# Applied Data Science with R Capstone project

Joshua Agbroko

January 2024

# Outline

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

# Executive Summary

- This research tries to analyze how weather would affect bike sharing demand in urban areas. The dataset used in this analysis is from Seoul Bike Sharing System, Open Weather API, World Cities, and Bike System in the World
- We developed a predictive model to forecast bike demand under varying weather conditions by leveraging:
  - Historical weather
  - Bike Rental records from multiple cities
- By integrating weather parameters such as temperature, humidity, wind speed, and precipitation, our model provide insights into how weather influences bike usage patterns.

# Introduction

- Bike sharing systems are a type of bicycle rental service in which the procedure of obtaining a membership, renting a bike, and returning the bike is all done through a network of kiosks located around a city.
- People can rent a bike from one location and return it to a different location on an as-needed basis using these systems
- In urban environments, bike-sharing systems play a pivotal role in promoting sustainable transportation and addressing congestion challenges.
- However, optimizing bike-sharing services requires a deep understanding of factors influencing demand, particularly the impact of weather conditions.
- Challenges faced include;
  - Dynamic nature of demands
  - Weather sensitivity
  - Optimization needs

# Methodology

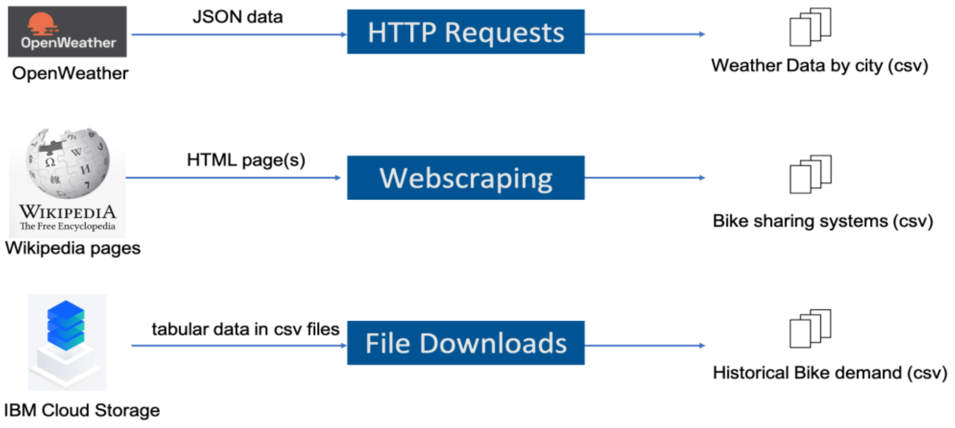
- Perform data collection
- Perform data wrangling
- Perform Exploratory Data Analysis (EDA) using SQL and visualization
- Perform predictive analysis using regression models
  - How to build the baseline model
  - How to improve the baseline model
- Build a R Shiny dashboard app

# Methodology

# Data collection

- Weather Data was Retrieved from  OpenWeatherAPI Using OpenWeather REST API to request weather data.

- Web scraping information about global bike-sharing systems from Wikipedia web pages, and downloading and aggregating tabular data from cloud storage.



# Data wrangling

- Many related inconsistencies and noises like inconsistent data formats, extracted fields containing HTML tags, reference links were eliminated using regular expressions.
- Detected and handled missing values that may affect the model’s predictive ability.
- Categorical variables were converted into indicator variables
- Numeric columns were normalized to transfer them all into a similar range
- Add the screenshots of data wrangling code cell and output for regular expressions, missing values handling, generating indicator columns to the Appendix section for peer-review

# EDA with SQL

- Performed SQL queries using RSQLite to :
  - Find valuable statistics such as total bike-sharing records or operational hours,
  - Filter data based on city or date,
  - Find patterns such as bike-sharing seasonality or similarities among bike-sharing systems across the world.

# EDA with data visualization

- Understanding the distributions of the data using histograms
- Finding correlations between important features using scatterplots
- Spotting outliers and irregular behavior in your features using box plots.

# Predictive analysis

- Embarked on building a predictive model to forecast bike-sharing demand, a crucial task for urban transportation planning and resource allocation. Began by collecting extensive weather data from the OpenWeatherAPI and integrating it with historical bike usage records from various cities. Leveraging a diverse set of machine learning algorithms including linear regression, random forest, gradient boosting, we constructed initial models to predict bike demand. These models were evaluated using key metrics such as Root Mean Squared Error (RMSE) and R-squared to gauge their accuracy and performance.

# Build a R Shiny dashboard

- Integrated interactive leaflet maps displaying cities' bike-sharing demand predictions with color-coded markers indicating predicted demand levels.
- Implemented dropdown menu functionality allowing users to select specific cities for detailed analysis.
- Incorporated line plots depicting temperature forecasts for the next five days, aiding in understanding weather patterns' influence on bike demand.
- Developed line plots illustrating bike-sharing demand predictions over time, facilitating trend analysis and forecasting assessment.

# Results

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

# EDA with SQL

# Busiest bike rental times

- DATE HOUR total_rentals

19/06/2018 18 3556

- This query result indicates that June 19, 2018 6:00 PM, had the most bike rentals recorded in the dataset.

# Hourly popularity and temperature by seasons

| SEASONS | avg_hourly_temperature | avg_bike_rentals_per_hour |
|---|---|---|
| Summer | 26.587710514337775 | 1034.0733695652175 |
| Autumn | 13.82157976251933 | 924.1104801239029 |
| Spring | 13.021685185185188 | 746.2541666666667 |
| Winter | -2.5404629629629607 | 225.5412037037037 |

- This analysis highlights the dynamic relationship between seasonal weather patterns, temperature, and bike rental activity, underscoring the need for data-driven decision-making in the bike-sharing industry

# Rental Seasonality

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| SEASONS | avg_hourly_bike_count | min_hourly_bike_count | max_hourly_bike_count | std_dev_hourly_bike_count |
|---|---|---|---|---|
| Winter | 225.5412037037037 | 3 | 937 | 150.3374234675032 |
| Spring | 746.2541666666667 | 2 | 3251 | 618.5247350747628 |
| Summer | 1034.0733695652175 | 9 | 3556 | 690.0884362677946 |
| Autumn | 924.1104801239029 | 2 | 3298 | 617.3884505672105 |

These observations provide insights into seasonal bike usage patterns, which can be valuable for bike-sharing system optimization. The observations underline the fact that far less bikes are rented during Winter.

# Weather Seasonality

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| SEASONS | avg_temperature | avg_humidity | avg_wind_speed | avg_visibility | avg_dew_point_temperature | avg_solar_radiation | avg_rainfall | avg_snowfall | avg_bike_count |
|---|---|---|---|---|---|---|---|---|---|
| Summer | 26.587710514337775 | 64.9814311594203 | 1.6094202898550731 | 1501.7454710144928 | 18.75013586956522 | 0.7612545289855057 | 0.2534873188405798 | 0 | 1034.0733695652175 |
| Autumn | 13.82157976251933 | 59.044914816726894 | 1.4921011874032002 | 1558.1744966442952 | 5.150593701600414 | 0.5227826535880221 | 0.11765616933402169 | 0.06350025813113062 | 924.1104801239029 |
| Spring | 13.021685185185188 | 58.75833333333333 | 1.8577777777777793 | 1240.911574074074 | 4.091388888888886 | 0.6803009259259252 | 0.1869444444444444 | 0 | 746.2541666666667 |
| Winter | -2.5404629629629607 | 49.74490740740741 | 1.9226851851851832 | 1445.987037037037 | -12.41666666666665 | 0.2981805555555558 | 0.03282407407407407 5 | 0.24749999999999997 | 225.5412037037037 |

Based on these observations, we can deduce that weather conditions, particularly temperature and solar radiation, may influence bike-sharing usage, with higher usage observed during warmer and sunnier seasons. Additionally, factors such as humidity, visibility, and precipitation may also impact bike-sharing patterns, albeit to a lesser extent.

# Bike-sharing info in Seoul

| | CITY | COUNTRY | LAT | LON | POPULATION | total_bikes_available |
|---|---|---|---|---|---|---|
| 1 | Seoul | Korea, South | 37.5833 | 127 | 21794000 | 20000 |

- This query result provides insight into the population size and availability of bicycles for shared use in Seoul.

# Cities similar to Seoul

```
print(result)
```

| | CITY | COUNTRY | LAT | LNG | POPULATION | total_bikes |
|---|---|---|---|---|---|---|
| 1 | Beijing | China | 39.9050 | 116.3914 | 19433000 | 16000 |
| 2 | Ningbo | China | 29.8750 | 121.5492 | 7639000 | 15000 |
| 3 | Shanghai | China | 31.1667 | 121.4667 | 22120000 | 19165 |
| 4 | Weifang | China | 36.7167 | 119.1000 | 9373000 | 20000 |
| 5 | Xi’an | China | 34.2667 | 108.9000 | 7135000 | 20000 |
| 6 | Zhuzhou | China | 27.8407 | 113.1469 | 3855609 | 20000 |
| 7 | Seoul | Korea, South | 37.5833 | 127.0000 | 21794000 | 20000 |

```
close(conn)
```

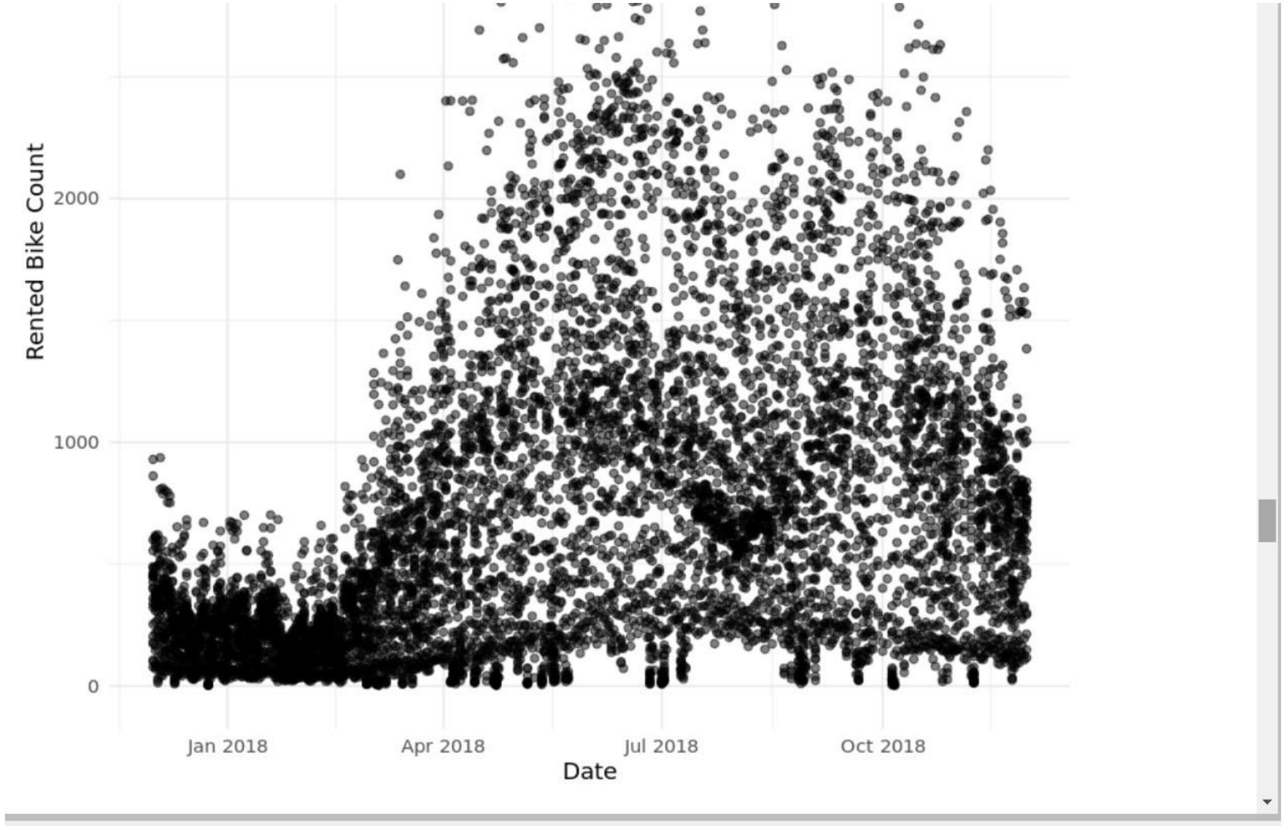
We can infer that these cities, especially Beijing and Shanghai likely have active bike-sharing systems or initiatives due to their large populations and significant numbers of available bicycles just like Seoul.

# EDA with Visualization

# Bike rental vs. Date

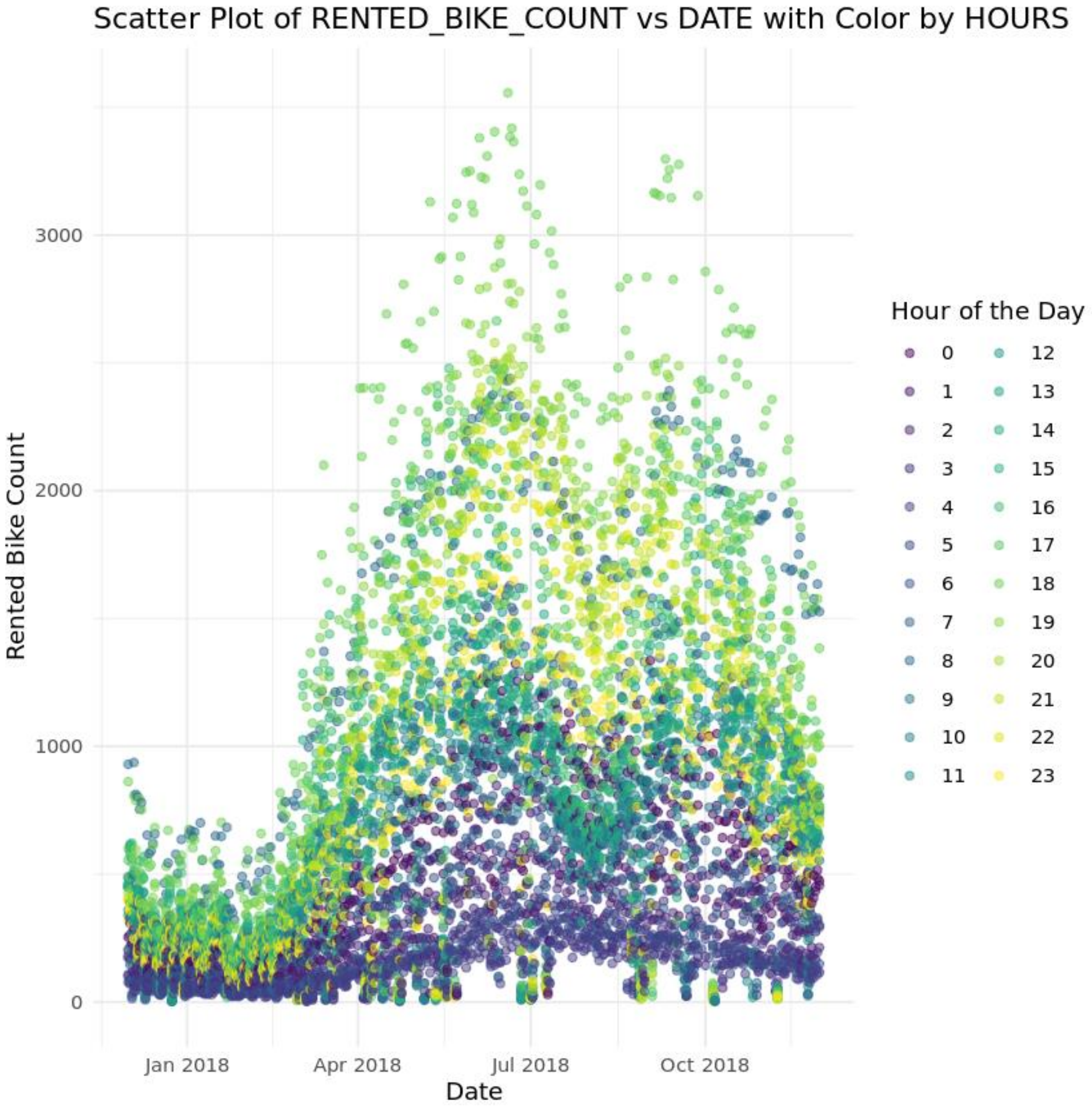
Scatter plot
of RENTED_BIKE_COUNT vs. DATE

Noticeably few bike rentals during the winter months. Sharp increase during the following Spring months which has the highest rentals. Summer and Autumn also have high number of bike rentals.



# Bike rental vs. Datetime

Afternoon to late evening has the highest number of bike rentals with nighttime as expected, having the lowest numbers

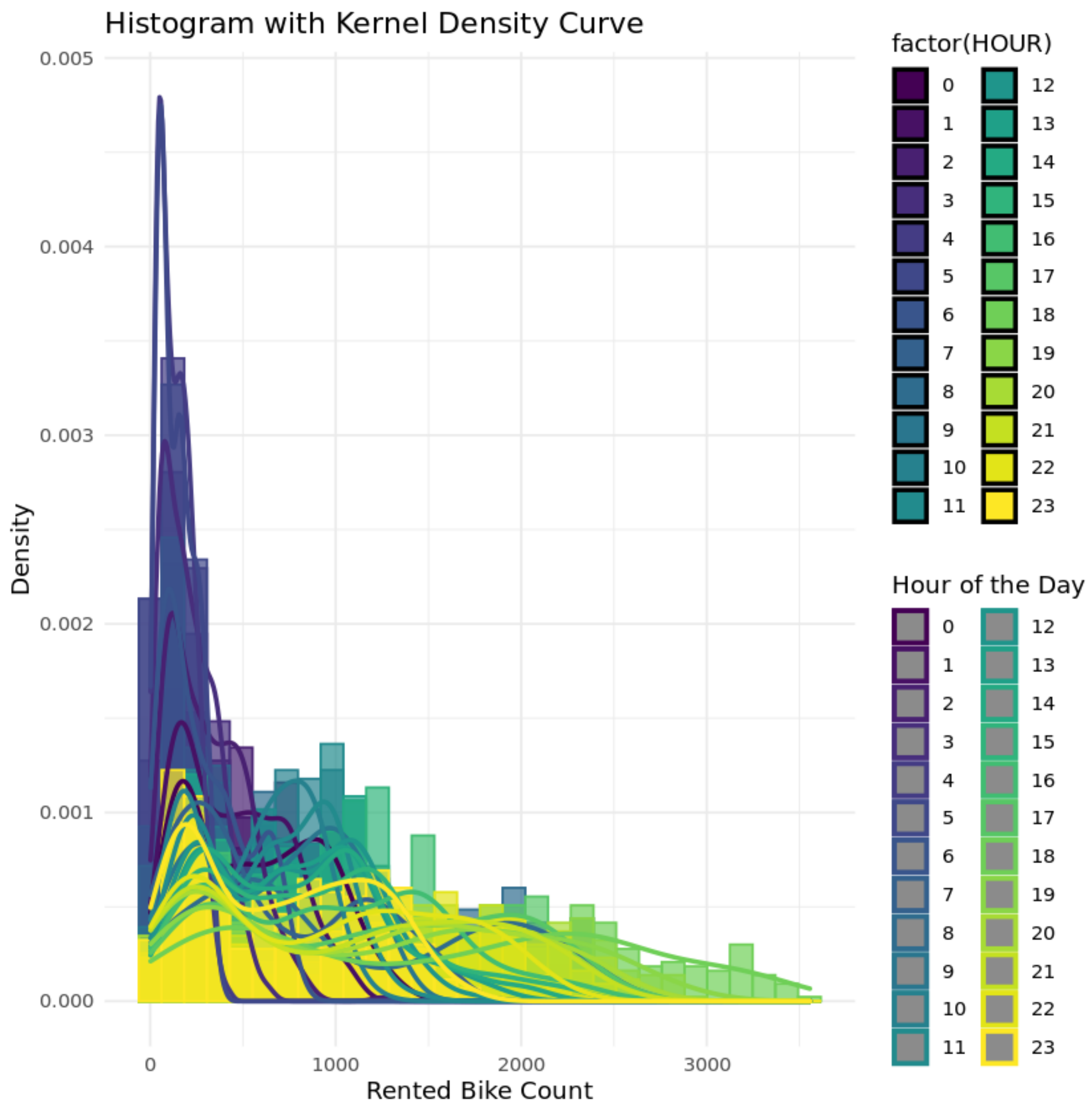


# Bike rental histogram

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

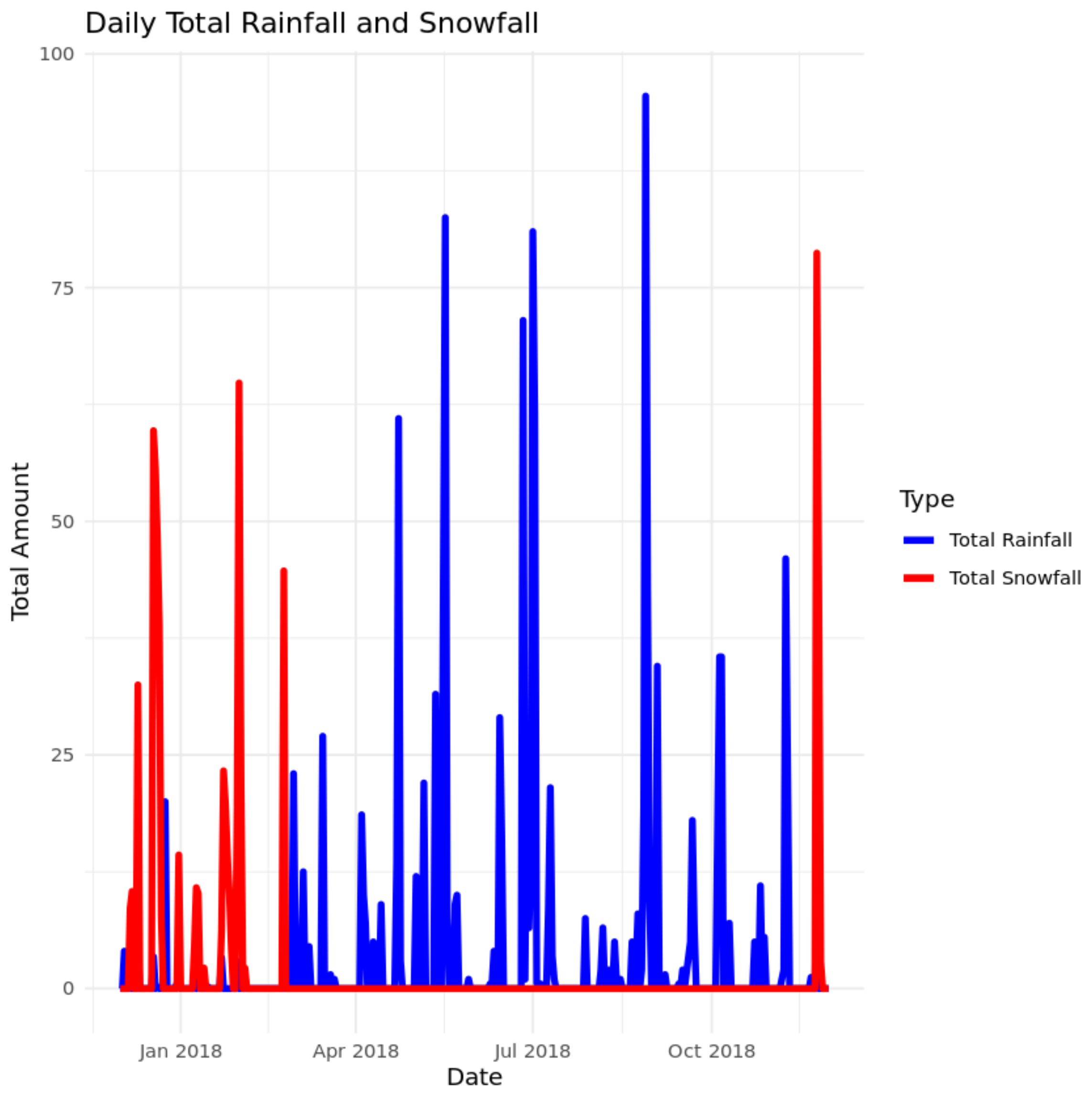
Judging by the 'bumps' at about 700, 900, and 1900, and 3200 bikes, it looks like there may be other modes hiding within subgroups of the data.

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

A barchart calculating the daily total rainfall and snowfall

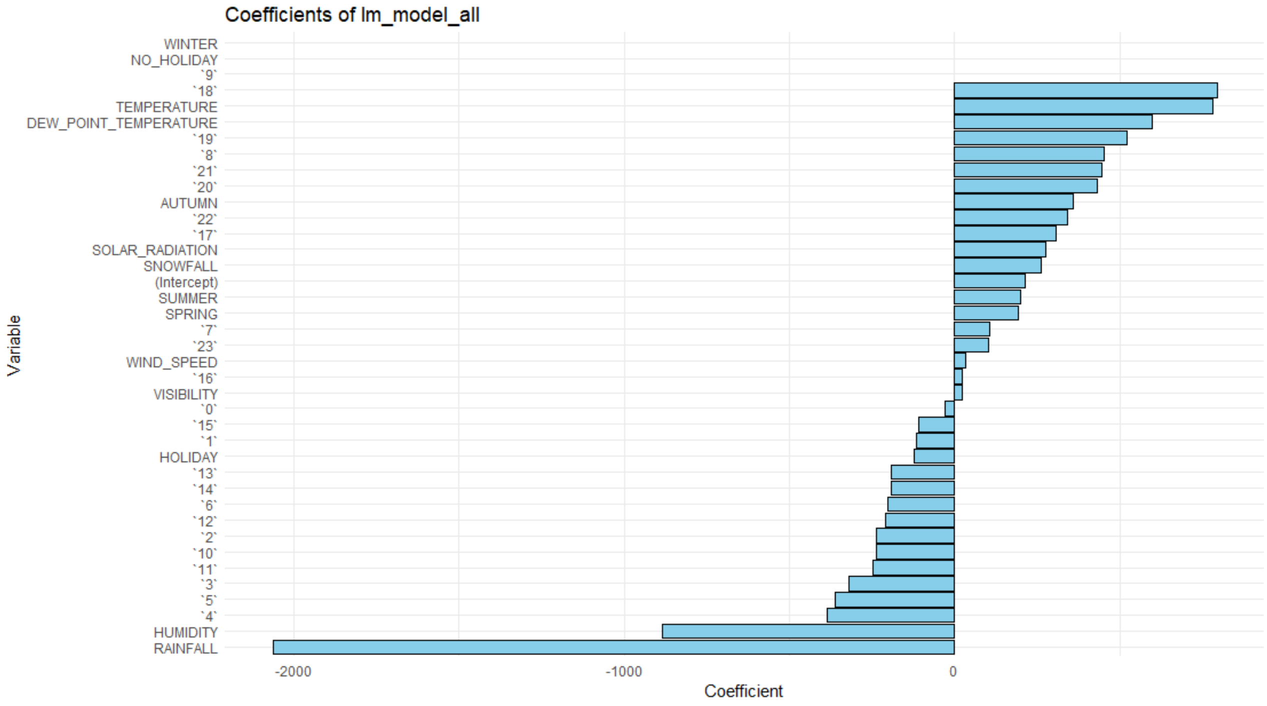


# Predictive analysis

# Ranked coefficients

From the coefficients of the model, we can find that Most of the highest predictors are weather variables such as Rainfall, Humidity, Temperature, and Dew Point Temperature. It means weather condition could influence people decision to rent bikes.

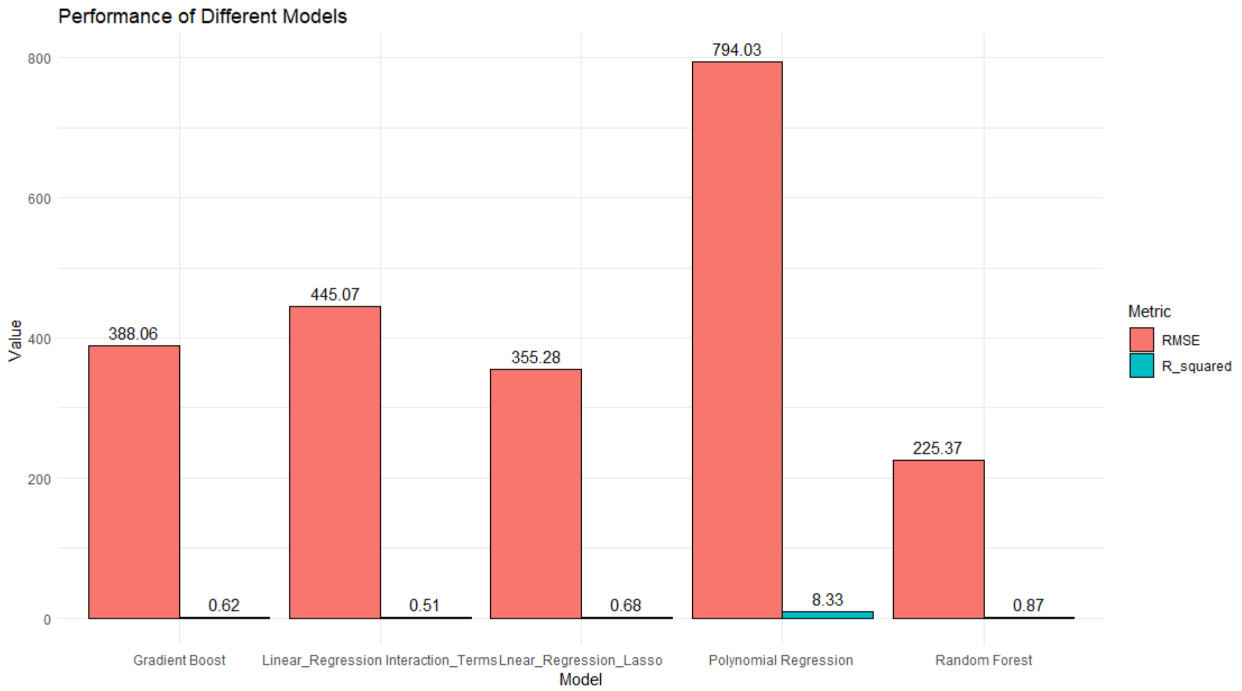
Another significant indicator is most evening time is highly correlated with higher number of bike rents



# Model evaluation

- Here is the result of RMSE and RSquared of each models created for the estimation

- Visualized are the refined models' RMSE and R-squared using grouped bar chart



# Find the best performing model

```
> rmse_rf
[1] 225.3694
> rsquared_rf
[1] 0.873195
> |
```

- The predictors <- model.matrix(RENTED_BIKE_COUNT ~

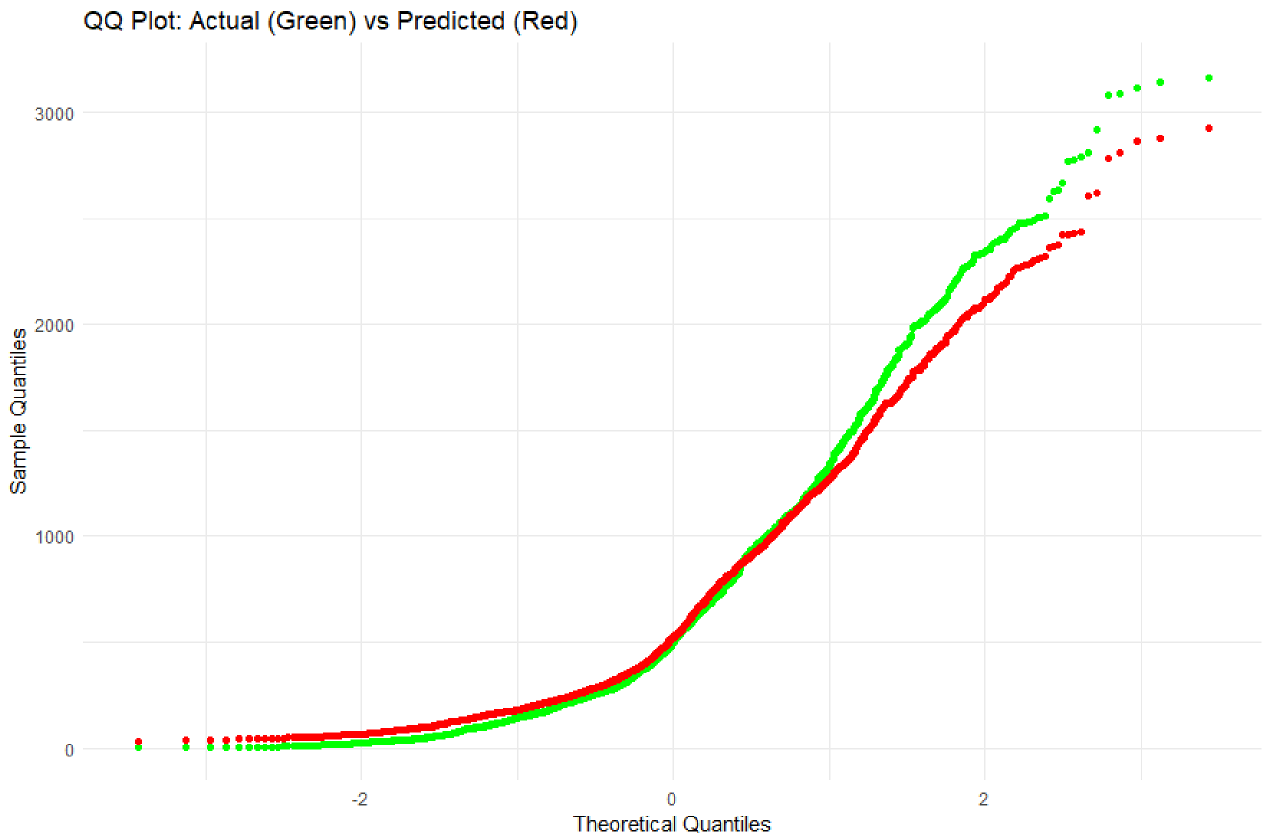
  TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL + SNOWFALL, data = train_data1)[, -1]

- response <- train_data$RENTED_BIKE_COUNT

- rf_model <- randomForest(predictors, response)

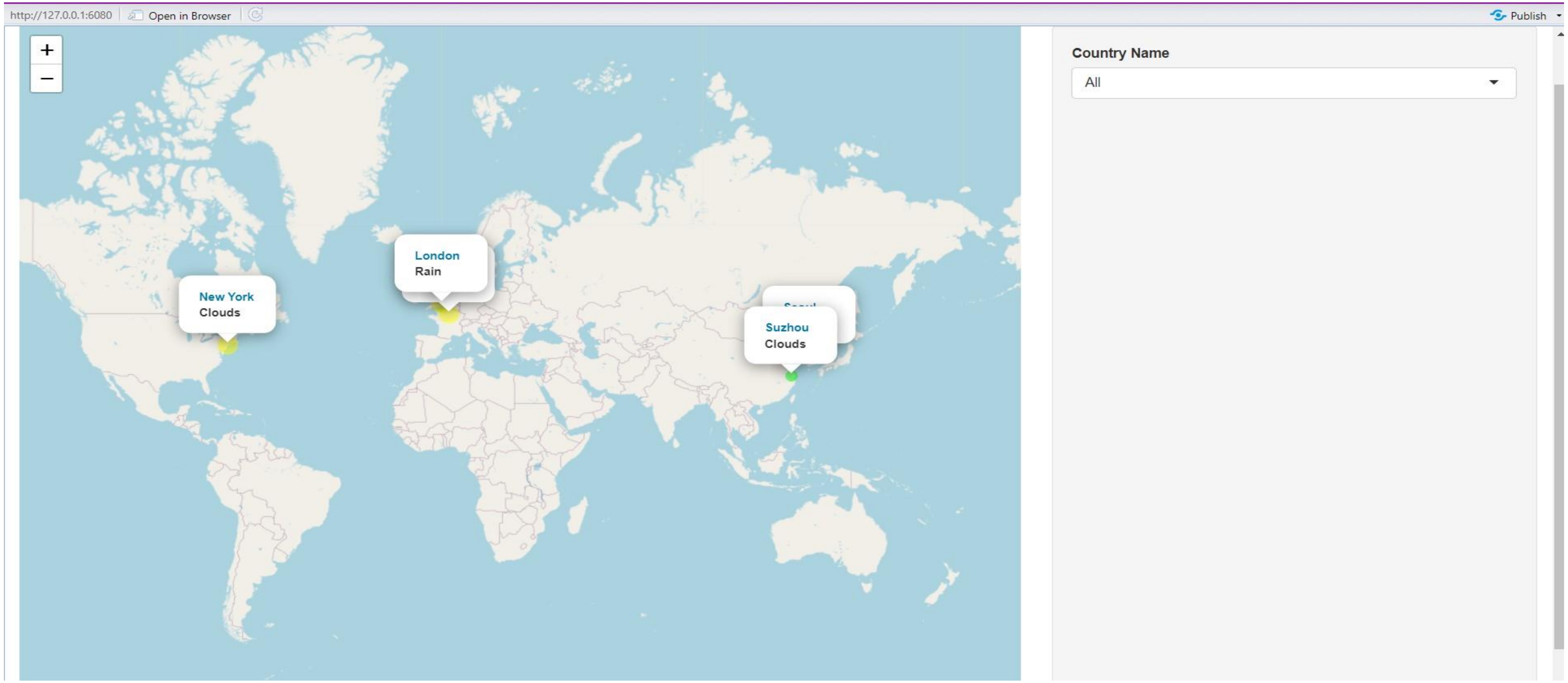
# Q-Q plot of the best model

the Q-Q plot of the best model’s test results vs the truths



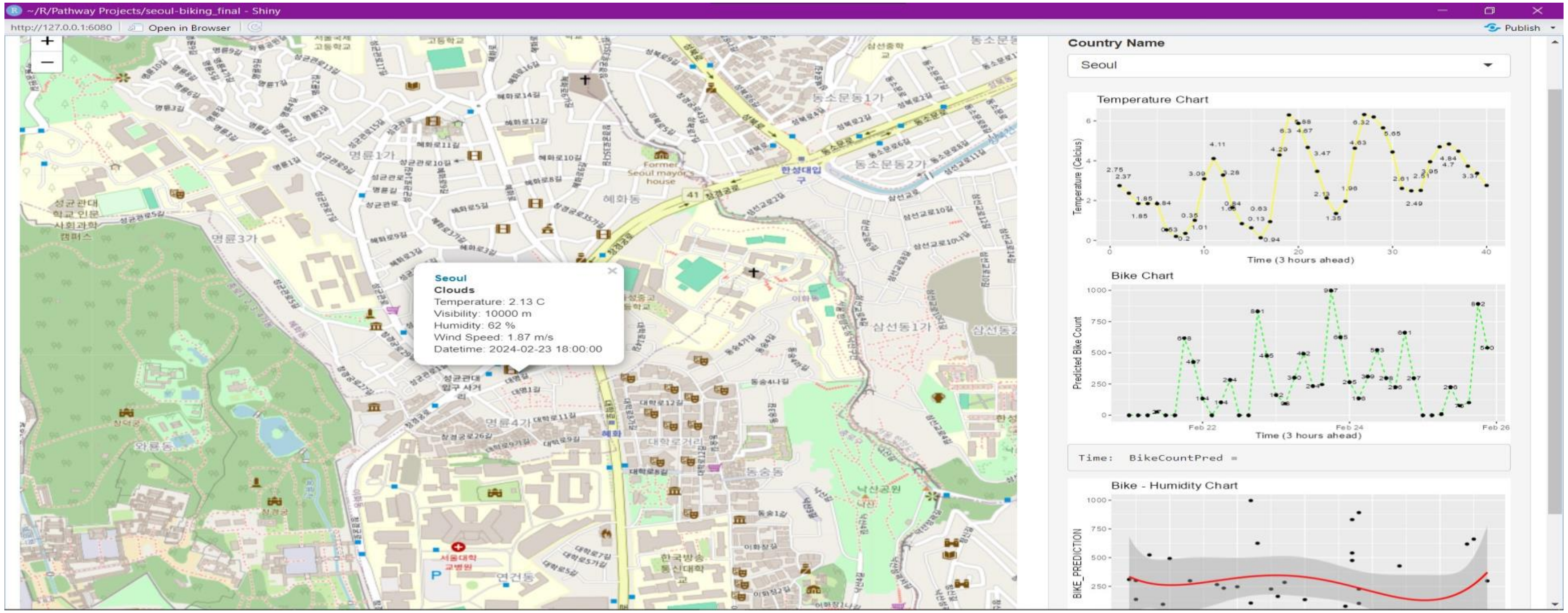
# Dashboard

# Bike-sharing Demand Prediction Dashboard Overview



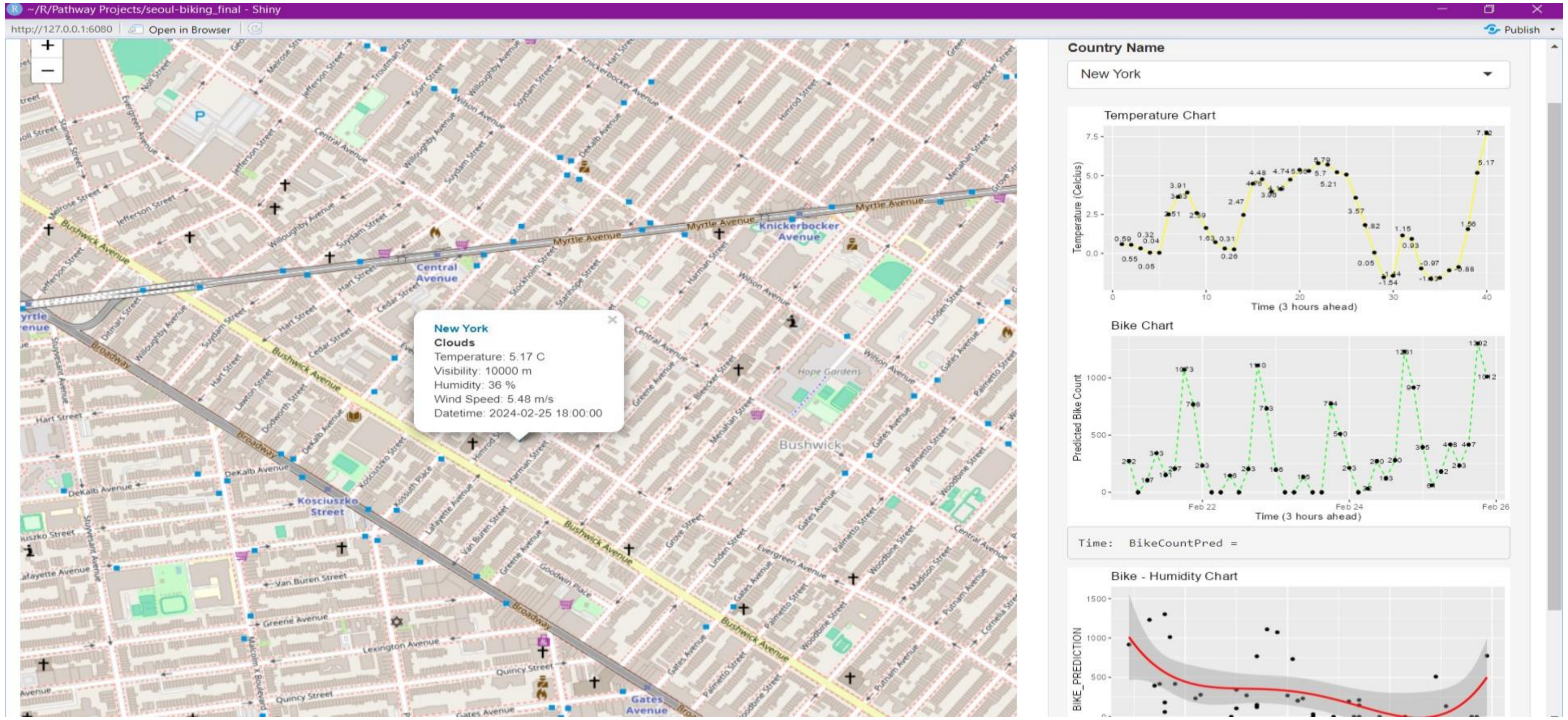

Main default page of Shiny app showing general overview of all cities from which we can select a specific city to show more details about each of its bike sharing demand information.

# Selected City : SEOUL




Specific city, Seoul, selected to show its temperature, bike prediction count and relative humidity at specific time intervals. Closer city view of Seoul also shown on map.

# Selected City : New York




Another city, this time New York, is selected on the shiny app showing this cities  specific temperature, humidity and bike prediction count data with a closer view of the city

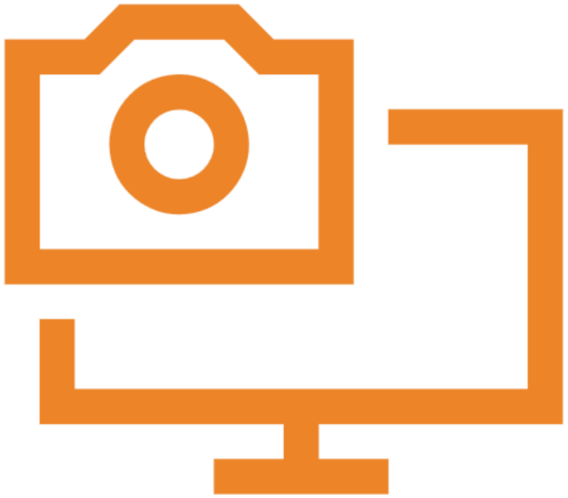
# CONCLUSION

- The observed patterns in bike-sharing demand highlight the significant influence of seasons on rental behavior. Rental activity peaks during the summer and autumn seasons, coinciding with warmer temperatures and favorable weather conditions, while experiencing a decline during winter and early spring. This seasonal variation underscores the importance of considering environmental factors and seasonal trends when designing bike-sharing services and infrastructure. Understanding how seasonality affects rental behavior can inform strategic decisions regarding service expansion, promotional campaigns, and resource allocation to meet fluctuating demand throughout the year.

- The insights gleaned from the model should enable stakeholders to make informed decisions, such as adjusting bike inventory levels based on weather forecasts, time of day patterns, and seasonality trends, thereby enhancing the overall user experience of bike-sharing systems.

- Moving forward, there are several avenues for further exploration and improvement. Enhancing the model's predictive accuracy by incorporating additional data sources, such as demographic information, traffic patterns, and special events, could provide deeper insights into bike-sharing dynamics

# APPENDIX



- Included are all relevant assets like R code snippets, SQL queries, charts,.

# Data Collection(Weather API)

*TODO:* Get the root HTML node

```
[2]: url <- "https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems"
     # Get the root HTML node by calling the `read_html()` method with URL
```

```
[3]: webpage <- read_html(url)

     table_nodes <- html_nodes(webpage, "table")
```

```
[5]: if (length(table_nodes) >= 1) {
       first_table_data <- html_table(table_nodes[[1]], fill = TRUE)

       # Print the extracted data from the first table as a data frame
       df_first_table <- as.data.frame(first_table_data)
       print(df_first_table)
     } else {
       cat("No tables found on the webpage.\n")
     }
```

```
                                   Country
1                                  Albania
2                                Argentina
3                                Argentina
4                                Argentina
5                                Argentina
```

# Data Collection(Weather API)

Summarize the bike sharing system data frame

```
[6]: # Summarize the dataframe
summary(df_first_table)
```

```
   Country            City               Name              System         
 Length:564         Length:564         Length:564         Length:564        
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
   Operator           Launched         Discontinued         Stations        
 Length:564         Length:564         Length:564         Length:564        
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
   Bicycles         Daily ridership   
 Length:564         Length:564        
 Class :character   Class :character  
 Mode  :character   Mode  :character  
```

Export the data frame as a csv file called `raw_bike_sharing_systems.csv`

```
[7]: # Export the dataframe into a csv file
write.csv(df_first_table, file = "raw_bike_sharing_systems.csv", row.names = FALSE)

# Print a message indicating the successful export
cat("Data frame exported to raw_bike_sharing_systems.csv\n")
```

```
Data frame exported to raw_bike_sharing_systems.csv
```

# Data Collection (Webscraping)

```r
# Converting the bike-sharing system table into a dataframe
table_nodes <- html_nodes(webpage, "table")

# Extracting information from the first table
if (length(table_nodes) >= 1) {
  first_table_data <- html_table(table_nodes[[1]], fill = TRUE)

  # Printing the extracted data from the first table as a data frame
  df_first_table <- as.data.frame(first_table_data)
  print(df_first_table)
} else {
  cat("No tables found on the webpage.\n")
}
```

```
                                Country
1                               Albania
2                             Argentina
3                             Argentina
4                             Argentina
5                             Argentina
6                             Australia
7                             Australia
8                             Australia
9                             Australia
10                            Australia
11                            Australia
12                              Austria
13                              Austria
14                              Austria
15                              Austria
16                              Austria
17                              Austria
18                           Bangladesh
19                              Belgium
20                              Belgium
21                              Belgium
```

Summarize the bike sharing system data frame

```r
# Summarize the dataframe
summary(df_first_table)
```

```
   Country              City                Name               System
 Length:560         Length:560         Length:560         Length:560
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
   Operator           Launched         Discontinued         Stations
 Length:560         Length:560         Length:560         Length:560
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
   Bicycles         Daily ridership
 Length:560         Length:560
 Class :character   Class :character
 Mode  :character   Mode  :character
```

Export the data frame as a csv file called `raw_bike_sharing_systems.csv`

```r
# Exporting the dataframe into a csv file
write.csv(df_first_table, file = "raw_bike_sharing_systems.csv", row.names = FALSE)

# Print a message indicating the successful export
cat("Data frame exported to raw_bike_sharing_systems.csv\n")
```

```
Data frame exported to raw_bike_sharing_systems.csv
```

# Data Wrangling (Regex)

To improve dataset readbility by both human and computer systems, we first need to standardize the column names of the datasets above using the following naming convention:

- Column names need to be UPPERCASE
- The word separator needs to be an underscore, such as in `COLUMN_NAME`

You can use the following dataset list and the `names()` function to get and set each of their column names, and convert them according to our defined naming convention.

```
[14]: dataset_list <- c('raw_bike_sharing_systems.csv', 'raw_seoul_bike_sharing.csv', 'raw_cities_weather_forecast.csv', 'raw_worldcities.csv')
```

*TODO*: Write a `for` loop to iterate over the above datasets and convert their column names

```
[15]: for (dataset_name in dataset_list){
    # Read dataset
    dataset <- read_csv(dataset_name)
    # Standardized its columns:

    # Convert all column names to uppercase
    names(dataset) <- toupper(names(dataset))
    # Replace any white space separators by underscores, using the str_replace_all function
    names(dataset) <- str_replace_all(names(dataset), " ", "_")
    # Save the dataset
    write.csv(dataset, dataset_name, row.names=FALSE)
}
```

```
Parsed with column specification:
cols(
  COUNTRY = col_character(),
  CITY = col_character(),
  NAME = col_character(),
  SYSTEM = col_character(),
  OPERATOR = col_character(),
  LAUNCHED = col_character()
```

```
  PRESSURE = col_double(),
  HUMIDITY = col_double(),
  WIND_SPEED = col_double(),
  WIND_DEG = col_double(),
  SEASON = col_character(),
  FORECAST_DATETIME = col_datetime(format = "")
)
Parsed with column specification:
cols(
  CITY = col_character(),
  CITY_ASCII = col_character(),
  LAT = col_double(),
  LNG = col_double(),
  COUNTRY = col_character(),
  ISO2 = col_character(),
  ISO3 = col_character(),
  ADMIN_NAME = col_character(),
  CAPITAL = col_character(),
  POPULATION = col_double(),
  ID = col_double()
)
```

*TODO*: Read the resulting datasets back and check whether their column names follow the naming convention

```
[16]: for (dataset_name in dataset_list){
    # Print a summary for each data set to check whether the column names were correctly converted
    dataset <- read_csv (dataset_name)
    summary(dataset)
}
```

```
Parsed with column specification:
cols(
  COUNTRY = col_character(),
  CITY = col_character(),
  NAME = col_character(),
  SYSTEM = col_character(),
  OPERATOR = col_character(),
  LAUNCHED = col_character(),
  DISCONTINUED = col_character()
```

# Data Wrangling (Regex)

```
)
Parsed with column specification:
cols(
  CITY = col_character(),
  CITY_ASCII = col_character(),
  LAT = col_double(),
  LNG = col_double(),
  COUNTRY = col_character(),
  ISO2 = col_character(),
  ISO3 = col_character(),
  ADMIN_NAME = col_character(),
  CAPITAL = col_character(),
  POPULATION = col_double(),
  ID = col_double()
)
```

### Process the web-scraped bike sharing system dataset

By now we have standardized all column names. Next, we will focus on cleaning up the values in the web-scraped bike sharing systems dataset.

[17]:
```
# First load the dataset
bike_sharing_df <- read_csv("raw_bike_sharing_systems.csv")
```

```
Parsed with column specification:
cols(
  COUNTRY = col_character(),
  CITY = col_character(),
  NAME = col_character(),
  SYSTEM = col_character(),
  OPERATOR = col_character(),
  LAUNCHED = col_character(),
  DISCONTINUED = col_character(),
  STATIONS = col_character(),
  BICYCLES = col_character(),
  DAILY_RIDERSHIP = col_character()
)
```

[18]:
```
# Print its head
head(bike_sharing_df)
```

A tibble: 6 × 10

| COUNTRY | CITY | NAME | SYSTEM | OPERATOR | LAUNCHED | DISCONTINUED | STATIONS | BICYCLES | DAILY_RIDERSHIP |
|---|---|---|---|---|---|---|---|---|---|
| <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> |
| Albania | Tirana | Ecovolis | NA | NA | March 2011 | NA | 8 | 200 | NA |
| Argentina | Mendoza | Metrobici | NA | NA | 2014 | NA | 2 | 40 | NA |
| Argentina | San Lorenzo, Santa Fe | Biciudad | Biciudad | NA | 27 November 2016 | NA | 8 | 80 | NA |
| Argentina | Buenos Aires | Ecobici | Serttel Brasil | Bike In Baires Consortium.[10] | 2010 | NA | 400 | 4000 | 21917 |
| Argentina | Rosario | Mi Bici Tu Bici[11] | NA | NA | 2 December 2015 | NA | 47 | 480 | NA |
| Australia | Melbourne[12] | Melbourne Bike Share | PBSC & 8D | Motivate | June 2010 | 30 November 2019[13] | 53 | 676 | NA |

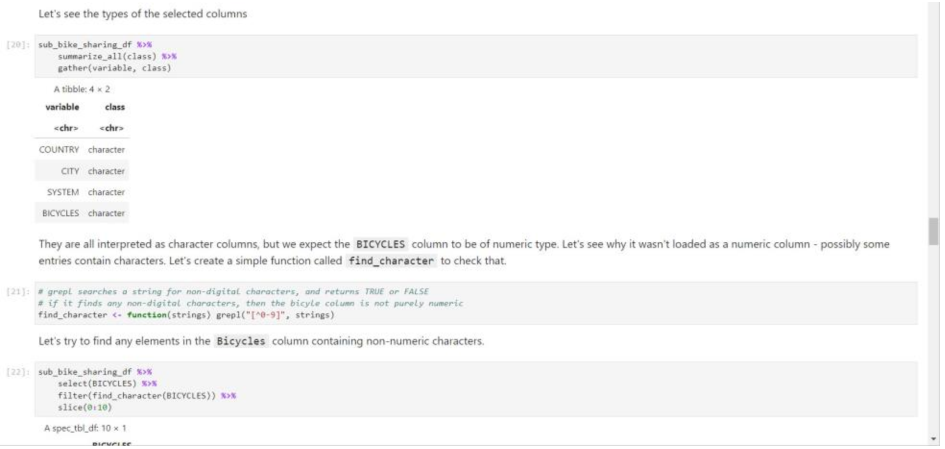
Even from the first few rows, you can see there is plenty of undesireable embedded textual content, such as the reference link included in `Melbourne[12]`.

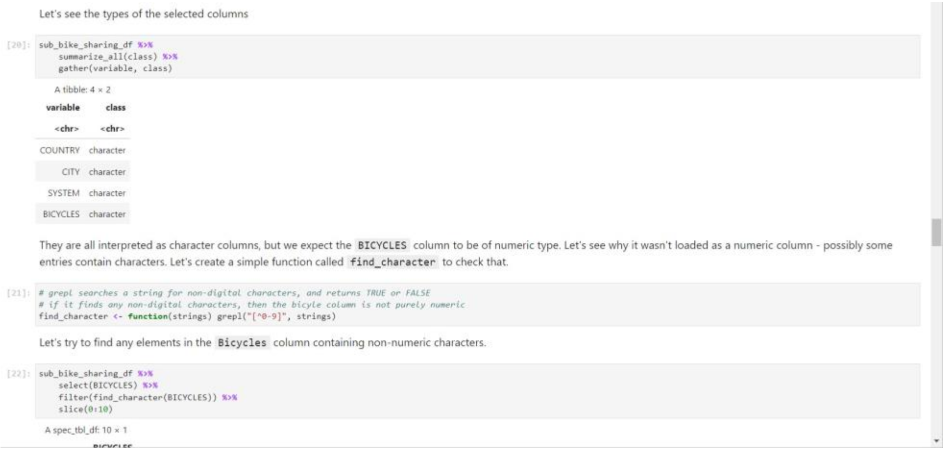
In this project, let's only focus on processing the following revelant columns (feel free to process the other columns for more practice):

- `COUNTRY` : Country name
- `CITY` : City name
- `SYSTEM` : Bike-sharing system name
- `BICYCLES` : Total number of bikes in the system

[19]:
```
# Select the four columns
sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)
```

# Data Wrangling (Regex)



# Data Wrangling (Regex)

```
    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.

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]`.

```
[23]: # 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)
```

```
[24]: # 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.

```
[25]: # 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> |
| Melbourne[12] |

# Data Wrangling (Regex)

| |
|---|
| EasyBike[58] |
| 4 Gen.[61] |
| 3 Gen. SmooveKey[113] |
| 3 Gen. Smoove[141][142][143][139] |
| 3 Gen. Smoove[179] |
| 3 Gen. Smoove[181] |
| 3 Gen. Smoove[183] |

So the `SYSTEM` column also has some reference links.

After some preliminary investigations, we identified that the `CITY` and `SYSTEM` columns have some undesired reference links, and the `BICYCLES` column has both reference links and some textual annotations.

Next, you need to use regular expressions to clean up the unexpected reference links and text annotations in numeric values.

## TASK: Remove undesired reference links using regular expressions

*TODO:* Write a custom function using `stringr::str_replace_all` to replace all reference links with an empty character for columns `CITY` and `SYSTEM`

[27]:
```
# remove reference link
remove_ref <- function(strings) {
    ref_pattern <- "\\[[A-z0-9]+\\]"
    # Replace all matched substrings with a white space using str_replace_all()
    strings <- str_replace_all(strings, ref_pattern, "\t")
    return(strings)
}
```

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

```
result <- sub_bike_sharing_df %>% mutate(CITY=remove_ref(CITY),
                                         SYSTEM=remove_ref(SYSTEM),
                                         BICYCLES=remove_ref(BICYCLES))
```

*TODO:* Use the following code to check whether all reference links are removed:

```
result %>%
    select(CITY, SYSTEM, BICYCLES) %>%
    filter(find_reference_pattern(CITY) | find_reference_pattern(SYSTEM) | find_reference_pattern(BICYCLES))
```

A spec_tbl_df: 0 × 3

| CITY | SYSTEM | BICYCLES |
|---|---|---|
| <chr> | <chr> | <chr> |

# Data Wrangling (Regex)

## TASK: Extract the numeric value using regular expressions

*TODO:* Write a custom function using `stringr::str_extract` to extract the first digital substring match and convert it into numeric type For example, extract the value '32' from `32 (including 6 rollers) [162]`.

[30]:
```
# Extract the first number
extract_num <- function(columns){
    # Define a digital pattern
    digitals_pattern <- "[0-9]+"
    # Find the first match using str_extract
    columns <- str_extract(columns, digitals_pattern)
    # Convert the result to numeric using the as.numeric() function
    columns <- as.numeric(columns)
}
```

*TODO:* Use the `dplyr::mutate()` function to apply `extract_num` on the `BICYCLES` column

[31]:
```
# Use the mutate() function on the BICYCLES column
result <- result %>%
          mutate (BICYCLES = extract_num(BICYCLES))
```

*TODO:* Use the summary function to check the descriptive statistics of the numeric `BICYCLES` column

[32]:
```
summary(result$BICYCLES)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
      5     100     350    2022    1400   78000      78 
```

*TODO:* Write the cleaned bike-sharing systems dataset into a csv file called `bike_sharing_systems.csv`

[34]:
```
# Write dataset to `bike_sharing_systems.csv`
write.csv(result, file = "bike_sharing_systems.csv", row.names = FALSE)
```

# EDA With SQL

### Establish your SQllte connection

Load the 'RSQLite' library, and use the 'dbConnect( )' function as you did in the previous lab to establish the connection to your SQLite database.

You are now ready to start running SQL queries using the RSQLite library as you did in Course 3.

[2]:
```
# provide your solution here
con <- dbConnect(SQLite(), "seoul.db")
```

[4]:
```
library(readr)
```

```
Warning message:
“replacing previous import ‘lifecycle::last_warnings’ by ‘rlang::last_warnings’ when loading ‘tibble’”Warning message:
“replacing previous import ‘ellipsis::check_dots_unnamed’ by ‘rlang::check_dots_unnamed’ when loading ‘tibble’”Warning message:
“replacing previous import ‘ellipsis::check_dots_used’ by ‘rlang::check_dots_used’ when loading ‘tibble’”Warning message:
“replacing previous import ‘ellipsis::check_dots_empty’ by ‘rlang::check_dots_empty’ when loading ‘tibble’”
```

[5]:
```
# Download the CSV files
world_cities <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/
bike_sharing_systems <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-Skills
cities_weather_forecast <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-Ski
seoul_bike_sharing <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNe

# Write the data to the SQLite database
dbWriteTable(con, "WORLD_CITIES", world_cities, overwrite = TRUE)
dbWriteTable(con, "BIKE_SHARING_SYSTEMS", bike_sharing_systems, overwrite = TRUE)
dbWriteTable(con, "CITIES_WEATHER_FORECAST", cities_weather_forecast, overwrite = TRUE)
dbWriteTable(con, "SEOUL_BIKE_SHARING", seoul_bike_sharing, overwrite = TRUE)
```

```
Parsed with column specification:
cols(
  CITY = col_character(),
  CITY_ASCII = col_character(),
```

[5]:
```
# Execute SQL queries to verify data loading
result <- dbGetQuery(con, "SELECT * FROM WORLD_CITIES LIMIT 5")
print("WORLD_CITIES table:")
print(result)
```

```
[1] "WORLD_CITIES table:"
     CITY CITY_ASCII     LAT      LNG     COUNTRY ISO2 ISO3  ADMIN_NAME CAPITAL
1   Tokyo      Tokyo 35.6897 139.6922       Japan   JP  JPN       Tōkyō primary
2 Jakarta    Jakarta -6.2146 106.8451   Indonesia   ID  IDN     Jakarta primary
3   Delhi      Delhi 28.6600  77.2300       India   IN  IND       Delhi   admin
4  Mumbai     Mumbai 18.9667  72.8333       India   IN  IND Mahārāshtra   admin
5  Manila     Manila 14.5958 120.9772 Philippines   PH  PHL      Manila primary
  POPULATION         ID
1   37977000 1392685764
2   34540000 1360771077
3   29617000 1356872604
4   23355000 1356226629
5   23088000 1608618140
```

# EDA With SQL

## Task 1 - Record Count

Determine how many records are in the seoul_bike_sharing dataset.

### Solution 1

```
# provide your solution here
query <- "
SELECT DATE, HOUR, SUM(RENTED_BIKE_COUNT) AS total_rentals
FROM seoul_bike_sharing
GROUP BY DATE, HOUR
ORDER BY total_rentals DESC
LIMIT 1;
"
result <- dbGetQuery(con, query)
result
```

A data.frame: 1 × 3

| DATE | HOUR | total_rentals |
|---|---|---|
| <chr> | <dbl> | <dbl> |
| 19/06/2018 | 18 | 3556 |

## Task 7 - Hourly popularity and temperature by season

Determine the average hourly temperature and the average number of bike rentals per hour over each season. List the top ten results by average bike count.

### Solution 7

```
query <- "
SELECT s.SEASONS,
       AVG(s.TEMPERATURE) AS avg_hourly_temperature,
       AVG(s.RENTED_BIKE_COUNT) AS avg_bike_rentals_per_hour
FROM seoul_bike_sharing s
GROUP BY s.SEASONS
ORDER BY avg_bike_rentals_per_hour DESC
LIMIT 10;
"
result <- dbGetQuery(con, query)
result
```

A data.frame: 4 × 3

| SEASONS | avg_hourly_temperature | avg_bike_rentals_per_hour |
|---|---|---|
| <chr> | <dbl> | <dbl> |
| Summer | 26.587711 | 1034.0734 |
| Autumn | 13.821580 | 924.1105 |
| Spring | 13.021685 | 746.2542 |
| Winter | -2.540463 | 225.5412 |

# EDA With SQL

**Find the average hourly bike count during each season.**

Also include the minimum, maximum, and standard deviation of the hourly bike count for each season.

> Hint : Use the SQRT(AVG(col*col) - AVG(col)*AVG(col) ) function where col refers to your column name for finding the standard deviation

### Solution 8

[12]:
```
# provide your solution here

query<- "
SELECT
    SEASONS,
    AVG(RENTED_BIKE_COUNT) AS avg_hourly_bike_count,
    MIN(RENTED_BIKE_COUNT) AS min_hourly_bike_count,
    MAX(RENTED_BIKE_COUNT) AS max_hourly_bike_count,
    SQRT(AVG(RENTED_BIKE_COUNT * RENTED_BIKE_COUNT) - AVG(RENTED_BIKE_COUNT) * AVG(RENTED_BIKE_COUNT)) AS std_dev_hourly_bike_count
FROM
    seoul_bike_sharing
GROUP BY
    SEASONS;
"
result <- dbGetQuery(con, query)

print(result)
```

```
  SEASONS avg_hourly_bike_count min_hourly_bike_count max_hourly_bike_count
1  Autumn              924.1105                     2                  3298
2  Spring              746.2542                     2                  3251
3  Summer             1034.0734                     9                  3556
4  Winter              225.5412                     3                   937
  std_dev_hourly_bike_count
1                  617.3885
2                  618.5247
3                  690.0884
```

[11]:
```
# Weather Seasonality
query <- "
SELECT
    SEASONS,
    AVG(TEMPERATURE) AS avg_temperature,
    AVG(HUMIDITY) AS avg_humidity,
    AVG(WIND_SPEED) AS avg_wind_speed,
    AVG(VISIBILITY) AS avg_visibility,
    AVG(DEW_POINT_TEMPERATURE) AS avg_dew_point_temperature,
    AVG(SOLAR_RADIATION) AS avg_solar_radiation,
    AVG(RAINFALL) AS avg_rainfall,
    AVG(SNOWFALL) AS avg_snowfall,
    AVG(RENTED_BIKE_COUNT) AS avg_bike_count
FROM
    seoul_bike_sharing
GROUP BY
    SEASONS
ORDER BY
    avg_bike_count DESC;
"
result <- dbGetQuery(con, query)

print(result)
```

```
  SEASONS avg_temperature avg_humidity avg_wind_speed avg_visibility
1  Summer       26.587711     64.98143       1.609420       1501.745
2  Autumn       13.821580     59.04491       1.492101       1558.174
3  Spring       13.021685     58.75833       1.857778       1240.912
4  Winter       -2.540463     49.74491       1.922685       1445.987
  avg_dew_point_temperature avg_solar_radiation avg_rainfall avg_snowfall
1                 18.750136           0.7612545   0.25348732   0.00000000
2                  5.150594           0.5227827   0.11765617   0.06350026
3                  4.091389           0.6803009   0.18694444   0.00000000
4                -12.416667           0.2981806   0.03282407   0.24750000
  avg_bike_count
1      1034.0734
2       924.1105
3       746.2542
4       225.5412
```

# EDA With SQL

### Task 10 - Total Bike Count and City Info for 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.

Notice that in this case, the CITY column will work for the WORLD_CITIES table, but in general you would have to use the CITY_ASCII column.

#### Solution 10

```
[8]:
# Execute SQL query to retrieve city information and total bikes available in Seoul
query <- "
SELECT
    wc.CITY,
    wc.COUNTRY,
    wc.LAT,
    wc.LNG AS LON,
    wc.POPULATION,
    bs.BICYCLES AS total_bikes_available
FROM
    WORLD_CITIES wc
JOIN
    BIKE_SHARING_SYSTEMS bs ON wc.CITY_ASCII = bs.CITY
WHERE
    wc.CITY = 'Seoul';
"
result <- dbGetQuery(con, query)

print(result)
```

```
   CITY      COUNTRY     LAT LON POPULATION total_bikes_available
1 Seoul Korea, South 37.5833 127   21794000                 20000
```

#### Solution 11

```
•[10]:
# provide your solution here
# Execute SQL query to retrieve city names and coordinates  with total bike counts between 15000 and 20000
query <- "
SELECT
    wc.CITY,
    wc.COUNTRY,
    wc.LAT,
    wc.LNG AS LNG,
    wc.POPULATION,
    bs.BICYCLES AS total_bikes
FROM
    WORLD_CITIES wc
JOIN
    BIKE_SHARING_SYSTEMS bs ON wc.CITY_ASCII = bs.CITY
WHERE
    bs.BICYCLES BETWEEN 15000 AND 20000;
"

# Execute the query
result <- dbGetQuery(con, query)

# Print the result
print(result)
```

```
      CITY      COUNTRY     LAT      LNG POPULATION total_bikes
1  Beijing        China 39.9050 116.3914   19433000       16000
2   Ningbo        China 29.8750 121.5492    7639000       15000
3 Shanghai        China 31.1667 121.4667   22120000       19165
4  Weifang        China 36.7167 119.1000    9373000       20000
5    Xi'an        China 34.2667 108.9000    7135000       20000
6  Zhuzhou        China 27.8407 113.1469    3855609       20000
7    Seoul Korea, South 37.5833 127.0000   21794000       20000
```

```
[ ]:
close(conn)
```

# EDA With Data Visualization

### Task 1 - Load the dataset

Ensure you read `DATE` as type `character`.

### Solution 1

```
[4]: seoul_bike_sharing_url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/

# Load the dataset with DATE as character type
seoul_bike_sharing <- read.csv(seoul_bike_sharing_url, colClasses = c(DATE = "character"))

# Display the structure of the dataset
str(seoul_bike_sharing)
```

```
'data.frame':	8465 obs. of  14 variables:
 $ DATE                 : chr  "01/12/2017" "01/12/2017" "01/12/2017" "01/12/2017" ...
 $ RENTED_BIKE_COUNT    : int  254 204 173 107 78 100 181 460 930 490 ...
 $ HOUR                 : int  0 1 2 3 4 5 6 7 8 9 ...
 $ 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 ...
```

### Task 2 - Recast `DATE` as a date

Use the format of the data, namely "%d/%m/%Y".

### Solution 2

```
[5]: # Recast DATE as a date with the format "%d/%m/%Y"
seoul_bike_sharing$DATE <- as.Date(seoul_bike_sharing$DATE, format = "%d/%m/%Y")

# Display the structure of the updated dataset
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                 : int  0 1 2 3 4 5 6 7 8 9 ...
 $ 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 ...
```

# EDA With Data Visualization

Task 3 - Cast `HOURS` as a categorical variable

Also, coerce its levels to be an ordered sequence. This will ensure your visualizations correctly utilize `HOURS` as a discrete variable with the expected ordering.

### Solution 3

[6]:
```
# provide your solution here
# Cast HOUR as a categorical variable with ordered levels
seoul_bike_sharing$HOUR <- factor(seoul_bike_sharing$HOUR, levels = 0:23, ordered = TRUE)
```

### Check the structure of the dataframe

[7]:
```
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

[8]:
```
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 `sumamry()` function to describe the `seoul_bike_sharing` dataset.

### Solution 4

[9]:
```
# 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      
```

# EDA With Data Visualization

```
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
3rd Qu.:0.9300   3rd Qu.: 0.0000   3rd Qu.:0.00000
Max.   :3.5200   Max.   :35.0000   Max.   :8.80000

      HOLIDAY      FUNCTIONING_DAY
Holiday   : 408   Yes:8465
No Holiday:8057
```

### Some Basic Observations:

- We can see from `DATE` that we have exactly a full year of data.
- No records have zero bike counts.
- Spring and Winter have the same count of records, while autumn has the least and Summer has the most.
- Temperature has a large range, so we might expect it to explain at least some of the variation in bike rentals.
- Precipitation seems to be quite rare, only happening in the fourth quartiles for both `RAINFALL` and `SNOWFALL`.
- The average `WINDSPEED` is very light at only 1.7 m/s, and even the maximum is only a moderate breeze (Google 'Beaufort Wind Scale' to find the different wind descriptions)

By now, you might agree that Exploratory Data Analysis can create more questions than answers. That's okay - you'll have a much deeper understanding and appreciation for your data as a result!

Task 5 - Based on the above stats, calculate how many Holidays there are.

Solution 5:

[10]:
```r
# provide your solution here

# Calculate the number of holidays
holiday_count <- table(seoul_bike_sharing$HOLIDAY)["Holiday"]

# Display the result
holiday_count
```

**Holiday:** 408

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

Solution 6

[11]:
```r
# provide your solution here

# Calculate the total number of records
total_records <- nrow(seoul_bike_sharing)

# Calculate the percentage
percentage_holidays <- (holiday_count / total_records) * 100

# Display the result
percentage_holidays
```

**Holiday:** 4.8198464264619

# EDA With Data Visualization

Task 7 - Given there is exactly a full year of data, determine how many records we expect to have.

Solution 7

[13]:
```r
# provide your solution here

# Assuming hourly data and a full year
days_in_year <- 365
hours_per_day <- 24

# Calculate expected number of records
expected_records <- days_in_year * hours_per_day

# Print the result
cat("Expected Number of Records:", expected_records, "\n")
```

```
Expected Number of Records: 8760
```

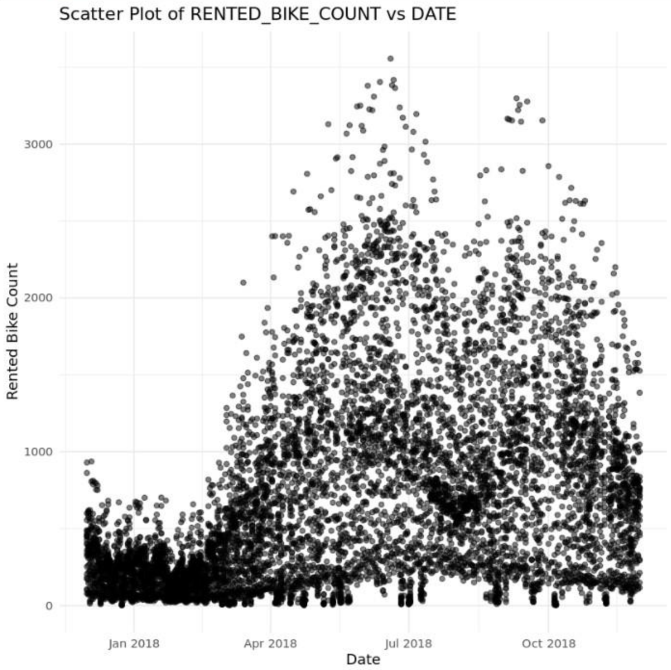
Task 8 - Given the observations for the 'FUNCTIONING_DAY' how many records must there be?

Solution 8

[15]:
```r
# provide your solution here

# Count the number of records for each level in FUNCTIONING_DAY
table(seoul_bike_sharing$FUNCTIONING_DAY)
```

```
 Yes
8465
```

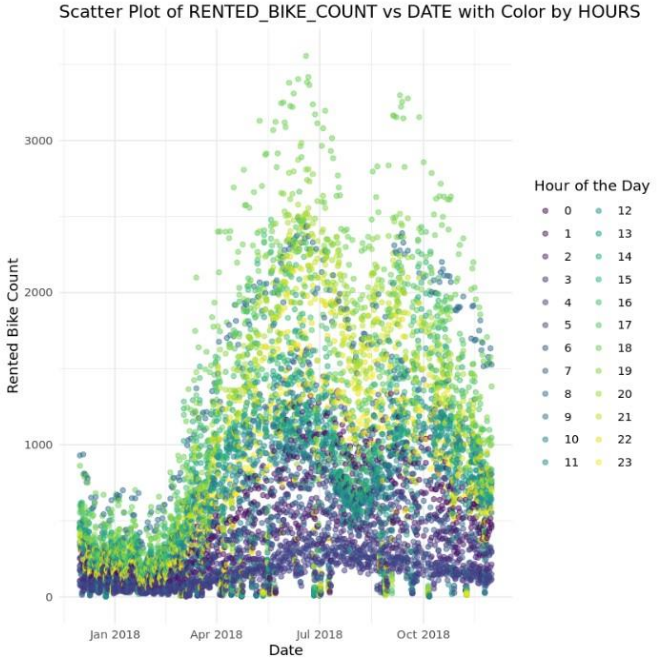
[20]:
```r
# SCATTER PLOT OF RENTED_BIKE_COUNT VS DATE

ggplot(seoul_bike_sharing, aes(x = DATE, y = RENTED_BIKE_COUNT)) +
  geom_point(alpha = 0.5) +  # Set opacity using alpha parameter
  labs(title = "Scatter Plot of RENTED_BIKE_COUNT vs DATE",
       x = "Date",
       y = "Rented Bike Count") +
  theme_minimal()
```



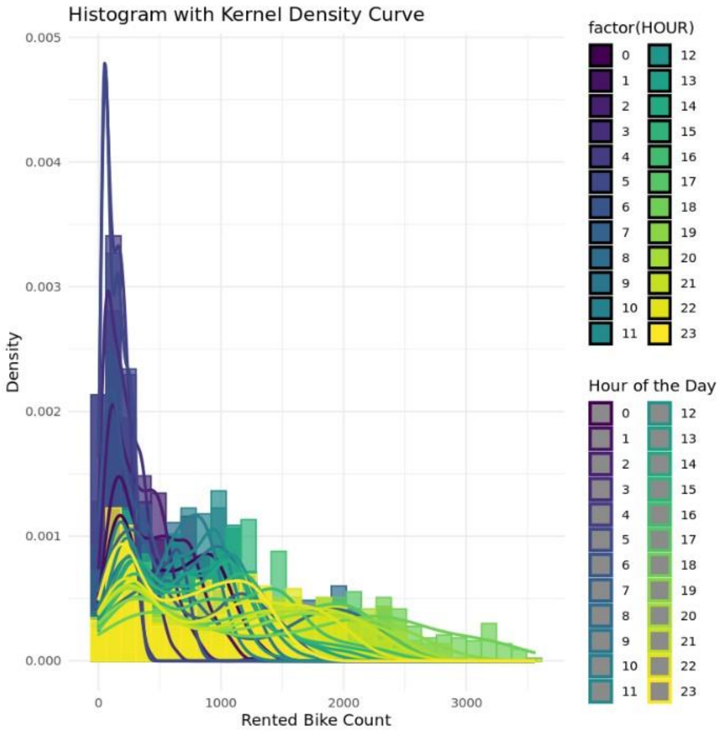

Ungraded Task: We can see some patterns emerging here.

# EDA With Data Visualization

[21]:
```r
# provide your solution here

ggplot(seoul_bike_sharing, aes(x = DATE, y = RENTED_BIKE_COUNT, color = factor(HOUR))) +
  geom_point(alpha = 0.5) +  # Set opacity using alpha parameter
  labs(title = "Scatter Plot of RENTED_BIKE_COUNT vs DATE with Color by HOURS",
       x = "Date",
       y = "Rented Bike Count",
       color = "Hour of the Day") +
  theme_minimal()
```

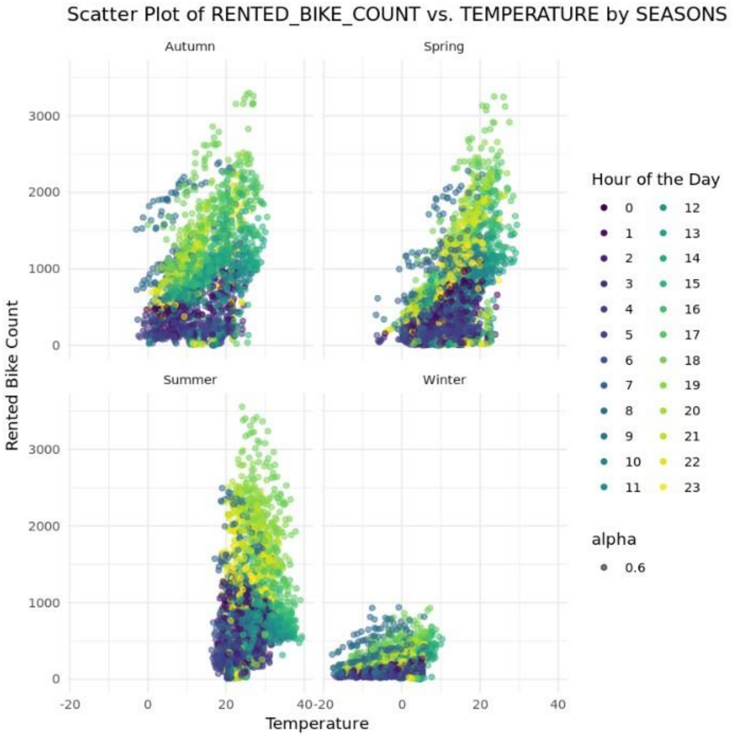



[22]:
```r
# provide your solution here

ggplot(seoul_bike_sharing, aes(x = RENTED_BIKE_COUNT, y = ..density..)) +
  geom_histogram(aes(fill = factor(HOUR), color = factor(HOUR)),
                 bins = 30, alpha = 0.7, position = "identity") +
  geom_density(aes(color = factor(HOUR)), size = 1) +
  labs(title = "Histogram with Kernel Density Curve",
       x = "Rented Bike Count",
       y = "Density",
       color = "Hour of the Day") +
  theme_minimal()
```



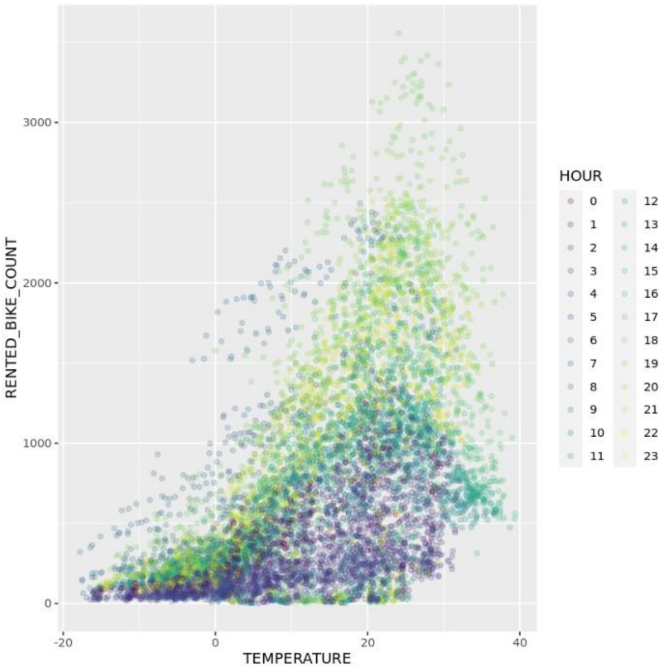
# EDA With Data Visualization

```
[23]: # provide your solution here

ggplot(seoul_bike_sharing, aes(x = TEMPERATURE, y = RENTED_BIKE_COUNT, color = factor(HOUR), alpha = 0.6)) +
  geom_point() +
  facet_wrap(~SEASONS) +
  labs(title = "Scatter Plot of RENTED_BIKE_COUNT vs. TEMPERATURE by SEASONS",
       x = "Temperature",
       y = "Rented Bike Count",
       color = "Hour of the Day") +
  theme_minimal()
```



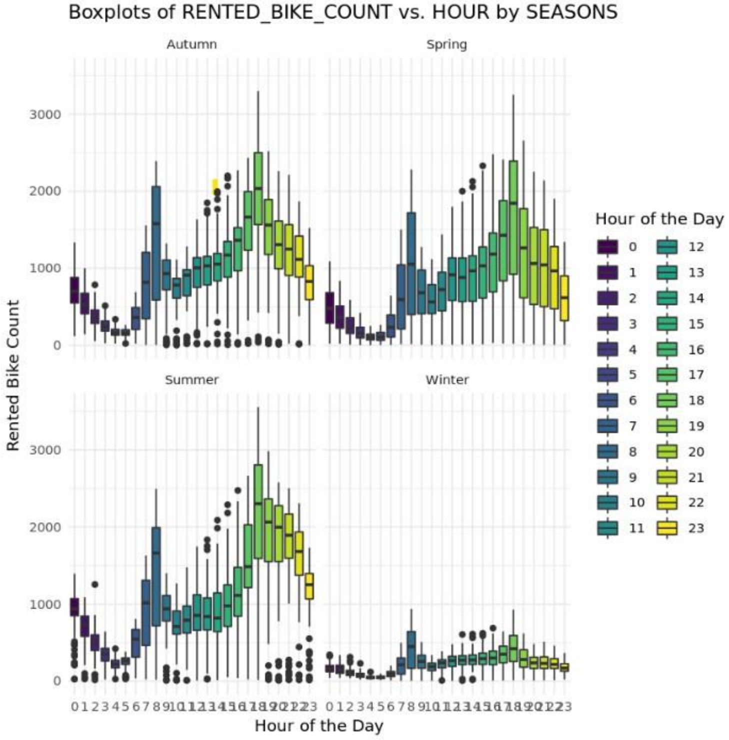

Comparing this plot to the same plot below, but without grouping by `SEASONS`, shows how important seasonality is in explaining bike rental counts.

```
[24]: ggplot(seoul_bike_sharing) +
    geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=1/5)
```



# EDA With Data Visualization

```r
[26]: # provide your solution here
ggplot(seoul_bike_sharing, aes(x = factor(HOUR), y = RENTED_BIKE_COUNT, fill = factor(HOUR))) +
  geom_boxplot() +
  facet_wrap(~SEASONS) +
  labs(title = "Boxplots of RENTED_BIKE_COUNT vs. HOUR by SEASONS",
       x = "Hour of the Day",
       y = "Rented Bike Count",
       fill = "Hour of the Day") +
  theme_minimal()
```




Task 15 - Group the data by `DATE`, and use the summarize() function to calculate the daily total rainfall and snowfall.

Also, go ahead and plot the results if you wish.

Solution 15

```r
[27]: # provide your solution here
daily_summary <- seoul_bike_sharing %>%
  group_by(DATE) %>%
  summarize(total_rainfall = sum(RAINFALL),
            total_snowfall = sum(SNOWFALL))

# Plot the results
ggplot(daily_summary, aes(x = DATE)) +
  geom_line(aes(y = total_rainfall, color = "Total Rainfall"), size = 1.5) +
  geom_line(aes(y = total_snowfall, color = "Total Snowfall"), size = 1.5) +
  labs(title = "Daily Total Rainfall and Snowfall",
       x = "Date",
       y = "Total Amount",
       color = "Type") +
  scale_color_manual(values = c("Total Rainfall" = "blue", "Total Snowfall" = "red")) +
  theme_minimal()
```