
## Critical values for test statistics:
##      1pct  5pct 10pct
## tau1 -2.58 -1.95 -1.62
```

Figure 162 - Code for installing VAR library and performing ADF test.

From the summary, the p-value presently is 0.858 which implies that NULL hypothesis of non-stationarity is true. Therefore, the data is presently not stational and will need to be differentiated. The adf test is also applied to confirm this as shown below:

```
adf.test(ts_df)

##
##  Augmented Dickey-Fuller Test
##
## data:  ts_df
## Dickey-Fuller = -2.9162, Lag order = 6, p-value = 0.1901
## alternative hypothesis: stationary
```

Figure 163 - ADF test code to check for stationarity.

p-value is 0.1901, therefore, the data is not stational.

8.5.1 Differentiating the data.

```
diff1_tsdf <- diff(ts_df)
```

Figure 164 - Code to perform first differentiation.

Checking the stationarity after the first differentiation:

```
adf.test(diff1_tsdf)

## Warning in adf.test(diff1_tsdf): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  diff1_tsdf
## Dickey-Fuller = -11.248, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

Figure 165 - Code that perform stationarity test on differentiated data.

After the first differentiation, the p-value is now 0.01 which is less than 0.05. Then, the NULL hypothesis of non-stationarity will be rejected. The data is now stational and can be fitted for the timeseries.

8.5.1 Checking the ACF and PACF plot after differentiating.

```
acf(diff1_tsdf, xlab = "Lag", main = "ACF plot for TSK after differentiating")
```

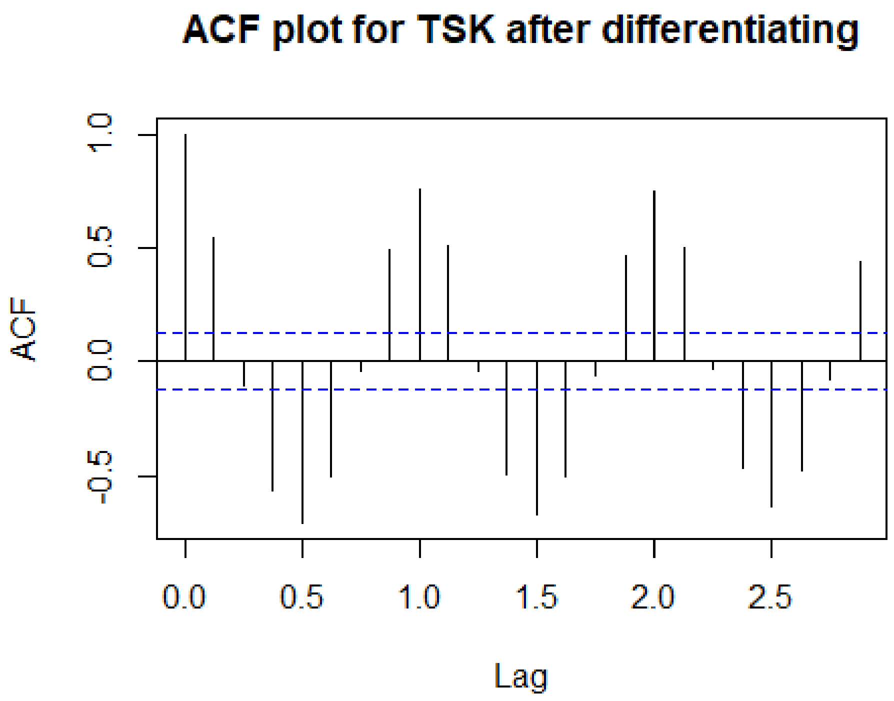


Figure 166 - ACF plot of the differentiated data.

```
pacf(diff1_tsdf, main = "PACF plot for TSK after differentiating")
```

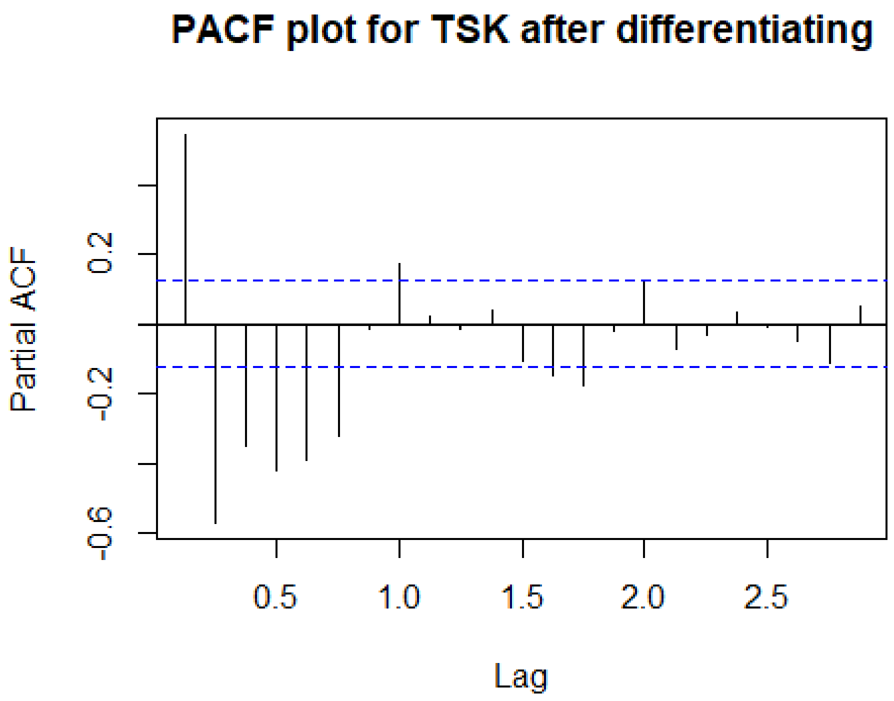


Figure 167 - PACF plot of the differentiated data.

There are six lines passing over to the significant area in the PACF plot, therefore, our (AR) p = (1,2,3,4,6). The differentiation was done only once to achieve stationarity, therefore, (I) d = 1. The q value is gotten from the ACF plot. The ACF plot shows a sinusoidal (sine wave) pattern. The wave was completed 2 times (2 complete cycle) in the graph; therefore, the MA (q) will be 2. This will be an information to be used in building several Arima models and selecting the optima in comparison to the Auto Arima

8.5 Splitting into Training and Testing Dataset

The dataset will be split in the ratio of training: testing = 80:20. Therefore, out of 31 days, the training dataset will have the values from 1st day to the 25th day while the testing dataset takes the 26th day to the last day 31st of the month.

```
tsdf_train <- window(diff1_tsdf, end = c(25, 8))
tsdf_train

## Time Series:
## Start = c(1, 2)
## End = c(25, 8)
## Frequency = 8
##   [1] -1.2  1.0 10.4  4.4 -6.1 -2.6 -1.9 -0.7  0.2  0.4  3.2  6.5  0.1 -4.2 -4.7
##  [16] -3.8 -1.0  1.6  5.8  7.7  0.9 -6.5 -3.1 -4.1  0.5  1.0  8.4  7.8  1.6 -4.8
##  [31] -6.8 -6.3 -1.4  0.4  2.6 13.3  3.8 -5.1 -7.1 -4.6 -1.5  1.8 10.8  7.4  0.5
##  [46] -4.9 -7.3 -5.5 -1.8  1.8 10.6  8.0  1.0 -5.2 -7.6 -5.7 -1.9  1.6  8.8  6.8
##  [61] -5.4 -4.4 -4.4 -4.5 -1.5  1.3  3.0 -0.4 -0.1 -0.8 -0.8 -0.2 -2.1  1.4  7.3
##  [76]  4.8  1.0 -4.2 -6.0 -4.5 -1.6  1.4  9.0 -2.3  0.2 -1.5 -1.3 -1.3 -2.5  1.4
##  [91]  9.6  3.5 -5.5 -3.5 -2.3 -0.9 -1.5  1.5  8.6  5.5  0.1 -5.5 -6.2 -4.9 -1.6
## [106]  2.5 10.3  6.9 -5.8 -2.7 -4.8 -4.0 -1.3  1.7  8.2  9.4  0.5 -6.4 -6.3 -2.8
## [121] -0.6 -0.5 -0.4  5.2  1.3 -2.2 -4.4 -2.7 -1.6  2.6  9.5  5.2 -2.7 -1.1 -6.4
## [136] -4.2 -1.9  1.6  8.2  5.7  4.8 -5.1 -6.9 -5.8 -1.3  2.8 10.9  7.0  0.4 -4.8
## [151] -6.7 -6.4 -1.7  2.4 10.6  2.1 -1.6 -1.8 -4.4 -4.8 -1.4  2.4 10.4  7.5 -7.5
## [166] -5.2 -2.8 -2.9 -1.3  0.0  2.6  4.1  4.8 -1.1 -6.8 -4.4 -1.7  2.4  9.7  7.6
## [181]  1.0 -5.0 -7.3 -5.6 -1.4  2.9  9.1  7.7  1.2 -5.4 -7.2 -4.8 -0.7  0.4  2.4
## [196] -1.7  4.4 -1.6 -2.2

tsdf_test <- window(diff1_tsdf, start = c(26, 1))
tsdf_test

## Time Series:
## Start = c(26, 1)
## End = c(31, 8)
## Frequency = 8
##  [1] -3.1 -1.2  1.2  7.2 10.5  0.8 -5.1 -6.7 -4.8 -0.8  3.2  9.1  7.1 -1.6 -6.3
## [16] -4.7 -4.7 -1.7  2.4  9.1  8.2  1.1 -5.0 -7.4 -6.6 -2.0  2.0  8.8  7.0  0.7
## [31] -6.3 -6.9 -5.0 -1.6  1.5  2.7  4.4  6.6 -5.4 -3.0 -3.8 -1.4  1.0  2.7 12.9
## [46]  3.0 -5.3 -6.3
```

Figure 168 - Code that separate differentiated timeseries data to training and testing set.

8.6 Auto Arima

The auto-Arima finds the optimal model using the auto-Arima function as shown in the code snippet below:

```
a_tsdf_train <- window(ts_df, end = c(25, 8))
a_tsdf_test <- window(ts_df, start = c(26, 1))
auto_arima_model <- auto.arima(a_tsdf_train)

auto_arima_model

## Series: a_tsdf_train
## ARIMA(2,0,2)(1,1,0)[8] with drift
##
## Coefficients:
##          ar1     ar2     ma1     ma2     sar1    drift
##       0.0775  0.2520  1.0060  0.3433  -0.4900  0.0303
## s.e.  0.3940  0.2493  0.3908  0.1950   0.0665  0.0537
##
## sigma^2 = 6.539:  log likelihood = -451.31
## AIC=916.61   AICc=917.22   BIC=939.41

summary(auto_arima_model)

## Series: a_tsdf_train
## ARIMA(2,0,2)(1,1,0)[8] with drift
##
## Coefficients:
##          ar1     ar2     ma1     ma2     sar1    drift
##       0.0775  0.2520  1.0060  0.3433  -0.4900  0.0303
## s.e.  0.3940  0.2493  0.3908  0.1950   0.0665  0.0537
##
## sigma^2 = 6.539:  log likelihood = -451.31
## AIC=916.61   AICc=917.22   BIC=939.41
##
## Training set error measures:
##                      ME     RMSE      MAE         MPE      MAPE      MASE
## Training set 0.01578869 2.466109 1.731381 0.001033953 0.5961141 0.5294238
##                     ACF1
## Training set 0.006148734
```

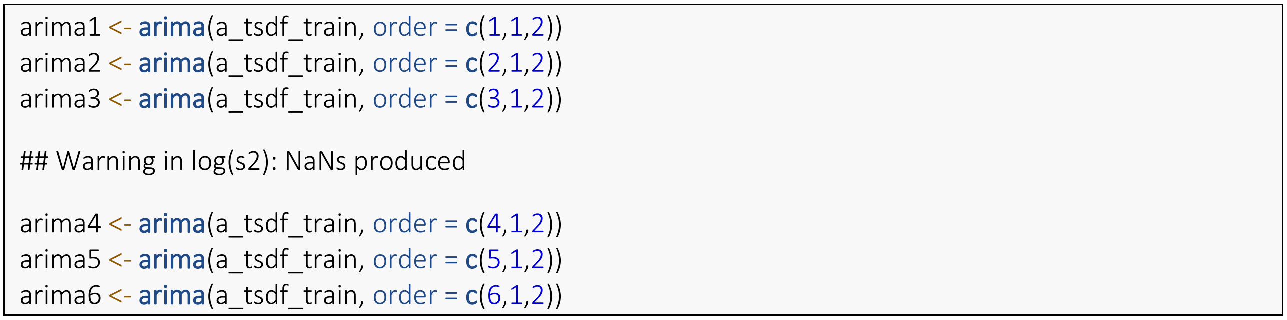
Figure 169 - Code that run auto Arima model.

From the Auto-Arima, optimum is gotten from (p, d, q) value of (2,0,2). However, for the manual Arima model, the (p,d,q) values will be deduced from the PACF and ACF graphs. Several models will be built and the best with lowest AIC will be chosen and compared with the auto Arima model.

8.7 ARIMA model

Here, several model will be built to select the best model using AIC. The model with the least AIC will be taunted as the best model. From the PACF plot, the p value will be selected.

Therefore, six manual Arima model will be built for order (p,d,q) = (1,1,2); (2,1,2); (3,1,2); (4,1,2), (5,1,2); (6,1,2). The model with the lowest AIC will be chosen as the best model.



```
arima1 <- arima(a_tsdf_train, order = c(1,1,2))
arima2 <- arima(a_tsdf_train, order = c(2,1,2))
arima3 <- arima(a_tsdf_train, order = c(3,1,2))

## Warning in log(s2): NaNs produced

arima4 <- arima(a_tsdf_train, order = c(4,1,2))
arima5 <- arima(a_tsdf_train, order = c(5,1,2))
arima6 <- arima(a_tsdf_train, order = c(6,1,2))
```

Figure 170 - Code that runs various manual Arima.

Checking the AIC of the models to select the optimal model as shown below. The Arima 4 has the lowest AIC, therefore, the summary of the model was viewed as shown.

```
aic_df2 <- tibble(
  model = c("arima1", "arima2", "arima3", "arima4", "arima5", "arima6"),
  aic_value <- c(arima1$aic, arima2$aic, arima3$aic, arima4$aic, arima5$aic, arima6$aic)
)

aic_df2

## # A tibble: 6 × 2
##   model  `aic_value <- ...`
##   <chr>               <dbl>
## 1 arima1              1099.
## 2 arima2               964.
## 3 arima3               964.
## 4 arima4               943.
## 5 arima5               943.
## 6 arima6               946.

summary(arima4)

##
## Call:
## arima(x = a_tsdf_train, order = c(4, 1, 2))
##
## Coefficients:
##          ar1      ar2     ar3      ar4      ma1     ma2
##       1.4717  -1.4688  0.6130  -0.3893  -1.2994  0.8072
## s.e.  0.0931   0.1417  0.1277   0.0721   0.0701  0.0904
##
## sigma^2 estimated as 6.089:  log likelihood = -464.32,  aic = 942.65
##
## Training set error measures:
##                      ME     RMSE      MAE        MPE      MAPE     MASE
## Training set 0.04436185 2.461393 1.793797 0.01070781 0.6196645 0.450771
##                     ACF1
## Training set -0.02750039
```

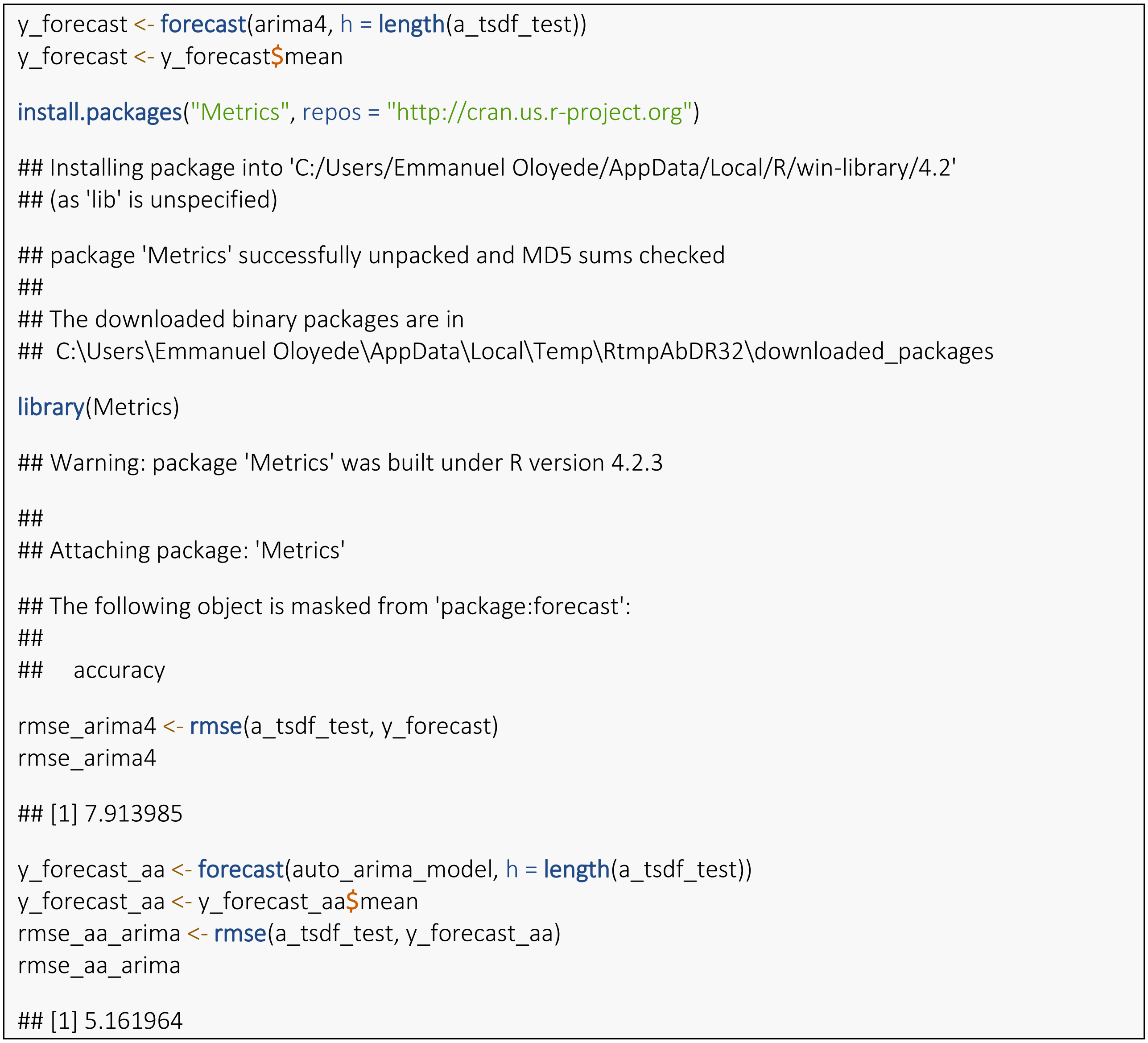
Figure 171 - Code that check AIC of the Manual Arima models.

Table 3 - AIC of Arima Models.

ARIMA MODEL	AIC
Arima1	1099.32
Arima2	963.75
Arima3	963.77
Arima4	942.64
Arima5	942.77
Arima6	946.31

The auto Arima gave an AIC value of 916.6 while the optimum model for manual ARIMA gave an AIC value of 942.64. Therefore, the Auto Arima will be selected for prediction.

8.8 Predict with Arima4 and Auto Arima.



```
y_forecast <- forecast(arima4, h = length(a_tsdf_test))
y_forecast <- y_forecast$mean

install.packages("Metrics", repos = "http://cran.us.r-project.org")

## Installing package into 'C:/Users/Emmanuel Oloyede/AppData/Local/R/win-library/4.2'
## (as 'lib' is unspecified)

## package 'Metrics' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\Emmanuel Oloyede\AppData\Local\Temp\RtmpAbDR32\downloaded_packages

library(Metrics)

## Warning: package 'Metrics' was built under R version 4.2.3

##
## Attaching package: 'Metrics'

## The following object is masked from 'package:forecast':
##
##     accuracy

rmse_arima4 <- rmse(a_tsdf_test, y_forecast)
rmse_arima4

## [1] 7.913985

y_forecast_aa <- forecast(auto_arima_model, h = length(a_tsdf_test))
y_forecast_aa <- y_forecast_aa$mean
rmse_aa_arima <- rmse(a_tsdf_test, y_forecast_aa)
rmse_aa_arima

## [1] 5.161964
```

Figure 172 - Code that predicts and calculate the rmse of the models.

Auto Arima has a lower AIC and lower RMSE, therefore, it is a better model and will be deployed to forecast the future temperature.

Table 4 - Summary of the best two models.

Timeseries Model	AIC	RMSE
Auto Arima	916.61	5.161964
Arima4	942.64	7.913985

8.8.1 Check Residual

Checking for the residual of the selected ARIMA model is as shown below.

```
checkresiduals(auto_arima_model)
```

Figure 173 - Code that check for residuals in Arima model.

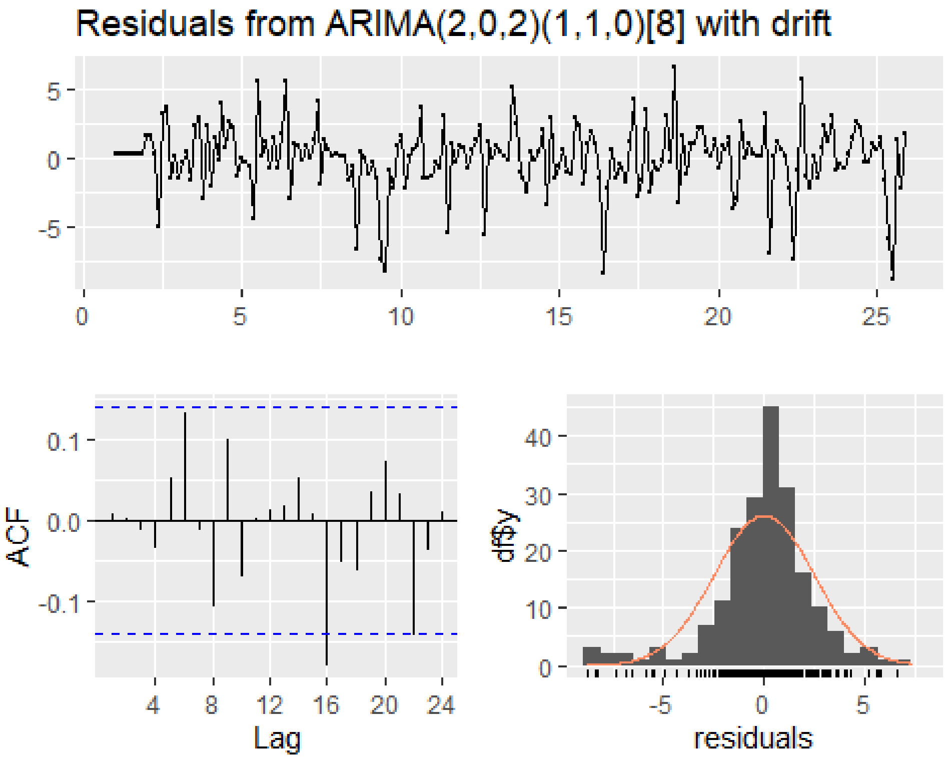


Figure 174 - Residuals of Auto Arima Model.

```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(2,0,2)(1,1,0)[8] with drift
## Q* = 17.622, df = 11, p-value = 0.09077
##
## Model df: 5.   Total lags used: 16
```

8.8.2 Forecasting using Auto Arima.

To forecast for the next month, since the data that was fed was taken at 3 hours interval daily, we will be forecasting a total of 8*30 = 240. Therefore, the code to forecast is written as shown.

```
arimaforecast2 <- forecast(auto_arima_model, h=240)
```

Plotting the forecast:

```
plot(arimaforecast2)
```

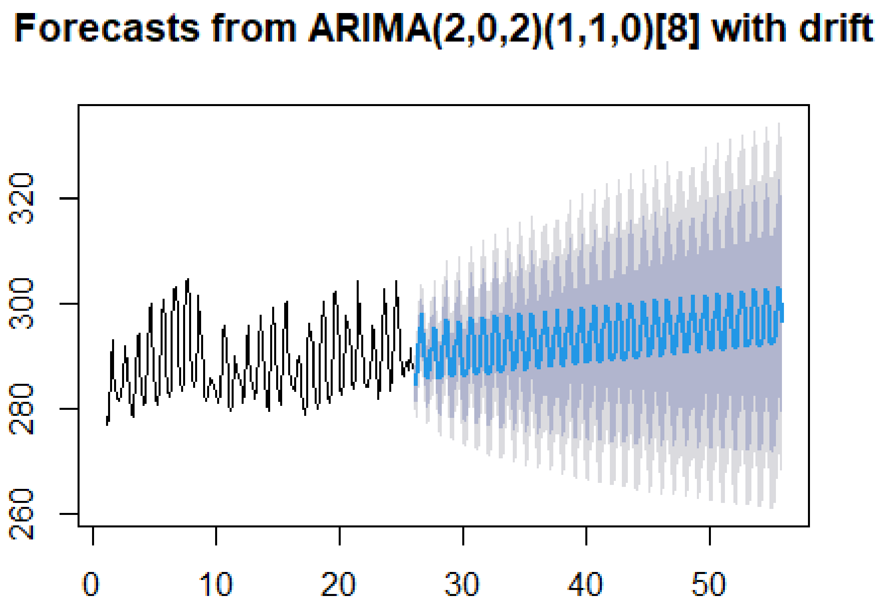


Figure 175 - 240 steps prediction using Auto Arima model.

9.0 DISCUSSION

The analysis above delved into the weather data for the month of May 2014 for Warrington in the United Kingdom. This report will be useful for the stakeholders – residents, government officials and chemical production to make informed decisions. Skin temperature is given bigger preferences because, it is a more common interest to all the identified stakeholders.

Univariate analysis done here include the line chart that checks the trend of average temperature which shows an oscillating temperature pattern. Therefore, the temperature for the month can be said to be very unstable due to the irregular pattern.

Bivariate analysis compares different variables together. Here, the relationship between skin temperature and soil temperature was explored through correlation testing. There is a strong relationship between these variables with correlation coefficient of 0.9. The specific humidity and soil moisture also have correlation; however, the relationship is weak in a negative direction because the correlation coefficient between them is between 0.1 and 0.3. Lastly, under bivariate, how wind speed affects the skin temperature was considered and the relationship between them is also weak in positive direction with their correlation coefficient being between 0.1 and 0.3.

For multivariate analysis, this is a vital part of the analysis because, not only does it allow deep understanding of how every variable relate with the targeted output variable, but the use of heatmap with correlation also serve as the feature selection technique for the machine learning section. Furthermore, regression analysis was also used to know the significance level of each variable on the output variable (TSK)

Machine learning is an important aspect, and three machine learning were built. The feature selection done in multivariate analysis helped the machine learning model to come out with high accuracy with the root mean square error recording for the models are not even up to 3. For the linear multiple regression, various tests were done like check for linearity, autocorrelation of error, heteroscedasticity, normality of error, Cook's distance plot, Outlier, and leverage plot. The dataset met some criteria and failed some. However, the test for linearity which is the essential test was passed. Two other models were built – Random Forest and Support Vector Regression. The linear multiple regression model has the best accuracy having recorded the least root mean square error.

Lastly, time series analysis was also performed to predict the future temperature to help the stakeholders prepare for the future. Auto Arima and Arima models were built, and prediction graph was generated to show the trend of the temperature for the subsequent month. Various visualisations were also done, like PACF and ACF plots. Tests like stationarity was performed and data differentiated once to ensure the time series data was stational. Also, outliers were also inspected and handled with appropriate method which in this case was left alone because these outliers are not far from the remaining data and cannot be outrightly regarded as outliers. The time series analysis concluded the analysis in this report.

10.0 CONCLUSION

The analysis done in this report is vital and essential to understand the pattern in which different weather variables affects the other and what to expect from one variable when other variables change significantly. Also, the analysis in this report can be leveraged on by stakeholders to plan, mitigate risk, properly allocate resources, and optimise their daily activities.

REFERENCES

Bissonette, J.A. (2021) Big Data, Exploratory Data Analyses and Questionable Research Practices: Suggestion for a Foundational Principle. *Wildlife Society Bulletin*, 45(3), pp. 366–370.

Buttrey, S. and Whitaker, L.R. (2017) *A data scientist's guide to acquiring, cleaning, and managing data in R*. *Journal of Statistical Software, 86.*

Delaporte, G., Cladière, M. and Camel, V. (2019) Missing value imputation and data cleaning in untargeted food chemical safety assessment by LC-HRMS. *Chemometrics and Intelligent Laboratory Systems*, 188, pp. 54–62.

Dong, J. *et al.* (2022) Modelling Soil Temperature by Tree-Based Machine Learning Methods in Different Climatic Regions of China. *Applied Sciences*, 12(10), p. 5088.

Dorman, M. *et al.* (2015) Amount vs. temporal pattern: On the importance of intra-annual climatic conditions on tree growth in a dry environment. *Journal of Arid Environments*, 118, pp. 65–68.

Esmaiil, M. *et al.* (2022) Estimation of daily reference evapotranspiration with limited climatic data using machine learning approaches across different climate zones in New Mexico. *Theoretical and Applied Climatology*, 147(1–2), pp. 575–587.

Goloborodko, A.A. *et al.* (2013) Pyteomics—a Python Framework for Exploratory Data Analysis and Rapid Software Prototyping in Proteomics. *Journal of the American Society for Mass Spectrometry*, 24(2), pp. 301–304.

Hao, Z., Singh, V.P. and Xia, Y. (2018) Seasonal Drought Prediction: Advances, Challenges, and Future Prospects. *Reviews of Geophysics*, 56(1), pp. 108–141.

Hsu, H. and Dirmeyer, P.A. (2021) Nonlinearity and Multivariate Dependencies in the Terrestrial Leg of Land-Atmosphere Coupling. *Water Resources Research*, 57(2).

Ichimura, T. and Kamada, S. (2017) Adaptive learning method of recurrent temporal deep belief network to analyze time series data. (2017) *International Joint Conference on Neural Networks (IJCNN)*, pp. 2346–2353.

Jaggia, S. *et al.* (2020) Applying the CRISP-DM Framework for Teaching Business Analytics. *Decision Sciences Journal of Innovative Education*, 18(4), pp. 612–634.

Lang, Z. *et al.* (2023) Forecast of Winter Precipitation Type Based on Machine Learning Method. *Entropy*, 25(1), p. 138.

Obsi, G.D., Hunde, F.D. and Weyessa, G. (2021) Meteorological data trend analysis and local community perception towards climate change: a case study of Jimma City, Southwestern Ethiopia. *Environment, Development and Sustainability*, 23(4), pp. 5885–5903.

Wang, Y., Shen, Z. and Jiang, Y. (2018) Comparison of ARIMA and GM(1,1) models for prediction of hepatitis B in China. *PLoS ONE*, 13(9).

Wen, X. *et al.* (2019) Association among Weather Conditions, Ambient Air Temperature, and Sedentary Time in Chinese Adults. *BioMed Research International*.

Wiemer, H., Drowatzky, L. and Ihlenfeldt, S. (2019) Data Mining Methodology for Engineering Applications (DMME)—A Holistic Extension to the CRISP-DM Model. *Applied Sciences*, 9(12), p. 2407.

Yoon, J.-N. *et al.* (2012) Statistical Analysis on Weather Conditions at Chungbuk National University Observatory in Jincheon, Korea. *Journal of Astronomy and Space Sciences*, 29(4), pp. 397–405.