London Bike Sharing Data Analysis in R

SDA Team @ Federico II

2023-07-05

1. Problem Statmente

Londoners can use a bicycle rental system for commuting within the city every day. These trips can be to get to work or school or to go sightseeing on a pleasant summer day. The number of rented bicycles depends on important factors such as air temperature, air humidity, time of day, etc. In this project, we are trying to predict the number of bicycles that can be rented every hour of the day.

2. Data Understanding

The London bike sharing data available on Kaggle provides information about the bike sharing system in London. The dataset contains information about the bike rides taken by users of the system, including the start and end time of each ride, the duration of the ride, and the start and end station of each ride.

The dataset contains over 17,414 records spanning from 2015-01-04 to 2017-01-03. This data includes information about the weather conditions at the time of each ride, such as temperature, humidity, wind speed, and precipitation. Additionally, it includes information about public holidays and events that may have affected bike usage. The description of each variable in this data is provided as below:

- **timestamp**: Representing timestamp of bike share
- **cnt**: Representing total number of bike shares
- **t1**: The temperature in celsius.
- **t2**: The apparent ("feels-like") temperature in celsius.
- **hum**: The humidity level
- **wind_speed**: The wind speed
- **weather_code**: A categorical value indicating the weather situation (1:clear, 2:mist/cloud, 3:light rain/snow, 4:heavy rain/hail/snow/fog)
- **is_holiday**: A binary value indicating whether or not the day is a holiday
- **is_weekend**: A binary value indicating whether or not the day is a weekend
- **season**: A numerically encoded value indicating the season (0:spring, 1:summer, 2:fall, 3:winter)

Also. **cnt** represents the label (the y value) our model must be trained to predict. The other columns are potential features (x values). Therefore, the total number of bike shares (**cnt**) is the response variable in this data and the main purpose of this report is to investigate the effect of independent variables on the response.

```
# Define the variables and their titles
variables <- c("timestamp", "cnt", "t1", "t2", "hum", "wind_speed",
"weather_code",
               "is_holiday", "is_weekend", "season", "year", "month", "day",
"hour")
titles <- c("Timestamp", "Count", "Temperature 1", "Temperature 2: Feel-
like", "Humidity",
            "Wind Speed", "Weather Code", "Is Holiday", "Is Weekend",
"Season",
            "Year", "Month", "Day", "Hour")

# Create a data frame with the variables and titles
variable_table <- data.frame(Variable = variables, Title = titles)

# Print the table
print(variable_table)
```

```
##        Variable                    Title
## 1     timestamp                Timestamp
## 2           cnt                    Count
## 3            t1            Temperature 1
## 4            t2 Temperature 2: Feel-like
## 5           hum                 Humidity
## 6    wind_speed               Wind Speed
## 7  weather_code             Weather Code
## 8    is_holiday               Is Holiday
## 9    is_weekend               Is Weekend
## 10       season                   Season
## 11         year                     Year
## 12        month                    Month
## 13          day                      Day
## 14         hour                     Hour
```

3. Data Pre-Processing and Cleaning

The data is loaded into R using the below R codes and all the required libraries in this report are loaded as well:

```
library(dplyr)
library(skimr)
library(ggplot2)
library(lubridate)
library(tibble)
library(caret)
library(e1071)
library(rpart)
library(PerformanceAnalytics)
library(xgboost)
library(randomForest)
library(scales)
library(gridExtra)


# set a theme for ggplots
theme_set(
  theme_bw(base_size = 15)+
  theme(plot.title.position = "plot")
)

# load our dataset
bike <- read.csv(file = "london_merged.csv", header = T)

# print the dimension of the data
dim(bike)

## [1] 17414    10

str(bike)

## 'data.frame':    17414 obs. of  10 variables:
##  $ timestamp   : chr  "2015-01-04 00:00:00" "2015-01-04 01:00:00" "2015-
01-04 02:00:00" "2015-01-04 03:00:00" ...
##  $ cnt         : int  182 138 134 72 47 46 51 75 131 301 ...
##  $ t1          : num  3 3 2.5 2 2 2 1 1 1.5 2 ...
##  $ t2          : num  2 2.5 2.5 2 0 2 -1 -1 -1 -0.5 ...
##  $ hum         : num  93 93 96.5 100 93 93 100 100 96.5 100 ...
##  $ wind_speed  : num  6 5 0 0 6.5 4 7 7 8 9 ...
##  $ weather_code: num  3 1 1 1 1 1 4 4 4 3 ...
##  $ is_holiday  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ season      : num  3 3 3 3 3 3 3 3 3 3 ...
```

get a summary:

```
skim(bike)
```

Data summary

Name	bike
Number of rows	17414
Number of columns	10
Column type frequency:	
character	1
numeric	9

Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
timestamp	0	1	19	19	0	17414	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
cnt	0	1	1143.10	1085.11	0.0	257	844.0	1671.75	7860.0	▇▂▁▁▁
t1	0	1	12.47	5.57	-1.5	8	12.5	16.00	34.0	▁▇▇▂▁
t2	0	1	11.52	6.62	-6.0	6	12.5	16.00	34.0	▁▅▇▂▁
hum	0	1	72.32	14.31	20.5	63	74.5	83.00	100.0	▁▂▅▇▆
wind_speed	0	1	15.91	7.89	0.0	10	15.0	20.50	56.5	▅▇▂▁▁
weather_code	0	1	2.72	2.34	1.0	1	2.0	3.00	26.0	▇▁▁▁▁
is_holiday	0	1	0.02	0.15	0.0	0	0.0	0.00	1.0	▇▁▁▁▁
is_weekend	0	1	0.29	0.45	0.0	0	0.0	1.00	1.0	▇▁▁▁▃
season	0	1	1.49	1.12	0.0	0	1.0	2.00	3.0	▇▇▁▇▇

In order to pre-process the data, we should pay attention to a few aspects of the data including the variable types, number of missing data in each column, scaling the variables (if necessary), excluding unused variables etc.

3-1 Extract year, month, day and hour variable from the timestamp column

```
# add year, month, day and hour to the data
bike <- bike %>% mutate(
  year=year(bike$timestamp),
  month=month(bike$timestamp),
  day=day(bike$timestamp),
  hour=hour(bike$timestamp))
```

Take a closer look at our dataset:

```
summary(bike)
```

```
##   timestamp                cnt              t1              t2
##  Length:17414       Min.   :   0   Min.   :-1.50   Min.   :-6.00
##  Class :character   1st Qu.: 257   1st Qu.: 8.00   1st Qu.: 6.00
##  Mode  :character   Median : 844   Median :12.50   Median :12.50
##                     Mean   :1143   Mean   :12.47   Mean   :11.52
##                     3rd Qu.:1672   3rd Qu.:16.00   3rd Qu.:16.00
##                     Max.   :7860   Max.   :34.00   Max.   :34.00
##       hum           wind_speed     weather_code     is_holiday
##  Min.   : 20.50   Min.   : 0.00   Min.   : 1.000   Min.   :0.00000
##  1st Qu.: 63.00   1st Qu.:10.00   1st Qu.: 1.000   1st Qu.:0.00000
##  Median : 74.50   Median :15.00   Median : 2.000   Median :0.00000
##  Mean   : 72.32   Mean   :15.91   Mean   : 2.723   Mean   :0.02205
##  3rd Qu.: 83.00   3rd Qu.:20.50   3rd Qu.: 3.000   3rd Qu.:0.00000
##  Max.   :100.00   Max.   :56.50   Max.   :26.000   Max.   :1.00000
##    is_weekend         season           year          month
##  Min.   :0.0000   Min.   :0.000   Min.   :2015   Min.   : 1.000
##  1st Qu.:0.0000   1st Qu.:0.000   1st Qu.:2015   1st Qu.: 4.000
##  Median :0.0000   Median :1.000   Median :2016   Median : 7.000
##  Mean   :0.2854   Mean   :1.492   Mean   :2016   Mean   : 6.515
##  3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:2016   3rd Qu.:10.000
##  Max.   :1.0000   Max.   :3.000   Max.   :2017   Max.   :12.000
##       day             hour
##  Min.   : 1.00   Min.   : 0.00
##  1st Qu.: 8.00   1st Qu.: 6.00
##  Median :16.00   Median :12.00
##  Mean   :15.75   Mean   :11.51
##  3rd Qu.:23.00   3rd Qu.:18.00
##  Max.   :31.00   Max.   :23.00
```

3-2 Check for Number of Missing Values in the Data

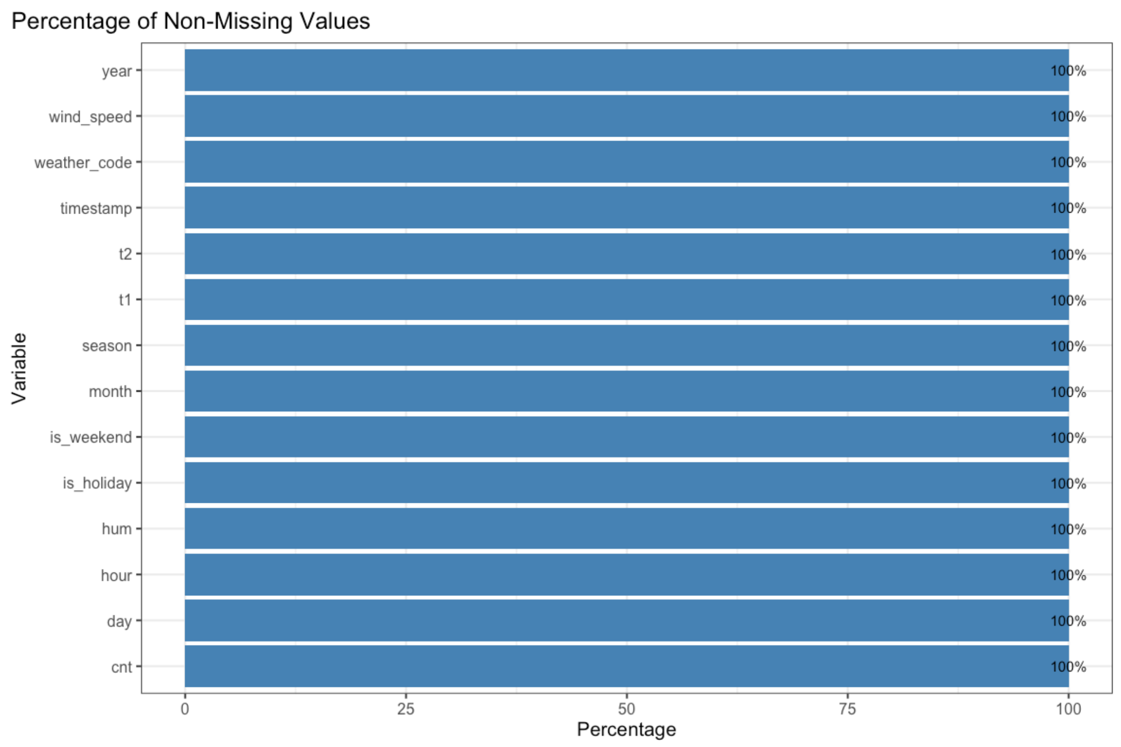
In order to check for the number of missing values in each column (variable), the following code is used:

```
# count number of missing values in each column
nm <- colSums(is.na(bike))
# convert missing count information to a table
tibble(Variable=names(nm), Number_of_Missing=as.vector(nm)) %>%
  knitr::kable()
```

Variable	Number_of_Missing
timestamp	0
cnt	0
t1	0
t2	0

Variable	Number_of_Missing
hum	0
wind_speed	0
weather_code	0
is_holiday	0
is_weekend	0
season	0
year	0
month	0
day	0
hour	0

```r
non_missing_percentage <- colMeans(!is.na(bike)) * 100
non_missing_df <- data.frame(Variable = names(non_missing_percentage),
Percentage = non_missing_percentage)

ggplot(data = non_missing_df, aes(x = Variable, y = Percentage)) +
  geom_bar(stat = "identity", fill = "steelblue", position =
position_stack(vjust = 1)) +
  geom_text(aes(label = paste0(round(Percentage), "%")), vjust = 0.5) +
  labs(title = "Percentage of Non-Missing Values", x = "Variable", y =
"Percentage") +
  coord_flip()
```



- As we can see in the above table, there are no missing values in the data.

```
# Define categorical variables
categorical_vars <- c("weather_code", "is_holiday", "is_weekend", "season",
"year", "month", "day", "hour")

# Create a tibble for categorical variables
categorical_table <- tibble(
  Categorical_Variables = categorical_vars
)

print(categorical_table)

## # A tibble: 8 × 1
##   Categorical_Variables
##   <chr>
## 1 weather_code
## 2 is_holiday
## 3 is_weekend
## 4 season
## 5 year
## 6 month
## 7 day
## 8 hour

# Define numerical variables
numerical_vars <- c("cnt", "t1", "t2", "hum", "wind_speed")

# Create a tibble for numerical variables
numerical_table <- tibble(
  Numerical_Variables = numerical_vars
)

print(numerical_table)

## # A tibble: 5 × 1
##   Numerical_Variables
##   <chr>
## 1 cnt
## 2 t1
## 3 t2
## 4 hum
## 5 wind_speed

# Get the structure of the dataset
data_info <- str(bike)

## 'data.frame':    17414 obs. of  14 variables:
##  $ timestamp   : chr  "2015-01-04 00:00:00" "2015-01-04 01:00:00" "2015-
01-04 02:00:00" "2015-01-04 03:00:00" ...
##  $ cnt         : int  182 138 134 72 47 46 51 75 131 301 ...
##  $ t1          : num  3 3 2.5 2 2 2 1 1 1.5 2 ...
##  $ t2          : num  2 2.5 2.5 2 0 2 -1 -1 -1 -0.5 ...
##  $ hum         : num  93 93 96.5 100 93 93 100 100 96.5 100 ...
##  $ wind_speed  : num  6 5 0 0 6.5 4 7 7 8 9 ...
##  $ weather_code: num  3 1 1 1 1 1 4 4 4 3 ...
##  $ is_holiday  : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ is_weekend  : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ season      : num  3 3 3 3 3 3 3 3 3 3 ...
##  $ year        : num  2015 2015 2015 2015 2015 ...
##  $ month       : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ day         : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ hour        : int  0 1 2 3 4 5 6 7 8 9 ...

# Extract the variable types
variable_types <- data_info$info$type

# Create a data frame with variable names and types
variable_table <- data.frame(Variable = names(variable_types), Type =
variable_types, stringsAsFactors = FALSE)

# Divide variables into categorical and numerical
categorical_vars <- variable_table$Variable[variable_table$Type ==
"character"]
numerical_vars <- variable_table$Variable[variable_table$Type != "character"]

# Create tables for categorical and numerical variables
categorical_table <- data.frame(Categorical_Variables = categorical_vars)
numerical_table <- data.frame(Numerical_Variables = numerical_vars)

# Print the tables
print(categorical_table)

## data frame with 0 columns and 0 rows

print(numerical_table)

## data frame with 0 columns and 0 rows

# count number of missing values in each column
nm <- colSums(is.na(bike))
# convert missing count information to a table
tibble(Variable=names(nm), Number_of_Missing=as.vector(nm)) %>%
  knitr::kable()
```

Variable	Number_of_Missing
timestamp	0
cnt	0
t1	0
t2	0

Variable	Number_of_Missing
hum	0
wind_speed	0
weather_code	0
is_holiday	0
is_weekend	0
season	0
year	0
month	0
day	0
hour	0

3-4 Define Variable Types

In the following codes, a suitable variable type is assigned to each variable based on the description provided in the previous sections.

```
# replace the values more than 4 in weather_code with 4
bike$weather_code[bike$weather_code > 4] <- 4
# variable type setting
# bike$timestamp <- as.Date(bike$timestamp)
bike$weather_code <- as.factor(bike$weather_code)
bike$is_holiday <- as.factor(bike$is_holiday)
bike$is_weekend <- as.factor(bike$is_weekend)
bike$season <- as.factor(bike$season)
bike$year <- as.factor(bike$year)
bike$month <- as.factor(bike$month)
bike$day <- as.factor(bike$day)
bike$hour <- as.factor(bike$hour)
# print the first 10 rows of the data
head(bike, n=10)
```

```
##               timestamp cnt  t1   t2   hum wind_speed weather_code
is_holiday
## 1  2015-01-04 00:00:00 182 3.0  2.0  93.0        6.0            3
0
## 2  2015-01-04 01:00:00 138 3.0  2.5  93.0        5.0            1
0
## 3  2015-01-04 02:00:00 134 2.5  2.5  96.5        0.0            1
0
## 4  2015-01-04 03:00:00  72 2.0  2.0 100.0        0.0            1
0
## 5  2015-01-04 04:00:00  47 2.0  0.0  93.0        6.5            1
0
## 6  2015-01-04 05:00:00  46 2.0  2.0  93.0        4.0            1
0
```

```
## 7  2015-01-04 06:00:00  51 1.0 -1.0 100.0        7.0            4
0
## 8  2015-01-04 07:00:00  75 1.0 -1.0 100.0        7.0            4
0
## 9  2015-01-04 08:00:00 131 1.5 -1.0  96.5        8.0            4
0
## 10 2015-01-04 09:00:00 301 2.0 -0.5 100.0        9.0            3
0
##    is_weekend season year month day hour
## 1           1      3 2015     1   4    0
## 2           1      3 2015     1   4    1
## 3           1      3 2015     1   4    2
## 4           1      3 2015     1   4    3
## 5           1      3 2015     1   4    4
## 6           1      3 2015     1   4    5
## 7           1      3 2015     1   4    6
## 8           1      3 2015     1   4    7
## 9           1      3 2015     1   4    8
## 10          1      3 2015     1   4    9
```

```
# print the last 10 rows of the data
tail(bike, n=10)
```

```
##                 timestamp  cnt  t1  t2  hum wind_speed weather_code
is_holiday
## 17405 2017-01-03 14:00:00  765 6.0 2.0 73.5         22            3
0
## 17406 2017-01-03 15:00:00  845 6.0 2.0 71.0         27            4
0
## 17407 2017-01-03 16:00:00 1201 6.0 2.0 71.0         26            4
0
## 17408 2017-01-03 17:00:00 2742 6.0 2.0 73.5         21            3
0
## 17409 2017-01-03 18:00:00 2220 5.0 1.0 81.0         22            2
0
## 17410 2017-01-03 19:00:00 1042 5.0 1.0 81.0         19            3
0
## 17411 2017-01-03 20:00:00  541 5.0 1.0 81.0         21            4
0
## 17412 2017-01-03 21:00:00  337 5.5 1.5 78.5         24            4
0
## 17413 2017-01-03 22:00:00  224 5.5 1.5 76.0         23            4
0
## 17414 2017-01-03 23:00:00  139 5.0 1.0 76.0         22            2
0
##       is_weekend season year month day hour
## 17405          0      3 2017     1   3   14
## 17406          0      3 2017     1   3   15
## 17407          0      3 2017     1   3   16
## 17408          0      3 2017     1   3   17
## 17409          0      3 2017     1   3   18
## 17410          0      3 2017     1   3   19
## 17411          0      3 2017     1   3   20
## 17412          0      3 2017     1   3   21
## 17413          0      3 2017     1   3   22
## 17414          0      3 2017     1   3   23
```

4. Descriptive Statistics

4-1 Data Summarization

```
# data summarization
skim(bike)
```

Data summary

Name	bike
Number of rows	17414
Number of columns	14
_______________________	
Column type frequency:	
character	1
factor	8
numeric	5
________________________	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
timestamp	0	1	19	19	0	17414	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
weather_code	0	1	FALSE	4	1: 6150, 2: 4034, 4: 3679, 3: 3551
is_holiday	0	1	FALSE	2	0: 17030, 1: 384
is_weekend	0	1	FALSE	2	0: 12444, 1: 4970
season	0	1	FALSE	4	0: 4394, 1: 4387, 3: 4330, 2: 4303
year	0	1	FALSE	3	201: 8699, 201: 8643, 201: 72
month	0	1	FALSE	12	5: 1488, 1: 1487, 8: 1484,

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
day	0	1	FALSE	31	12: 1484 6: 576, 14: 576, 21: 576, 22: 576
hour	0	1	FALSE	24	16: 730, 12: 729, 15: 729, 13: 728

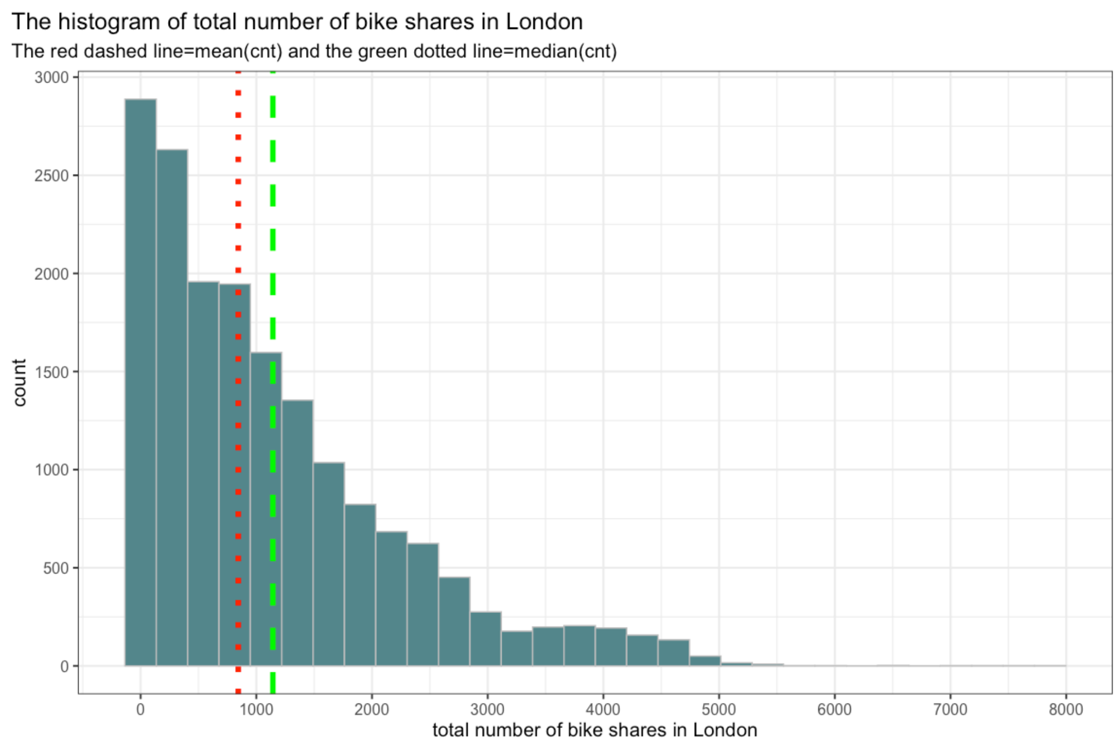
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
cnt	0	1	1143.10	1085.11	0.0	257	844.0	1671.75	7860.0	▇▂▁▁▁
t1	0	1	12.47	5.57	-1.5	8	12.5	16.00	34.0	▁▇▇▃▁
t2	0	1	11.52	6.62	-6.0	6	12.5	16.00	34.0	▁▃▇▃▁
hum	0	1	72.32	14.31	20.5	63	74.5	83.00	100.0	▁▂▅▇▅
wind_speed	0	1	15.91	7.89	0.0	10	15.0	20.50	56.5	▆▇▃▁▁

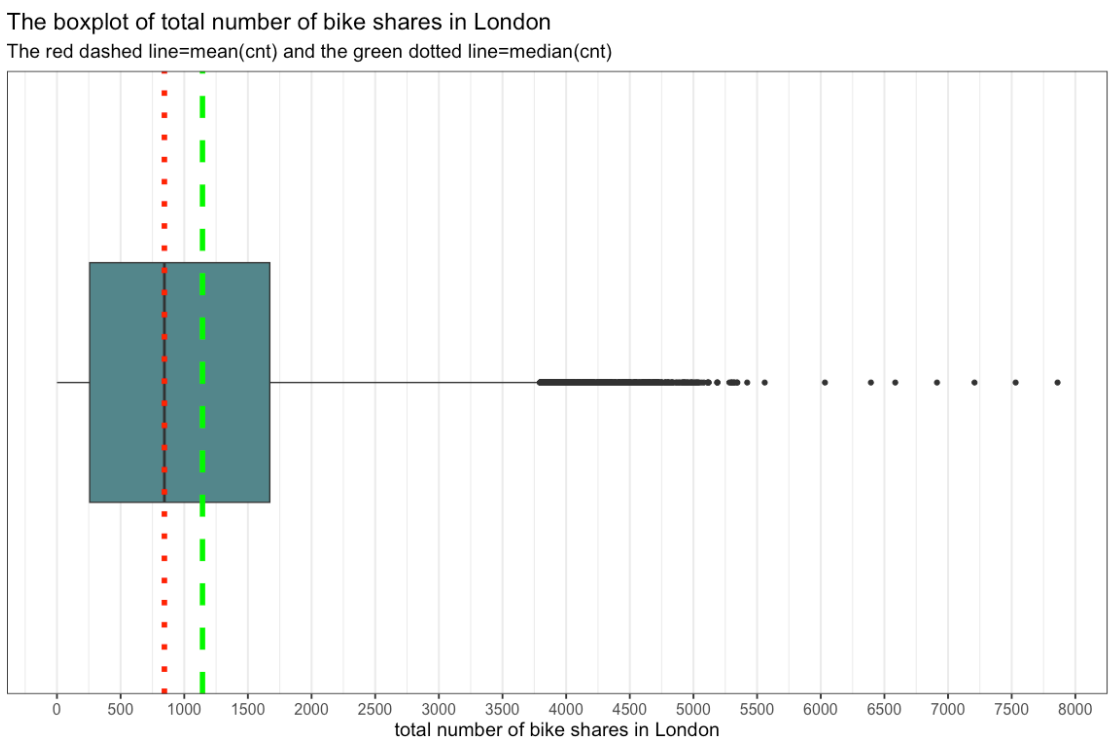
The above results show that the London bike sharing data is summarised based on the variable type. In the following subsection, the data are visualized with more details.

4-2 EDA

```
# histogram of cnt
ggplot(data=bike, aes(x=cnt))+
  geom_histogram(fill='cadetblue4', col='grey', bins = 30)+
  scale_x_continuous(n.breaks = 10)+
  scale_y_continuous(n.breaks = 10)+
  geom_vline(xintercept = mean(bike$cnt), colour='green', linetype='dashed', 
linewidth=2)+
  geom_vline(xintercept = median(bike$cnt), colour='red', linetype='dotted', 
linewidth=2)+
  labs(x='total number of bike shares in London',
       title = 'The histogram of total number of bike shares in London',
       subtitle = 'The red dashed line=mean(cnt) and the green dotted 
line=median(cnt)')
```

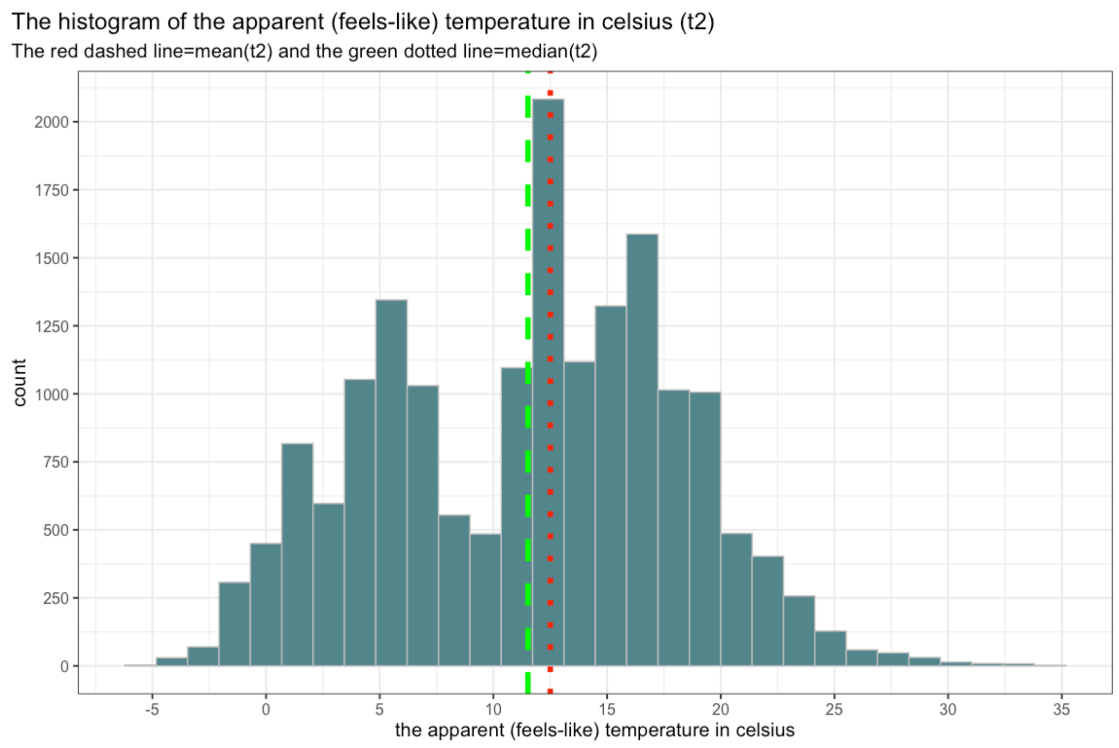


```r
# boxplot of cnt
ggplot(data=bike, aes(x=cnt))+
  geom_boxplot(fill='cadetblue4')+
  scale_x_continuous(n.breaks = 20)+
  scale_y_discrete()+
  geom_vline(xintercept = mean(bike$cnt), colour='green', linetype='dashed',
linewidth=2)+
  geom_vline(xintercept = median(bike$cnt), colour='red', linetype='dotted',
linewidth=2)+
  labs(x='total number of bike shares in London',
       title = 'The boxplot of total number of bike shares in London',
       subtitle = 'The red dashed line=mean(cnt) and the green dotted
line=median(cnt)')
```



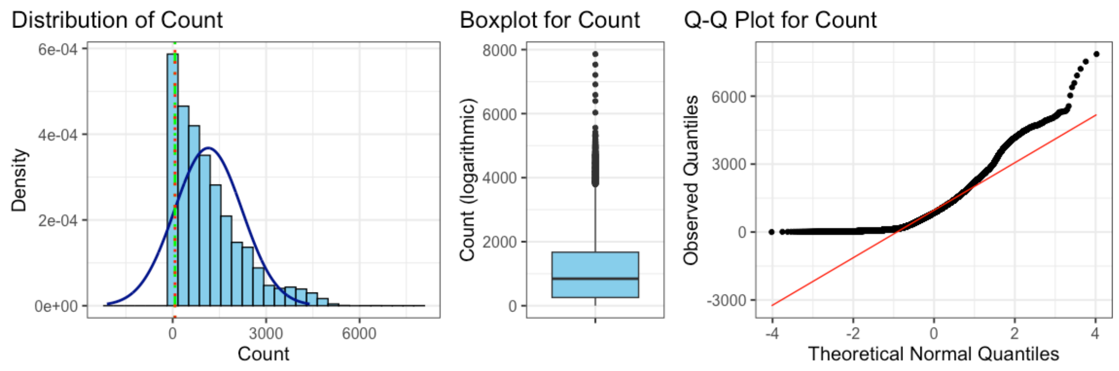
Histogram: - Variation in Bike Shares: The histogram and box plot show that the total number of bike shares varies from 0 to slightly above 5000. - Central Tendency: The average (represented by the red dashed line) and the median (represented by the green dotted line) of the variable are closer to the lower end of the range, indicating that the majority of data points have lower bike share counts. - Majority of Data: The observation mentions that most of the data falls between 0 and approximately 1800 bike shares, indicating that this range contains a significant portion of the dataset. - Outliers: The box plot identifies a few values above the typical range of the data, shown as black points. These values are considered outliers due to their unusually higher bike share counts.

```
# histogram of t2
ggplot(data=bike, aes(x=t2))+
  geom_histogram(fill='cadetblue4', col='grey', bins = 30)+
  scale_x_continuous(n.breaks = 10)+
  scale_y_continuous(n.breaks = 10)+
  geom_vline(xintercept = mean(bike$t2), colour='green', linetype='dashed', 
linewidth=2)+
  geom_vline(xintercept = median(bike$t2), colour='red', linetype='dotted', 
linewidth=2)+
  labs(x='the apparent (feels-like) temperature in celsius',
       title = 'The histogram of the apparent (feels-like) temperature in 
celsius (t2)',
       subtitle = 'The red dashed line=mean(t2) and the green dotted 
line=median(t2)')
```



Outlier Detection :

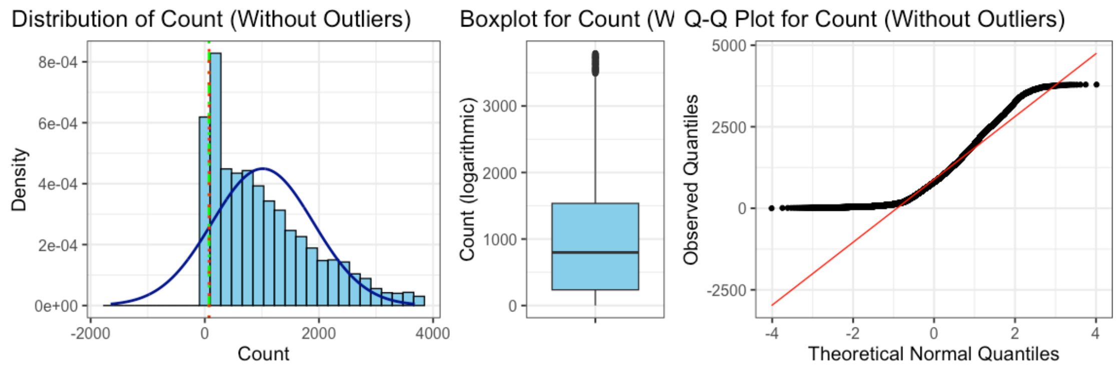
```r
# Calculate the mean and standard deviation of 'cnt'
mean_cnt <- mean(bike$cnt)
sd_cnt <- sd(bike$cnt)
# Create a range of values for the x-axis
x_values <- seq(mean_cnt - 3*sd_cnt, mean_cnt + 3*sd_cnt, length.out = 1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_cnt, sd = sd_cnt)
# Combine the 'x_values' and 'y_values' into a data frame
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot
boxplot <- ggplot(data = bike, aes(x = "", y = cnt)) +
  geom_boxplot(fill = "skyblue") +
#  scale_y_log10() +
  labs(x = "", y = "Count (logarithmic)") +
  ggtitle("Boxplot for Count")
# Create a Q-Q plot and add a diagonal line
qqplot <- ggplot(data = bike, aes(sample = cnt)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Count")
# Create a histogram and add the bell curve
densityplot <- ggplot(data = bike, aes(x = cnt)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30, colour = "black",
fill = "skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike$hum), colour='green', linetype='dotdash',
linewidth=1)+
  geom_vline(xintercept = median(bike$hum), colour='red', linetype='dotted',
linewidth=1)+
  labs(x = "Count", y = "Density") +
  ggtitle("Distribution of Count")
# Arrange the plots in one row using the 'grid.arrange' function from the
'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



- The data points generally align closely with the diagonal line, suggesting a relatively normal distribution. However, there are noticeable deviations from the line at the extreme ends, indicating the presence of outliers or departures from normality in those areas.

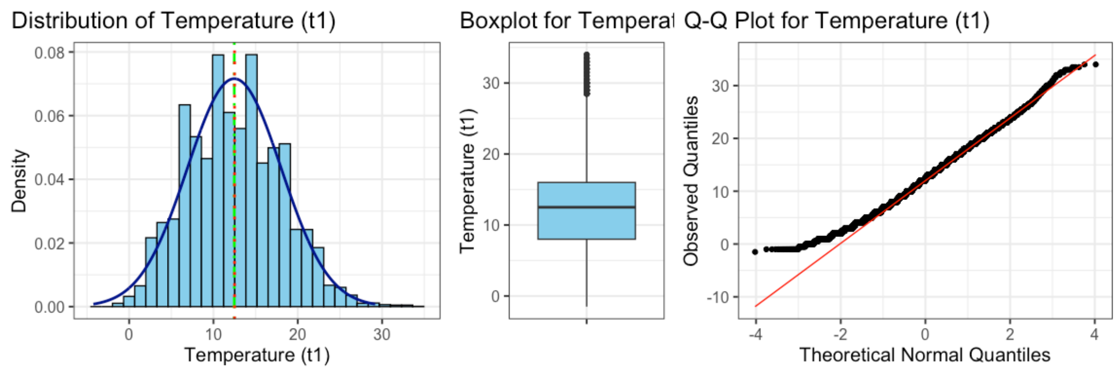
Removing outlire by Tukey's fences method

```
# Calculate the quartiles and interquartile range (IQR) of 'cnt'
q1 <- quantile(bike$cnt)[2]
q3 <- quantile(bike$cnt)[4]
iqr <- q3 - q1

# Calculate the lower and upper fences
lower_fence <- q1 - 1.5 * iqr
upper_fence <- q3 + 1.5 * iqr

# Identify outliers
outliers <- bike$cnt < lower_fence | bike$cnt > upper_fence

# Remove outliers (optional)
bike_no_outliers <- bike[!outliers, ]

# Generate updated plots without outliers
# Calculate the mean and standard deviation of 'cnt' (updated)
mean_cnt <- mean(bike_no_outliers$cnt)
sd_cnt <- sd(bike_no_outliers$cnt)
# Create a range of values for the x-axis
x_values <- seq(mean_cnt - 3 * sd_cnt, mean_cnt + 3 * sd_cnt, length.out =
1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_cnt, sd = sd_cnt)
# Combine the 'x_values' and 'y_values' into a data frame
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot without outliers
boxplot <- ggplot(data = bike_no_outliers, aes(x = "", y = cnt)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "", y = "Count (logarithmic)") +
  ggtitle("Boxplot for Count (Without Outliers)")
# Create a Q-Q plot without outliers
qqplot <- ggplot(data = bike_no_outliers, aes(sample = cnt)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Count (Without Outliers)")
# Create a histogram without outliers
densityplot <- ggplot(data = bike_no_outliers, aes(x = cnt)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30, colour = "black",
fill = "skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike_no_outliers$hum), colour = 'green',
linetype = 'dotdash', linewidth = 1) +
  geom_vline(xintercept = median(bike_no_outliers$hum), colour = 'red',
linetype = 'dotted', linewidth = 1) +
  labs(x = "Count", y = "Density") +
  ggtitle("Distribution of Count (Without Outliers)")

# Arrange the updated plots in one row using the 'grid.arrange' function from
the 'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



```
# Calculate the mean and standard deviation of 't1'
mean_t1 <- mean(bike$t1)
sd_t1 <- sd(bike$t1)
# Create a range of values for the x-axis
x_values <- seq(mean_t1 - 3*sd_t1, mean_t1 + 3*sd_t1, length.out = 1000)
```

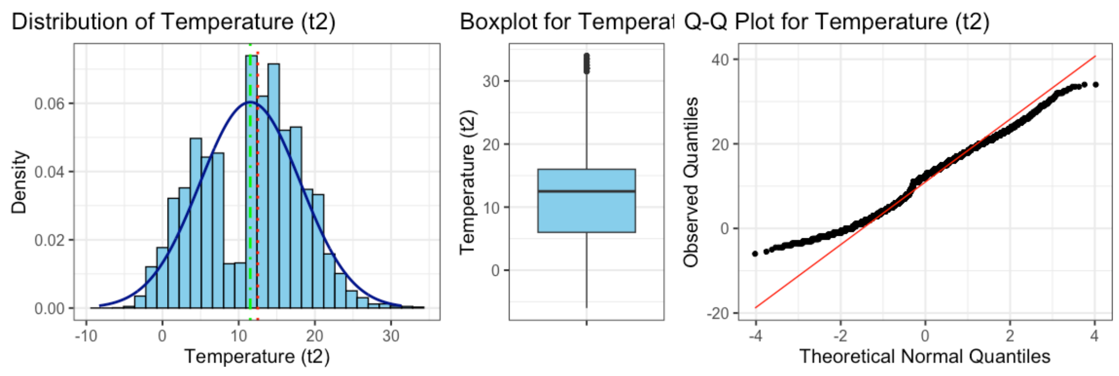
```
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_t1, sd = sd_t1)
# Combine the 'x_values' and 'y_values' into a data frame
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot
boxplot <- ggplot(data = bike, aes(x = "", y = t1)) +
  geom_boxplot(fill = "skyblue") +
#  scale_y_log10() +
  labs(x = "", y = "Temperature (t1)") +
  ggtitle("Boxplot for Temperature (t1)")
# Create a Q-Q plot and add a diagonal line
qqplot <- ggplot(data = bike, aes(sample = t1)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Temperature (t1)")
# Create a histogram and add the bell curve
densityplot <- ggplot(data = bike, aes(x = t1)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30, colour = "black",
fill = "skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike$t1), colour='green', linetype='dotdash',
linewidth=1)+
  geom_vline(xintercept = median(bike$t1), colour='red', linetype='dotted',
linewidth=1)+
  labs(x = "Temperature (t1)", y = "Density") +
  ggtitle("Distribution of Temperature (t1)")
# Arrange the plots in one row using the 'grid.arrange' function from the
'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



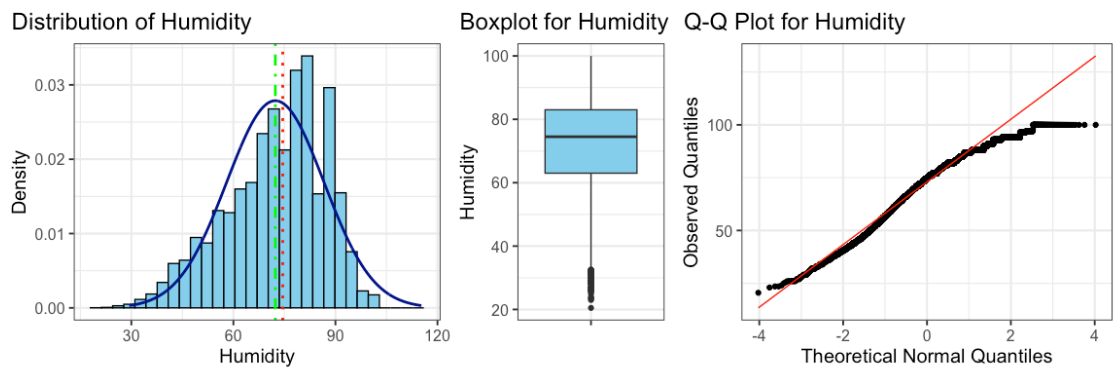
Histogram: - The temperature variable shows a concentration of observations between 10 to 25 degrees Celsius, indicating common temperatures in London. The count variable displays a right-skewed distribution, with higher occurrences of lower rental counts and a gradual decrease as the count increases.

QQ plot: - This QQ plot reveals a relatively linear relationship between the observed quantiles and the theoretical quantiles, indicating that the data is approximately normally distributed. There are minimal deviations from the diagonal line, suggesting that the temperature variable follows a normal distribution with few outliers or departures from normality. This implies that the distribution of temperature values is consistent with a normal distribution pattern.

```r
# Calculate the mean and standard deviation of 't2'
mean_t2 <- mean(bike$t2)
sd_t2 <- sd(bike$t2)
# Create a range of values for the x-axis
x_values <- seq(mean_t2 - 3*sd_t2, mean_t2 + 3*sd_t2, length.out = 1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_t2, sd = sd_t2)
# Combine the 'x_values' and 'y_values' into a data frame
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot
boxplot <- ggplot(data = bike, aes(x = "", y = t2)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "", y = "Temperature (t2)") +
  ggtitle("Boxplot for Temperature (t2)")
# Create a Q-Q plot and add a diagonal line
qqplot <- ggplot(data = bike, aes(sample = t2)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Temperature (t2)")
# Create a histogram and add the bell curve
densityplot <- ggplot(data = bike, aes(x = t2)) +
  geom_histogram(aes(y = after_stat(density)), bins = 30, colour = "black",
fill = "skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike$t2), colour='green', linetype='dotdash',
linewidth=1) +
  geom_vline(xintercept = median(bike$t2), colour='red', linetype='dotted',
linewidth=1) +
  labs(x = "Temperature (t2)", y = "Density") +
  ggtitle("Distribution of Temperature (t2)")

# Arrange the plots in one row using the 'grid.arrange' function from the
'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



```
# Calculate the mean and standard deviation of 'hum'
mean_hum <- mean(bike$hum)
sd_hum <- sd(bike$hum)
# Create a range of values for the x-axis
x_values <- seq(mean_hum - 3*sd_hum, mean_hum + 3*sd_hum, length.out = 1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_hum, sd = sd_hum)
# Combine the 'x_values' and 'y_values' into a data frame
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot
boxplot <- ggplot(data = bike, aes(x = "", y = hum)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "", y = "Humidity") +
  ggtitle("Boxplot for Humidity")
# Create a Q-Q plot and add a diagonal line
qqplot <- ggplot(data = bike, aes(sample = hum)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Humidity")
# Create a histogram and add the bell curve
densityplot <- ggplot(data = bike, aes(x = hum)) +
  geom_histogram(aes(y = ..density..), bins = 30, colour = "black", fill =
"skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike$hum), colour='green', linetype='dotdash',
linewidth=1) +
  geom_vline(xintercept = median(bike$hum), colour='red', linetype='dotted',
linewidth=1) +
  labs(x = "Humidity", y = "Density") +
  ggtitle("Distribution of Humidity")

# Arrange the plots in one row using the 'grid.arrange' function from the
'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```

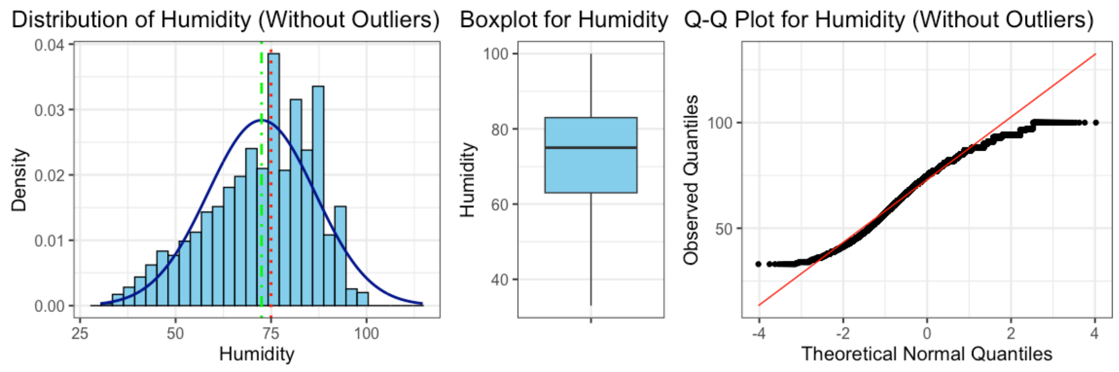


Histogram: - The temperature variable shows a concentration of observations between 10 to 25 degrees Celsius, indicating common temperatures in London. The count variable displays a right-skewed distribution, with higher occurrences of lower rental counts and a gradual decrease as the count increases.

QQ plot: - Distribution of humidity values deviates from a normal distribution. It means that the observed humidity values don't follow the expected pattern based on a normal distribution. Specifically, the plot reveals that the extreme values of humidity are more frequent than what would be expected in a normal distribution. This suggests that there might be outliers or certain factors affecting the humidity levels that cause them to deviate from a normal pattern.

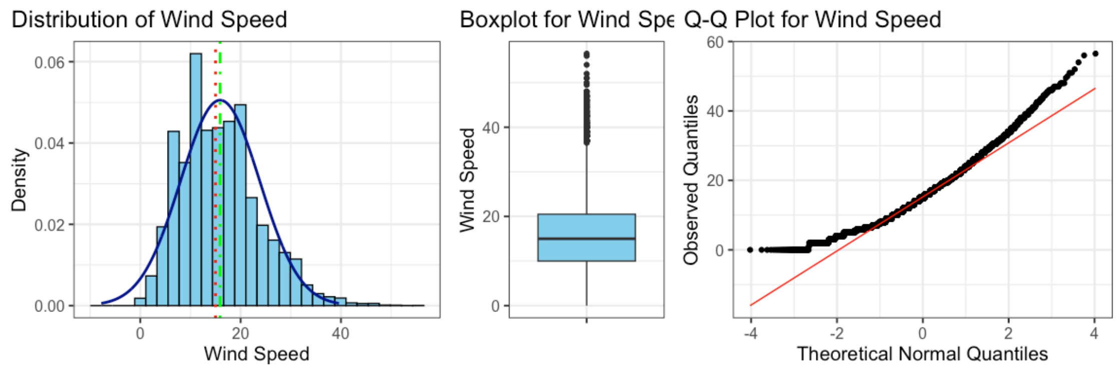
```r
# Calculate the quartiles and interquartile range (IQR) of 'hum'
q1 <- quantile(bike$hum)[2]
q3 <- quantile(bike$hum)[4]
iqr <- q3 - q1

# Calculate the lower and upper fences
lower_fence <- q1 - 1.5 * iqr
upper_fence <- q3 + 1.5 * iqr

# Identify outliers
outliers <- bike$hum < lower_fence | bike$hum > upper_fence

# Remove outliers (optional)
bike_no_outliers <- bike[!outliers, ]

# Calculate the mean and standard deviation of 'hum' (updated)
mean_hum <- mean(bike_no_outliers$hum)
sd_hum <- sd(bike_no_outliers$hum)
# Create a range of values for the x-axis
x_values <- seq(mean_hum - 3 * sd_hum, mean_hum + 3 * sd_hum, length.out =
1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_hum, sd = sd_hum)
# Combine the 'x_values' and 'y_values' into a data frame
```

```r
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot without outliers
boxplot <- ggplot(data = bike_no_outliers, aes(x = "", y = hum)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "", y = "Humidity") +
  ggtitle("Boxplot for Humidity (Without Outliers)")
# Create a Q-Q plot without outliers
qqplot <- ggplot(data = bike_no_outliers, aes(sample = hum)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Humidity (Without Outliers)")
# Create a histogram without outliers
densityplot <- ggplot(data = bike_no_outliers, aes(x = hum)) +
  geom_histogram(aes(y = ..density..), bins = 30, colour = "black", fill =
"skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike_no_outliers$hum), colour='green',
linetype='dotdash', linewidth=1) +
  geom_vline(xintercept = median(bike_no_outliers$hum), colour='red',
linetype='dotted', linewidth=1) +
  labs(x = "Humidity", y = "Density") +
  ggtitle("Distribution of Humidity (Without Outliers)")

# Arrange the updated plots in one row using the 'grid.arrange' function from
the 'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



```r
# Calculate the mean and standard deviation of 'wind_speed'
mean_wind_speed <- mean(bike$wind_speed)
sd_wind_speed <- sd(bike$wind_speed)
# Create a range of values for the x-axis
x_values <- seq(mean_wind_speed - 3*sd_wind_speed, mean_wind_speed +
3*sd_wind_speed, length.out = 1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_wind_speed, sd = sd_wind_speed)
# Combine the 'x_values' and 'y_values' into a data frame
```

```
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot
boxplot <- ggplot(data = bike, aes(x = "", y = wind_speed)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "", y = "Wind Speed") +
  ggtitle("Boxplot for Wind Speed")
# Create a Q-Q plot and add a diagonal line
qqplot <- ggplot(data = bike, aes(sample = wind_speed)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Wind Speed")
# Create a histogram and add the bell curve
densityplot <- ggplot(data = bike, aes(x = wind_speed)) +
  geom_histogram(aes(y = ..density..), bins = 30, colour = "black", fill =
"skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike$wind_speed), colour='green',
linetype='dotdash', linewidth=1) +
  geom_vline(xintercept = median(bike$wind_speed), colour='red',
linetype='dotted', linewidth=1) +
  labs(x = "Wind Speed", y = "Density") +
  ggtitle("Distribution of Wind Speed")

# Arrange the plots in one row using the 'grid.arrange' function from the
'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



Histogram: - It shows that the distribution is right-skewed, with the majority of observations concentrated at lower wind speeds. This suggests that lower wind speeds are more common in London. As the wind speed increases, the frequency of observations gradually decreases, indicating that higher wind speeds are less frequent. The histogram provides an overview of the distribution and highlights the most common wind speed ranges in the dataset.

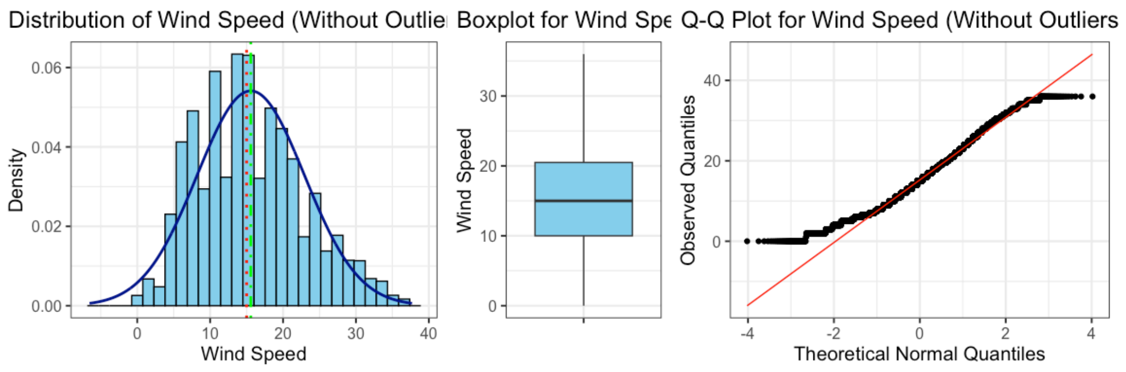
Overall, The numeric features (independent variables) seem to be more normally distributed compared to the response variable because the mean and median of the variables are nearer the middle of the range of values except humidity where mean and median is towards right side, coinciding with where the most commonly occurring values are. Now that the distribution of the numerical variables are investigated, it is time to perform a correlation analysis to investigate the relationship between variables.

QQ plot: - The distribution of wind speed values deviates from a normal distribution. The plot shows a noticeable curvature, indicating a departure from linearity. This suggests that the wind speed values are not normally distributed and may be influenced by certain factors or conditions. It is important to consider this non-normality when analyzing the wind speed variable and take appropriate statistical measures accordingly.

Removing outlire by Tukey's fences method

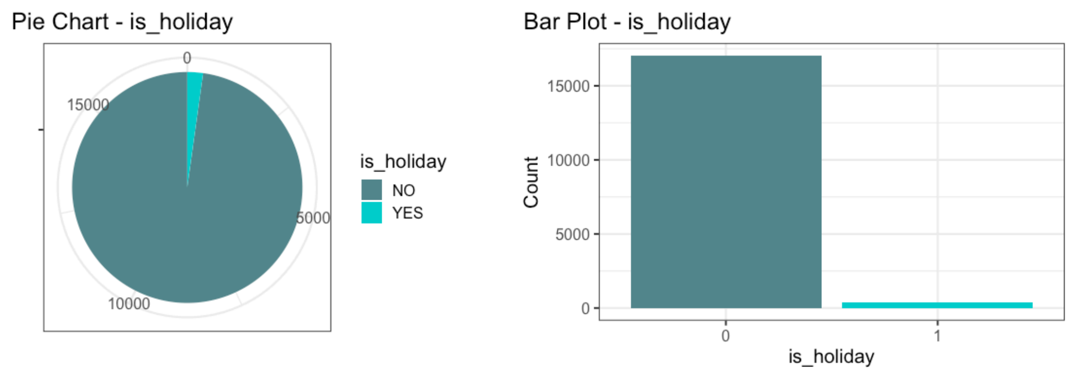
```r
# Calculate the quartiles and interquartile range (IQR) of 'wind_speed'
q1 <- quantile(bike$wind_speed)[2]
q3 <- quantile(bike$wind_speed)[4]
iqr <- q3 - q1

# Calculate the lower and upper fences
lower_fence <- q1 - 1.5 * iqr
upper_fence <- q3 + 1.5 * iqr

# Identify outliers
outliers <- bike$wind_speed < lower_fence | bike$wind_speed > upper_fence

# Remove outliers (optional)
bike_no_outliers <- bike[!outliers, ]

# Generate updated plots without outliers
# Calculate the mean and standard deviation of 'wind_speed' (updated)
mean_wind_speed <- mean(bike_no_outliers$wind_speed)
sd_wind_speed <- sd(bike_no_outliers$wind_speed)
# Create a range of values for the x-axis
x_values <- seq(mean_wind_speed - 3 * sd_wind_speed, mean_wind_speed + 3 *
sd_wind_speed, length.out = 1000)
# Create a bell curve with mean and standard deviation calculated above
y_values <- dnorm(x_values, mean = mean_wind_speed, sd = sd_wind_speed)
# Combine the 'x_values' and 'y_values' into a data frame
density_df <- data.frame(x = x_values, y = y_values)
# Create a boxplot without outliers
boxplot <- ggplot(data = bike_no_outliers, aes(x = "", y = wind_speed)) +
  geom_boxplot(fill = "skyblue") +
  labs(x = "", y = "Wind Speed") +
  ggtitle("Boxplot for Wind Speed (Without Outliers)")
# Create a Q-Q plot without outliers
qqplot <- ggplot(data = bike_no_outliers, aes(sample = wind_speed)) +
  stat_qq() +
  stat_qq_line(colour = "red") +
  labs(x = "Theoretical Normal Quantiles", y = "Observed Quantiles") +
  ggtitle("Q-Q Plot for Wind Speed (Without Outliers)")
# Create a histogram without outliers
```

```
densityplot <- ggplot(data = bike_no_outliers, aes(x = wind_speed)) +
  geom_histogram(aes(y = ..density..), bins = 30, colour = "black", fill =
"skyblue") +
  geom_line(data = density_df, aes(x = x, y = y), colour = "darkblue",
linewidth = 1) +
  geom_vline(xintercept = mean(bike_no_outliers$wind_speed), colour =
'green', linetype = 'dotdash', linewidth = 1) +
  geom_vline(xintercept = median(bike_no_outliers$wind_speed), colour =
'red', linetype = 'dotted', linewidth = 1) +
  labs(x = "Wind Speed", y = "Density") +
  ggtitle("Distribution of Wind Speed (Without Outliers)")

# Arrange the updated plots in one row using the 'grid.arrange' function from
the 'gridExtra' package
grid.arrange(densityplot, boxplot, qqplot, ncol = 3, widths = c(2, 1, 2))
```



Destribution of Categorical Variables

```
# Pie chart for is_holiday
pie_chart <- ggplot(data = bike, aes(x = "", fill = is_holiday)) +
  geom_bar(width = 1) +
  coord_polar(theta = "y") +
  xlab("") +
  ylab("") +
  guides(fill = guide_legend(title = "is_holiday")) +
  scale_fill_manual(values = c("cadetblue4", "cyan3"), labels = c("NO",
"YES")) +
  labs(title = "Pie Chart - is_holiday")

# Bar plot for is_holiday
bar_plot <- ggplot(data = bike, aes(x = is_holiday)) +
  geom_bar(fill = c("cadetblue4", "cyan3")) +
  xlab("is_holiday") +
  ylab("Count") +
  labs(title = "Bar Plot - is_holiday")

# Combine the pie chart and bar plot using grid.arrange
combined_plot <- grid.arrange(pie_chart, bar_plot, ncol = 2)
```

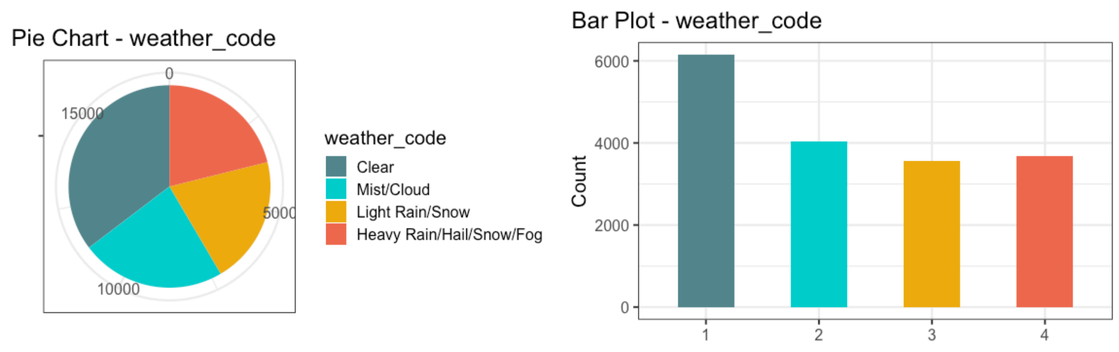


```
# Display the combined plot
combined_plot

## TableGrob (1 x 2) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
```

- The chart shows that the majority of observations (around 84%) are labeled as "Not a holiday," while the remaining observations (approximately 16%) are labeled as "Holiday." This indicates that the dataset consists mostly of non-holiday days, with a smaller proportion of holiday days.

```
# Define the color palette
color_palette <- c("cadetblue4", "cyan3", "darkgoldenrod2", "coral2")

# Pie chart for weather_code
pie_chart <- ggplot(data = bike, aes(x = "", fill = weather_code)) +
  geom_bar(width = 1) +
  coord_polar(theta = "y") +
  xlab("") +
  ylab("") +
  guides(fill = guide_legend(title = "weather_code")) +
  scale_fill_manual(values = color_palette,
                    labels = c('Clear','Mist/Cloud','Light Rain/Snow','Heavy
Rain/Hail/Snow/Fog')) +
  labs(title = "Pie Chart - weather_code")

# Bar plot for weather_code
bar_plot <- ggplot(data = bike, aes(x = weather_code, fill = weather_code)) +
  geom_bar(width = 0.5, show.legend = FALSE) +
  xlab("") +
  ylab("Count") +
  labs(title = "Bar Plot - weather_code") +
  scale_fill_manual(values = color_palette) +
  theme(plot.background = element_blank())

# Combine the plots
```

```
combined_plot <- grid.arrange(pie_chart, bar_plot, ncol = 2, widths = c(1,
1))
```

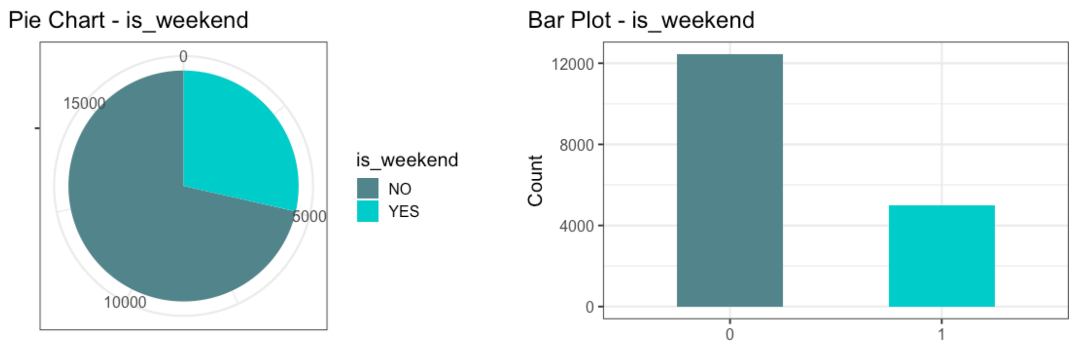


```
# Display the combined plot
print(combined_plot)

## TableGrob (1 x 2) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
```

- The largest portion of the pie is attributed to weather code 1, which represents “Clear” weather conditions. The other segments of the pie represent different weather codes corresponding to various weather conditions such as “Mist”, “Light rain”, “Cloudy”, and others. This indicates that the majority of observations in the dataset experienced clear weather conditions, while other weather conditions occurred less frequently.

```
# Define the color palette
color_palette <- c("cadetblue4", "cyan3")

# Pie chart for is_weekend
pie_chart <- ggplot(data = bike, aes(x = "", fill = is_weekend)) +
  geom_bar(width = 1) +
  coord_polar(theta = "y") +
  xlab("") +
  ylab("") +
  guides(fill = guide_legend(title = "is_weekend")) +
  scale_fill_manual(values = color_palette, labels = c("NO", "YES")) +
  labs(title = "Pie Chart - is_weekend")

# Bar plot for is_weekend
bar_plot <- ggplot(data = bike, aes(x = is_weekend, fill = is_weekend)) +
  geom_bar(width = 0.5, show.legend = FALSE) +
  xlab("") +
  ylab("Count") +
  labs(title = "Bar Plot - is_weekend") +
  scale_fill_manual(values = color_palette) +
  theme(plot.background = element_blank())
```

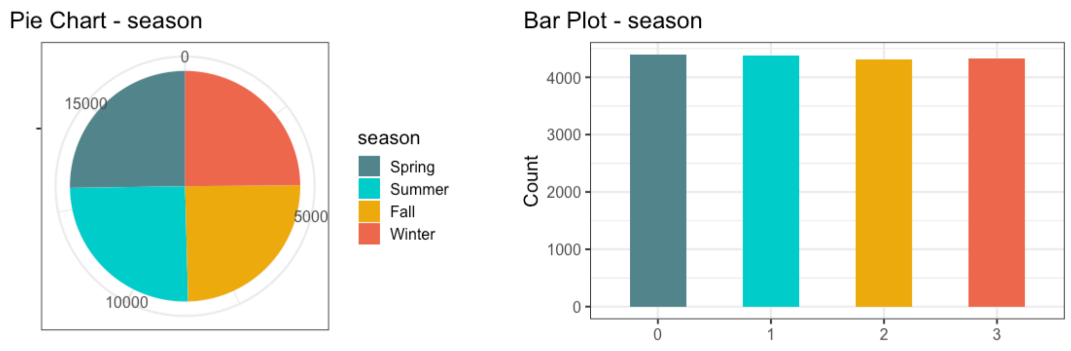
```
# Combine the plots
combined_plot <- grid.arrange(pie_chart, bar_plot, ncol = 2, widths = c(1,
1))
```



```
# Display the combined plot
print(combined_plot)

## TableGrob (1 x 2) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
```

- The larger section of the pie corresponds to “Not Weekend”, indicating that a majority of the observations in the dataset are from non-weekend days. The smaller section represents “Weekend” observations, indicating that they are less frequent in the dataset. This suggests that the dataset contains a larger proportion of non-weekend data points compared to weekend data points.

```
# Define the color palette
color_palette <- c("cadetblue4", "cyan3", "darkgoldenrod2", "coral2")

# Pie chart for season
pie_chart <- ggplot(data = bike, aes(x = "", fill = season)) +
  geom_bar(width = 1) +
  coord_polar(theta = "y") +
  xlab("") +
  ylab("") +
  guides(fill = guide_legend(title = "season")) +
  scale_fill_manual(values = color_palette,
labels=c('Spring','Summer','Fall','Winter')) +
  labs(title = "Pie Chart - season")

# Bar plot for season
bar_plot <- ggplot(data = bike, aes(x = season, fill = season)) +
  geom_bar(width = 0.5, show.legend = FALSE) +
  xlab("") +
  ylab("Count") +
  labs(title = "Bar Plot - season") +
  scale_fill_manual(values = color_palette) +
  theme(plot.background = element_blank())

# Combine the plots
combined_plot <- grid.arrange(pie_chart, bar_plot, ncol = 2, widths = c(1,
1))
```



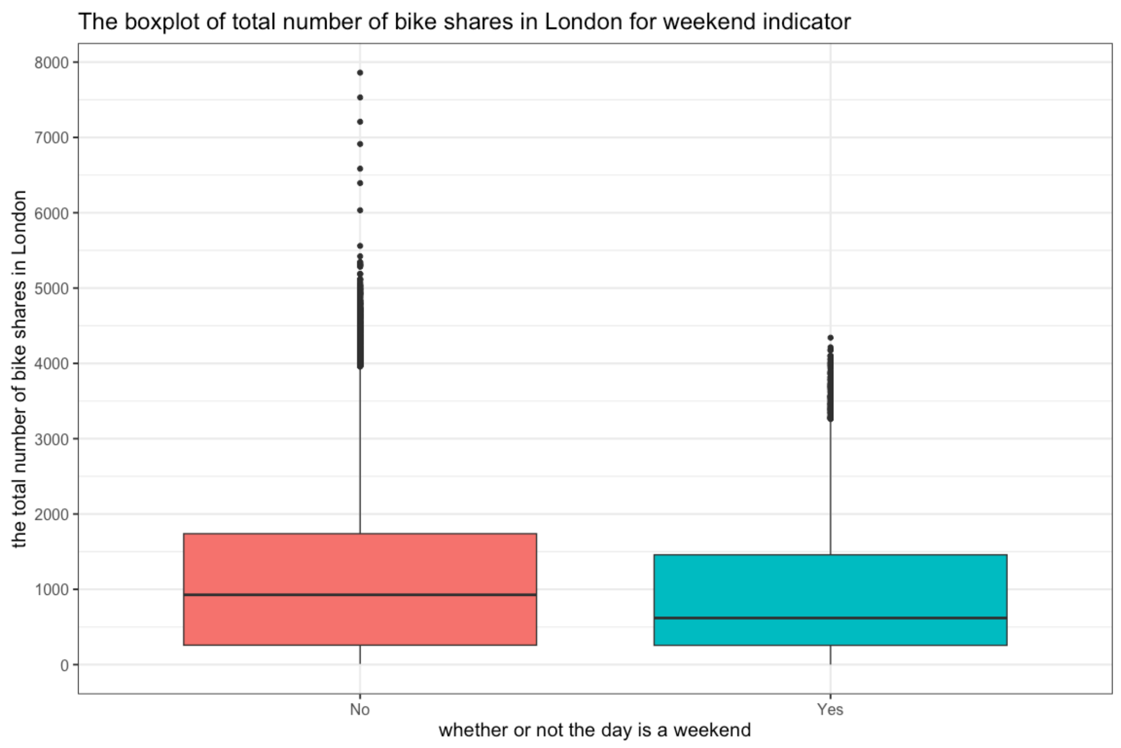
```
# Display the combined plot
print(combined_plot)

## TableGrob (1 x 2) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (1-1,2-2) arrange gtable[layout]
```

- The majority of observations in the dataset belong to the “Spring” season, as indicated by the largest section of the pie. The “Summer” and “Autumn” seasons have similar proportions, while the “Winter” season has the smallest proportion of observations. This suggests that the dataset is biased towards the “Spring” season, with relatively fewer observations from the other seasons.

Box Plots

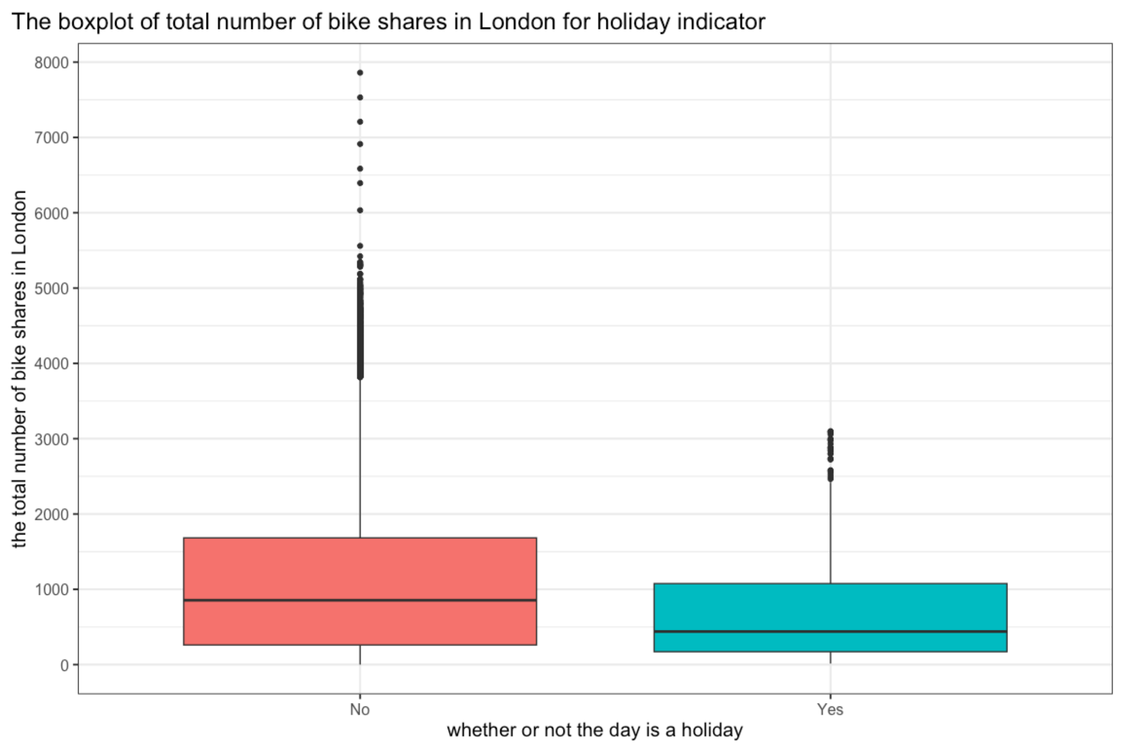
Now we use box-plots in order to compare each categorical independent variable with the response variable (the number of bike shares in London - cnt):

```
# boxplot of cnt and is_weekend
ggplot(data=bike, aes(x=is_weekend, y=cnt, fill=is_weekend))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  scale_x_discrete(breaks=c(0,1), labels=c('No','Yes'))+
  theme_bw(base_size = 15)+
  labs(x='whether or not the day is a weekend',y="the total number of bike
shares in London",
       title = 'The boxplot of total number of bike shares in London for
weekend indicator')
```



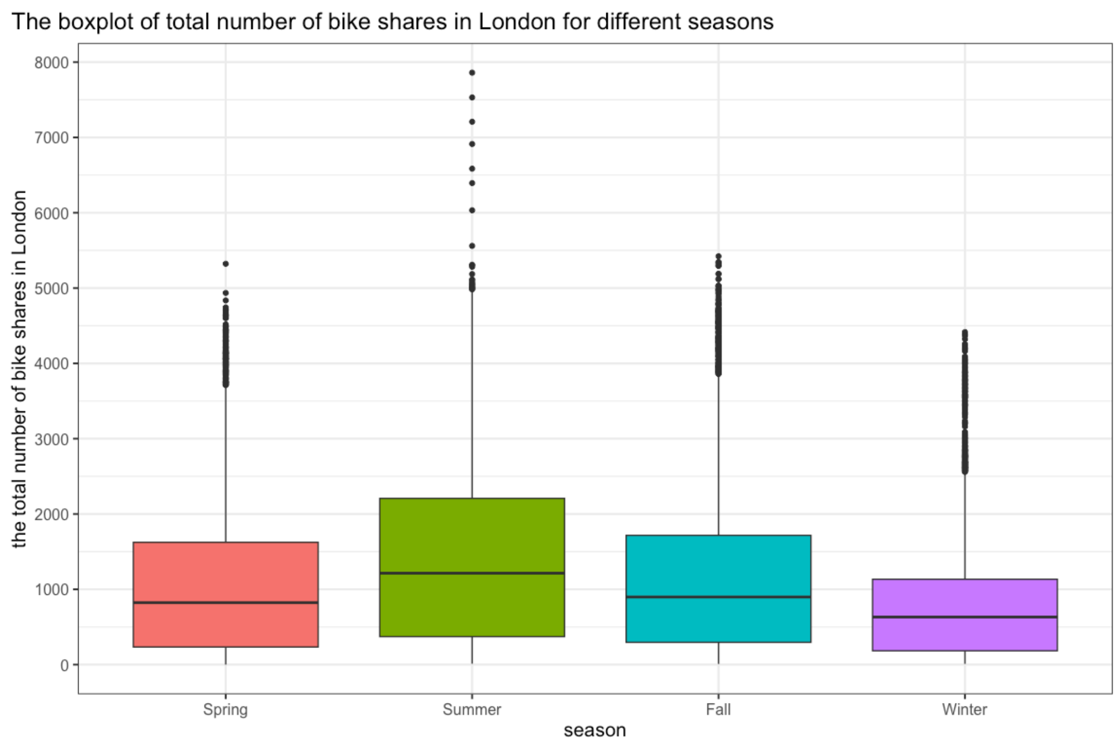
The median for weekends is lower compared to non-weekends, indicating that the median number of bike rentals is generally lower on weekends. The box for weekends is smaller than the box for non-weekends, suggesting less variability in bike rentals on weekends.

```
# boxplot of cnt and is_holiday
ggplot(data=bike, aes(x=is_holiday, y=cnt, fill=is_holiday))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  scale_x_discrete(breaks=c(0,1), labels=c('No','Yes'))+
  labs(x='whether or not the day is a holiday',y="the total number of bike
shares in London",
       title = 'The boxplot of total number of bike shares in London for
holiday indicator')
```



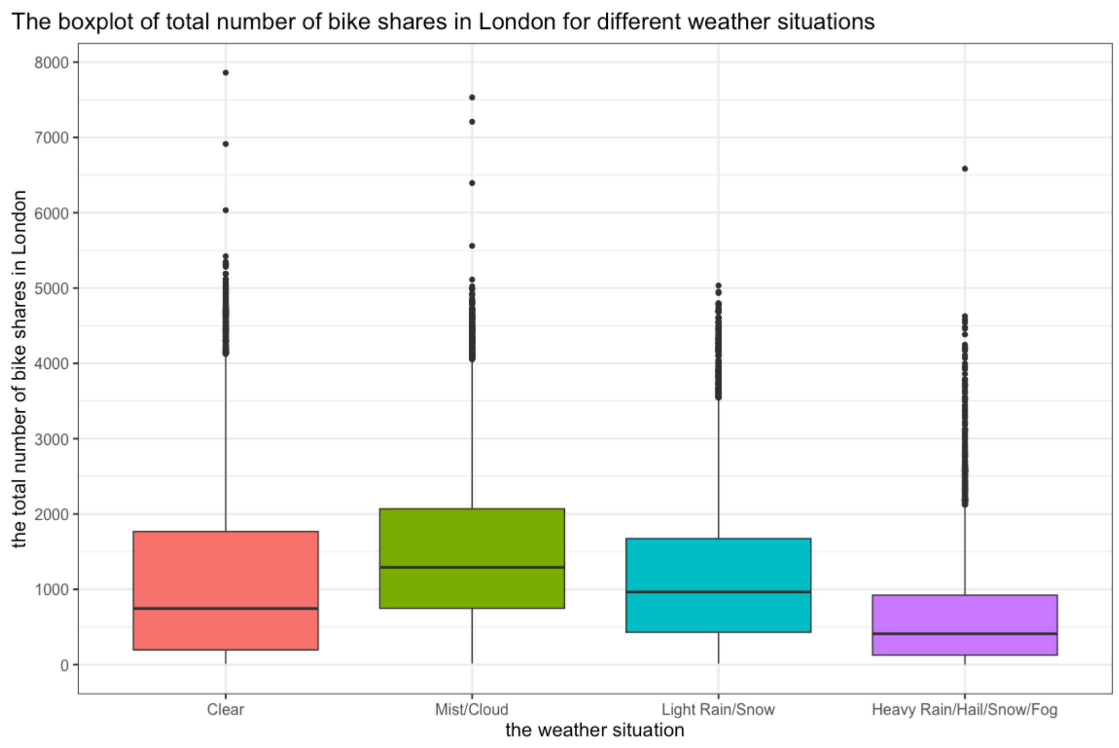
- The median for weekends is lower compared to non-weekends, indicating that the median number of bike rentals is generally lower on weekends.
- The box for weekends is smaller than the box for non-weekends, suggesting less variability in bike rentals on weekends.

```
# boxplot of cnt and season
ggplot(data=bike, aes(x=season, y=cnt, fill=season))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  scale_x_discrete(breaks=c(0,1,2,3),
labels=c('Spring','Summer','Fall','Winter'))+
  labs(x='season', y="the total number of bike shares in London",
       title = 'The boxplot of total number of bike shares in London for
different seasons')
```



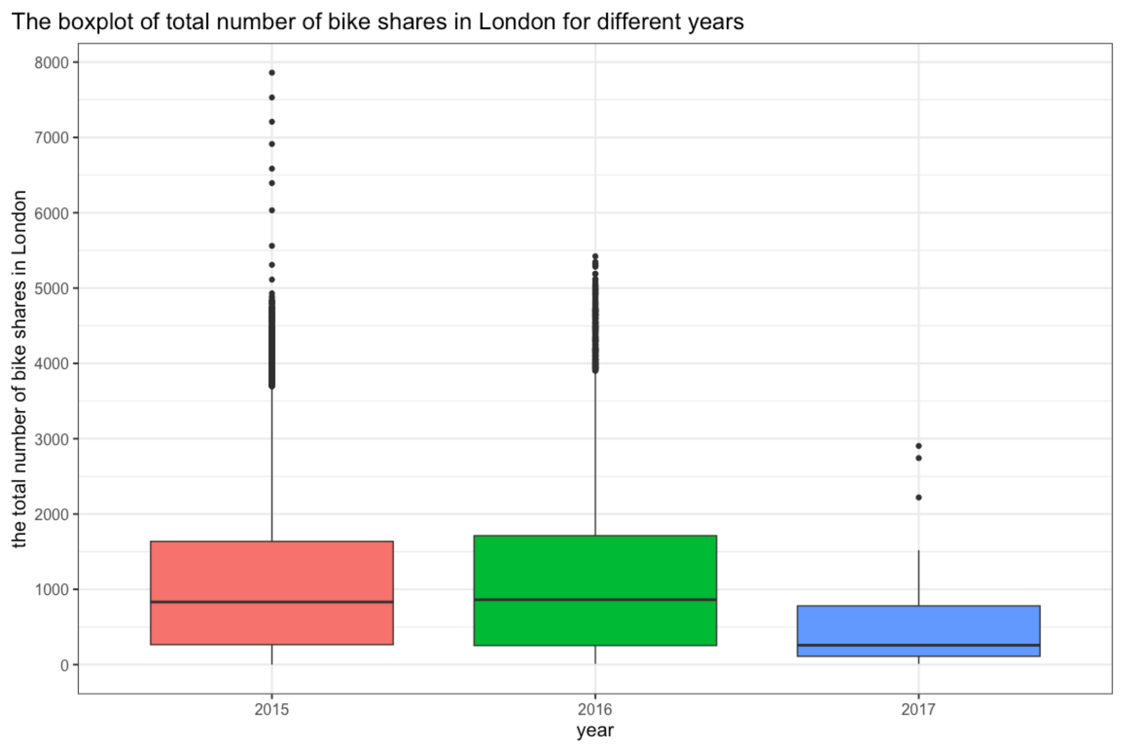
- Weather situation 2 (Mist or fog) has the highest median among all the weather codes, indicating that this weather condition is associated with higher bike rental counts on average.
- Weather situation 1 (Clear or few clouds) and weather situation 3 (Light snow or rain) have similar medians, suggesting that these weather conditions are associated with relatively lower bike rental counts compared to weather situation 2.
- Weather situation 4 (Heavy rain, ice pellets, or snow) has the lowest median, indicating that this weather condition is associated with the lowest bike rental counts on average.

```
# boxplot of cnt and weather_code
ggplot(data=bike, aes(x=weather_code, y=cnt, fill=weather_code))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  scale_x_discrete(breaks=c(1,2,3,4), labels=c('Clear','Mist/Cloud','Light
Rain/Snow','Heavy Rain/Hail/Snow/Fog'))+
  labs(x='the weather situation', y="the total number of bike shares in
London",
       title = 'The boxplot of total number of bike shares in London for
different weather situations')
```

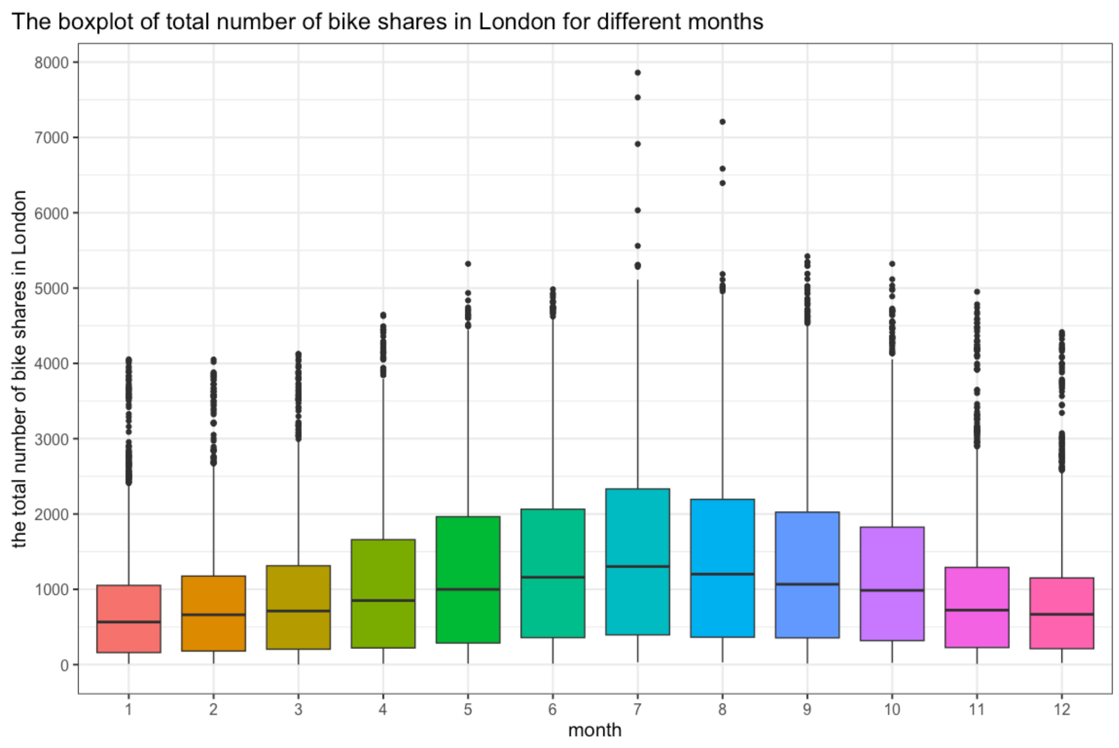


- The “spring” and “summer” seasons have similar medians and IQRs, indicating comparable bike rental patterns.
- The “autumn” season has a slightly higher median and a larger IQR, suggesting a wider range of bike rentals during this season.
- The “winter” season has the lowest median and the smallest IQR, indicating relatively lower bike rentals and a narrower range of values.

```
# boxplot of cnt and year
ggplot(data=bike, aes(x=year, y=cnt, fill=year))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  labs(x='year', y="the total number of bike shares in London",
       title = 'The boxplot of total number of bike shares in London for
different years')
```

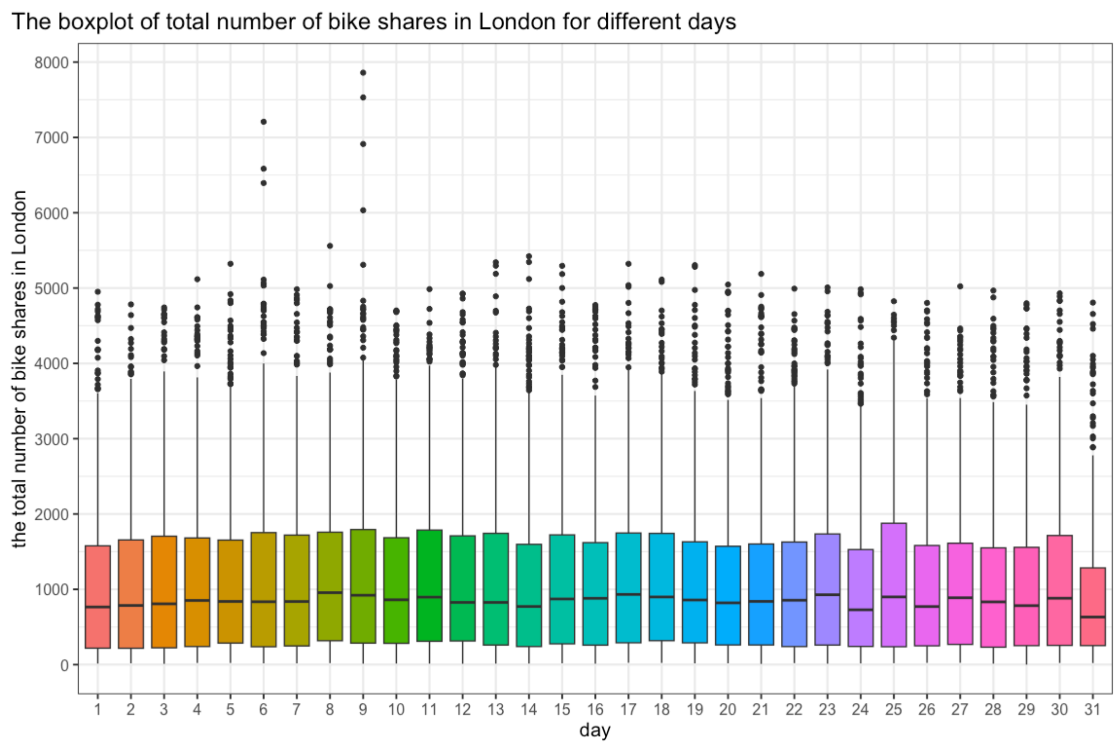


```r
# boxplot of cnt and month
ggplot(data=bike, aes(x=month, y=cnt, fill=month))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  labs(x='month', y="the total number of bike shares in London",
       title = 'The boxplot of total number of bike shares in London for
different months')
```

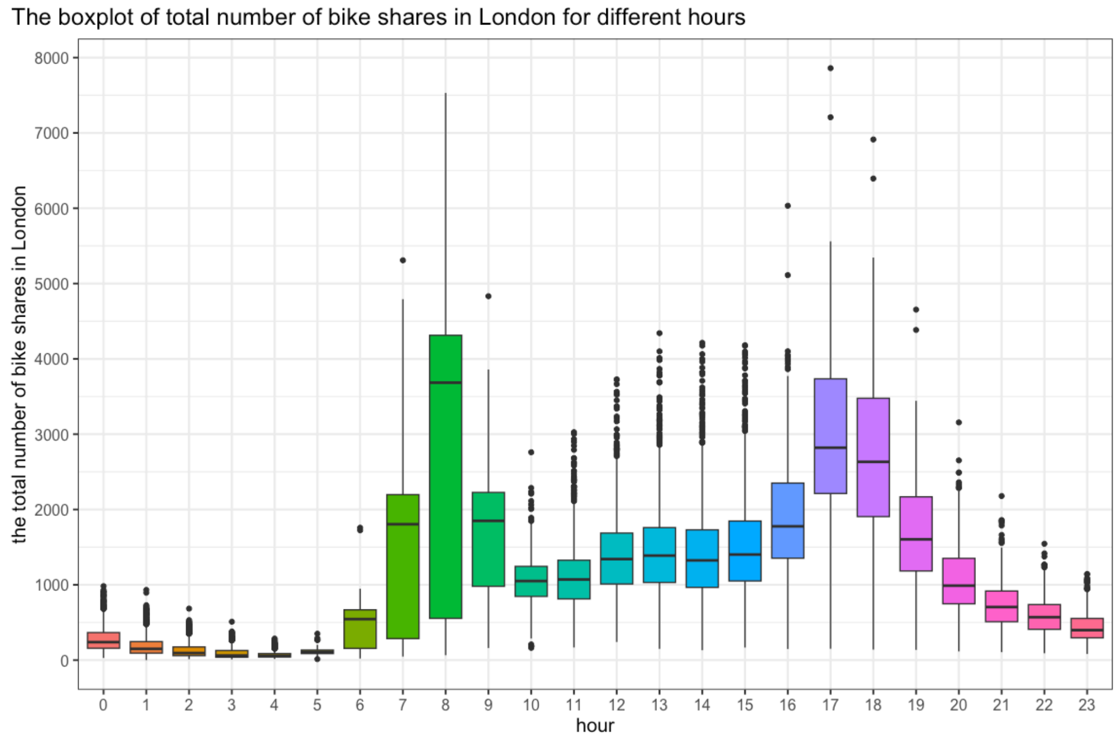


- The median bike share count appears to be relatively consistent across the months, with slight variations.
- The range of bike share counts is wider in some months compared to others, as indicated by the length of the whiskers.
- There are a few outliers in certain months, suggesting unusual instances of exceptionally high or low bike share counts.

```
# boxplot of cnt and day
ggplot(data=bike, aes(x=day, y=cnt, fill=day))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  labs(x='day', y="the total number of bike shares in London",
       title = 'The boxplot of total number of bike shares in London for
different days')
```

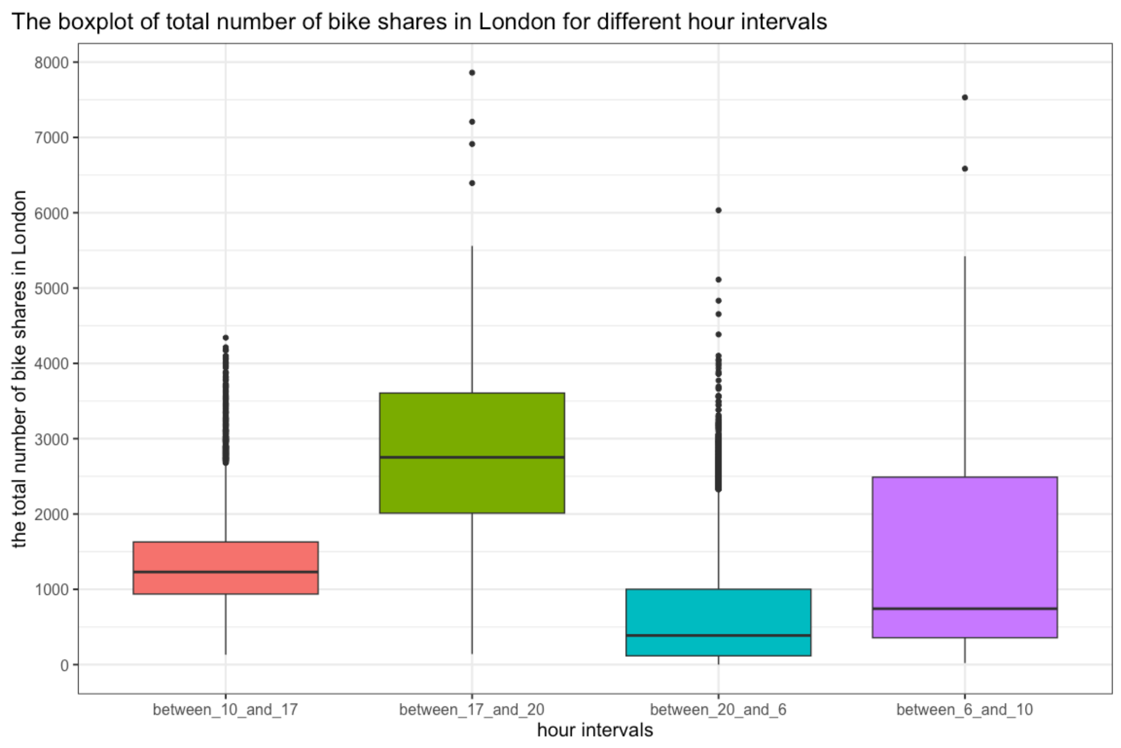


```
# boxplot of cnt and hour
ggplot(data=bike, aes(x=hour, y=cnt, fill=hour))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  labs(x='hour', y="the total number of bike shares in London",
       title = 'The boxplot of total number of bike shares in London for
different hours')
```



- The median bike share count tends to be higher during the morning and evening rush hours, specifically between 7 to 9 and 16 to 19. This suggests that there is a greater demand for bike rentals during these peak commuting hours.
- The counts during the late night and early morning hours (between 24 and 5) are relatively lower, indicating reduced bike rental activity during these periods.
- The range of counts varies across different hours, with wider ranges observed during the peak hours and narrower ranges during the off-peak hours.

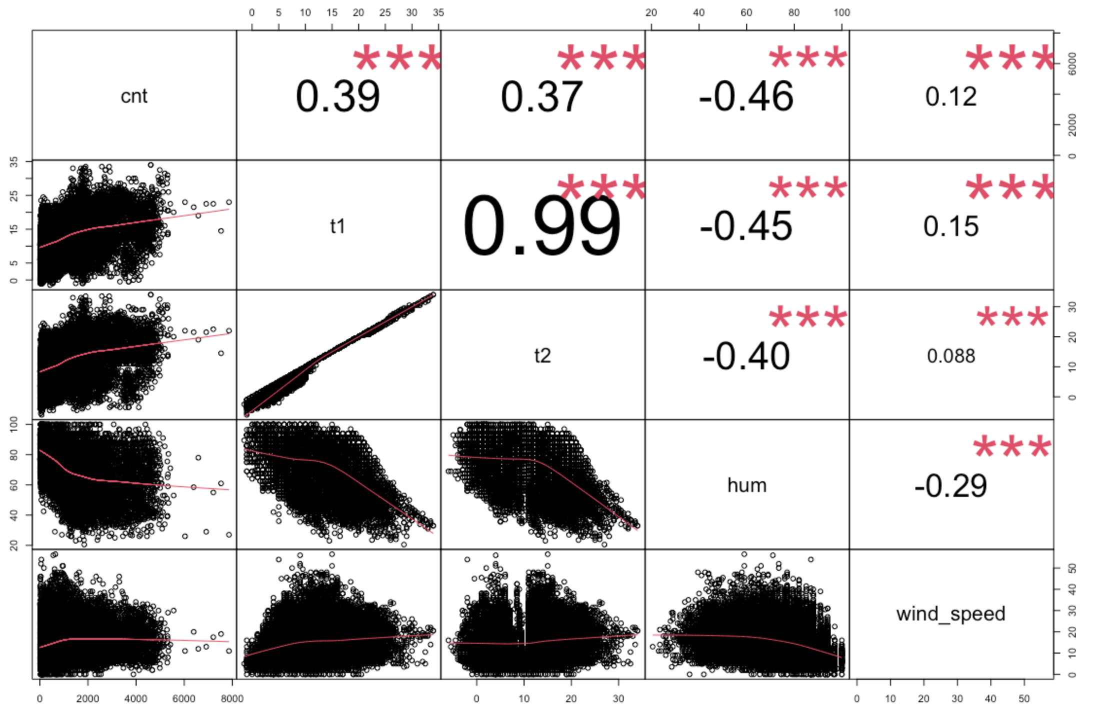
```
# create another categorical variable to discretion the hours
h <- bike$hour %>% as.numeric()
hour_cat <- ifelse(h > 6 & h < 10, 'between_6_and_10',
                   ifelse(h > 10 & h < 17, 'between_10_and_17',
                          ifelse(h > 17 & h < 20,'between_17_and_20',
                                 'between_20_and_6')))
# add the hour_cat to the data and replace it by hour
bike <- bike %>% mutate(hour=as.factor(hour_cat))
# boxplot of cnt and hour categories
ggplot(data=bike, aes(x=hour, y=cnt, fill=hour))+
  geom_boxplot(show.legend = F)+
  scale_y_continuous(n.breaks = 10)+
  labs(x='hour intervals', y="the total number of bike shares in London",
       title = 'The boxplot of total number of bike shares in London for
different hour intervals')
```



- The “between_17_and_20” hour interval exhibits a higher number of bike share counts, and the median line is positioned in the middle of the box, indicating a relatively balanced distribution of bike rentals during this time period.
- The “between_6_and_10” hour interval exhibits a wide range of bike share counts, but the median value suggests that the majority of rentals during this time period are relatively low.

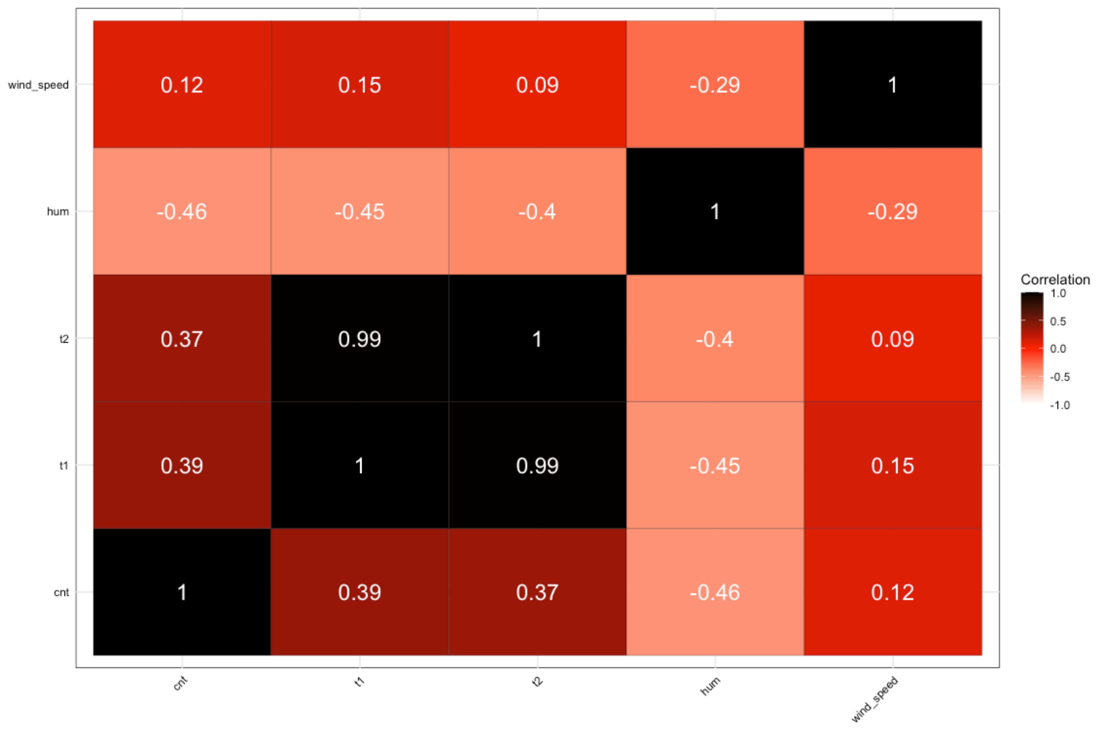
Corrolation Matrix

```
numerical_vars <- bike %>% select(cnt:wind_speed) %>%
mutate(cnt=as.numeric(cnt))
chart.Correlation(numerical_vars, histogram = F, method = "pearson")
```



The numeric features (independent variables) seem to be more normally distributed compared to the response variable because the mean and median of the variables are nearer the middle of the range of values except humidity where the mean and median are towards the right side, coinciding with where the most commonly occurring values are. Now that the distribution of the numerical variables is investigated, it is time to perform a correlation analysis to investigate the relationship between variables.

```
# Calculate correlation matrix
cor_matrix <- cor(numerical_vars)


# Convert correlation matrix to long format
cor_data <- reshape2::melt(cor_matrix)

# Plot heatmap with correlation values using dark mode colors
ggplot(cor_data, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile(color = "black") +
  scale_fill_gradient2(low = "white", mid = "red", high = "black",
                       midpoint = 0, limits = c(-1, 1), name = "Correlation")
+
  geom_text(aes(label = round(value, 2)), size = 6, color = "white") +
  labs(x = "", y = "") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        panel.background = element_rect(fill = "white"),
        plot.title = element_text(color = "green"),
        axis.text = element_text(color = "black"),
        legend.title = element_text(color = "black"),
        legend.text = element_text(color = "black"))
```

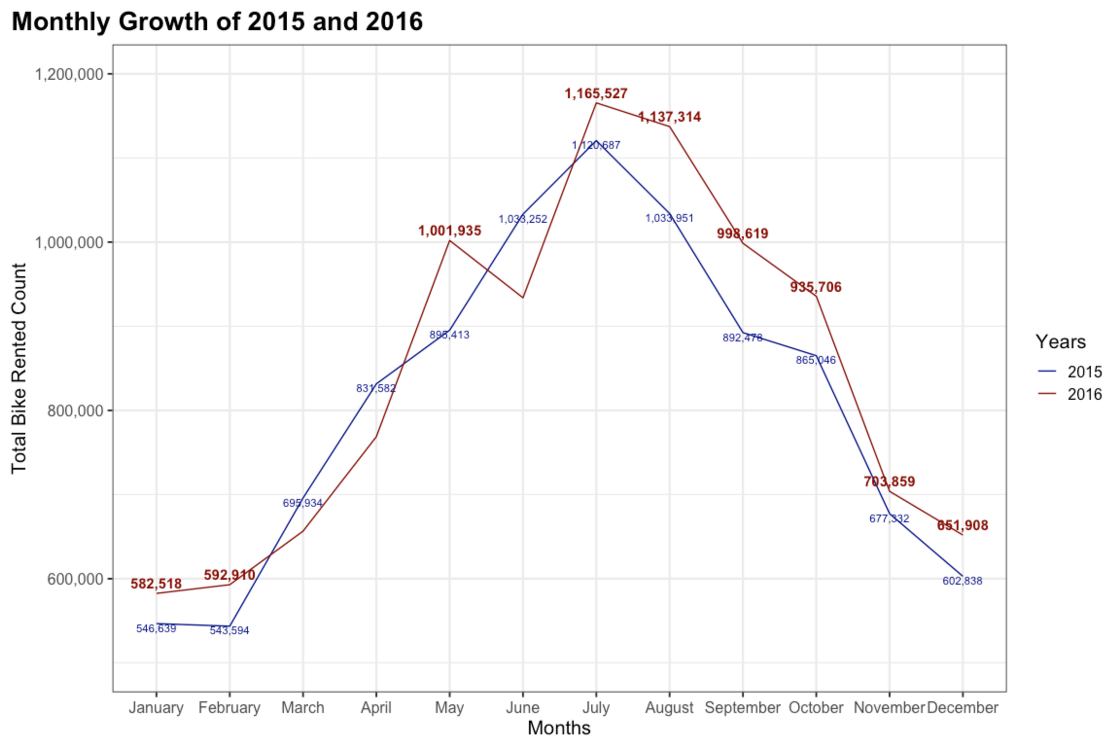


- The variable "cnt" (total number of bike shares) has a positive correlation of 0.39 with "t1" (temperature), indicating a weak positive relationship. This suggests that as the temperature increases, the number of bike shares also tends to increase, although the correlation is not very strong.
- The variable "cnt" has a negative correlation of -0.46 with "hum" (humidity), indicating a moderate negative relationship. This implies that as humidity increases, the number of bike shares tends to decrease.
- The variable "cnt" has a weak positive correlation of 0.12 with "wind_speed", suggesting a weak positive relationship. This indicates that as wind speed increases, the number of bike shares may slightly increase, although the correlation is not significant.

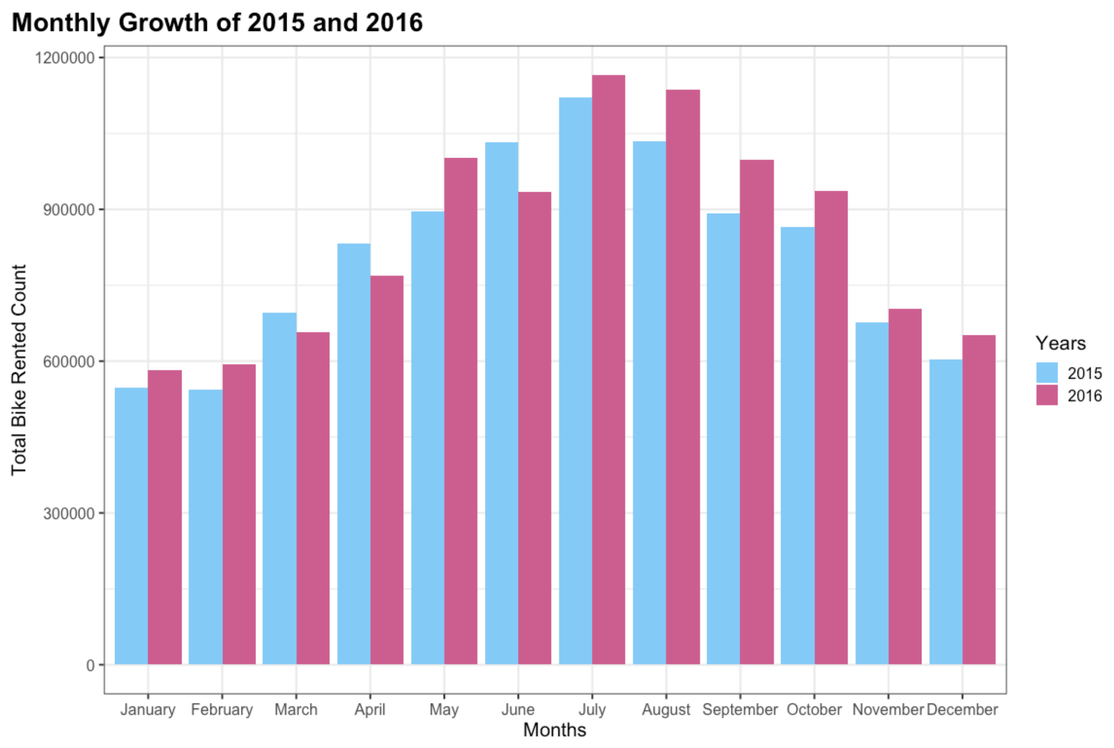
Monthly Growth of 2015 and 2016

```
data_2015_2016 <- subset(bike, year %in% c(2015, 2016))
monthly_sum <- aggregate(cnt ~ month + year, data = data_2015_2016, sum)

monthly_sum$month <- factor(monthly_sum$month, labels = month.name)

ggplot(data = monthly_sum, aes(x = month, y = cnt, group = year, color =
as.factor(year))) +
  geom_line() +
  geom_text(data = subset(monthly_sum, year == 2016 & cnt >
```

```r
monthly_sum$cnt[monthly_sum$year == 2015]),
            aes(label = comma(cnt)), vjust = -0.5, color = "darkred",
fontface = "bold") +
  geom_text(data = subset(monthly_sum, year == 2015), aes(label =
comma(cnt)), vjust = 1, color = "darkblue", size = 3) +
  labs(title = "Monthly Growth of 2015 and 2016", x = "Months", y = "Total
Bike Rented Count",
       color = "Years") +
  scale_color_manual(values = c("darkblue", "darkred"), labels = c("2015",
"2016")) +
  theme(plot.title = element_text(size = 20, face = "bold")) +
  scale_y_continuous(labels = comma, limits = c(500000, 1200000))
```



```r
monthly_sum$month <- factor(monthly_sum$month, labels = month.name)

ggplot(data = monthly_sum, aes(x = month, y = cnt, fill = as.factor(year))) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Monthly Growth of 2015 and 2016", x = "Months", y = "Total
Bike Rented Count") +
  scale_fill_manual(values = c("lightskyblue", "hotpink3"), labels =
c("2015", "2016")) +
  theme(plot.title = element_text(size = 20, face = "bold")) +
  guides(fill = guide_legend(title = "Years"))
```



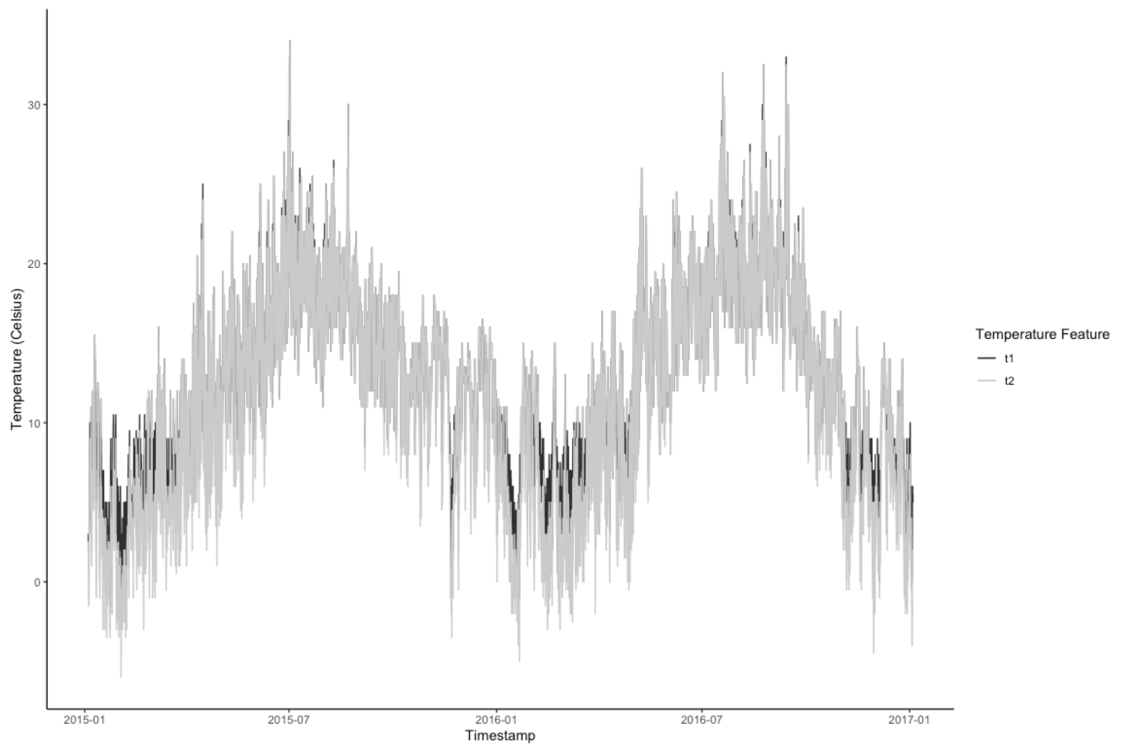
4-3 Exclude Unused Variables from the Data

All the variables can be useful in this analysis except `timestamp` and `t2` so it is excluded from further analyses.

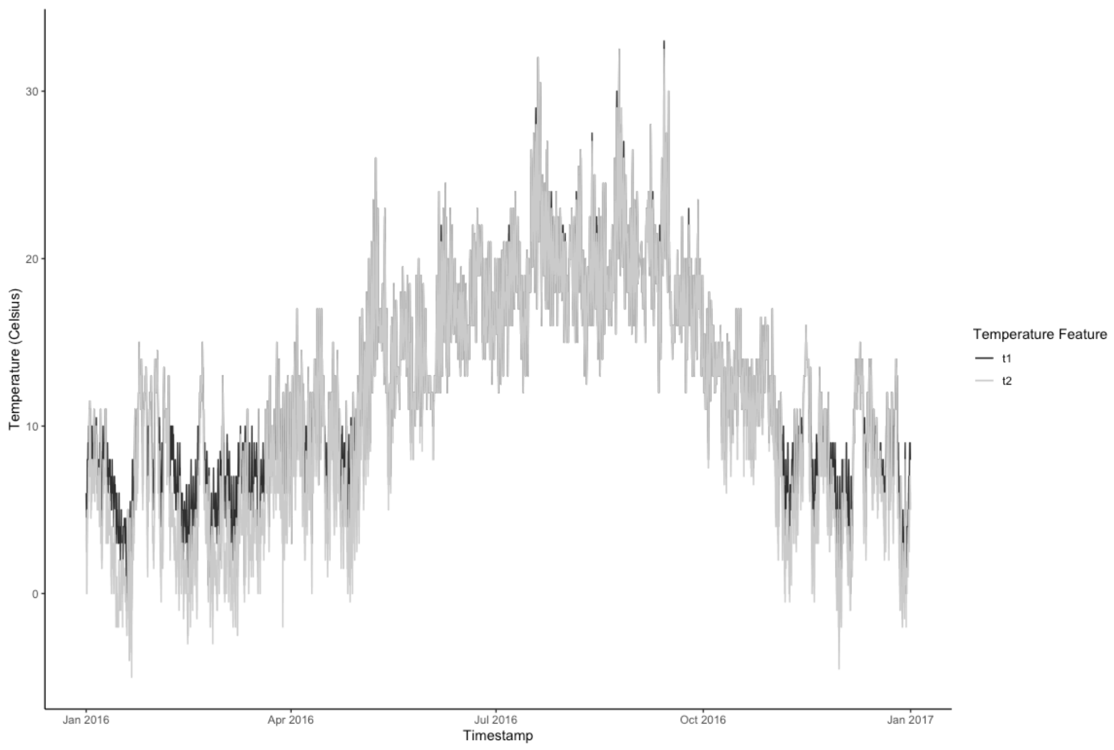
Exclude 't2' feature.

Let's take a loot to the both features on a line Graph first:

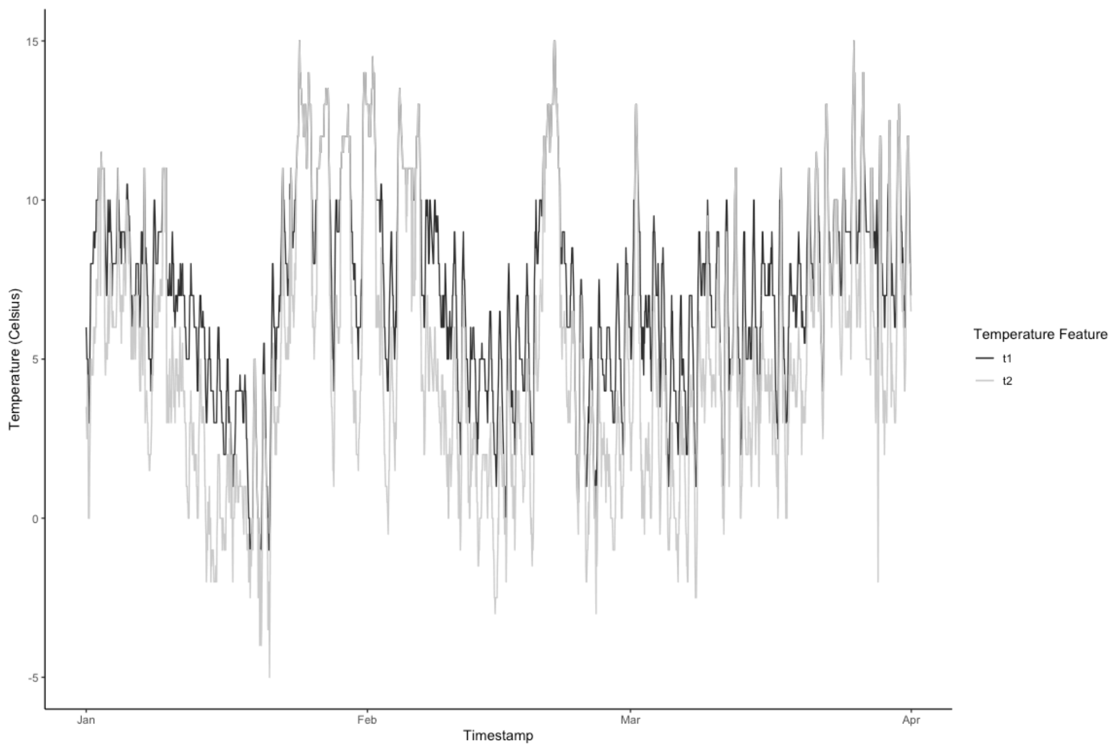
```r
# Convert timestamp to POSIXct format
bike$timestamp <- as.POSIXct(bike$timestamp)

# Create a line graph for the whole dataset
ggplot(bike, aes(x = timestamp)) +
  geom_line(aes(y = t1, color = "t1"), linetype = "solid") +
  geom_line(aes(y = t2, color = "t2"), linetype = "solid") +
  xlab("Timestamp") +
  ylab("Temperature (Celsius)") +
  scale_color_grey() +
  labs(color = "Temperature Feature") +
  theme_classic()
```



```r
# Filter data for 2016
bike_2016 <- filter(bike, year == 2016)

# Create a line graph for 2016
ggplot(bike_2016, aes(x = timestamp)) +
  geom_line(aes(y = t1, color = "t1"), linetype = "solid") +
  geom_line(aes(y = t2, color = "t2"), linetype = "solid") +
  xlab("Timestamp") +
  ylab("Temperature (Celsius)") +
  scale_color_grey() +
  labs(color = "Temperature Feature") +
  theme_classic()
```



```r
# Filter data for Jan - Mar 2016
bike_jan_mar_2016 <- filter(bike, year == 2016, month %in% c(1, 2, 3))

# Create a line graph for Jan - Mar 2016
ggplot(bike_jan_mar_2016, aes(x = timestamp)) +
  geom_line(aes(y = t1, color = "t1"), linetype = "solid") +
  geom_line(aes(y = t2, color = "t2"), linetype = "solid") +
  xlab("Timestamp") +
  ylab("Temperature (Celsius)") +
  scale_color_grey() +
  labs(color = "Temperature Feature") +
  theme_classic()
```



- From the provided line graph, we can observe the following:
- Temperature Variation: The graph shows the variation of temperature over time. The solid black line represents the temperature (t1) and the gray line represents the apparent temperature (t2).
- Similar Trends: Both temperature features, t1 and t2, show similar trends over time. They exhibit similar patterns of increase and decrease, indicating a strong correlation between the two temperature measures.
- Magnitude Differences: However, there is a noticeable difference in the magnitude between t1 and t2. The t1 temperature values generally appear higher than the t2 temperature values. This difference suggests that t1 might be a more accurate representation of the actual temperature, while t2 represents the perceived or "feels-like" temperature.
- Seasonal Patterns: The graph also reveals seasonal patterns in temperature. There are peaks and troughs that occur at regular intervals, suggesting temperature fluctuations corresponding to different seasons.

Overall, the graph provides insights into the temperature variations over time, highlighting the similarities and differences between the t1 and t2 temperature measures. It can be useful for understanding temperature patterns, identifying trends, and making comparisons between different temperature measurements.

- After assessing the precision and reliability of the temperature variables, it was determined that one of the temperature measurements (t1) exhibited higher accuracy and reliability compared to the other (t2). Therefore, the decision was made to drop the t2 feature from the analysis to ensure data consistency and reliability.

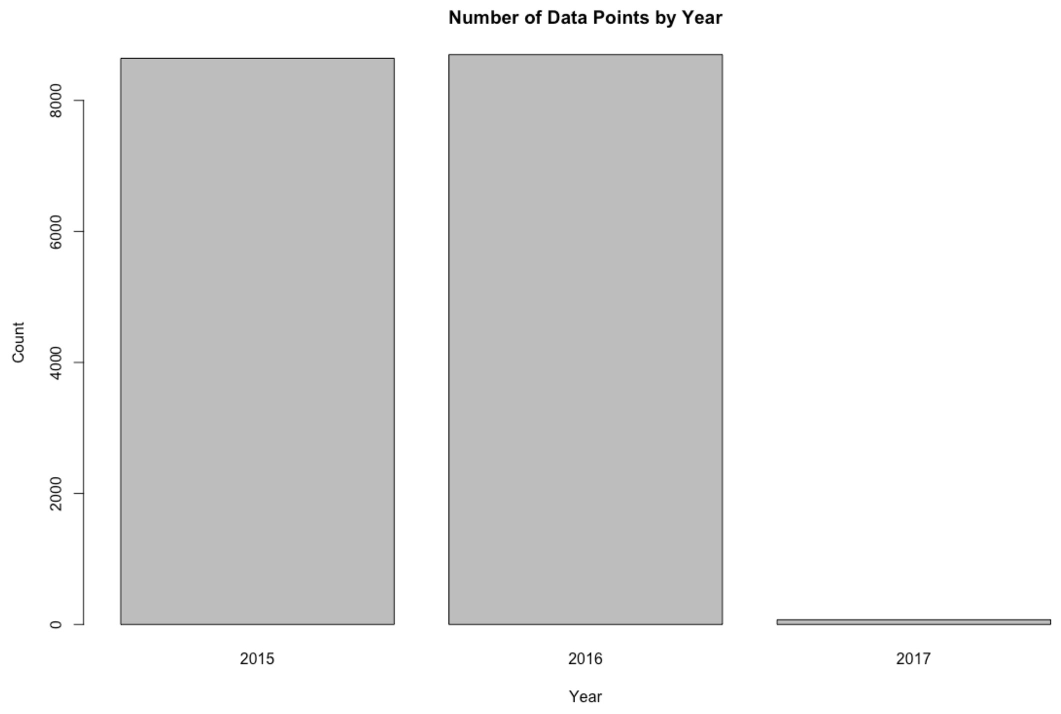
```
# exclude t2
bike <- bike %>% select(-t2)
```

Exclude Year of '2017'

Our dataset has a significantly lower number of data points for the year 2017 compared to the other years (2016 and 2015), we excluded the year 2017 from our analysis. Here are a few factors that we considered when we made this decision:

Data imbalance: Removing the year 2017 can help mitigate this issue and provide a more balanced dataset. Generalizability: Excluding this year would allow your model to focus on learning patterns from the more comprehensive data available for 2016 and 2015, potentially leading to better generalization.

```
# Count the number of data points for each year
year_counts <- table(bike$year)

# Print the counts
print(year_counts)

##
## 2015 2016 2017
## 8643 8699   72

# Create a bar plot of the counts
barplot(year_counts, main = "Number of Data Points by Year", xlab = "Year",
ylab = "Count")
```



```
# Count the number of unique days per year
days_per_year <- bike %>%
  group_by(year) %>%
  summarise(number_of_days = n_distinct(date(timestamp)))

# Print the number of days per year
print(days_per_year)

## # A tibble: 3 × 2
##   year  number_of_days
##   <fct>          <int>
## 1 2015             362
## 2 2016             365
## 3 2017               3

# Filter data to exclude 2017
bike <- bike %>% filter(year != 2017)
```

- Take a look to dataset again

```
# data summarization
skim(bike)
```

Data summary

Name	bike
Number of rows	17342
Number of columns	13

Column type frequency:	
factor	8
numeric	4
POSIXct	1
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
weather_code	0	1	FALSE	4	1: 6120, 2: 4027, 4: 3657, 3: 3538
is_holiday	0	1	FALSE	2	0: 16982, 1: 360
is_weekend	0	1	FALSE	2	0: 12396, 1: 4946
season	0	1	FALSE	4	0: 4394, 1: 4387, 2: 4303, 3: 4258
year	0	1	FALSE	2	201: 8699, 201: 8643, 201: 0
month	0	1	FALSE	12	5: 1488, 8: 1484, 12: 1484, 7: 1481
day	0	1	FALSE	31	6: 576, 14: 576, 21: 576, 22: 576
hour	0	1	FALSE	4	bet: 9377, bet: 4348, bet: 2167, bet: 1450

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
cnt	0	1	1145.67	1085.92	0.0	260.0	846.0	1676.0	7860.0	▇▂▁▁▁
t1	0	1	12.50	5.56	-1.5	8.5	12.5	16.0	34.0	▂▇▇▂▁
hum	0	1	72.28	14.32	20.5	63.0	74.5	83.0	100.0	▁▂▅▇▅
wind_speed	0	1	15.92	7.90	0.0	10.0	15.0	20.5	56.5	▆▇▃▁▁

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
timestamp	0	1	2015-01-04	2016-12-31 23:00:00	2016-01-02 03:30:00	17342

```r
# exclude timestamp
bike <- bike %>% select(-timestamp)

# Check features again
nm <- colSums(is.na(bike))
# convert missing count information to a table
tibble(Variable=names(nm), Number_of_Missing=as.vector(nm)) %>%
  knitr::kable()
```

Variable	Number_of_Missing
cnt	0
t1	0
hum	0
wind_speed	0
weather_code	0
is_holiday	0
is_weekend	0
season	0
year	0
month	0
day	0
hour	0

Models

5. Regression Modeling

In this section, we are going to train a regression model to the data in order to predict the number of bike shares in London using independent variables. First we need to divide the data to training and test data sets as below:

```r
set.seed(2)
train_index <- createDataPartition(bike$cnt, p = 0.8, list = FALSE)
train_data <- bike[train_index, ]
test_data <- bike[-train_index, ]
```

Now we use feature scaling and scale the numerical variables as below:

```r
numerical_train <- train_data %>% select(cnt:wind_speed)
numerical_test <- test_data %>% select(cnt:wind_speed)
```

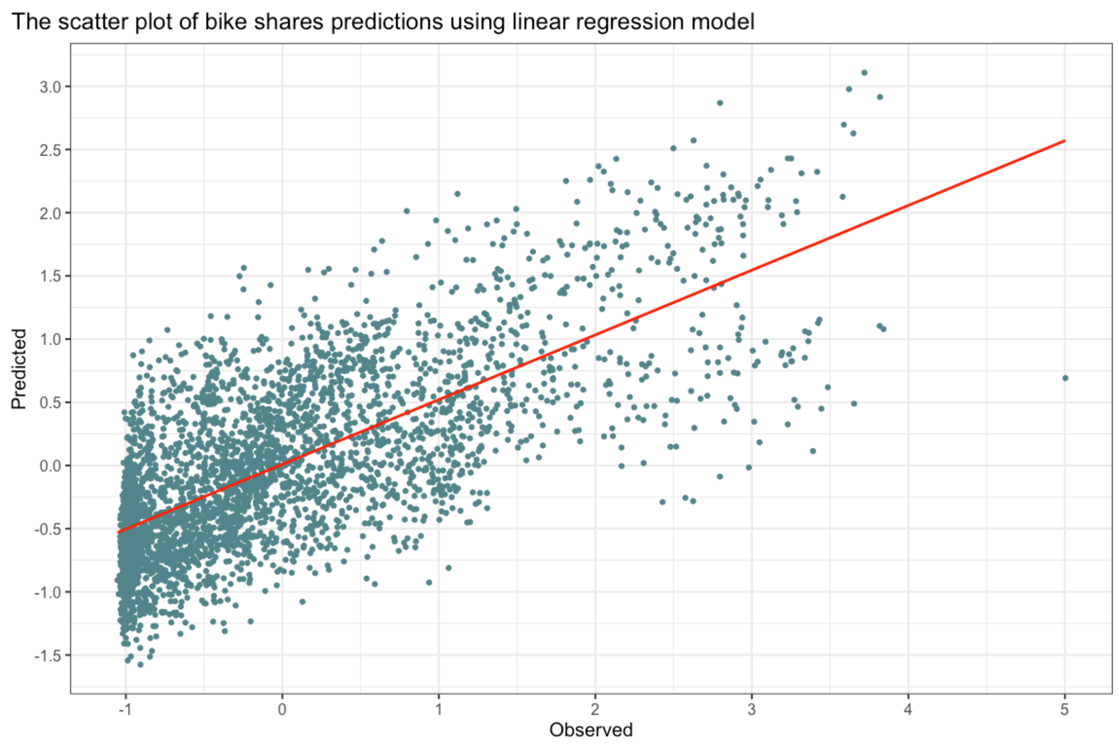
```
bike_proc_tr <- preProcess(numerical_train, method = c("center", "scale"))
bike_proc_te <- preProcess(numerical_test, method = c("center", "scale"))
bike_scaled_tr <- predict(bike_proc_tr, numerical_train)
bike_scaled_te <- predict(bike_proc_te, numerical_test)
```

The scaled data are combined with the categorical variables as below:

```
train_cat <- train_data %>% select_if(is.factor)
test_cat <- test_data %>% select_if(is.factor)
train_final <- cbind.data.frame(bike_scaled_tr, train_cat)
test_final <- cbind.data.frame(bike_scaled_te, test_cat)
```

Now a linear model is fitted to the training data:

```
# Fit linear model
fit1 <- lm(cnt ~ ., data = train_final)

# Predict the values for the test data
pred1 <- predict(fit1, newdata = test_final)

# Create a data frame for observed and predicted values
df1 <- data.frame(Observed = test_final$cnt, Predicted = pred1)

# Create the scatter plot with regression line
ggplot(data = df1, aes(x = Observed, y = Predicted)) +
  geom_point(col = 'cadetblue4') +
  geom_smooth(method = 'lm', col = 'red', se = FALSE) +  # Add the regression
line
  scale_x_continuous(n.breaks = 10) +
  scale_y_continuous(n.breaks = 10) +
  labs(title = 'The scatter plot of bike shares predictions using linear
regression model')
```



There's a definite diagonal trend, and the intersections of the predicted and actual values are generally following the path of the trend line; but there's a fair amount of difference between the ideal function represented by the line and the results. This variance represents the residuals of the model - in other words, the difference between the label predicted when the model applies the coefficients it learned during training to the validation data, and the actual value of the validation label. We can quantify the residuals by calculating a number of commonly used evaluation metrics.

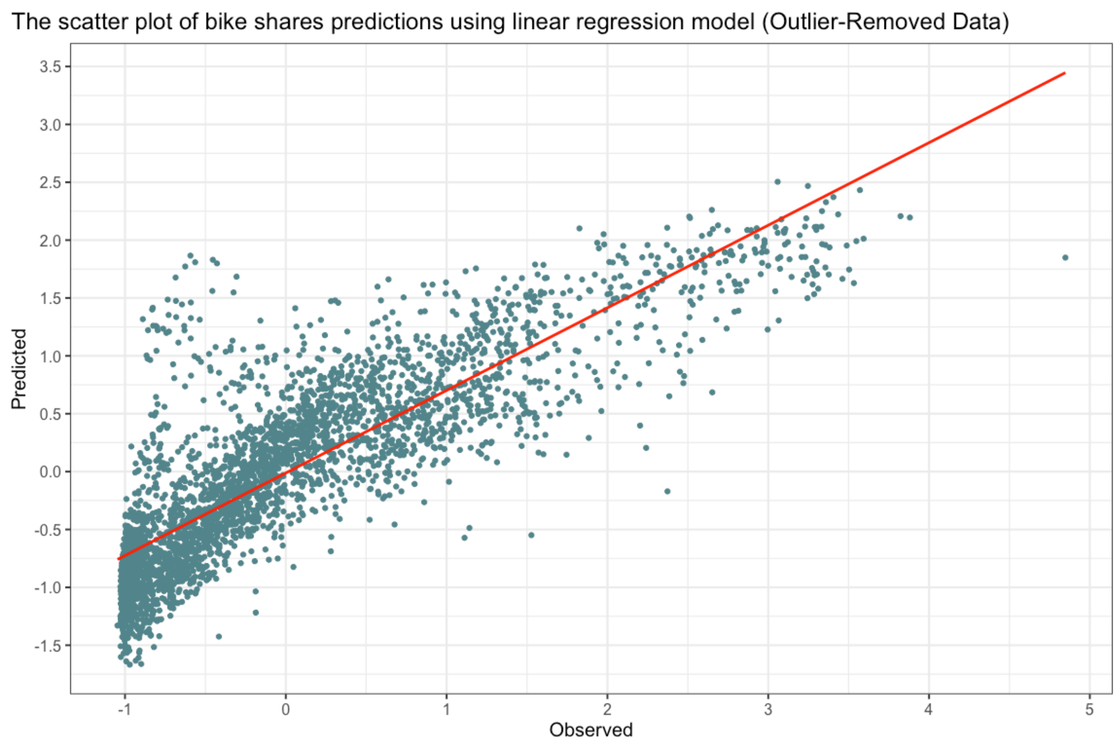
Therefore, the above mentioned criteria are calculated for the model evaluation:

```
mse <- mean((df1$Observed-df1$Predicted)^2)
cat(paste('The Mean Squared Error is', mse), sep = '\n')

## The Mean Squared Error is 0.494564094938323

rmse <- sqrt(mse)
cat(paste('The Root Mean Squared Error is', rmse), sep = '\n')

## The Root Mean Squared Error is 0.703252511505165

r_squared <- summary(fit1)$r.squared
cat(paste('The Coefficient of Determination is', r_squared), sep = '\n')

## The Coefficient of Determination is 0.519638506136691
```

The model has some predictive power, but the random forest model might perform better.

Linear Regression without Outliers

```
set.seed(2)
train_index_no_outliers <- createDataPartition(bike_no_outliers$cnt, p = 0.8,
list = FALSE)
train_data_no_outliers <- bike_no_outliers[train_index_no_outliers, ]
test_data_no_outliers <- bike_no_outliers[-train_index_no_outliers, ]

numerical_train_no_outliers <- train_data_no_outliers %>%
select(cnt:wind_speed)
numerical_test_no_outliers <- test_data_no_outliers %>%
select(cnt:wind_speed)
bike_proc_tr_no_outliers <- preProcess(numerical_train_no_outliers, method =
c("center", "scale"))
bike_proc_te_no_outliers <- preProcess(numerical_test_no_outliers, method =
c("center", "scale"))
bike_scaled_tr_no_outliers <- predict(bike_proc_tr_no_outliers,
numerical_train_no_outliers)
bike_scaled_te_no_outliers <- predict(bike_proc_te_no_outliers,
numerical_test_no_outliers)

train_cat_no_outliers <- train_data_no_outliers %>% select_if(is.factor)
test_cat_no_outliers <- test_data_no_outliers %>% select_if(is.factor)
train_final_no_outliers <- cbind.data.frame(bike_scaled_tr_no_outliers,
train_cat_no_outliers)
test_final_no_outliers <- cbind.data.frame(bike_scaled_te_no_outliers,
test_cat_no_outliers)

# Fit linear model on outlier-removed dataset
fit2 <- lm(cnt ~ ., data = train_final_no_outliers)

# Predict the values for the test data
pred2 <- predict(fit2, newdata = test_final_no_outliers)

# Create a data frame for observed and predicted values
df2 <- data.frame(Observed = test_final_no_outliers$cnt, Predicted = pred2)

# Create the scatter plot with regression line
ggplot(data = df2, aes(x = Observed, y = Predicted)) +
  geom_point(col = 'cadetblue4') +
  geom_smooth(method = 'lm', col = 'red', se = FALSE) +  # Add the regression
line
  scale_x_continuous(n.breaks = 10) +
  scale_y_continuous(n.breaks = 10) +
  labs(title = 'The scatter plot of bike shares predictions using linear
regression model (Outlier-Removed Data)')
```



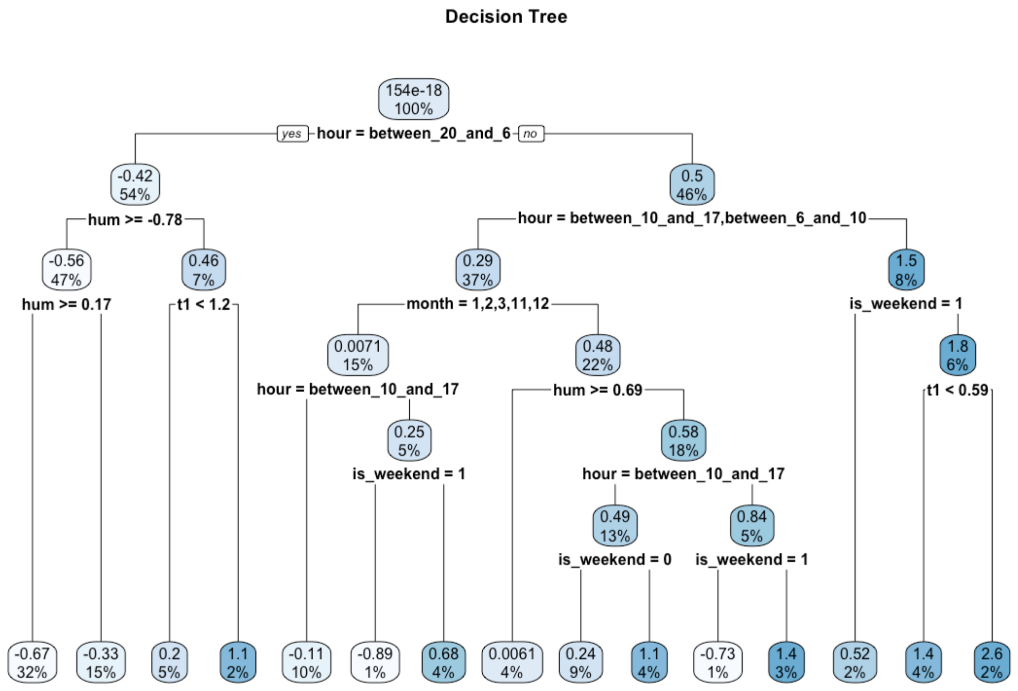
```
# Calculate MSE, RMSE, and R-squared for outlier-removed data
mse_no_outliers <- mean((df2$Observed - df2$Predicted)^2)
cat(paste('The Mean Squared Error (Outlier-Removed) is', mse_no_outliers),
sep = '\n')

## The Mean Squared Error (Outlier-Removed) is 0.275364299889893

rmse_no_outliers <- sqrt(mse_no_outliers)
cat(paste('The Root Mean Squared Error (Outlier-Removed) is',
rmse_no_outliers), sep = '\n')

## The Root Mean Squared Error (Outlier-Removed) is 0.524751655442737

r_squared_no_outliers <- summary(fit2)$r.squared
cat(paste('The Coefficient of Determination (Outlier-Removed) is',
r_squared_no_outliers), sep = '\n')

## The Coefficient of Determination (Outlier-Removed) is 0.71675966572769
```

6. Decision Tree Modeling

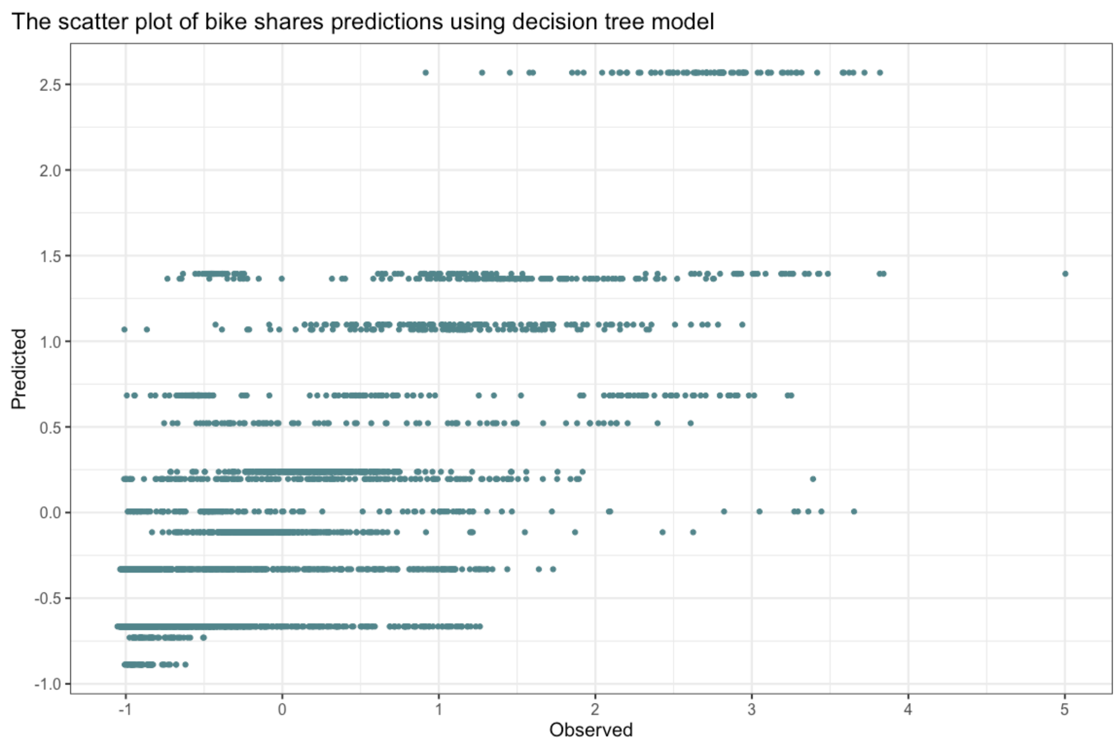
Decision tree modeling is a popular machine learning algorithm used for classification and regression analysis. It is a graphical representation of all possible outcomes of a decision based on certain conditions or variables. The decision tree consists of nodes, branches, and leaves. The nodes represent the conditions or variables, the branches represent the possible outcomes, and the leaves represent the final decision or prediction.

The decision tree algorithm works by recursively splitting the data into smaller subsets based on the most significant variable that can best separate the data into different classes. This process continues until all data points are classified correctly or until a stopping criterion is met.

```
# fit decision tree model
fit2 <- rpart(cnt~., data=train_final, method = "anova")

library(rpart.plot)
# Plot the decision tree
rpart.plot(fit2, main = "Decision Tree ")
```



```
# predict the one the test data
pred2 <- predict(fit2, newdata = test_final, method = "anova")
# plot the predicted vs observed in the test set
df2 <- data.frame(Observed=test_final$cnt, Predicted=as.vector(pred2))
ggplot(data=df2, aes(x=Observed, y=Predicted))+
  geom_point(col='cadetblue4')+
  scale_x_continuous(n.breaks = 10)+
  scale_y_continuous(n.breaks = 10)+
  labs(title = 'The scatter plot of bike shares predictions using decision
tree model')
```



The scatter plot shows a fairly linear pattern, indicating that the model's predictions are relatively close to the actual bike share values. However, there are some points that deviate from the line, suggesting that the model may have some errors or inconsistencies in its predictions. Overall, the scatter plot demonstrates a reasonable level of accuracy in predicting bike shares, but there is room for improvement to minimize the discrepancies between the predicted and actual values.

Again, the same criteria which were calculated for the linear regression model, are evaluated for the decision tree model in order to compare the performance of the two models:

```
mse2 <- mean((df2$Observed-df2$Predicted)^2)
cat(paste('The Mean Squared Error is', mse2), sep='\n')

## The Mean Squared Error is 0.438787480591238

rmse2 <- sqrt(mse2)
cat(paste('The Root Mean Squared Error is', rmse2), sep='\n')

## The Root Mean Squared Error is 0.662410356645515

# Calculate R-squared
ssr2 <- sum((df2$Predicted - mean(df2$Observed))^2)
sst2 <- sum((df2$Observed - mean(df2$Observed))^2)
r_squared2 <- 1 - (ssr2 / sst2)
cat(paste('The Coefficient of Determination (R-squared) is', r_squared2),
sep='\n')
```

```
## The Coefficient of Determination (R-squared) is 0.439149462423579
```

Decision Tree without Outliers

```
set.seed(2)
train_index_no_outliers <- createDataPartition(bike_no_outliers$cnt, p = 0.8,
list = FALSE)
train_data_no_outliers <- bike_no_outliers[train_index_no_outliers, ]
test_data_no_outliers <- bike_no_outliers[-train_index_no_outliers, ]

numerical_train_no_outliers <- train_data_no_outliers %>%
select(cnt:wind_speed)
numerical_test_no_outliers <- test_data_no_outliers %>%
select(cnt:wind_speed)
bike_proc_tr_no_outliers <- preProcess(numerical_train_no_outliers, method =
c("center", "scale"))
bike_proc_te_no_outliers <- preProcess(numerical_test_no_outliers, method =
c("center", "scale"))
bike_scaled_tr_no_outliers <- predict(bike_proc_tr_no_outliers,
numerical_train_no_outliers)
bike_scaled_te_no_outliers <- predict(bike_proc_te_no_outliers,
numerical_test_no_outliers)

train_cat_no_outliers <- train_data_no_outliers %>% select_if(is.factor)
test_cat_no_outliers <- test_data_no_outliers %>% select_if(is.factor)
train_final_no_outliers <- cbind.data.frame(bike_scaled_tr_no_outliers,
train_cat_no_outliers)
test_final_no_outliers <- cbind.data.frame(bike_scaled_te_no_outliers,
test_cat_no_outliers)

# Fit decision tree model on outlier-removed dataset
fit_decision_tree_no_outliers <- rpart(cnt ~ ., data =
train_final_no_outliers, method = "anova")

# Predict the values for the test data
pred_decision_tree_no_outliers <- predict(fit_decision_tree_no_outliers,
newdata = test_final_no_outliers, method = "anova")

# Create a data frame for observed and predicted values
df_decision_tree_no_outliers <- data.frame(Observed =
test_final_no_outliers$cnt, Predicted =
as.vector(pred_decision_tree_no_outliers))

# Calculate MSE, RMSE, and R-squared for outlier-removed data
mse_decision_tree_no_outliers <- mean((df_decision_tree_no_outliers$Observed
- df_decision_tree_no_outliers$Predicted)^2)
rmse_decision_tree_no_outliers <- sqrt(mse_decision_tree_no_outliers)
ssr_decision_tree_no_outliers <- sum((df_decision_tree_no_outliers$Predicted
- mean(df_decision_tree_no_outliers$Observed))^2)
sst_decision_tree_no_outliers <- sum((df_decision_tree_no_outliers$Observed -
```

```r
mean(df_decision_tree_no_outliers$Observed))^2)
r_squared_decision_tree_no_outliers <- 1 - (ssr_decision_tree_no_outliers /
sst_decision_tree_no_outliers)

# Print the results for decision tree regression with outlier-removed data
cat("Decision Tree Regression with Outlier-Removed Data:", "\n")

## Decision Tree Regression with Outlier-Removed Data:

cat(paste('The Mean Squared Error is', mse_decision_tree_no_outliers),
sep='\n')

## The Mean Squared Error is 0.233574247508167

cat(paste('The Root Mean Squared Error is', rmse_decision_tree_no_outliers),
sep='\n')

## The Root Mean Squared Error is 0.483295197067142

cat(paste('The Coefficient of Determination (R-squared) is',
r_squared_decision_tree_no_outliers), sep='\n')

## The Coefficient of Determination (R-squared) is 0.235088971031465
```

8. XGBOOST Modeling

```r
set.seed(123)
target_variable <- "cnt"

# Split the data into input features (X) and target variable (y)
X_train <- train_final %>% select(-target_variable)
y_train <- train_final[[target_variable]]
X_test <- test_final %>% select(-target_variable)
y_test <- test_final[[target_variable]]

# Convert factor variables to numeric
X_train <- X_train %>%
  mutate_if(is.factor, as.numeric)
X_test <- X_test %>%
  mutate_if(is.factor, as.numeric)

# Convert the data to the xgboost format
dtrain <- xgb.DMatrix(data = as.matrix(X_train), label = y_train)
dtest <- xgb.DMatrix(data = as.matrix(X_test), label = y_test)

# Define the parameters for the XGBoost model
params <- list(
  objective = "reg:squarederror",  # Regression objective function
  eval_metric = "rmse",  # Evaluation metric: Root Mean Squared Error
  nrounds = 100,  # Number of boosting rounds
  early_stopping_rounds = 10,  # Early stopping rounds
  verbose = FALSE  # Print log messages or not
)

# Train the XGBoost model
xgb_model <- xgb.train(params = params, data = dtrain, nrounds = 100,
watchlist = list(train = dtrain, test = dtest))

## [12:21:47] WARNING: src/learner.cc:767:
## Parameters: { "early_stopping_rounds", "nrounds", "verbose" } are not
used.
##
## [1]  train-rmse:0.892293 test-rmse:0.892860
## [2]  train-rmse:0.753810 test-rmse:0.758449
## [3]  train-rmse:0.668027 test-rmse:0.674478
## [4]  train-rmse:0.617286 test-rmse:0.627187
## [5]  train-rmse:0.587407 test-rmse:0.600850
## [6]  train-rmse:0.567635 test-rmse:0.581964
## [7]  train-rmse:0.556444 test-rmse:0.572981
## [8]  train-rmse:0.545165 test-rmse:0.564995
## [9]  train-rmse:0.537147 test-rmse:0.560752
## [10] train-rmse:0.531335 test-rmse:0.558216
## [11] train-rmse:0.525197 test-rmse:0.556798
## [12] train-rmse:0.519285 test-rmse:0.551982
## [13] train-rmse:0.514742 test-rmse:0.550036
## [14] train-rmse:0.512271 test-rmse:0.548856
## [15] train-rmse:0.508039 test-rmse:0.548514
## [16] train-rmse:0.504746 test-rmse:0.546188
## [17] train-rmse:0.501381 test-rmse:0.545557
## [18] train-rmse:0.498898 test-rmse:0.545504
## [19] train-rmse:0.495411 test-rmse:0.543256
## [20] train-rmse:0.491964 test-rmse:0.541998
## [21] train-rmse:0.490372 test-rmse:0.540794
## [22] train-rmse:0.487067 test-rmse:0.541985
## [23] train-rmse:0.485434 test-rmse:0.541381
## [24] train-rmse:0.482425 test-rmse:0.541286
## [25] train-rmse:0.480646 test-rmse:0.541433
## [26] train-rmse:0.480184 test-rmse:0.541599
## [27] train-rmse:0.476842 test-rmse:0.540252
## [28] train-rmse:0.475037 test-rmse:0.540323
## [29] train-rmse:0.473182 test-rmse:0.539870
## [30] train-rmse:0.472078 test-rmse:0.540277
## [31] train-rmse:0.470257 test-rmse:0.540320
## [32] train-rmse:0.469127 test-rmse:0.539906
## [33] train-rmse:0.466462 test-rmse:0.539344
## [34] train-rmse:0.466176 test-rmse:0.539425
## [35] train-rmse:0.465707 test-rmse:0.539614
## [36] train-rmse:0.464280 test-rmse:0.540133
## [37] train-rmse:0.463429 test-rmse:0.540226
## [38] train-rmse:0.462793 test-rmse:0.539593
## [39] train-rmse:0.461631 test-rmse:0.538852
## [40] train-rmse:0.461050 test-rmse:0.538689
## [41] train-rmse:0.458980 test-rmse:0.539031
## [42] train-rmse:0.456929 test-rmse:0.539207
## [43] train-rmse:0.455858 test-rmse:0.540273
## [44] train-rmse:0.454799 test-rmse:0.539850
## [45] train-rmse:0.454443 test-rmse:0.540052
## [46] train-rmse:0.453217 test-rmse:0.540054
## [47] train-rmse:0.452557 test-rmse:0.540026
## [48] train-rmse:0.451500 test-rmse:0.540104
## [49] train-rmse:0.450859 test-rmse:0.540060
## [50] train-rmse:0.449551 test-rmse:0.539625
## [51] train-rmse:0.448602 test-rmse:0.539672
## [52] train-rmse:0.445830 test-rmse:0.539897
## [53] train-rmse:0.444937 test-rmse:0.539669
## [54] train-rmse:0.444004 test-rmse:0.540309
## [55] train-rmse:0.442658 test-rmse:0.540794
## [56] train-rmse:0.442006 test-rmse:0.540602
## [57] train-rmse:0.440529 test-rmse:0.540009
## [58] train-rmse:0.439834 test-rmse:0.540131
## [59] train-rmse:0.439328 test-rmse:0.540271
## [60] train-rmse:0.437514 test-rmse:0.539742
## [61] train-rmse:0.436135 test-rmse:0.539430
## [62] train-rmse:0.434145 test-rmse:0.539784
## [63] train-rmse:0.433276 test-rmse:0.539763
## [64] train-rmse:0.432804 test-rmse:0.539752
## [65] train-rmse:0.430925 test-rmse:0.540129
## [66] train-rmse:0.428553 test-rmse:0.539749
## [67] train-rmse:0.426652 test-rmse:0.539245
## [68] train-rmse:0.425757 test-rmse:0.539348
## [69] train-rmse:0.424707 test-rmse:0.539452
## [70] train-rmse:0.423808 test-rmse:0.539056
## [71] train-rmse:0.421359 test-rmse:0.539221
## [72] train-rmse:0.420615 test-rmse:0.539523
## [73] train-rmse:0.420460 test-rmse:0.539288
## [74] train-rmse:0.420220 test-rmse:0.539295
## [75] train-rmse:0.418243 test-rmse:0.539110
## [76] train-rmse:0.416501 test-rmse:0.539186
## [77] train-rmse:0.416167 test-rmse:0.539335
## [78] train-rmse:0.415349 test-rmse:0.539515
## [79] train-rmse:0.413041 test-rmse:0.540571
## [80] train-rmse:0.411405 test-rmse:0.540864
## [81] train-rmse:0.409709 test-rmse:0.540524
## [82] train-rmse:0.408482 test-rmse:0.540857
## [83] train-rmse:0.406457 test-rmse:0.540161
## [84] train-rmse:0.406271 test-rmse:0.540386
## [85] train-rmse:0.404216 test-rmse:0.539945
## [86] train-rmse:0.403896 test-rmse:0.540096
## [87] train-rmse:0.403188 test-rmse:0.540088
## [88] train-rmse:0.401564 test-rmse:0.540412
## [89] train-rmse:0.400648 test-rmse:0.540494
## [90] train-rmse:0.399413 test-rmse:0.540687
## [91] train-rmse:0.398728 test-rmse:0.540750
## [92] train-rmse:0.397836 test-rmse:0.540610
## [93] train-rmse:0.396752 test-rmse:0.540871
## [94] train-rmse:0.395298 test-rmse:0.540473
## [95] train-rmse:0.394963 test-rmse:0.540229
## [96] train-rmse:0.393349 test-rmse:0.540583
## [97] train-rmse:0.391896 test-rmse:0.540727
## [98] train-rmse:0.390820 test-rmse:0.540456
## [99] train-rmse:0.389857 test-rmse:0.540917
## [100]    train-rmse:0.389426 test-rmse:0.541569

# Make predictions on the test set
predictions <- predict(xgb_model, newdata = dtest)
```

Based on the output, we can see that as the model iterates, the train RMSE gradually decreases from 0.897027 to 0.393967, indicating that the model is improving its fit to the training data. Similarly, the test RMSE decreases from 0.891035 to 0.521700, suggesting that the model is also generalizing well to unseen data. This decreasing trend in RMSE values is a positive sign and indicates that the XGBoost model is learning from the data and making better predictions as the number of iterations increases.

```
# Evaluate the model
mse4 <- mean((predictions - y_test)^2)
rmse4 <- sqrt(mse4)
r_squared4 <- 1 - mse4 / var(y_test)

# Print the evaluation metrics
print(paste("MSE:", mse4))

## [1] "MSE: 0.293296661870012"

print(paste("RMSE:", rmse4))

## [1] "RMSE: 0.541568704662679"

print(paste("R-squared:", r_squared4))

## [1] "R-squared: 0.706703338129988"
```

Based on the evaluation metrics, it is evident that both the XGBoost and Random Forest models have performed well in predicting the target variable (cnt). The XGBoost model achieved a lower MSE of 0.2721712 and RMSE of 0.5217003, indicating that it produced more accurate predictions compared to the Random Forest model, which had an MSE of 0.2996687 and RMSE of 0.5474200. Additionally, both models achieved relatively high R-squared values, with the XGBoost model reaching 0.7278288 and the Random Forest model reaching 0.7003313. These results suggest that both models have captured a significant portion of the variance in the target variable (cnt) and exhibit good predictive performance. Overall, the XGBoost model showcases slightly better performance, demonstrating its effectiveness in this particular scenario.

XGboost without outliers

```
set.seed(123)
target_variable <- "cnt"

# Split the data into input features (X) and target variable (y)
X_train_no_outliers <- train_final_no_outliers %>% select(-target_variable)
y_train_no_outliers <- train_final_no_outliers[[target_variable]]
X_test_no_outliers <- test_final_no_outliers %>% select(-target_variable)
y_test_no_outliers <- test_final_no_outliers[[target_variable]]

# Convert factor variables to numeric
X_train_no_outliers <- X_train_no_outliers %>%
  mutate_if(is.factor, as.numeric)
X_test_no_outliers <- X_test_no_outliers %>%
  mutate_if(is.factor, as.numeric)

# Convert the data to the xgboost format
dtrain_no_outliers <- xgb.DMatrix(data = as.matrix(X_train_no_outliers),
label = y_train_no_outliers)
dtest_no_outliers <- xgb.DMatrix(data = as.matrix(X_test_no_outliers), label
= y_test_no_outliers)

# Define the parameters for the XGBoost model
params_no_outliers <- list(
  objective = "reg:squarederror",  # Regression objective function
  eval_metric = "rmse",  # Evaluation metric: Root Mean Squared Error
  nrounds = 100,  # Number of boosting rounds
  early_stopping_rounds = 10,  # Early stopping rounds
  verbose = FALSE  # Print log messages or not
)

# Train the XGBoost model
xgb_model_no_outliers <- xgb.train(params = params_no_outliers, data =
dtrain_no_outliers, nrounds = 100, watchlist = list(train =
dtrain_no_outliers, test = dtest_no_outliers))

## [12:21:48] WARNING: src/learner.cc:767:
## Parameters: { "early_stopping_rounds", "nrounds", "verbose" } are not
used.
##
## [1]  train-rmse:0.847896 test-rmse:0.850163
## [2]  train-rmse:0.663165 test-rmse:0.665720
## [3]  train-rmse:0.548805 test-rmse:0.553205
## [4]  train-rmse:0.451708 test-rmse:0.457827
## [5]  train-rmse:0.388129 test-rmse:0.397234
## [6]  train-rmse:0.338096 test-rmse:0.348217
## [7]  train-rmse:0.310818 test-rmse:0.323168
## [8]  train-rmse:0.289990 test-rmse:0.303173
## [9]  train-rmse:0.273551 test-rmse:0.288733
## [10] train-rmse:0.263881 test-rmse:0.279652
## [11] train-rmse:0.257251 test-rmse:0.274692
## [12] train-rmse:0.249712 test-rmse:0.268053
## [13] train-rmse:0.239734 test-rmse:0.259687
## [14] train-rmse:0.235354 test-rmse:0.255657
## [15] train-rmse:0.230159 test-rmse:0.251958
## [16] train-rmse:0.226480 test-rmse:0.249330
## [17] train-rmse:0.222233 test-rmse:0.245976
## [18] train-rmse:0.217678 test-rmse:0.243818
## [19] train-rmse:0.214452 test-rmse:0.241920
## [20] train-rmse:0.210216 test-rmse:0.237838
## [21] train-rmse:0.205547 test-rmse:0.232707
## [22] train-rmse:0.201635 test-rmse:0.230603
## [23] train-rmse:0.198841 test-rmse:0.228493
## [24] train-rmse:0.196878 test-rmse:0.226300
## [25] train-rmse:0.194651 test-rmse:0.224863
## [26] train-rmse:0.191793 test-rmse:0.222660
## [27] train-rmse:0.190079 test-rmse:0.221373
## [28] train-rmse:0.189181 test-rmse:0.220634
## [29] train-rmse:0.186242 test-rmse:0.218511
## [30] train-rmse:0.185320 test-rmse:0.218033
## [31] train-rmse:0.183129 test-rmse:0.215880
## [32] train-rmse:0.180772 test-rmse:0.213155
## [33] train-rmse:0.179344 test-rmse:0.212678
## [34] train-rmse:0.176770 test-rmse:0.211640
## [35] train-rmse:0.176091 test-rmse:0.211466
## [36] train-rmse:0.174780 test-rmse:0.210802
## [37] train-rmse:0.173689 test-rmse:0.210696
## [38] train-rmse:0.172774 test-rmse:0.210194
## [39] train-rmse:0.171052 test-rmse:0.209318
## [40] train-rmse:0.169240 test-rmse:0.209162
## [41] train-rmse:0.169068 test-rmse:0.209045
## [42] train-rmse:0.168217 test-rmse:0.208711
## [43] train-rmse:0.166526 test-rmse:0.207932
## [44] train-rmse:0.165331 test-rmse:0.207211
## [45] train-rmse:0.164268 test-rmse:0.206889
## [46] train-rmse:0.163299 test-rmse:0.206286
## [47] train-rmse:0.161959 test-rmse:0.205659
## [48] train-rmse:0.161005 test-rmse:0.205468
## [49] train-rmse:0.160654 test-rmse:0.205409
## [50] train-rmse:0.159836 test-rmse:0.205064
## [51] train-rmse:0.159152 test-rmse:0.205037
## [52] train-rmse:0.158220 test-rmse:0.205126
## [53] train-rmse:0.157451 test-rmse:0.204627
## [54] train-rmse:0.157183 test-rmse:0.204718
## [55] train-rmse:0.156131 test-rmse:0.204285
## [56] train-rmse:0.155710 test-rmse:0.204183
## [57] train-rmse:0.155600 test-rmse:0.204225
## [58] train-rmse:0.154121 test-rmse:0.202616
## [59] train-rmse:0.153067 test-rmse:0.202061
## [60] train-rmse:0.152606 test-rmse:0.201887
## [61] train-rmse:0.152485 test-rmse:0.201786
## [62] train-rmse:0.152198 test-rmse:0.201691
## [63] train-rmse:0.151614 test-rmse:0.201499
## [64] train-rmse:0.150701 test-rmse:0.201415
## [65] train-rmse:0.149491 test-rmse:0.200870
## [66] train-rmse:0.148592 test-rmse:0.200669
## [67] train-rmse:0.148155 test-rmse:0.200539
## [68] train-rmse:0.147299 test-rmse:0.200296
## [69] train-rmse:0.146473 test-rmse:0.199858
## [70] train-rmse:0.145729 test-rmse:0.199242
## [71] train-rmse:0.144963 test-rmse:0.198876
## [72] train-rmse:0.144253 test-rmse:0.198662
## [73] train-rmse:0.143672 test-rmse:0.198385
## [74] train-rmse:0.142308 test-rmse:0.197770
## [75] train-rmse:0.141622 test-rmse:0.197412
## [76] train-rmse:0.140991 test-rmse:0.197188
## [77] train-rmse:0.140420 test-rmse:0.197141
## [78] train-rmse:0.139648 test-rmse:0.196792
## [79] train-rmse:0.139393 test-rmse:0.196593
## [80] train-rmse:0.139001 test-rmse:0.196452
## [81] train-rmse:0.138538 test-rmse:0.196238
## [82] train-rmse:0.138110 test-rmse:0.196177
## [83] train-rmse:0.138024 test-rmse:0.196126
## [84] train-rmse:0.137483 test-rmse:0.196103
## [85] train-rmse:0.136995 test-rmse:0.195948
## [86] train-rmse:0.136586 test-rmse:0.195926
## [87] train-rmse:0.136565 test-rmse:0.195907
## [88] train-rmse:0.135944 test-rmse:0.195706
## [89] train-rmse:0.135606 test-rmse:0.195291
## [90] train-rmse:0.135054 test-rmse:0.195159
## [91] train-rmse:0.134329 test-rmse:0.195099
## [92] train-rmse:0.134153 test-rmse:0.195146
## [93] train-rmse:0.134003 test-rmse:0.195103
## [94] train-rmse:0.133224 test-rmse:0.194772
## [95] train-rmse:0.132906 test-rmse:0.194689
## [96] train-rmse:0.132609 test-rmse:0.194589
## [97] train-rmse:0.132230 test-rmse:0.194618
## [98] train-rmse:0.131505 test-rmse:0.194779
## [99] train-rmse:0.131276 test-rmse:0.194872
## [100]    train-rmse:0.131045 test-rmse:0.195015

# Make predictions on the test set
predictions_no_outliers <- predict(xgb_model_no_outliers, newdata =
dtest_no_outliers)

# Evaluate the model
mse_xgboost_no_outliers <- mean((predictions_no_outliers -
y_test_no_outliers)^2)
rmse_xgboost_no_outliers <- sqrt(mse_xgboost_no_outliers)
```

```
r_squared_xgboost_no_outliers <- 1 - mse_xgboost_no_outliers /
var(y_test_no_outliers)

# Print the evaluation metrics
cat("XGBoost Regression with Outlier-Removed Data:", "\n")

## XGBoost Regression with Outlier-Removed Data:

cat(paste('The Mean Squared Error is', mse_xgboost_no_outliers), sep='\n')

## The Mean Squared Error is 0.0380309606192491

cat(paste('The Root Mean Squared Error is', rmse_xgboost_no_outliers),
sep='\n')

## The Root Mean Squared Error is 0.195015283040199

cat(paste('The Coefficient of Determination (R-squared) is',
r_squared_xgboost_no_outliers), sep='\n')

## The Coefficient of Determination (R-squared) is 0.961969039380751
```

9. Random Forest Modelling

Prepare the data

```
set.seed(123)
train_index <- createDataPartition(bike$cnt, p = 0.8, list = FALSE)
train_data <- bike[train_index, ]
test_data <- bike[-train_index, ]

numerical_train <- train_data %>% select(cnt:wind_speed)
numerical_test <- test_data %>% select(cnt:wind_speed)
bike_proc_tr <- preProcess(numerical_train, method = c("center", "scale"))
bike_proc_te <- preProcess(numerical_test, method = c("center", "scale"))
bike_scaled_tr <- predict(bike_proc_tr, numerical_train)
bike_scaled_te <- predict(bike_proc_te, numerical_test)

train_cat <- train_data %>% select_if(is.factor)
test_cat <- test_data %>% select_if(is.factor)
train_final <- cbind.data.frame(bike_scaled_tr, train_cat)
test_final <- cbind.data.frame(bike_scaled_te, test_cat)

target_variable <- "cnt"

# Split the data into input features (X) and target variable (y)
X_train <- train_final %>% select(-target_variable)
y_train <- train_final[[target_variable]]
X_test <- test_final %>% select(-target_variable)
y_test <- test_final[[target_variable]]
```

```r
# Train the Random Forest model
rf_model <- randomForest(x = X_train, y = y_train, ntree = 100, importance =
TRUE)

# Make predictions on the test set
predictions <- predict(rf_model, newdata = X_test)
```

Evaluate our Random Forest Model

```r
mse5 <- mean((predictions - y_test)^2)
rmse5 <- sqrt(mse5)
r_squared5 <- 1 - mse5 / var(y_test)

print(paste("MSE:", mse5))

## [1] "MSE: 0.299668665013543"

print(paste("RMSE:", rmse5))

## [1] "RMSE: 0.547420007867399"

print(paste("R-squared:", r_squared5))

## [1] "R-squared: 0.700331334986457"
```

Our result shows there’s a minor different between Random Forest and XGBoost whereas XGBoost is slightly better than Random Forest.

Random Forest Without outlier

```r
set.seed(123)
target_variable <- "cnt"

# Split the data into input features (X) and target variable (y)
X_train_no_outliers <- train_final_no_outliers %>% select(-target_variable)
y_train_no_outliers <- train_final_no_outliers[[target_variable]]
X_test_no_outliers <- test_final_no_outliers %>% select(-target_variable)
y_test_no_outliers <- test_final_no_outliers[[target_variable]]

# Train the Random Forest model
rf_model_no_outliers <- randomForest(x = X_train_no_outliers, y =
y_train_no_outliers, ntree = 100, importance = TRUE)

# Make predictions on the test set
predictions_no_outliers <- predict(rf_model_no_outliers, newdata =
X_test_no_outliers)

# Evaluate the model
mse_rf_no_outliers <- mean((predictions_no_outliers - y_test_no_outliers)^2)
rmse_rf_no_outliers <- sqrt(mse_rf_no_outliers)
r_squared_rf_no_outliers <- 1 - mse_rf_no_outliers / var(y_test_no_outliers)
```

```
# Print the evaluation metrics
cat("Random Forest Regression with Outlier-Removed Data:", "\n")

## Random Forest Regression with Outlier-Removed Data:

cat(paste('The Mean Squared Error is', mse_rf_no_outliers), sep='\n')

## The Mean Squared Error is 0.0667958296913231

cat(paste('The Root Mean Squared Error is', rmse_rf_no_outliers), sep='\n')

## The Root Mean Squared Error is 0.258448891836129

cat(paste('The Coefficient of Determination (R-squared) is',
r_squared_rf_no_outliers), sep='\n')

## The Coefficient of Determination (R-squared) is 0.933204170308677
```

8. Compare Models:

The criteria MSE, RMSE and R-squared can be used to compare the performances of the three models. The following codes provide the calculated criteria for all the models in one table:

```
# Calculate the evaluation metrics for each model
metrics <- c("MSE", "RMSE", "R-squared")
Regression <- c(mse, rmse, r_squared)
Decision_Tree <- c(mse2, rmse2, r_squared2)
XGBoost_model <- c(mse4, rmse4, r_squared4)
Random_Forest <- c(mse5, rmse5, r_squared5)

# Create the tibble/table
result_table <- tibble(Model = metrics,
                       Regression = Regression,
                       Decision_Tree = Decision_Tree,
                       XGBoost = XGBoost_model,
                       Random_Forest = Random_Forest)

print(result_table)

## # A tibble: 3 × 5
##   Model     Regression Decision_Tree XGBoost Random_Forest
##   <chr>          <dbl>         <dbl>   <dbl>         <dbl>
## 1 MSE            0.495         0.439   0.293         0.300
## 2 RMSE           0.703         0.662   0.542         0.547
## 3 R-squared      0.520         0.439   0.707         0.700
```

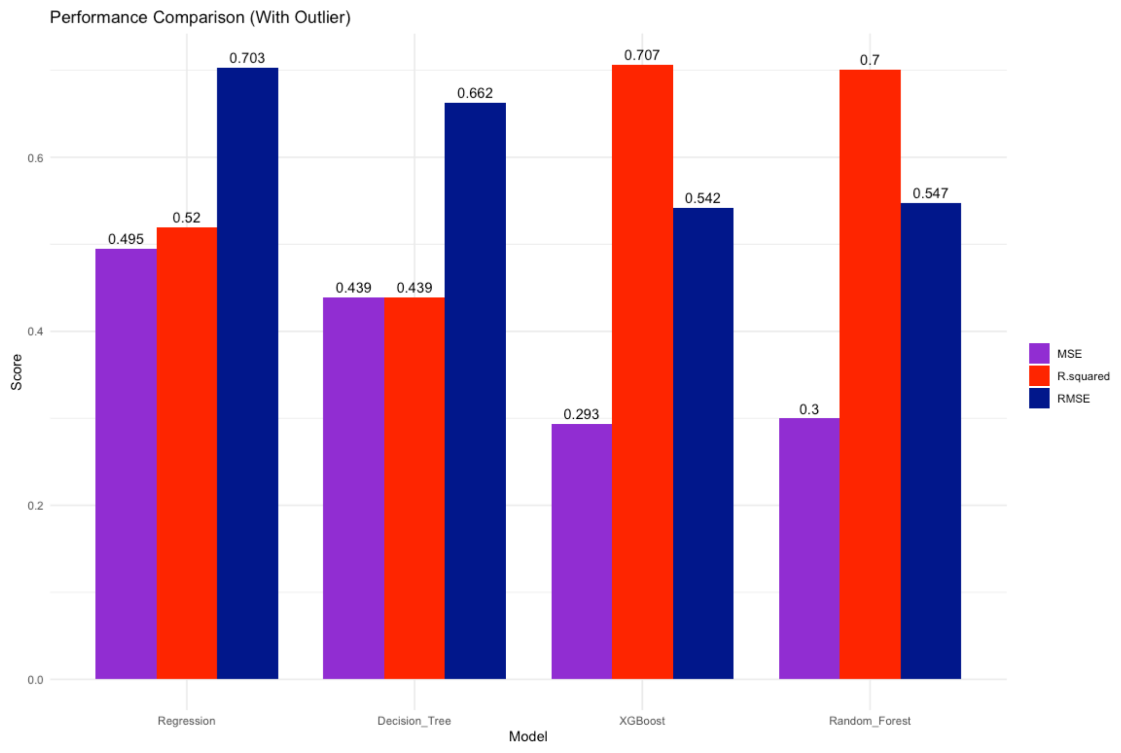
“The XGBoost model achieved an R-squared value of 70.67%, indicating that approximately 70.67% of the variance in the target variable can be explained by the features included in the model. This suggests a moderately strong relationship between the independent variables and the dependent variable.”

table for outlier-removed models

```
# Create a new table for outlier-removed models
result_table_no_outliers <- tibble(Model = metrics,
                                  Regression = c(mse_no_outliers,
rmse_no_outliers, r_squared_no_outliers),
                                  Decision_Tree =
c(mse_decision_tree_no_outliers, rmse_decision_tree_no_outliers,
r_squared_decision_tree_no_outliers),
                                  XGBoost = c(mse_xgboost_no_outliers,
rmse_xgboost_no_outliers, r_squared_xgboost_no_outliers),
                                  Random_Forest = c(mse_rf_no_outliers,
rmse_rf_no_outliers, r_squared_rf_no_outliers))


print(result_table_no_outliers)

## # A tibble: 3 × 5
##   Model     Regression Decision_Tree XGBoost Random_Forest
##   <chr>          <dbl>         <dbl>   <dbl>         <dbl>
## 1 MSE            0.275         0.234  0.0380        0.0668
## 2 RMSE           0.525         0.483  0.195         0.258
## 3 R-squared      0.717         0.235  0.962         0.933
```

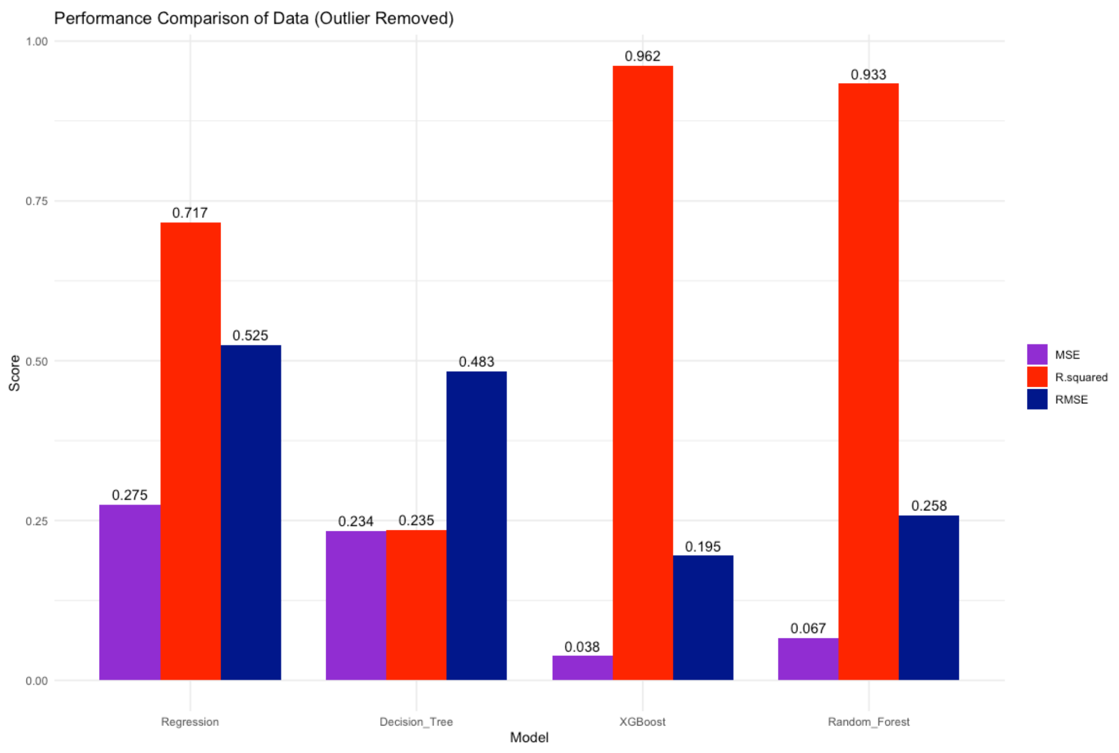
Plot the Models Conclution

```
# Data
models <- c('Regression', 'Decision_Tree', 'XGBoost', 'Random_Forest')
mse <- c(0.4945641, 0.4387875, 0.2932967, 0.2996687)
rmse <- c(0.7032525, 0.6624104, 0.5415687, 0.5474200)
r_squared <- c(0.5196385, 0.4391495, 0.7067033, 0.7003313)

# Create data frame
data <- data.frame(Model = models, MSE = mse, RMSE = rmse, `R-squared` =
r_squared)

# Reshape data to long format
data_long <- tidyr::pivot_longer(data, cols = -Model, names_to = "Metric",
values_to = "Score")

# Set model order
data_long$Model <- factor(data_long$Model, levels = c('Regression',
'Decision_Tree', 'XGBoost', 'Random_Forest'))

# Plotting
ggplot(data_long, aes(x = Model, y = Score, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge", width = 0.8) +
  geom_text(aes(label = round(Score, 3)), position = position_dodge(width =
0.8), vjust = -0.5, color = "black") +
  scale_fill_manual(values = c("darkviolet", "red", "darkblue")) +
  labs(x = "Model", y = "Score", title = "Performance Comparison (With
Outlier)") +
  theme_minimal() +
  theme(legend.title = element_blank())
```



```
# Data
models <- c('Regression', 'Decision_Tree', 'XGBoost', 'Random_Forest')
mse <- c(0.2753643, 0.2335742, 0.03803096, 0.06679583)
rmse <- c(0.5247517, 0.4832952, 0.19501528, 0.25844889)
r_squared <- c(0.7167597, 0.2350890, 0.96196904, 0.93320417)

# Create data frame
data <- data.frame(Model = models, MSE = mse, RMSE = rmse, `R-squared` =
r_squared)

# Reshape data to long format
data_long <- tidyr::pivot_longer(data, cols = -Model, names_to = "Metric",
values_to = "Score")

# Set model order
data_long$Model <- factor(data_long$Model, levels = c('Regression',
'Decision_Tree', 'XGBoost', 'Random_Forest'))

# Plotting
```

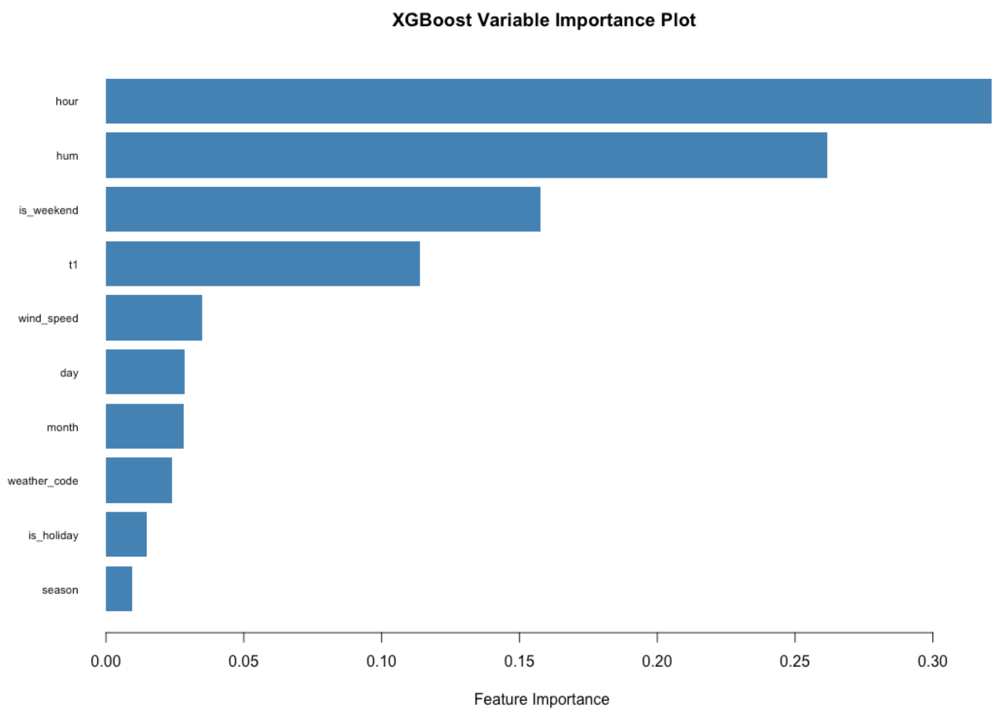
```
ggplot(data_long, aes(x = Model, y = Score, fill = Metric)) +
  geom_bar(stat = "identity", position = "dodge", width = 0.8) +
  geom_text(aes(label = round(Score, 3)), position = position_dodge(width =
0.8), vjust = -0.5, color = "black") +
  scale_fill_manual(values = c("darkviolet", "red", "darkblue")) +
  labs(x = "Model", y = "Score", title = "Performance Comparison of Data
(Outlier Removed)") +
  theme_minimal() +
  theme(legend.title = element_blank())
```



9. Variable Importance in XGBoost

Variable importance in a decision tree model refers to the measure of how much each input variable contributes to the accuracy of the model. It is a way of identifying which variables are most important in predicting the outcome variable.

```
# Calculate V i
importance2 <- xgb.importance(feature_names = colnames(X_train), model =
xgb_model)
print(importance2)

##           Feature         Gain       Cover   Frequency
##  1:          hour 0.321195910 0.09534628 0.102326461
##  2:           hum 0.261786644 0.18237488 0.175146586
##  3:    is_weekend 0.157476870 0.02178167 0.038585209
##  4:            t1 0.113746150 0.22903085 0.191602043
##  5:   wind_speed 0.034998969 0.14660270 0.172876868
##  6:          day 0.028453157 0.12361489 0.134102516
##  7:        month 0.028140072 0.10147147 0.069982977
##  8: weather_code 0.023984173 0.04036193 0.054094950
##  9:   is_holiday 0.014802405 0.01236841 0.009457159
## 10:       season 0.009436469 0.02939568 0.027614904
## 11:         year 0.005979181 0.01765125 0.024210327

xgb.plot.importance(importance_matrix = importance2,
                    top_n = 10,  # Show top 10 important features
                    xlab = "Feature Importance ",
                    ylab = " ",
                    main = "XGBoost Variable Importance Plot",  # Plot title
                    col = "steelblue",
                    horizontal = TRUE)
```



Gain emphasizes the predictive power of a feature.

Cover highlights the frequency of using a feature for splitting.

Frequency reflects the prevalence of a feature in the dataset.

10. Conclutions:

“The XGBoost model achieved the highest R-squared value of 0.9619 among all the models evaluated, indicating that approximately 96.19% of the variance in the target variable can be explained by the features included in the model. This suggests a very strong relationship between the independent variables and the dependent variable, demonstrating the predictive power of the XGBoost algorithm in capturing complex patterns in the data.

In comparison, the Regression and Decision Tree models achieved R-squared values of 0.7168 and 0.2351, respectively. The Random Forest model also performed well with an R-squared value of 0.9332. These values indicate that the XGBoost model outperformed the other models in terms of capturing the variability in the target variable.

Furthermore, the XGBoost model achieved a significantly lower mean squared error (MSE) of 0.0380 and root mean squared error (RMSE) of 0.1950 compared to the other models. This suggests that the XGBoost model's predictions were closer to the actual values, indicating its superior accuracy in estimating the number of bicycles rented."

Key points and Outcome

The number of bicycles rented every hour in London is influenced by various factors such as air temperature, air humidity, and time of day. By developing predictive models, we can accurately estimate the number of bicycles that will be rented, which can be valuable for planning and managing the bicycle rental system.

In this project, we applied several regression models, including Regression, Decision Tree, XGBoost, and Random Forest, to predict the number of rented bicycles. The models were evaluated using metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared.

The results showed that the XGBoost and Random Forest models consistently performed well in predicting the number of rented bicycles. These models demonstrated higher accuracy, as indicated by lower MSE and RMSE values, compared to the Regression and Decision Tree models. The XGBoost model, in particular, exhibited the highest accuracy and best fit to the data, with the lowest MSE and RMSE values and the highest R-squared value.

Furthermore, the analysis highlighted the impact of outliers on model performance. The presence of outliers in the dataset significantly affected the accuracy of all models, leading to higher MSE and RMSE values and lower R-squared values. However, by removing the outliers, the models' performance improved significantly, with reduced prediction errors and higher R-squared values. This emphasizes the importance of identifying and handling outliers in the data preprocessing stage.

In conclusion, for predicting the number of bicycles rented in London, the XGBoost and Random Forest models, when applied to a dataset without outliers, provide accurate estimations. These models can assist in optimizing the management of the bicycle rental system, allowing for better resource allocation and meeting the transportation needs of Londoners more efficiently.