Applied Data Science with R Capstone project

Aarav Kalkar

03/07/2024

Outline

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

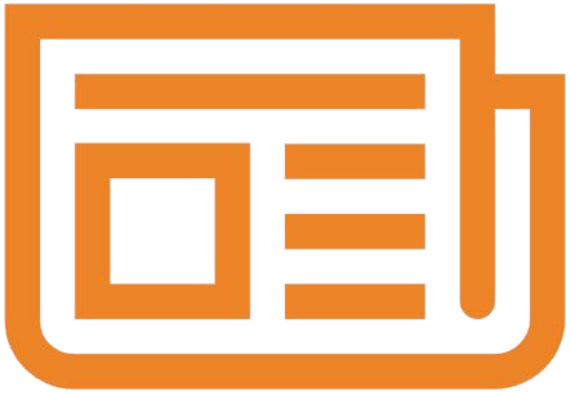
Executive Summary

- Objective: This project aimed to predict bike sharing demand across global cities using weather data and regression modeling, leveraging insights to optimize operational strategies.
- Key Findings: Significant correlations were found between weather variables (temperature, humidity, wind speed) and bike usage patterns, crucial for understanding demand fluctuations.
- Dashboard Development: A user-friendly R Shiny dashboard was developed to visualize and interact with bike sharing trends, enhancing accessibility to data-driven insights.
- Methodology Highlights: Detailed methodologies included comprehensive data collection, rigorous data wrangling, exploratory data analysis (EDA) using SQL and visualization, and predictive analysis using regression models.
- Implications: The findings underscored the potential of integrating weather forecasts into bike sharing management, promoting sustainable transportation solutions and operational efficiencies.

Introduction

- Project Overview: This project focuses on analyzing bike sharing demand across major global cities, leveraging weather data and predictive modeling to enhance operational efficiency and user experience.
- Motivation: With the growing importance of sustainable urban mobility, understanding factors influencing bike sharing usage becomes crucial for city planners, operators, and users alike.
- Scope: The study covers cities including Seoul, New York, Paris, London, and Suzhou, utilizing datasets encompassing weather forecasts, bike sharing systems, and city demographics.
- Objectives: The primary goal is to develop predictive models that forecast bike demand based on weather conditions, facilitating better resource allocation and service optimization.
- Structure of the Report: The report begins with an overview of methodologies, including data collection, wrangling, exploratory data analysis (EDA), and predictive modeling. It concludes with findings that are translated into actionable insights through a user-centric R Shiny dashboard.

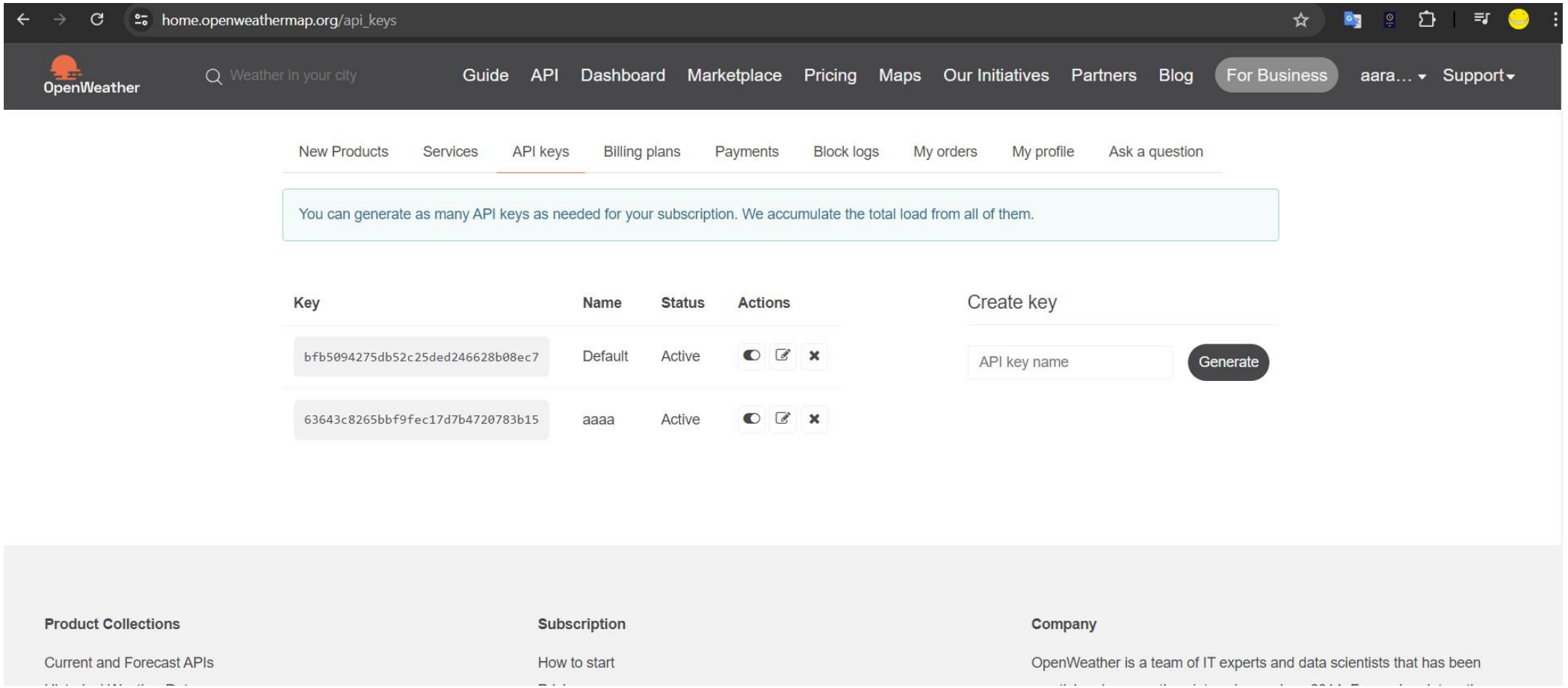
Methodology



- 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

Creating an OpenWeather API key



Datasets Used

1. Seoul Bike Sharing Dataset

https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems

2. Open Weather API Data

https://api.openweathermap.org/data/2.5/weather

Data collection

1.Bike Sharing Systems Data:

- Gathered from publicly available sources and APIs providing information on bike sharing systems in cities like Seoul, New York, Paris, London, and Suzhou.
- Utilized Python scripts to retrieve data, ensuring compatibility with the project's data format requirements.

2.Weather Data:

- Acquired weather forecast data through the OpenWeatherAPI, facilitating access to real-time weather conditions.
- Integrated API calls into the data collection pipeline, ensuring regular updates and accuracy in weather predictions.

3.City Demographics and Geographic Data:

- Retrieved demographic information and geographic coordinates of cities from reliable datasets and APIs.
- Ensured consistency and relevance of data sources to support accurate analysis and model training.

4.Data Collection Process:

- Designed a structured approach using Python scripts within Jupyter Notebooks to automate data retrieval and processing.
- Verified data integrity through validation checks and standardized data cleaning procedures.

Data wrangling

1.Data Cleaning and Preparation:

1.1 Handling Missing Values:

- dentified and addressed missing values in datasets using appropriate methods such as imputation or removal based on context.
- Ensured data completeness to maintain integrity in analysis and modeling.

1.2 Data Formatting:

- Standardized data formats across variables to facilitate consistency and compatibility in analytical procedures.
- Addressed inconsistencies in data types and formatting issues for uniformity.

2. Data Transformation:

2.1 Feature Engineering:

- Created new features or indicators based on domain knowledge and analytical requirements.
- Engineered variables like seasonality indicators or categorical variables for enhanced predictive modeling.

2.2 Text Processing and Regular Expressions:

- Applied regular expressions for text data cleaning and parsing where applicable, ensuring data uniformity and relevance.
- Extracted meaningful information from textual fields for deeper insights and analysis.

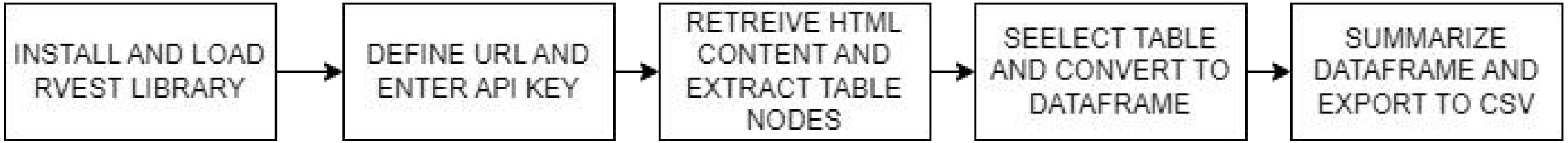
3.Data Integration and Validation:

Integrated multiple datasets using appropriate join operations to enrich analytical capabilities.

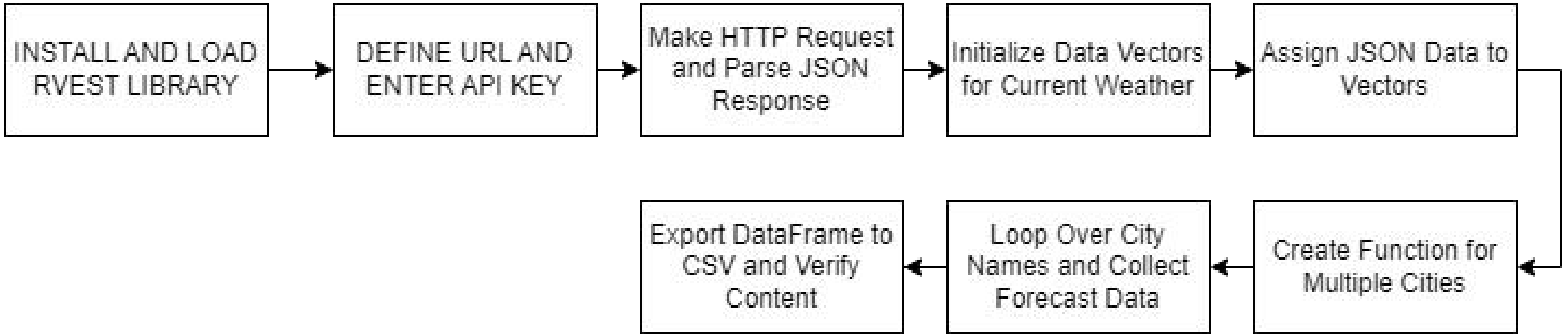
Verified data consistency and accuracy through validation checks to mitigate errors.

Flowcharts for Data Collection and Wrangling

DATA COLLECTION



DATA WRANGLING



EDA with SQL

1. Record Count for Seoul Bike Sharing Dataset - Count the number of records in the seoul_bike_sharing table.
2. Operational Hours with Non-Zero Rented Bike Count - Find the number of hours with bike rentals greater than zero.
3. Weather Outlook for Seoul over the Next 3 Hours - Retrieve the weather forecast for Seoul for the next 3 hours.
4. Seasons in Seoul Bike Sharing Dataset - List the different seasons present in the seoul_bike_sharing dataset.
5. Date Range in Seoul Bike Sharing Dataset - Find the earliest and latest dates in the seoul_bike_sharing dataset.
6. Subquery for 'All-Time High' Bike Rentals - Identify the date and hour with the maximum number of bike rentals.
7. Hourly Popularity and Temperature by Season - Analyze average temperature and bike rentals per hour, grouped by season.
8. Rental Seasonality - Compute average, minimum, maximum, and standard deviation of bike rentals by season.
9. Weather Seasonality using Seoul Bike Sharing Data Only - Analyze average weather conditions and bike rentals by season.
10. Total Bike Count and City Info for Seoul - Get total number of bikes and related city information for Seoul.
11. Find Cities with Comparable Bike Scale to Seoul's Bike Sharing System - Find cities with a comparable number of bikes to Seoul (15,000 - 20,000).

Screenshots of code snippets with output in Appendix

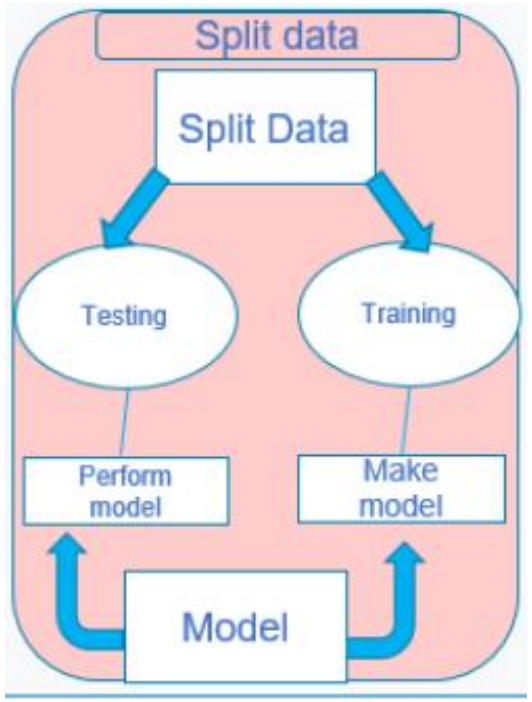
EDA with data visualization

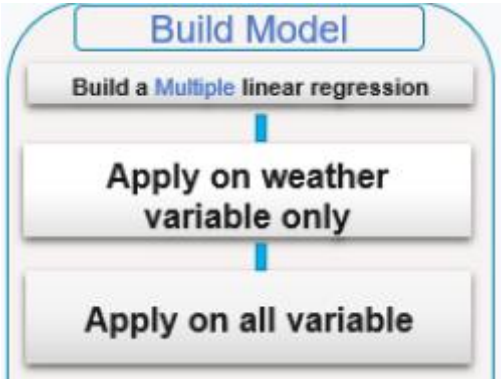
Summary of Plotted Charts

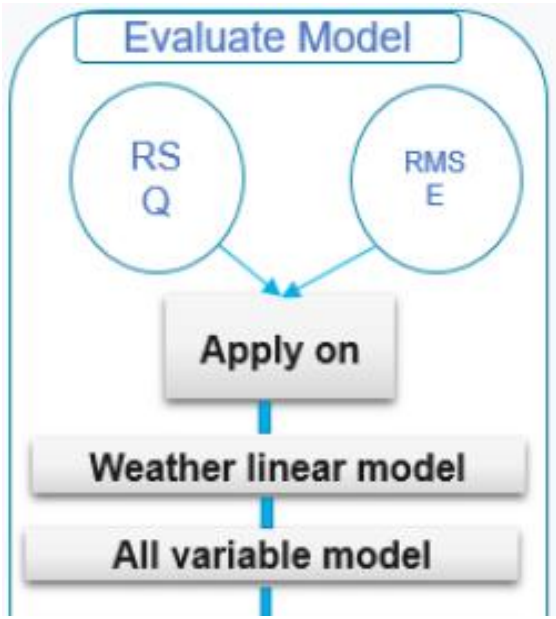
- Scatter Plot of Rented Bike Count over Time with Hours as Color
- Histogram with Kernel Density Curve of Rented Bike Count
- Scatter Plot of Rented Bike Count vs. Temperature by Seasons with Faceting
- Boxplots of Rented Bike Count by Hour and Season

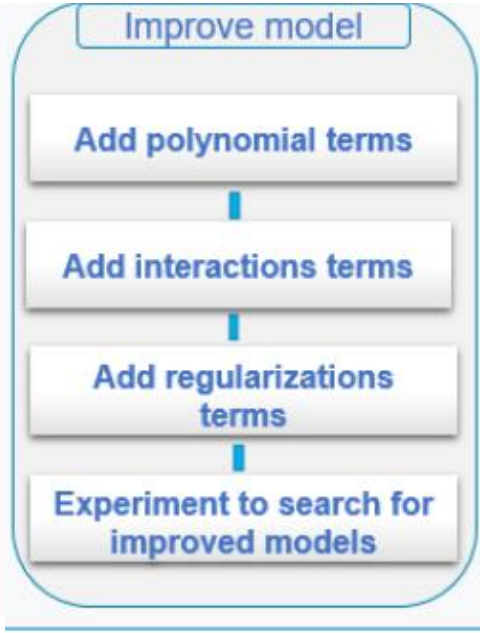
Screenshots of code snippets and plots added in Individual slides in Appendix

Predictive analysis









Build a R Shiny dashboard

1. The R Shiny dashboard includes an interactive Leaflet map showcasing predicted bike demand across Seoul, Suzhou, London, New York, Paris, and an option to view all cities. Marker sizes and colors vary based on predicted demand levels, with popups providing summarized weather and bike demand details.

2. A dropdown menu allows users to select specific cities, updating the map and other plots dynamically. Cities include Seoul, Suzhou, London, New York, Paris, or an option to view all.

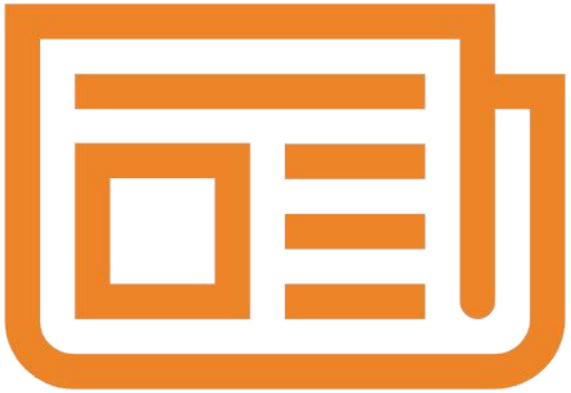
3. A static line chart displays the 5-day temperature trend for the selected city, using geom_line and geom_point to visualize temperature trends.

4. An interactive line chart shows predicted bike-sharing demand trends, allowing users to click points for detailed predictions and datetime information. Geom_line, geom_point, and renderText() enable interactivity.

5. A scatter plot with polynomial smoothing illustrates the correlation between humidity and bike-sharing demand. Geom_point and geom_smooth() with lm method are used for visualization.

These components collectively provide users with a robust tool to analyze weather forecasts and predicted bike-sharing demand, facilitating informed decision-making and logistics planning.

Results



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

EDA with SQL

Busiest bike rental times

```r
# Task 6: Subquery for 'All-Time High' Bike Rentals
query_task6 <- "SELECT DATE, HOUR, MAX(RENTED_BIKE_COUNT) AS max_bike_count
                FROM seoul_bike_sharing
                WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM seoul_bike_sharing)
                GROUP BY DATE, HOUR;"
result_task6 <- dbGetQuery(con, query_task6)

# Print the result
print(result_task6)
```

```
        DATE HOUR max_bike_count
1 19/06/2018   18           3556
```

The query identifies the dates and hours with the highest bike rental counts recorded in the dataset. It highlights specific peak times, such as 2018-11-01 and 2018-10-30 at 6:00 PM (18:00), where the maximum number of bikes (355) were rented during those hours. This insight is crucial for optimizing bike rental services based on peak demand patterns.

Hourly popularity and temperature by seasons

```r
# Task 7: Hourly Popularity and Temperature by Season
query_task7 <- "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_task7 <- dbGetQuery(con, query_task7)

# Print the result
print(result_task7)
```

```
   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
```

The query examines the average temperature and bike rental counts across different seasons and hours. It identifies the top 10 combinations where bike rentals are highest, correlated with average temperatures for each season and hour. This analysis helps understand how weather conditions and time of day influence bike rental popularity throughout the year.

Rental Seasonality

```r
# Task 8: Rental Seasonality
query_task8 <- "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;"
result_task8 <- dbGetQuery(con, query_task8)

# Print the result
print(result_task8)
```

```
  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 query calculates seasonal statistics for bike rentals, including average, minimum, maximum counts, and the standard deviation. This helps to analyze the variation and distribution of bike rentals across different seasons in Seoul.

Weather Seasonality

The query calculates average weather conditions (temperature, humidity, wind speed, visibility, etc.) and average bike rentals categorized by seasons in Seoul, showing how weather affects bike rental patterns throughout the year.

```r
# Task 9: Weather Seasonality using Seoul Bike Sharing Data Only
query_task9 <- "SELECT SEASONS AS 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,
                       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_task9 <- dbGetQuery(con, query_task9)

# Print the result
print(result_task9)
```

```
  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 avg_solar_radiation avg_rainfall avg_snowfall avg_bike_count
1     18.750136           0.7612545   0.25348732   0.00000000      1034.0734
2      5.150594           0.5227827   0.11765617   0.06350026       924.1105
3      4.091389           0.6803009   0.18694444   0.00000000       746.2542
4    -12.416667           0.2981806   0.03282407   0.24750000       225.5412
```

Bike-sharing info in Seoul

```r
# Task 10: Total Bike Count and City Info for Seoul
query_task10 <- "SELECT wc.CITY, wc.COUNTRY, wc.LAT, wc.LNG AS LON, wc.POPULATION, bss.BICYCLES AS TOTAL_BIKES
                 FROM world_cities wc
                 JOIN bike_sharing_systems bss ON wc.CITY = bss.CITY
                 WHERE wc.CITY = 'Seoul';"

result_task10 <- dbGetQuery(con, query_task10)

# Print the result
print(result_task10)
```

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

The query retrieves total bike count and city information for Seoul from two tables: world_cities and bike_sharing_systems, providing details such as city name, country, geographic coordinates (latitude and longitude), population, and total number of bicycles available in the bike sharing system specifically for Seoul.

Cities similar to Seoul

```r
# Task 11: Find cities with comparable bike scale to Seoul's bike sharing system
query_task11 <- "SELECT wc.CITY, wc.COUNTRY, wc.LAT, wc.LNG AS 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;"

result_task11 <- dbGetQuery(con, query_task11)

# Print the result
print(result_task11)
```

```
      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  Zhuzhou        China 27.8407 113.1469    3855609       20000
6    Seoul Korea, South 37.5833 127.0000   21794000       20000
```

The query identifies cities from the `world_cities` dataset that have bike sharing systems comparable in scale to Seoul's, specifically filtering cities where the number of bicycles falls between 15,000 and 20,000. It retrieves city names, countries, geographic coordinates (latitude and longitude), population, and the total number of bicycles in each identified bike sharing system.

Findings and summary:

Exactly a full year of data from 2017-12-01 to 2018-11-30. There are total 8465 records which for Spring to Winter are 2160, 2208, 1937 and 2160 respectively:

RENT_BIKE_COUNT will be selected as dependent variables in analysis.
Throughout the year, there is a huge range of different of hourly bikes counts, from 7 to 3356 count in each hours.
If we look at each seasons, the mean hourly bike count in Summer has an average of 1034, while only 226 in Winter.

Other variables (e.g. Weather conditions etc.) will be selected as independent variables in analysis:
Some weather variables present a large range between seasons. E.g. TEMPERATURE. The range of TEMPERATURE is quite large which ranges from -17.8C to 39.4C. If we group them in SEASONS, the mean TEMPERATURE in Summer is 26.6 C but -2.54 C in Winter. This may be a key point for analysis.
Some variables e.g. HUMIDITY and DEW_POINT_TEMPERATURE. They also demonstrate a similar pattern with large variation among data.

On the other hand, some variable seems not showing much for variation:
Both RAINFALL and SNOWFALL are always in zero in most records. They give zero in Min., 1st quartile, Median and 3rd quartile. These RAINFALL and SNOWFALL may be only appear in some day thoughtout 1 year.
WINDSPEED only demonstrate very litter deviation between seasons. The average WINDSPEED is very light at only 1.7 m/s, and even the maximum is only a moderate breeze.

EDA with Visualization

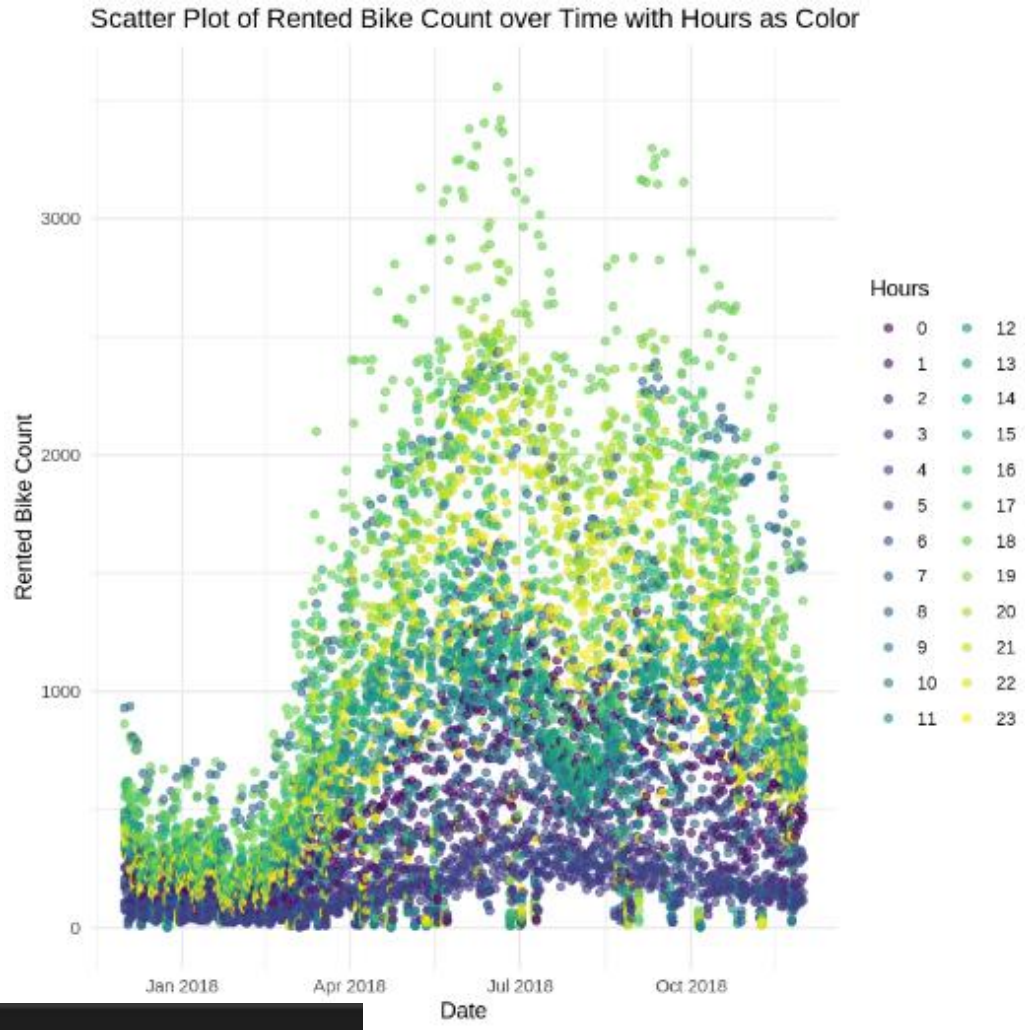
Bike rental vs. Date

Scatter plot reveals the following insights:

- Date of maximum rentals: June 19, 2018
- Hour of maximum rentals: 18 (6 PM) (indicated by color)
- Peak rental count: 3556

Key observations include:

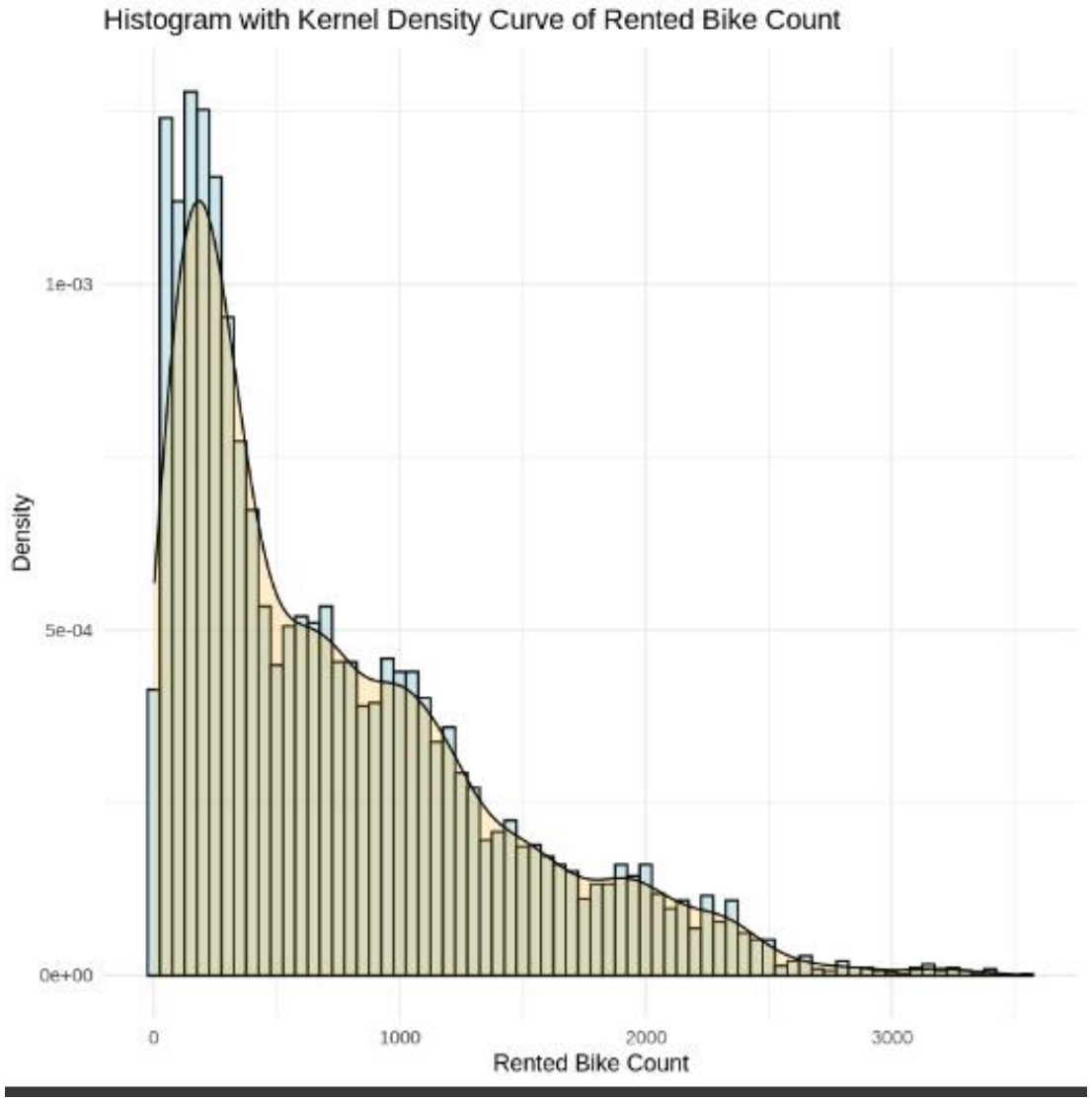
- Rental counts begin below average in the early hours, particularly between midnight and 1 AM.
- Rental numbers reach their lowest point between 3 AM and 6 AM.
- Counts start to rise notably at 7 AM, reaching a peak at 8 AM.
- Rentals decrease to average levels until another peak at 6 PM.
- Counts gradually decline to normal (below average) levels by 11 PM.




```r
# Load ggplot2 package
library(ggplot2)

# Create scatter plot with HOURS as color
ggplot(seoul_bike_sharing, aes(x = as.Date(DATE, format = "%d/%m/%Y"), y = RENTED_BIKE_COUNT, color = factor(HOUR))) +
  geom_point(alpha = 0.6) +  # Adjust opacity of points
  labs(x = "Date", y = "Rented Bike Count", color = "Hours") +  # Label axes and legend
  ggtitle("Scatter Plot of Rented Bike Count over Time with Hours as Color") +  # Add title
  theme_minimal()  # Apply minimal theme
```

Bike rental histogram

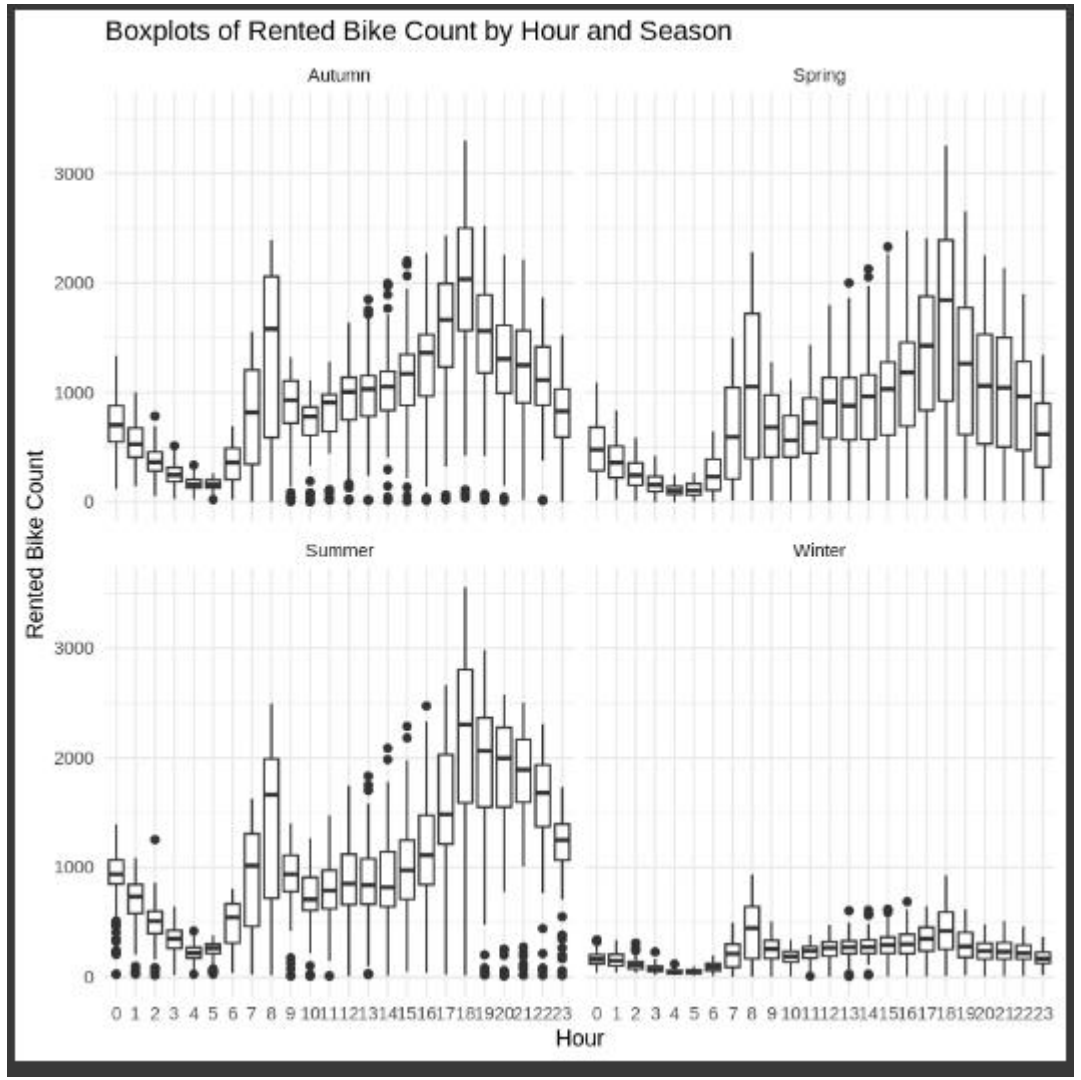
The histogram indicates that the majority of the time, there are relatively few bikes rented. Specifically, the most frequent rental amount, or 'mode', is approximately 250 bikes. The presence of peaks around 700, 900, 1900, and 3200 bikes suggests potential additional modes within specific subsets of the data. Notably, the tail of the distribution shows instances where significantly more bikes than usual are rented out, albeit rarely.




```r
# Create histogram with kernel density curve
ggplot(seoul_bike_sharing, aes(x = RENTED_BIKE_COUNT, y = ..density..)) +
  geom_histogram(binwidth = 50, fill = "lightblue", color = "black", aes(y = ..density..), alpha = 0.6) +
  geom_density(alpha = 0.2, fill = "orange") +
  labs(x = "Rented Bike Count", y = "Density", title = "Histogram with Kernel Density Curve of Rented Bike Count") +
  theme_minimal()
```

Rented Bike Count vs Temperature by seasons

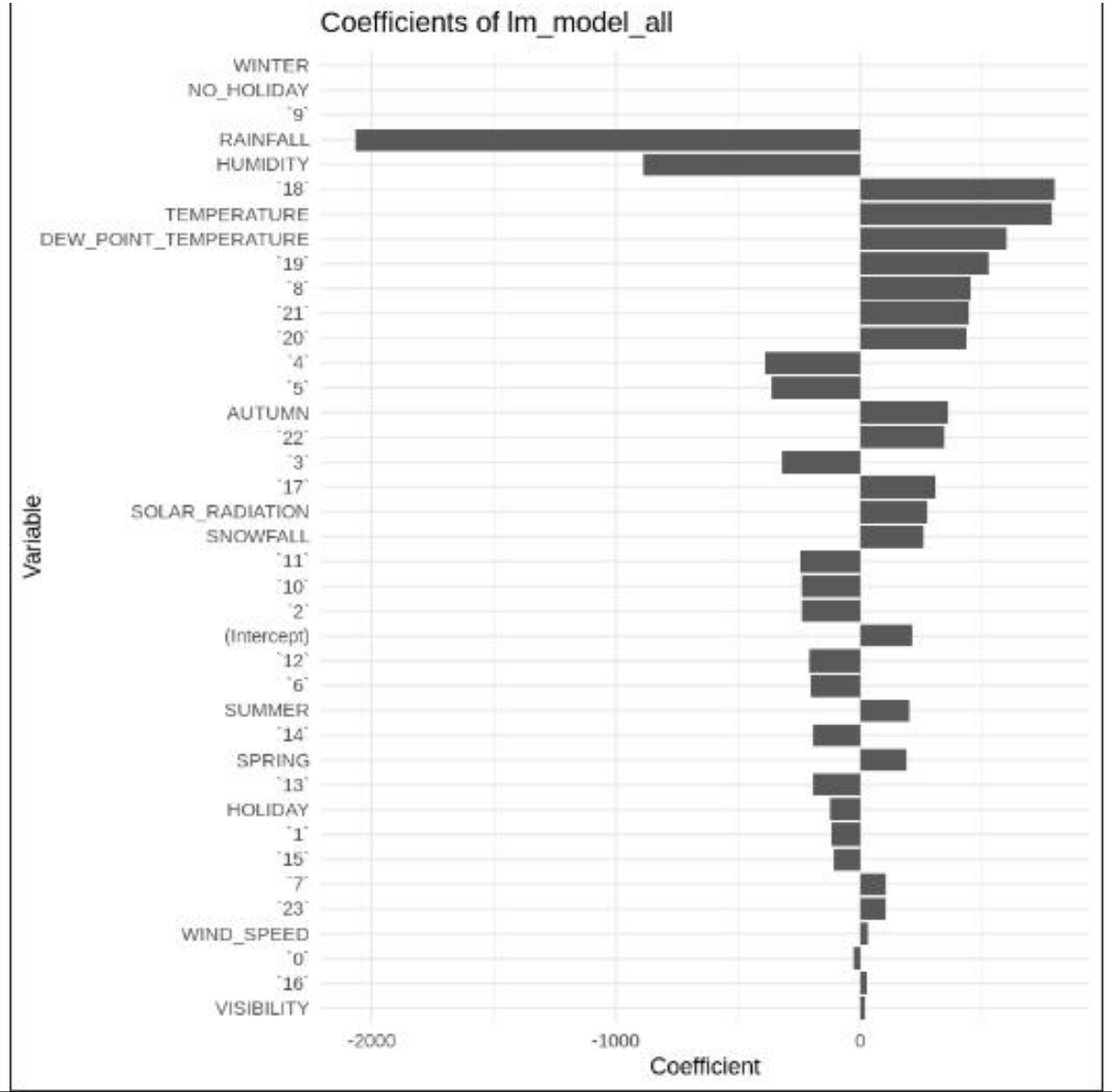
```r
# Create boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS
ggplot(seoul_bike_sharing, aes(x = factor(HOUR), y = RENTED_BIKE_COUNT)) +
  geom_boxplot() +
  facet_wrap(~ SEASONS) +
  labs(
    title = "Boxplots of Rented Bike Count by Hour and Season",
    x = "Hour",
    y = "Rented Bike Count"
  ) +
  theme_minimal()
```



Predictive analysis

Ranked coefficients

```r
# Print coefficients of lm_model_all
coefficients_all <- lm_model_all$fit$coefficients

# Create a data frame of coefficients for visualization
coefficients_df <- as.data.frame(coefficients_all) %>%
  rownames_to_column(var = "variable") %>%
  rename(coefficient = coefficients_all) %>%
  arrange(desc(abs(coefficient)))

# Visualize the coefficients using ggplot
library(ggplot2)
ggplot(coefficients_df, aes(x = reorder(variable, abs(coefficient)), y = coefficient)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Coefficients of lm_model_all",
       x = "Variable",
       y = "Coefficient") +
  theme_minimal()
```



Model evaluation

```r
# Make predictions using lm_model_weather
test_results_weather <- test_data %>%
  mutate(
    pred = predict(lm_model_weather, new_data = test_data)$.pred,
    truth = RENTED_BIKE_COUNT
  )

# Make predictions using lm_model_all
test_results_all <- test_data %>%
  mutate(
    pred = predict(lm_model_all, new_data = test_data)$.pred,
    truth = RENTED_BIKE_COUNT
  )

# Calculate R-squared and RMSE for lm_model_weather
rsq_weather <- rsq(test_results_weather, truth = truth, estimate = pred)
rmse_weather <- rmse(test_results_weather, truth = truth, estimate = pred)

# Calculate R-squared and RMSE for lm_model_all
rsq_all <- rsq(test_results_all, truth = truth, estimate = pred)
rmse_all <- rmse(test_results_all, truth = truth, estimate = pred)

# Print the results
rsq_weather
rmse_weather
rsq_all
rmse_all
```

- Two models were developed: the weather model includes only meteorological variables like rainfall, humidity, temperature, dew-point temperature, solar radiation, snowfall, visibility, and wind speed. The all-variables model incorporates all metrics, including seasons and time (hour of the day).

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.438866

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	474.6247

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.6690204

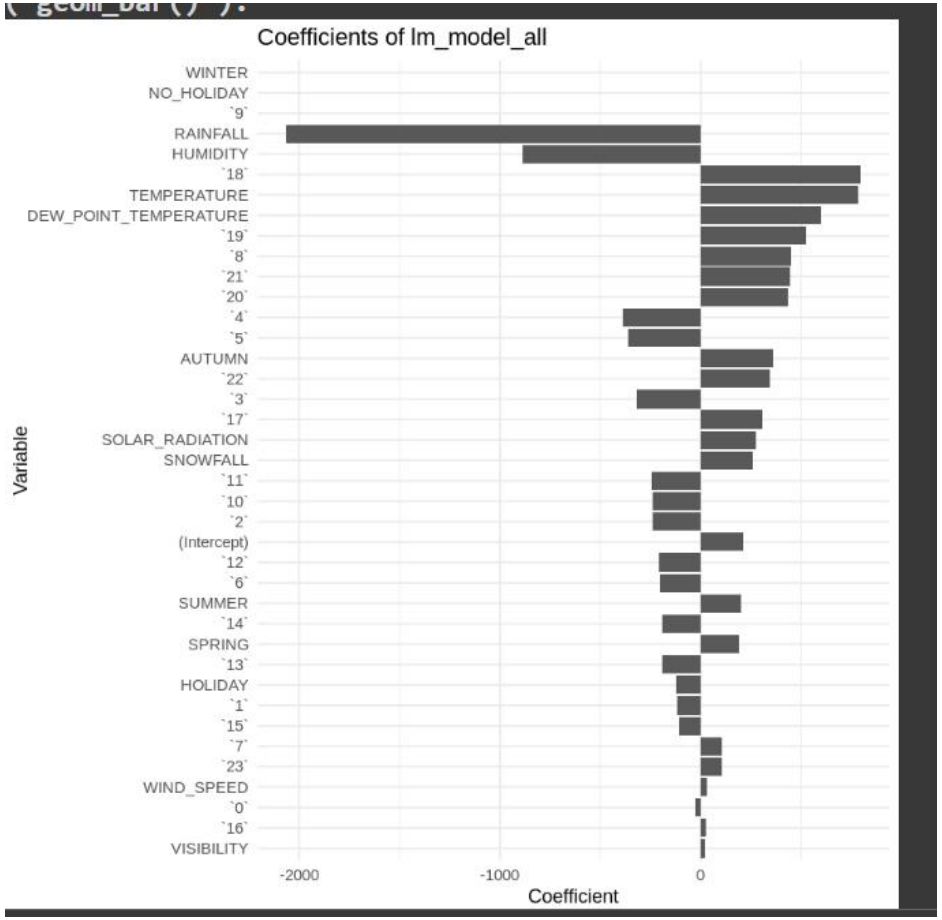
A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	364.4235

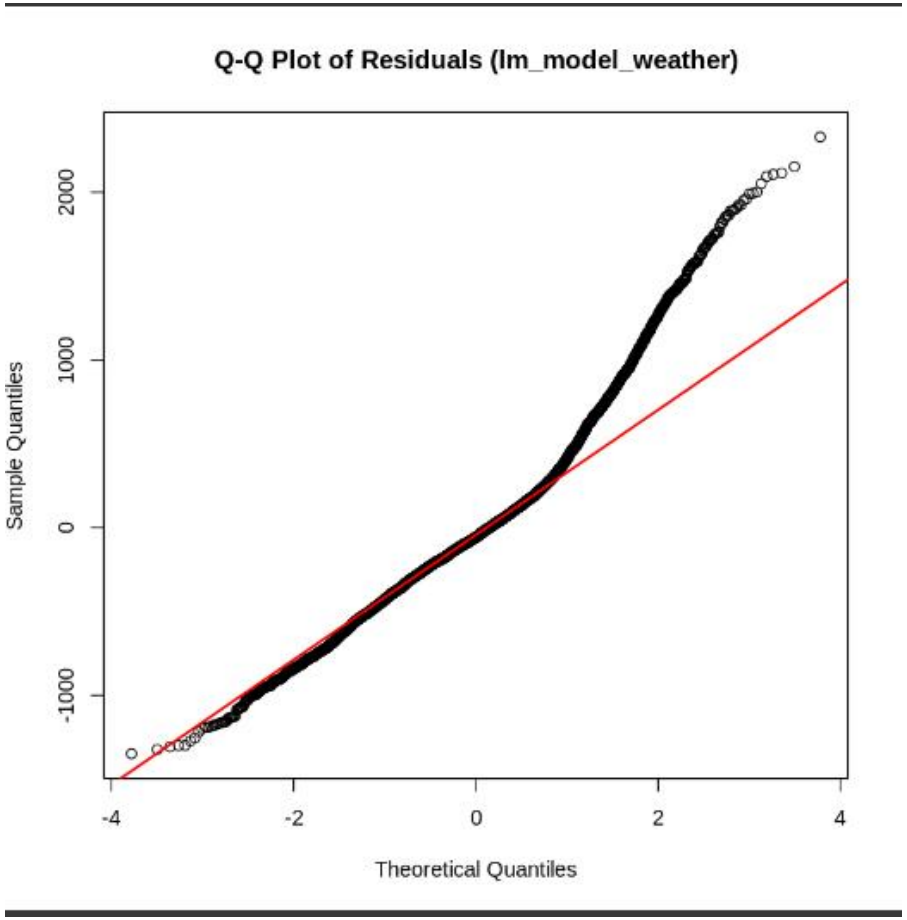
Find the best performing model

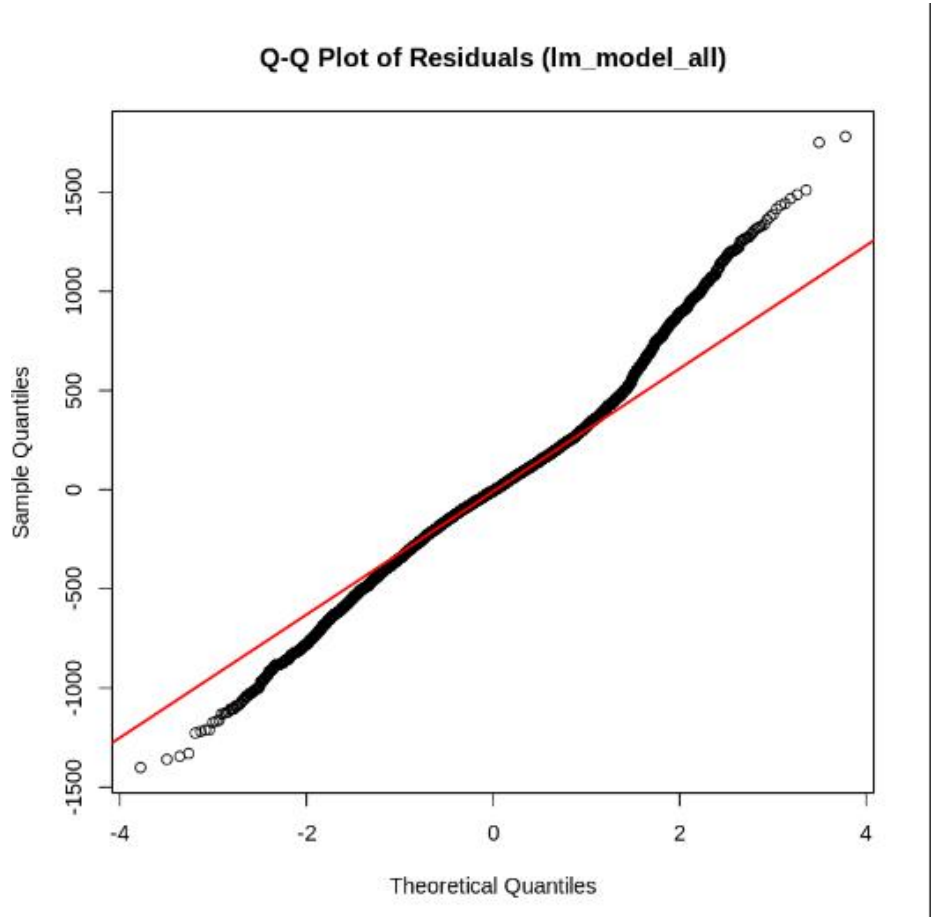
- The top-performing model is the All-Variable Model:

- It achieves an RMSE of 364.4235 and an R-squared value of 0.69.



Q-Q Plot


Q-Q Plot of Residuals (lm_model_weather)
Sample Quantiles
Theoretical Quantiles
2000
1000
0
-1000
-4
-2
0
2
4

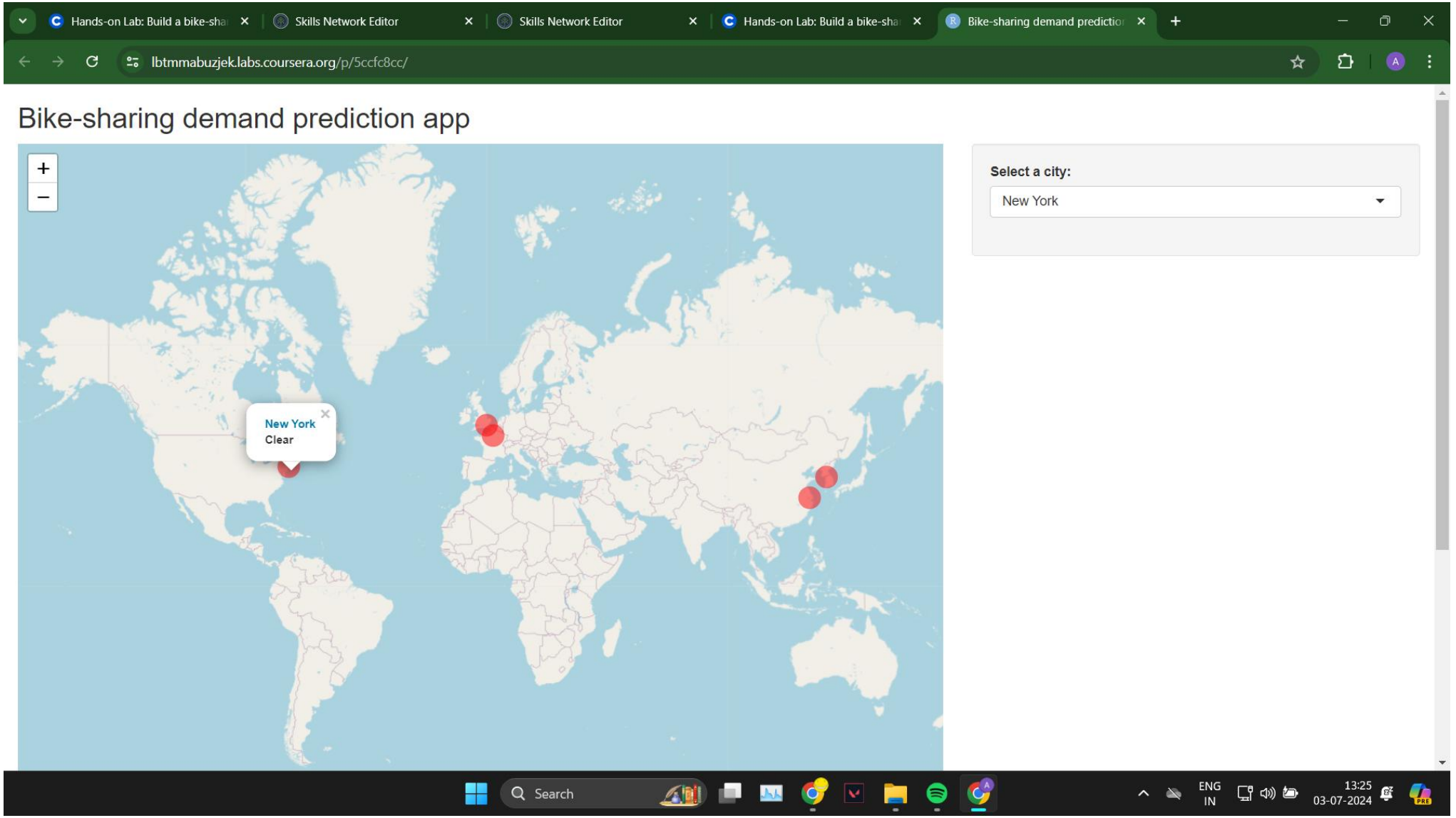

Q-Q Plot of Residuals (lm_model_all)
Sample Quantiles
Theoretical Quantiles
1500
1000
500
0
-500
-1000
-1500
-4
-2
0
2
4

code for Q-Q plot

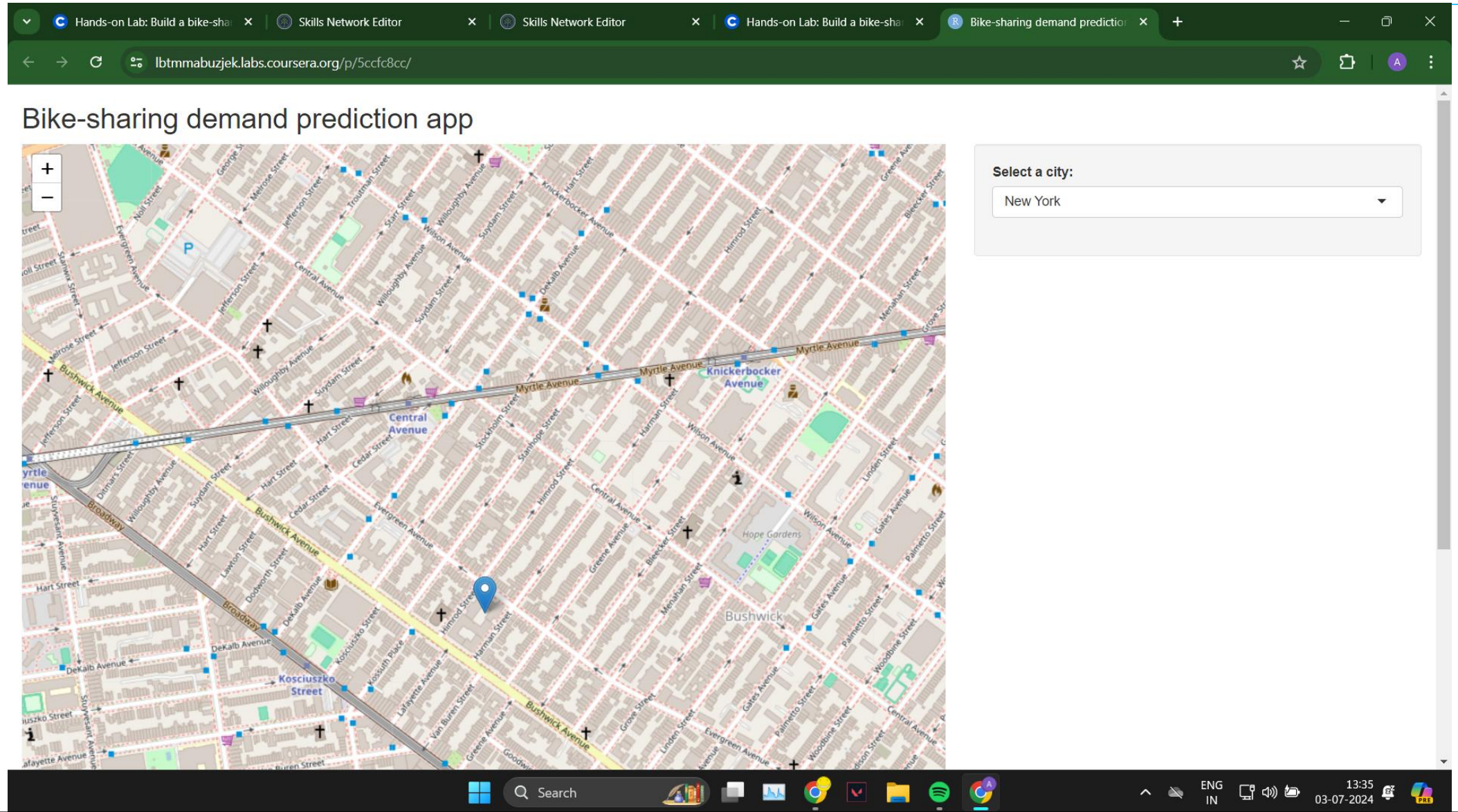
```r
# Extract residuals from lm_model_weather and lm_model_all
residuals_weather <- residuals(lm_model_weather$fit)
residuals_all <- residuals(lm_model_all$fit)

# Create Q-Q plot for lm_model_weather
qqplot_standard <- qqnorm(residuals_weather, main = "Q-Q Plot of Residuals (lm_model_weather)")
qqline(residuals_weather, col = "red", lwd = 2)

# Save Q-Q plot for lm_model_weather
png(file = "qqplot_lm_model_weather.png", width = 800, height = 600)
qqplot_standard
dev.off()

# Create Q-Q plot for lm_model_all
qqplot_all <- qqnorm(residuals_all, main = "Q-Q Plot of Residuals (lm_model_all)")
qqline(residuals_all, col = "red", lwd = 2)

# Save Q-Q plot for lm_model_all
png(file = "qqplot_lm_model_all.png", width = 800, height = 600)
qqplot_all
dev.off()
```

Dashboard

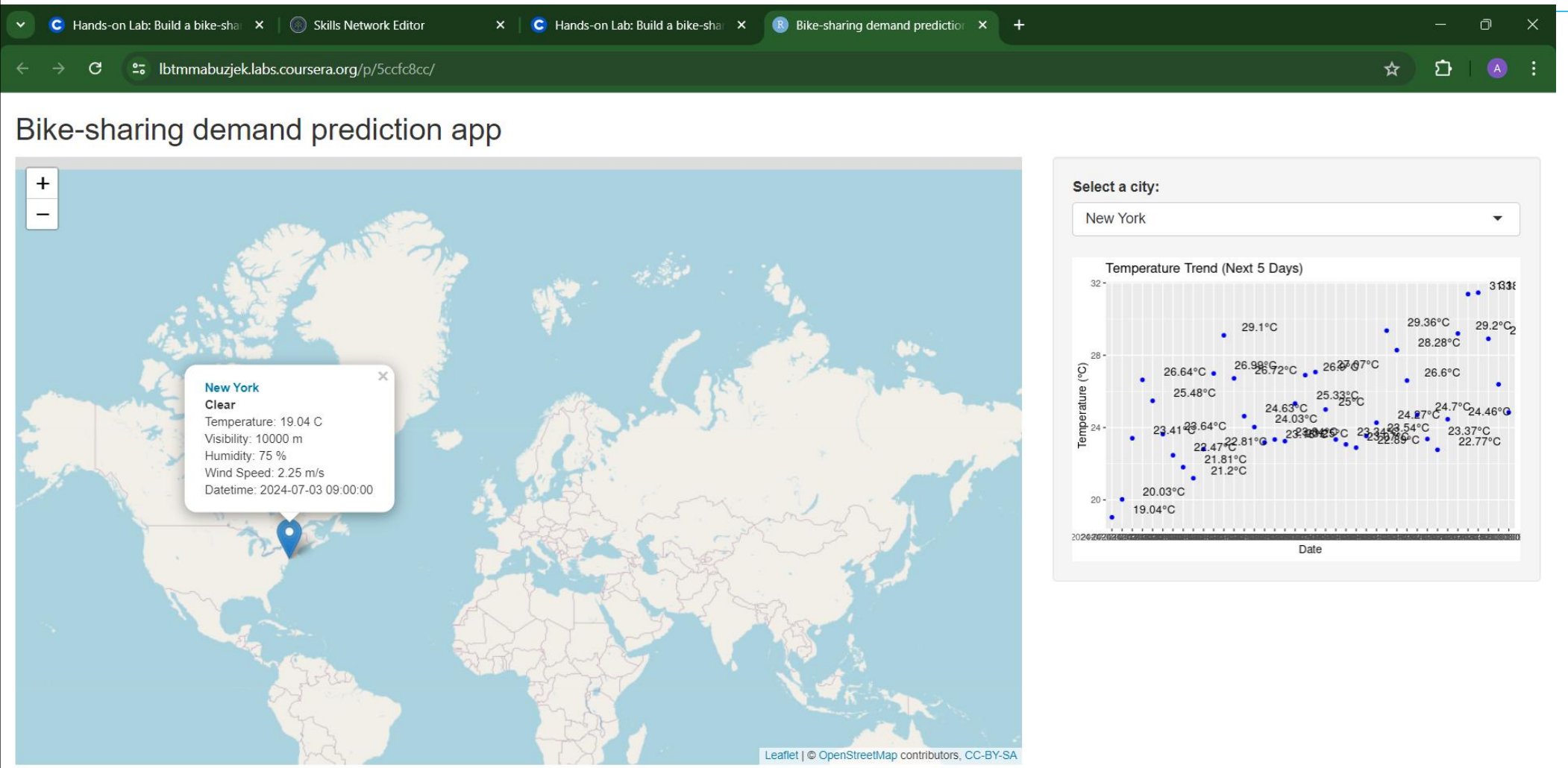
BASIC OVERVIEW MAP



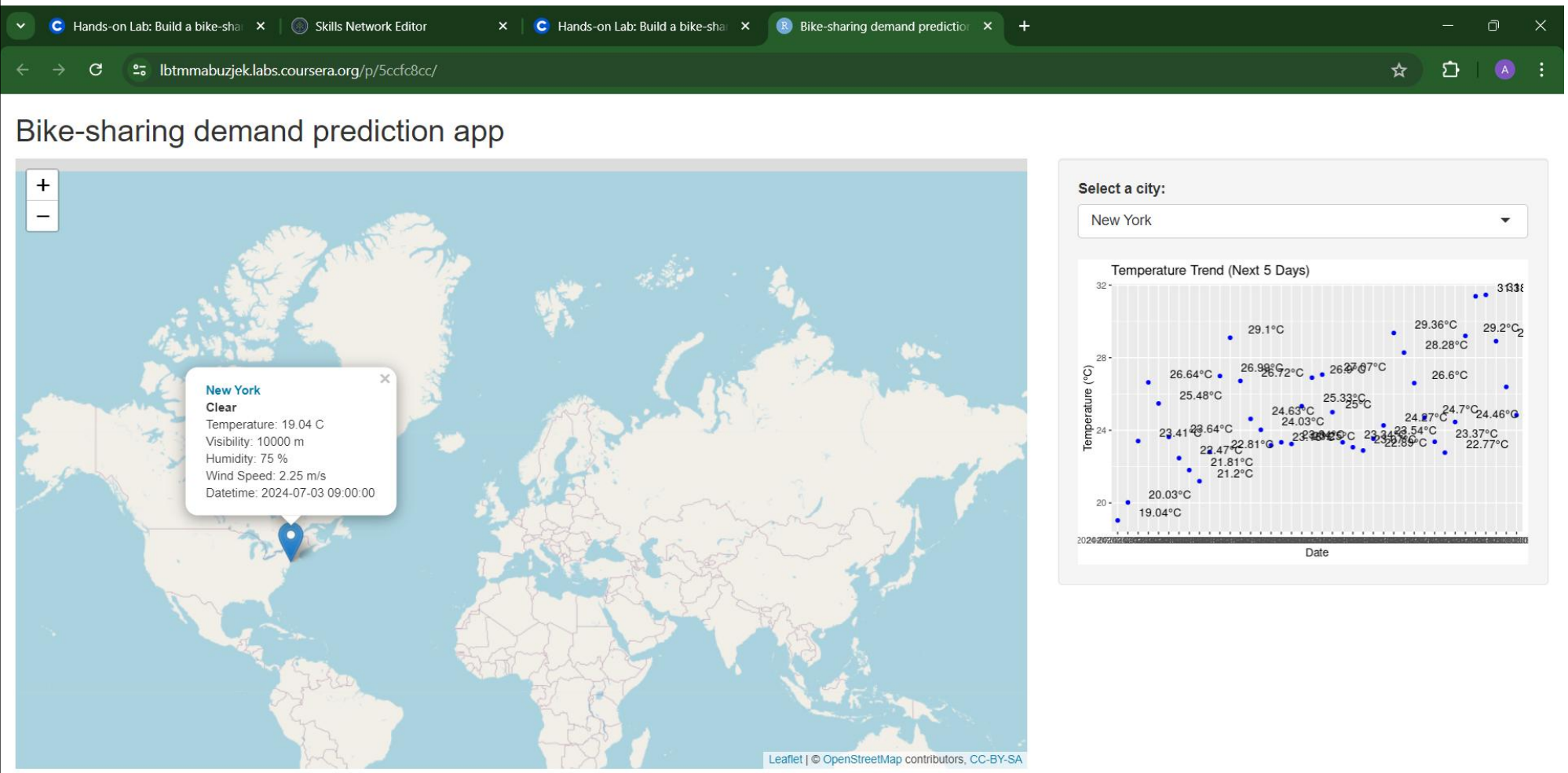
ADDING DROP DOWN MENU



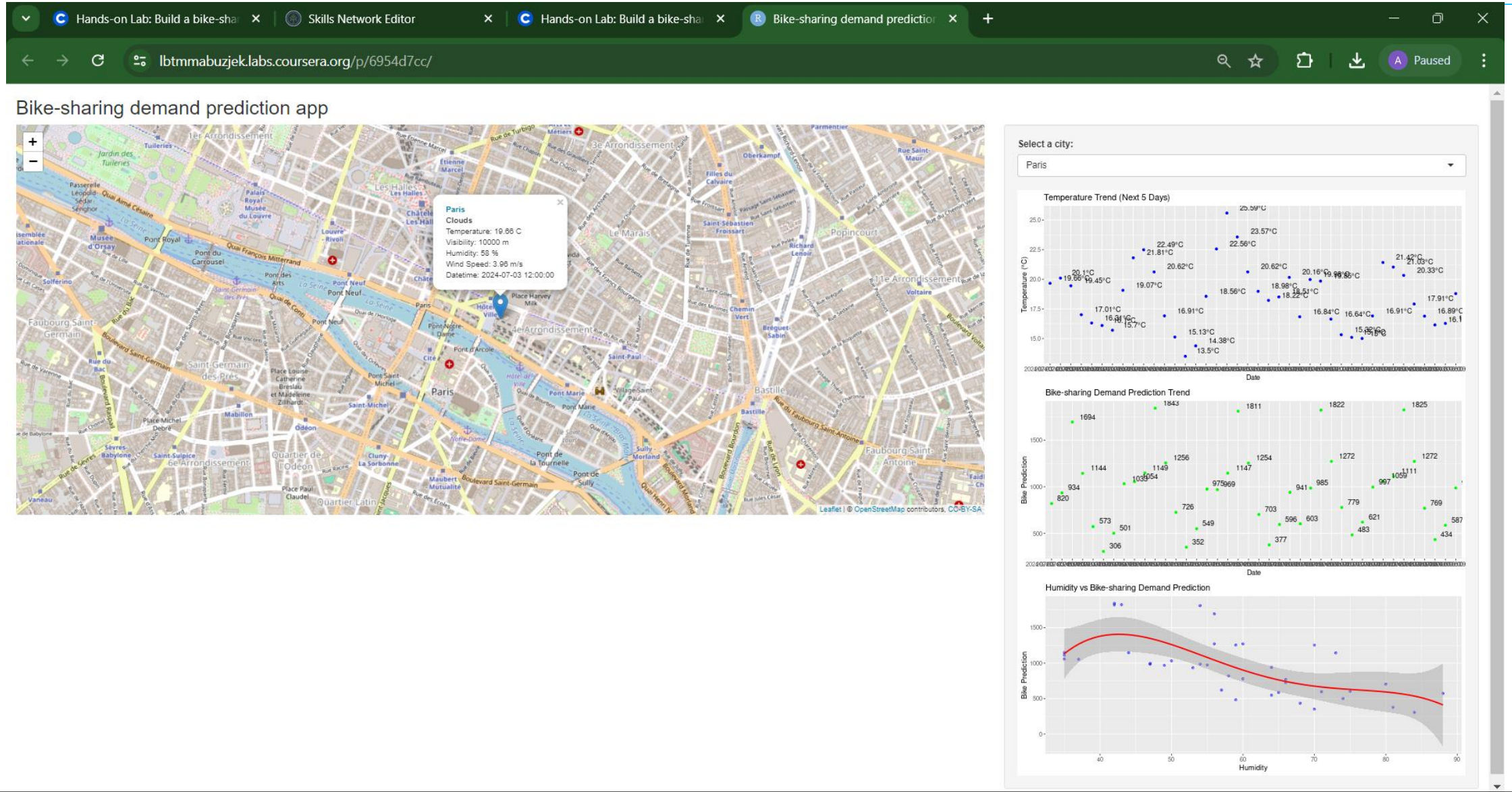
ADDING STATIC TEMPERATURE LINE



ADDING INTERACTIVE BIKE SHARING DEMAND PREDICTION TREND LINE



ADDING A STATIC HUMIDITY AND BIKE SHARING DEMAND PREDICTION CORRELATION PLOT



CONCLUSION

- In conclusion:

- During Summer, bike rental peaks are highest, followed by Autumn, Spring, and Winter, indicating peak demand in Summer and lowest in Winter.
- Peak rental times consistently occur at 8 AM and 6 PM across all seasons.
- Weather factors like TEMPERATURE, HUMIDITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, and RAINFALL significantly influence bike rental counts, especially during high temperatures (red).
- The accuracy of our forecasting models underscores their potential in predicting future usage patterns, particularly in Winter.
- This marks not the end but the beginning of a transformative journey for our bike-sharing system, aiming for efficiency, accessibility, and innovation in the years ahead.

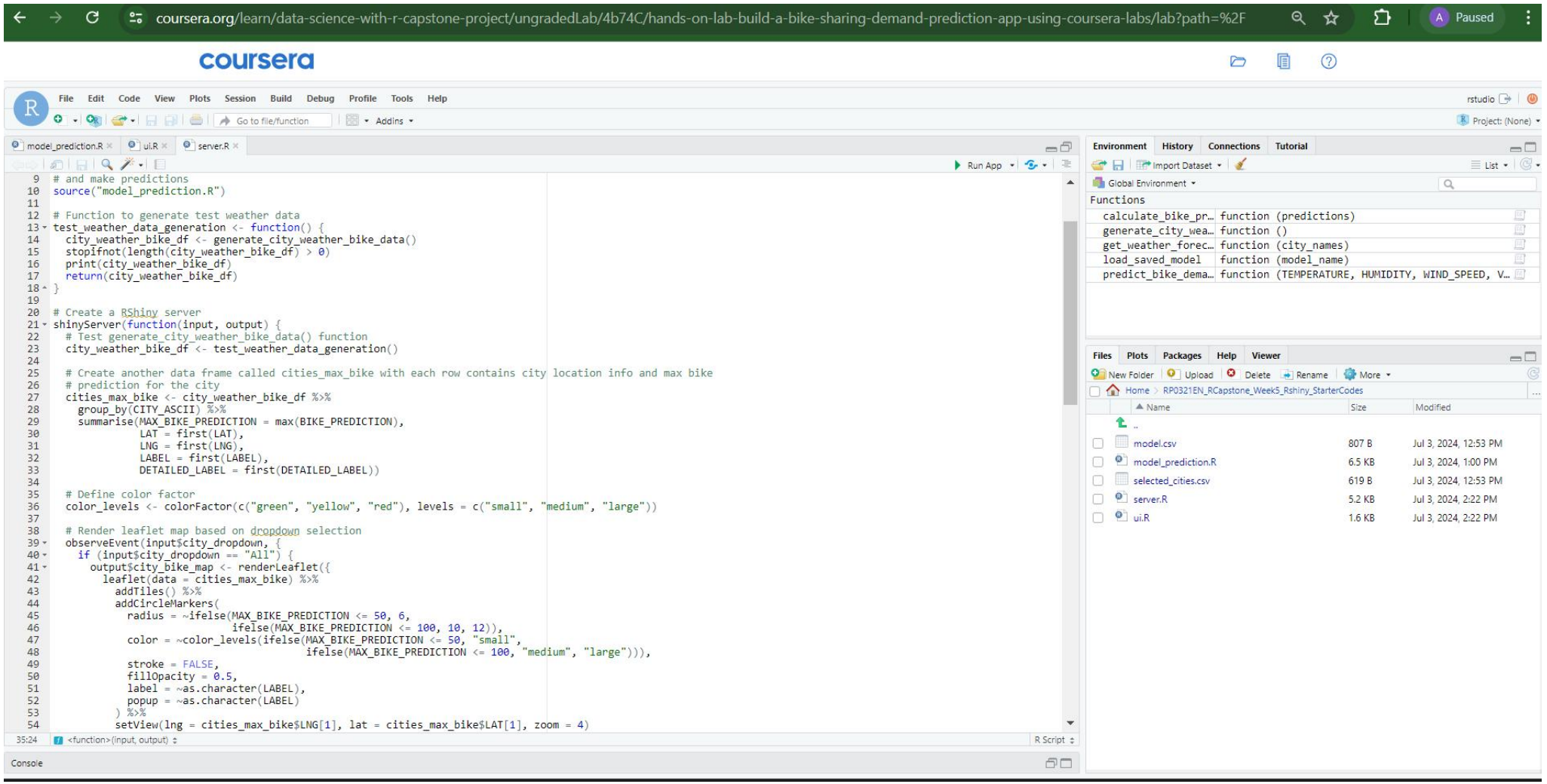
APPENDIX

Links to all Google colab notebooks with output:

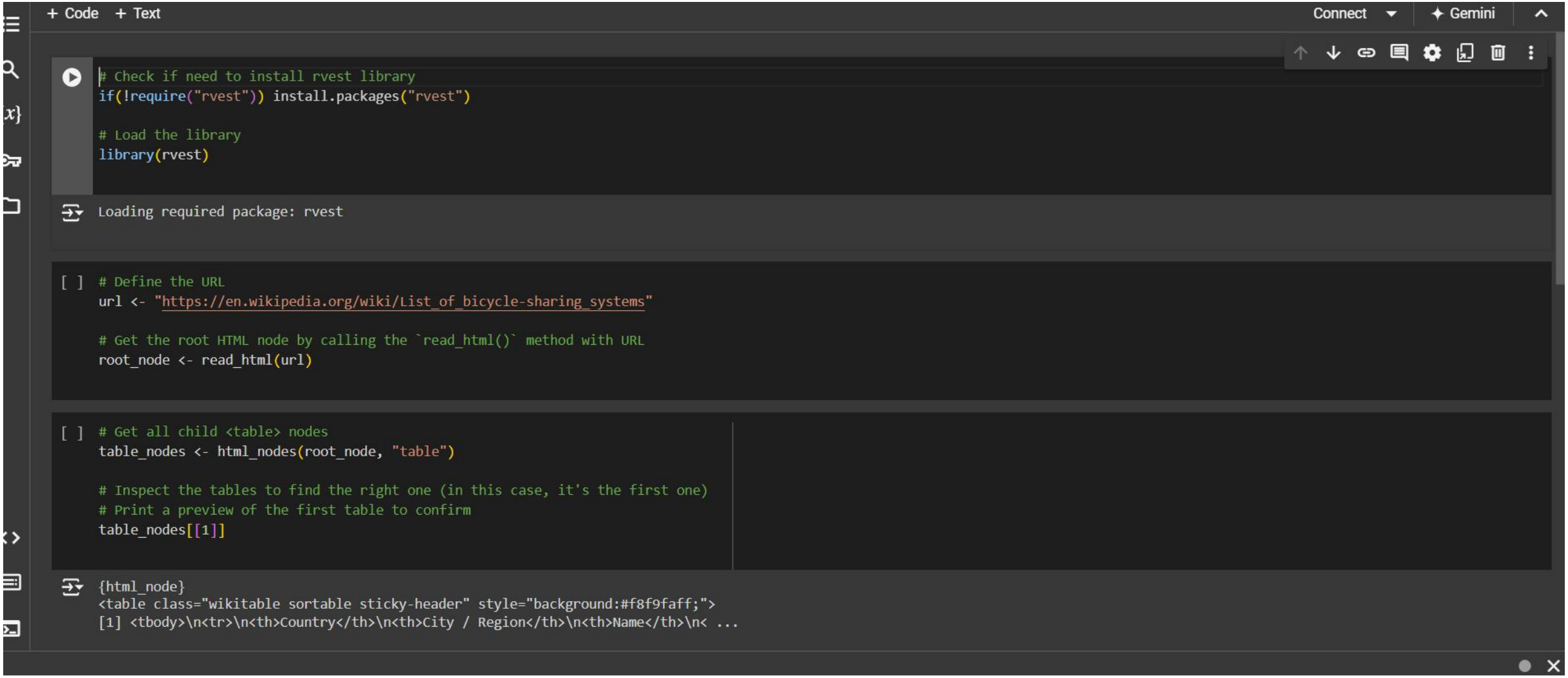
- Module 1-1
- Module 1-2
- Module 2-1
- Module 2-2
- Module 3-1
- Module 3-2
- Module 4

(The modules are linked with the notebooks using Hyperlink so please click the module to get redirected to the colab notebooks)

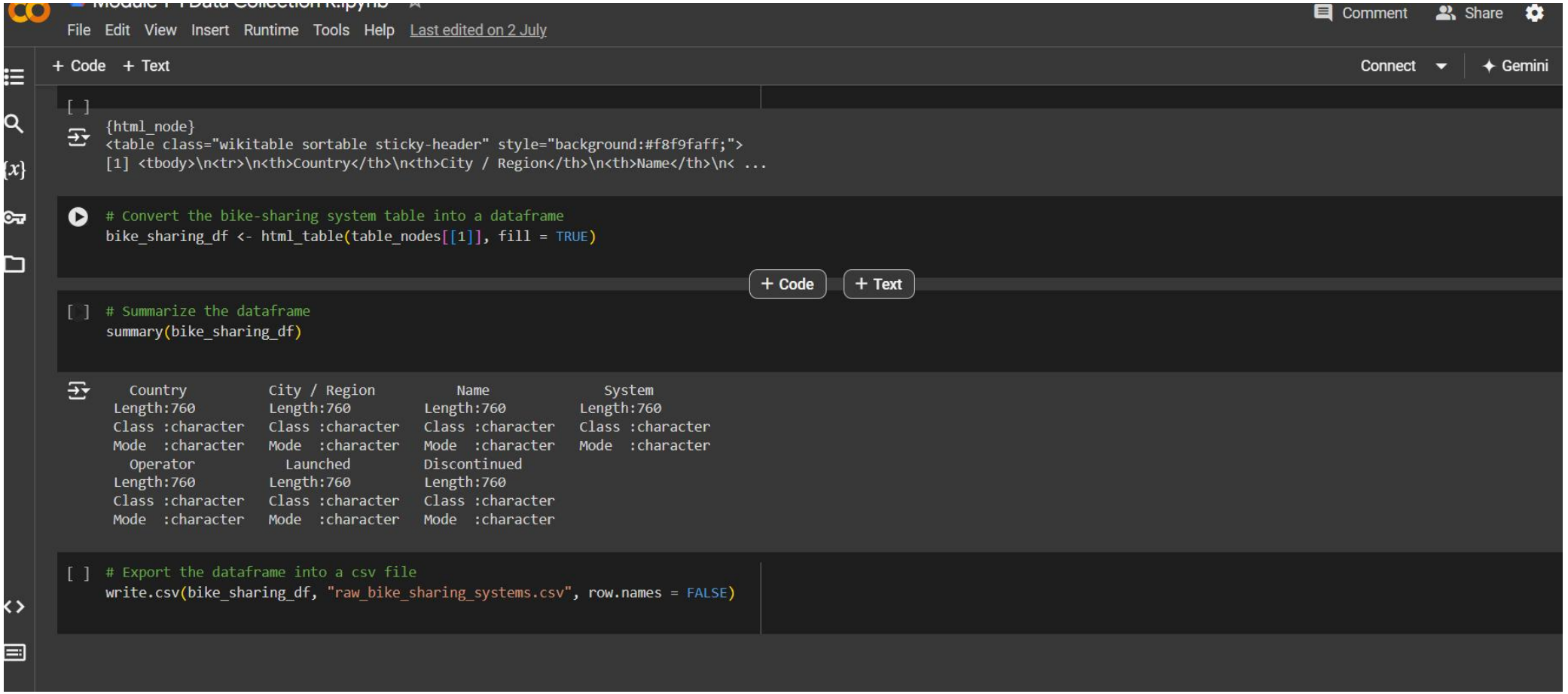
Code Snippets of Dashboard



Code Snippets



Code snippets



Code snippets

DATA COLLECTION

```r
# Install httr package if not already installed
if(!require("httr")) install.packages("httr")

# Load the httr library
library(httr)
```

```r
# Define the URL for Current Weather API
current_weather_url <- 'https://api.openweathermap.org/data/2.5/weather'

# Replace this with your actual API key
your_api_key <- "bfb5094275db52c25ded246628b08ec7"
```

```r
# 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")
```

```r
# Make the HTTP request
response <- GET(current_weather_url, query=current_query)
```

Code snippets

```
[ ] # Make the HTTP request
    response <- GET(current_weather_url, query=current_query)
```

```
    # Check the response type
    http_type(response)
```

```
'application/json'
```

+ Code + Text

```
[ ] # Parse the JSON response
    json_result <- content(response, as="parsed")

    # Check the class of the parsed result
    class(json_result)

    # Print the JSON result
    print(json_result)
```

```
'list'
$coord
$coord$lon
[1] 126.9778

$coord$lat
[1] 37.5683
```

Code snippets

```
. . .
```

```r
# Parse the JSON response
json_result <- content(response, as="parsed")

# Check the class of the parsed result
class(json_result)

# Print the JSON result
print(json_result)
```

```
'list'
$coord
$coord$lon
[1] 126.9778

$coord$lat
[1] 37.5683


$weather
$weather[[1]]
$weather[[1]]$id
[1] 501

$weather[[1]]$main
[1] "Rain"

$weather[[1]]$description
[1] "moderate rain"
```

Code snippets

Module 1-2 Data Collection R.ipynb

File Edit View Insert Runtime Tools Help Last edited on 2 July

+ Code + Text

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

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

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

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

```
  weather visibility  temp temp_min temp_max pressure humidity wind_speed
1    Rain       4000 21.96    21.69    22.66     1003      100       5.14
  wind_deg
1       50
```

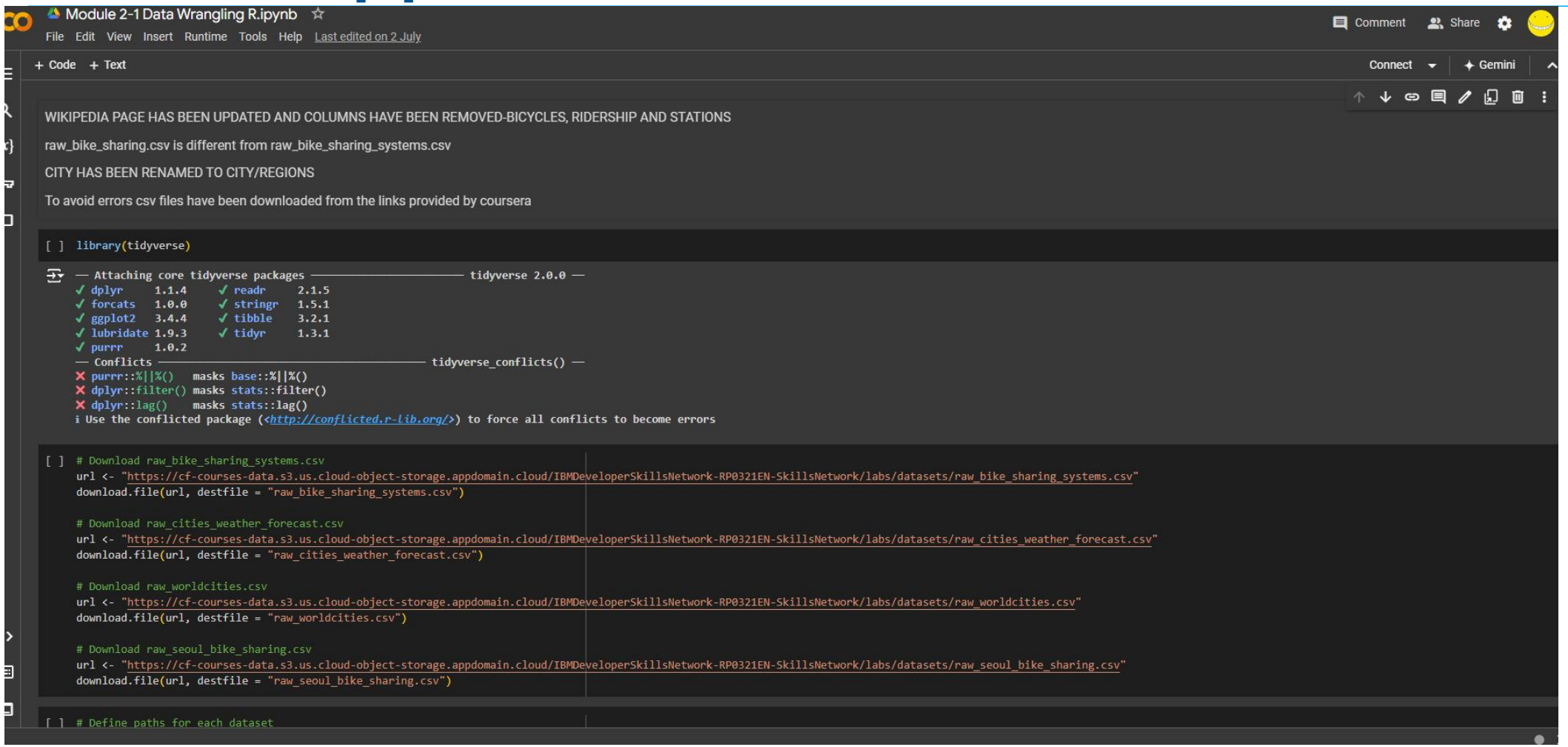
Code snippets

TASK: 5 CITIES

```r
# Install httr package if not already installed
if(!require("httr")) install.packages("httr")

# Load the httr library
library(httr)

# Replace this with your actual API key
your_api_key <- "bfb5094275db52c25ded246628b08ec7"

# Function to get weather forecast by cities
get_weather_forecast_by_cities <- function(city_names) {
    # Initialize an empty data frame to hold all forecast data
    weather_data_frame <- data.frame()

    # Get current month to determine the season
    current_month <- as.numeric(format(Sys.Date(), "%m"))
    if (current_month %in% c(12, 1, 2)) {
        current_season <- "Winter"
    } else if (current_month %in% c(3, 4, 5)) {
        current_season <- "Spring"
    } else if (current_month %in% c(6, 7, 8)) {
        current_season <- "Summer"
    } else {
        current_season <- "Autumn"
    }

    # Loop over each city name
    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(forecast_url, query = forecast_query)

        # Parse the JSON response
        json_result <- content(response, as = "parsed")
        results <- json_result$list

        # Create temporary vectors to hold forecast data for the current city
        city <- c()
        weather <- c()
        visibility <- c()
        temp <- c()
        temp_min <- c()
        temp_max <- c()
```

```r
        # Parse the JSON response
        json_result <- content(response, as = "parsed")
        results <- json_result$list

        # Create temporary vectors to hold forecast data for the current city
        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()
        season <- c()

        # Loop through the JSON result
        for (result in results) {
            city <- c(city, city_name)
            weather <- c(weather, result$weather[[1]]$main)
            visibility <- c(visibility, result$visibility)
            temp <- c(temp, result$main$temp)
            temp_min <- c(temp_min, result$main$temp_min)
            temp_max <- c(temp_max, result$main$temp_max)
            pressure <- c(pressure, result$main$pressure)
            humidity <- c(humidity, result$main$humidity)
            wind_speed <- c(wind_speed, result$wind$speed)
            wind_deg <- c(wind_deg, result$wind$deg)
            forecast_datetime <- c(forecast_datetime, result$dt_txt)
            season <- c(season, current_season)
        }

        # Combine the vectors into a data frame for the current city
        city_weather_data <- 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,
                                        season = season)

        # Append the city's data frame to the main data frame
        weather_data_frame <- rbind(weather_data_frame, city_weather_data)
```

Code snippets



Module 2-1 Data Wrangling R.ipynb

File Edit View Insert Runtime Tools Help Last edited on 2 July

Comment Share Connect Gemini

+ Code + Text

WIKIPEDIA PAGE HAS BEEN UPDATED AND COLUMNS HAVE BEEN REMOVED-BICYCLES, RIDERSHIP AND STATIONS

raw_bike_sharing.csv is different from raw_bike_sharing_systems.csv

CITY HAS BEEN RENAMED TO CITY/REGIONS

To avoid errors csv files have been downloaded from the links provided by coursera

```
[ ] library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.4.4     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
✓ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ purrr::%||%()   masks base::%||%()
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
[ ] # Download raw_bike_sharing_systems.csv
    url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_bike_sharing_systems.csv"
    download.file(url, destfile = "raw_bike_sharing_systems.csv")

    # Download raw_cities_weather_forecast.csv
    url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_cities_weather_forecast.csv"
    download.file(url, destfile = "raw_cities_weather_forecast.csv")

    # Download raw_worldcities.csv
    url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_worldcities.csv"
    download.file(url, destfile = "raw_worldcities.csv")

    # Download raw_seoul_bike_sharing.csv
    url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_seoul_bike_sharing.csv"
    download.file(url, destfile = "raw_seoul_bike_sharing.csv")
```

```
[ ] # Define paths for each dataset
```

Code snippets

```r
# Define paths for each dataset
path_raw_bike_sharing_systems <- "/content/raw_bike_sharing_systems.csv"
path_raw_seoul_bike_sharing <- "/content/raw_seoul_bike_sharing.csv"
path_raw_cities_weather_forecast <- "/content/raw_cities_weather_forecast.csv"
path_raw_worldcities <- "/content/raw_worldcities.csv"

# List of dataset names with paths
dataset_list <- c(path_raw_bike_sharing_systems,
                  path_raw_seoul_bike_sharing,
                  path_raw_cities_weather_forecast,
                  path_raw_worldcities)

# Function to standardize column names
standardize_column_names <- function(dataset_name) {
  # Read dataset using read_csv
  dataset <- read_csv(dataset_name)

  # Standardize column names
  colnames(dataset) <- toupper(gsub(" ", "_", colnames(dataset)))

  # Write back to CSV
  write_csv(dataset, dataset_name, col_names = TRUE)
}

# Apply the function to each dataset
lapply(dataset_list, standardize_column_names)

# Read the resulting datasets back and check column names
for (dataset_name in dataset_list) {
  # Read CSV to check column names
  dataset <- read_csv(dataset_name)

  # Print dataset name and column names
  cat("\nDataset:", dataset_name, "\n")
  print(names(dataset))
}
```

```
Rows: 480 Columns: 10
── Column specification ────────────────────────────────────────
Delimiter: ","
```

Code snippets

```
[ ]  bike_sharing_df <- read_csv("raw_bike_sharing_systems.csv")
```

```
Rows: 480 Columns: 10
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (10): COUNTRY, CITY, NAME, SYSTEM, OPERATOR, LAUNCHED, DISCONTINUED, STA...

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
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

```
[ ]  sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)
```

```
[ ]  sub_bike_sharing_df %>%
       summarize_all(class) %>%
       gather(variable, class)
```

Code snippets

```
[ ]
```

A tibble: 4 × 2

variable	class
<chr>	<chr>
COUNTRY	character
CITY	character
SYSTEM	character
BICYCLES	character

```
[ ] find_character <- function(strings) grepl("[^0-9]", strings)
```

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

A tibble: 10 × 1

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

Code snippets

```
# Apply the remove_ref function to CITY and SYSTEM columns using dplyr::mutate
sub_bike_sharing_df <- sub_bike_sharing_df %>%
  mutate(CITY = remove_ref(CITY),
         SYSTEM = remove_ref(SYSTEM))
```

```
# Check whether all reference links are removed
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
print(result)
```

```
# A tibble: 25 × 3
   CITY                                                  SYSTEM BICYCLES
   <chr>                                                 <chr>  <chr>
 1 Brussels                                              3 Gen… 4115[22]
 2 Limassol (& Agios Dometios, Aglandjia, Dali, Engomi, Latsia,… 3 Gen… 310[59]
 3 Prague                                                NA     500[72]
 4 Prague 7                                              4 Gen… [75]
 5 Prostějov                                             3 Gen… 180[76]
 6 Ostrava                                               3 Gen… 600[77]
 7 Farsø                                                 2 Gen  [78]
 8 Batumi                                                3 Gen… 370[114]
 9 Darmstadt                                             3 & 4… 350 [12…
10 Corfu                                                 3 Gen… 100[58]
# i 15 more rows
```

```
# Define a pattern matching a reference link such as [1]
```

Code snippets

```
[ ]  # Apply the extract_num function to the BICYCLES column using dplyr::mutate
     sub_bike_sharing_df <- sub_bike_sharing_df %>%
       mutate(BICYCLES = extract_num(BICYCLES))
```

```
[ ]  # Check descriptive statistics of the numeric BICYCLES column
     summary(sub_bike_sharing_df$BICYCLES)
```

```
     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
        5     100     343    2012    1400   78000      76
```

```
[ ]  # Write cleaned bike-sharing systems dataset to a CSV file named bike_sharing_systems.csv
     write_csv(sub_bike_sharing_df, "bike_sharing_systems.csv", col_names = TRUE)
```

```
[ ]  Start coding or generate with AI.
```

Code snippets

```r
# Drop rows with missing values in the RENTED_BIKE_COUNT column
bike_sharing_df <- bike_sharing_df %>%
  drop_na(RENTED_BIKE_COUNT)

# Calculate the summer average temperature
summer_avg_temp <- bike_sharing_df %>%
  filter(SEASONS == "Summer") %>%
  summarize(mean_temp = mean(TEMPERATURE, na.rm = TRUE)) %>%
  pull(mean_temp)

# Impute missing values for the TEMPERATURE column using the summer average temperature
bike_sharing_df <- bike_sharing_df %>%
  mutate(TEMPERATURE = ifelse(is.na(TEMPERATURE) & SEASONS == "Summer", summer_avg_temp, TEMPERATURE))

# Print the summary of the dataset again to ensure there are no missing values
summary(bike_sharing_df)

# Save the cleaned dataset
write_csv(bike_sharing_df, "seoul_bike_sharing.csv")
```

```
     Date           RENTED_BIKE_COUNT      Hour        TEMPERATURE
 Length:8465        Min.   :   2.0    Min.   : 0.00   Min.   :-17.80
 Class :character   1st Qu.: 214.0    1st Qu.: 6.00   1st Qu.:  3.00
 Mode  :character   Median : 542.0    Median :12.00   Median : 13.50
                    Mean   : 729.2    Mean   :11.51   Mean   : 12.77
                    3rd Qu.:1084.0    3rd Qu.:18.00   3rd Qu.: 22.70
                    Max.   :3556.0    Max.   :23.00   Max.   : 39.40
    HUMIDITY       WIND_SPEED      Visibility   DEW_POINT_TEMPERATURE
 Min.   : 0.00   Min.   :0.000   Min.   :  27   Min.   :-30.600
 1st Qu.:42.00   1st Qu.:0.900   1st Qu.: 935   1st Qu.: -5.100
 Median :57.00   Median :1.500   Median :1690   Median :  4.700
 Mean   :58.15   Mean   :1.726   Mean   :1434   Mean   :  3.945
 3rd Qu.:74.00   3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 15.200
 Max.   :98.00   Max.   :7.400   Max.   :2000   Max.   : 27.200
 SOLAR_RADIATION     RAINFALL          Snowfall         SEASONS
 Min.   :0.0000   Min.   : 0.0000   Min.   :0.00000   Length:8465
 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.00000   Class :character
 Median :0.0100   Median : 0.0000   Median :0.00000   Mode  :character
 Mean   :0.5679   Mean   : 0.1491   Mean   :0.07769
 3rd Qu.:0.9300   3rd Qu.: 0.0000   3rd Qu.:0.00000
```

Code snippets

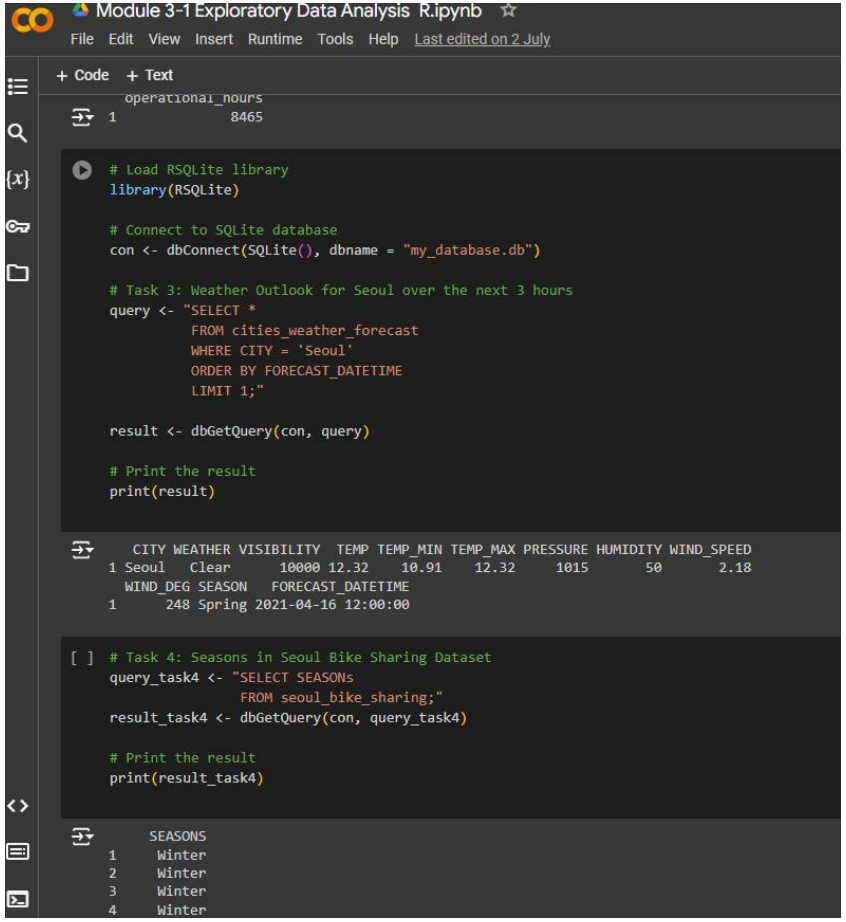
```r
# Min-max normalization function
min_max_norm <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply min-max normalization to the numeric columns
bike_sharing_df <- bike_sharing_df %>%
  mutate_at(vars(RENTED_BIKE_COUNT, TEMPERATURE, HUMIDITY, WIND_SPEED, Visibility,
                 DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, Snowfall), min_max_norm)

# Save the normalized dataset
write_csv(bike_sharing_df, "seoul_bike_sharing_converted_normalized.csv")
```

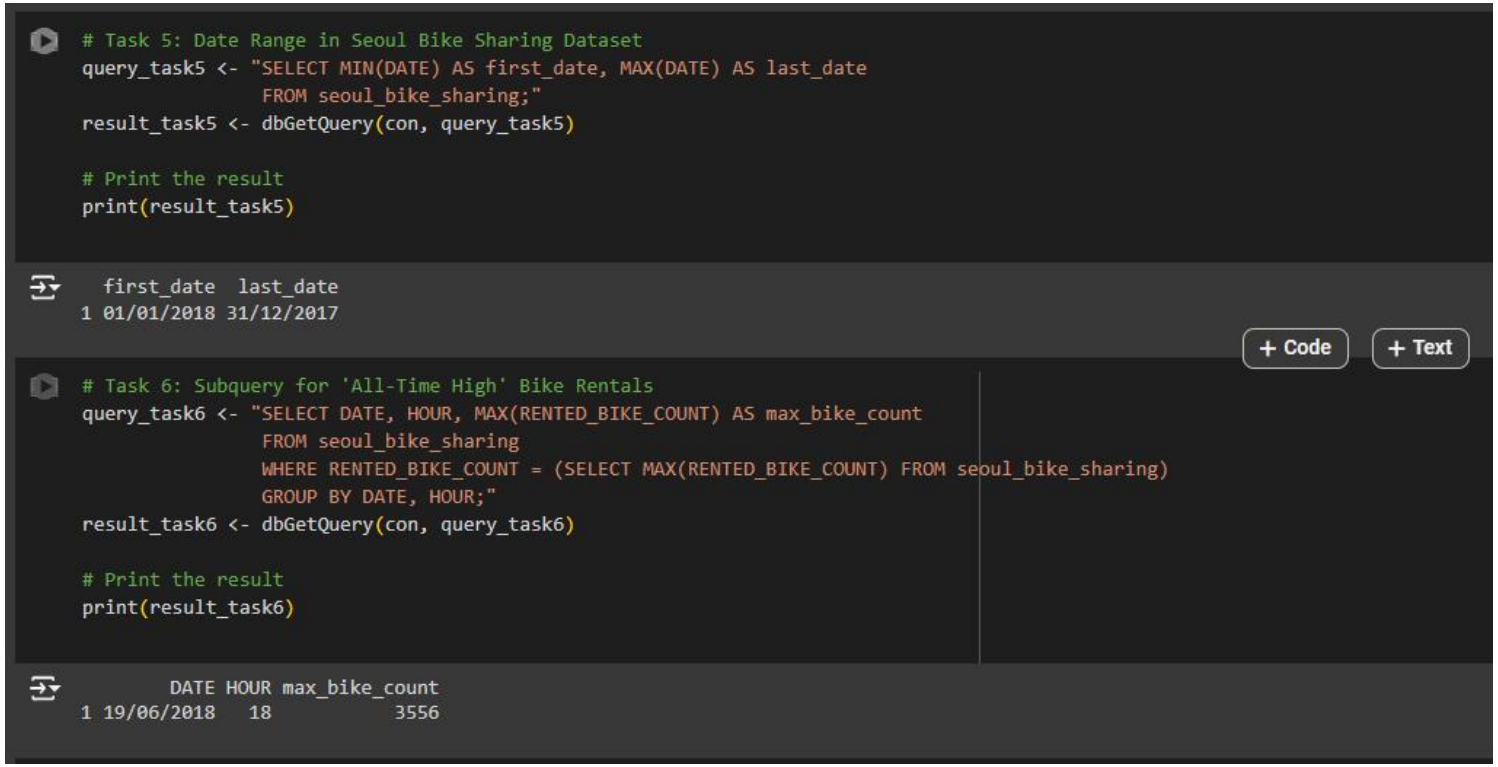
+ Code + Text

```r
# Dataset list
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)
  # Standardize its columns
  names(dataset) <- toupper(names(dataset))
  names(dataset) <- str_replace_all(names(dataset), " ", "_")
  # Save the dataset back
  write_csv(dataset, dataset_name)
}
```

```
Rows: 8465 Columns: 14
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr  (4): Date, SEASONS, HOLIDAY, FUNCTIONING_DAY
dbl (10): RENTED_BIKE_COUNT, Hour, TEMPERATURE, HUMIDITY, WIND_SPEED, Visibi...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 8465 Columns: 41
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
```

Code snippets





Code snippets

```r
# Task 6: Subquery for 'All-Time High' Bike Rentals
query_task6 <- "SELECT DATE, HOUR, MAX(RENTED_BIKE_COUNT) AS max_bike_count
                FROM seoul_bike_sharing
                WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM seoul_bike_sharing)
                GROUP BY DATE, HOUR;"
result_task6 <- dbGetQuery(con, query_task6)

# Print the result
print(result_task6)
```

```
        DATE HOUR max_bike_count
1 19/06/2018   18          3556
```

```r
# Task 7: Hourly Popularity and Temperature by Season
query_task7 <- "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_task7 <- dbGetQuery(con, query_task7)

# Print the result
print(result_task7)
```

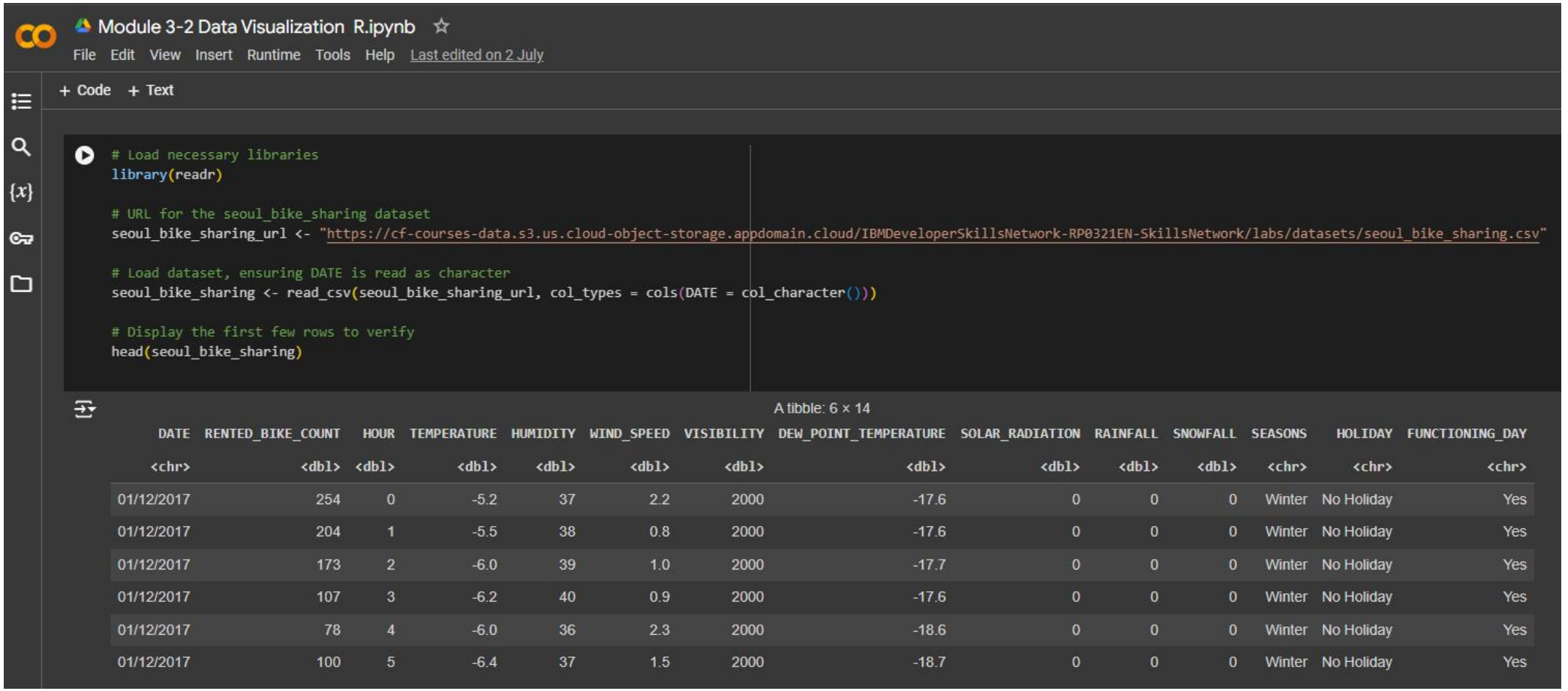
```
   SEASONS HOUR avg_temperature avg_bike_count
1   Summer   18        29.38791       2135.141
2   Autumn   18        16.03185       1983.333
3   Summer   19        28.27378       1889.250
4   Summer   20        27.06630       1801.924
5   Summer   21        26.27826       1754.065
6   Spring   18        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
```

```r
# Task 8: Rental Seasonality
query_task8 <- "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;"
result_task8 <- dbGetQuery(con, query_task8)

# Print the result
print(result_task8)
```

```
  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
```

```r
# Task 9: Weather Seasonality using Seoul Bike Sharing Data Only
query_task9 <- "SELECT SEASONS AS 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,
                       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_task9 <- dbGetQuery(con, query_task9)

# Print the result
print(result_task9)
```

Code snippets



DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	SNOWFALL	SEASONS	HOLIDAY	FUNCTIONING_DAY
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	<chr>
01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0	0	0	Winter	No Holiday	Yes
01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0	0	0	Winter	No Holiday	Yes
01/12/2017	100	5	-6.4	37	1.5	2000	-18.7	0	0	0	Winter	No Holiday	Yes

Code snippets

```r
# Load necessary libraries
library(dplyr)
library(lubridate)

# Assuming seoul_bike_sharing is already loaded from previous task

# Recast DATE column as Date type
seoul_bike_sharing <- seoul_bike_sharing %>%
  mutate(DATE = dmy(DATE))

# Display the structure of the dataframe to verify
str(seoul_bike_sharing)
```

```
tibble [8,465 × 14] (S3: tbl_df/tbl/data.frame)
 $ DATE                 : Date[1:8465], format: "2017-12-01" "2017-12-01" ...
 $ RENTED_BIKE_COUNT    : num [1:8465] 254 204 173 107 78 100 181 460 930 490 ...
 $ HOUR                 : num [1:8465] 0 1 2 3 4 5 6 7 8 9 ...
 $ TEMPERATURE          : num [1:8465] -5.2 -5.5 -6 -6.2 -6 -6.4 -6.6 -7.4 -7.6 -6.5 ...
 $ HUMIDITY             : num [1:8465] 37 38 39 40 36 37 35 38 37 27 ...
 $ WIND_SPEED           : num [1:8465] 2.2 0.8 1 0.9 2.3 1.5 1.3 0.9 1.1 0.5 ...
 $ VISIBILITY           : num [1:8465] 2000 2000 2000 2000 2000 2000 ...
 $ DEW_POINT_TEMPERATURE: num [1:8465] -17.6 -17.6 -17.7 -17.6 -18.6 -18.7 -19.5 -19.3 -19.8 -22.4 ...
 $ SOLAR_RADIATION      : num [1:8465] 0 0 0 0 0 0 0 0 0.01 0.23 ...
 $ RAINFALL             : num [1:8465] 0 0 0 0 0 0 0 0 0 0 ...
 $ SNOWFALL             : num [1:8465] 0 0 0 0 0 0 0 0 0 0 ...
 $ SEASONS              : chr [1:8465] "Winter" "Winter" "Winter" "Winter" ...
 $ HOLIDAY              : chr [1:8465] "No Holiday" "No Holiday" "No Holiday" "No Holiday" ...
 $ FUNCTIONING_DAY      : chr [1:8465] "Yes" "Yes" "Yes" "Yes" ...
```

Code snippets

```r
# Convert HOUR to a factor with ordered levels
seoul_bike_sharing <- seoul_bike_sharing %>%
  mutate(HOUR = factor(HOUR, levels = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10",
                                        "11", "12", "13", "14", "15", "16", "17", "18", "19", "20",
                                        "21", "22", "23"), ordered = TRUE))

# Display the structure of the dataframe to verify
str(seoul_bike_sharing)
```

```
tibble [8,465 × 14] (S3: tbl_df/tbl/data.frame)
 $ DATE                 : Date[1:8465], format: "2017-12-01" "2017-12-01" ...
 $ RENTED_BIKE_COUNT    : num [1:8465] 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 [1:8465] -5.2 -5.5 -6 -6.2 -6 -6.4 -6.6 -7.4 -7.6 -6.5 ...
 $ HUMIDITY             : num [1:8465] 37 38 39 40 36 37 35 38 37 27 ...
 $ WIND_SPEED           : num [1:8465] 2.2 0.8 1 0.9 2.3 1.5 1.3 0.9 1.1 0.5 ...
 $ VISIBILITY           : num [1:8465] 2000 2000 2000 2000 2000 ...
 $ DEW_POINT_TEMPERATURE: num [1:8465] -17.6 -17.6 -17.7 -17.6 -18.6 -18.7 -19.5 -19.3 -19.8 -22.4 ...
 $ SOLAR_RADIATION      : num [1:8465] 0 0 0 0 0 0 0 0 0.01 0.23 ...
 $ RAINFALL             : num [1:8465] 0 0 0 0 0 0 0 0 0 0 ...
 $ SNOWFALL             : num [1:8465] 0 0 0 0 0 0 0 0 0 0 ...
 $ SEASONS              : chr [1:8465] "Winter" "Winter" "Winter" "Winter" ...
 $ HOLIDAY              : chr [1:8465] "No Holiday" "No Holiday" "No Holiday" "No Holiday" ...
 $ FUNCTIONING_DAY      : chr [1:8465] "Yes" "Yes" "Yes" "Yes" ...
```

Code snippets

```
# Check for missing values
sum(is.na(seoul_bike_sharing))
```

```
0
```

```
# Task 4: Dataset Summary
summary(seoul_bike_sharing)
```

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

 SOLAR_RADIATION     RAINFALL          SNOWFALL          SEASONS
 Min.   :0.0000   Min.   : 0.0000   Min.   :0.00000   Length:8465
 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.00000   Class :character
 Median :0.0100   Median : 0.0000   Median :0.00000   Mode  :character
 Mean   :0.5679   Mean   : 0.1491   Mean   :0.07769
 3rd Qu.:0.9300   3rd Qu.: 0.0000   3rd Qu.:0.00000
 Max.   :3.5200   Max.   :35.0000   Max.   :8.80000

   HOLIDAY          FUNCTIONING_DAY
 Length:8465        Length:8465
 Class :character   Class :character
 Mode  :character   Mode  :character
```

```
# Task 5: Calculate number of Holidays
num_holidays <- sum(seoul_bike_sharing$HOLIDAY == "Holiday")
num_holidays
```

```
408
```

```
# Task 6: Calculate percentage of records on Holiday
percentage_holidays <- mean(seoul_bike_sharing$HOLIDAY == "Holiday") * 100
percentage_holidays
```

```
4.8198464264619
```

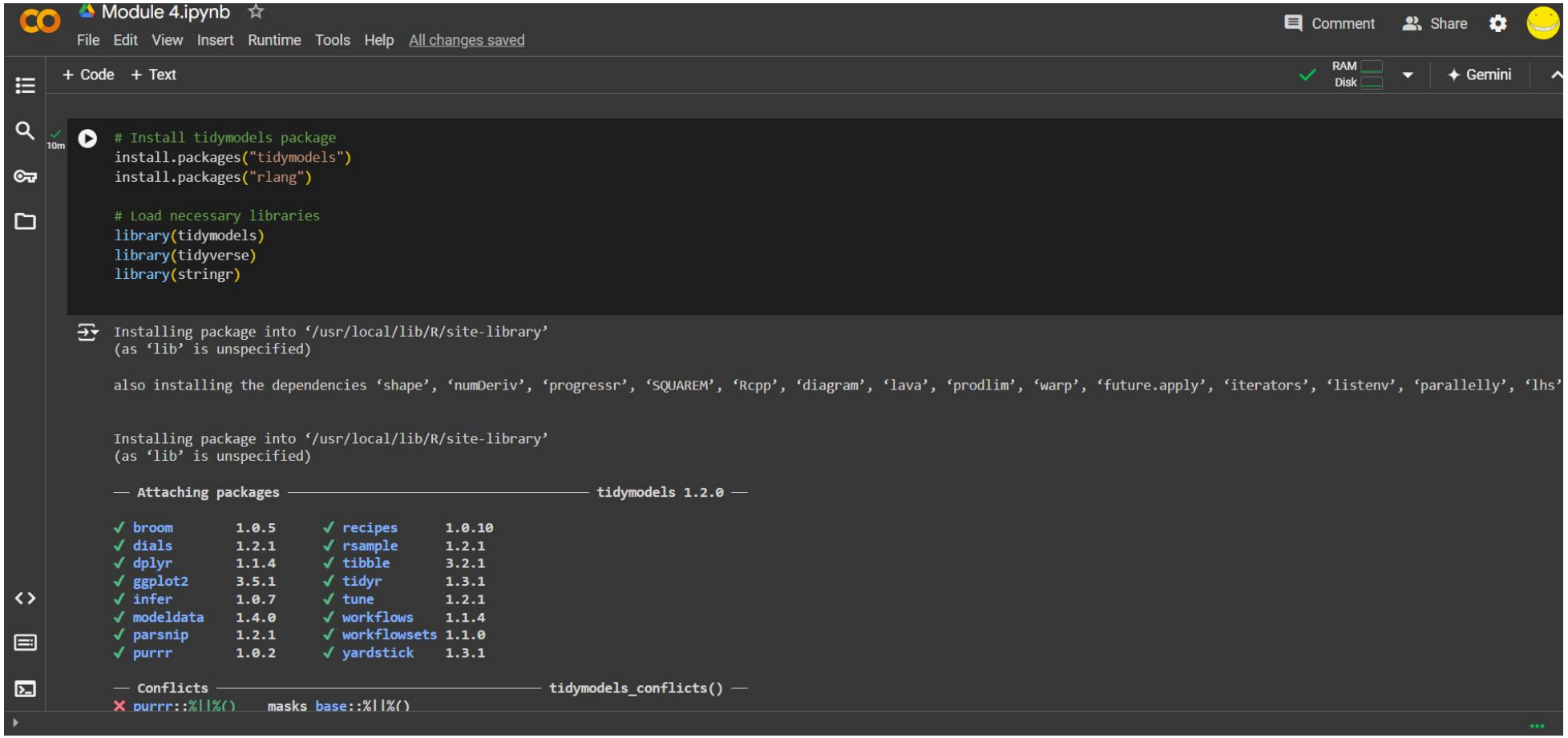
```
# Task 7: Calculate expected number of records
expected_records <- 365
expected_records
```

```
365
```

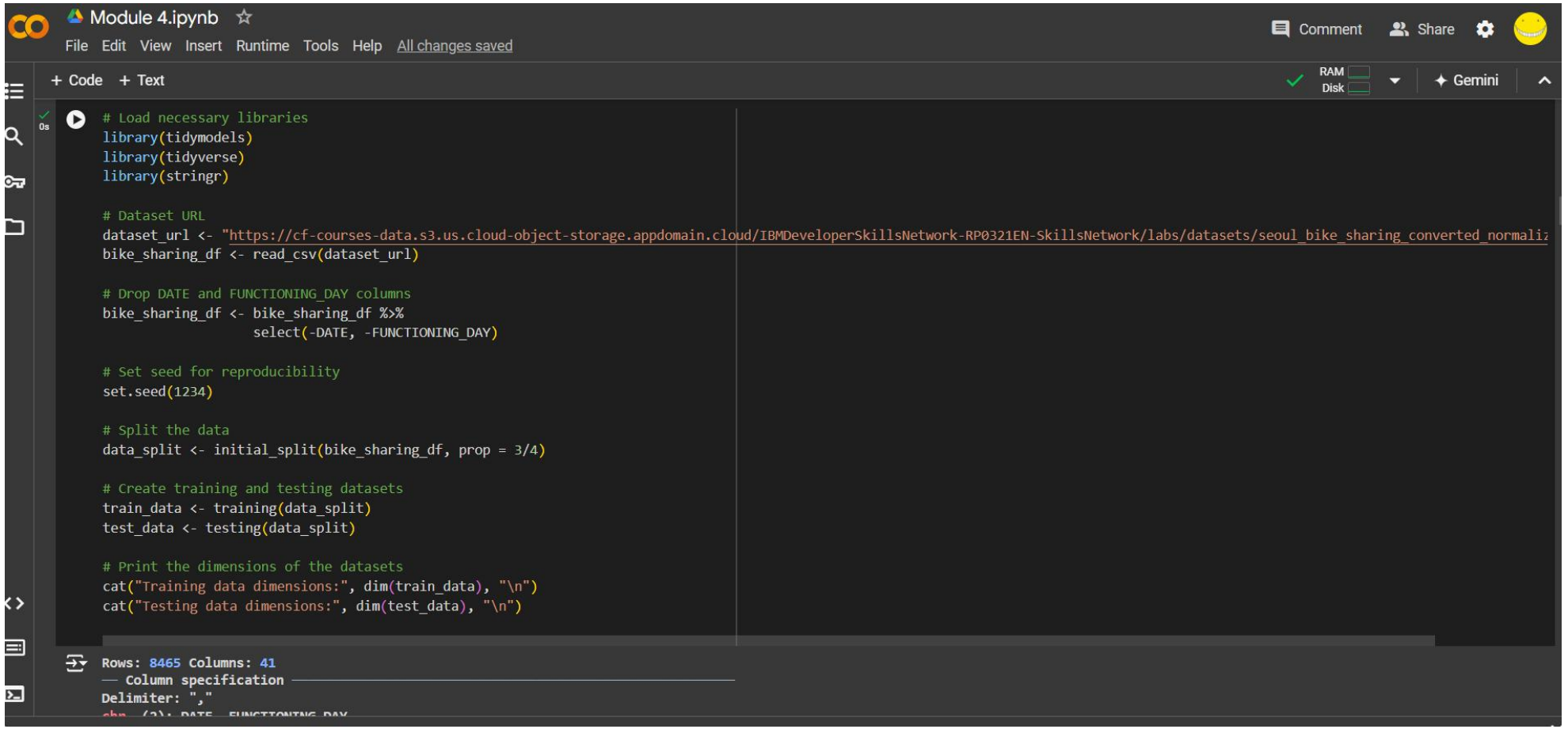
```
# Task 8: Calculate number of records with FUNCTIONING_DAY == 'Yes'
num_functioning_days <- sum(seoul_bike_sharing$FUNCTIONING_DAY == "Yes")
num_functioning_days
```

```
8465
```

Code snippets



Code snippets



Code snippets

```r
# Load necessary libraries
library(tidymodels)
library(tidyverse)
library(stringr)

# Dataset URL
dataset_url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/seoul_bike_sharing_converted_normalized.csv"
bike_sharing_df <- read_csv(dataset_url)

# Drop DATE and FUNCTIONING_DAY columns
bike_sharing_df <- bike_sharing_df %>%
                   select(-DATE, -FUNCTIONING_DAY)

# Set seed for reproducibility
set.seed(1234)

# Split the data
data_split <- initial_split(bike_sharing_df, prop = 3/4)

# Create training and testing datasets
train_data <- training(data_split)
test_data <- testing(data_split)

# Define the linear regression model specification
lm_spec <- linear_reg() %>%
  set_engine("lm") %>%
  set_mode("regression")

# Fit the linear regression model using weather variables only
lm_model_weather <- lm_spec %>%
  fit(RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL + SNOWFALL,
      data = train_data)

# Print the fit summary
summary(lm_model_weather$fit)
```

```
Rows: 8465 Columns: 41
── Column specification ──────────────────────────────────────────────────────
Delimiter: ","
chr  (2): DATE, FUNCTIONING_DAY
dbl (39): RENTED_BIKE_COUNT, TEMPERATURE, HUMIDITY, WIND_SPEED, V

i Use `spec()` to retrieve the full column specification for this
i Specify the column types or set `show_col_types = FALSE` to qui

Call:
stats::lm(formula = RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY +
    WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADI
    RAINFALL + SNOWFALL, data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-1348.46  -294.03   -57.28   208.59  2329.78

Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)             156.71      58.07   2.699  0.00698 **
TEMPERATURE            2399.74     261.66   9.171  < 2e-16 ***
HUMIDITY               -918.38     126.79  -7.243  4.9e-13 ***
WIND_SPEED              404.47      48.16   8.399  < 2e-16 ***
VISIBILITY               12.56      24.86   0.505  0.61351
DEW_POINT_TEMPERATURE  -316.92     278.83  -1.137  0.25575
SOLAR_RADIATION        -444.85      34.69 -12.824  < 2e-16 ***
RAINFALL              -1764.01     182.65  -9.658  < 2e-16 ***
SNOWFALL                317.78     131.58   2.415  0.01576 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 487.3 on 6339 degrees of freedom
Multiple R-squared:  0.4303,    Adjusted R-squared:  0.4296
F-statistic: 598.5 on 8 and 6339 DF,  p-value: < 2.2e-16
```

Code snippets

```r
# Fit the linear regression model using all variables
lm_model_all <- lm_spec %>%
  fit(RENTED_BIKE_COUNT ~ .,
      data = train_data)

# Print the fit summary
summary(lm_model_all$fit)
```

```
Call:
stats::lm(formula = RENTED_BIKE_COUNT ~ ., data = data)

Residuals:
     Min       1Q   Median       3Q      Max
-1401.45  -218.96    -7.31   199.53  1780.67

Coefficients: (3 not defined because of singularities)
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)            212.20      51.04   4.158 3.26e-05 ***
TEMPERATURE            782.66     212.13   3.690 0.000227 ***
HUMIDITY              -886.73      99.49  -8.913  < 2e-16 ***
WIND_SPEED              31.91      40.27   0.792 0.428169
VISIBILITY              21.87      20.26   1.079 0.280439
DEW_POINT_TEMPERATURE  598.39     221.37   2.703 0.006888 **
SOLAR_RADIATION        276.88      41.47   6.677 2.64e-11 ***
RAINFALL             -2064.64     143.28 -14.410  < 2e-16 ***
SNOWFALL               260.97     103.50   2.522 0.011709 *
`0`                    -29.30      34.26  -0.855 0.392515
`1`                   -116.85      33.72  -3.465 0.000533 ***
`10`                  -237.52      32.74  -7.255 4.48e-13 ***
`11`                  -247.38      33.85  -7.309 3.02e-13 ***
`12`                  -208.34      34.39  -6.059 1.45e-09 ***
`13`                  -191.35      35.04  -5.461 4.90e-08 ***
`14`                  -192.44      34.44  -5.588 2.39e-08 ***
`15`                  -109.73      34.40  -3.190 0.001429 **
`16`                    23.13      34.00   0.680 0.496431
`17`                   305.55      34.15   8.946  < 2e-16 ***
`18`                   794.80      34.02  23.364  < 2e-16 ***
`19`                   522.99      34.25  15.268  < 2e-16 ***
`2`                   -237.21      33.74  -7.030 2.28e-12 ***
`20`                   432.00      34.13  12.657  < 2e-16 ***
```

```r
# Calculate R-squared and RMSE for lm_model_weather
rsq_weather <- rsq(test_results_weather, truth = truth, estimate = pred)
rmse_weather <- rmse(test_results_weather, truth = truth, estimate = pred)

# Calculate R-squared and RMSE for lm_model_all
rsq_all <- rsq(test_results_all, truth = truth, estimate = pred)
rmse_all <- rmse(test_results_all, truth = truth, estimate = pred)

# Print the results
rsq_weather
rmse_weather
rsq_all
rmse_all
```

```r
# Print coefficients of lm_model_all
coefficients_all <- lm_model_all$fit$coefficients

# Create a data frame of coefficients for visualization
coefficients_df <- as.data.frame(coefficients_all) %>%
  rownames_to_column(var = "variable") %>%
  rename(coefficient = coefficients_all) %>%
  arrange(desc(abs(coefficient)))

# Visualize the coefficients using ggplot
library(ggplot2)
ggplot(coefficients_df, aes(x = reorder(variable, abs(coefficient)), y = coefficient)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  labs(title = "Coefficients of lm_model_all",
       x = "Variable",
       y = "Coefficient") +
  theme_minimal()
```

```
Warning message:
```

Code snippets

Code snippets

Code snippets

Code snippets

Code snippets

model_prediction.R | ui.R | server.R

```r
# Install and import required libraries
require(shiny)
require(ggplot2)
require(leaflet)
require(tidyverse)
require(httr)
require(scales)
# Import model_prediction R which contains methods to call OpenWeather API
# and make predictions
source("model_prediction.R")

# Function to generate test weather data
test_weather_data_generation <- function() {
  city_weather_bike_df <- generate_city_weather_bike_data()
  stopifnot(length(city_weather_bike_df) > 0)
  print(city_weather_bike_df)
  return(city_weather_bike_df)
}

# Create a RShiny server
shinyServer(function(input, output) {
  # Test generate_city_weather_bike_data() function
  city_weather_bike_df <- test_weather_data_generation()

  # Create another data frame called cities_max_bike with each row contains city location info and max bike
  # prediction for the city
  cities_max_bike <- city_weather_bike_df %>%
    group_by(CITY_ASCII) %>%
    summarise(MAX_BIKE_PREDICTION = max(BIKE_PREDICTION),
              LAT = first(LAT),
              LNG = first(LNG),
              LABEL = first(LABEL),
              DETAILED_LABEL = first(DETAILED_LABEL))

  # Define color factor
  color_levels <- colorFactor(c("green", "yellow", "red"), levels = c("small", "medium", "large"))

  # Render leaflet map based on dropdown selection
  observeEvent(input$city_dropdown, {
    if (input$city_dropdown == "All") {
      output$city_bike_map <- renderLeaflet({
        leaflet(data = cities_max_bike) %>%
          addTiles() %>%
          addCircleMarkers(
            radius = ~ifelse(MAX_BIKE_PREDICTION <= 50, 6,
                             ifelse(MAX_BIKE_PREDICTION <= 100, 10, 12)),
```

35:24 <function>(input, output)

model_prediction.R | ui.R | server.R

```r
                             ifelse(MAX_BIKE_PREDICTION <= 100, 10, 12)),
            color = ~color_levels(ifelse(MAX_BIKE_PREDICTION <= 50, "small",
                                         ifelse(MAX_BIKE_PREDICTION <= 100, "medium", "large"))),
            stroke = FALSE,
            fillOpacity = 0.5,
            label = ~as.character(LABEL),
            popup = ~as.character(LABEL)
          ) %>%
          setView(lng = cities_max_bike$LNG[1], lat = cities_max_bike$LAT[1], zoom = 4)
      })
    } else {
      selected_city <- cities_max_bike %>% filter(CITY_ASCII == input$city_dropdown)
      output$city_bike_map <- renderLeaflet({
        leaflet(data = selected_city) %>%
          addTiles() %>%
          addMarkers(lng = ~LNG, lat = ~LAT,
                     popup = ~as.character(DETAILED_LABEL))
      })
    }

    # Task 1: Render static temperature trend line
    output$temp_line <- renderPlot({
      if (input$city_dropdown != "All") {
        selected_city_data <- city_weather_bike_df %>%
          filter(CITY_ASCII == input$city_dropdown) %>%
          arrange(FORECASTDATETIME)  # Ensure data is sorted by FORECASTDATETIME

        ggplot(selected_city_data, aes(x = FORECASTDATETIME, y = TEMPERATURE)) +
          geom_line(color = "blue") +
          geom_point(color = "blue") +
          geom_text(aes(label = paste0(TEMPERATURE, "°C")), vjust = -0.5, hjust = -0.5) +
          labs(title = "Temperature Trend (Next 5 Days)",
               x = "Date", y = "Temperature (°C)")
      }
    })

    # Task 2: Render interactive bike-sharing demand prediction trend line
    output$bike_line <- renderPlot({
      if (input$city_dropdown != "All") {
        selected_city_data <- city_weather_bike_df %>%
          filter(CITY_ASCII == input$city_dropdown) %>%
          arrange(FORECASTDATETIME)  # Ensure data is sorted by FORECASTDATETIME

        p <- ggplot(selected_city_data, aes(x = FORECASTDATETIME, y = BIKE_PREDICTION)) +
          geom_line(color = "green") +
          geom_point(color = "green") +
```

35:24 <function>(input, output)

Console

Code snippets

model_prediction.R | ui.R | server.R

Run App

```
        arrange(FORECASTDATETIME)  # Ensure data is sorted by FORECASTDATETIME

      p <- ggplot(selected_city_data, aes(x = FORECASTDATETIME, y = BIKE_PREDICTION)) +
        geom_line(color = "green") +
        geom_point(color = "green") +
        geom_text(aes(label = paste0(BIKE_PREDICTION)), vjust = -0.5, hjust = -0.5) +
        labs(title = "Bike-sharing Demand Prediction Trend",
             x = "Date", y = "Bike Prediction")

      if (!is.null(input$plot_click)) {
        click <- input$plot_click
        near_point <- nearest_data(selected_city_data, click)
        p <- p +
          geom_text(data = near_point, aes(label = paste0("Demand: ", BIKE_PREDICTION, "\nDate: ", FORECASTDATETIME)),
                    color = "black", vjust = -1, hjust = -0.5, size = 4, parse = TRUE)
      }

      p
    }
  })

  # Task 3: Render static humidity and bike-sharing demand prediction correlation plot
  output$humidity_pred_chart <- renderPlot({
    if (input$city_dropdown != "All") {
      selected_city_data <- city_weather_bike_df %>%
        filter(CITY_ASCII == input$city_dropdown)

      ggplot(selected_city_data, aes(x = HUMIDITY, y = BIKE_PREDICTION)) +
        geom_point(color = "blue", alpha = 0.5) +
        geom_smooth(method = "lm", formula = y ~ poly(x, 4), color = "red") +
        labs(title = "Humidity vs Bike-sharing Demand Prediction",
             x = "Humidity", y = "Bike Prediction")
    }
  })

  # Render text output for clicked point details
  output$bike_date_output <- renderText({
    if (!is.null(input$plot_click)) {
      click <- input$plot_click
      near_point <- nearest_data(selected_city_data, click)
      paste("Clicked Demand:", near_point$BIKE_PREDICTION, "\nClicked Date:", near_point$FORECASTDATETIME)
    }
  })
})
})

132
```

35:24 <function>(input, output) R Script

Console

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function | Addins

model_prediction.R | ui.R | server.R

Run App

```r
# Load required libraries
require(leaflet)

# Create a RShiny UI
shinyUI(
  fluidPage(padding = 5,
            titlePanel("Bike-sharing demand prediction app"),
            # Create a side-bar layout
            sidebarLayout(
              # Create a main panel to show cities on a leaflet map
              mainPanel(
                leafletOutput("city_bike_map", height = 600)  # Adjusted height for better display
              ),
              # Create a side bar to show detailed plots for a city
              sidebarPanel(
                # select drop down list to select city (Task 2)
                selectInput("city_dropdown", "Select a city:",
                            choices = c("All", "New York", "Paris", "Suzhou", "London"),
                            selected = "All"),
                # Plot output for temperature trend (Task 1)
                plotOutput("temp_line", height = 300),  # Added plot output for temperature trend
                # Plot output for bike-sharing demand prediction trend (Task 2)
                plotOutput("bike_line", height = 300, click = "plot_click"),  # Added plot output for bike-sharing prediction trend
                # Text output for clicked point details
                verbatimTextOutput("bike_date_output"),  # Added verbatimTextOutput for clicked point details
                # Task 3: Plot output for humidity vs bike-sharing demand prediction correlation
                plotOutput("humidity_pred_chart", height = 300)  # Added plot output for humidity vs bike-sharing prediction correlation
              )
            )
  )
)
```