E-Bike Sharing Demand Prediction in R

Amruta Ghube

12/12/2023

Outline

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

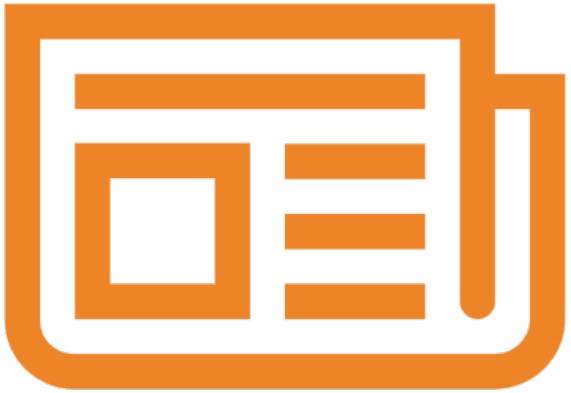
Executive Summary

- Gathered relevant datasets for bike-sharing demand analysis.
- Processed and cleaned the collected data to prepare it for analysis.
- Utilized SQL and visualization techniques to explore patterns, trends, and relationships in the data.
- Employed regression models to predict bike-sharing demand, starting with a baseline model.
- Created an initial model for prediction, establishing a starting point for further enhancements.
- Investigated methods to refine and improve the predictive models for more accurate forecasts.
- Developed an interactive R Shiny dashboard application to visualize and interact with the bike-sharing demand predictions.

Introduction

- Data Collection
- Data Wrangling
- Predictive modeling
- Trend Visualization
  - City Level Insights
  - Correlation Analysis
- Drill-Down Functionality

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

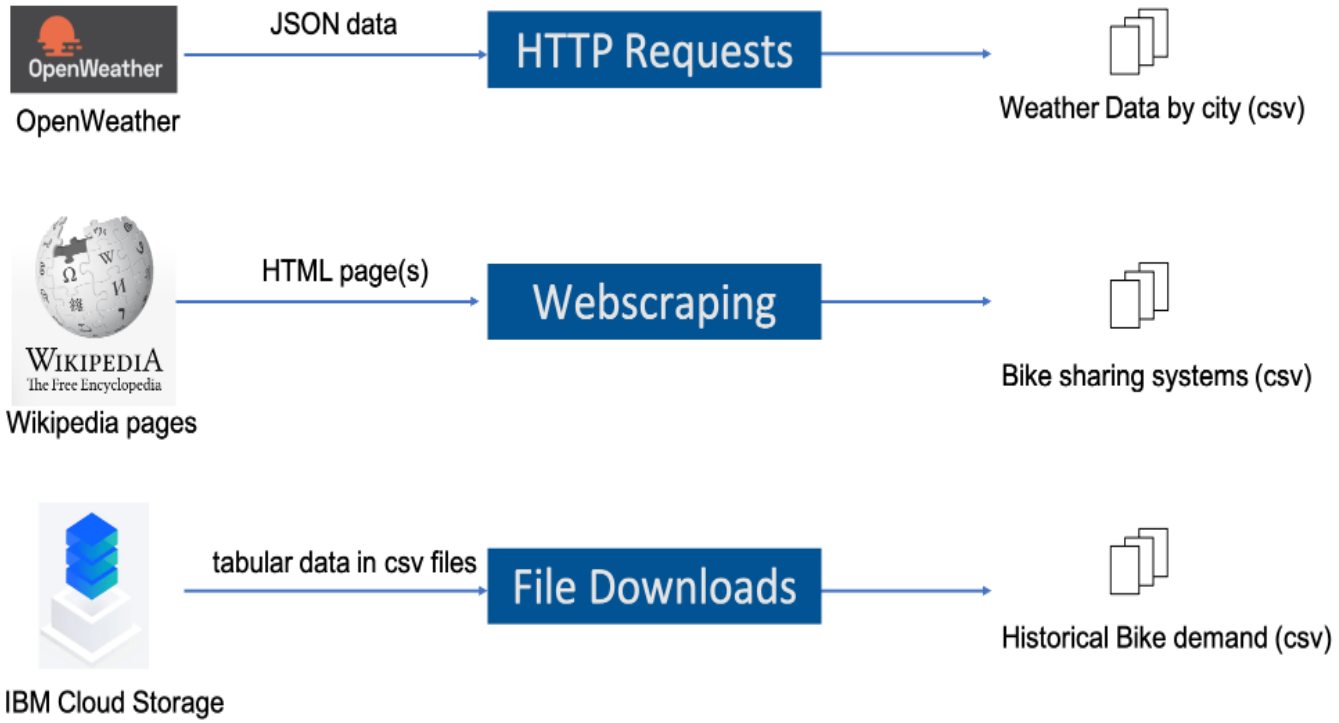
Methodology

Data collection

- Methods used for Data collection:
    1. Data Collection with Web Scraping – Extracted Bike sharing systems HTML table from a Wiki page and converted it into a data frame
        - raw_bike_sharing_systems.csv
    2. Data Collection with OpenWeather API Notebook- Issued an HTTP request to the live weather API, converted the resulting JSON file into a dataframe and saved it in a CSV file.
        - cities_weather_forecast.csv
    3. Downloaded datasets as CSV files from cloud storage- Downloaded following aggregated datasets from cloud storage
        - raw_worldcities.csv
        - raw_seoul_bike_sharing.csv

Data collection

- Data Collection Process Flowchart:



Data wrangling

- Data wrangling was performed to remove the noise from data and convert the undesired data format to a format that was used for the analysis.
  1. Performed Data wrangling with stringr and regular expressions
     - Standardized column names for all collected datasets
     - Removed undesired reference links using regular expressions
     - Extracted numeric values using regular expressions
  2. Performed Data wrangling with dplyr
     - Detected and handled missing values
     - Created indicator (dummy) variables for categorical variables
     - Normalized the data data

EDA with SQL

1. Established SQLite connection
2. Determined number of records in the seoul_bike_sharing dataset-8465
3. Determined number of hours that has non-zero rented bike count- 24
4. Queried the weather forecast for Seoul over the next 3 hours
5. Identified the seasons included in the Seoul bike sharing dataset
6. Identified the first and last dates in the Seoul bike sharing dataset
7. Determined the date and hour that had the most bike rentals-3556
8. Determined Hourly popularity and temperature by season
9. Determined Rental Seasonality
10. Determined Weather Seasonality
11. Determined the total number of bikes available in Seoul and city information like City, country, latitude, longitude and population
12. Identified all cities with total bike counts between 15000 to 20000

EDA with data visualization

- Hourly trend of rented bike count by season using a line plot.
- Relationship between temperature and rented bike count using a scatter plot.
- Scatter plot of rented bike count versus date, adjusted for opacity to show density.
- Scatter plot of rented bike count over time with hourly variation color-coded.
- Histogram overlaid with a kernel density curve to display the distribution of rented bike count.
- Bar chart showing the daily total rainfall and snowfall.

Predictive analysis

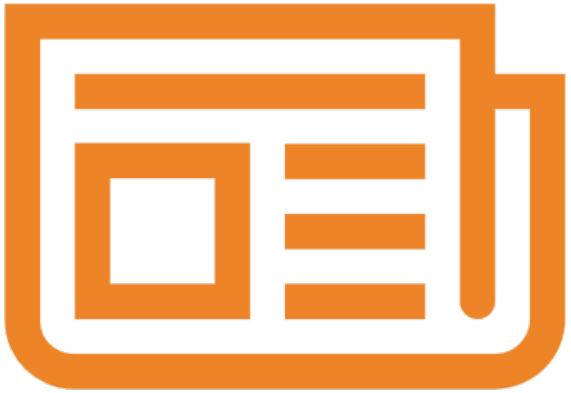
- Finding the best performing model
  - Built a predictive model for bike-sharing demand using linear regression
  - Evaluated the model using R-square and RMSE values
    - Identified important variables by looking at the coefficient values and found 4 important variables- Rainfall, Humidity, Dewpoint temperature and Winter (dummy variable)
  - Improved the model:
    - Added higher order polynomial terms on the important variables
    - Added three interactions terms-
      - RAINFALL * HUMIDITY, HUMIDITY * TEMPERATURE, DEW_POINT_TEMPERATURE * TEMPERATURE
    - Added regularizations terms-
      - Lasso
      - Ridge
      - Elastic net
  - Found the best performing model by comparing RMSE and R-square values of 5 different models

Build a R Shiny dashboard

The dashboard presents visualizations that help users explore temperature trends, bike-sharing demand predictions, and the relationship between humidity and bike demand. Interactive elements allow users to choose cities and obtain detailed information by interacting with the plotted data points.

- **Plots:**
  1. **Leaflet Map:**
     - Displays cities with markers.
     - Allows users to select specific cities.
  2. **Temperature Trend Plot:**
     - Shows the forecasted temperature trend for the selected city over time.
     - Provides insights into temperature variations in the next 5 days.
  3. **Bike-Sharing Prediction Trend Plot:**
     - Illustrates the trend in bike-sharing demand predictions for the selected city.
     - Offers an understanding of expected bike demand over time.
  4. **Humidity-Bike Prediction Scatter Plot:**
     - Explores the correlation between humidity and bike-sharing demand predictions.
     - Demonstrates how bike demand correlates with humidity levels.
- **Interactions:**
  1. **City Selection Dropdown:**
     - Allows users to select a city from a dropdown list.
     - Triggers updates in plotted data and visualizations based on the selected city.
  2. **Click Events:**
     - Enables clicking on plotted points to retrieve specific details (e.g., datetime, bike prediction).
     - Shows detailed information about clicked points on the plots.

Results



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

EDA with SQL

Busiest bike rental times

- Find dates and hours which had the most bike rentals:
  - Date and hour with the most bike rentals: Date: 19/06/2018 Hour: 18 Max Rentals: 3556
- Present your query result with a short explanation here
  - The SQL query retrieves the date and hour combination that experienced the highest number of bike rentals from a Seoul bike-sharing dataset. It utilizes aggregation by date and hour, sorting the results by the maximum rentals in descending order and limits the output to one entry. The printed result displays the date, hour, and the corresponding maximum number of bike rentals recorded in the dataset.

Task 6 - Subquery - 'all-time high'

determine which date and hour had the most bike rentals.

Solution 6

```
]: # provide your solution here


# Execute SQL query to find the date and hour with the most bike rentals
query_max_rentals <- "SELECT DATE, HOUR, MAX(RENTED_BIKE_COUNT) AS MAX_RENTALS
                      FROM SEOUL_BIKE_SHARING
                      GROUP BY DATE, HOUR
                      ORDER BY MAX_RENTALS DESC
                      LIMIT 1"
result_max_rentals <- dbGetQuery(con, query_max_rentals)

# Print the result
cat("Date and hour with the most bike rentals:",
    "Date:", result_max_rentals$DATE,
    "Hour:", result_max_rentals$HOUR,
    "Max Rentals:", result_max_rentals$MAX_RENTALS, "\n")
```

```
Date and hour with the most bike rentals: Date: 19/06/2018 Hour: 18 Max Rentals: 3556
```

Hourly popularity and temperature by seasons

- Find hourly popularity and temperature by season

	SEASONS	HOUR	AVG_TEMPERATURE	AVG_BIKE_COUNT
1	Summer	18	29.38791	2135.141
2	Autumn	18	16.03185	1983.333
3	Summer	19	28.27378	1889.250
4	Summer	20	27.06630	1801.924
5	Summer	21	26.27826	1754.065
6	Spring	18	15.97222	1689.311
7	Summer	22	25.69891	1567.870
8	Autumn	17	17.27778	1562.877
9	Summer	17	30.07691	1526.293
10	Autumn	19	15.06346	1515.568

- Present your query result with a short explanation here

  - This query result calculates the average hourly temperature and bike rentals per hour categorized by seasons (e.g., Summer, Autumn, Spring). The result table shows the top 10 records with the highest average bike rentals (`AVG_BIKE_COUNT`) along with their respective average temperatures (`AVG_TEMPERATURE`) for different seasons and hours. For example, it displays that during Summer at 18:00, the average temperature was around 29.39°C with an average bike count of approximately 2135.14 bike rentals.

Solution 7

```
)]: # provide your solution here

# Execute SQL query to calculate average hourly temperature and average bike rentals per hour by season
query_avg_hourly <- "SELECT SEASONS,
                            HOUR,
                            AVG(TEMPERATURE) AS AVG_TEMPERATURE,
                            AVG(RENTED_BIKE_COUNT) AS AVG_BIKE_COUNT
                     FROM SEOUL_BIKE_SHARING
                     GROUP BY SEASONS, HOUR
                     ORDER BY AVG_BIKE_COUNT DESC
                     LIMIT 10"
result_avg_hourly <- dbGetQuery(con, query_avg_hourly)

# Print the result
print(result_avg_hourly)
```

Rental Seasonality

- Rental Seasonality

  The query result illustrates the average, minimum, and maximum bike rental counts along with the standard deviation across different seasons and hours:

  Seasonal Trends:

  - Autumn: Shows a rise in bike rentals from early morning (minimum: 24, maximum: 3298) peaking around 6-8 AM and gradually declining afterward.
  - Spring: Exhibits a similar trend with the highest rentals around 5-6 PM (minimum: 22, maximum: 3251), maintaining a relatively steady pattern throughout the day.
  - Summer: Demonstrates a consistent increase in bike rentals throughout the day (minimum: 14, maximum: 3556), peaking in the evening around 5-6 PM.
  - Winter: Displays significantly lower rentals compared to other seasons, with a slight increase in the morning hours (minimum: 3, maximum: 937).

  Hourly Analysis:

  - Each season showcases fluctuations in bike rentals throughout the day, with varying peaks and lows during different hours.
  - Standard Deviation (STD_DEV_BIKE_COUNT): Indicates the variability in rental counts within each hour and season. Higher standard deviations signify more fluctuation in bike rentals during that specific hour across the given season.

Solution 8

```
[]: # provide your solution here

# Execute SQL query to calculate average, minimum, maximum, and standard deviation of hourly bike count by season
query_seasonality <- "SELECT SEASONS,
                             HOUR,
                             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, HOUR"
result_seasonality <- dbGetQuery(con, query_seasonality)

# Print the result
print(result_seasonality)
```

	SEASONS	HOUR	AVG_BIKE_COUNT	MIN_BIKE_COUNT	MAX_BIKE_COUNT	STD_DEV_BIKE_COUNT
1	Autumn	0	709.43750	119	1336	219.14298
2	Autumn	1	552.50000	144	1001	191.54216
3	Autumn	2	377.47500	55	785	144.90134
4	Autumn	3	256.55000	28	514	102.53108
5	Autumn	4	169.02500	24	338	58.63957
6	Autumn	5	163.41250	24	264	53.88174
7	Autumn	6	359.48750	23	691	180.27049
8	Autumn	7	788.87654	5	1556	457.96861
9	Autumn	8	1345.03704	6	2391	758.35956
10	Autumn	9	848.43210	5	1322	334.52653
11	Autumn	10	715.27160	2	1113	252.56839
12	Autumn	11	802.95062	20	1284	302.57238
13	Autumn	12	934.64198	17	1634	347.69066
14	Autumn	13	1002.66667	18	1849	369.30997
15	Autumn	14	1058.82716	17	1995	403.85277
16	Autumn	15	1156.70370	8	2200	455.21810
17	Autumn	16	1293.20988	14	2270	498.10638
18	Autumn	17	1562.87654	23	2432	554.31649

Weather Seasonality

- Weather Seasonality

  The query result showcases the average weather attributes and bike rental counts across different seasons:

  Temperature & Weather Conditions:

  - Summer records the highest average temperature (26.59°C) and humidity (64.98%), indicating warmer and more humid weather conditions.
  - Winter stands out with significantly lower temperatures (-2.54°C) and higher visibility, suggesting colder weather with clearer skies.

  Precipitation:

  - Summer displays higher rainfall (0.25 mm) compared to other seasons, while Autumn shows a notable average snowfall (0.06 mm), likely indicating weather transitions.

  Bike Rentals:

  - Summer exhibits the highest average bike rental count (1034.07), correlating potentially with the warmer weather conducive to outdoor activities.
  - Winter records the lowest bike rental count (225.54), possibly due to colder conditions limiting outdoor biking.

Solution 9

[42]:
```
# provide your solution here

# Execute SQL query to calculate average weather parameters and bike count per season
query_weather_seasonality <- "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_weather_seasonality <- dbGetQuery(con, query_weather_seasonality)

# Print the result
print(result_weather_seasonality)
```

```
  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
```

Bike-sharing info in Seoul

- Find the total Bike count and city info for Seoul
  - The query result provides specific information about Seoul, South Korea, and its bike-sharing system:

  Seoul's Demographics:
    - City & Country: Represents Seoul in South Korea.
    - Population: Shows a large population of 21,794,000 people residing in Seoul.
    - Geographical Coordinates (Latitude & Longitude): Indicates the precise location of Seoul on the map.
  - Bike-Sharing System:
    - Total Bikes: Reveals that Seoul's bike-sharing system comprises around 20,000 bicycles available for public use.

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

# Execute SQL query to join WORLD_CITIES and BIKE_SHARING_SYSTEMS and retrieve city information for Seoul
query_seoul_info <- "SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, SUM(BSS.BICYCLES) AS TOTAL_BIKES
                     FROM WORLD_CITIES WC
                     JOIN BIKE_SHARING_SYSTEMS BSS ON WC.CITY = BSS.CITY
                     WHERE WC.CITY = 'Seoul'
                     GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION"
result_seoul_info <- dbGetQuery(con, query_seoul_info)

# Print the result
print(result_seoul_info)
```

```
   CITY      COUNTRY     LAT LNG POPULATION TOTAL_BIKES
1 Seoul Korea, South 37.5833 127   21794000       20000
```

Cities similar to Seoul

The SQL query aims to identify cities with a comparable bike scale to Seoul's bike-sharing system (ranging from 15,000 to 20,000 bicycles). Here's an overview of the findings:

Cities with Similar Bike Scale:

- Beijing, Shanghai, Weifang, Zhuzhou (China): These cities in China have bike-sharing systems with bicycles ranging from 15,000 to 20,000, comparable to Seoul.
- Ningbo (China): Features a system with around 15,000 bicycles, close to Seoul's bike scale.

Population and Geographical Spread:

- Shows the population sizes and geographic coordinates (latitude and longitude) of these cities, indicating their potential size and locations relative to Seoul.

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

# Execute SQL query to find cities with comparable bike scale to Seoul's bike sharing system
query_comparable_cities <- "SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, BSS.BICYCLES AS TOTAL_BIKES
                            FROM WORLD_CITIES WC
                            JOIN BIKE_SHARING_SYSTEMS BSS ON WC.CITY = BSS.CITY
                            WHERE BSS.BICYCLES BETWEEN 15000 AND 20000
                            GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, BSS.BICYCLES"
result_comparable_cities <- dbGetQuery(con, query_comparable_cities)

# Print the result
print(result_comparable_cities)
```

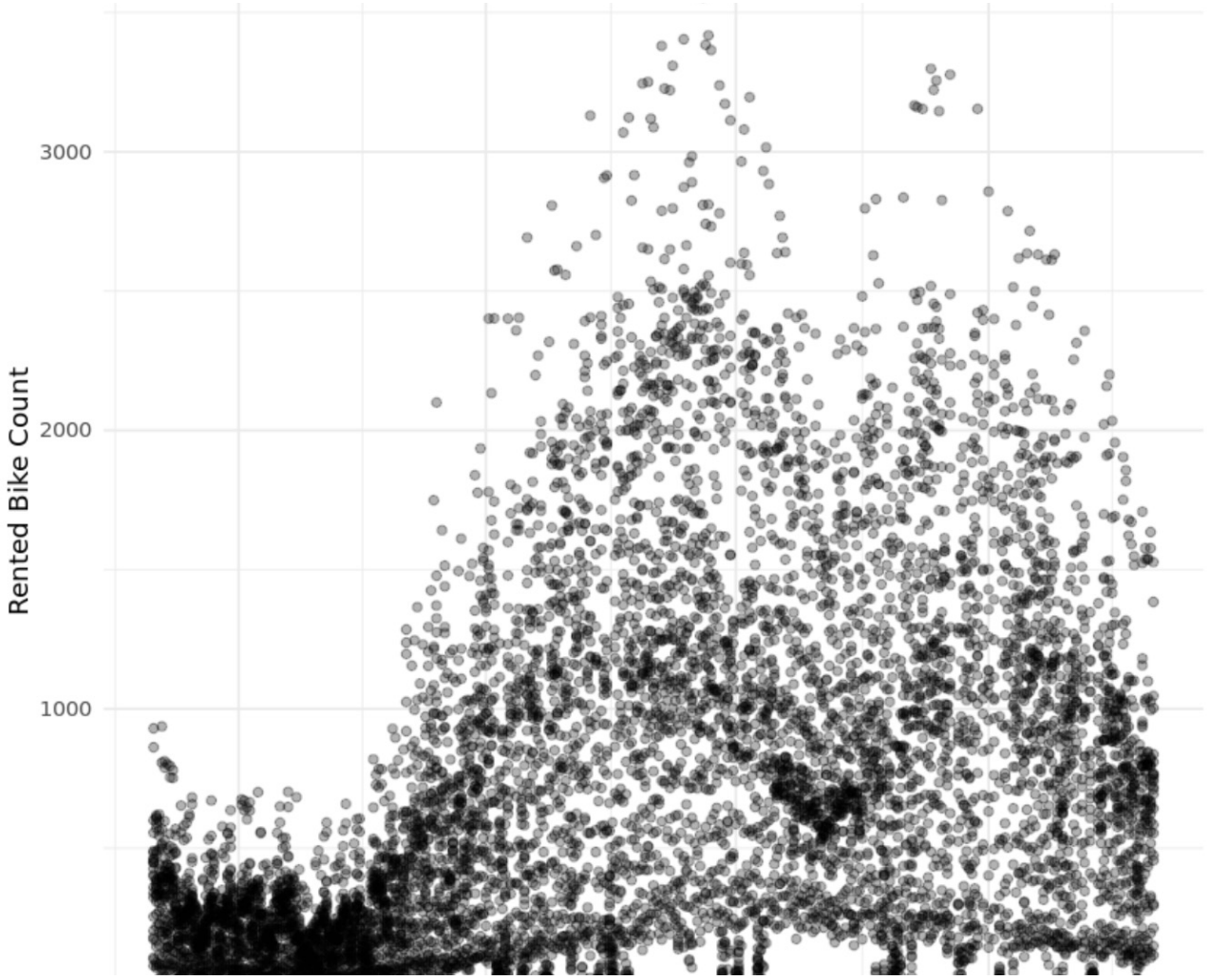
```
      CITY      COUNTRY     LAT      LNG POPULATION TOTAL_BIKES
1  Beijing        China 39.9050 116.3914   19433000       16000
2   Ningbo        China 29.8750 121.5492    7639000       15000
3    Seoul Korea, South 37.5833 127.0000   21794000       20000
4 Shanghai        China 31.1667 121.4667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

EDA with Visualization

Bike rental vs. Date

Show a scatter plot
of RENTED_BIKE_COUNT vs. DATE

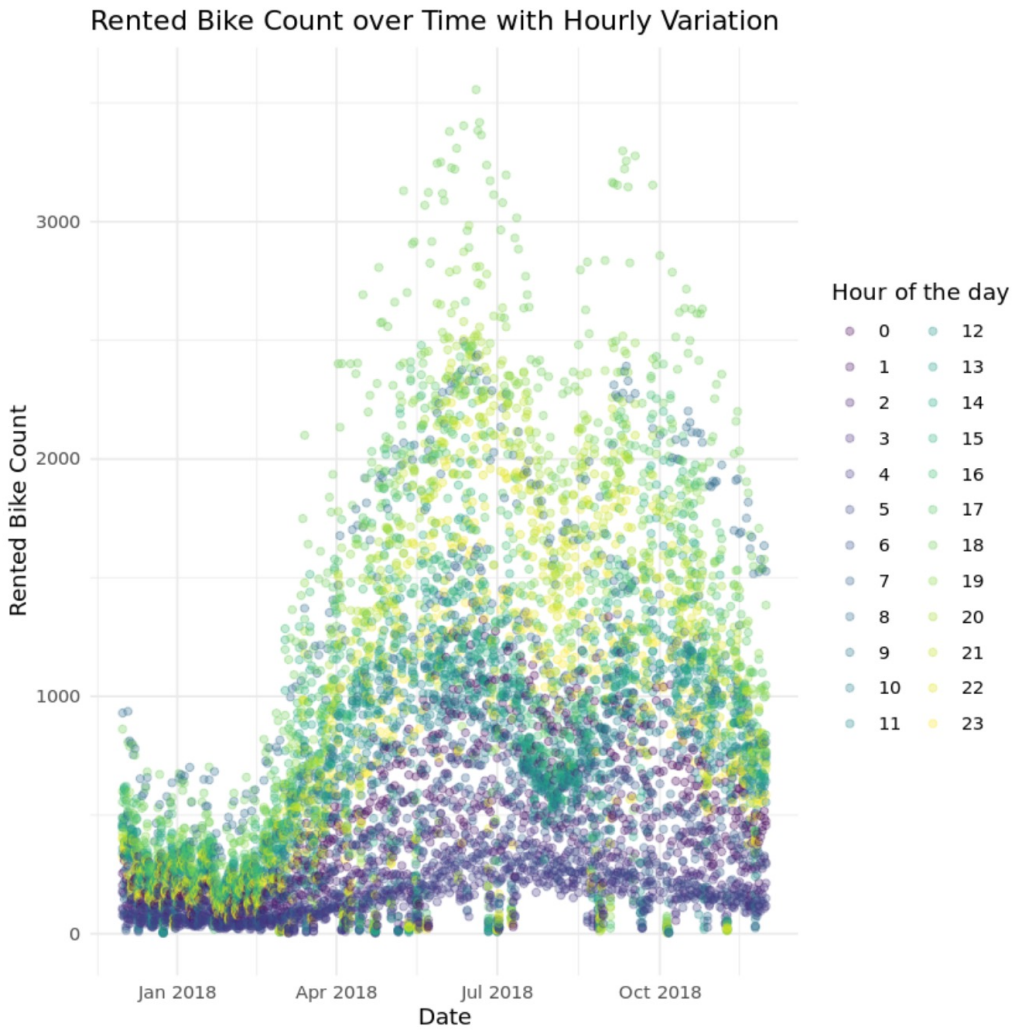
- This code creates a scatter plot where DATE is on the x-axis and RENTED_BIKE_COUNT is on the y-axis. The alpha parameter in geom_point adjusts the opacity of the points, allowing you to see overlapping points more clearly without them obscuring each other too much.



Bike rental vs. Datetime

Show the same plot of
the RENTED_BIKE_COUNT time series, but
now add HOURS as the colour

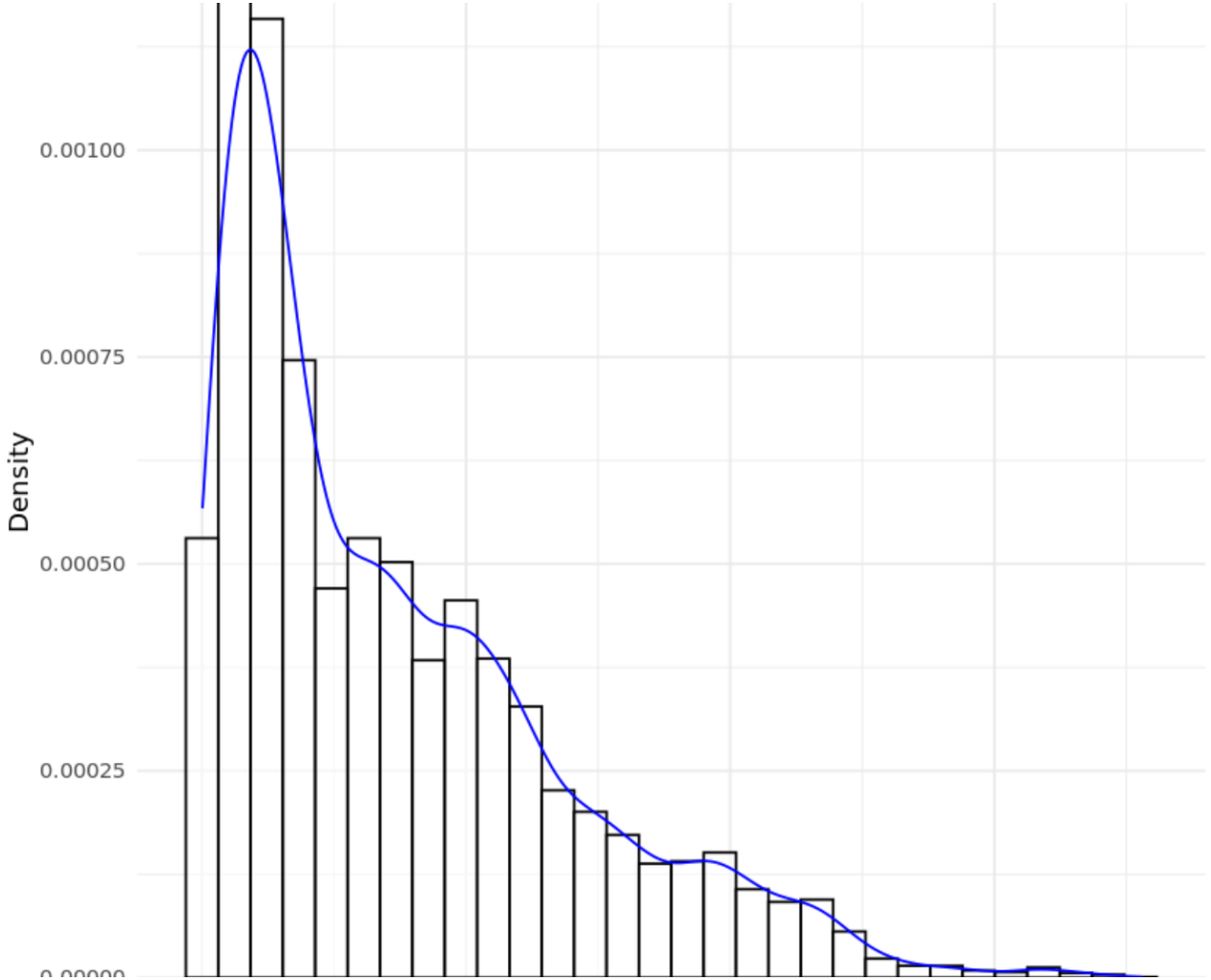
- This modified code generates a scatter plot where DATE is on the x-axis, RENTED_BIKE_COUNT is on the y-axis, and the points are colored based on the HOUR.



Bike rental histogram

Show a histogram overlaid with a kernel density curve

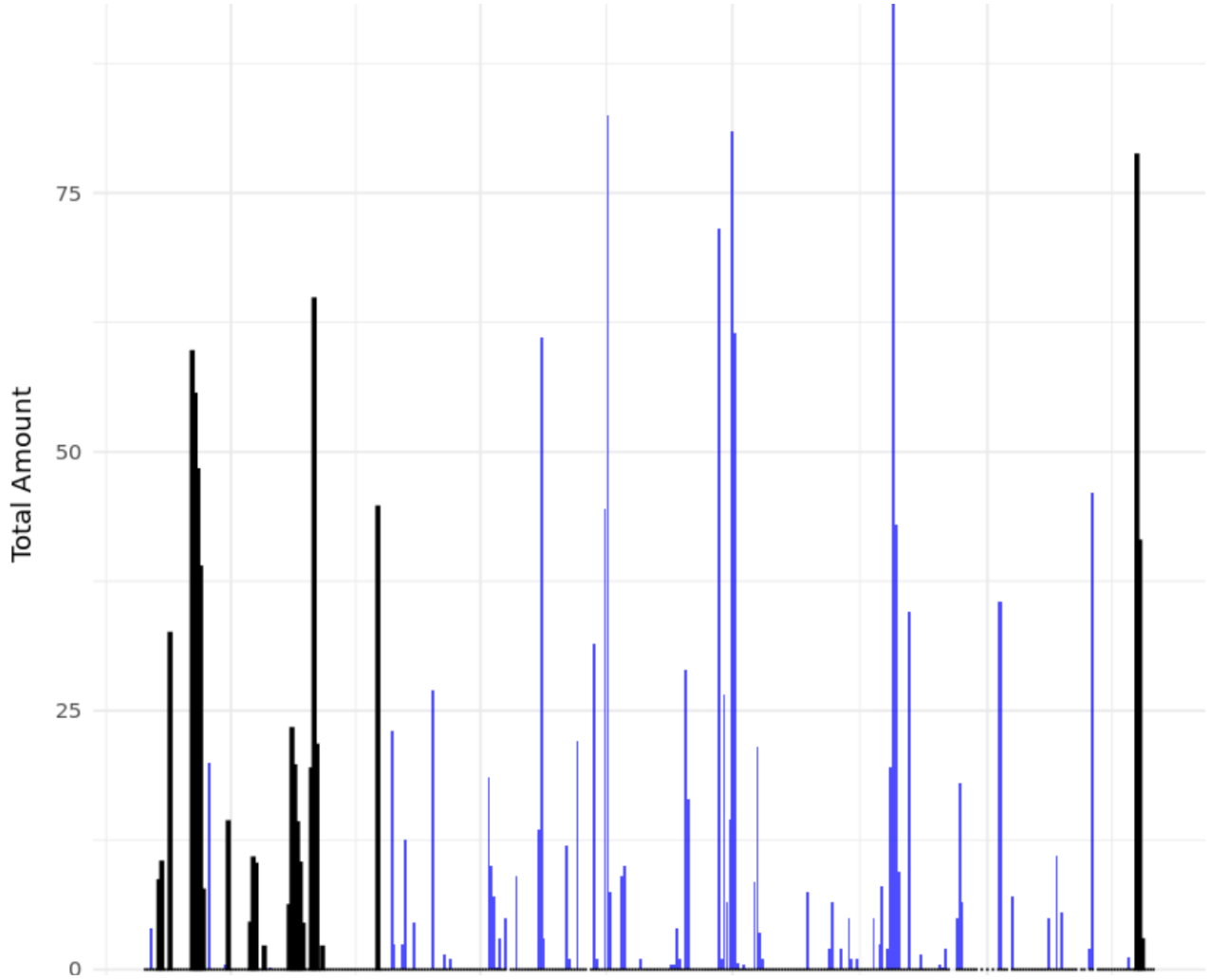
- In this code, geom_histogram() is used to create the histogram, and by setting y = ..density.. in the aesthetics, it normalizes the y-axis to represent density. The fill parameter is set to "white" and the color parameter is set to "black" to differentiate the histogram bars from the kernel density plot. The alpha values for both the histogram and density plot are adjusted to ensure that the density plot is clearly visible without obscuring the histogram.



Daily total rainfall and snowfall

Show a barchart calculating the daily total rainfall and snowfall

- This code first groups the data by DATE and calculates the total rainfall and snowfall for each day. Then, it creates a bar chart using geom_bar for both total rainfall and snowfall. The fill, color, and alpha parameters are adjusted to differentiate between the bars representing rainfall and snowfall.

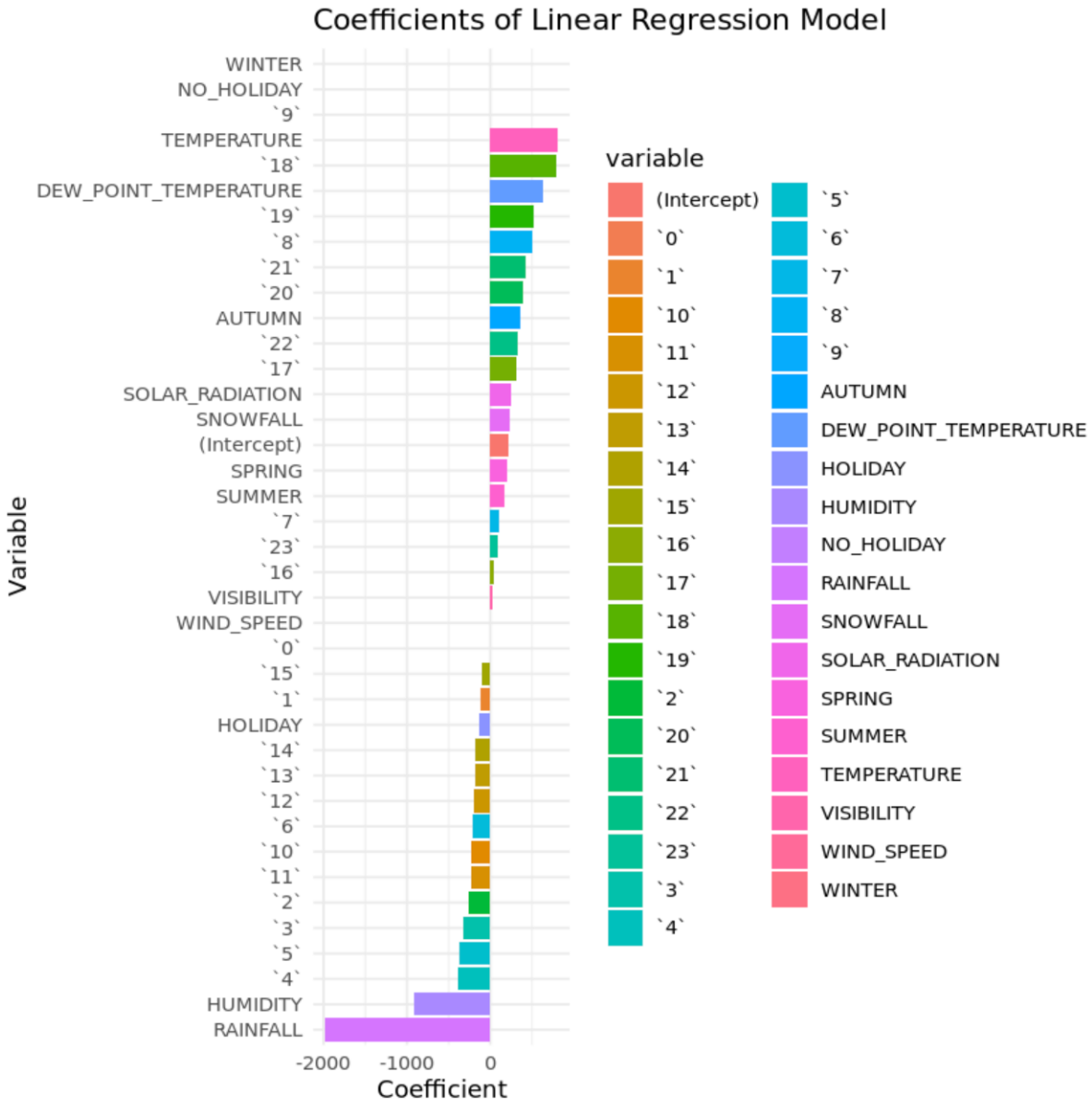


Predictive analysis

Ranked coefficients

Show a screenshot of the ranked coefficients bar chart for the baseline model

- We can see that Rainfall has the highest absolute coefficient value but it is negative indicating that higher rainfall leads to lower bike rents.
- Temperature coefficient is high and positive indicating that higher temperature denotes warm climate encouraging to use the bike

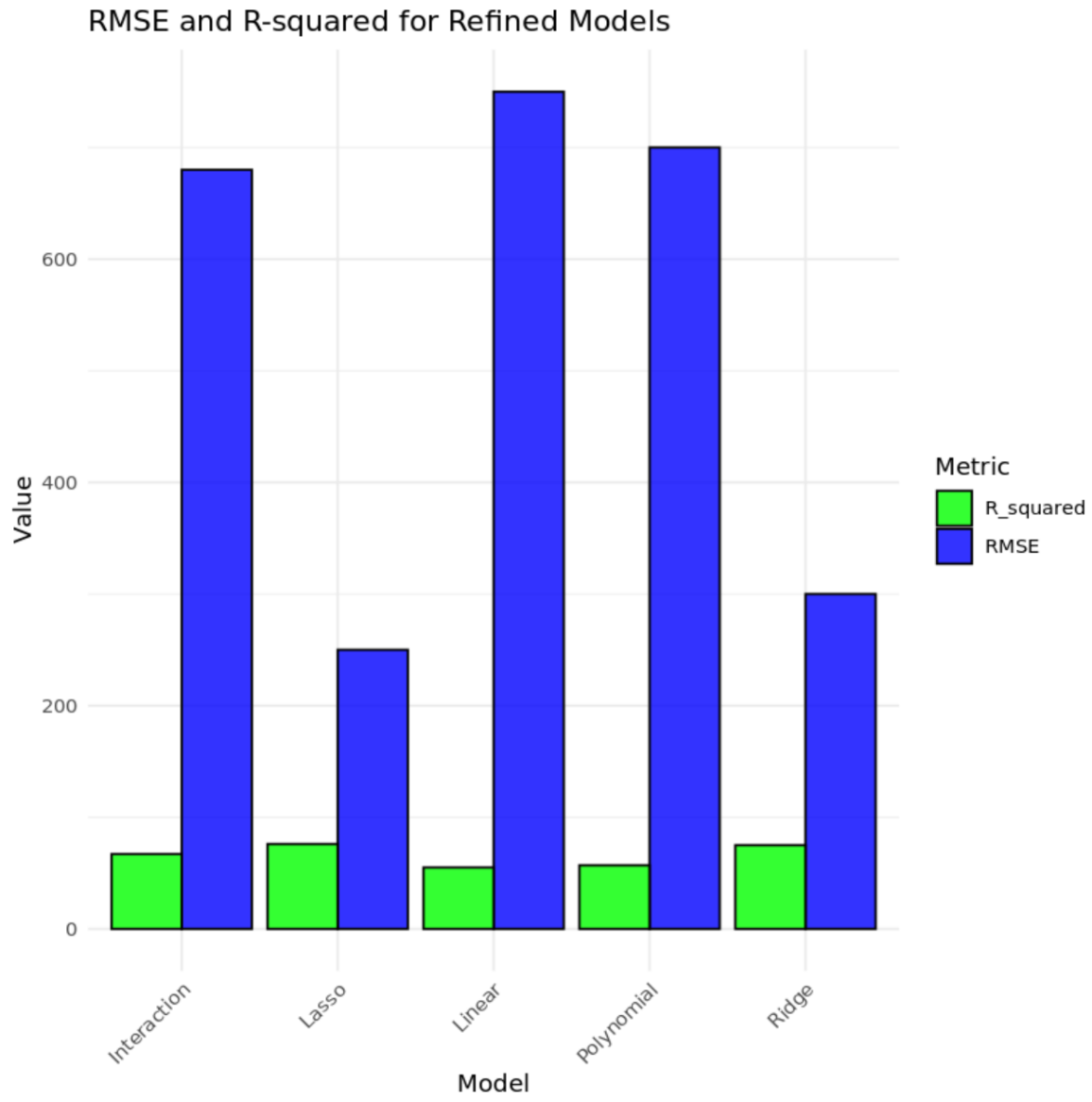


Model evaluation

Built at least 5 different models using polynomial terms, interaction terms, and regularizations

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

Note- R-squared value is multiplied by 100 for a grouped bar chart visualization



Find the best performing model

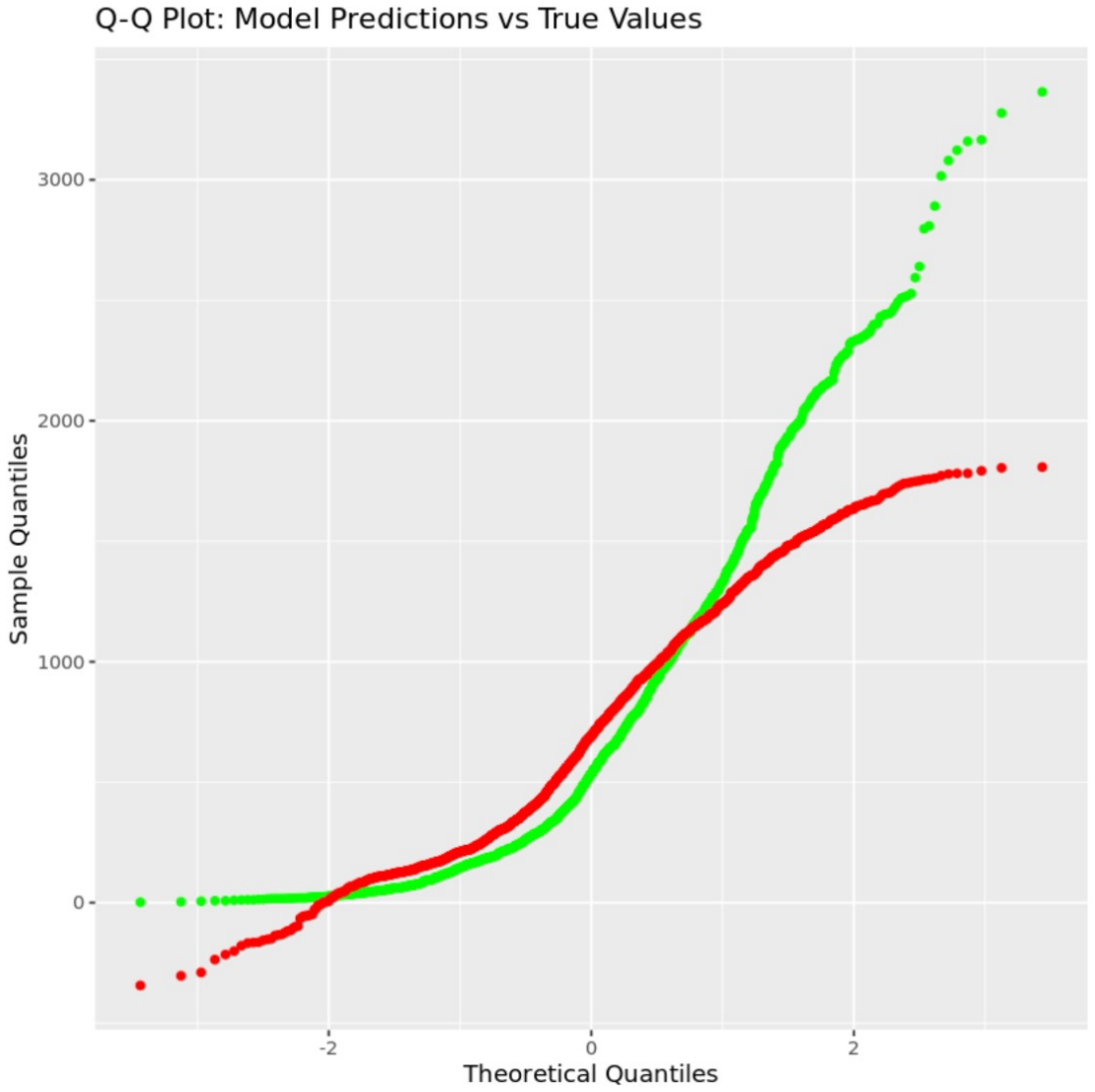
- Lasso is the best performing model with:
  - RMSE = 250.377
  - R-squared = 0.76
- Lasso model formula:

```
# Lasso regression using glmnet package
lasso_model <- glmnet(as.matrix(seoul_bike_sharing[, -which(names(seoul_bike_sharing) %in% c("RENTED_BIKE_COUNT"))]),
                      seoul_bike_sharing$RENTED_BIKE_COUNT,
                      alpha = 1, lambda = 0.1)  # Adjust lambda for regularization strength
summary(lasso_model)
```

Q-Q plot of the best-Lasso model

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

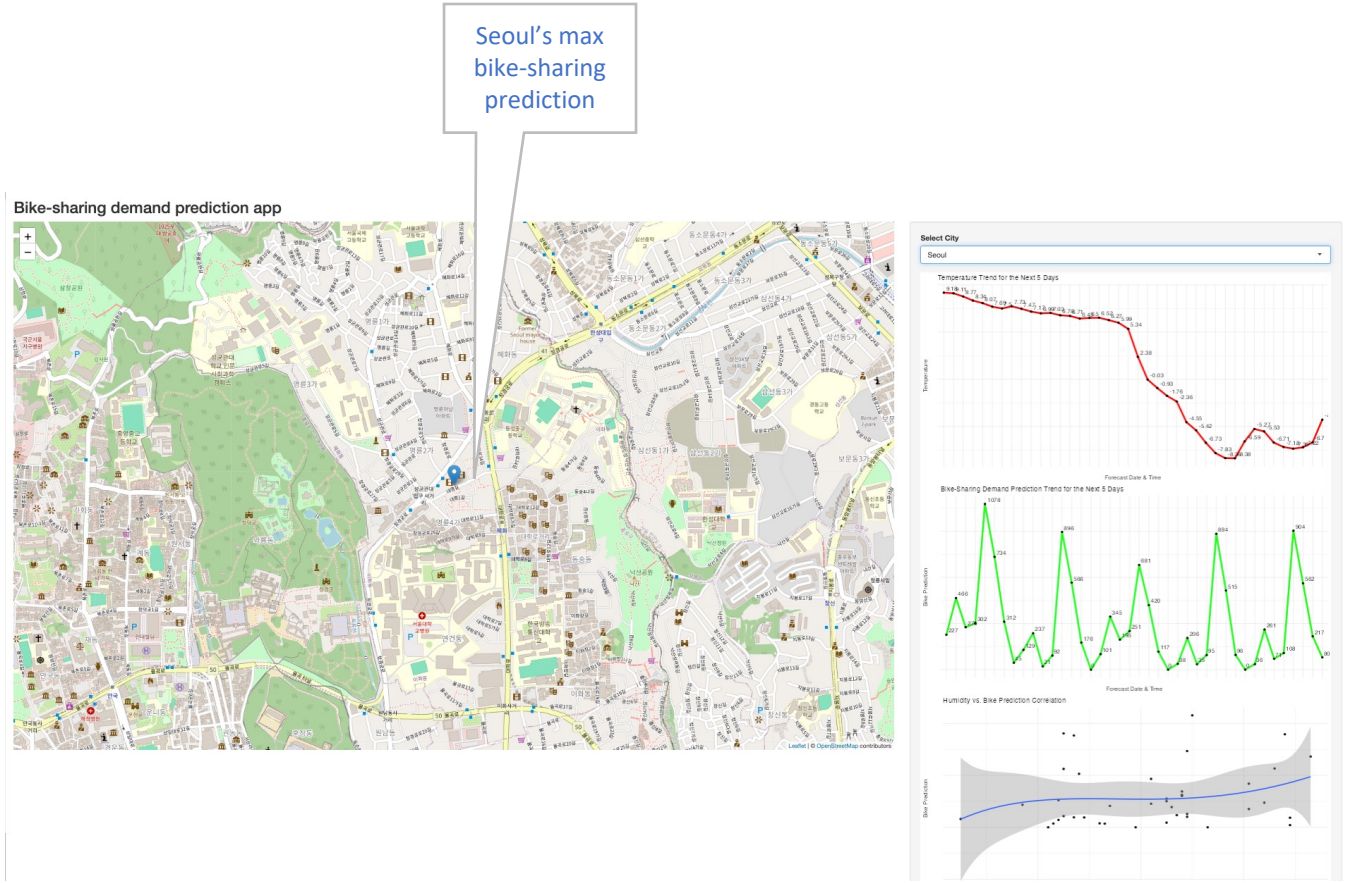
```r
# Calculate residuals
lasso_residuals <- test_data$RENTED_BIKE_COUNT - predictions_lasso

# Create a Q-Q plot
ggplot(data = test_data, aes(sample = RENTED_BIKE_COUNT)) +
  stat_qq(color = 'green') +
  stat_qq(aes(sample = predictions_lasso), color = 'red') +
  ggtitle("Q-Q Plot: Model Predictions vs True Values") +
  xlab("Theoretical Quantiles") +
  ylab("Sample Quantiles")
```



Dashboard

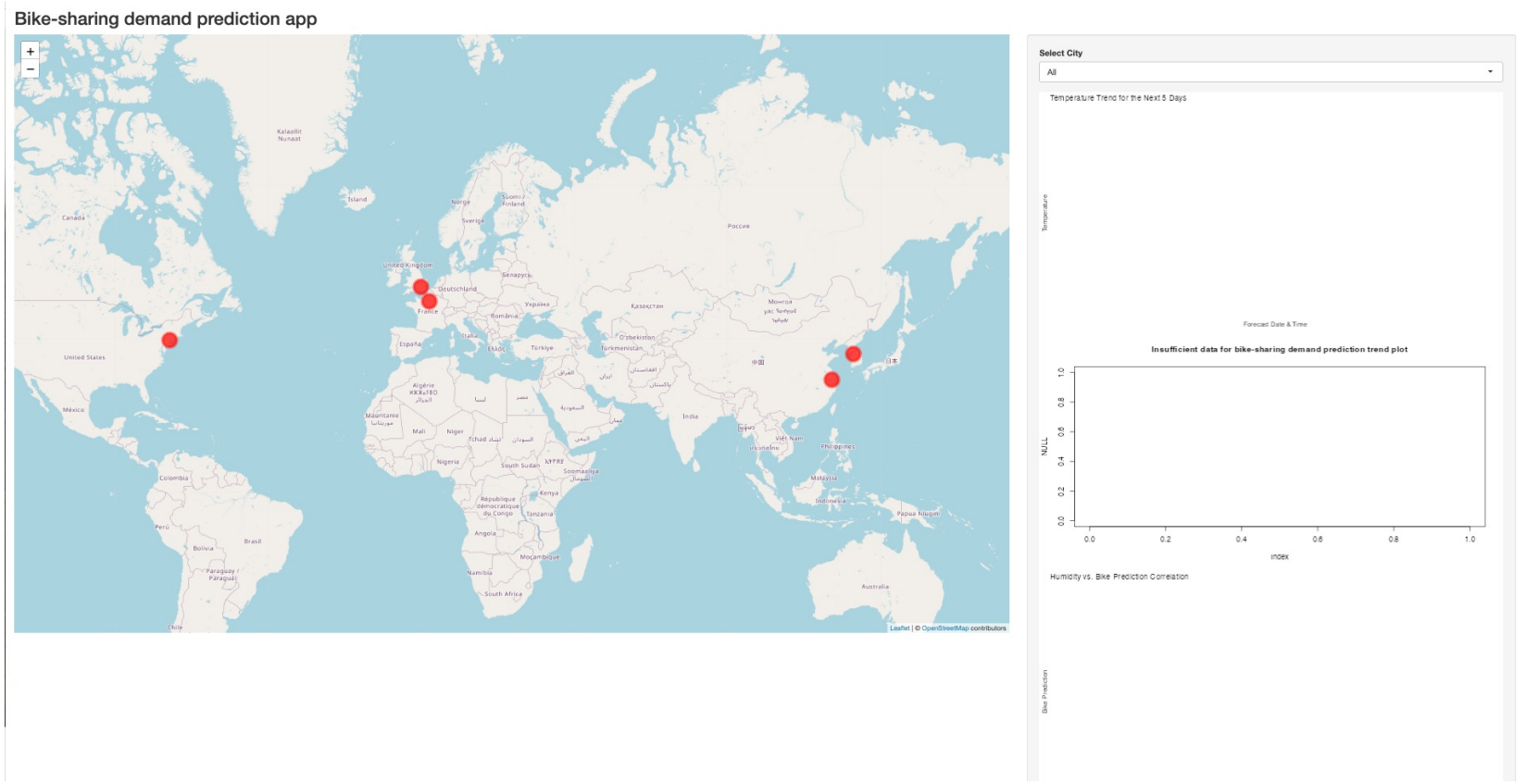
Cities’ max bike-sharing prediction

- In this figure the selected city is Seoul and the app shows max bike-sharing prediction as 1078



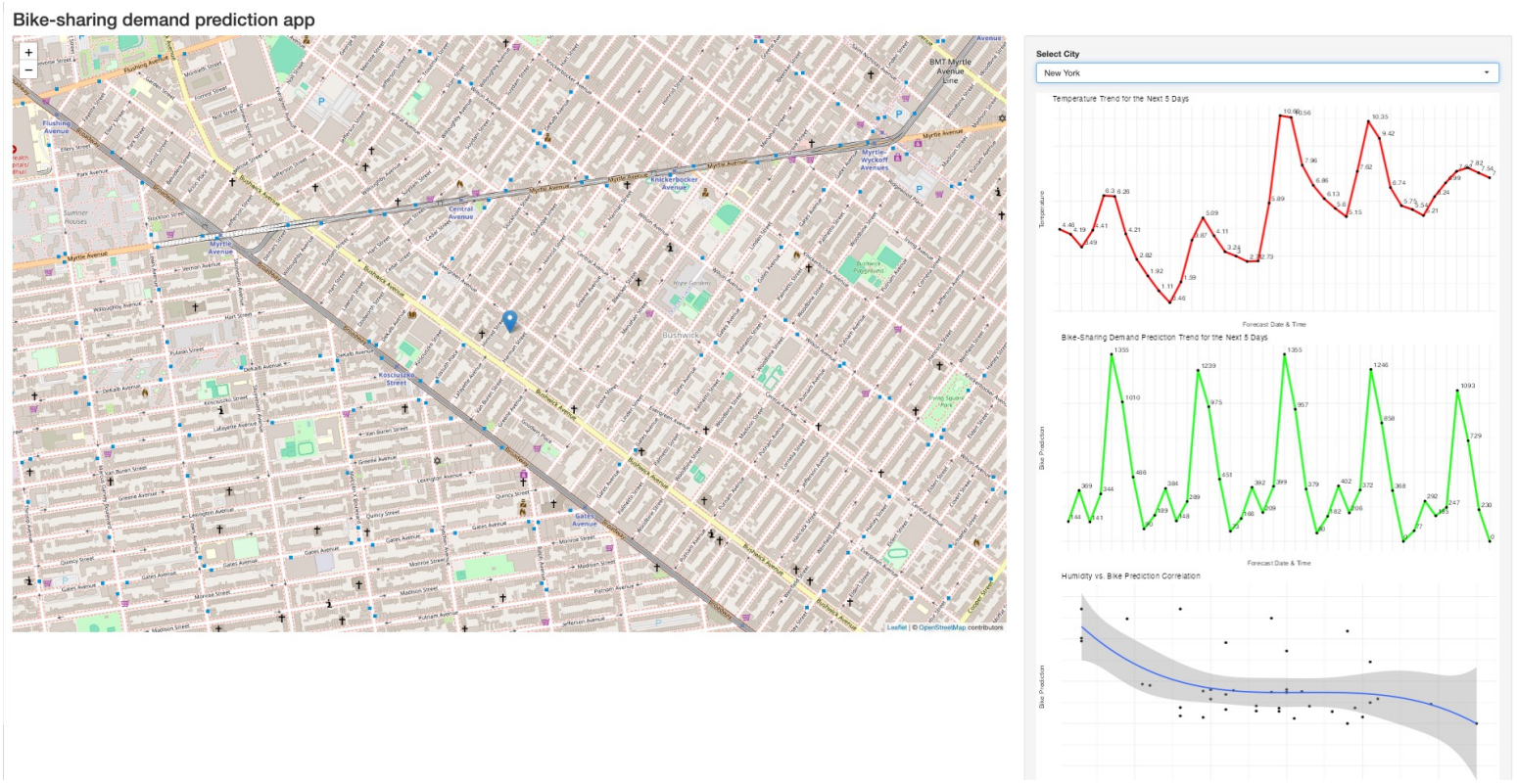
Option to select a city in an App

Currently the map shows Data for all the cities with an option to select a city



Selecting a specific city in an App

New York City is selected currently in the bike-sharing demand prediction app



CONCLUSION

- The Bike-sharing demand prediction dashboard solves various problems:
- **Demand Anticipation:** Predicts future bike-sharing demand in different cities, aiding urban planners and bike-sharing companies in resource allocation and operational planning.
- **Resource Optimization:** Helps optimize bike fleet distribution based on predicted demand, reducing unnecessary oversupply or scarcity of bikes in specific areas.
- **User Accessibility:** Provides a user-friendly interface for stakeholders to visualize and understand bike-sharing demand trends, promoting informed decision-making.
- **Traffic Management:** Assists in managing traffic congestion by encouraging bike usage, potentially reducing vehicular traffic and associated environmental impacts.
- **Operational Efficiency:** Enhances the efficiency of bike-sharing systems by forecasting demand trends, facilitating proactive measures to meet user needs effectively.

APPENDIX

- R code snippets, SQL queries, charts, Notebook outputs, or data sets that you may have created during this project

APPENDIX

- Data collection-
  Web scraping

TODO: Get the root HTML node

[6]:
```r
url <- "https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems"
# Get the root HTML node by calling the `read_html()` method with URL


html <- read_html(url)

# Get all table nodes under the root HTML node
table_nodes <- html_nodes(html, "table")

# Print each table to identify the bike sharing table
for (i in seq_along(table_nodes)) {
  cat("Table", i, ":\n")
  print(table_nodes[[i]])
  cat("\n\n")
}
```

```
Table 1 :
{html_node}
<table class="wikitable sortable" style="text-align:left">
[1] <tbody>\n<tr>\n<th>Country</th>\n<th>City</th>\n<th>Name</th>\n<th>System ...


Table 2 :
{html_node}
<table class="nowraplinks mw-collapsible autocollapse navbox-inner" style="border-spacing:0;background:transparent;color:inherit">
[1] <tbody>\n<tr><th scope="col" class="navbox-title" colspan="2">\n<link rel ...


Table 3 :
{html_node}
<table class="nowraplinks navbox-subgroup" style="border-spacing:0">
[1] <tbody>\n<tr>\n<th scope="row" class="navbox-group" style="width:1%">East ...


Table 4 :
{html_node}
<table class="nowraplinks navbox-subgroup" style="border-spacing:0">
[1] <tbody>\n<tr>\n<th scope="row" class="navbox-group" style="width:1%">Cana ...
```

Note that this HTML page at least contains three child `<table>` nodes under the root HTML node. So, you will need to use `html_nodes(root_node, "table")` function to get all its child `<table>` nodes:

```
<html>
  <table>(table1)</table>
  <table>(table2)</table>
  <table>(table3)</table>
  ...
</html>
```

table_nodes <- html_nodes(root_node, "table")

APPENDIX-

- Data collection- OpenWeather API

Get the current weather data for a city using OpenWeather API

[13]:
```
# need to be replaced by your real API key
your_api_key <- "09e99c267041ec495f998b8b71729144"
# Input `q` is the city name
# Input `appid` is your API KEY,
# Input `units` are preferred units such as Metric or Imperial
current_query <- list(q = "Seoul", appid = your_api_key, units="metric")

current_query
```

$q	'Seoul'
$appid	'09e99c267041ec495f998b8b71729144'
$units	'metric'

Now we can make a HTTP request to the current weather API

[14]:
```
response <- GET(current_weather_url, query=current_query)
```

If we check the response type, we can see it is in JSON format

[16]:
```
http_type(response)
```

'application/json'

JSON is an open standard file and data interchange format that uses human-readable text to store and transmit data objects. To read the JSON HTTP response, you can use the `content()` function to parse it as a named list in R.

[17]:
```
json_result <- content(response, as="parsed")
```

If you use the `class()` function, you can see it is a R `List` object

Now let's print the JSON result.

[19]:
```
json_result
```

$coord	**$lon**	126.9778
	$lat	37.5683
$weather	1. **$id**	803
	$main	'Clouds'
	$description	'broken clouds'
	$icon	'04d'
$base	'stations'	
$main	**$temp**	9.51

APPENDIX-

- Data collection- OpenWeather API

TASK: Get 5-day weather forecasts for a list of cities using the OpenWeather API

Now you should be familiar with the usage of OpenWeather API. Next, you need to complete a task to get 5-day weather forecasts for a list of cities

[ ]:

24]:
```r
# Create some empty vectors to hold data temporarily

# City name column
city <- c()
# Weather column, rainy or cloudy, etc
weather <- c()
# Sky visibility column
visibility <- c()
# Current temperature column
temp <- c()
# Max temperature column
temp_min <- c()
# Min temperature column
temp_max <- c()
# Pressure column
pressure <- c()
# Humidity column
humidity <- c()
# Wind speed column
wind_speed <- c()
# Wind direction column
wind_deg <- c()
# Forecast timestamp
forecast_datetime <- c()
# Season column
# Note that for season, you can hard code a season value from levels Spring, Summer, Autumn, and Winter based on your current month.
season <- c()
```

APPENDIX-

- Data collection- OpenWeather API

```
[ ]: library(httr)
library(jsonlite)

# Get forecast data for a given city list
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/forecast'

    # Create query parameters
    forecast_query <- list(q = city_name, appid = your_api_key, units = "metric")

    # Make HTTP GET call for the given city
    response <- GET(url = forecast_url, query = forecast_query)

    # Check if the response is successful
    if (http_type(response) == "application/json") {
      # Parse the JSON response
      forecast_data <- content(response, "parsed")

      # Extract relevant information from the forecast_data
      city <- rep(city_name, length(forecast_data$list))
      date <- sapply(forecast_data$list, function(x) x$dt_txt)
      temperature <- sapply(forecast_data$list, function(x) x$main$temp)
      humidity <- sapply(forecast_data$list, function(x) x$main$humidity)

      # Combine the extracted information into a data frame
      city_forecast <- data.frame(city, date, temperature, humidity)

      # Append the city_forecast to the overall data frame
      df <- rbind(df, city_forecast)
    } else {
      # Handle the case where the API request was not successful
      print(paste("Error fetching data for", city_name))
    }
  }

  # Return the final data frame
  return(df)
}

# Example usage with a list of cities
city_list <- c("London", "Paris", "Berlin")
weather_forecast_data <- get_weather_forecast_by_cities(city_list)
```

APPENDIX-

- Data Wrangling-
  - Using regular expression
  - 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

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`

[38]:
```
# remove reference link
remove_ref <- function(strings) {
    # ref_pattern <- "Define a pattern matching a reference link such as [1]"
    ref_pattern <- "\\[[A-z0-9]+\\]"

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

    # Trim the reslt if you want
    result <- trimws(result)

    return(result)
}

# sub_bike_sharing_df$CITY <- remove_ref(sub_bike_sharing_df$CITY)
# sub_bike_sharing_df$SYSTEM <- remove_ref(sub_bike_sharing_df$SYSTEM)
```

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

[49]:
```
# sub_bike_sharing_df %>% mutate(column1=remove_ref(column1), ... )

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:

[50]:
```
result %>%
    select(CITY, SYSTEM, BICYCLES) %>%
    filter(find_reference_pattern(CITY) | find_reference_pattern(SYSTEM) | find_reference_pattern(BICYCLES))
```

A spec_tbl_df: 0 × 3

APPENDIX-

- Data Wrangling- missing values handling.

TASK: Detect and handle missing values

The `RENTED_BIKE_COUNT` column has about 295 missing values, and `TEMPERATURE` has about 11 missing values. Those missing values could be caused by not being recorded, or from malfunctioning bike-sharing systems or weather sensor networks. In any cases, the identified missing values have to be properly handled.

Let's first handle missing values in `RENTED_BIKE_COUNT` column:

Considering `RENTED_BIKE_COUNT` is the response variable/dependent variable, i.e., we want to predict the `RENTED_BIKE_COUNT` using other predictor/independent variables later, and we normally can not allow missing values for the response variable, so missing values for response variable must be either dropped or imputed properly.

We can see that `RENTED_BIKE_COUNT` only has about 3% missing values (295 / 8760). As such, you can safely drop any rows whose `RENTED_BIKE_COUNT` has missing values.

TODO: Drop rows with missing values in the `RENTED_BIKE_COUNT` column

[67]:
```
cat("Dimensions before dropping rows:", dim(bike_sharing_df))

# Drop rows with `RENTED_BIKE_COUNT` column == NA
bike_sharing_df <- bike_sharing_df %>%
  drop_na(RENTED_BIKE_COUNT)
```

```
Dimensions before dropping rows: 8760 14
```

[68]:
```
# Print the dataset dimension again after those rows are dropped
cat("Dimensions after dropping rows:", dim(bike_sharing_df))
```

```
Dimensions after dropping rows: 8465 14
```

APPENDIX-

- Data Wrangling-

Generating indicator columns

TODO: Convert `SEASONS`, `HOLIDAY`, `FUNCTIONING_DAY`, and `HOUR` columns into indicator columns.

Note that if `FUNCTIONING_DAY` only contains one categorical value after missing values removal, then you don't need to convert it to an indicator column.

[17]:
```
library(dplyr)
#install.packages("fastDummies")
library(fastDummies)
```

```
Thank you for using fastDummies!
To acknowledge our work, please cite the package:
Kaplan, J. & Schlegel, B. (2023). fastDummies: Fast Creation of Dummy (Binary) Columns and Rows from Categorical Variables. Version 1.7.1.
URL: https://github.com/jacobkap/fastDummies, https://jacobkap.github.io/fastDummies/.
```

[18]:
```
# Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.

# Convert categorical columns to indicator columns
bike_sharing_df2 <- bike_sharing_df %>%
  dummy_cols(select_columns = c("SEASONS", "HOLIDAY", "FUNCTIONING_DAY", "HOUR"), remove_first_dummy = TRUE)
```

[19]:
```
cat("Columns before dummy variables")
colnames(bike_sharing_df)
```

```
Columns before dummy variables
'DATE' · 'RENTED_BIKE_COUNT' · 'HOUR' · 'TEMPERATURE' · 'HUMIDITY' · 'WIND_SPEED' · 'VISIBILITY' · 'DEW_POINT_TEMPERATURE' · 'SOLAR_RADIATION' ·
'RAINFALL' · 'SNOWFALL' · 'SEASONS' · 'HOLIDAY' · 'FUNCTIONING_DAY'
```

[20]:
```
cat("Columns after dummy variables")
colnames(bike_sharing_df2)
```

```
Columns after dummy variables
'DATE' · 'RENTED_BIKE_COUNT' · 'HOUR' · 'TEMPERATURE' · 'HUMIDITY' · 'WIND_SPEED' · 'VISIBILITY' · 'DEW_POINT_TEMPERATURE' · 'SOLAR_RADIATION' ·
'RAINFALL' · 'SNOWFALL' · 'SEASONS' · 'HOLIDAY' · 'FUNCTIONING_DAY' · 'SEASONS_Spring' · 'SEASONS_Summer' · 'SEASONS_Winter' · 'HOLIDAY_No Holiday' ·
'FUNCTIONING_DAY_' · 'HOUR_1' · 'HOUR_2' · 'HOUR_3' · 'HOUR_4' · 'HOUR_5' · 'HOUR_6' · 'HOUR_7' · 'HOUR_8' · 'HOUR_9' · 'HOUR_10' · 'HOUR_11' ·
'HOUR_12' · 'HOUR_13' · 'HOUR_14' · 'HOUR_15' · 'HOUR_16' · 'HOUR_17' · 'HOUR_18' · 'HOUR_19' · 'HOUR_20' · 'HOUR_21' · 'HOUR_22' · 'HOUR_23'
```

APPENDIX-

- EDA with SQL-

Add screenshots of all required SQL queries to the Appendix section

Establish your SQlIte 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.

[30]:
```
# install.packages("DBI")
# install.packages("RSQLite")
```

[31]:
```
# provide your solution here
library(tidyverse)
library(DBI)
library(RSQLite)
```

[32]:
```
db_file <- "sql_database.db"
con <- dbConnect(RSQLite::SQLite(), dbname = db_file)
```

Task 1 - Record Count

Determine how many records are in the seoul_bike_sharing dataset.

Solution 1

[34]:
```
# provide your solution here


query <- "SELECT COUNT(*) FROM SEOUL_BIKE_SHARING"
result <- dbGetQuery(con, query)

cat("Number of records in SEOUL_BIKE_SHARING:", result$`COUNT(*)`, "\n")
```

```
Number of records in SEOUL_BIKE_SHARING: 8465
```

Task 2 - Operational Hours

Determine how many hours had non-zero rented bike count.

Solution 2

[35]:
```
# provide your solution here


# Execute SQL query
query <- "SELECT COUNT(DISTINCT HOUR) FROM SEOUL_BIKE_SHARING WHERE RENTED_BIKE_COUNT > 0"
result <- dbGetQuery(con, query)


cat("Number of hours with non-zero rented bike count:", result$`COUNT(DISTINCT HOUR)`, "\n")
```

```
Number of hours with non-zero rented bike count: 24
```

APPENDIX-

- EDA with SQL-

Task 3 - Weather Outlook

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 query.

Solution 3

[36]:
```
# provide your solution here

# Execute SQL query to get the weather forecast for Seoul over the next 3 hours
query <- "SELECT * FROM CITIES_WEATHER_FORECAST WHERE CITY = 'Seoul' LIMIT 1"
result <- dbGetQuery(con, query)

# Print the result
print(result)
```

```
   CITY WEATHER VISIBILITY  TEMP TEMP_MIN TEMP_MAX PRESSURE HUMIDITY WIND_SPEED
1 Seoul   Clear      10000 12.32    10.91    12.32     1015       50       2.18
  WIND_DEG SEASON FORECAST_DATETIME
1      248 Spring        1618574400
```

Task 4 - Seasons

Find which seasons are included in the seoul bike sharing dataset.

Solution 4

[37]:
```
# provide your solution here


# Execute SQL query to find distinct seasons in the SEOUL_BIKE_SHARING dataset
query <- "SELECT DISTINCT SEASONS FROM SEOUL_BIKE_SHARING"
result <- dbGetQuery(con, query)

# Print the result
cat("Distinct seasons in SEOUL_BIKE_SHARING:", toString(result$SEASONS), "\n")
```

```
Distinct seasons in SEOUL_BIKE_SHARING: Winter, Spring, Summer, Autumn
```

Task 5 - Date Range

Find the first and last dates in the Seoul Bike Sharing dataset.

Solution 5

[38]:
```
# provide your solution here

# Execute SQL query to find the first date in the SEOUL_BIKE_SHARING dataset
query_first_date <- "SELECT MIN(DATE) AS FIRST_DATE FROM SEOUL_BIKE_SHARING"
result_first_date <- dbGetQuery(con, query_first_date)

# Execute SQL query to find the last date in the SEOUL_BIKE_SHARING dataset
query_last_date <- "SELECT MAX(DATE) AS LAST_DATE FROM SEOUL_BIKE_SHARING"
result_last_date <- dbGetQuery(con, query_last_date)

# Print the results
cat("First date in SEOUL_BIKE_SHARING:", result_first_date$FIRST_DATE, "\n")
cat("Last date in SEOUL_BIKE_SHARING:", result_last_date$LAST_DATE, "\n")
```

```
First date in SEOUL_BIKE_SHARING: 01/01/2018
Last date in SEOUL_BIKE_SHARING: 31/12/2017
```

Task 6 - Subquery - 'all-time high'

determine which date and hour had the most bike rentals.

Solution 6

[39]:
```
# provide your solution here


# Execute SQL query to find the date and hour with the most bike rentals
query_max_rentals <- "SELECT DATE, HOUR, MAX(RENTED_BIKE_COUNT) AS MAX_RENTALS
                      FROM SEOUL_BIKE_SHARING
                      GROUP BY DATE, HOUR
                      ORDER BY MAX_RENTALS DESC
                      LIMIT 1"
result_max_rentals <- dbGetQuery(con, query_max_rentals)

# Print the result
cat("Date and hour with the most bike rentals:",
    "Date:", result_max_rentals$DATE,
    "Hour:", result_max_rentals$HOUR,
    "Max Rentals:", result_max_rentals$MAX_RENTALS, "\n")
```

```
Date and hour with the most bike rentals: Date: 19/06/2018 Hour: 18 Max Rentals: 3556
```

APPENDIX-

- EDA with SQL-

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

[40]:
```
# provide your solution here

# Execute SQL query to calculate average hourly temperature and average bike rentals per hour by season
query_avg_hourly <- "SELECT SEASONS,
                            HOUR,
                            AVG(TEMPERATURE) AS AVG_TEMPERATURE,
                            AVG(RENTED_BIKE_COUNT) AS AVG_BIKE_COUNT
                     FROM SEOUL_BIKE_SHARING
                     GROUP BY SEASONS, HOUR
                     ORDER BY AVG_BIKE_COUNT DESC
                     LIMIT 10"
result_avg_hourly <- dbGetQuery(con, query_avg_hourly)

# Print the result
print(result_avg_hourly)
```

```
   SEASONS HOUR AVG_TEMPERATURE AVG_BIKE_COUNT
1   Summer   18        29.38791       2135.141
2   Autumn   18        16.03185       1983.333
3   Summer   19        28.27378       1889.250
4   Summer   20        27.06630       1801.924
5   Summer   21        26.27826       1754.065
6   Spring   18        15.97222       1689.311
7   Summer   22        25.69891       1567.870
8   Autumn   17        17.27778       1562.877
9   Summer   17        30.07691       1526.293
10  Autumn   19        15.06346       1515.568
```

APPENDIX-

- EDA with SQL-

Task 8 - Rental Seasonality

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

1]:
```
# provide your solution here

# Execute SQL query to calculate average, minimum, maximum, and standard deviation of hourly bike count by season
query_seasonality <- "SELECT SEASONS,
                             HOUR,
                             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_CO
                      FROM SEOUL_BIKE_SHARING
                      GROUP BY SEASONS, HOUR"
result_seasonality <- dbGetQuery(con, query_seasonality)

# Print the result
print(result_seasonality)
```

```
   SEASONS HOUR AVG_BIKE_COUNT MIN_BIKE_COUNT MAX_BIKE_COUNT STD_DEV_BIKE_COUNT
1   Autumn    0      709.43750            119           1336          219.14298
2   Autumn    1      552.50000            144           1001          191.54216
3   Autumn    2      377.47500             55            785          144.90134
4   Autumn    3      256.55000             28            514          102.53108
5   Autumn    4      169.02500             24            338           58.63957
6   Autumn    5      163.41250             24            264           53.88174
```

Task 9 - Weather Seasonality

Consider the weather over each season. On average, what were the TEMPERATURE, HUMIDITY, WIN
season?

Include the average bike count as well , and rank the results by average bike count so you can see if it is co

Solution 9

2]:
```
# provide your solution here

# Execute SQL query to calculate average weather parameters and bike count per season
query_weather_seasonality <- "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_weather_seasonality <- dbGetQuery(con, query_weather_seasonality)

# Print the result
print(result_weather_seasonality)
```

```
  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
```

APPENDIX-

- EDA with SQL-

Solution 10

[43]:
```
# provide your solution here

# Execute SQL query to join WORLD_CITIES and BIKE_SHARING_SYSTEMS and retrieve city information for Seoul
query_seoul_info <- "SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, SUM(BSS.BICYCLES) AS TOTAL_BIKES
                     FROM WORLD_CITIES WC
                     JOIN BIKE_SHARING_SYSTEMS BSS ON WC.CITY = BSS.CITY
                     WHERE WC.CITY = 'Seoul'
                     GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION"
result_seoul_info <- dbGetQuery(con, query_seoul_info)

# Print the result
print(result_seoul_info)
```

```
   CITY      COUNTRY     LAT LNG POPULATION TOTAL_BIKES
1 Seoul Korea, South 37.5833 127   21794000       20000
```

Task 11 - Find all city names and coordinates with comparable bike scale to Seoul's bike sharing system

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.

Later we will ask you to visualize these similar cities on leaflet, with some weather data.

Solution 11

[44]:
```
# provide your solution here

# Execute SQL query to find cities with comparable bike scale to Seoul's bike sharing system
query_comparable_cities <- "SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, BSS.BICYCLES AS TOTAL_BIKES
                            FROM WORLD_CITIES WC
                            JOIN BIKE_SHARING_SYSTEMS BSS ON WC.CITY = BSS.CITY
                            WHERE BSS.BICYCLES BETWEEN 15000 AND 20000
                            GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, BSS.BICYCLES"
result_comparable_cities <- dbGetQuery(con, query_comparable_cities)

# Print the result
print(result_comparable_cities)
```

```
      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    Seoul Korea, South 37.5833 127.0000   21794000       20000
4 Shanghai        China 31.1667 121.4667   22120000       19165
5  Weifang        China 36.7167 119.1000    9373000       20000
6  Zhuzhou        China 27.8407 113.1469    3855609       20000
```

[46]:
```
dbDisconnect(con)
# close(con)
```

APPENDIX-

- EDA with Data Visualization-

Task 10 - Create a scatter plot of `RENTED_BIKE_COUNT` vs `DATE`.

Tune the opacity using the `alpha` parameter such that the points don't obscure each other too much.

Solution 10

```
# provide your solution here
# Convert DATE column to date format
seoul_bike_sharing$DATE <- as.Date(seoul_bike_sharing$DATE)

# Create scatter plot
ggplot(seoul_bike_sharing, aes(x = DATE, y = RENTED_BIKE_COUNT)) +
  geom_point(alpha = 0.3) +  # Adjust alpha for opacity
  labs(title = "Rented Bike Count over Time",
       x = "Date", y = "Rented Bike Count") +
  theme_minimal()
```

Task 11 - Create the same plot of the `RENTED_BIKE_COUNT` time series, but now add `HOURS` as the colour.

Solution 11

```
# provide your solution here
seoul_bike_sharing$DATE <- as.Date(seoul_bike_sharing$DATE)

# Create scatter plot with HOUR as color
ggplot(seoul_bike_sharing, aes(x = DATE, y = RENTED_BIKE_COUNT, color = as.factor(HOUR))) +
  geom_point(alpha = 0.3) +  # Adjust alpha for opacity
  labs(title = "Rented Bike Count over Time with Hourly Variation",
       x = "Date", y = "Rented Bike Count", color = "Hour of the day") +
  theme_minimal()
```

APPENDIX-

- EDA with Data Visualization-

Task 12 - Create a histogram overlaid with a kernel density curve

Normalize the histogram so the y axis represents 'density'. This can be done by setting `y=..density..` in the aesthetics of the histogram.

▼ Click here for a hint

Set the colour to something like black and the fill to white so you can see the kernel density plot layer better.

▼ Click here for another hint

Set the color and alpha such that your denstiy plot is clearly visible, without obscuring the histogram.

Solution 12

```
# provide your solution here
# Create histogram with kernel density curve
ggplot(seoul_bike_sharing, aes(x = RENTED_BIKE_COUNT)) +
  geom_histogram(aes(y = ..density..), fill = "white", color = "black", alpha = 0.6) +
  geom_density(color = "blue", alpha = 0.8) +  # Adjust color and alpha for the density plot
  labs(title = "Distribution of Rented Bike Count",
       x = "Rented Bike Count", y = "Density") +
  theme_minimal()
```

```
# provide your solution here

# Convert DATE column to date format
seoul_bike_sharing$DATE <- as.Date(seoul_bike_sharing$DATE)

# Grouping data by DATE and calculating total rainfall and snowfall for each day
daily_totals <- seoul_bike_sharing %>%
  group_by(DATE) %>%
  summarise(total_rainfall = sum(RAINFALL, na.rm = TRUE),
            total_snowfall = sum(SNOWFALL, na.rm = TRUE))

# Create a bar chart for daily total rainfall and snowfall
ggplot(daily_totals, aes(x = DATE)) +
  geom_bar(aes(y = total_rainfall), stat = "identity", fill = "blue", alpha = 0.7) +
  geom_bar(aes(y = total_snowfall), stat = "identity", fill = "white", color = "black", alpha = 0.7) +
  labs(title = "Daily Total Rainfall and Snowfall",
       x = "Date", y = "Total Amount") +
  theme_minimal()
```