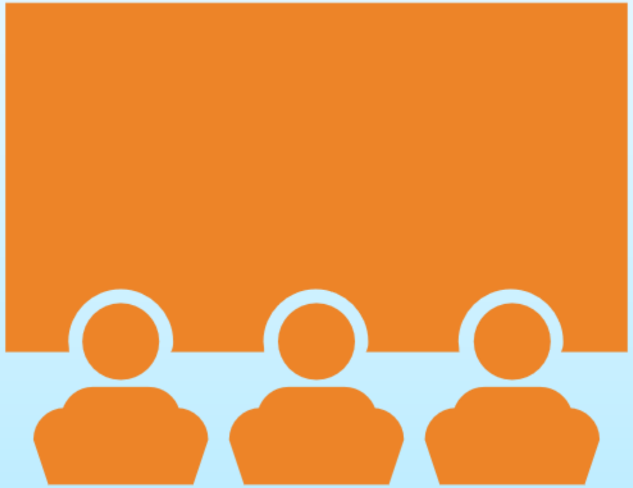
Applied Data Science with R Capstone project

BY Carmen Amelia Ampuero Gonzales

January 23rd, 2025

Outline



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

Executive Summary

This project analyzes the impact of weather conditions on bike-sharing demand in urban areas, leveraging weather data from the OpenWeather API for cities like Seoul, Suzhou, London, New York, and Paris.

A regression-based model was developed to forecast demand based on weather factors. The model demonstrated strong predictive performance with an RMSE of 312 and an R^2 value of 0.76.

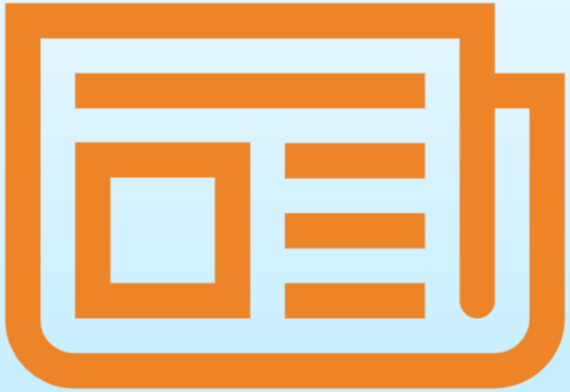
An interactive R Shiny dashboard was created to visualize these predictions, offering city-specific trends and insights on how weather affects bike-sharing usage.

The project provides valuable tools for optimizing resource allocation and improving service efficiency for urban planners and bike-sharing operators.

Introduction

This project focuses on **predicting bike-sharing demand** to improve urban mobility planning and bike availability in high-demand areas. By **analyzing weather data**, cities can optimize bike-sharing systems and support sustainable transportation. The primary goal is to build an accurate predictive model and present the results through an interactive R Shiny dashboard. The analysis covers five global cities—Seoul, Suzhou, London, New York, and Paris—representing diverse climates. The project uses tools such as R for data analysis, ggplot2 for visualization, and the OpenWeather API for weather data collection.

Methodology

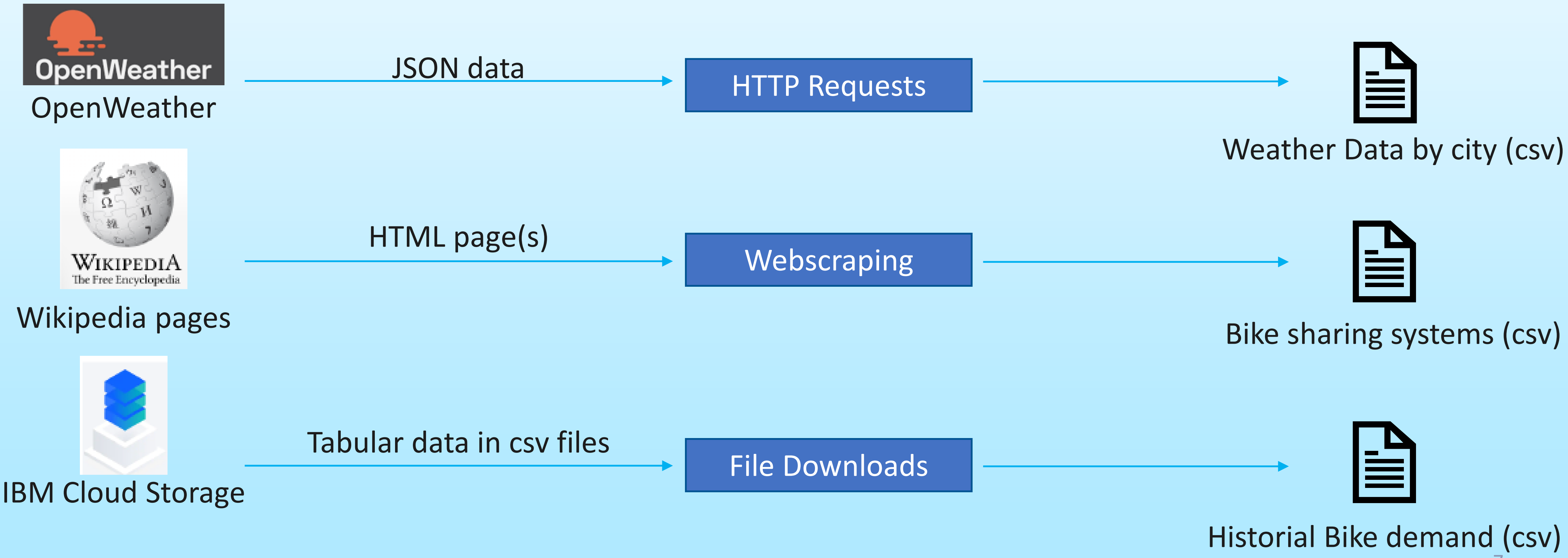


- Perform data collection
- Perform data wrangling
  - With Regular Expressions
  - With dplyr
- Perform exploratory data analysis (EDA) using SQL and visualization
- Perform predictive analysis using regression models
  - How to build the baseline model
  - How to improve the baseline model
- Build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

Methodology

Data collection

- bike_sharing_system.csv: Data extracted from Wiki page, converted into a data frame, and wrote to a cvs file.
- cities_weather_forecast.csv: Used OpenWeather API to get 5-day weather forecast for a list of provided cities, and wrote the data into a cvs file.
- world_cities.csv and seoul_bike_ sharing.csv data: Downloaded datasets in cvs files from cloud storage.



Data wrangling with Regular Expressions

For all collected datasets:

1. Standardized column names

- Uppercase for column names and underscored as separator: COLUMN_NAME
- Used *for* loop to iterate over the above datasets and converted their column names.

From bike-sharing systems dataset:

2. Removed undesired reference links

- Custom function with *stringr::str_replace_all* and *dplyr::mutate()*.

3. Extracted only the numeric value from undesired text annotations

- Custom function with *stringr::str_extract* and *dplyr::mutate()*.

Data wrangling with dplyr

For Seoul bike sharing:

1. Detect and handle missing values

- Dropped rows with missing values in the RENTED_BIKE_COUNT column.
- For TEMPERATURE column, missing values were found in SEASONS == Summer.
  - Imputed missing values using the mean temperature value.

2. Create indicator (dummy) variables for categorical variables

- First:
  - HOUR $\xrightarrow{\text{to}}$ Numerical variable
- Second:
  - SEASONS $\xrightarrow{\text{to}}$ Indicator (dummy) variable
  - HOLIDAY $\xrightarrow{\text{to}}$ Indicator (dummy) variable
  - FUNCTIONING DAY $\xrightarrow{\text{to}}$ Indicator (dummy) variable
  - HOUR $\xrightarrow{\text{to}}$ Indicator (dummy) variable

3. Normalize data using Min-max

$$x_{new} = \frac{x_{old} - x_{min}}{x_{max} - x_{min}}$$

EDA with SQL

SQL queries:

From the **SEOUL_BIKE_SHARING** dataset

1. Record count
2. Operational hours
3. Weather forecast over the next 3 hours
4. Seasons
5. Date range
6. Date and hour with the most bike rentals
7. Hourly popularity and temperature by season
8. Average hourly bike count per season.
9. Weather seasonality

From **WORLD_CITIES** and the **BIKE_SHARING_SYSTEMS** tables

10. Total Bike Count and City Info for Seoul
11. Find all city names and coordinates with comparable bike scale to Seoul's bike sharing system

EDA with data visualization

Plotted charts:

- Scatter plot of RENTED_BIKE_COUNT vs. DATE
- Scatter plot of RENTED_BIKE_COUNT time series, adding HOURS as the color.
- Histogram of RENTED_BIKE_COUNT
- Scatter plot of RENTED_BIKE_COUNT vs. TEMPERATURE by SEASONS
- Boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS

Predictive analysis

Predict Hourly Rented Bike Count using Basic Linear Regression Models:

- Split data into training and testing datasets
- Built a linear regression model using only the weather variables
- Built a linear regression model using both weather and date/time variables
- Evaluated the models and identified important variables

Refine the Baseline Regression Models:

- Added higher order terms
- Added interaction terms
- Added regularization
- Experimented to find the best performed model

Predictive analysis

Building a Baseline Regression Model

Split data into training and testing datasets → Built a linear regression model using weather variables → Built a linear regression model using weather and date/time variables → Evaluated models and identified important variables

Refine the Baseline Regression Models

Added higher order terms on the important variables → Added interaction terms → Added regularization → Experimented to find the best performed model

Build a R Shiny dashboard

Interactions:

- Added a basic max bike prediction overview map
- Added a select input (dropdown) to select a specific city

Plots:

- Added a static temperature trend line
- Added an interactive bike-sharing demand prediction trend line
- Added a static humidity and bike-sharing demand prediction correlation plot

Results

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

EDA with SQL

Busiest bike rental times

```
> dbGetQuery(conn, "
+ SELECT DATE, HOUR, RENTED_BIKE_COUNT AS Maximum_COUNT
+ FROM SEOUL_BIKE_SHARING
+ WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)")
        DATE HOUR Maximum_COUNT
1 19/06/2018   18          3556
```

On June 19, 2018, at 6:00 PM, the highest number of bike rentals in Seoul was recorded.

Hourly popularity and temperature by seasons

```
> dbGetQuery(conn, "SELECT SEASONS, HOUR, AVG(RENTED_BIKE_COUNT), AVG(TEMPERATURE)
+           FROM SEOUL_BIKE_SHARING
+           GROUP BY SEASONS, HOUR
+           ORDER BY AVG(RENTED_BIKE_COUNT) DESC
+           LIMIT 10")
   SEASONS HOUR AVG(RENTED_BIKE_COUNT) AVG(TEMPERATURE)
1   Summer   18               2135.141         29.38791
2   Autumn   18               1983.333         16.03185
3   Summer   19               1889.250         28.27378
4   Summer   20               1801.924         27.06630
5   Summer   21               1754.065         26.27826
6   Spring   18               1689.311         15.97222
7   Summer   22               1567.870         25.69891
8   Autumn   17               1562.877         17.27778
9   Summer   17               1526.293         30.07691
10  Autumn   19               1515.568         15.06346
```

The most popular recorded time for bike rentals is at 6 PM on a summer day with a temperature of around 29 degrees Celsius.

Rental Seasonality

```
> dbGetQuery(conn, "
+ SELECT
+     SEASONS,
+     AVG(RENTED_BIKE_COUNT) AS Avg_Bike_Count,
+     MIN(RENTED_BIKE_COUNT) AS Min_Bike_Count,
+     MAX(RENTED_BIKE_COUNT) AS Max_Bike_Count,
+     SQRT(AVG(RENTED_BIKE_COUNT * RENTED_BIKE_COUNT) - AVG(RENTED_BIKE_COUNT) * AVG(R
ENTED_BIKE_COUNT)) AS Std_Dev_Bike_Count
+ FROM SEOUL_BIKE_SHARING
+ GROUP BY SEASONS")
  SEASONS Avg_Bike_Count Min_Bike_Count Max_Bike_Count Std_Dev_Bike_Count
1  Autumn       924.1105              2           3298           617.3885
2  Spring       746.2542              2           3251           618.5247
3  Summer      1034.0734              9           3556           690.0884
4  Winter       225.5412              3            937           150.3374
```

The season with the most rentals is summer, with an average of 1,034 bikes rented per day.

Weather Seasonality

```
> dbGetQuery(conn, "
+ SELECT
+     SEASONS,
+     AVG(TEMPERATURE) AS Avg_Temperature,
+     AVG(HUMIDITY) AS Avg_Humidity,
+     AVG(WIND_SPEED) AS Avg_WindSpeed,
+     AVG(VISIBILITY) AS Avg_Visibility,
+     AVG(DEW_POINT_TEMPERATURE) AS Avg_DewPointTemp,
+     AVG(SOLAR_RADIATION) AS Avg_SolarRad,
+     AVG(RAINFALL) AS Avg_Rainfall,
+     AVG(SNOWFALL) AS Avg_Snowfall,
+     AVG(RENTED_BIKE_COUNT) AS Avg_RentedBikes
+ FROM SEOUL_BIKE_SHARING
+ GROUP BY SEASONS
+ ORDER BY AVG(RENTED_BIKE_COUNT) DESC")
  SEASONS Avg_Temperature Avg_Humidity Avg_WindSpeed Avg_Visibility Avg_DewPointTemp Avg_SolarRad Avg_Rainfall Avg_Snowfall Avg_RentedBikes
1  Summer       26.587711     64.98143      1.609420       1501.745         18.750136    0.7612545   0.25348732   0.00000000       1034.0734
2  Autumn       13.821580     59.04491      1.492101       1558.174          5.150594    0.5227827   0.11765617   0.06350026        924.1105
3  Spring       13.021685     58.75833      1.857778       1240.912          4.091389    0.6803009   0.18694444   0.00000000        746.2542
4  Winter       -2.540463     49.74491      1.922685       1445.987        -12.416667    0.2981806   0.03282407   0.24750000        225.5412
```

- Summer days are the most preferred for bike rentals.
- The colder the temperature, the fewer bike rentals there are.

Bike-sharing info in Seoul

```
> dbGetQuery(conn, "SELECT B.BICYCLES, B.CITY, B.COUNTRY, W.LAT, W.LNG, W.POPULATION
+ FROM BIKE_SHARING_SYSTEMS AS B
+ LEFT JOIN WORLD_CITIES AS W
+ ON B.CITY = W.CITY_ASCII
+ WHERE B.CITY = 'Seoul'")
  BICYCLES  CITY     COUNTRY     LAT LNG POPULATION
1    20000 Seoul South Korea 37.5833 127   21794000
```

- Seoul, the capital city of South Korea, has a population of 21,794,000 people.
- It is located at a latitude of 37.58° and a longitude of 127°.
- There are 20,000 bikes available for rent in the city.

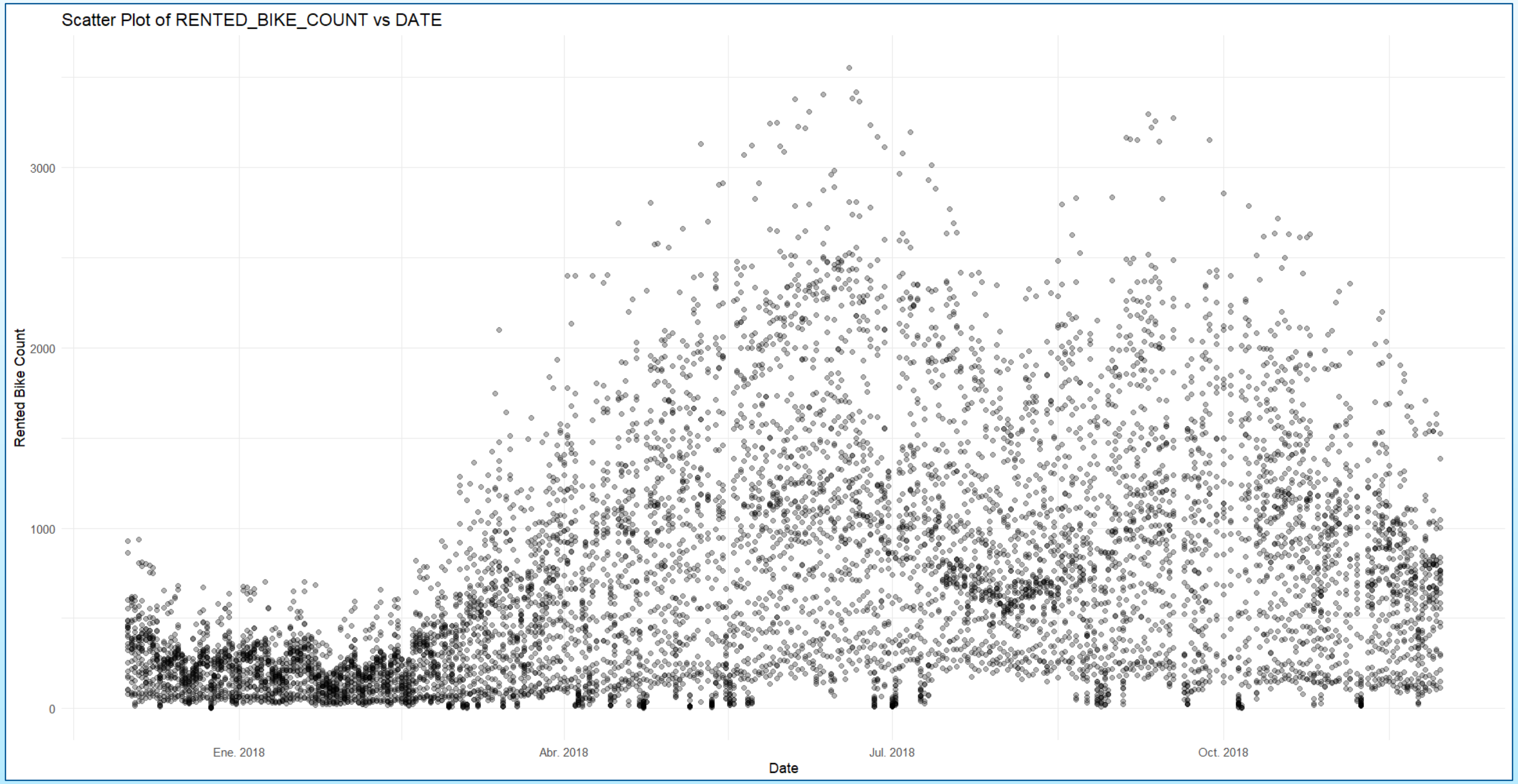
Cities similar to Seoul

```
> dbGetQuery(conn, "
+ SELECT
+     B.BICYCLES,
+     B.CITY,
+     B.COUNTRY,
+     W.LAT,
+     W.LNG,
+     W.POPULATION
+ FROM BIKE_SHARING_SYSTEMS AS B
+ LEFT JOIN WORLD_CITIES AS W
+     ON B.CITY = W.CITY_ASCII
+ WHERE B.CITY = 'Seoul'
+    OR (B.BICYCLES BETWEEN 15000 AND 20000)
+ ORDER BY B.BICYCLES DESC
+ ")
  BICYCLES     CITY     COUNTRY     LAT      LNG POPULATION
1    20000  Kunshan       China      NA       NA         NA
2    20000  Weifang       China 36.7167 119.1000    9373000
3    20000     Xi'an      China 34.2667 108.9000    7135000
4    20000  Zhuzhou       China 27.8407 113.1469    3855609
5    20000     Seoul South Korea 37.5833 127.0000   21794000
6    19165 Shanghai       China 31.1667 121.4667   22120000
7    18000   Xuzhou       China      NA       NA         NA
8    16000  Beijing       China 39.9050 116.3914   19433000
9    15000   Ningbo       China 29.8750 121.5492    7639000
```

Cities in China have bike-sharing systems comparable in scale to Seoul's.

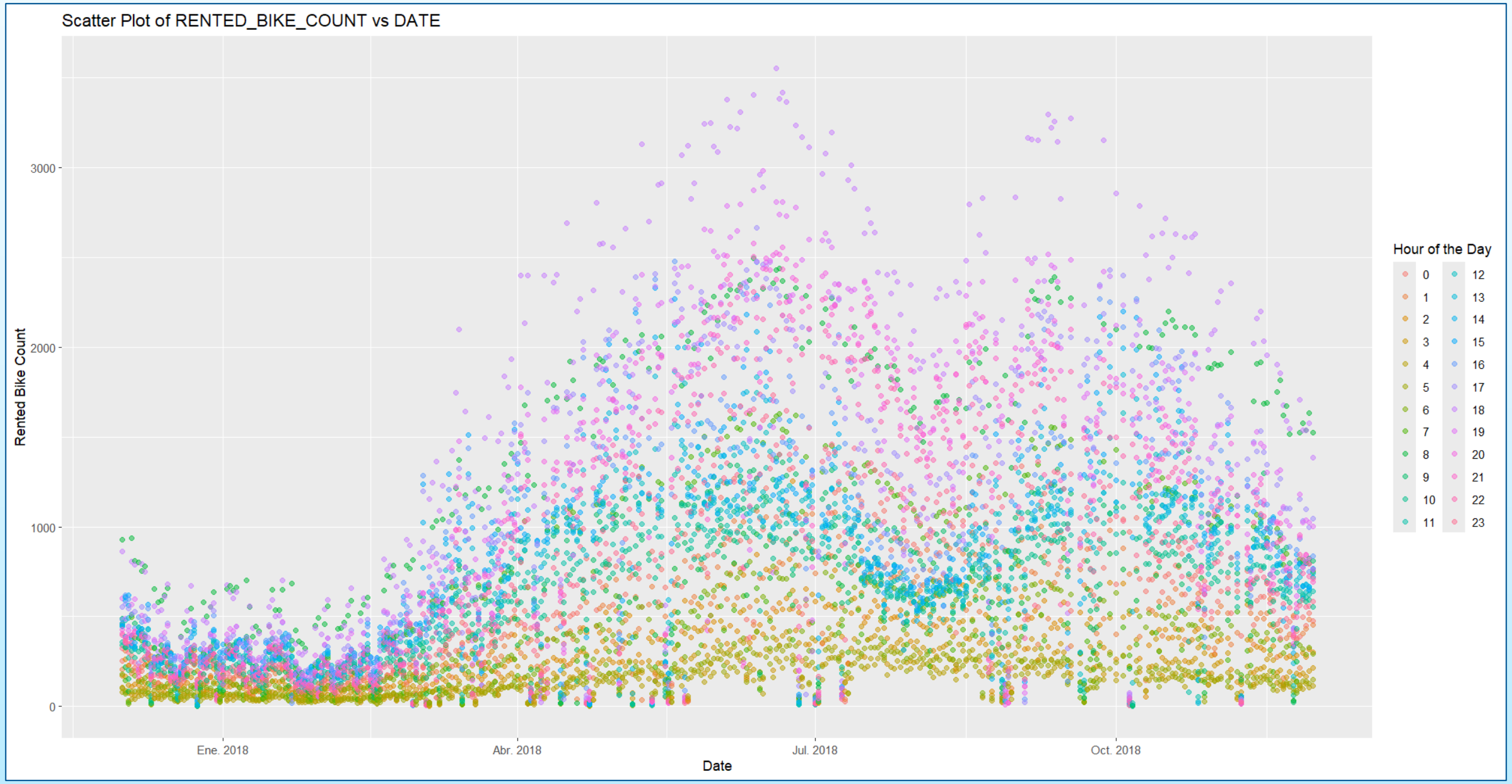
EDA with Visualization

Bike rental vs. Date



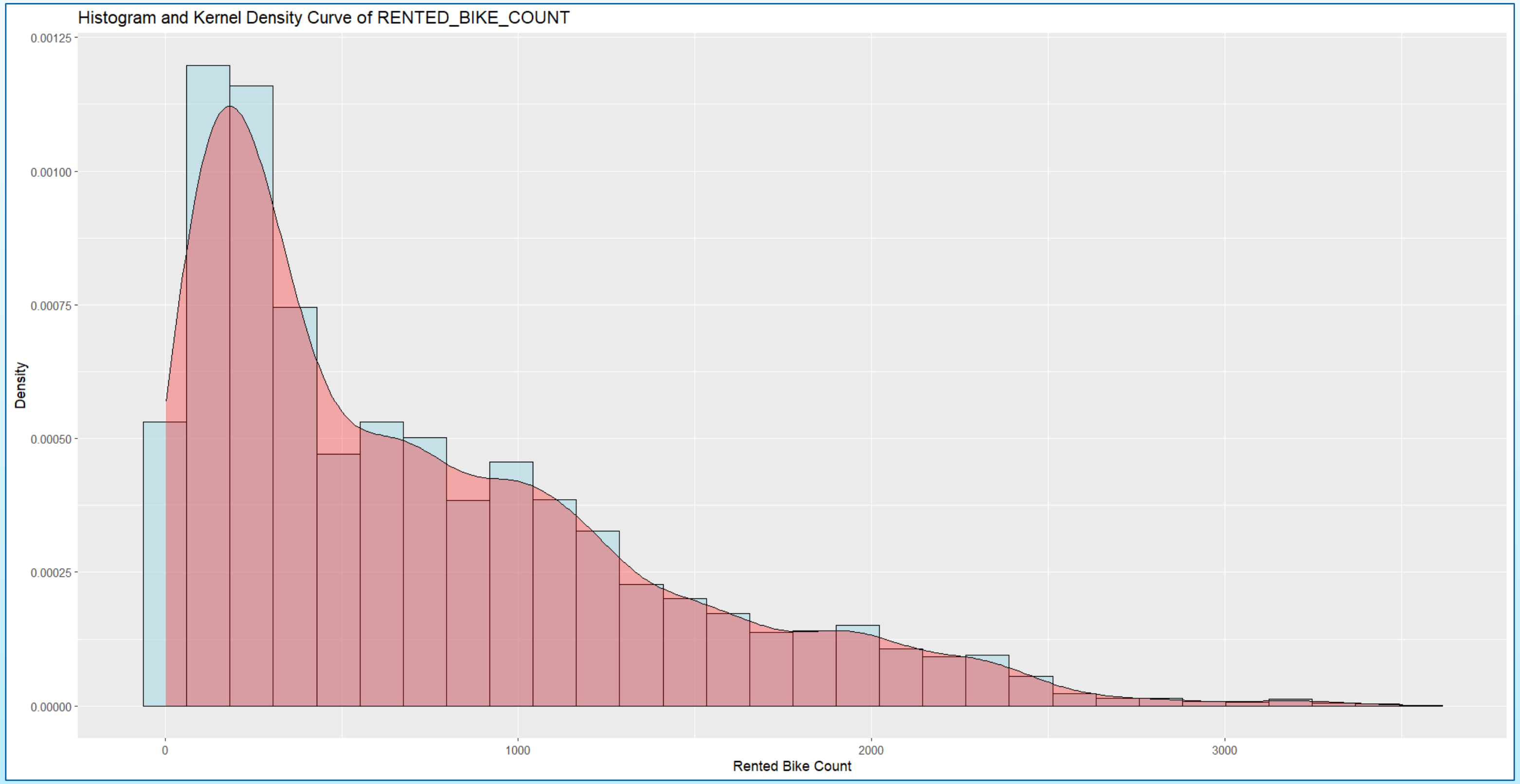
The highest number of bike rentals was recorded from May to October.

Bike rental vs. Datetime



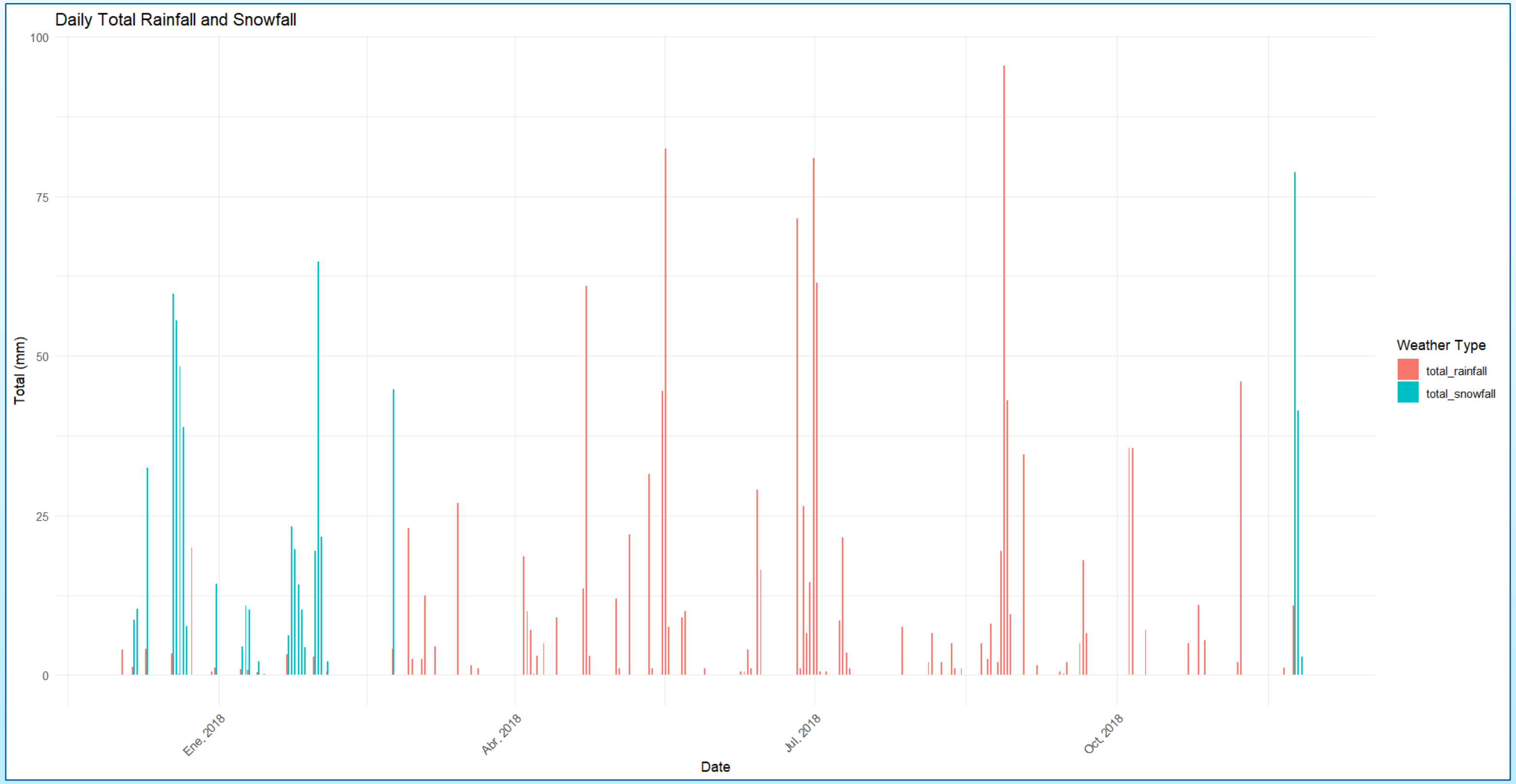
The highest number of bike rentals was recorded from May to October, primarily at 8 AM and between 5 PM and 7 PM.

Bike rental histogram



Generally, fewer than 1,000 bikes are rented, and it is rare for around 3,000 bikes to be rented.

Daily total rainfall and snowfall

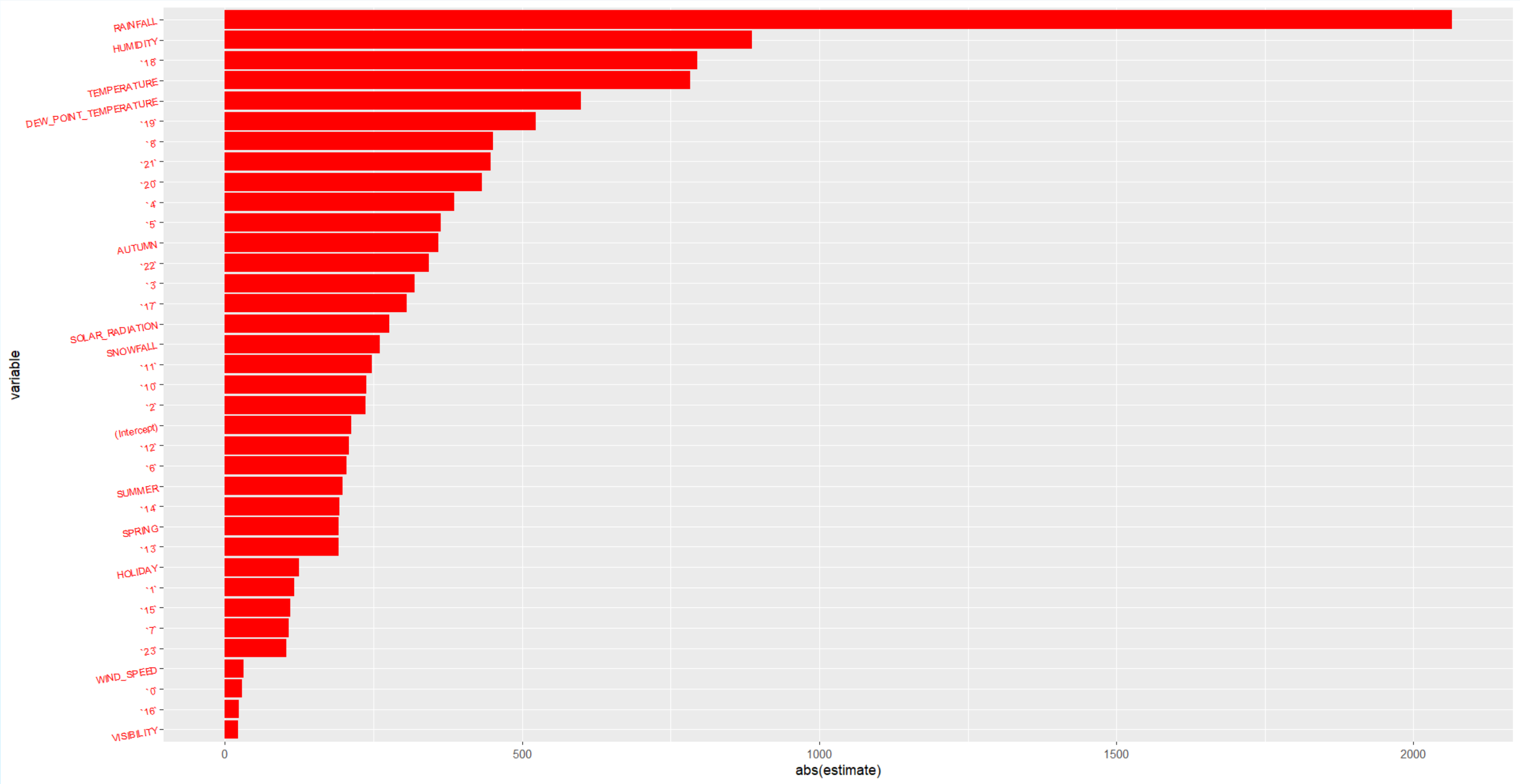


Snowfall primarily occurs during winter days.

Rainfall occurs almost year-round, with its main peaks in June, July, and September.

Predictive analysis

Ranked coefficients

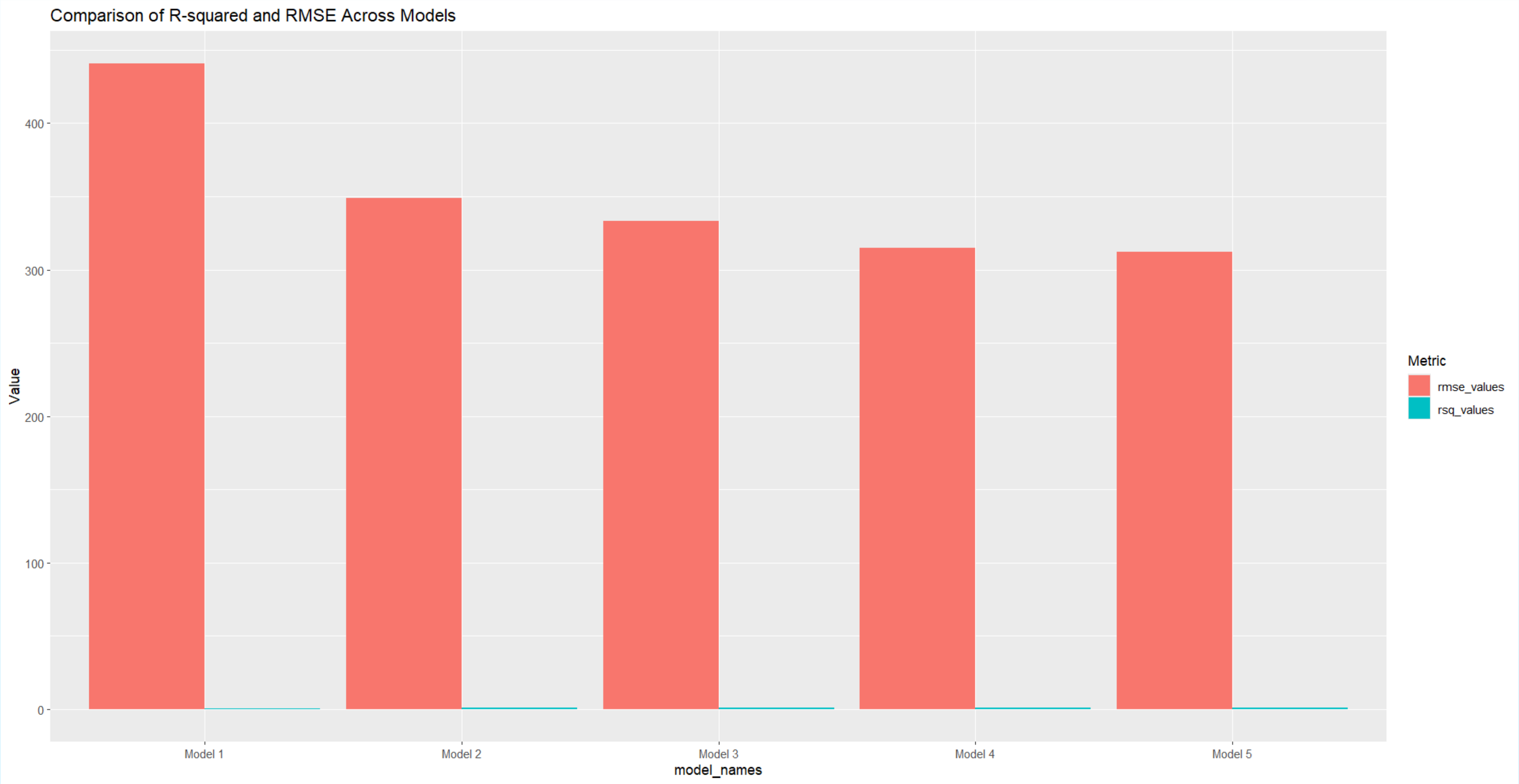


Rainfall, humidity, the 6 PM temperature, and overall temperature are the strongest predictors of bike rentals.

Rainfall and humidity make biking less comfortable and practical.

Temperature, especially at 6 PM—a key time for commuting and leisure—encourages biking when conditions are favorable. Together, these factors highlight the influence of weather on outdoor activity preferences.

Model evaluation

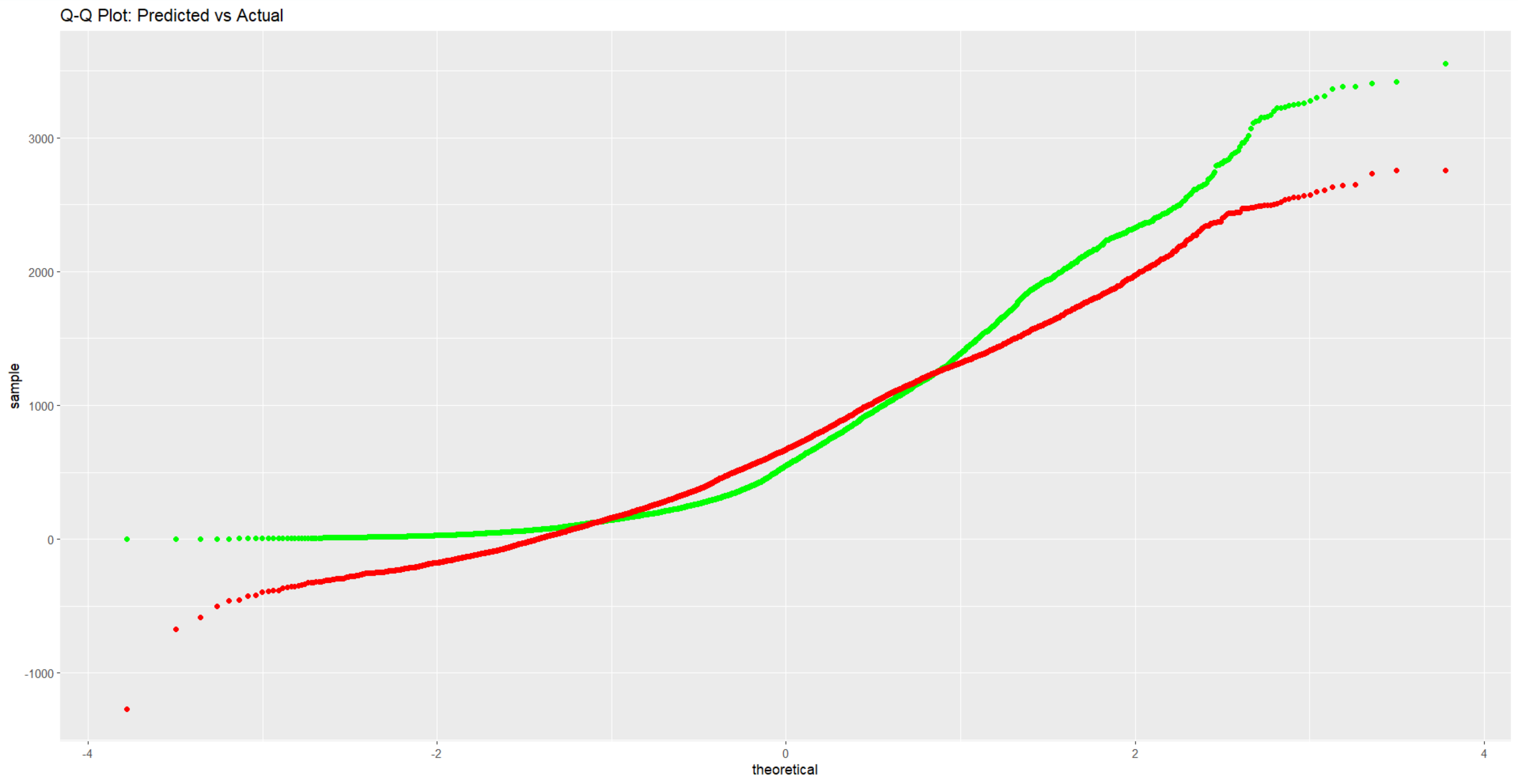


Refined models’ RMSE and R-squared

Find the best performing model

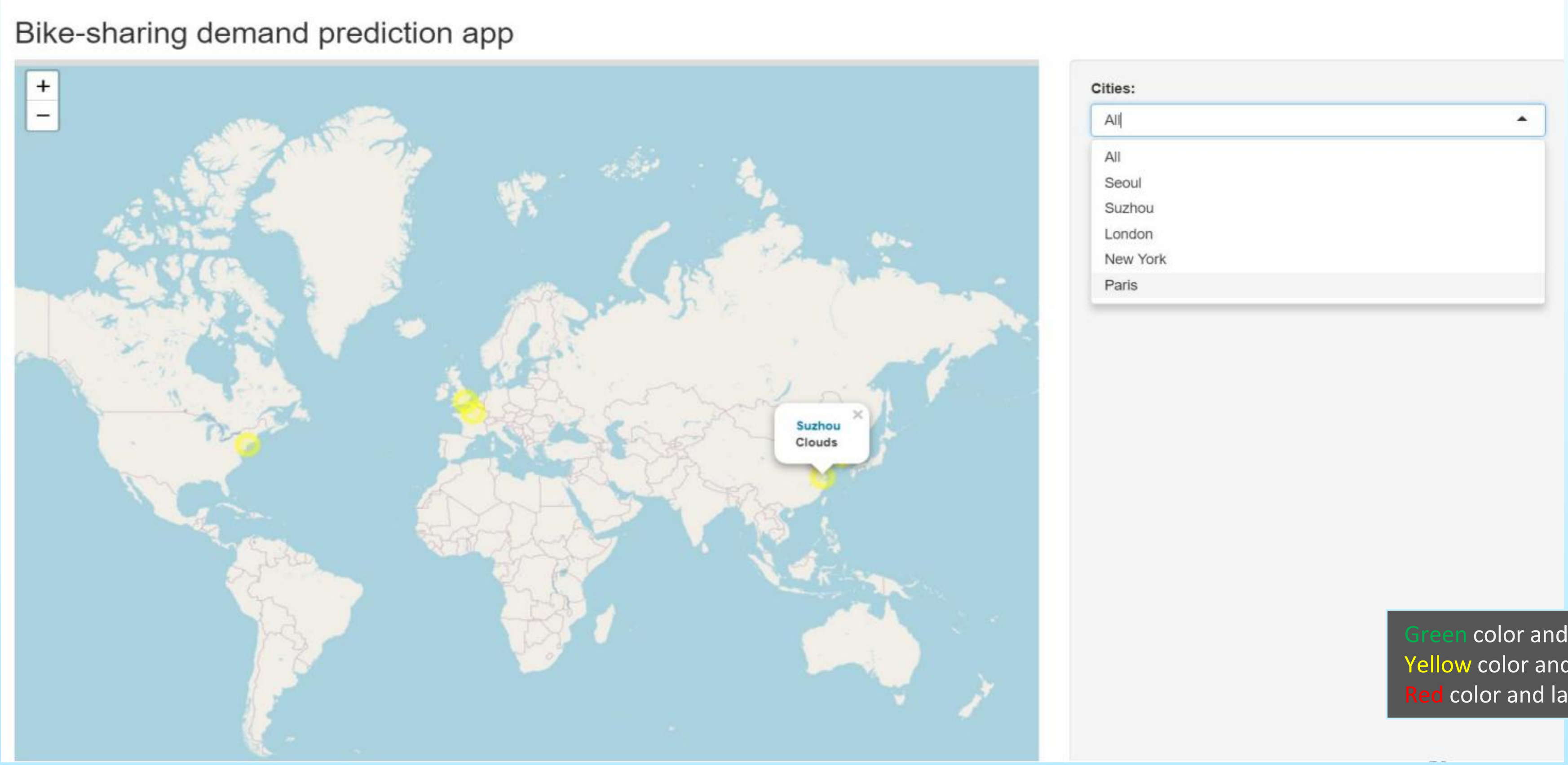
```
> model5_fit <- model5 %>%
+   fit(RENTED_BIKE_COUNT ~ . + poly(TEMPERATURE, 6) + WINTER * `18` + poly(DEW_POINT_TEMPERATURE,
6) + poly(SOLAR_RADIATION, 6) + poly(VISIBILITY, 6) + SUMMER * `18` + TEMPERATURE * HUMIDITY + pol
y(HUMIDITY, 6) + RAINFALL * TEMPERATURE + SNOWFALL * TEMPERATURE + RAINFALL * HUMIDITY + SNOWFALL
* HUMIDITY, data = train_data)
>
> model5_train_results <- model5_fit %>%
+   predict(new_data = train_data) %>%
+   mutate(truth = train_data$RENTED_BIKE_COUNT)
>
> model5$.pred <- replace(model5_train_results$.pred, model5_train_results$.pred < 0, 0)
>
> rsq_model5 <- rsq(model5_train_results, truth = truth, estimate = .pred)
> rmse_model5 <- rmse(model5_train_results, truth = truth, estimate = .pred)
> print(rsq_model5)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rsq     standard       0.766
> print(rmse_model5)
# A tibble: 1 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard        312.
```

Q-Q plot of the best model



Dashboard

Global Bike Demand Prediction

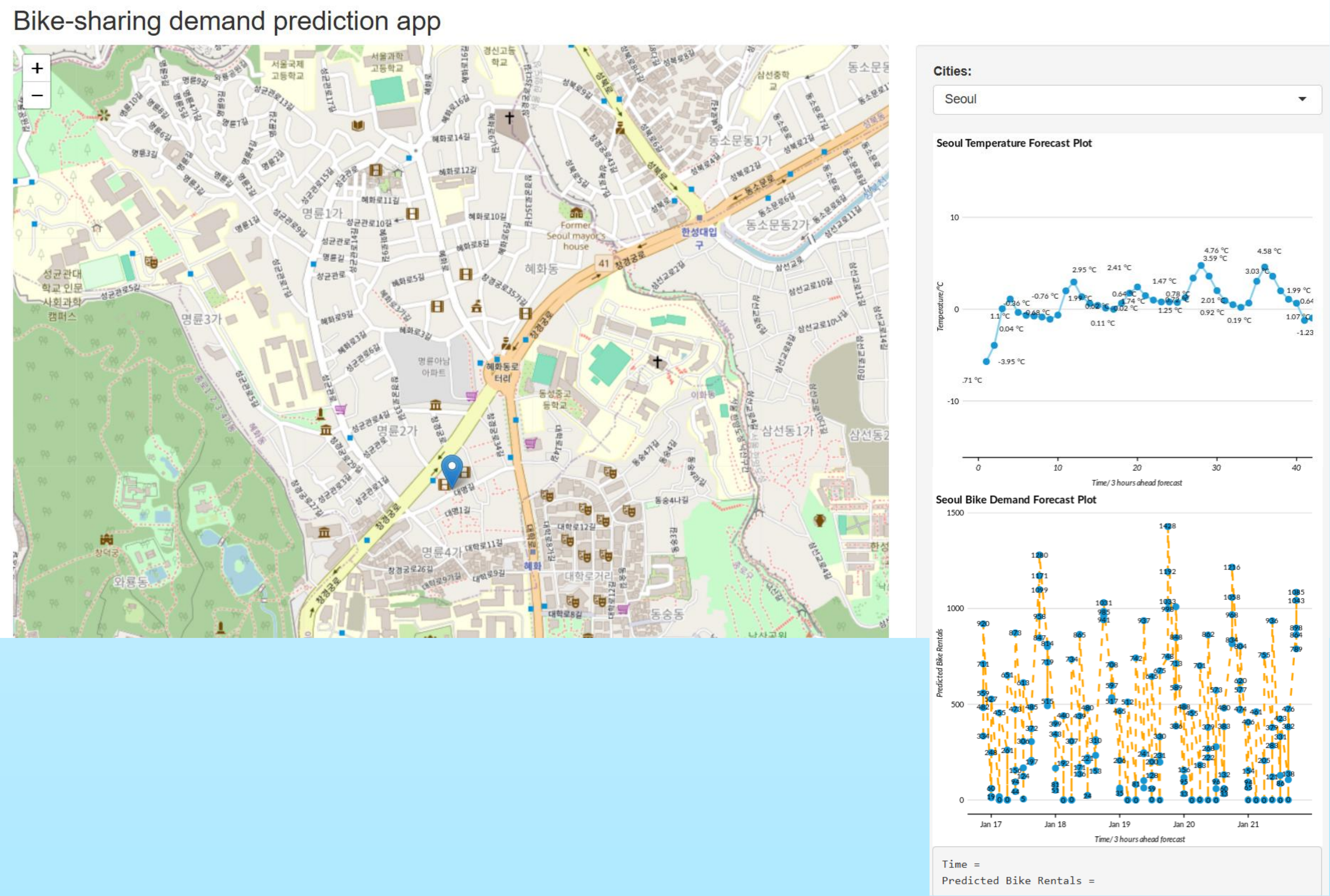


Green color and small radius: low demand
Yellow color and medium radius: normal demand
Red color and large radius: large demand

Map of predicted bike demand for the cities of Seoul, Suzhou, London, New York and Paris.

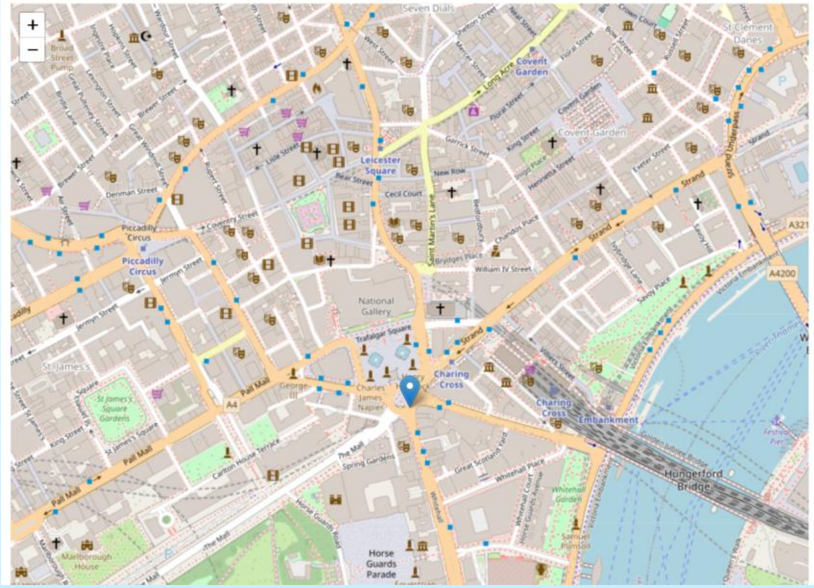
Dashboard City View: Seoul

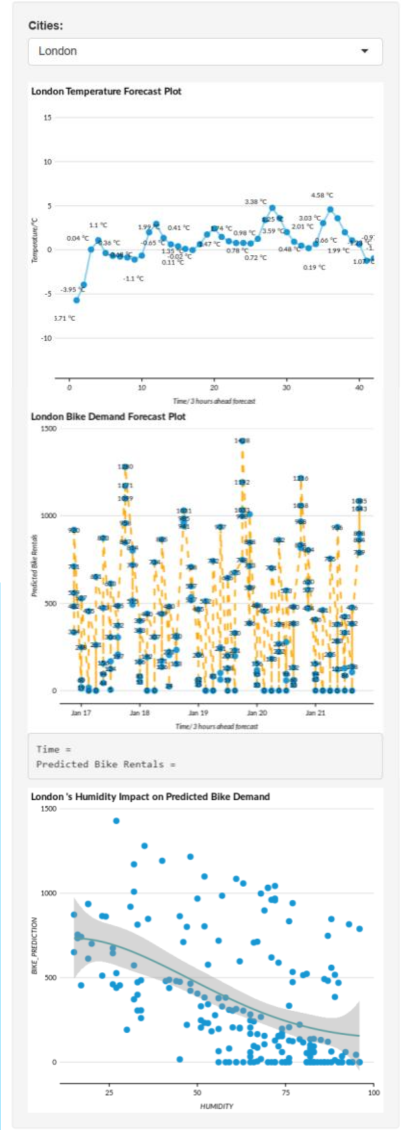
The following screenshot displays Seoul's temperature forecast and bike demand forecast. However, the impact of humidity on predicted bike demand is missing due to technical reasons.



Dashboard City View: London

The following screenshot displays London’s temperature forecast, bike demand forecast, and the impact of humidity on predicted bike demand.





CONCLUSION

- Investigated weather's impact on bike demand in Seoul using Bike Rentals data and refined regression models.
- Best model (GLM with Regularization) explained 77% of demand variation ($R^2 = 0.766$, RMSE = 312).
- Key findings: summer had highest demand, winter the lowest; extreme temperatures reduced bike use.
- Humidity, rainfall, and peak hours (6 PM, 8 AM) significantly influenced demand.
- Created a map-based R-Shiny dashboard for five cities, showing temperature, bike demand, and humidity impact forecasts.
- Findings help city planners manage fleets and highlight areas for further research, including fuel prices and cultural factors.

APPENDIX: Webscraping

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

root_node <-read_html(table_url)

table_nodes <- html_node(root_node, "table")

# Extract content from table_node and convert the data into a dataframe
data_frame <- html_table(table_nodes)
head(data_frame)
tail(data_frame)
names(data_frame)

# Export the dataframe into a csv file
write.csv(data_frame, file = "C:/Users/Amelia/Desktop/IBM CERTIFICATE/9 Capstone Project/raw_bike_sharing_systems.csv", row.names = FALSE)
```

APPENDIX: OpenWeather API call

```r
your_api_key <- "a393f0ea9de1d0a60106be937226fd46"
# Create some empty vectors to hold data temporarily
city <- c()
weather <- c()
visibility <- c()
temp <- c()
temp_min <- c()
temp_max <- c()
pressure <- c()
humidity <- c()
wind_speed <- c()
wind_deg <- c()
forecast_datetime <- c()

# Get 5 -day weather forecast for a list of cities
get_weather_forecast_by_cities <- function(city_names) {
  df <- data.frame()
  for (city_name in city_names) {
    #forecast API URL
    forecast_url <-'https://api.openweathermap.org/data/2.5/weather'
    #create query parameter
    forecast_query <- list(q=city_name,appid=your_api_key, units="metric")
    #make HTTP GET call for the given city
    response <- GET(forecast_url, query=forecast_query)
    json_result <- content(response, as="parsed")
    results <- json_result$list
    #Loop the json result
    for(result in results) {
      city <- c(city, city_name)
    }
    # Add R lists into a data frame
    city <- c(city, json_result$name)
    weather <- c(weather, json_result$weather[[1]]$main)
    visibility <- c(visibility, json_result$visibility)
    temp <- c(temp, json_result$main$temp)
    temp_min <-c(temp_min, json_result$main$temp_min)
    temp_max <- c(temp_max, json_result$main$temp_max)
    pressure <- c(pressure, json_result$main$pressure)
    humidity <- c(humidity, json_result$main$humidity)
    wind_speed <- c(wind_speed, json_result$wind$speed)
    wind_deg <-c(wind_deg, json_result$wind$deg)
    forecast_datetime <- c(forecast_datetime,json_result$dt)

    #Combine all vector into data frame
    df <- data.frame(city = city,
                     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,
                     forecast_datetime=forecast_datetime)
  }
  return(df)
}

cities <- c("Seoul", "Washington, D.C.", "Paris", "Suzhou")
cities_weather_df <- get_weather_forecast_by_cities(cities)
print(cities_weather_df)

# Write cities_weather_df to `cities_weather_forecast.csv`
write.csv(cities_weather_df, file = "C:/Users/Amelia/Desktop/IBM CERTIFICATE/9 Capstone Project/cities_weather_df.csv", row.names=FALSE)
```

APPENDIX: Data Wrangling – Regular Expressions

```r
dataset_list <- c('raw_bike_sharing_systems.csv', 'raw_seoul_bike_sharing.csv', 'raw_cities_weather_forecast.csv', 'raw_worldcities.csv')

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)
}


# Print a summary for each data set to check whether the column names were correctly converted

for (dataset_name in dataset_list) {
  dataset <- read_csv(dataset_name)
  print(colnames(dataset))
}

####TASK: Remove undesired reference links from the scraped bike-sharing systems dataset

# First load the dataset
bike_sharing_df <- read_csv("raw_bike_sharing_systems.csv")
head(bike_sharing_df)

# Select the four columns
sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)
#Types of the selected columns
sub_bike_sharing_df %>%
  summarize_all(class) %>%
  gather(variable, class)

#ABOUT COLUMN BICLYCLE:
#Let's see why it wasn't loaded as a numeric column:possibly some entries contain characters.
# grepl searches a string for non-digital characters, and returns TRUE or FALSE
# if it finds any non-digital characters, then the bicyle column is not purely numeric
find_character <- function(strings) grepl("[^0-9]", strings)

#To find any elements in the Bicycles column containing non-numeric characters
sub_bike_sharing_df %>%
  select(BICYCLES) %>%
  filter(find_character(BICYCLES)) %>%
  slice(0:10)
#RESULT: many rows have non-numeric characters

# 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)
#RESULT: COUNTRY IS CLEAN

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

```r
#RESULT: CITY HAS REFERENCE LINKS TO BE REMOVED

# Check whether the System column has any reference links
sub_bike_sharing_df %>%
  select(SYSTEM) %>%
  filter(find_reference_pattern(SYSTEM)) %>%
  slice(0:10)
#RESULT: SYSTEM HAS REFERENCE LINKS TO BE REMOVED

##CONCLUSION:
#CITY and SYSTEM columns have some undesired reference links
#BICYCLES column has both reference links and some textual annotations.


####TASK: Remove undesired reference links using regular expressions

# To replace all reference links with an empty character for columns CITY and SYSTEM
# Define the remove_ref function
remove_ref <- function(strings) {
  # Pattern to match reference links, e.g., [1], [2], etc.
  ref_pattern <- "\\[[\\d+\\]"  # Matches text like [1], [23], etc.

  # Replace all matched substrings with a white space using str_replace_all
  result <- str_replace_all(strings, ref_pattern, "")

  # Trim the result to remove any unnecessary spaces
  result <- str_trim(result)

  # Return the cleaned string
  return(result)
}

# Use the function to remove the reference links
# Apply remove_ref to CITY and SYSTEM columns
sub_bike_sharing_df <- sub_bike_sharing_df %>%
  mutate(SYSTEM = remove_ref(SYSTEM),
         CITY = remove_ref(CITY))
# Select specific columns and filter rows with references
result <- sub_bike_sharing_df %>%
  select(CITY, SYSTEM, BICYCLES) %>%
  filter(find_reference_pattern(CITY) |
           find_reference_pattern(SYSTEM) |
           find_reference_pattern(BICYCLES))

# Print the result to check if any references remain
print(result)
head(result)


####TASK: Extract the numeric value using regular expressions

# Extract the first number
extract_num <- function(columns){

  # Define a digital pattern
  digitals_pattern <- "\\d+"  # Matches any sequence of digits

  # Find the first match using str_extract
  result <- str_extract(columns, digitals_pattern)

  # Convert the result to numeric using the as.numeric() function
  result <- as.numeric(result)

  # Return the numeric result
  return(result)
}

# Use the mutate() function on the BICYCLES column
sub_bike_sharing_df <- sub_bike_sharing_df %>%
  mutate(BICYCLES = extract_num(BICYCLES))

summary(sub_bike_sharing_df$BICYCLES)

# Write the dataset to a CSV file
write.csv(sub_bike_sharing_df, "C:/Users/Amelia/Desktop/IBM CERTIFICATE/9 Capstone Project/bike_sharing_systems.csv", row.names = FALSE)
```

APPENDIX: Data Wrangling – Regular Expressions

```r
bike_sharing_df <- read_csv("raw_seoul_bike_sharing.csv")
head(bike_sharing_df)

summary(bike_sharing_df)
dim(bike_sharing_df)
map(bike_sharing_df, ~sum(is.na(.))) #RENTED_BIKE_COUNT HAS 295 NAs

####TASK: Detect and handle missing values

# Drop rows with `RENTED_BIKE_COUNT` column == NA
bike_sharing_df <- bike_sharing_df %>% drop_na(RENTED_BIKE_COUNT)
# Print the dataset dimension again after those rows are dropped
dim(bike_sharing_df)

#TEMPERATURE
bike_sharing_df %>%
  filter(is.na(TEMPERATURE))
#RESULT: NAs related to SUMMER

# Calculate the summer average temperature
summer_temp <- bike_sharing_df[bike_sharing_df$SEASONS == "Summer", ] #[rows with summer, empty cuz all columns are in]
summer_avg_temp <- mean(summer_temp$TEMPERATURE, na.rm=TRUE)
print(summer_avg_temp)

# Impute missing values for TEMPERATURE column with summer average temperature
bike_sharing_df <- bike_sharing_df %>%
  mutate(TEMPERATURE = replace_na(TEMPERATURE, summer_avg_temp))

# Print the summary of the dataset again to make sure no missing values in all columns
summary(bike_sharing_df)

# Save the dataset as `seoul_bike_sharing.csv`
write.csv(bike_sharing_df, "C:/Users/Amelia/Desktop/IBM CERTIFICATE/9 Capstone Project/seoul_bike_sharing.csv", row.names = FALSE)
####TASK: Create indicator (dummy) variables for categorical variables

#HOUR
bike_sharing_df <-bike_sharing_df %>%
  mutate(HOUR = as.character(HOUR))
class(bike_sharing_df$HOUR)

#SEASONS
bike_sharing_df$SEASONS <- factor(bike_sharing_df$SEASONS)
#Create dummy columns for each season using `mutate()` and `spread()`
bike_sharing_df <- bike_sharing_df %>%
  mutate(dummy = 1) %>%  # Create a dummy column with 1
  spread(
    key = SEASONS,    # Spread the SEASONS values into separate columns
    value = dummy,    # Use the dummy column for spreading
    fill = 0          # If a season is missing, fill with 0
  )


#HOLIDAY
bike_sharing_df$HOLIDAY <- factor(bike_sharing_df$HOLIDAY)
bike_sharing_df <- bike_sharing_df %>%
  mutate(dummy = 1) %>%  # Create a dummy column with 1
  spread(
    key = HOLIDAY,    # Spread the SEASONS values into separate columns
    value = dummy,    # Use the dummy column for spreading
    fill = 0          # If a season is missing, fill with 0
  )

#FUNCTIONING_DAY
summary(bike_sharing_df$FUNCTIONING_DAY)
bike_sharing_df$FUNCTIONING_DAY <- factor(bike_sharing_df$FUNCTIONING_DAY)

# Print the dataset summary again to make sure the indicator columns are created properly
summary(bike_sharing_df)

# Save the dataset as `seoul_bike_sharing_converted.csv`
write.csv(bike_sharing_df, "C:/Users/Amelia/Desktop/IBM CERTIFICATE/9 Capstone Project/seoul_bike_sharing_converted.csv", row.names = FALSE)
```

```r
####TASK: Normalize data

# Apply min-max normalization for each specified column
SCALED_bike_sharing_df <- bike_sharing_df %>%
  mutate(
    RENTED_BIKE_COUNT = (RENTED_BIKE_COUNT - min(RENTED_BIKE_COUNT)) / (max(RENTED_BIKE_COUNT) - min(RENTED_BIKE_COUNT)),
    TEMPERATURE = (TEMPERATURE - min(TEMPERATURE)) / (max(TEMPERATURE) - min(TEMPERATURE)),
    HUMIDITY = (HUMIDITY - min(HUMIDITY)) / (max(HUMIDITY) - min(HUMIDITY)),
    WIND_SPEED = (WIND_SPEED - min(WIND_SPEED)) / (max(WIND_SPEED) - min(WIND_SPEED)),
    VISIBILITY = (VISIBILITY - min(VISIBILITY)) / (max(VISIBILITY) - min(VISIBILITY)),
    DEW_POINT_TEMPERATURE = (DEW_POINT_TEMPERATURE - min(DEW_POINT_TEMPERATURE)) / (max(DEW_POINT_TEMPERATURE) - min(DEW_POINT_TEMPERATURE)),
    SOLAR_RADIATION = (SOLAR_RADIATION - min(SOLAR_RADIATION)) / (max(SOLAR_RADIATION) - min(SOLAR_RADIATION)),
    RAINFALL = (RAINFALL - min(RAINFALL)) / (max(RAINFALL) - min(RAINFALL)),
    SNOWFALL = (SNOWFALL - min(SNOWFALL)) / (max(SNOWFALL) - min(SNOWFALL))
  )

summary(SCALED_bike_sharing_df$RENTED_BIKE_COUNT)
summary(SCALED_bike_sharing_df$TEMPERATURE)

# Save the dataset as `seoul_bike_sharing_converted_normalized.csv`
write.csv(bike_sharing_df, "C:/Users/Amelia/Desktop/IBM CERTIFICATE/9 Capstone Project/seoul_bike_sharing_converted_normalized.csv", row.names = FALSE)


#### Standardize the column names again for the new datasets

dataset_list <- c('seoul_bike_sharing.csv', 'seoul_bike_sharing_converted.csv', 'seoul_bike_sharing_converted_normalized.csv')

for (dataset_name in dataset_list){
  # Read dataset
  dataset <- read_csv(dataset_name)
  # Standardized its columns:
  # Convert all columns names to uppercase
  names(dataset) <- toupper(names(dataset))
  # Replace any white space separators by underscore, using str_replace_all function
  names(dataset) <- str_replace_all(names(dataset), " ", "_")
  # Save the dataset back
  write.csv(dataset, dataset_name, row.names=FALSE)
}
```

APPENDIX: EDA - SQL

```r
#Establish Connection
conn <- dbConnect(RSQLite::SQLite(),"RDB.sqlite")

#Read Datasets
SEOUL_BIKE_SHARING <- read_csv("seoul_bike_sharing.csv")
CITIES_WEATHER_FORECAST <- read_csv("cities_weather_forecast.csv")
BIKE_SHARING_SYSTEMS <- read_csv("bike_sharing_systems.csv")
WORLD_CITIES <- read_csv("world_cities.csv")

#Load Tables
dbWriteTable(conn, "SEOUL_BIKE_SHARING", SEOUL_BIKE_SHARING, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "CITIES_WEATHER_FORECAST", CITIES_WEATHER_FORECAST, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "BIKE_SHARING_SYSTEMS", BIKE_SHARING_SYSTEMS, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "WORLD_CITIES", WORLD_CITIES, overwrite=TRUE, header = TRUE)
dbListTables(conn)


#### TASK 1: Determine how many records are in the seoul_bike_sharing dataset.
dbGetQuery(conn, 'SELECT COUNT(*) AS Records FROM SEOUL_BIKE_SHARING')

#### TASK 2: Determine how many hours had non-zero rented bike count.
dbGetQuery(conn, "SELECT count(HOUR) as Numer_of_hours FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT > 0")

#### TASK 3: Query the the weather forecast for Seoul over the next 3 hours.
#Recall that the records in the CITIES_WEATHER_FORECAST dataset are 3 hours apart, so we just need the first record from the quer

dbGetQuery(conn, "SELECT * FROM CITIES_WEATHER_FORECAST
WHERE CITY = 'Seoul'
Limit 1")

#### TASK 4: Find which seasons are included in the seoul bike sharing dataset.
dbGetQuery(conn, "SELECT distinct SEASONS as Seasons
FROM SEOUL_BIKE_SHARING")

#### TASK 5: Find the first and last dates in the Seoul Bike Sharing dataset.
dbGetQuery(conn, "SELECT MIN(DATE) as Start_Date, MAX(DATE) as End_Date
FROM SEOUL_BIKE_SHARING")
#### TASK 6: Determine which date and hour had the most bike rentals.
dbGetQuery(conn, "
SELECT DATE, HOUR, RENTED_BIKE_COUNT AS Maximum_COUNT
FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)")

#### TASK 7: 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.
dbGetQuery(conn, "SELECT SEASONS, HOUR, AVG(RENTED_BIKE_COUNT), AVG(TEMPERATURE)
          FROM SEOUL_BIKE_SHARING
          GROUP BY SEASONS, HOUR
          ORDER BY AVG(RENTED_BIKE_COUNT) DESC
          LIMIT 10")

#### TASK 8: Find the average hourly bike count during each season.
#Include:  min, max, and sd of the hourly bike count for each season
dbGetQuery(conn, "
SELECT
    SEASONS,
    AVG(RENTED_BIKE_COUNT) AS Avg_Bike_Count,
    MIN(RENTED_BIKE_COUNT) AS Min_Bike_Count,
    MAX(RENTED_BIKE_COUNT) AS Max_Bike_Count,
    SQRT(AVG(RENTED_BIKE_COUNT * RENTED_BIKE_COUNT) - AVG(RENTED_BIKE_COUNT) * AVG(RENTED_BIKE_COUNT)) AS Std_Dev_Bike_Count
FROM SEOUL_BIKE_SHARING
GROUP BY SEASONS")
```

```r
#### TASK 9: Consider the weather over each season.
#On avg, what were the TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, and SNOWFALL per season?
#Include the average bike count as well , and rank the results by average bike count
dbGetQuery(conn, "
SELECT
    SEASONS,
    AVG(TEMPERATURE) AS Avg_Temperature,
    AVG(HUMIDITY) AS Avg_Humidity,
    AVG(WIND_SPEED) AS Avg_WindSpeed,
    AVG(VISIBILITY) AS Avg_Visibility,
    AVG(DEW_POINT_TEMPERATURE) AS Avg_DewPointTemp,
    AVG(SOLAR_RADIATION) AS Avg_SolarRad,
    AVG(RAINFALL) AS Avg_Rainfall,
    AVG(SNOWFALL) AS Avg_Snowfall,
    AVG(RENTED_BIKE_COUNT) AS Avg_RentedBikes
FROM SEOUL_BIKE_SHARING
GROUP BY SEASONS
ORDER BY AVG(RENTED_BIKE_COUNT) DESC")

#### TASK 10: Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes avaialble
dbGetQuery(conn, "SELECT B.BICYCLES, B.CITY, B.COUNTRY, W.LAT, W.LNG, W.POPULATION
FROM BIKE_SHARING_SYSTEMS AS B
LEFT JOIN WORLD_CITIES AS W
ON B.CITY = W.CITY_ASCII
WHERE B.CITY = 'Seoul'")

#TASK 11: 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.
dbGetQuery(conn, "
SELECT
    B.BICYCLES,
    B.CITY,
    B.COUNTRY,
    W.LAT,
    W.LNG,
    W.POPULATION
FROM BIKE_SHARING_SYSTEMS AS B
LEFT JOIN WORLD_CITIES AS W
    ON B.CITY = W.CITY_ASCII
WHERE B.CITY = 'Seoul'
   OR (B.BICYCLES BETWEEN 15000 AND 20000)
ORDER BY B.BICYCLES DESC
")

dbListTables(conn)

# Once you're done with the connection, close it
dbDisconnect(conn)
```

APPENDIX: EDA - SQL

```r
#Establish Connection
conn <- dbConnect(RSQLite::SQLite(),"RDB.sqlite")

#Read Datasets
SEOUL_BIKE_SHARING <- read_csv("seoul_bike_sharing.csv")
CITIES_WEATHER_FORECAST <- read_csv("cities_weather_forecast.csv")
BIKE_SHARING_SYSTEMS <- read_csv("bike_sharing_systems.csv")
WORLD_CITIES <- read_csv("world_cities.csv")

#Load Tables
dbWriteTable(conn, "SEOUL_BIKE_SHARING", SEOUL_BIKE_SHARING, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "CITIES_WEATHER_FORECAST", CITIES_WEATHER_FORECAST, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "BIKE_SHARING_SYSTEMS", BIKE_SHARING_SYSTEMS, overwrite=TRUE, header = TRUE)
dbWriteTable(conn, "WORLD_CITIES", WORLD_CITIES, overwrite=TRUE, header = TRUE)
dbListTables(conn)


#### TASK 1: Determine how many records are in the seoul_bike_sharing dataset.
dbGetQuery(conn, 'SELECT COUNT(*) AS Records FROM SEOUL_BIKE_SHARING')

#### TASK 2: Determine how many hours had non-zero rented bike count.
dbGetQuery(conn, "SELECT count(HOUR) as Numer_of_hours FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT > 0")

#### TASK 3: Query the the weather forecast for Seoul over the next 3 hours.
#Recall that the records in the CITIES_WEATHER_FORECAST dataset are 3 hours apart, so we just need the first record from the quer

dbGetQuery(conn, "SELECT * FROM CITIES_WEATHER_FORECAST
WHERE CITY = 'Seoul'
Limit 1")

#### TASK 4: Find which seasons are included in the seoul bike sharing dataset.
dbGetQuery(conn, "SELECT distinct SEASONS as Seasons
FROM SEOUL_BIKE_SHARING")

#### TASK 5: Find the first and last dates in the Seoul Bike Sharing dataset.
dbGetQuery(conn, "SELECT MIN(DATE) as Start_Date, MAX(DATE) as End_Date
FROM SEOUL_BIKE_SHARING")
#### TASK 6: Determine which date and hour had the most bike rentals.
dbGetQuery(conn, "
SELECT DATE, HOUR, RENTED_BIKE_COUNT AS Maximum_COUNT
FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)")

#### TASK 7: 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.
dbGetQuery(conn, "SELECT SEASONS, HOUR, AVG(RENTED_BIKE_COUNT), AVG(TEMPERATURE)
          FROM SEOUL_BIKE_SHARING
          GROUP BY SEASONS, HOUR
          ORDER BY AVG(RENTED_BIKE_COUNT) DESC
          LIMIT 10")

#### TASK 8: Find the average hourly bike count during each season.
#Include:  min, max, and sd of the hourly bike count for each season
dbGetQuery(conn, "
SELECT
    SEASONS,
    AVG(RENTED_BIKE_COUNT) AS Avg_Bike_Count,
    MIN(RENTED_BIKE_COUNT) AS Min_Bike_Count,
    MAX(RENTED_BIKE_COUNT) AS Max_Bike_Count,
    SQRT(AVG(RENTED_BIKE_COUNT * RENTED_BIKE_COUNT) - AVG(RENTED_BIKE_COUNT) * AVG(RENTED_BIKE_COUNT)) AS Std_Dev_Bike_Count
FROM SEOUL_BIKE_SHARING
GROUP BY SEASONS")
```

```r
#### TASK 9: Consider the weather over each season.
#On avg, what were the TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, and SNOWFALL per season?
#Include the average bike count as well , and rank the results by average bike count
dbGetQuery(conn, "
SELECT
    SEASONS,
    AVG(TEMPERATURE) AS Avg_Temperature,
    AVG(HUMIDITY) AS Avg_Humidity,
    AVG(WIND_SPEED) AS Avg_WindSpeed,
    AVG(VISIBILITY) AS Avg_Visibility,
    AVG(DEW_POINT_TEMPERATURE) AS Avg_DewPointTemp,
    AVG(SOLAR_RADIATION) AS Avg_SolarRad,
    AVG(RAINFALL) AS Avg_Rainfall,
    AVG(SNOWFALL) AS Avg_Snowfall,
    AVG(RENTED_BIKE_COUNT) AS Avg_RentedBikes
FROM SEOUL_BIKE_SHARING
GROUP BY SEASONS
ORDER BY AVG(RENTED_BIKE_COUNT) DESC")

#### TASK 10: Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes avaialble
dbGetQuery(conn, "SELECT B.BICYCLES, B.CITY, B.COUNTRY, W.LAT, W.LNG, W.POPULATION
FROM BIKE_SHARING_SYSTEMS AS B
LEFT JOIN WORLD_CITIES AS W
ON B.CITY = W.CITY_ASCII
WHERE B.CITY = 'Seoul'")

#TASK 11: 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.
dbGetQuery(conn, "
SELECT
    B.BICYCLES,
    B.CITY,
    B.COUNTRY,
    W.LAT,
    W.LNG,
    W.POPULATION
FROM BIKE_SHARING_SYSTEMS AS B
LEFT JOIN WORLD_CITIES AS W
    ON B.CITY = W.CITY_ASCII
WHERE B.CITY = 'Seoul'
   OR (B.BICYCLES BETWEEN 15000 AND 20000)
ORDER BY B.BICYCLES DESC
")

dbListTables(conn)

# Once you're done with the connection, close it
dbDisconnect(conn)
```