Applied Data Science with R Capstone project

Emmanuel Oluwaseun Okomolehin

23 Aug 2023

Outline

- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

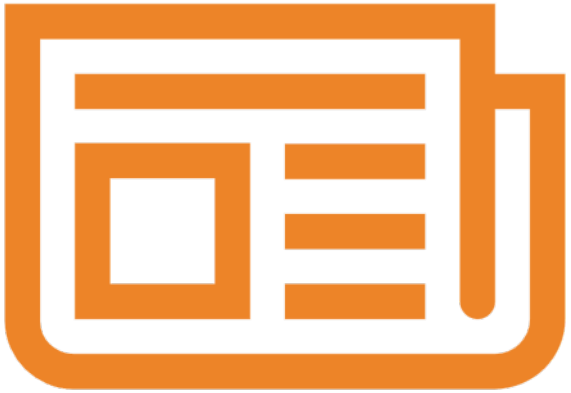
Executive Summary

- Objective: Analyze the Influence of Season, Weather, and Time on Bike Demand using Seoul Bike Sharing Data.
- Dataset: Utilized historical bike-sharing data spanning multiple seasons and years, alongside weather conditions and timestamps.
- Key Findings:
  - **Seasonal Patterns:** Spring and fall experience peak demand, while winter sees the lowest usage due to weather constraints.
  - **Weather Impact:** Rainy days lead to decreased demand; clear weather and moderate temperatures correlate with higher usage.
  - **Time of Day:** Peak demand during the morning (8-10 AM) and evening (5-7 PM) commuter hours; off-peak hours exhibit lower demand.
- Implications:
  - **Operational Efficiency:** Optimize bike allocation based on season and weather forecasts to meet demand.
  - **Marketing Strategies:** Targeted promotions during peak hours could enhance user engagement.
  - **User Experience:** Provide real-time weather information to assist users' decisions.

Introduction

- Purpose of the Report: Analyzing the Impact of Season, Weather, and Time on Bike Sharing Demand using Seoul Bike Sharing Dataset.
- Objective: Provide insights into factors influencing bike usage patterns to optimize resource allocation and enhance user experience.
- Seoul Bike Sharing Dataset: Historical data containing bike usage, weather conditions, and timestamps.

Methodology



- Perform data collection
- Perform data wrangling
- Perform exploratory data analysis (EDA) using SQL and visualization
- Perform predictive analysis using regression models
  - How to build the baseline model
  - How to improve the baseline model
- Build a R Shiny dashboard app

Methodology

Data Analysis

Feature Selection

Machine Learning

Data collection

I collected real-time current and forecasted weather data for cities using the **OpenWeather API**. It can give current weather data for any location including over 200,000 cities and 5-day forecasts for free (with limited API usage). I used HTTP requests to call those weather APIs and get the weather data.


```
# $weather is also a list with one element, its $main element indicates the weather status such as clear or rain
weather <- c(weather, json_result$weather[[1]]$main)
# Get Visibility
visibility <- c(visibility, json_result$visibility)
# Get current temperature
temp <- c(temp, json_result$main$temp)
# Get min temperature
temp_min <- c(temp_min, json_result$main$temp_min)
# Get max temperature
temp_max <- c(temp_max, json_result$main$temp_max)
# Get pressure
pressure <- c(pressure, json_result$main$pressure)
# Get humidity
humidity <- c(humidity, json_result$main$humidity)
# Get wind speed
wind_speed <- c(wind_speed, json_result$wind$speed)
# Get wind direction
wind_deg <- c(wind_deg, json_result$wind$deg)
```

Combine all vectors as columns of a data frame

```
# Combine all vectors
weather_data_frame <- data.frame(weather=weather,
                                 visibility=visibility,
                                 temp=temp,
                                 temp_min=temp_min,
                                 temp_max=temp_max,
                                 pressure=pressure,
                                 humidity=humidity,
                                 wind_speed=wind_speed,
                                 wind_deg=wind_deg)
```

```
# Check the generated data frame
print(weather_data_frame)
```

```
  weather visibility  temp temp_min temp_max pressure humidity wind_speed
1  Clouds      10000 25.03    23.66    25.78     1004       83       4.12
  wind_deg
1      140
```

Data wrangling

- Use regular expressions, along with the stringr package (part of tidyverse), to clean up the bike-sharing systems data that you previously web-scraped from the wiki page
- Standardize column names for all collected datasets
- Remove undesired reference links from the scraped bike-sharing systems dataset
- Extract only the numeric value from undesired text annotations
- Detect and handle missing values
- Create indicator (dummy) variables for categorical variables
- Normalize data

steps
- Column names put in UPPERCASE
- The word separator changed to be an underscore

steps
- write a custom function using stringr::str_replace_all to replace all reference links with an empty character for columns CITY and SYSTEM
- Use the dplyr::mutate() function to apply the remove_ref function to the CITY and SYSTEM columns

steps
- Write a custom function using stringr::str_extract to extract the first digital substring match and convert it into numeric type
- Drop rows with missing values in the RENTED_BIKE_COUNT column

steps
- Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.
- Apply min-max normalization on RENTED_BIKE_COUNT, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, SNOWFALL
- Save the dataset

```r
# Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.
dummy_bike_sharing_df <- bike_sharing_df %>%
        mutate(dummy =1) %>%
        spread(key = SEASONS, value = dummy, fill = 0)

dummy_bike_sharing_df <- dummy_bike_sharing_df %>%
        mutate(dummy =1) %>%
        spread(HOLIDAY, dummy, 0)

dummy_bike_sharing_df <- dummy_bike_sharing_df %>%
        mutate(dummy =1) %>%
        spread(key = HOUR, value = dummy, fill = 0)
```

```r
# Print the dataset summary again to make sure the indicator columns are created properly
summary(dummy_bike_sharing_df)
glimpse(dummy_bike_sharing_df)
```

```
 Class :character   1st Qu.: 214.0   1st Qu.:  3.00   1st Qu.:42.00
 Mode  :character   Median : 542.0   Median : 13.50   Median :57.00
                    Mean   : 729.2   Mean   : 12.77   Mean   :58.15
                    3rd Qu.:1084.0   3rd Qu.: 22.70   3rd Qu.:74.00
                    Max.   :3556.0   Max.   : 39.40   Max.   :98.00
   WIND_SPEED      VISIBILITY   DEW_POINT_TEMPERATURE SOLAR_RADIATION
 Min.   :0.000   Min.   :  27   Min.   :-30.600       Min.   :0.0000
 1st Qu.:0.900   1st Qu.: 935   1st Qu.: -5.100       1st Qu.:0.0000
 Median :1.500   Median :1690   Median :  4.700       Median :0.0100
 Mean   :1.726   Mean   :1434   Mean   :  3.945       Mean   :0.5679
 3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 15.200       3rd Qu.:0.9300
 Max.   :7.400   Max.   :2000   Max.   : 27.200       Max.   :3.5200
     RAINFALL        SNOWFALL      FUNCTIONING_DAY       Autumn
 Min.   : 0.0000   Min.   :0.00000   Length:8465        Min.   :0.0000
 1st Qu.: 0.0000   1st Qu.:0.00000   Class :character   1st Qu.:0.0000
 Median : 0.0000   Median :0.00000   Mode  :character   Median :0.0000
 Mean   : 0.1491   Mean   :0.07769                      Mean   :0.2288
 3rd Qu.: 0.0000   3rd Qu.:0.00000                      3rd Qu.:0.0000
 Max.   :35.0000   Max.   :8.80000                      Max.   :1.0000
```

```r
# remove reference link
remove_ref <- function(strings) {
    ref_pattern <- "\\[[A-z0-9]+\\]"
    # Replace all matched substrings with a white space using str_replace_all()
    string <- str_replace_all(strings, ref_pattern, " ")
    # Trim the result if you want
    string <- trimws(string)
    # return(result)
}
sub_bike_sharing_df$SYSTEM <- remove_ref(sub_bike_sharing_df$SYSTEM)
sub_bike_sharing_df$CITY <- remove_ref(sub_bike_sharing_df$CITY)
sub_bike_sharing_df$BICYCLES <- remove_ref(sub_bike_sharing_df$BICYCLES)
```

TODO: Use the `dplyr::mutate()` function to apply the `remove_ref` function to the `CITY` and `SYSTEM` columns

```r
# sub_bike_sharing_df %>% mutate(column1=remove_ref(column1), ... )
sub_bike_sharing_df %>%
 mutate(CITY=remove_ref(CITY), SYSTEM=remove_ref(SYSTEM), BICYCLES=remove_ref(BICYCLES))
```

A spec_tbl_df: 480 × 4

COUNTRY	CITY	SYSTEM	BICYCLES
<chr>	<chr>	<chr>	<chr>
Albania	Tirana	NA	200
Argentina	Mendoza	NA	40
Argentina	San Lorenzo, Santa Fe	Biciudad	80
Argentina	Buenos Aires	Serttel Brasil	4000
Argentina	Rosario	NA	480
Australia	Melbourne	PBSC & BD	676
Australia	Brisbane	3 Gen. Cyclocity	2000
Australia	Melbourne	4 Gen. oBike	1250
Australia	Sydney	4 Gen. oBike	1250
Australia	Sydney	4 Gen. Ofo	600
Australia	Sydney	Reddy Go	2000
Austria	Vienna	3 Gen. Cyclocity	1500
Austria	Burgenland	3 Gen. nextbike	NA

EDA with SQL

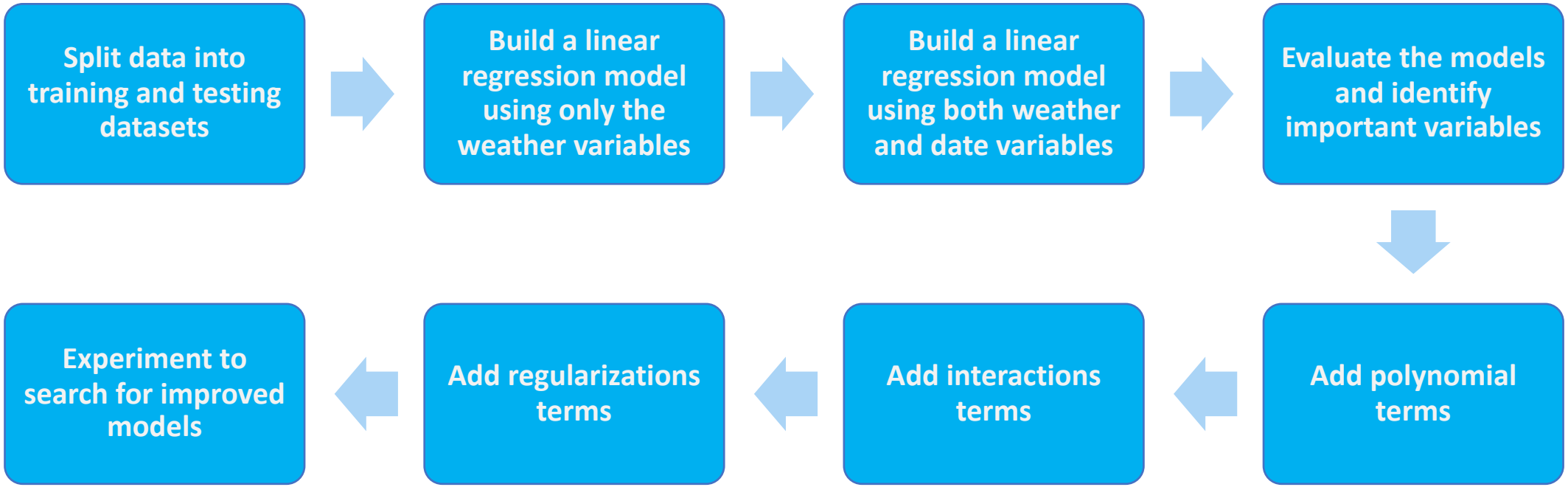
- Load the library RSQLite
- Download the saved csv files and load them into 4 tables.
- Use dbConnect() to create the SQLite Database and dbExecutive() to create the tables.
- Use dbWriteTable() to input data from csv into the table created.
- Use dbGetQuery() to query the database for insight from all the the table dataset
- And dbDisconnect(conn) to close the SQL

EDA with data visualization

- Create a scatter plot of RENTED_BIKE_COUNT vs DATE.
- Create a scatter plot of the RENTED_BIKE_COUNT time series with HOURS as the colour.
- Create a histogram overlaid with a kernel density curve for RENTED_BIKE_COUNT
- Use a scatter plot to visualize the correlation between RENTED_BIKE_COUNT and TEMPERATURE by SEASONS
- Create four boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS
- Create a line graph of rainfall and snowfall over the year

Predictive analysis

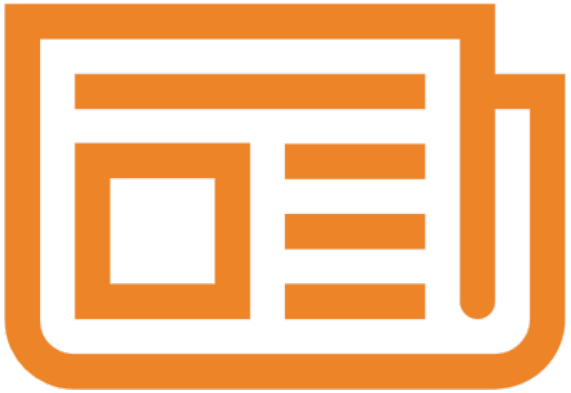
- Use tidymodels to build some baseline linear regression models.
- Improve linear regression model by adding polynomial and interaction terms, and regularization.



Build a R Shiny dashboard

- Build a R Shiny app with user interface (ui.R) main and side panel with leaflet to show the max bike-sharing demand for each city on an interactive map from the data wrangled. Use ggplot to render some detailed plots such as bike-sharing prediction trend, temperature trend, humidity and bike-sharing demand prediction correlation.

Results



- Exploratory data analysis results
- Predictive analysis results
- A dashboard demo in screenshots

EDA with SQL

exploratory data analysis using SQL

queries with the RSQLite R package

Busiest bike rental times

Date = 19/06/2018 Hour = 18

To get the date and hour with most bikes shared, the table is queried by selecting the date and hour where the the number or count of bike rented is at maximum.

```
# provide your solution here
dbGetQuery(conn, 'SELECT DATE, HOUR  FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT= (SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)')
```

A data.frame: 1 × 2

DATE	HOUR
<chr>	<int>
19/06/2018	18

Hourly popularity and temperature by seasons

Hour = 18, Temp = 29.38791, Summer season

The query for hour and temp grouped by season is ordered in descending order with most popular demand of bike at the top.

```
# provide your solution here
dbGetQuery(conn, 'SELECT HOUR, AVG(TEMPERATURE) AVG_TEMP, AVG(RENTED_BIKE_COUNT) AVG_RENTED_BIKE_COUNT, SEASONS FROM
GROUP BY SEASONS, HOUR
ORDER BY RENTED_BIKE_COUNT DESC
LIMIT 10')
```

A data.frame: 10 × 4

HOUR	AVG_TEMP	AVG_RENTED_BIKE_COUNT	SEASONS
<int>	<dbl>	<dbl>	<chr>
18	29.38791	2135.141	Summer
19	28.27378	1889.250	Summer
20	27.06630	1801.924	Summer
8	24.53587	1418.598	Summer
17	30.07691	1526.293	Summer
21	26.27826	1754.065	Summer
22	25.69891	1567.870	Summer
18	16.03185	1983.333	Autumn
19	15.06346	1515.568	Autumn
17	17.27778	1562.877	Autumn

Rental Seasonality

- The average hourly bike count during each season include the minimum, maximum, and standard deviation of the hourly bike count for each season.

```
# provide your solution here
dbGetQuery(conn, 'SELECT AVG(RENTED_BIKE_COUNT) AVG_RENTED_BIKE_COUNT, MIN(RENTED_BIKE_COUNT) MIN_RENTED_BIKE_COUNT,
MAX(RENTED_BIKE_COUNT) MAX_RENTED_BIKE_COUNT,
SQRT(AVG(RENTED_BIKE_COUNT*RENTED_BIKE_COUNT) - AVG(RENTED_BIKE_COUNT)*AVG(RENTED_BIKE_COUNT)) STANDARD_DEVIATION,
SEASONS
FROM SEOUL_BIKE_SHARING
GROUP BY SEASONS')
```

A data.frame: 4 × 5

AVG_RENTED_BIKE_COUNT	MIN_RENTED_BIKE_COUNT	MAX_RENTED_BIKE_COUNT	STANDARD_DEVIATION	SEASONS
<dbl>	<int>	<int>	<dbl>	<chr>
924.1105	2	3298	617.3885	Autumn
746.2542	2	3251	618.5247	Spring
1034.0734	9	3556	690.0884	Summer
225.5412	3	937	150.3374	Winter

Weather Seasonality

On average, what the TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, and SNOWFALL per season are and Including the average bike count to see if it correlated with the weather.

```
# provide your solution here
dbGetQuery(conn, 'SELECT AVG(TEMPERATURE) AVG_TEMP,
AVG(HUMIDITY) AVG_HUMIDITY,
AVG(WIND_SPEED) AVG_WIND_SPEED,
AVG(VISIBILITY) AVG_VISIBILITTY,
AVG(DEW_POINT_TEMPERATURE) AVG_DEW_POINT_TEMP,
AVG(SOLAR_RADIATION) AVG_S_RADIATION,
AVG(RAINFALL) AVG_RAINFALL,
AVG(SNOWFALL) AVG_SNOWFALL,
SEASONS,
AVG(RENTED_BIKE_COUNT)  AVG_RENTED_BIKE_COUNT  FROM
SEOUL_BIKE_SHARING
GROUP BY SEASONS
ORDER BY AVG_RENTED_BIKE_COUNT = (SELECT AVG(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)
')
```

TY	AVG_WIND_SPEED	AVG_VISIBILITTY	AVG_DEW_POINT_TEMP	AVG_S_RADIATION	AVG_RAINFALL	AVG_SNOWFALL	SEASONS	AVG_RENTED_BIKE_COUNT
bl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<dbl>
491	1.492101	1558.174	5.150594	0.5227827	0.11765617	0.06350026	Autumn	924.1105
333	1.857778	1240.912	4.091389	0.6803009	0.18694444	0.00000000	Spring	746.2542
143	1.609420	1501.745	18.750136	0.7612545	0.25348732	0.00000000	Summer	1034.0734
491	1.922685	1445.987	-12.416667	0.2981806	0.03282407	0.24750000	Winter	225.5412

Bike-sharing info in Seoul

Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes avaialble in Seoul, plus the following city information about Seoul: CITY, COUNTRY, LAT, LON, POPULATION, in a single view.

```
# provide your solution here
dbGetQuery(conn, 'SELECT C.CITY_ASCII CITY, C.COUNTRY, C.LAT, C.LNG, C.POPULATION, B.BICYCLES
FROM WORLD_CITIES C, BIKE_SHARING_SYSTEMS B
WHERE C.CITY_ASCII = B.CITY
AND C.CITY_ASCII = "Seoul"
')
```

A data.frame: 1 × 6

CITY	COUNTRY	LAT	LNG	POPULATION	BICYCLES
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>
Seoul	Korea, South	37.5833	127	21794000	20000

Cities similar to Seoul

Find all cities with total bike counts between 15000 and 20000. Return the city and country names, plus the coordinates (LAT, LNG), population, and number of bicycles for each city.

```
# provide your solution here
dbGetQuery(conn, 'SELECT C.CITY_ASCII CITY, C.COUNTRY, C.LAT, C.LNG, C.POPULATION, B.BICYCLES
FROM WORLD_CITIES C, BIKE_SHARING_SYSTEMS B
WHERE C.CITY_ASCII = B.CITY
AND B.BICYCLES BETWEEN 15000 AND 20000
')
```

A data.frame: 7 × 6

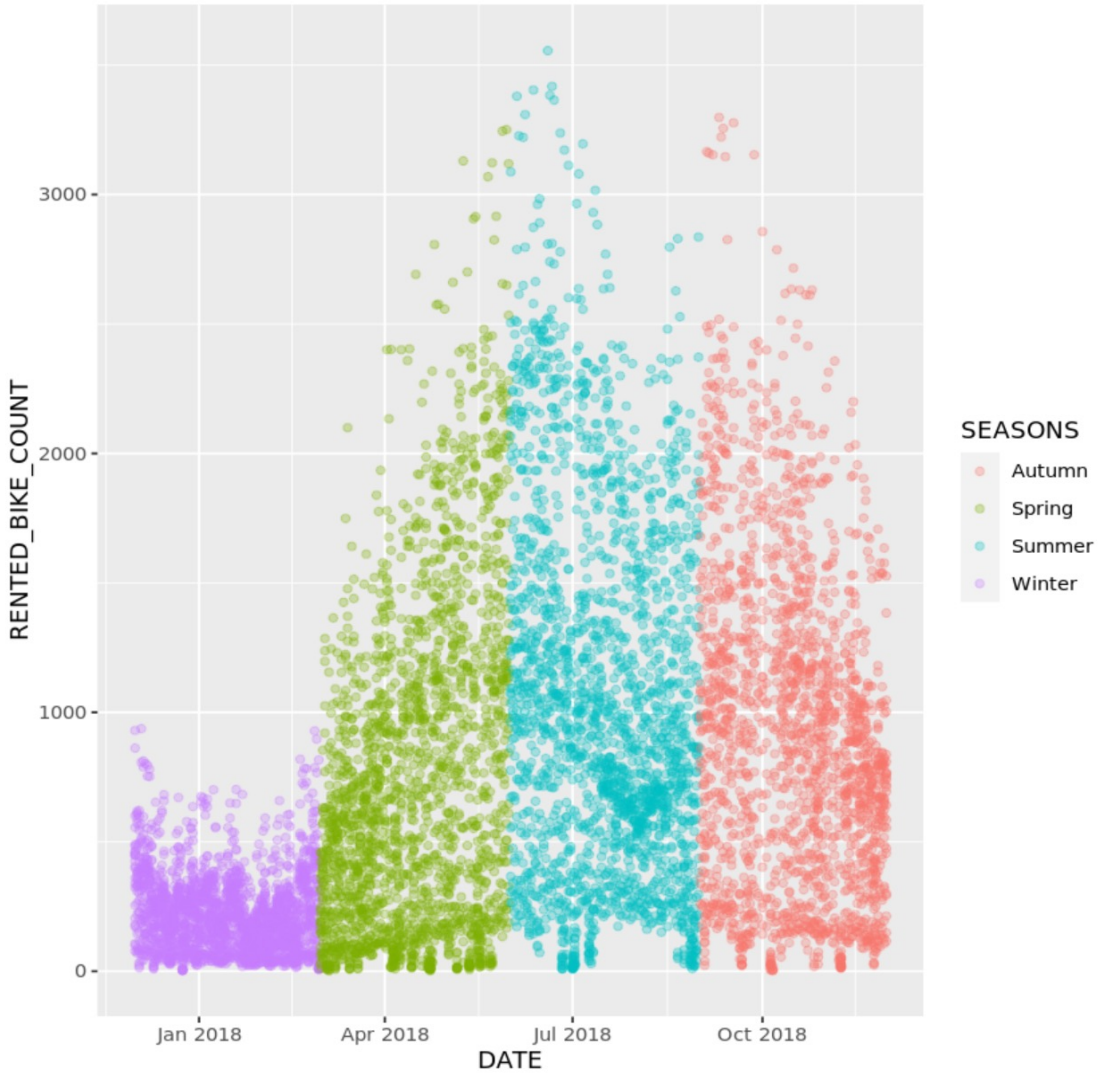
CITY	COUNTRY	LAT	LNG	POPULATION	BICYCLES
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>
Beijing	China	39.9050	116.3914	19433000	16000
Ningbo	China	29.8750	121.5492	7639000	15000
Shanghai	China	31.1667	121.4667	22120000	19165
Weifang	China	36.7167	119.1000	9373000	20000
Xi'an	China	34.2667	108.9000	7135000	20000
Zhuzhou	China	27.8407	113.1469	3855609	20000
Seoul	Korea, South	37.5833	127.0000	21794000	20000

EDA with Visualization

R notebook to perform exploratory data analysis using tidyverse and the ggplot2 R packages.

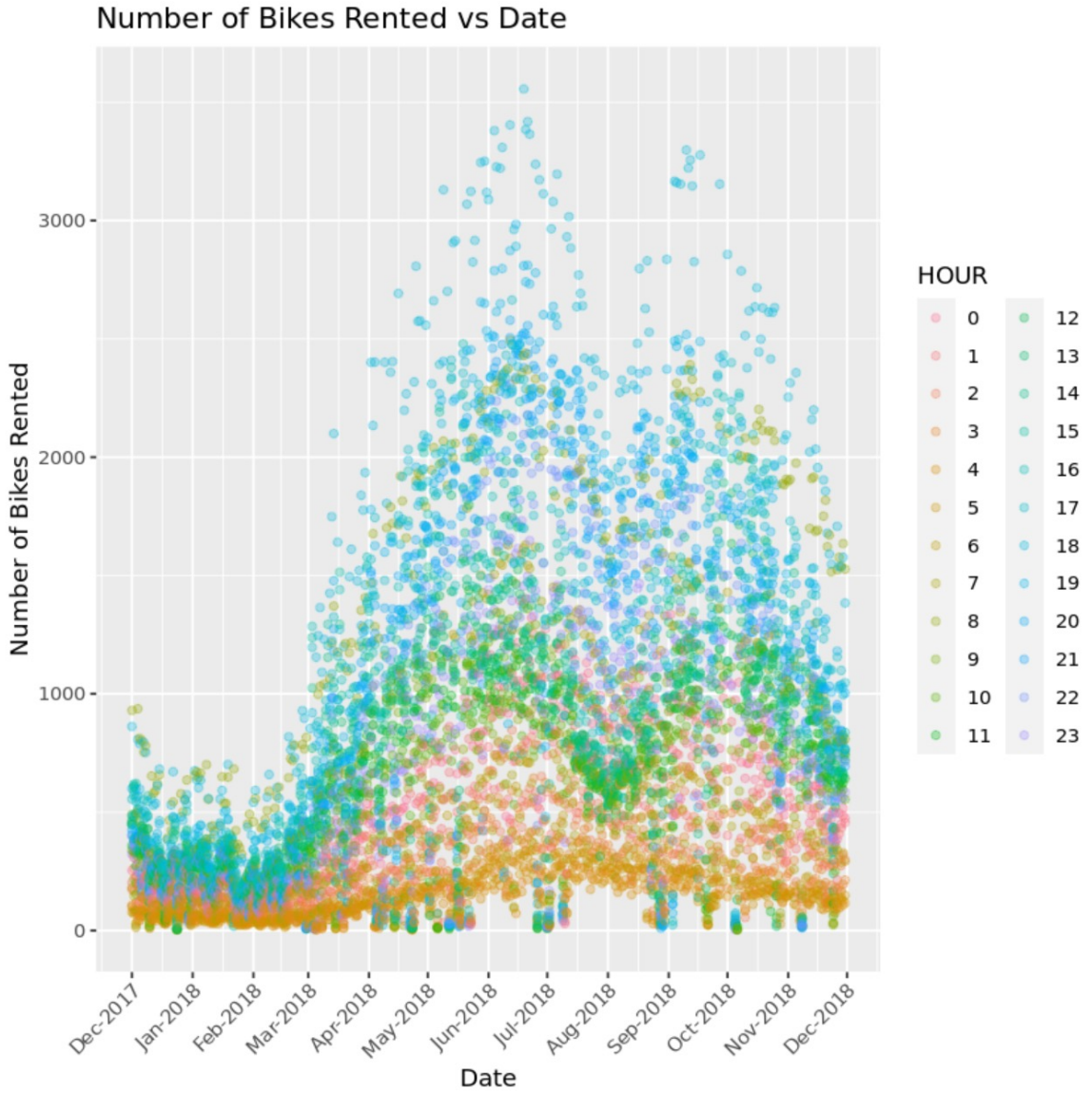
Bike rental vs. Date

bikes are rented the lowest in the winter, the demand begin to increase from spring and hit climax at summer and begin to decrease towards the end of Autumn.



Bike rental vs. Datetime

demand for bikes occurs more during the day, but if you care to hit the market fully, there are still thousands that demand at night the demand for bikes is more during the day than at night, as most humans get busier during the day.

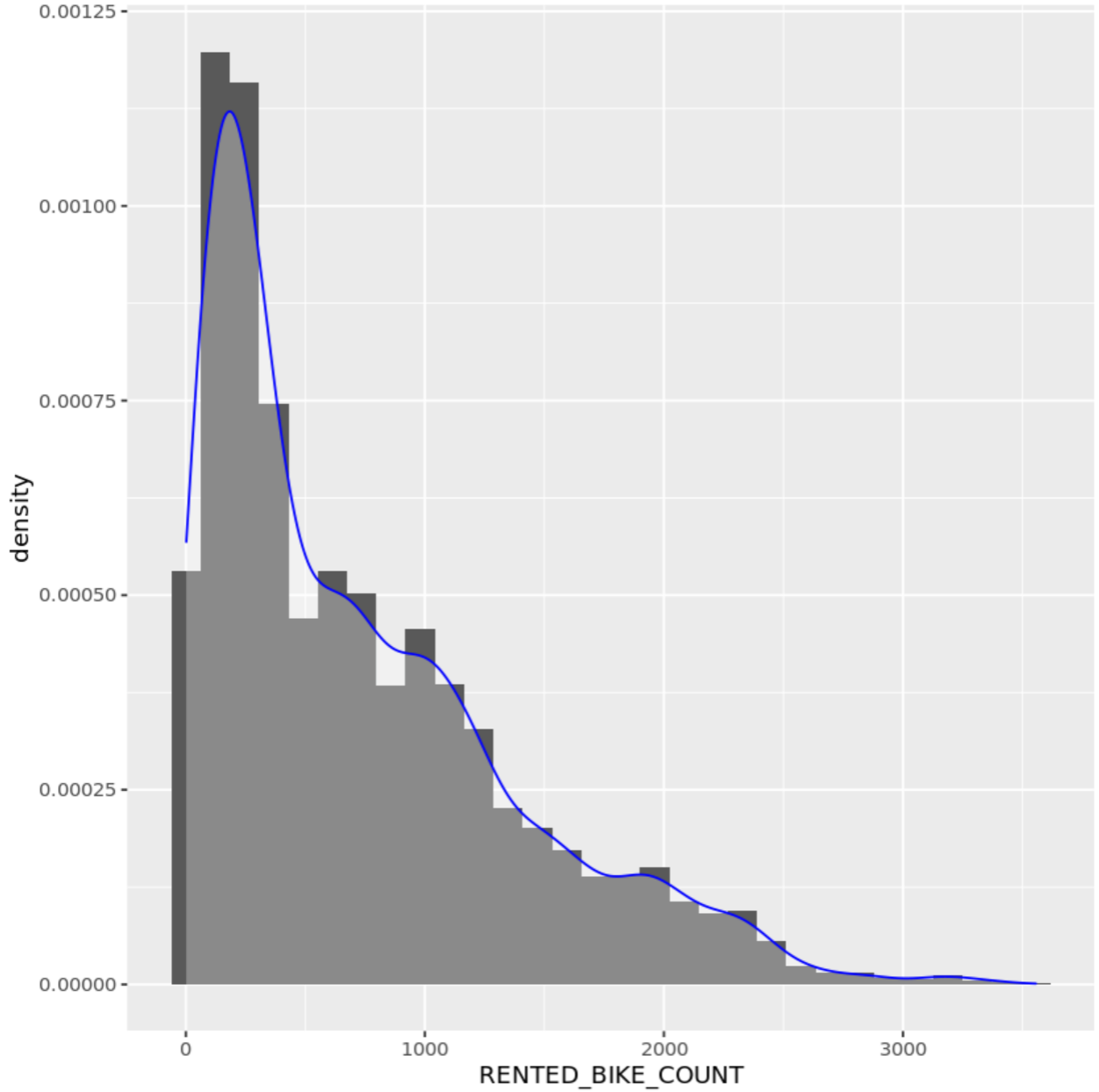


Bike rental histogram

We can see from the histogram that most of the time there are relatively few bikes rented. Indeed, the 'mode', or the most frequent number of bikes rented, is about 250.

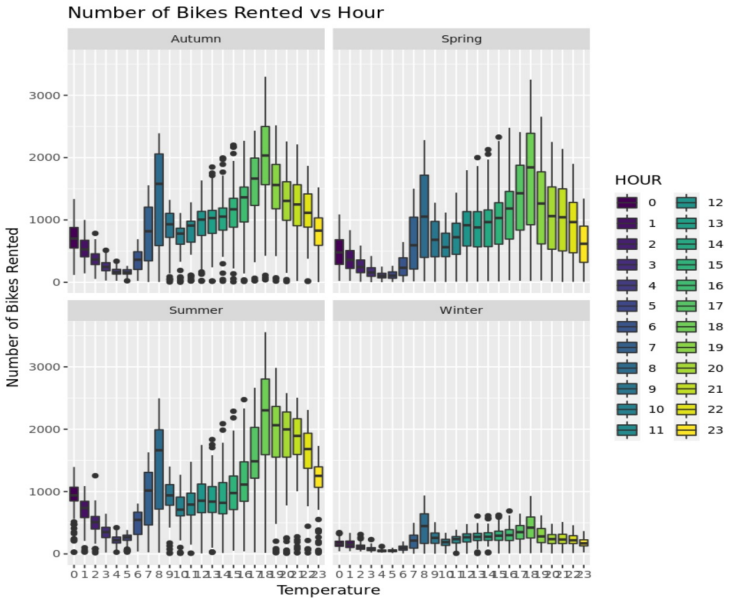
Judging by the ‘bumps’ at about 700, 900, 1900, and 3200 bikes, it looks like there may be other modes hiding within subgroups of the data.

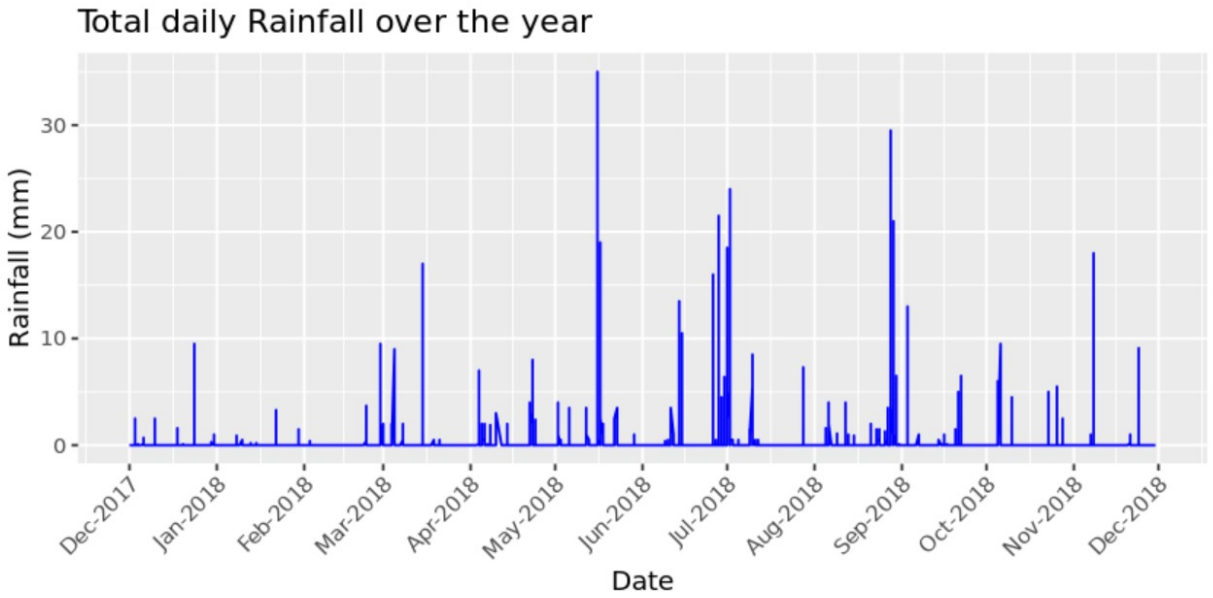
Interestingly, judging from the tail of the distribution, on rare occasions there are many more bikes rented out than usual.

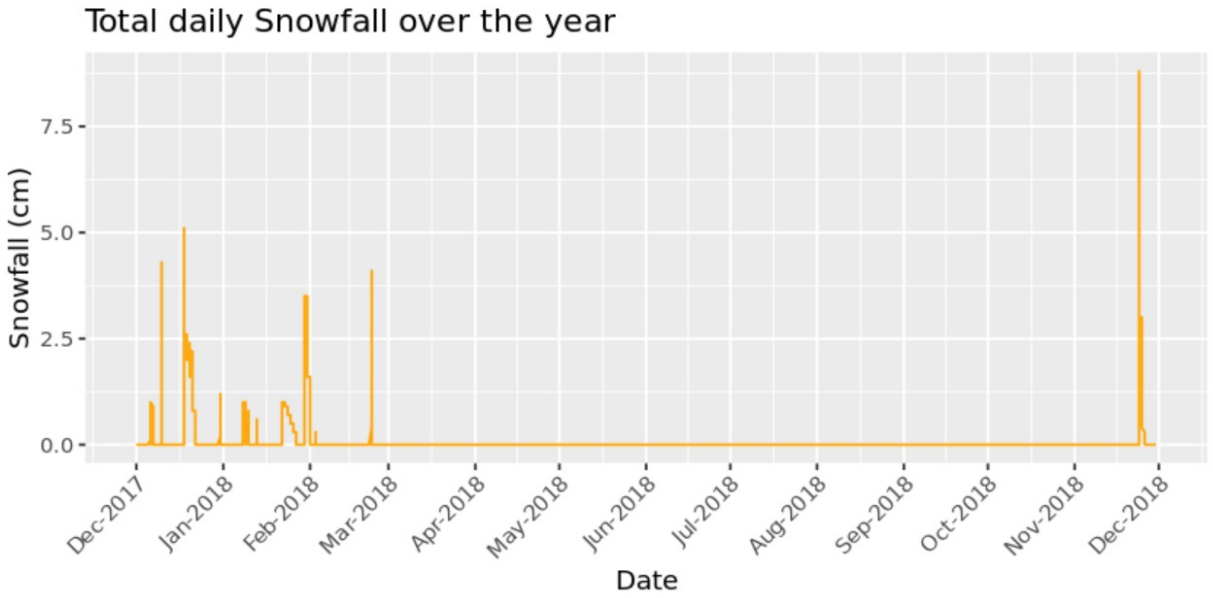


Daily total rainfall and snowfall

Snowfall/cold temperature has inverse effect on bike rental, bike rent hit climax during the hour of 18.







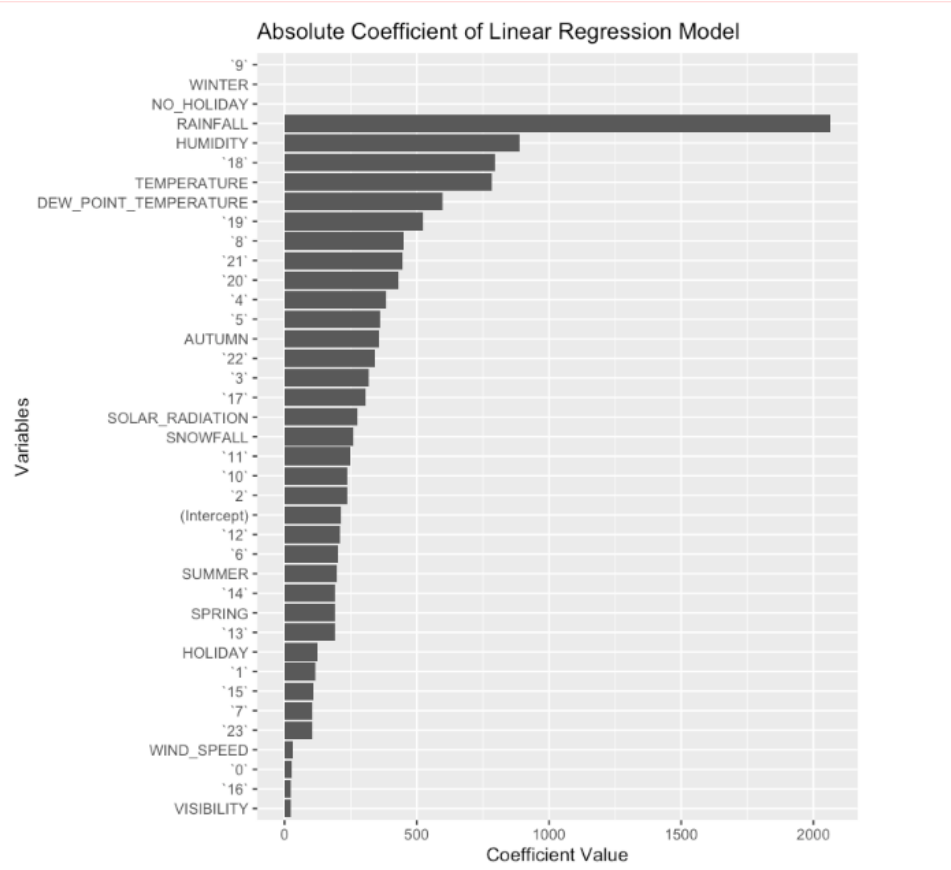
Predictive analysis

tidymodels

Ranked coefficients

Variables with larger coefficients in the model means they attribute more in the prediction of RENTED_BIKE_COUNT.

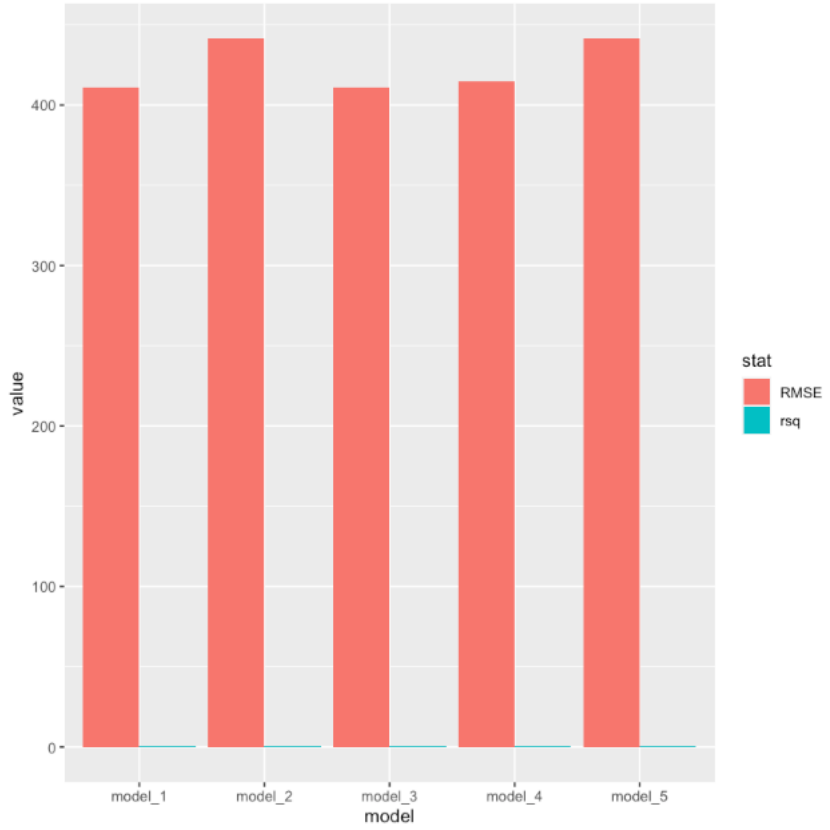
From the bar graph when trying to make a prediction app, attention will be paid to parameters that prominently influence bike sharing



Model evaluation

The lower value of RMSE implies higher accuracy of a regression model. However, a higher value of R square is considered desirable.

From the bar chart model_3 is most desirable.



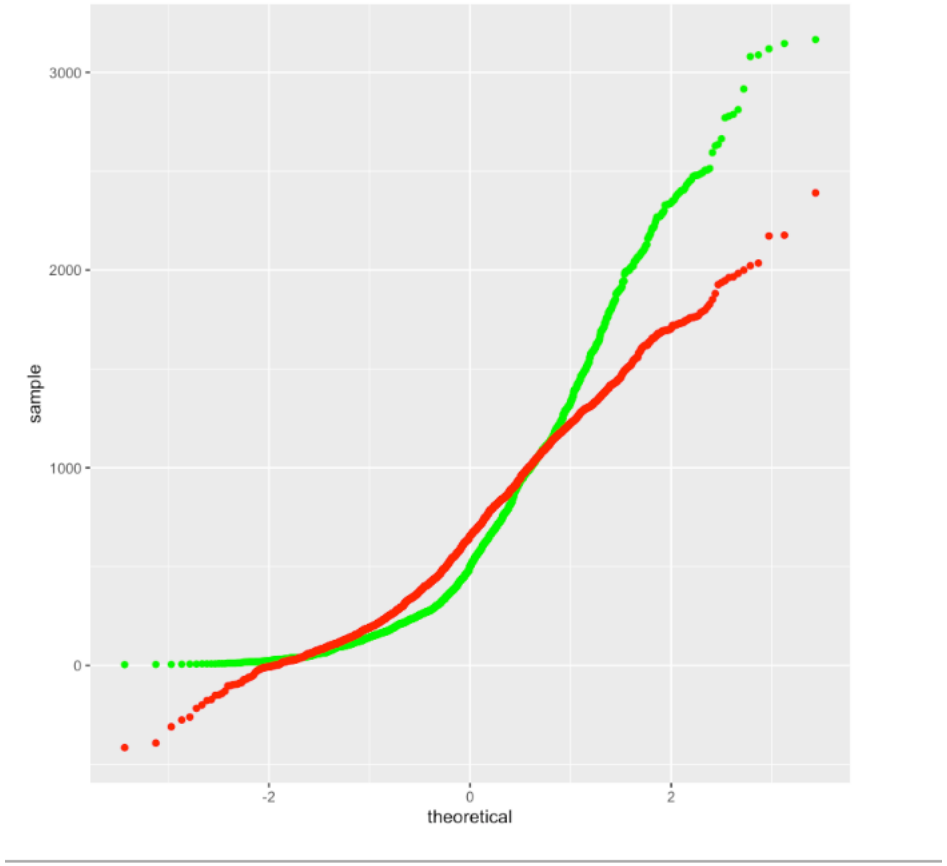
Find the best performing model

- Select the best performing model with:
  - RMSE must be less than 330
  - R-squared must be larger than 0.72
  - Shown a screenshot of the model performance

- Show its model formula here (RENTED_BIKE_COUNT ~ x1 + x2 + x3 ….)

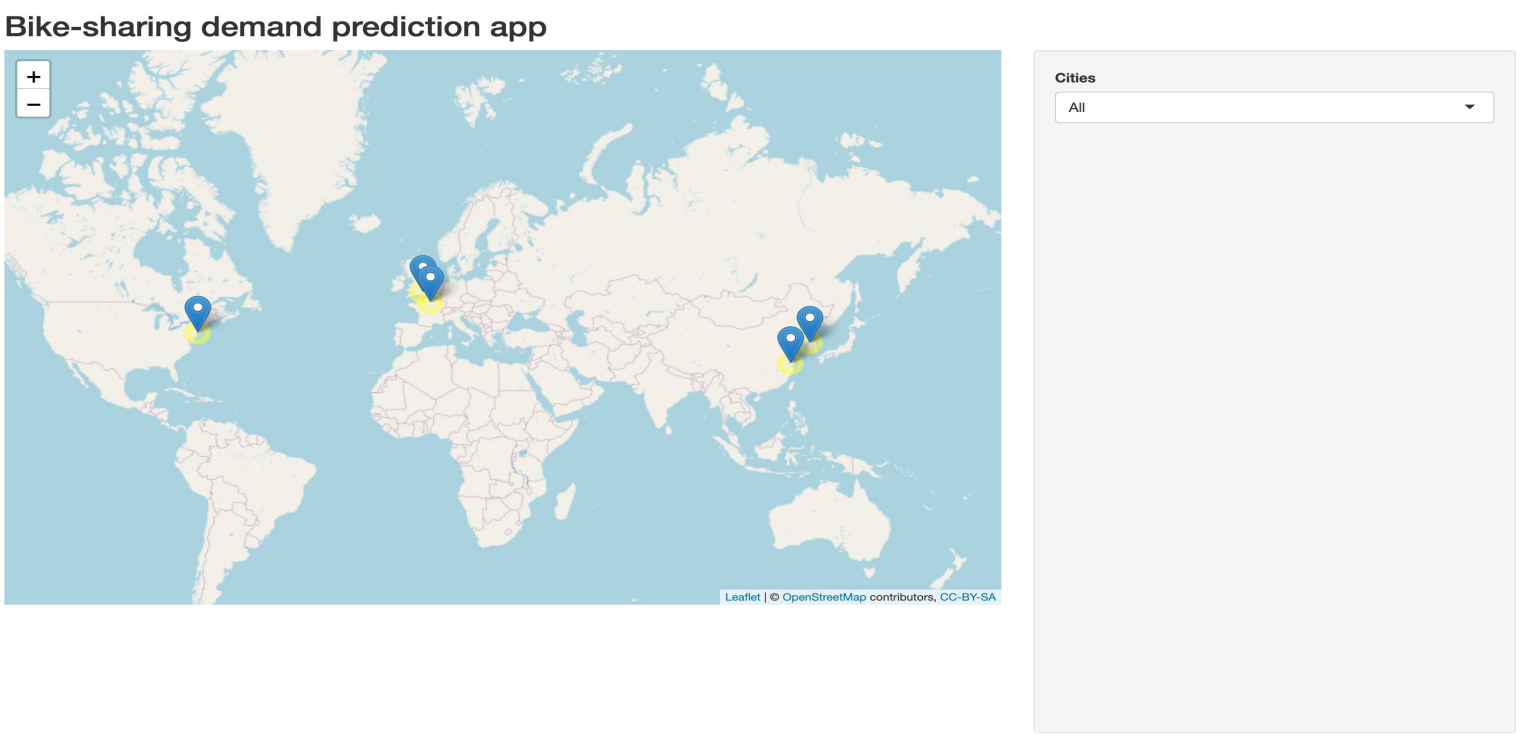
- You could optionally present their final coefficients here

Q-Q plot of the best model



Dashboard

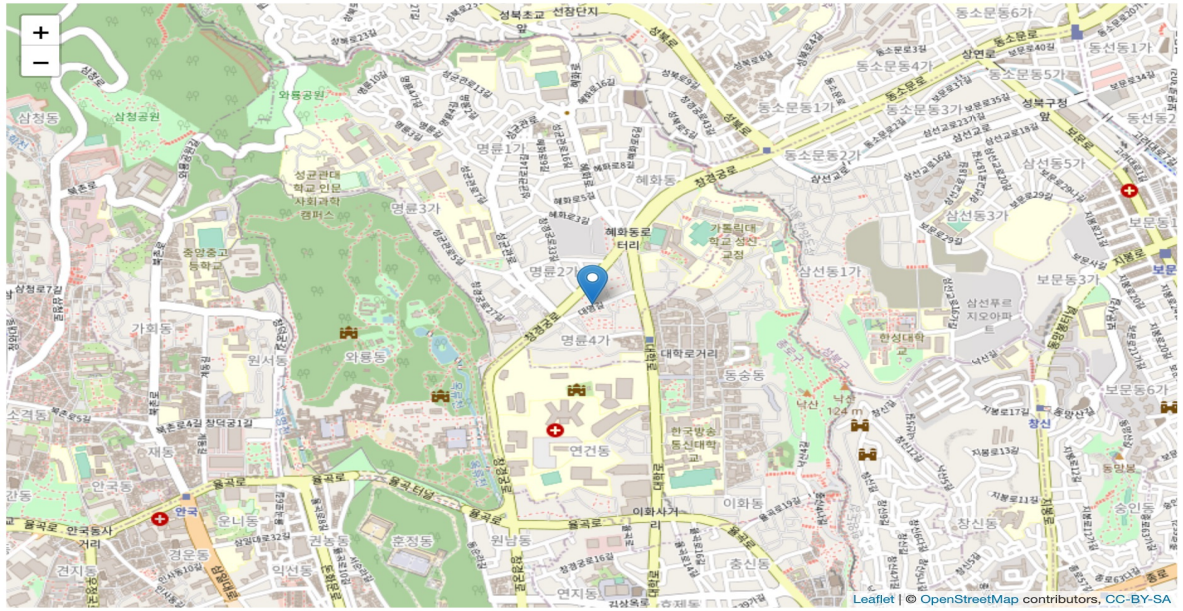
Dashboard for bike-sharing prediction on a map

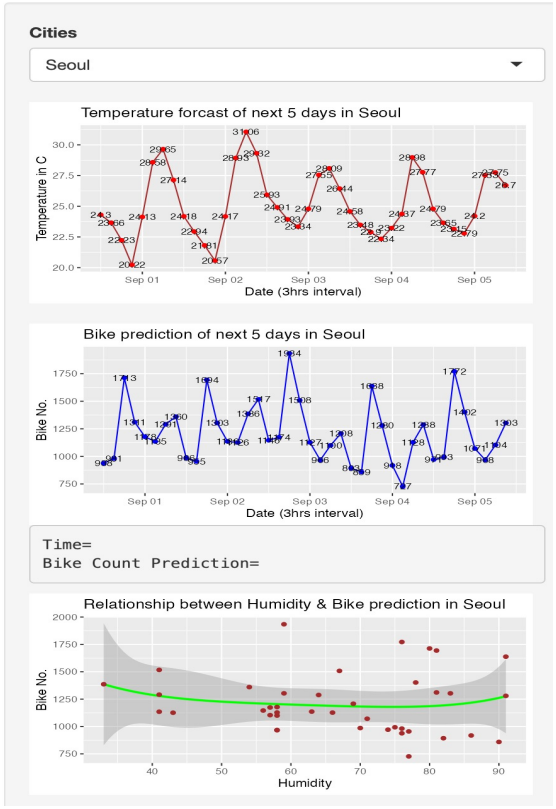


- Title panel is showing the title of the page, the design layout is sidebar layout with main panel as the map of cities and sidebar panel with selection drop down input for cities.

Dashboard for Seoul bike-sharing prediction

Bike-sharing demand prediction app

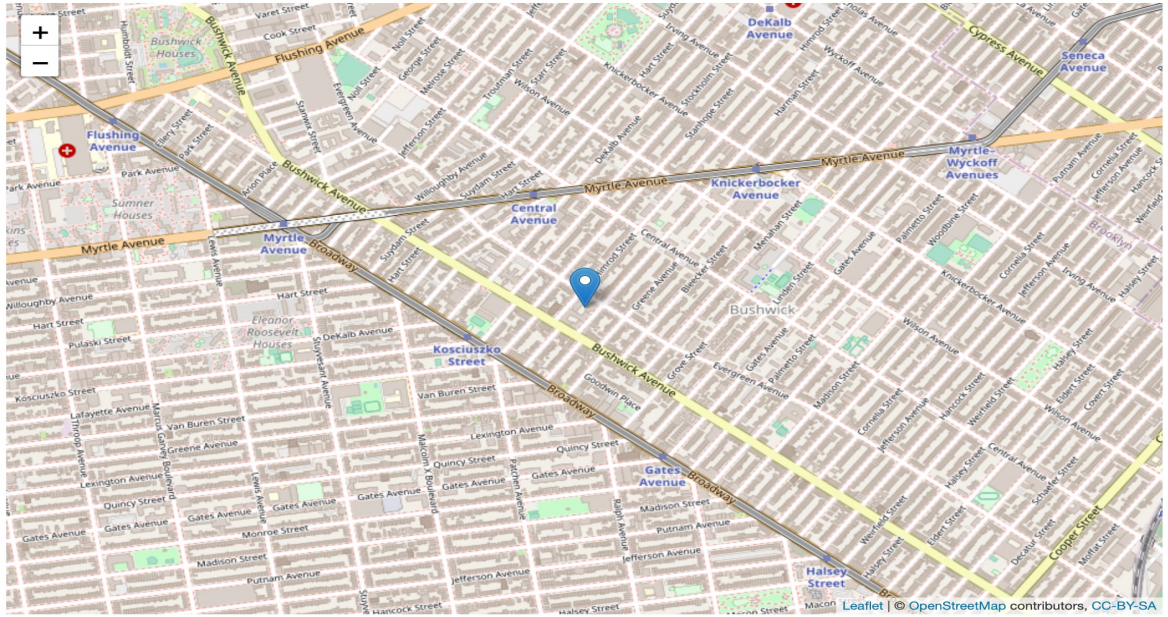


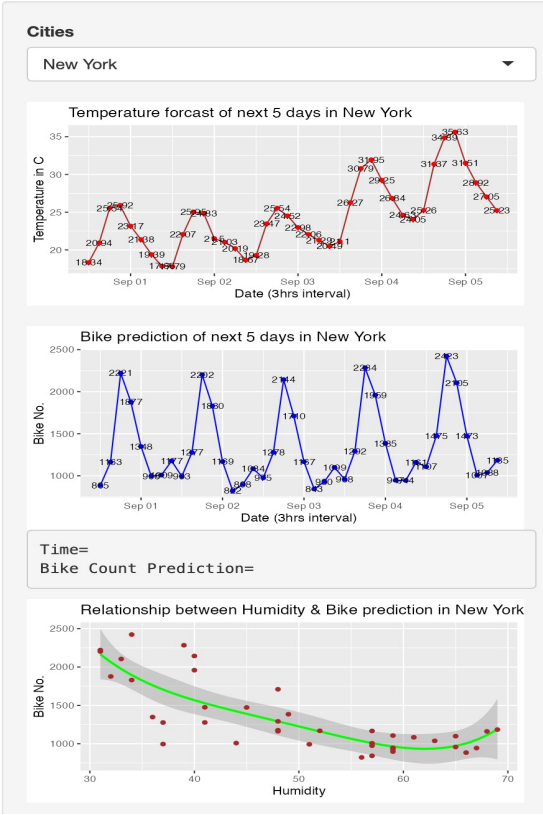


- On the map shows Seoul, on the side bar is the trend of temperature, humidity in correlation with bike demand over a 5-day forecast. Colder temperature means lesser demand.

Dashboard for New York bike-sharing prediction

Bike-sharing demand prediction app



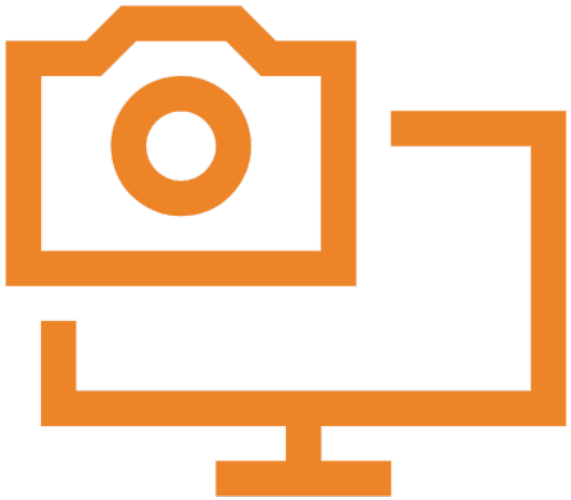


- On the map shows Seoul, on the side bar is the trend of temperature, humidity in correlation with bike demand over a 5-day forecast. Higher temperature means more demand.

CONCLUSION

- **Strategic Insights:** Understanding seasonality, weather, and timing improves resource allocation and user satisfaction.
- **Data-Informed Decisions:** Implement findings to enhance operational efficiency and user experience.
- **Future Direction:** Continued analysis can refine strategies and adapt to changing user patterns.

APPENDIX



```
world_cities_df <- dbExecute(conn, "CREATE TABLE WORLD_CITIES (
                    CITY VARCHAR(50),
                    CITY_ASCII VARCHAR(50),
                    LAT DECIMAL(20,2),
                    LNG DECIMAL(20,2),
                    COUNTRY VARCHAR(50),
                    ISO2 VARCHAR(5),
                    ISO3 VARCHAR(5),
                    ADMIN_NAME VARCHAR(100),
                    CAPITAL VARCHAR(50),
                    POPULATION BIGINT,
                    ID BIGINT NOT NULL
)", errors = FALSE)
# Create query for bike_sharing
bike_sharing_systems_df <- dbExecute(conn, "CREATE TABLE BIKE_SHARING_SYSTEMS (
                        COUNTRY VARCHAR(20),
                        CITY VARCHAR(87),
                        SYSTEM VARCHAR(40),
                        BICYCLES VARCHAR(5)
)", errors = FALSE)
# Create query for cities_weather_forecast
cities_weather_forecast_df <- dbExecute(conn, "CREATE TABLE CITIES_WEATHER_FORECAST (
    CITY VARCHAR(16),
    WEATHER VARCHAR(6),
    VISIBILITY SMALLINT,
    TEMP DECIMAL(6,2),
    TEMP_MIN DECIMAL(6,2),
    TEMP_MAX DECIMAL(6,2),
    PRESSURE SMALLINT,
    HUMIDITY SMALLINT,
    WIND_SPEED DECIMAL(6,2),
    WIND_DEG SMALLINT,
    SEASON VARCHAR(6),
    FORECAST_DATETIME TIMESTAMP
)", error = FALSE)
# Create query for seoul_bike_sharing
df4 <- dbExecute(conn, "CREATE TABLE SEOUL_BIKE_SHARING (
    DATE VARCHAR(30),
    RENTED_BIKE_COUNT SMALLINT,
    HOUR SMALLINT,
    TEMPERATURE DECIMAL(4,1),
    HUMIDITY SMALLINT,
    WIND_SPEED DECIMAL(3,1),
    VISIBILITY SMALLINT,
    DEW_POINT_TEMPERATURE DECIMAL(4,1),
    SOLAR_RADIATION DECIMAL(5,2),
    RAINFALL DECIMAL(3,1),
    SNOWFALL DECIMAL(3,1),
    SEASONS VARCHAR(10),
    HOLIDAY VARCHAR(20),
    FUNCTIONING_DAY VARCHAR(5)
)", error = FALSE)
```

```
world_cities_df <- read.csv("https://cf-courses-data.s3.us.cloud-object-storage.appdoma
bike_sharing_systems_df <- read.csv("https://cf-courses-data.s3.us.cloud-object-storage
cities_weather_forecast_df <- read.csv("https://cf-courses-data.s3.us.cloud-object-stor
seoul_bike_sharing_df <- read.csv("https://cf-courses-data.s3.us.cloud-object-storage.a
```

```
dbWriteTable(conn, "WORLD_CITIES", world_cities_df, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "BIKE_SHARING_SYSTEMS", bike_sharing_systems_df, overwrite=TRUE, hea
dbWriteTable(conn, "CITIES_WEATHER_FORECAST", cities_weather_forecast_df, overwrite=TRU
dbWriteTable(conn, "SEOUL_BIKE_SHARING", seoul_bike_sharing_df, overwrite=TRUE, header
```

```
sub_bike_sharing_df %>%
    select(BICYCLES) %>%
    filter(find_character(BICYCLES)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

BICYCLES
<chr>
4115[22]
310[59]
500[72]
[75]
180[76]
600[77]
[78]
initially 800 (later 2500)
100 (220)
370[114]

As you can see, many rows have non-numeric characters, such as `32 (including 6 rollers) [162]` and `1000[253]`. This is actually very common for a table scraped from Wiki when no input validation is enforced.

Later, you will use regular expressions to clean them up.

Next, let's take a look at the other columns, namely `COUNTRY`, `CITY`, and `SYSTEM`, to see if they contain any undesired reference links, such as in `Melbourne[12]`.

```
# Define a 'reference link' character class,
# `[A-z0-9]` means at least one character
# `\\[` and `\\]` means the character is wrapped by [], such as for [12] or [abc]
ref_pattern <- "\\[[A-z0-9]+\\]"
find_reference_pattern <- function(strings) grepl(ref_pattern, strings)
```

```
# Check whether the COUNTRY column has any reference links
sub_bike_sharing_df %>%
    select(COUNTRY) %>%
    filter(find_reference_pattern(COUNTRY)) %>%
    slice(0:10)
```

A spec_tbl_df: 0 × 1

COUNTRY
<chr>

Ok, looks like the `COUNTRY` column is clean. Let's check the `CITY` column.

```
# Check whether the CITY column has any reference links
sub_bike_sharing_df %>%
    select(CITY) %>%
    filter(find_reference_pattern(CITY)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

CITY
<chr>

```
str(seoul_bike_sharing)
```

```
'data.frame':   8465 obs. of  14 variables:
 $ DATE                 : Date, format: "2017-12-01" "2017-12-01" ...
 $ RENTED_BIKE_COUNT    : int  254 204 173 107 78 100 181 460 930 490 ...
 $ HOUR                 : Ord.factor w/ 24 levels "0"<"1"<"2"<"3"<..: 1 2 3 4 5 6 7 8
9 10 ...
 $ TEMPERATURE          : num  -5.2 -5.5 -6 -6.2 -6 -6.4 -6.6 -7.4 -7.6 -6.5 ...
 $ HUMIDITY             : int  37 38 39 40 36 37 35 38 37 27 ...
 $ WIND_SPEED           : num  2.2 0.8 1 0.9 2.3 1.5 1.3 0.9 1.1 0.5 ...
 $ VISIBILITY           : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 1928 ...
 $ DEW_POINT_TEMPERATURE: num  -17.6 -17.6 -17.7 -17.6 -18.6 -18.7 -19.5 -19.3 -19.8 -
22.4 ...
 $ SOLAR_RADIATION      : num  0 0 0 0 0 0 0 0 0.01 0.23 ...
 $ RAINFALL             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ SNOWFALL             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ SEASONS              : Factor w/ 4 levels "Autumn","Spring",..: 4 4 4 4 4 4 4 4 4 4
...
 $ HOLIDAY              : Factor w/ 2 levels "Holiday","No Holiday": 2 2 2 2 2 2 2 2 2
2 ...
 $ FUNCTIONING_DAY      : Factor w/ 1 level "Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Finally, ensure there are no missing values

```
sum(is.na(seoul_bike_sharing))
```

0

Descriptive Statistics

Now you are all set to take a look at some high level statistics of the `seoul_bike_sharing` dataset.

Task 4 - Dataset Summary

Use the base R `sumanry()` function to describe the `seoul_bike_sharing` dataset.

Solution 4

```
# provide your solution here
summary(seoul_bike_sharing)
```

```
       DATE            RENTED_BIKE_COUNT      HOUR       TEMPERATURE
 Min.   :2017-12-01   Min.   :   2.0    7      : 353   Min.   :-17.80
 1st Qu.:2018-02-27   1st Qu.: 214.0    8      : 353   1st Qu.:  3.00
 Median :2018-05-28   Median : 542.0    9      : 353   Median : 13.50
 Mean   :2018-05-28   Mean   : 729.2    10     : 353   Mean   : 12.77
 3rd Qu.:2018-08-24   3rd Qu.:1084.0    11     : 353   3rd Qu.: 22.70
 Max.   :2018-11-30   Max.   :3556.0    12     : 353   Max.   : 39.40
                                        (Other):6347
    HUMIDITY       WIND_SPEED      VISIBILITY   DEW_POINT_TEMPERATURE
 Min.   : 0.00   Min.   :0.000   Min.   :  27   Min.   :-30.600
 1st Qu.:42.00   1st Qu.:0.900   1st Qu.: 935   1st Qu.: -5.100
 Median :57.00   Median :1.500   Median :1690   Median :  4.700
 Mean   :58.15   Mean   :1.726   Mean   :1434   Mean   :  3.945
 3rd Qu.:74.00   3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 15.200
 Max.   :98.00   Max.   :7.400   Max.   :2000   Max.   : 27.200

 SOLAR_RADIATION     RAINFALL          SNOWFALL          SEASONS
 Min.   :0.0000   Min.   : 0.0000   Min.   :0.00000   Autumn:1937
 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.00000   Spring:2160
 Median :0.0100   Median : 0.0000   Median :0.00000   Summer:2208
 Mean   :0.5679   Mean   : 0.1491   Mean   :0.07769   Winter:2160
```

```
# provide your solution here
holiday_count <- table(seoul_bike_sharing$HOLIDAY)
Holidays <- holiday_count['Holiday']
Holidays
```

Holiday: 408

Task 6 - Calculate the percentage of records that fall on a holiday.

Solution 6