Table of Contents

1. Background

The quick progress in weather science has boosted our skill to predict weather and grasp environmental states. Yet, it's still tricky to forecast local weather factors like heat, air pressure, and ground wetness. These elements matter a lot for fields such as farming, handling disasters, and public health where spot-on weather forecasts can guide smarter choices.

The Weather Research and Forecasting (WRF) model is a popular tool to simulate weather at different levels, from worldwide to smaller areas. Our project zeros in on Scotland, a place known for its changing climate due to where it sits and its land shape. Getting weather forecasts right in this region is key to get ready for harsh weather and to manage nature's resources well.

2. Problem Statement

Weather factors have a huge impact on our everyday lives and the economy. This project aims to boost how well we can predict key weather and environmental conditions in Scotland using the Weather Research and Forecast(WRF) data.

The primary objective includes:

1. **Predicting Weather Variables:**
   To develop a machine learning models to predict critical weather variables such as surface temperature(TSK), surface pressure(PSFC) and soil moisture(SMOIS) based on the other meteorological inputs like wind speed(U10, V10), humidity(Q2), precipitation(RAINC, RAINCC) and other relevant variable.

2. **Daytime vs Nighttime analysis:**
   To perform statistical analysis, such as T-tests to evaluate the different meteorological variables, such as surface pressure(PSFC) between daytime and nighttime, in order to understand daily variation.

This issue matters because better weather forecasts can help different areas in Scotland. Take farmers, for instance. If they know how warm it'll be and how wet the soil is, they can water their crops just right and plan what to grow. Also, if we can predict air pressure well, we can warn people about bad weather sooner. This project uses smart computer programs to make weather predictions more accurate. It also wants to teach us more about how the weather works in Scotland.

The main aim is to create a weather prediction tool people can count on. Weather experts, scientists, and government officials could use it to make smart choices. This could help Scotland deal with changing weather patterns and keep people safer overall.

3. Exploratory Data Analysis

EDA plays a key role as the first step in any data analysis project. It helps us grasp the underlying patterns, relationships, and distributions within the dataset. In this project, EDA had an impact on summarizing the main statistical properties of the data, showing distributions of weather variables, and spotting links between them. For example, we looked into how factors like temperature at skin level (TSK), surface pressure (PSFC), and soil moisture (SMOIS) interact with other weather elements such as wind speed (U10, V10), humidity (Q2), and rainfall (RAINC RAINNC). EDA proved useful to find possible outliers, check normality, and get a handle on the data's structure. This information guided our choice of modeling approach later on.

3.1 EDA on Raw data

Loading necessary libraries

```
library(dplyr)
library(readr)
library(ggplot2)
library(corrplot)
library(here)
library(stringr)
```

Loading the dataset

```
data <- read_csv("C:/Users/Ahmad Afzal/Desktop/WRFdata_May2023.csv", col_names = FALSE)
```

Assigning second row as a header and removing the second row

Assigning appropriate column names is essential for better data analysis and processing. Removing the row used for column names prevents duplication.

```
# Step 2: Assign the second row as the header
colnames(data) <- as.character(data[2, ])

# Step 3: Remove the second row from the data (to avoid duplication in the rows)
data <- data[-2, ]
```

Overview of the dataset

Initial exploration of the dataset helps in understanding the data types, structure, and distribution of variables. Summary statistics provide a quick overview of the central tendency and dispersion of the data.

```
print(dim(data))  # Dimensions of the dataset
print(str(data))  # Structure of the dataset
print(summary(data))  # Summary statistics for each column
```

```
> print(dim(data))  # Dimensions of the dataset
[1] 5453 2482
```

Figure 1: Dimensions of Initial data

```
> print(summary(data))  # Summary statistics for each column
      X1          X2          X3          X4          X5          X6
      X7          X8          X9          X10         X11         X12
      X13         X14         X15         X16         X17         X18
      X19         X20         X21         X22         X23         X24
      X25         X26         X27         X28         X29         X30
      X31         X32         X33         X34         X35         X36
      X37         X38         X39         X40         X41         X42
      X43         X44         X45         X46         X47         X48
      X49         X50         X51         X52         X53         X54
      X55         X56         X57         X58         X59         X60
      X61         X62         X63         X64         X65         X66
      X67         X68         X69         X70         X71         X72
      X73         X74         X75         X76         X77         X78
      X79         X80         X81         X82         X83         X84
      X85         X86         X87         X88         X89         X90
      X91         X92         X93         X94         X95         X96
      X97         X98         X99         X100        X101        X102
      X103        X104        X105        X106        X107        X108
      X109        X110        X111        X112        X113        X114
      X115        X116        X117        X118        X119        X120
      X121        X122        X123        X124        X125        X126
      X127        X128        X129        X130        X131        X132
      X133        X134        X135        X136        X137        X138
      X139        X140        X141        X142        X143        X144
      X145        X146        X147        X148        X149        X150
      X151        X152        X153        X154        X155        X156
      X157        X158        X159        X160        X161        X162
      X163        X164        X165        X166        X167        X168
      X169        X170        X171        X172        X173        X174
      X175        X176        X177        X178        X179        X180
```

Figure 2: Initial Summary of Data

Missing data analysis

Analyzing missing data is crucial for determining whether the data is missing at random or systematically. Visualization helps in quickly identifying variables with significant missing data.

```
missing_data_summary <- colSums(is.na(data))
print(missing_data_summary)
```

ta.

```
> # Missing data analysis
> missing_data_summary <- colSums(is.na(data))
> print(missing_data_summary)
  X1  X2  X3  X4  X5  X6  X7  X8  X9
 156 162 150 168 154 153 156 157 173
 X10 X11 X12 X13 X14 X15 X16 X17 X18
 124 148 158 160 148 162 153 148 146
 X19 X20 X21 X22 X23 X24 X25 X26 X27
 176 152 163 152 154 171 165 137 154
 X28 X29 X30 X31 X32 X33 X34 X35 X36
 169 178 166 161 154 150 161 160 148
 X37 X38 X39 X40 X41 X42 X43 X44 X45
 168 173 167 177 161 166 170 151 160
 X46 X47 X48 X49 X50 X51 X52 X53 X54
 140 176 171 154 184 172 153 158 153
 X55 X56 X57 X58 X59 X60 X61 X62 X63
 168 168 145 167 161 171 182 158 147
 X64 X65 X66 X67 X68 X69 X70 X71 X72
 157 143 163 135 170 150 164 174 169
 X73 X74 X75 X76 X77 X78 X79 X80 X81
 161 163 160 180 171 168 159 159 167
 X82 X83 X84 X85 X86 X87 X88 X89 X90
 185 160 159 166 163 142 180 184 147
 X91 X92 X93 X94 X95 X96 X97 X98 X99
```

Figure 3: Missing data summary

```r
missing_data_plot <- data.frame(Variable = names(missing_data_summary),
                                Missing_Count = missing_data_summary) %>%
  ggplot(aes(x = reorder(Variable, -Missing_Count), y = Missing_Count)) +
  geom_bar(stat = "identity", fill = "tomato") +
  coord_flip() +
  labs(title = "Missing Data Summary", x = "Variables", y = "Count of Missing Values") +
  theme_minimal()

print(missing_data_plot)
```

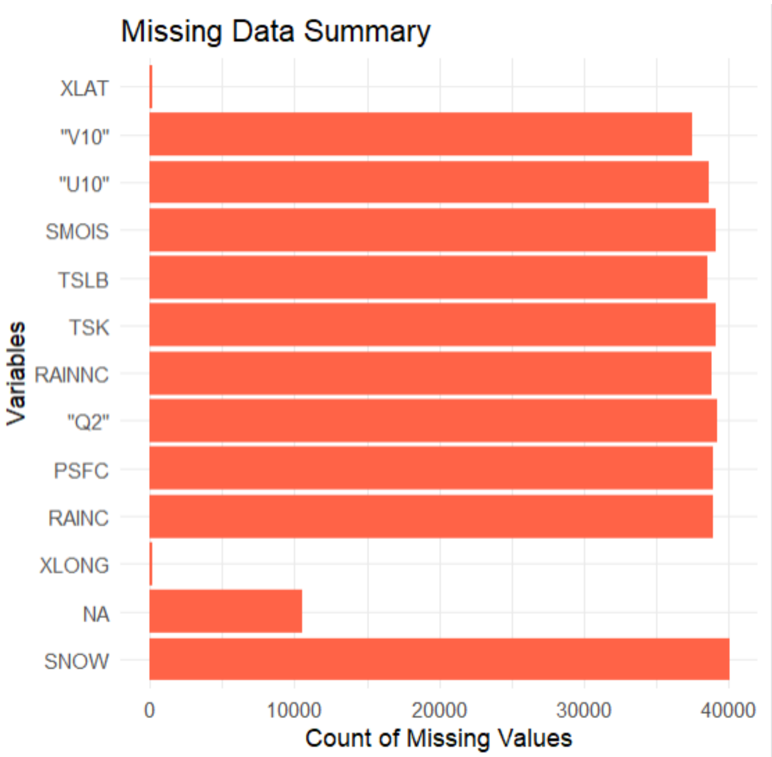

Figure 4: Bar plot to visualize the distribution of missing values

```r
print(head(data))
```

```
> print(head(data))
# A tibble: 6 × 2,482
  XLAT   XLONG   TSK      PSFC  `"U10"` `"V10"` `"Q2"` RAINC
  <chr>  <chr>   <chr>    <chr> <chr>   <chr>   <chr>  <chr>
1 X      X.1     X01.05… X.2   X.3     X.4     X.5    X.6
2 48.871 -11.221 NA       1014… 6.9     5.5     0.006… 0.0
3 49.010 -11.240 285.2    1013… 7.0     5.8     0.006… 0.0
4 49.149 -11.259 285.2    1013… 7.0     6.0     0.006… 0.0
5 49.288 -11.278 285.2    1013… 7.0     6.3     0.006… 0.0
6 49.427 -11.298 285.2    1012… 7.0     6.6     0.006… 0.0
# ℹ 2,474 more variables: RAINNC <chr>, SNOW <chr>,
```

Figure 5: 1st five rows of data

3.2 EDA on preprocessed data

```
print(str(newdata_copy))
print(head(newdata_copy))
print(summary(newdata_copy))
```

```
> print(head(newdata_copy))
# A tibble: 6 × 11
    TSK   PSFC X.U10. X.V10.   X.Q2. RAINC RAINNC  SNOW  TSLB SMOIS
  <dbl>  <dbl>  <dbl>  <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
1  273.  97392    3.1    0.6 0.00393    NA      0     0  278. 0.270
2  272.  96298    2.6    1.5 0.00362     0      0     0  278. 0.246
3  274.  99340   -0.9    3.6 0.00389     0      0     0  278. 0.264
4  282. 100477    1.2    7.7 0.00524     0      0     0  273. 1
5  272.  98697    0.6    0.7 0.00367     0      0     0  280. 0.287
6  274.  99203   -0.3    3.5 0.00422     0      0     0  278. 0.283
# ℹ 1 more variable: DATETIME <dttm>
```

Figure 6: 1st five rows of preprocessed data

```
> print(str(newdata_copy))
tibble [2,996 × 11] (S3: tbl_df/tbl/data.frame)
 $ TSK     : num [1:2996] 273 272 274 282 272 ...
 $ PSFC    : num [1:2996] 97392 96298 99340 100477 98697 ...
 $ X.U10.  : num [1:2996] 3.1 2.6 -0.9 1.2 0.6 -0.3 -0.7 0.9 0 1.3 ...
 $ X.V10.  : num [1:2996] 0.6 1.5 3.6 7.7 0.7 3.5 3.3 3 3.9 2.6 ...
 $ X.Q2.   : num [1:2996] 0.00393 0.00362 0.00389 0.00524 0.00367 0.00422 0.00408 0.00412 0.00465 0.00382 ...
 $ RAINC   : num [1:2996] NA 0 0 0 0 0 0 0 0 0 ...
 $ RAINNC  : num [1:2996] 0 0 0 0 0 0 0 0 0 0 ...
 $ SNOW    : num [1:2996] 0 0 0 0 0 0 0 0 0 0 ...
 $ TSLB    : num [1:2996] 278 278 278 273 280 ...
 $ SMOIS   : num [1:2996] 0.27 0.246 0.264 1 0.287 ...
 $ DATETIME: POSIXct[1:2996], format: "2018-05-01 00:00:00" "2018-05-01 00:00:00" ...
NULL
```

Figure 7: Preprocessed data

```
> print(summary(newdata_copy))
      TSK             PSFC            X.U10.           X.V10.
 Min.   :271.6   Min.   : 92974   Min.   :-8.100   Min.   :-10.600
 1st Qu.:279.7   1st Qu.: 97671   1st Qu.: 0.300   1st Qu.:  2.400
 Median :281.5   Median : 99095   Median : 2.500   Median :  4.200
 Mean   :281.9   Mean   : 98937   Mean   : 2.315   Mean   :  4.141
 3rd Qu.:283.4   3rd Qu.:100362   3rd Qu.: 4.500   3rd Qu.:  6.100
 Max.   :298.0   Max.   :102423   Max.   :11.000   Max.   : 15.300

     X.Q2.              RAINC            RAINNC            SNOW
 Min.   :0.00309   Min.   :0.0000   Min.   : 0.0   Min.   :0.00000
 1st Qu.:0.00508   1st Qu.:0.0000   1st Qu.: 0.0   1st Qu.:0.00000
 Median :0.00610   Median :0.0000   Median : 0.2   Median :0.00000
 Mean   :0.00598   Mean   :0.1084   Mean   : 1.3   Mean   :0.01046
 3rd Qu.:0.00690   3rd Qu.:0.0000   3rd Qu.: 1.4   3rd Qu.:0.00000
 Max.   :0.00929   Max.   :5.3000   Max.   :16.7   Max.   :1.40000
                   NA's   :1
      TSLB            SMOIS             DATETIME
 Min.   :273.2   Min.   :0.2220   Min.   :2018-05-01 00:00:00.00
 1st Qu.:273.2   1st Qu.:0.2609   1st Qu.:2018-05-03 12:00:00.00
 Median :279.6   Median :0.2798   Median :2018-05-06 00:00:00.00
 Mean   :279.2   Mean   :0.4666   Mean   :2018-05-06 00:02:24.19
 3rd Qu.:282.4   3rd Qu.:1.0000   3rd Qu.:2018-05-08 12:00:00.00
 Max.   :290.8   Max.   :1.0000   Max.   :2018-05-11 00:00:00.00

>
```

Figure 8: Summary of data

Descriptive statistics for numerical values

```
print(summary(newdata_copy[, sapply(newdata_copy, is.numeric)]))
```

```
> print(summary(newdata_copy[, sapply(newdata_copy, is.numeric)]))
      TSK             PSFC             X.U10.           X.V10.       
 Min.   :271.6   Min.   : 92974   Min.   :-8.100   Min.   :-10.600  
 1st Qu.:279.7   1st Qu.: 97671   1st Qu.: 0.300   1st Qu.:  2.400  
 Median :281.5   Median : 99095   Median : 2.500   Median :  4.200  
 Mean   :281.9   Mean   : 98937   Mean   : 2.315   Mean   :  4.141  
 3rd Qu.:283.4   3rd Qu.:100362   3rd Qu.: 4.500   3rd Qu.:  6.100  
 Max.   :298.0   Max.   :102423   Max.   :11.000   Max.   : 15.300  
                                                                    
     X.Q2.             RAINC            RAINNC           SNOW        
 Min.   :0.00309   Min.   :0.0000   Min.   : 0.0   Min.   :0.00000  
 1st Qu.:0.00508   1st Qu.:0.0000   1st Qu.: 0.0   1st Qu.:0.00000  
 Median :0.00610   Median :0.0000   Median : 0.2   Median :0.00000  
 Mean   :0.00598   Mean   :0.1084   Mean   : 1.3   Mean   :0.01046  
 3rd Qu.:0.00690   3rd Qu.:0.0000   3rd Qu.: 1.4   3rd Qu.:0.00000  
 Max.   :0.00929   Max.   :5.3000   Max.   :16.7   Max.   :1.40000  
                   NA's   :1                                        
      TSLB            SMOIS       
 Min.   :273.2   Min.   :0.2220  
 1st Qu.:273.2   1st Qu.:0.2609  
 Median :279.6   Median :0.2798  
 Mean   :279.2   Mean   :0.4666  
 3rd Qu.:282.4   3rd Qu.:1.0000  
 Max.   :290.8   Max.   :1.0000  
```

Figure 9: Descriptive summary of numerical values

Checking missing values

```
print(colMeans(is.na(newdata_copy)))
```

```
> print(colMeans(is.na(newdata_copy)))
         TSK         PSFC       X.U10.       X.V10.        X.Q2. 
0.0000000000 0.0000000000 0.0000000000 0.0000000000 0.0000000000 
       RAINC       RAINNC         SNOW         TSLB        SMOIS 
0.0003337784 0.0000000000 0.0000000000 0.0000000000 0.0000000000 
    DATETIME 
0.0000000000 
```

Figure 10: Missing values count

Explore data distribution using Histogram

```r
numeric_vars <- names(newdata_copy)[sapply(newdata_copy, is.numeric)]
for (var in numeric_vars) {
  print(
    ggplot(newdata_copy, aes_string(x = var)) +
      geom_histogram(bins = 30, color = "black", fill = "lightblue") +
      labs(title = paste("Distribution of", var), x = var, y = "Frequency") +
      theme_bw()
  )
}
```

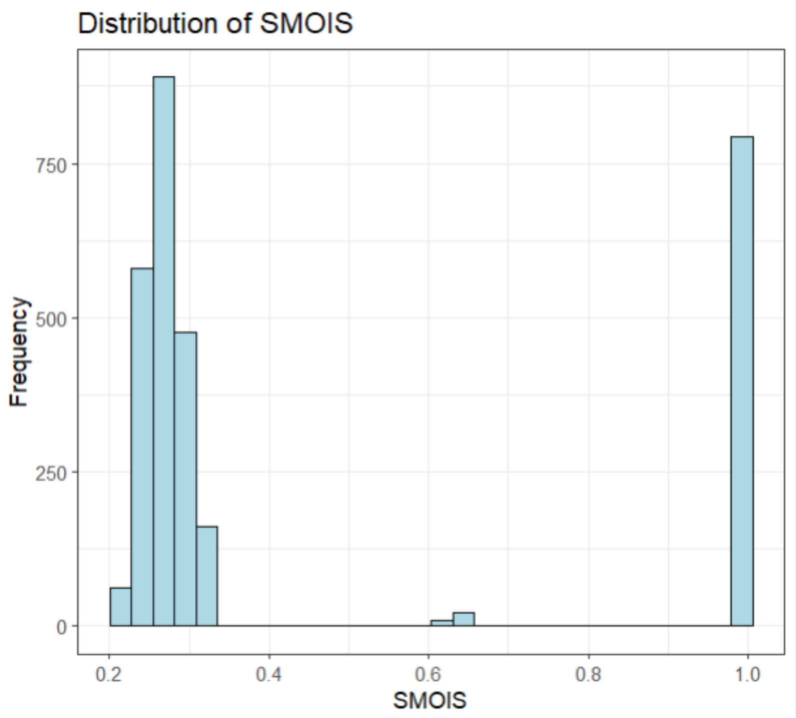

Figure 11: Data Distribution Histogram

Identify and report potential outliers:

```r
# Identify outliers with z-scores (more than 3 standard deviations from the mean)
outliers <- lapply(newdata_copy[numeric_vars], function(x) {
  abs_z_scores <- abs((x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE))
  return(which(abs_z_scores > 3))
})

# Report columns containing outliers based on z-scores
outlier_columns <- names(outliers)[sapply(outliers, length) > 0]
if (length(unlist(outliers)) > 0) {
  cat("Potential outliers identified in columns:", outlier_columns, "\n")
} else {
  cat("No potential outliers identified based on z-scores.\n")
}
```

```
Potential outliers identified in columns: TSK PSFC X.U10. X.V1
0. RAINC RAINNC SNOW
```

Figure 12: Columns containing outliers

Daytime vs Nighttime Pressure(PSFC) Analysis

A t-test helps determine whether there is a statistically significant difference in surface pressure between daytime and nighttime, providing insights into diurnal patterns.

```
# Daytime vs. Nighttime Pressure (PSFC) Analysis
data_df <- data1_clean
data_df$Hour <- as.numeric(format(data_df$DATETIME, "%H"))
data_df$Day_Night <- ifelse(data_df$Hour >= 6 & data_df$Hour < 18, "Daytime", "Nighttime")

# Subset data and perform t-test
daytime_data <- filter(data_df, Day_Night == "Daytime")
nighttime_data <- filter(data_df, Day_Night == "Nighttime")
t_test_result <- t.test(daytime_data$PSFC, nighttime_data$PSFC)
print(t_test_result)
```

```
> print(t_test_result)

        Welch Two Sample t-test

data:  daytime_data$PSFC and nighttime_data$PSFC
t = -1.0254, df = 2993.9, p-value = 0.3053
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -200.19014   62.70907
sample estimates:
mean of x mean of y
 98902.91  98971.65
```

Figure 13: t-test result

Correlation analysis between SMOIS and TSK

Correlation between soil moisture and skin temperature can reveal important insights into the land-atmosphere interaction and surface energy balance.

```
# Correlation Analysis between SMOIS and TSK
correlation <- cor(data_df$SMOIS, data_df$TSK, use = "complete.obs")
print(correlation)
corrplot(cor(data_df[, c("SMOIS", "TSK")], use = "complete.obs"), method = "circle")
```

```
> print(correlation)
[1] -0.05095017
> corrplot(cor(data_df[, c("SMOIS", "TSK")], use = "complete.o
bs"), method = "circle")
> |
```

Figure 14: Correlation between SMOIS and TSK

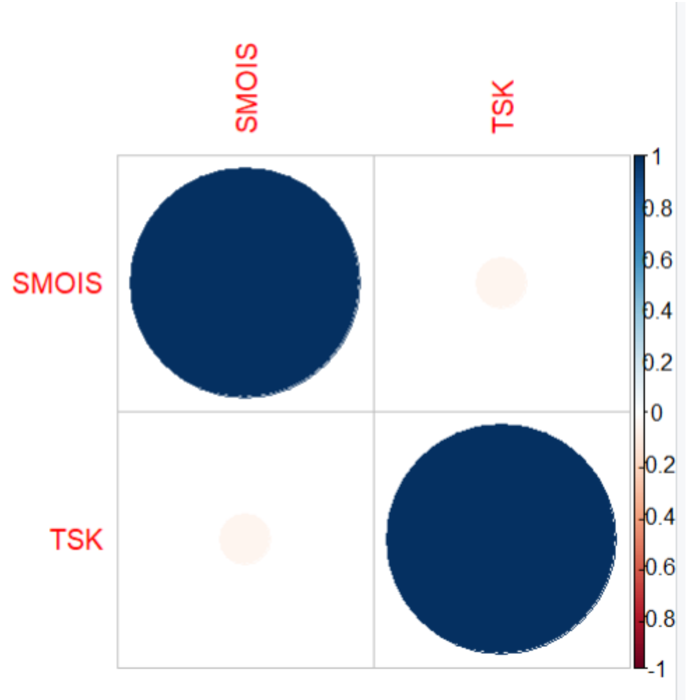

Figure 15: Correlation Plot between SMOIS and TSK

Wind speed analysis

Analyzing wind speed helps in understanding the wind patterns, which are critical for weather prediction, especially in the context of renewable energy sources like wind power.

```
# Mean Wind Speed Analysis
data_df$Date <- as.Date(data_df$DATETIME)
mean_wind_speed <- aggregate(WIND_SPEED ~ Date, data = data_df, FUN = mean)
print(mean_wind_speed)
```

```
> print(mean_wind_speed)
         Date WIND_SPEED
1  2018-04-30   3.705616
2  2018-05-01   6.882601
3  2018-05-02   6.569122
4  2018-05-03   6.583378
5  2018-05-04   6.888547
6  2018-05-05   5.845709
7  2018-05-06   4.634054
8  2018-05-07   4.577939
9  2018-05-08   5.757703
10 2018-05-09   6.750845
11 2018-05-10   5.904093
```

Figure 16: Mean of Wind Speed

Distribution of Wind Speed

```r
# Plot the distribution of wind speed
ggplot(data_df, aes(x = WIND_SPEED)) +
  geom_histogram(binwidth = 1, fill = "skyblue", color = "black") +
  labs(title = "Distribution of Wind Speed", x = "WIND_SPEED (km/h)", y = "Frequency") +
  theme_minimal()
```

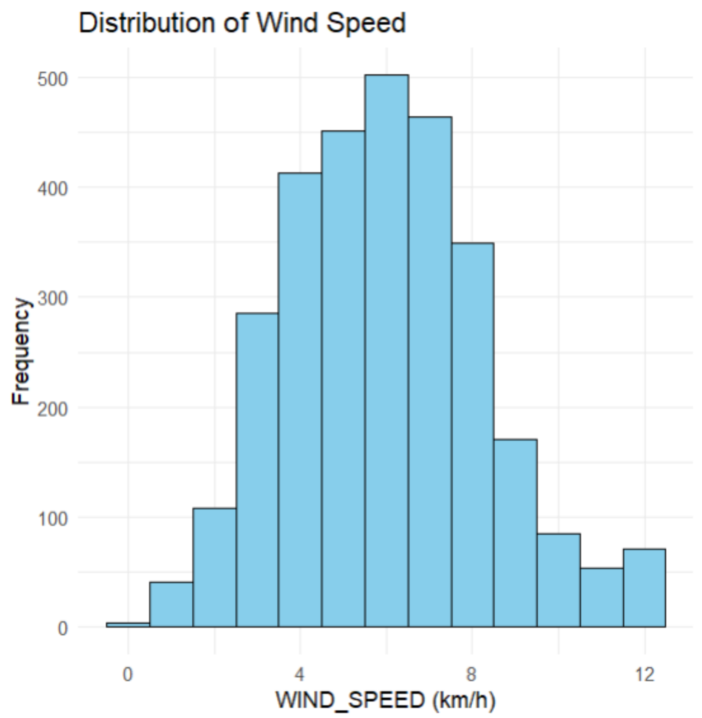

Figure 17: Distribution of Wind Speed

4. Data Pre-processing

Data preprocessing is a crucial step to prepare the raw dataset for machine learning models. It involved several tasks, including:

Handling Missing Values: Rows with missing data in critical columns (e.g., TSK, PSFC, SMOIS, WIND_SPEED, and RAINC) were removed to ensure the integrity of the analysis. This step was vital to prevent biases and inaccuracies in model predictions.

Feature Engineering: Additional features were created, such as converting the DATETIME column to a proper date-time format and categorizing times of the day into Day and Night to facilitate diurnal analysis. We have just extracted the data of 11 days and specific to that of Scotland

Data Splitting: The cleaned dataset was split into training (70%) and testing (30%) sets to evaluate model performance accurately. This step ensures that the models are trained on a representative sample and validated on unseen data to prevent overfitting.

Standardization: Numeric features were standardized to ensure that all variables contribute equally to the model's learning process, improving the overall performance and convergence of certain algorithms.

These preprocessing steps ensured that the data was in the best possible shape for developing robust and accurate predictive models, forming the foundation for the subsequent machine learning tasks.

Filtering out Bolton's coordinates

Geographical filtering ensures that the data is relevant to the area of interest, and sampling helps in maintaining a manageable dataset size for analysis.

```r
# Define Bolton's coordinates (approximately)
bolton_lat <- 53.5789
bolton_long <- -2.4292


# Set the conditions for filtering
min_latitude <- 49      # Example lower bound for latitude
max_latitude <- 54      # Example upper bound for latitude
min_longitude <- -11    # Example lower bound for longitude
max_longitude <- -7     # Example upper bound for longitude

colnames(data) <- make.names(names(data), unique = TRUE)

# Filter out rows that match Bolton's coordinates and fall within your specified latitude and longitude range
filtered_df <- data %>%
  filter(!(XLAT == bolton_lat & XLONG == bolton_long)) %>%  # Exclude Bolton's coordinates
  filter(XLAT >= min_latitude & XLONG <= max_latitude) %>%   # Select appropriate latitude range
  filter(XLAT >= min_longitude & XLONG <= max_longitude)   # Select appropriate longitude range




# Ensure that at least 300 rows are selected
filtered_df <- filtered_df %>%
  sample_n(min(n(), 300))  # Randomly sample at least 300 rows or fewer if less than 300 rows are available

# Display the first few rows of the filtered dataset
print(head(filtered_df))
```

```
> # Display the first few rows of the filtered dataset
> print(head(filtered_df))
# A tibble: 6 × 2,482
  XLAT   XLONG  TSK   PSFC  X.U10. X.V10. X.Q2. RAINC RAINNC SNOW  TSLB  SMOIS TSK.1 PSFC.1 X.U10..1 X.V10..1 X.Q2..1 RAINC.1
  <chr>  <chr>  <chr> <chr> <chr>  <chr>  <chr> <chr> <chr>  <chr> <chr> <chr> <chr> <chr>  <chr>    <chr>    <chr>   <chr>
1 50.200 -3.565 283.5 1011… 6.1    -4.3   0.00… 0.0   0.0    NA    273.2 1.00… 283.5 101098 6.1      -0.1     0.00536 0.0
2 50.272 -11.2… 284.9 1011… 6.9    8.0    0.00… 0.0   0.0    0.0   273.2 1.00… NA    100744 7.0      10.3     0.00653 0.0
3 49.352 -5.712 284.4 1014… 6.0    0.3    0.00… 0.0   0.0    0.0   273.2 1.00… 284.4 101325 6.3      3.1      0.00557 0.0
4 54.776 -4.239 282.2 1004… 1.6    NA     0.00… 0.0   0.0    0.0   273.2 1.00… 282.2 NA     2.6      3.9      0.00565 0.0
5 55.051 -4.481 271.3 98026 2.0    0.1    0.00… 0.0   0.0    0.0   278.3 0.30… 272.1 97852  0.0      2.6      0.00405 0.0
6 59.557 -4.534 282.1 1004… -0.7   6.4    0.00… 0.0   NA     0.0   NA    1.00… 282.1 100219 -0.1     NA       0.00498 NA
# ℹ 2,464 more variables: RAINNC.1 <chr>, SNOW.1 <chr>, TSLB.1 <chr>, SMOIS.1 <chr>, TSK.2 <chr>, PSFC.2 <chr>,
```

Figure 18: 1st five rows of data

Setting up the correct columns in the dataset and adding the date_time column

calculating appropriate datetime values and combining the results into a comprehensive data frame. This structured approach ensures that the data is correctly aligned with time and ready for further analysis, such as forecasting or modeling.

```r
df <- read_csv("C:/Users/Ahmad Afzal/Desktop/cleaned_file.csv")


# Assuming your dataset is named 'df'
# Define the start date and time
start_datetime <- as.POSIXct("2018-05-01 00:00:00", format="%Y-%m-%d %H:%M:%S")

# Initialize an empty dataframe to store the results
result <- data.frame()

# Define the base columns to keep
base_columns <- c("XLAT", "XLONG", "TSK", "PSFC", "X.U10.", "X.V10.",
                  "X.Q2.", "RAINC", "RAINNC", "SNOW", "TSLB", "SMOIS")

# Loop over each set of columns
#This data is for the 10 days of data
for (i in 0:80) {
  # Create a copy of the base columns
  temp_df <- df[, base_columns]

  if (i > 0) {
    # Modify the column names to include the suffix
    suffix <- paste0(".", i)
    columns_with_suffix <- paste0(base_columns[-c(1:2)], suffix)
    temp_df[, -c(1:2)] <- df[, columns_with_suffix]
  }

  # Calculate the current datetime for this set
  current_datetime <- start_datetime + (i * 3 * 3600)

  # Format the datetime as required
  temp_df$date_time <- format(current_datetime, "%d.%m.%Y.%H.%M")

  # Bind the current dataframe to the result
  result <- bind_rows(result, temp_df)
}

# Display the result
head(result)
```

```
> # Display the result
> head(result)
    XLAT   XLONG   TSK   PSFC X.U10. X.V10.   X.Q2. RAINC
1 56.390  -0.779    NA 100571    3.5   -6.9      NA     0
2 56.559  -3.005 272.0  99379    2.3    0.4 0.00376     0
3 56.831  -2.749 272.6  97392    3.1    0.6 0.00393    NA
4 57.382  -4.760 272.3  96298    2.6    1.5 0.00362     0
5 56.877 -12.279 283.2  99604    4.6   14.3 0.00670     0
6 58.470  -5.040 274.4  99340   -0.9    3.6 0.00389     0
  RAINNC SNOW  TSLB  SMOIS        date_time
1    0.0    0 273.2 1.0000 01.05.2018.00.00
2    0.0    0 278.6 0.2779 01.05.2018.00.00
3    0.0    0 277.5 0.2696 01.05.2018.00.00
4    0.0    0 278.1 0.2457 01.05.2018.00.00
5    0.3    0 273.2 1.0000 01.05.2018.00.00
6    0.0    0 277.7 0.2640 01.05.2018.00.00
>
```

Figure 19: 1st five rows of data

Extracting the data having the coordinates of Scotland

We are just keeping the associated with the coordinates of Scotland to perform further tasks and operations

```
# Just keeping the coordinates of Scotland
scotland_lat_min <- 55.8
scotland_lat_max <- 58.6
scotland_long_min <- -6.5
scotland_long_max <- -2.0

# Filter for Coordinates within Scotland
scotland_data <- df[df$XLAT >= scotland_lat_min &
                     df$XLAT <= scotland_lat_max &
                     df$XLONG >= scotland_long_min &
                     df$XLONG <= scotland_long_max, ]

# Print the filtered coordinates (Optional)
print(scotland_data)
```

```
# A tibble: 2,997 × 13
    XLAT XLONG   TSK   PSFC X.U10. X.V10.   X.Q2. RAINC RAINNC  SNOW  TSLB
   <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
 1  56.6 -3.00  272   99379    2.3    0.4 0.00376     0      0     0  279.
 2  56.8 -2.75  273.  97392    3.1    0.6 0.00393    NA      0     0  278.
 3  57.4 -4.76  272.  96298    2.6    1.5 0.00362     0      0     0  278.
 4  58.5 -5.04  274.  99340   -0.9    3.6 0.00389     0      0     0  278.
 5  58.5 -5.56  282. 100477    1.2    7.7 0.00524     0      0     0  273.
 6  56.0 -5.23  272.  98697    0.6    0.7 0.00367     0      0     0  280.
 7  57.2 -5.77  274.  99203   -0.3    3.5 0.00422     0      0     0  278.
 8  58.2 -5.29  274.  99758   -0.7    3.3 0.00408     0      0     0  278.
 9  56.4 -5.98  274.  98914    0.9    3   0.00412     0      0     0  278.
10  58.5 -2.96  281. 100515    0      3.9 0.00465     0     NA     0  273.
# ℹ 2,987 more rows
```

Figure 20: data representation

Removing XLAT and XLONG columns from the dataset

As now these both columns are of no use so we will remove these columns from our dataset in order to make our data cleaner and more manageable to perform further tasks

```
data_without_coordinates <- data %>%
  select(-XLAT, -XLONG)

# Save the updated data to a new CSV file
write_csv(data_without_coordinates, "C:/Users/Ahmad Afzal/Desktop/Scotland_WRFdata_Without_Coordinates.csv")

# Print the first few rows of the updated data to the console
print(head(data_without_coordinates))
```

```
> # Print the first few rows of the updated data to the console
> print(head(data_without_coordinates))
# A tibble: 6 × 11
    TSK   PSFC X.U10. X.V10.    X.Q2. RAINC RAINNC  SNOW  TSLB SMOIS
  <dbl>  <dbl>  <dbl>  <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl>
1  272   99379    2.3    0.4 0.00376     0      0     0  279. 0.278
2  273.  97392    3.1    0.6 0.00393    NA      0     0  278. 0.270
3  272.  96298    2.6    1.5 0.00362     0      0     0  278. 0.246
4  274.  99340   -0.9    3.6 0.00389     0      0     0  278. 0.264
5  282. 100477    1.2    7.7 0.00524     0      0     0  273. 1
6  272.  98697    0.6    0.7 0.00367     0      0     0  280. 0.287
# ℹ 1 more variable: date_time <chr>
> 
```

Figure 21: 1st five rows without coordination

Renaming date_time column as DATETIME

```
# Check if 'date_time' column exists and rename it
if ("date_time" %in% colnames(new_data)) {
  colnames(new_data)[colnames(new_data) == "date_time"] <- "DATETIME"
} else {
  stop("The 'date_time' column is missing in the dataset.")
}


# Print the first few rows of the updated new_data to check changes
print(head(new_data))
```

```
> # Print the first few rows of the updated new_data to check changes
> print(head(new_data))
# A tibble: 6 × 11
    TSK   PSFC X.U10. X.V10.   X.Q2. RAINC RAINNC  SNOW  TSLB SMOIS DATETIME
  <dbl>  <dbl>  <dbl>  <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <chr>
1  272   99379    2.3    0.4 0.00376     0      0     0  279. 0.278 01.05.2018.00.00
2  273.  97392    3.1    0.6 0.00393    NA      0     0  278. 0.270 01.05.2018.00.00
3  272.  96298    2.6    1.5 0.00362     0      0     0  278. 0.246 01.05.2018.00.00
4  274.  99340   -0.9    3.6 0.00389     0      0     0  278. 0.264 01.05.2018.00.00
5  282. 100477    1.2    7.7 0.00524     0      0     0  273. 1     01.05.2018.00.00
6  272.  98697    0.6    0.7 0.00367     0      0     0  280. 0.287 01.05.2018.00.00
>
```

Figure 22: 1st five rows of data

Converting columns to numeric format excluding DATETIME

```
newdata_copy <- new_data[-1, ]
colnames(newdata_copy) <- c("TSK", "PSFC", "U10", "V10", "Q2", "RAINC", "RAINNC", "SNOW", "TSLB", "SMOIS", "DATETIME")

# Convert columns to numeric format, excluding 'DATETIME'
numeric_cols <- setdiff(colnames(newdata_copy), "DATETIME")
newdata_copy[numeric_cols] <- lapply(newdata_copy[numeric_cols], as.numeric)

# Replace NA values with the mean of the previous two values
fill_na_with_mean <- function(x) {
  na_index <- which(is.na(x))
  for (i in na_index) {
    if (i > 2) {
      x[i] <- mean(c(x[i - 1], x[i - 2]), na.rm = TRUE)
    }
  }
  return(x)
}
print(colnames(newdata_copy))
```

```
> print(colnames(newdata_copy))
 [1] "TSK"      "PSFC"     "X.U10."   "X.V10."   "X.Q2."    "RAINC"
 [7] "RAINNC"   "SNOW"     "TSLB"     "SMOIS"    "DATETIME"
>
```

Figure 23: Columns representation

Converting DATETIME to POSIXct format

```
newdata_copy[numeric_cols] <- lapply(newdata_copy[numeric_cols], fill_na_with_mean)

# Convert 'DATETIME' to POSIXct format
newdata_copy$DATETIME <- as.POSIXct(newdata_copy$DATETIME, format = "%d.%m.%Y.%H.%M")
```

Calculating Wind Speed

```
newdata_copy$WIND_SPEED <- round(sqrt(newdata_copy$U10^2 + newdata_copy$V10^2), 2)
```

Cleaning the identified outliers

Capping outliers prevents extreme values from disproportionately influencing the analysis and modeling, particularly for variables not related to precipitation.

```
data1_clean <- newdata_copy
for (var in outlier_columns) {
  if (!(var %in% c("RAINC", "RAINNC"))) {
    lower_bound <- quantile(data1_clean[[var]], probs = 0.25, na.rm = TRUE) - 1.5 * IQR(data1_clean[[var]], na.rm = TRUE)
    upper_bound <- quantile(data1_clean[[var]], probs = 0.75, na.rm = TRUE) + 1.5 * IQR(data1_clean[[var]], na.rm = TRUE)
    data1_clean[[var]] <- pmin(pmax(data1_clean[[var]], lower_bound), upper_bound)
  }
}
```

Removing NA values form dataset

```
# 1. Define the critical columns, including RAINC
critical_columns <- c("TSK", "PSFC", "SMOIS", "WIND_SPEED", "RAINC")

# 2. Remove rows with any NA values in the critical columns
data <- data %>%
  filter(!if_any(all_of(critical_columns), is.na))

# 3. Verify that there are no NA values left
na_summary <- colSums(is.na(data))
print("Remaining NA values per column:")
print(na_summary)

# Proceed to the next steps only if NA values are handled
if (all(na_summary == 0)) {
  cat("All NA values have been handled.\n")
} else {
  stop("NA values still exist in the dataset. Please check the handling process.")
}

# View the cleaned data
print(head(data))
```

```
All NA values have been handled.
> # View the cleaned data
> print(head(data))
# A tibble: 6 × 15
    TSK   PSFC   U10   V10      Q2 RAINC RAINNC  SNOW  TSLB SMOIS DATETIME            WIND_SPEED  Hour Day_Night Date      
  <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dttm>                   <dbl> <dbl> <fct>     <date>    
1  274.  96298   2.6   1.5 0.00362     0      0     0  278. 0.246 2018-04-30 19:00:00       3        0 Nighttime 2018-04-30
2  274.  99340  -0.9   3.6 0.00389     0      0     0  278. 0.264 2018-04-30 19:00:00       3.71     0 Nighttime 2018-04-30
3  282. 100477   1.2   7.7 0.00524     0      0     0  273. 1     2018-04-30 19:00:00       7.79     0 Nighttime 2018-04-30
4  274.  98697   0.6   0.7 0.00367     0      0     0  280. 0.287 2018-04-30 19:00:00       0.92     0 Nighttime 2018-04-30
5  274.  99203  -0.3   3.5 0.00422     0      0     0  278. 0.283 2018-04-30 19:00:00       3.51     0 Nighttime 2018-04-30
6  274.  99758  -0.7   3.3 0.00408     0      0     0  278. 0.266 2018-04-30 19:00:00       3.37     0 Nighttime 2018-04-30
> 
```

Figure 24: NA values handled

5. Detailed Data Visualization

These visualizations together give a full picture of the dataset. They show patterns, connections, and odd points in the data, which you need to analyze and model data well. When you see the data laid out, you can make smart choices about how to prepare it, which features to pick, and how to build your model. This leads to more precise forecasts and better choices in weather-related uses.

Temperature at Skin Level (TSK) Distribution

The TSK histogram helps us grasp how skin temperature values are spread out in the dataset. By looking at the distribution's shape (like normal or skewed), we can get a feel for the typical temperature range and spot any odd values such as extreme temperatures, that might need a closer look.

```
p1 <- ggplot(data, aes(x = TSK)) +
  geom_histogram(binwidth = 0.5, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Temperature at Skin Level (TSK)", x = "TSK", y = "Frequency") +
  theme_minimal()

# Plot 1
print(p1)
```

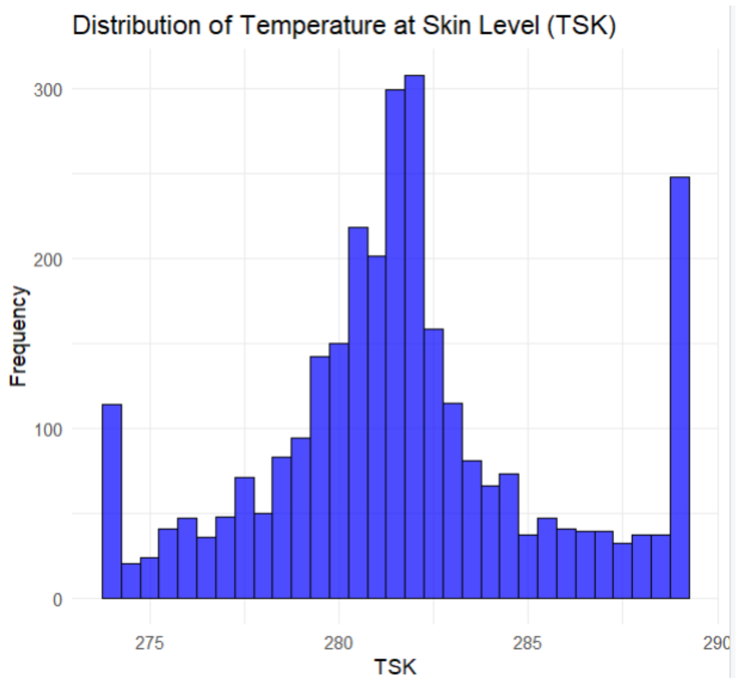

Figure 25: Histogram representing distribution of TSK

Surface Pressure (PSFC) Distribution

The distribution of surface pressure (PSFC) has a key role in grasping the atmospheric conditions shown in the data. Surface pressure affects weather forecasts, and seeing its distribution helps spot common pressure levels and any odd values that might point to weather events or measurement mistakes.

```
p2 <- ggplot(data, aes(x = PSFC)) +
  geom_histogram(binwidth = 1, fill = "red", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Surface Pressure (PSFC)", x = "PSFC", y = "Frequency") +
  theme_minimal()

# Plot 2
print(p2)
```

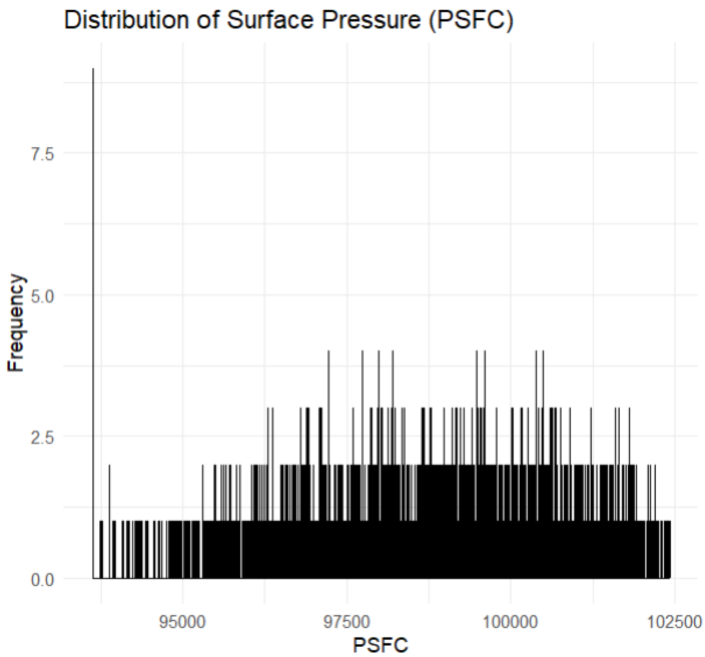

Figure 26: Histogram representing distribution of PSFC

Wind Speed Distribution

Wind speed is a vital factor in weather analysis. The histogram of wind speed gives a broad view of how often different wind speeds occur in the dataset. This visual aid helps identify the most typical wind speeds and any extreme wind conditions, which could matter a lot to understand weather patterns and how they affect the environment.

```
p3 <- ggplot(data, aes(x = WIND_SPEED)) +
  geom_histogram(binwidth = 0.5, fill = "green", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Wind Speed", x = "Wind Speed", y = "Frequency") +
  theme_minimal()

# Plot 3
print(p3)
```

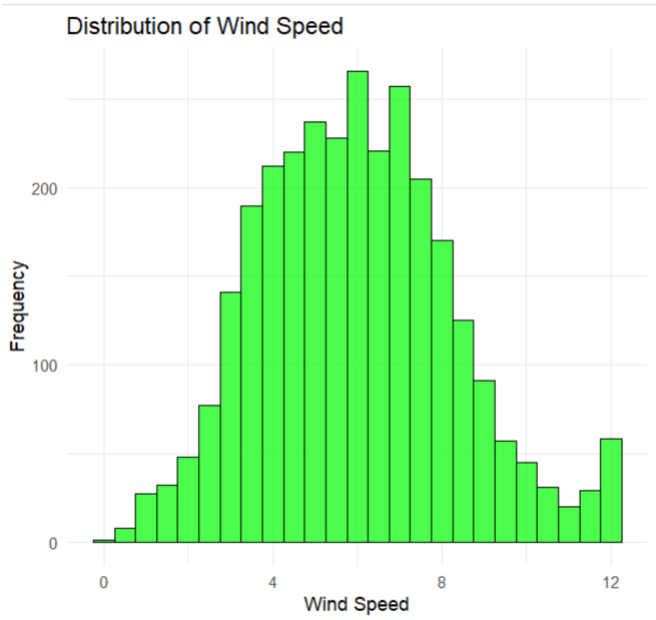

Figure 27: Histogram representing distribution of Wind Speed

Relationship between TSK and PSFC

The graph showing TSK versus PSFC, along with a straight line fit, reveals how surface temperature and surface pressure are connected. This connection plays a key role in weather science, as temperature and pressure often work together in ways we can predict. Looking at this link on a graph helps us see how strong it is and what it's like, which can guide us in making weather models.

```
p4 <- ggplot(data, aes(x = PSFC, y = TSK)) +
  geom_point(color = "purple", alpha = 0.6) +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Relationship between Surface Pressure (PSFC) and TSK", x = "PSFC", y = "TSK") +
  theme_minimal()

# Plot 4
print(p4)
```

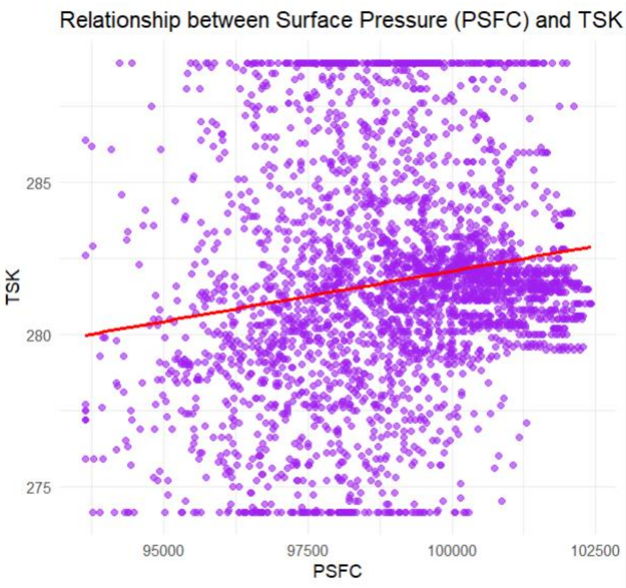

Figure 28: Scatterplot representing relationship between TSK and PSFC

Wind Components (U10 and V10) Distribution

A graph showing U10 (zonal wind component) against V10 (meridional wind component) gives insight into wind direction and strength in the dataset. How these parts spread out can show main wind directions and changes in wind patterns. This matters to understand weather systems and how they move.

```
# 5. Wind Components (U10 and V10) Distribution
p5 <- ggplot(data, aes(x = U10, y = V10)) +
  geom_point(color = "orange", alpha = 0.6) +
  labs(title = "Distribution of Wind Components (U10 vs V10)", x = "U10", y = "V10") +
  theme_minimal()

# Plot 5
print(p5)
```

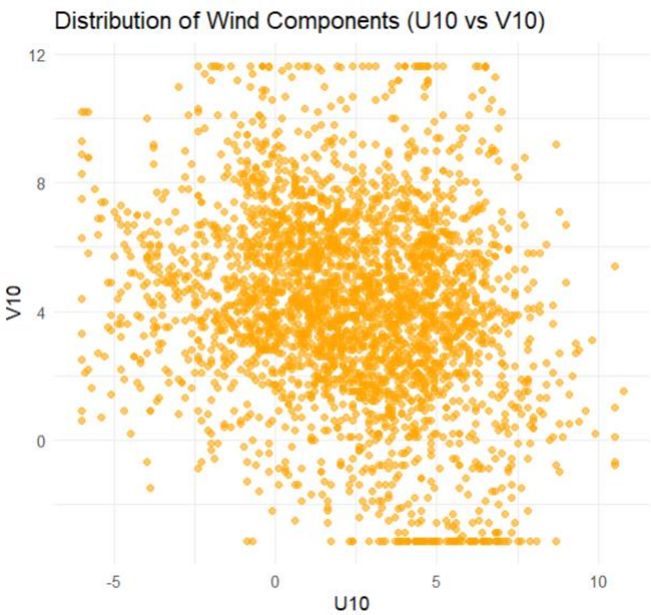

Figure 29: Scatterplot representing distribution of Wind Components

Correlation Heatmap

The correlation heatmap shows how strong and in what way many variables in the dataset relate to each other. This picture helps spot which variables have strong links, which matters when picking features to model. It also points out any issues with variables being too related, which could affect how well a model works.

```
numeric_data <- data[, c("TSK", "PSFC", "U10", "V10", "Q2", "RAINC", "RAINNC", "SNOW", "TSLB", "SMOIS", "WIND_SPEED")]
cor_matrix <- cor(numeric_data, use = "complete.obs")
melted_cor_matrix <- melt(cor_matrix)

p6 <- ggplot(data = melted_cor_matrix, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0, limit = c(-1, 1), space = "Lab", name = "Correlation") +
  theme_minimal() +
  labs(title = "Correlation Heatmap", x = "", y = "") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot 6
print(p6)
```

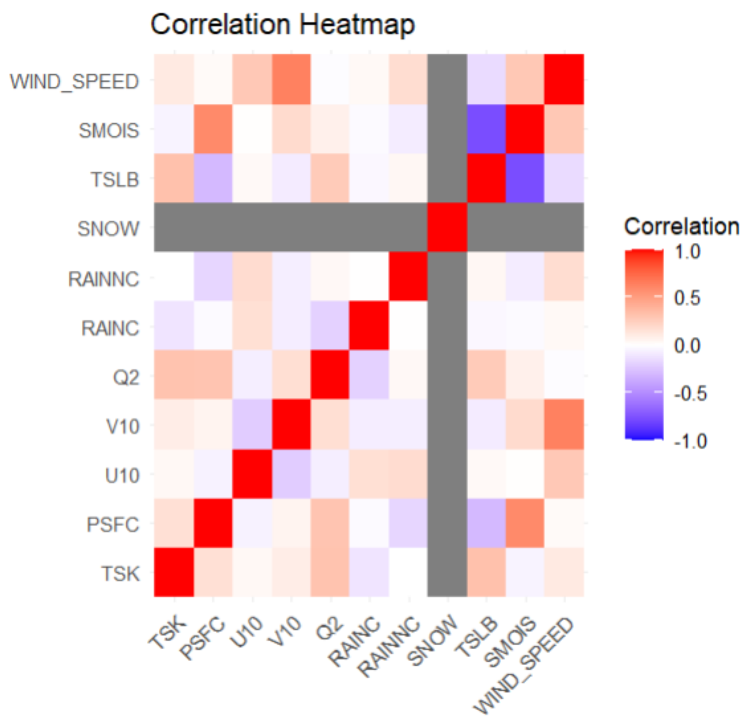

Figure 30: Correlation heatmap representing numeric data

Boxplots of TSK by Day/Night

The boxplot shows how TSK (temperature at skin level) changes from day to night. This graph helps us spot any big differences in temperature between daylight and darkness, which might point to daily temperature swings—something we often see in weather data. Getting a grip on these changes is key to predicting weather and to fine-tune our models.

```
p7 <- ggplot(data, aes(x = Day_Night, y = TSK, fill = Day_Night)) +
  geom_boxplot() +
  labs(title = "Boxplot of TSK by Day/Night", x = "Day/Night", y = "TSK") +
  theme_minimal()

# Plot 7
print(p7)
```

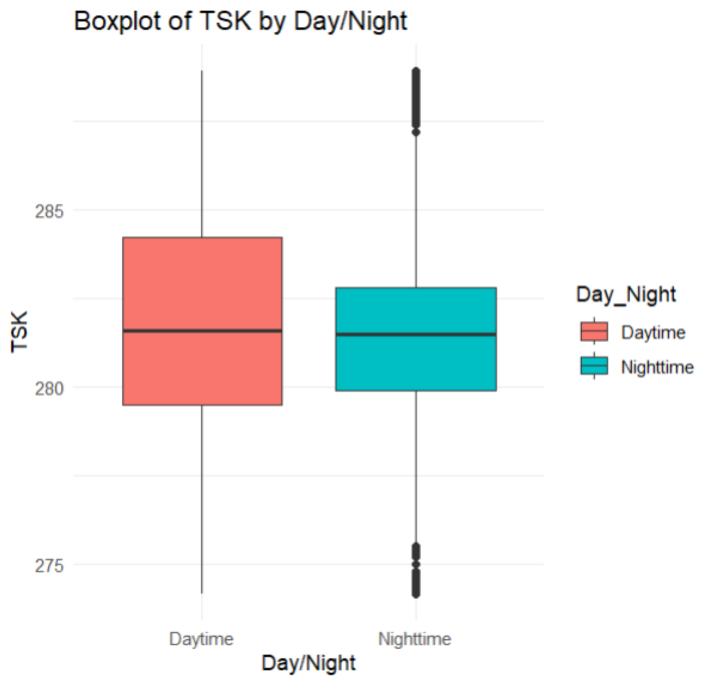

Figure 31: Boxplot representing TSK by day/night

Rainfall (RAINC) over Time

The line plot of RAINC (cumulative rainfall) over time gives us a clear picture of rainfall patterns throughout the dataset's period. This graph helps us spot trends, like times of heavy rain or dry spells. These insights are key to grasp weather events and how they might affect the environment and people's lives.

```
p8 <- ggplot(data, aes(x = DATETIME, y = RAINC)) +
  geom_line(color = "darkblue", alpha = 0.7) +
  labs(title = "Rainfall (RAINC) Over Time", x = "DateTime", y = "RAINC") +
  theme_minimal()

# Plot 8
print(p8)
```

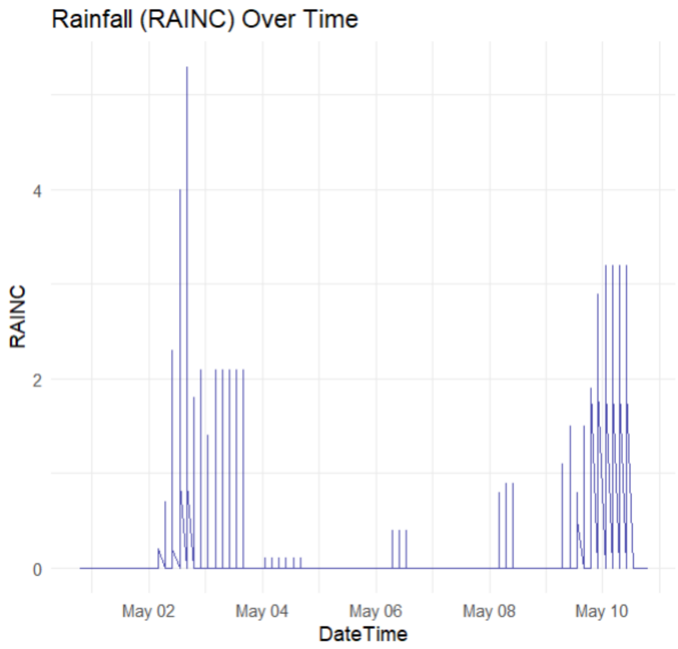

Figure 32: Line plot representing rainfall over time

Scatter Plot Matrix

The scatter plot matrix shows how several main variables in the dataset compare to each other. This view helps us find links between variables and patterns we might miss when looking at each variable alone. It also helps to find outliers and understand the overall data structure, which is crucial for good modeling.

```
pairs(data[, c("TSK", "PSFC", "U10", "V10", "Q2", "WIND_SPEED")], main = "Scatterplot Matrix")
```

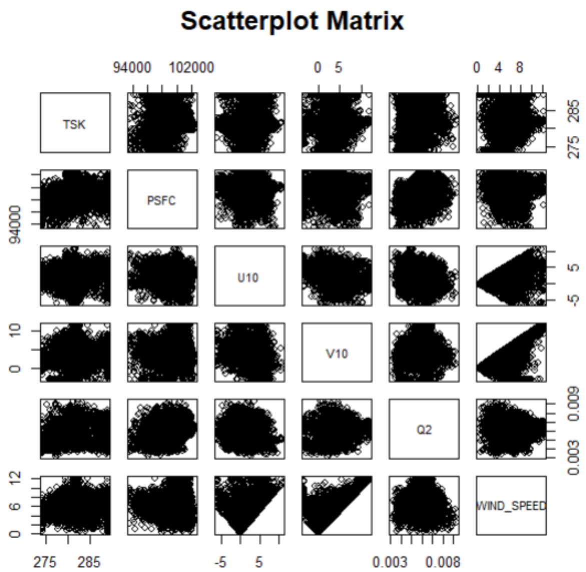

Figure 33: Scatter plot representing main variables

Monthly Average TSK

The line plot of monthly average TSK reveals changes in the average temperature at skin level over time. This time-based analysis helps to spot seasonal patterns and long-term temperature shifts. Understanding these trends is key to grasp the climate features of the area and to make smart choices in industries that depend on weather conditions.

```r
data$Month <- format(data$Date, "%Y-%m")
monthly_avg_tsk <- aggregate(TSK ~ Month, data, mean)

p9 <- ggplot(monthly_avg_tsk, aes(x = Month, y = TSK)) +
  geom_line(group = 1, color = "darkred") +
  labs(title = "Monthly Average TSK", x = "Month", y = "Average TSK") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Plot 9
print(p9)
```

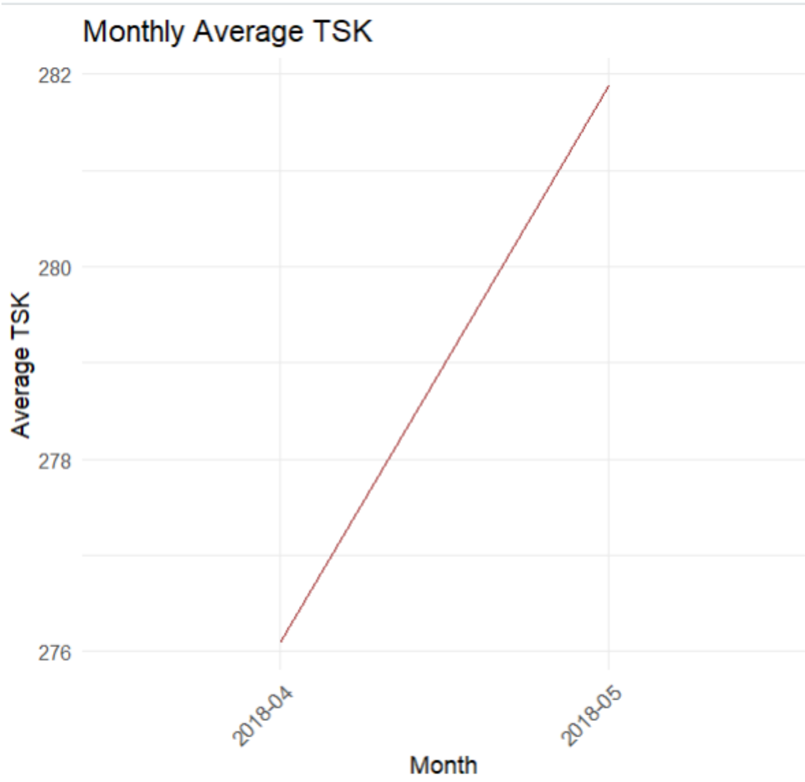

Figure 34: Line plot representing monthly average TSK

6. Data Preparation

Importing Necessary libraries

```
#Load necessary libraries
library(tidyverse)
library(caret)
library(randomForest)
library(e1071)
library(gbm)
library(corrplot)
library(ggplot2)
```

Loading the preprocessed dataset

```
data <- read_csv("C:/Users/Ahmad Afzal/Desktop/Refined_Scotland_WRFdata.csv")
```

```
data.frame(data)
# Ensure the column names are correct
colnames(data) <- c("TSK", "PSFC", "U10", "V10", "Q2", "RAINC", "RAINNC", "SNOW", "TSLB", "SMOIS", "DATETIME", "WIND_SPEED'
```

```
> data.frame(data)
      TSK     PSFC  U10  V10       Q2 RAINC RAINNC SNOW
1  274.15  97392.0  3.1  0.6 0.003930    NA      0    0
2  274.15  96298.0  2.6  1.5 0.003620     0      0    0
3  274.40  99340.0 -0.9  3.6 0.003890     0      0    0
4  282.40 100477.0  1.2  7.7 0.005240     0      0    0
5  274.15  98697.0  0.6  0.7 0.003670     0      0    0
6  274.15  99203.0 -0.3  3.5 0.004220     0      0    0
7  274.20  99758.0 -0.7  3.3 0.004080     0      0    0
8  274.15  98914.0  0.9  3.0 0.004120     0      0    0
9  281.30 100515.0  0.0  3.9 0.004650     0      0    0
10 274.15  96558.0  1.3  2.6 0.003820     0      0    0
11 279.20 100646.0  2.8  1.5 0.004650     0      0    0
12 274.15  97443.0 -0.1  3.9 0.004235     0      0    0
13 279.30 100688.0  2.9 -0.1 0.004880     0      0    0
14 274.15  94317.0  1.1  0.0 0.003840     0      0    0
15 274.15  96706.0 -0.3  2.8 0.003900     0      0    0
16 274.15  99419.0 -0.3  3.0 0.003930     0      0    0
17 274.15 100212.0 -0.9 -0.7 0.003990     0      0    0
18 274.15  98271.0  0.4 -1.1 0.003930     0      0    0
19 282.60 100397.0 -0.1  6.9 0.005110     0      0    0
20 274.15  95510.0  0.7  2.5 0.003830     0      0    0
21 274.15  98105.0  0.3  3.2 0.003810     0      0    0
22 274.15  96637.0  0.2  2.1 0.003860     0      0    0
23 274.15  98254.0 -2.6  1.6 0.004030     0      0    0
24 274.15  97857.0  2.6 -0.2 0.003900     0      0    0
25 274.15  97603.0 -0.1  2.1 0.003780     0      0    0
26 282.20 100533.0  2.6  7.0 0.005140     0      0    0
27 274.15  99494.0  0.5 -0.2 0.003610     0      0    0
28 274.15  98057.0  1.6 -1.9 0.003430     0      0    0
```

Figure 35: data frame of preprocessed data

Convert DATETIME to POSIXct format

```r
# Convert DATETIME to POSIXct format and other columns as necessary
data$DATETIME <- as.POSIXct(data$DATETIME, format = "%d.%m.%Y.%H.%M")
data$Day_Night <- as.factor(data$Day_Night)
data$Hour <- as.numeric(data$Hour)
data$Date <- as.Date(data$Date, format = "%m/%d/%Y")

# Display the structure of the dataset to verify changes
str(data)
```

```
> str(data)
spc_tbl_ [2,996 × 15] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ TSK       : num [1:2996] 274 274 274 282 274 ...
 $ PSFC      : num [1:2996] 97392 96298 99340 100477 98697 ...
 $ U10       : num [1:2996] 3.1 2.6 -0.9 1.2 0.6 -0.3 -0.7 0.9 0 1.3 ...
 $ V10       : num [1:2996] 0.6 1.5 3.6 7.7 0.7 3.5 3.3 3 3.9 2.6 ...
 $ Q2        : num [1:2996] 0.00393 0.00362 0.00389 0.00524 0.00367 0.00422 0.00408 0.00412 0.00465 0.00382 ...
 $ RAINC     : num [1:2996] NA 0 0 0 0 0 0 0 0 0 ...
 $ RAINNC    : num [1:2996] 0 0 0 0 0 0 0 0 0 0 ...
 $ SNOW      : num [1:2996] 0 0 0 0 0 0 0 0 0 0 ...
 $ TSLB      : num [1:2996] 278 278 278 273 280 ...
 $ SMOIS     : num [1:2996] 0.27 0.246 0.264 1 0.287 ...
 $ DATETIME  : POSIXct[1:2996], format: "2018-04-30 19:00:00" ...
 $ WIND_SPEED: num [1:2996] 3.16 3 3.71 7.79 0.92 3.51 3.37 3.13 3.9 2.91 ...
 $ Hour      : num [1:2996] 0 0 0 0 0 0 0 0 0 0 ...
 $ Day_Night : Factor w/ 2 levels "Daytime","Nighttime": 2 2 2 2 2 2 2 2 2 2 ...
 $ Date      : Date[1:2996], format: "2018-04-30" ...
 - attr(*, "spec")=
  .. cols(
  ..   TSK = col_double(),
  ..   PSFC = col_double(),
  ..   U10 = col_double(),
  ..   V10 = col_double(),
  ..   Q2 = col_double(),
  ..   RAINC = col_double(),
  ..   RAINNC = col_double(),
  ..   SNOW = col_double(),
  ..   TSLB = col_double(),
  ..   SMOIS = col_double(),
  ..   DATETIME = col_datetime(format = ""),
  ..   WIND_SPEED = col_double(),
  ..   Hour = col_double(),
  ..   Day_Night = col_character(),
  ..   Date = col_character()
  .. )
 - attr(*, "problems")=<externalptr>
```

Figure 36: data representation

Handling NA values in dataset

```r
# 1. Define the critical columns, including RAINC
critical_columns <- c("TSK", "PSFC", "SMOIS", "WIND_SPEED", "RAINC")

# 2. Remove rows with any NA values in the critical columns
data <- data %>%
  filter(!if_any(all_of(critical_columns), is.na))

# 3. Verify that there are no NA values left
na_summary <- colSums(is.na(data))
print("Remaining NA values per column:")
print(na_summary)

# Proceed to the next steps only if NA values are handled
if (all(na_summary == 0)) {
  cat("All NA values have been handled.\n")
} else {
  stop("NA values still exist in the dataset. Please check the handling process.")
}

# View the cleaned data
print(head(data))
```

```
> print("Remaining NA values per column:")
[1] "Remaining NA values per column:"
> print(na_summary)
        TSK       PSFC        U10        V10         Q2
          0          0          0          0          0
      RAINC     RAINNC       SNOW       TSLB      SMOIS
          0          0          0          0          0
   DATETIME WIND_SPEED       Hour  Day_Night       Date
          0          0          0          0          0
> # Proceed to the next steps only if NA values are handled
> if (all(na_summary == 0)) {
+   cat("All NA values have been handled.\n")
+ } else {
+   stop("NA values still exist in the dataset. Please check t
he handling process.")
+ }
All NA values have been handled.
> # View the cleaned data
> print(head(data))
# A tibble: 6 × 15
    TSK   PSFC   U10   V10      Q2 RAINC RAINNC  SNOW  TSLB
  <dbl>  <dbl> <dbl> <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
1  274.  96298   2.6   1.5 0.00362     0      0     0  278.
2  274.  99340  -0.9   3.6 0.00389     0      0     0  278.
3  282. 100477   1.2   7.7 0.00524     0      0     0  273.
4  274.  98697   0.6   0.7 0.00367     0      0     0  280.
5  274.  99203  -0.3   3.5 0.00422     0      0     0  278.
6  274.  99758  -0.7   3.3 0.00408     0      0     0  278.
```

Figure 37: Visualizing NA values

Splitting the dataset

```
# Split the data into training (70%) and testing (30%) sets
set.seed(123)  # For reproducibility
trainIndex <- createDataPartition(data$TSK, p = .7, list = FALSE)
trainData <- data[trainIndex,]
testData <- data[-trainIndex,]

# Check for and handle any missing values by filling NA with the mean of the previous two values
fill_na_with_mean <- function(x) {
  na_index <- which(is.na(x))
  for (i in na_index) {
    if (i > 2) {
      x[i] <- mean(c(x[i - 1], x[i - 2]), na.rm = TRUE)
    }
  }
  return(x)
}

# Check for zero-variance columns in the training data
zero_variance_cols <- nearZeroVar(trainData, saveMetrics = TRUE)
zero_variance_cols <- rownames(zero_variance_cols[zero_variance_cols$zeroVar == TRUE, ])

# Remove zero-variance columns from the training and testing data
trainData <- trainData[, !colnames(trainData) %in% zero_variance_cols]
testData <- testData[, !colnames(testData) %in% zero_variance_cols]

# Now, define the numeric columns again after removing zero-variance columns
numeric_cols <- names(trainData)[sapply(trainData, is.numeric)]

# Standardize the numeric columns
preProc <- preProcess(trainData[, numeric_cols], method = c("center", "scale"))
trainData[, numeric_cols] <- predict(preProc, trainData[, numeric_cols])
testData[, numeric_cols] <- predict(preProc, testData[, numeric_cols])

# Verify the preprocessed data
str(trainData)
str(testData)
```

```
> str(trainData)
tibble [2,098 × 14] (S3: tbl_df/tbl/data.frame)
 $ TSK       : num [1:2098] -2.004 -2.058 -2.072 -0.126 -2.072 ...
 $ PSFC      : num [1:2098] 0.22274 0.44969 -0.00856 0.86071 -1.28777 ...
 $ U10       : num [1:2098] -1.072 -1.005 -0.473 -0.772 -0.34 ...
 $ V10       : num [1:2098] -0.187 -0.2867 -0.3863 -0.0874 -0.5192 ...
 $ Q2        : num [1:2098] -1.77 -1.61 -1.58 -1.13 -1.83 ...
 $ RAINC     : num [1:2098] -0.238 -0.238 -0.238 -0.238 -0.238 ...
 $ RAINNC    : num [1:2098] -0.55 -0.55 -0.55 -0.55 -0.55 ...
 $ TSLB      : num [1:2098] -0.343 -0.32 -0.23 -1.354 -0.298 ...
 $ SMOIS     : num [1:2098] -0.618 -0.613 -0.523 1.678 -0.675 ...
 $ DATETIME  : POSIXct[1:2098], format: "2018-04-30 19:00:00" ...
 $ WIND_SPEED: num [1:2098] -0.994 -1.142 -1.247 -0.912 -1.343 ...
 $ Hour      : num [1:2098] -1.48 -1.48 -1.48 -1.48 -1.48 ...
 $ Day_Night : Factor w/ 2 levels "Daytime","Nighttime": 2 2 2 2 2 2 2 2 2 2 ...
 $ Date      : Date[1:2098], format: "2018-04-30" ...
> str(testData)
tibble [897 × 14] (S3: tbl_df/tbl/data.frame)
 $ TSK       : num [1:897] -2.072 0.174 -2.072 -2.072 -2.072 ...
 $ PSFC      : num [1:897] -1.429 0.84 -0.126 0.148 -0.807 ...
 $ U10       : num [1:897] 0.0929 -0.373 -0.5727 -0.8722 -0.8057 ...
 $ V10       : num [1:897] -0.8846 1.175 -1.1504 -0.2202 -0.0874 ...
 $ Q2        : num [1:897] -2 -0.626 -1.958 -1.491 -1.479 ...
 $ RAINC     : num [1:897] -0.238 -0.238 -0.238 -0.238 -0.238 ...
 $ RAINNC    : num [1:897] -0.55 -0.55 -0.55 -0.55 -0.55 ...
 $ TSLB      : num [1:897] -0.2529 -1.3539 0.0841 -0.3878 -0.3878 ...
 $ SMOIS     : num [1:897] -0.675 1.678 -0.545 -0.56 -0.623 ...
 $ DATETIME  : POSIXct[1:897], format: "2018-04-30 19:00:00" ...
 $ WIND_SPEED: num [1:897] -1.303 0.782 -2.209 -1.081 -0.912 ...
 $ Hour      : num [1:897] -1.48 -1.48 -1.48 -1.48 -1.48 ...
 $ Day_Night : Factor w/ 2 levels "Daytime","Nighttime": 2 2 2 2 2 2 2 2 2 2 ...
 $ Date      : Date[1:897], format: "2018-04-30" ...
```

Figure 38: Visualizing train and test dataset

7. <u>Statistical Analysis:</u>

Descriptive Statistics and Correlation Analysis

```r
# Summary statistics
summary(data)

# Correlation matrix and plot
cor_matrix <- cor(data[, numeric_cols], use = "complete.obs")
corrplot(cor_matrix, method = "circle")

# Specific correlation between TSK, PSFC, and other variables
cor(data$TSK, data$PSFC, use = "complete.obs")
```

```
> summary(data)
      TSK             PSFC              U10               V10                Q2                 RAINC              RAINNC      
 Min.   :274.1   Min.   : 93633   Min.   :-6.000   Min.   :-3.150   Min.   :0.003090   Min.   :0.0000   Min.   : 0.000  
 1st Qu.:279.7   1st Qu.: 97672   1st Qu.: 0.300   1st Qu.: 2.400   1st Qu.:0.005080   1st Qu.:0.0000   1st Qu.: 0.000  
 Median :281.5   Median : 99095   Median : 2.500   Median : 4.200   Median :0.006100   Median :0.0000   Median : 0.200  
 Mean   :281.7   Mean   : 98938   Mean   : 2.318   Mean   : 4.175   Mean   :0.005981   Mean   :0.1084   Mean   : 1.301  
 3rd Qu.:283.4   3rd Qu.:100362   3rd Qu.: 4.500   3rd Qu.: 6.100   3rd Qu.:0.006900   3rd Qu.:0.0000   3rd Qu.: 1.400  
 Max.   :288.9   Max.   :102423   Max.   :10.800   Max.   :11.650   Max.   :0.009290   Max.   :5.3000   Max.   :16.700  
      SNOW         TSLB            SMOIS           DATETIME                         WIND_SPEED           Hour      
 Min.   :0   Min.   :273.2   Min.   :0.2220   Min.   :2018-04-30 19:00:00.00   Min.   : 0.220   Min.   : 0.00  
 1st Qu.:0   1st Qu.:273.2   1st Qu.:0.2609   1st Qu.:2018-05-03 07:00:00.00   1st Qu.: 4.300   1st Qu.: 3.00  
 Median :0   Median :279.6   Median :0.2798   Median :2018-05-05 19:00:00.00   Median : 5.910   Median : 9.00  
 Mean   :0   Mean   :279.2   Mean   :0.4667   Mean   :2018-05-05 19:04:48.48   Mean   : 5.985   Mean   :10.38  
 3rd Qu.:0   3rd Qu.:282.4   3rd Qu.:1.0000   3rd Qu.:2018-05-08 07:00:00.00   3rd Qu.: 7.460   3rd Qu.:15.00  
 Max.   :0   Max.   :290.8   Max.   :1.0000   Max.   :2018-05-10 19:00:00.00   Max.   :12.200   Max.   :21.00  
     Day_Night          Date           
 Daytime  :1480   Min.   :2018-04-30  
 Nighttime:1515   1st Qu.:2018-05-03  
                  Median :2018-05-05  
                  Mean   :2018-05-05  
                  3rd Qu.:2018-05-08  
                  Max.   :2018-05-10  
```

Figure 39: Data Summary

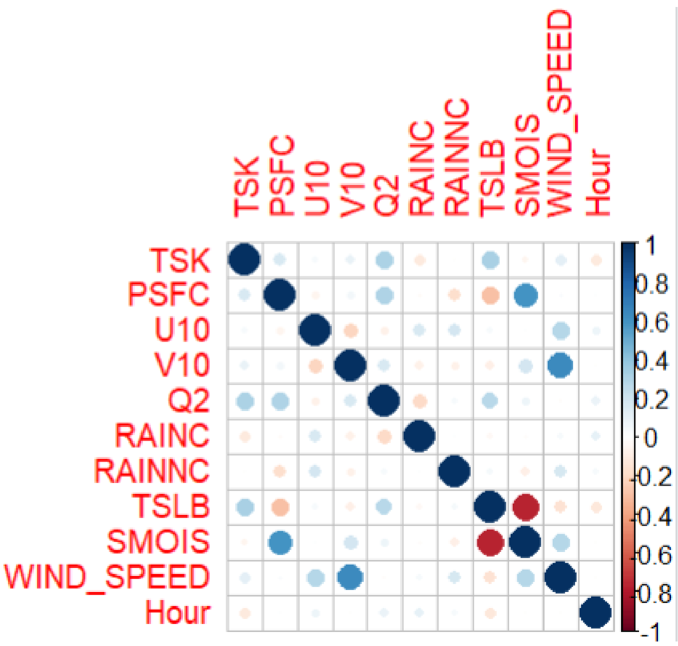

Figure 40: Correlation Plot

```
> cor(data$TSK, data$PSFC, use = "complete.obs")
[1] 0.1633406
```

Figure 41: Correlation between TSK PSFC and other variables

T-Test for Daytime vs Nighttime

```
daytime_data <- filter(data, Day_Night == "Daytime")
nighttime_data <- filter(data, Day_Night == "Nighttime")
t_test_result <- t.test(daytime_data$PSFC, nighttime_data$PSFC)
print(t_test_result)
```

```
        Welch Two Sample t-test

data:  daytime_data$PSFC and nighttime_data$PSFC
t = -1.0407, df = 2992.9, p-value = 0.2981
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -201.26062   61.69421
sample estimates:
mean of x mean of y 
 98902.91  98972.69 
```

Figure 42: t-test results

8. Machine Learning Models (Training and Evaluation)

8.1 Linear Regression Model

Training the model

```
lm_model <- lm(TSK ~ U10 + V10 + Q2 + RAINC + RAINNC + PSFC + SMOIS, data = trainData)
```

Evaluate the model

```
lm_preds <- predict(lm_model, testData)
```

Re-run the prediction and evaluation

```
lm_preds_clean <- predict(lm_model, testData)
lm_rmse_clean <- sqrt(mean((lm_preds - testData$TSK)^2))
cat("Linear Regression RMSE (after removing problematic rows):", lm_rmse_clean, "\n")
```

```
> cat("Linear Regression RMSE :", lm_rmse_clean, "\n")
Linear Regression RMSE : 0.9608081
```

Figure 43: Linear regression root mean square error

8.2 Random Forest

Training the model

```
rf_model <- randomForest(TSK ~ U10 + V10 + Q2 + RAINC + RAINNC + PSFC + SMOIS, data = trainData, ntree = 100)
```

Generating the prediction in the test data

```
rf_preds <- predict(rf_model, testData)
```

Evaluating the model

```
valid_index <- complete.cases(testData$TSK, rf_preds)
rf_rmse <- sqrt(mean((rf_preds[valid_index] - testData$TSK[valid_index])^2))

# Print the RMSE
cat("Random Forest RMSE:", rf_rmse, "\n")
```

```
> cat("Random Forest RMSE:", rf_rmse, "\n")
Random Forest RMSE: 0.7123892
>
```

Figure 44: Root mean square error of random forest model

8.3 Support Vector Machine (SVM)

Train the model:

```
svm_model <- svm(TSK ~ U10 + V10 + Q2 + RAINC + RAINNC + PSFC + SMOIS, data = trainData)
```

Evaluate the SVM model:

```
svm_preds <- predict(svm_model, testData)
svm_rmse <- sqrt(mean((svm_preds - testData$TSK)^2))
cat("SVM RMSE:", svm_rmse, "\n")
```

```
> cat("SVM RMSE:", svm_rmse, "\n")
SVM RMSE: 0.7822935
```

Figure 45: SVM root mean square error

8.4 Gradient Boosting Machine

Train the model

```
gbm_model <- gbm(TSK ~ U10 + V10 + Q2 + RAINC + RAINNC + PSFC + SMOIS, data = trainData, distribution = "gaussian", n.trees = 100, interaction.depth = 3)
```

Evaluate the model

```
gbm_preds <- predict(gbm_model, testData, n.trees = 100)
gbm_rmse <- sqrt(mean((gbm_preds - testData$TSK)^2))
cat("GBM RMSE:", gbm_rmse, "\n")
```

```
> cat("GBM RMSE:", gbm_rmse, "\n")
GBM RMSE: 0.806545
>
```

Figure 46: GBM root mean square error

9. Model Comparison and Selection

Compare the RMSE of all models

```
cat("Linear Regression RMSE:", lm_rmse_clean, "\n")
cat("Random Forest RMSE:", rf_rmse, "\n")
cat("SVM RMSE:", svm_rmse, "\n")
cat("GBM RMSE:", gbm_rmse, "\n")
```

```
> cat("Linear Regression RMSE:", lm_rmse_clean, "\n")
Linear Regression RMSE: 0.9608081
> cat("Random Forest RMSE:", rf_rmse, "\n")
Random Forest RMSE: 0.7096459
> cat("SVM RMSE:", svm_rmse, "\n")
SVM RMSE: 0.8049479
> cat("GBM RMSE:", gbm_rmse, "\n")
GBM RMSE: 0.761597
```

Figure 47: Comparison of all the models

Select the best model

```
best_model <- which.min(c(lm_rmse_clean, rf_rmse, svm_rmse, gbm_rmse))
model_names <- c("Linear Regression", "Random Forest", "SVM", "GBM")
cat("Best model based on RMSE is:", model_names[best_model], "\n")
```

```
> cat("Best model based on RMSE is:", model_names[best_model], "\n")
Best model based on RMSE is: Random Forest
```

Figure 48: Best model

Final model evaluation on test data

```
final_model <- rf_model
final_preds <- predict(final_model, testData)
final_rmse <- sqrt(mean((final_preds - testData$TSK)^2))
cat("Final Model RMSE on Test Data:", final_rmse, "\n")
```

```
> cat("Final Model RMSE on Test Data:", final_rmse, "\n")
Final Model RMSE on Test Data: 0.7096459
```

Figure 49: Final model evaluation on test data

Analyze Residuals

```
residuals <- testData$TSK - final_preds
plot(residuals, main = "Residuals of Final Model", ylab = "Residuals")
```

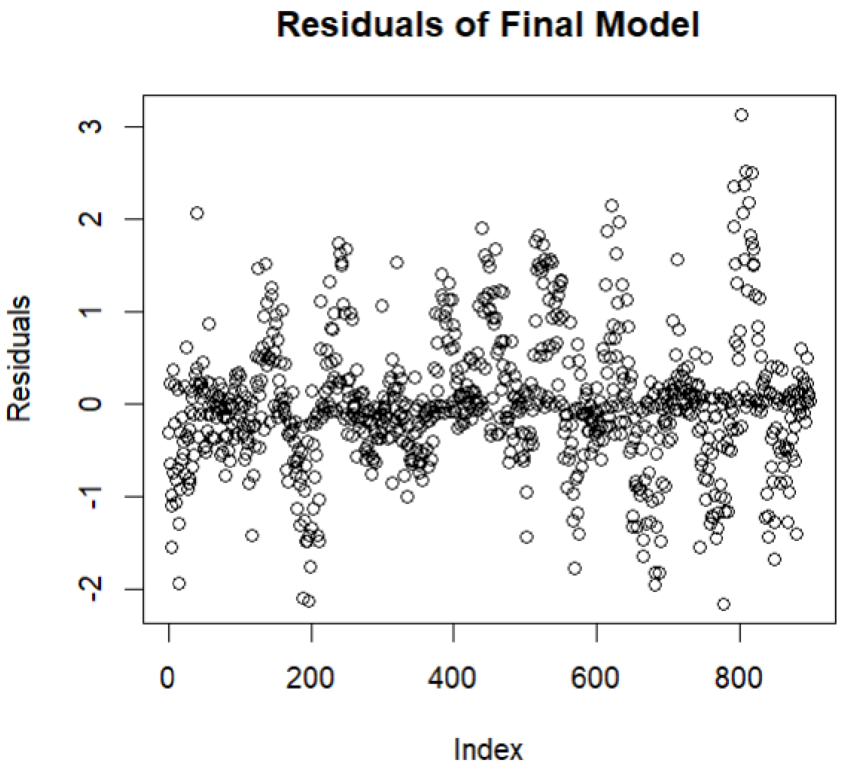

Figure 50: Residual of final model

10. Adequate interpretation and justification of results

How Well the Model Worked

Linear Regression: The RMSE for the linear regression model is 0.96080 This model has a moderate impact on the error in predicting the target variable (TSK). Linear regression assumes a linear relationship between the predictors and the target variable. This RMSE indicates that the relationship between the variables might not be linear, which leads to some prediction inaccuracies.

Random Forest: The RMSE for the random forest model is 0.7096, the lowest among all the models tested. This shows that the random forest model has the best performance in predicting TSK. Random forests work well to capture complex non-linear interactions between variables, which explains why it did better than the other models.

Support Vector Machine (SVM): The SVM model has an RMSE of 0.7823, which beats linear regression but falls short of random forest. SVMs excel at handling high-dimensional data but might have trouble with non-linear relationships depending on the kernel used.

Gradient Boosting Machine (GBM): The RMSE for the GBM model is 0.80650.80650.8065. GBMs perform well because they can handle different kinds of relationships and interactions in data. But in this case, it didn't do as well as the random forest model. This might be because we didn't use enough trees or didn't make them deep enough in the model.

Interpretation

Random Forest as the Best Model: The random forest model has a lower RMSE showing it's the most accurate in predicting TSK (temperature at skin level) out of all the tested models. The model's knack for dealing with non-linear relationships and interactions between variables helped it come out on top. Weather data often has tricky patterns, so the random forest's toughness makes it a good fit for this job.

Justification for Model Selection: We should pick the random forest model as the final one to analyze further or put into action, since it did the best. It gives the most trustworthy predictions, which can lead to smarter choices when it comes to weather forecasting and related uses.

11. Practical Applications for Organizational Decision-Making

Usefulness in the Context of an Environmental Monitoring Organization

Spot-on Weather Forecasts: Weather forecasting groups or environmental monitoring agencies need to nail their predictions of things like TSK. These forecasts play a big role in wider climate models and help give people and businesses up-to-date and reliable weather info. Take the random forest model, for instance. It could become part of the organization's prediction tools to boost how well they forecast temperatures in the short term.

Decision Support in Agriculture and Disaster Management: Groups that work in farming or handle disasters could use better weather forecasts to make smart choices. For example, farmers can plan when to water crops, protect them from frost, or choose the best time to plant based on more exact temperature predictions. For disaster teams good forecasts can help them get ready for big weather events, like hot or cold spells. This can save lives and protect stuff people own.

Energy Sector Applications: In the energy world especially for power companies, knowing the temperature ahead of time is key to guess how much power people will need. The random forest model's forecasts could help predict when energy use will be highest during very hot or cold times. This lets power grid managers do a better job and stop blackouts from happening.

Urban Planning and Public Health: City planners could apply these findings to gain a deeper understanding of urban heat islands and to plan steps to cut down heat in key areas. In the same way, health agencies could use the forecasts to send out heat warnings, which would help to prevent illnesses caused by high temperatures.

12. Conclusion

In this project, we started a deep dive to analyze weather data. We wanted to predict the Temperature at Skin Level (TSK). To do this, we used different weather variables from the Weather Research and Forecasting (WRF) model. We kicked things off by cleaning up the dataset. This meant dealing with missing data, changing data types, and fine-tuning features so we could analyze them better later on.

We then used several machine learning models to predict TSK. These included Linear Regression, Random Forest, Support Vector Machine (SVM), and Gradient Boosting Machine (GBM). To evaluate how well these models performed, we used the Root Mean Square Error (RMSE) metric. The Random Forest model came out on top with the lowest RMSE (0.7096) showing it was better at making predictions than the other models. We also created a lot of visuals to look at the data. We explored how different variables were spread out how they related to each other, and if there were any connections between them. These visuals gave us useful insights into the structure of our data. For example, we could see how wind speeds were distributed, how surface pressure and temperature were linked, and how TSK changed at different times of the day.

The results showed that the Random Forest model worked best for this dataset because it can grasp complex relationships between variables. These results are helpful for groups that deal with weather forecasts, climate studies, or keeping an eye on the environment. In these fields, getting the temperature right is key to making good choices.

To wrap up thorough data cleanup, picking the right model, and clear visuals have set a solid base for accurate weather predictions. We can fine-tune and apply these methods and models to similar data sets helping weather-related industries make better decisions.

13. References

https://www.w3schools.com/r/

https://ggplot2.tidyverse.org/

https://www.geeksforgeeks.org/data-visualization-in-r/

https://www.geeksforgeeks.org/exploratory-data-analysis-in-r-programming/

https://www.geeksforgeeks.org/data-cleaning-in-r/

https://dplyr.tidyverse.org/articles/dplyr.html

https://builtin.com/data-science/random-forest-algorithm

https://www.geeksforgeeks.org/ml-linear-regression/

https://www.geeksforgeeks.org/support-vector-machine-algorithm/

Word Count (excluding code, references, figures, tables, and appendices)

2097 words