

Applied Data Science with R Capstone project

Aarav Kalkar
03/07/2024

Outline



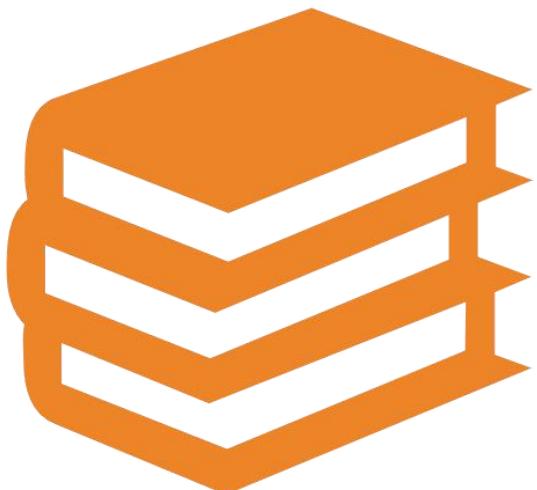
- Executive Summary
- Introduction
- Methodology
- Results
- Conclusion
- Appendix

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

The screenshot shows the OpenWeather API keys management interface. At the top, there's a navigation bar with links for Guide, API, Dashboard, Marketplace, Pricing, Maps, Our Initiatives, Partners, Blog, For Business, and user account information. Below the navigation is a sub-navigation bar with links for New Products, Services, API keys (which is underlined), Billing plans, Payments, Block logs, My orders, My profile, and Ask a question.

A central message states: "You can generate as many API keys as needed for your subscription. We accumulate the total load from all of them."

Key	Name	Status	Actions
fbf5094275db52c25ded246628b08ec7	Default	Active	
63643c8265bbf9fec17d7b4720783b15	aaaa	Active	

To the right, there's a "Create key" section with a text input field labeled "API key name" and a "Generate" button.

At the bottom, there are three footer sections: "Product Collections" (Current and Forecast APIs), "Subscription" (How to start), and "Company" (OpenWeather is a team of IT experts and data scientists that has been).

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:

- Identified 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.

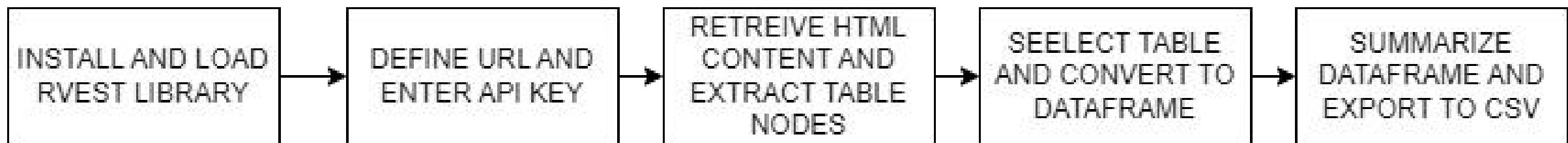
3. Data Integration and Validation:

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

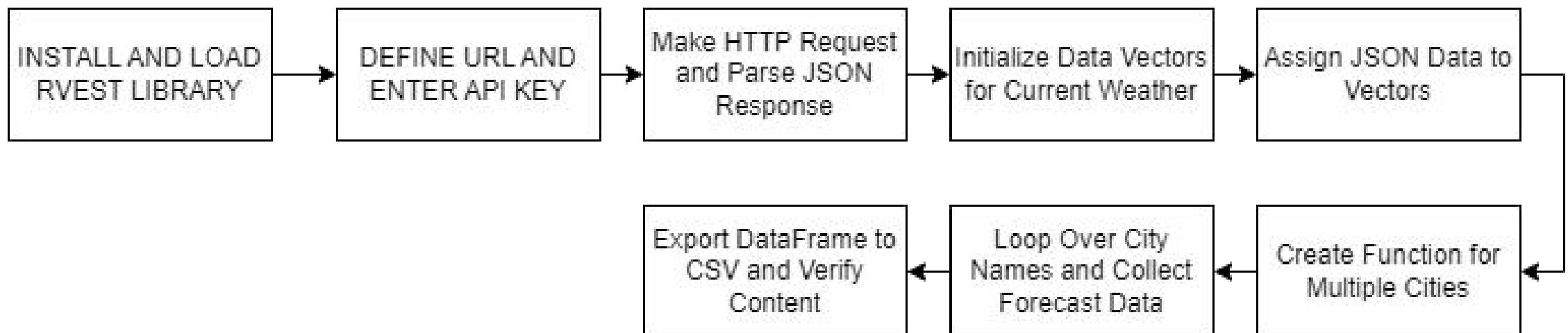
Verified data consistency and accuracy through validation checks to mitigate errors.

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

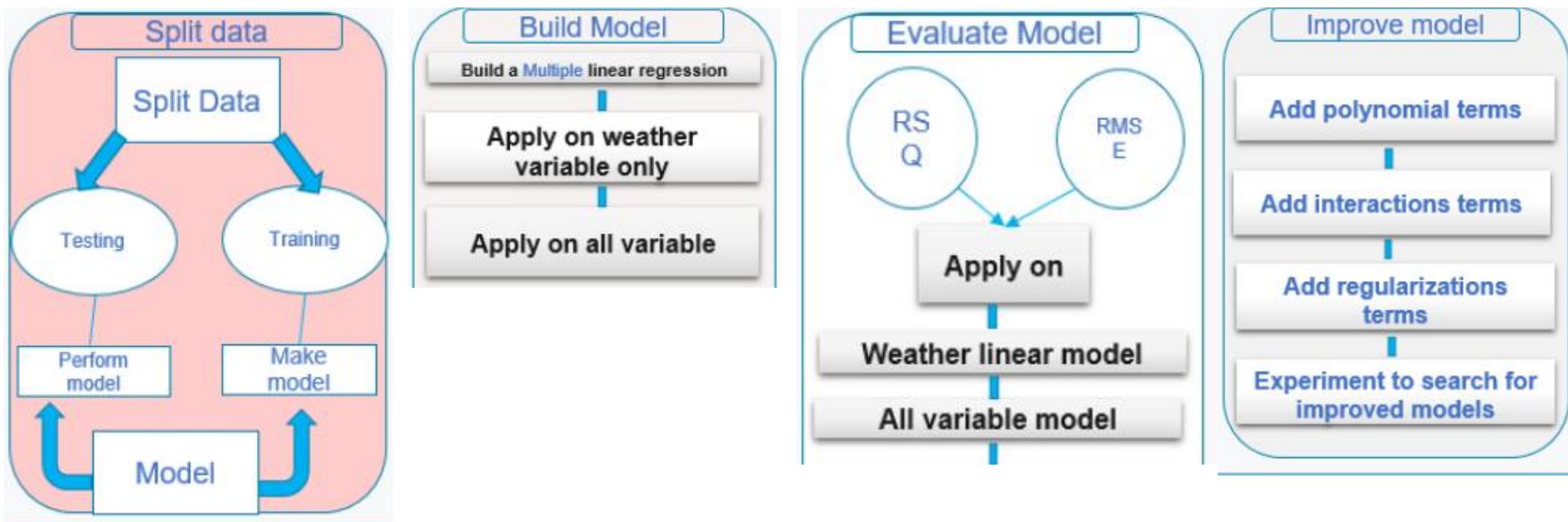
EDA with data visualization

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

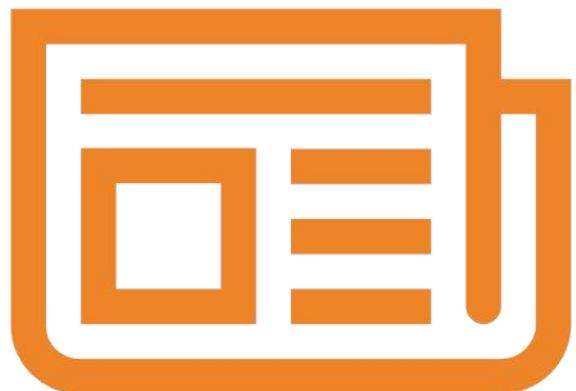


Build a R Shiny dashboard

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

```
# 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

```
# 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

```
▶ # 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.

```
# 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	avg_dew_point	avg_solar_radiation	avg_rainfall	avg_snowfall	avg_bike_count
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					
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

```
# 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

```
▶ # 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

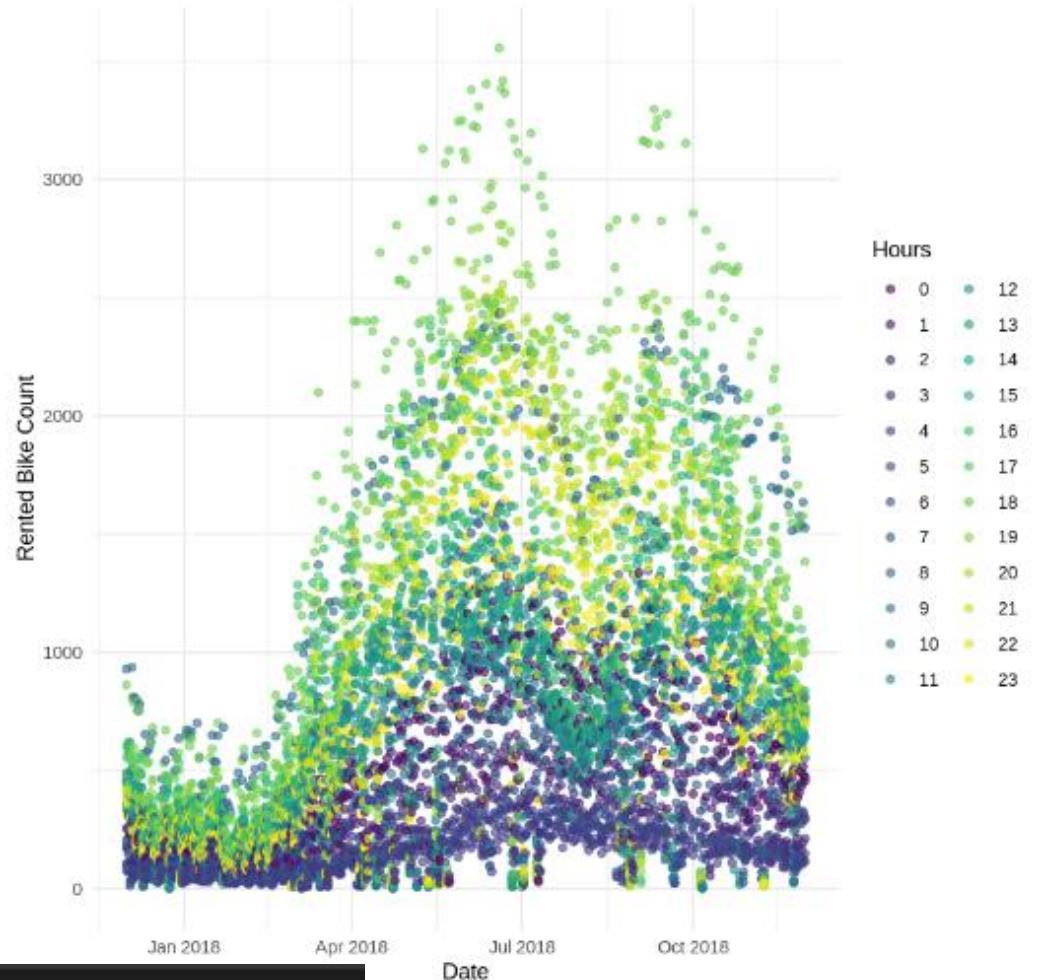
Bike rental vs. Date

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.



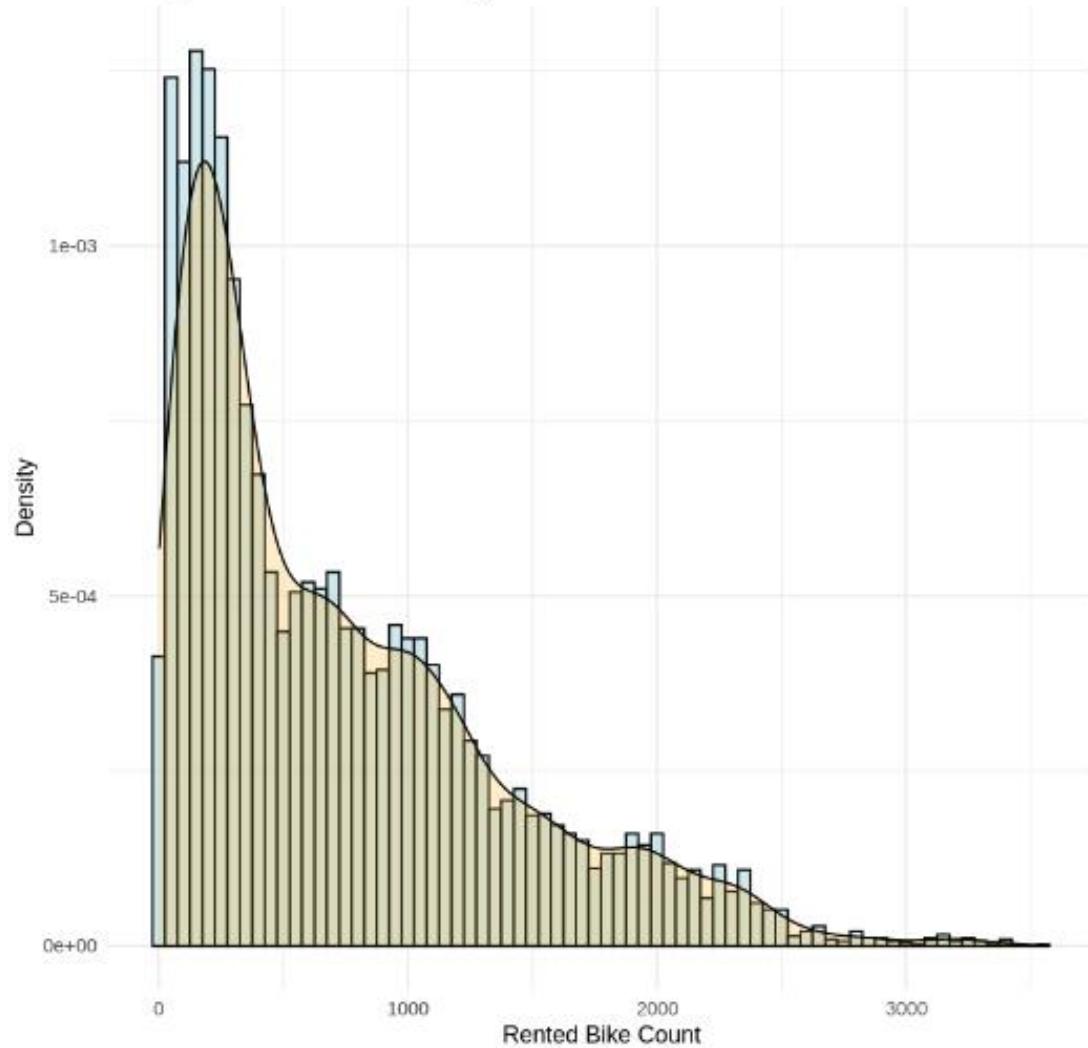
```
# Load ggplot2 package
library(ggplot2)

# 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.

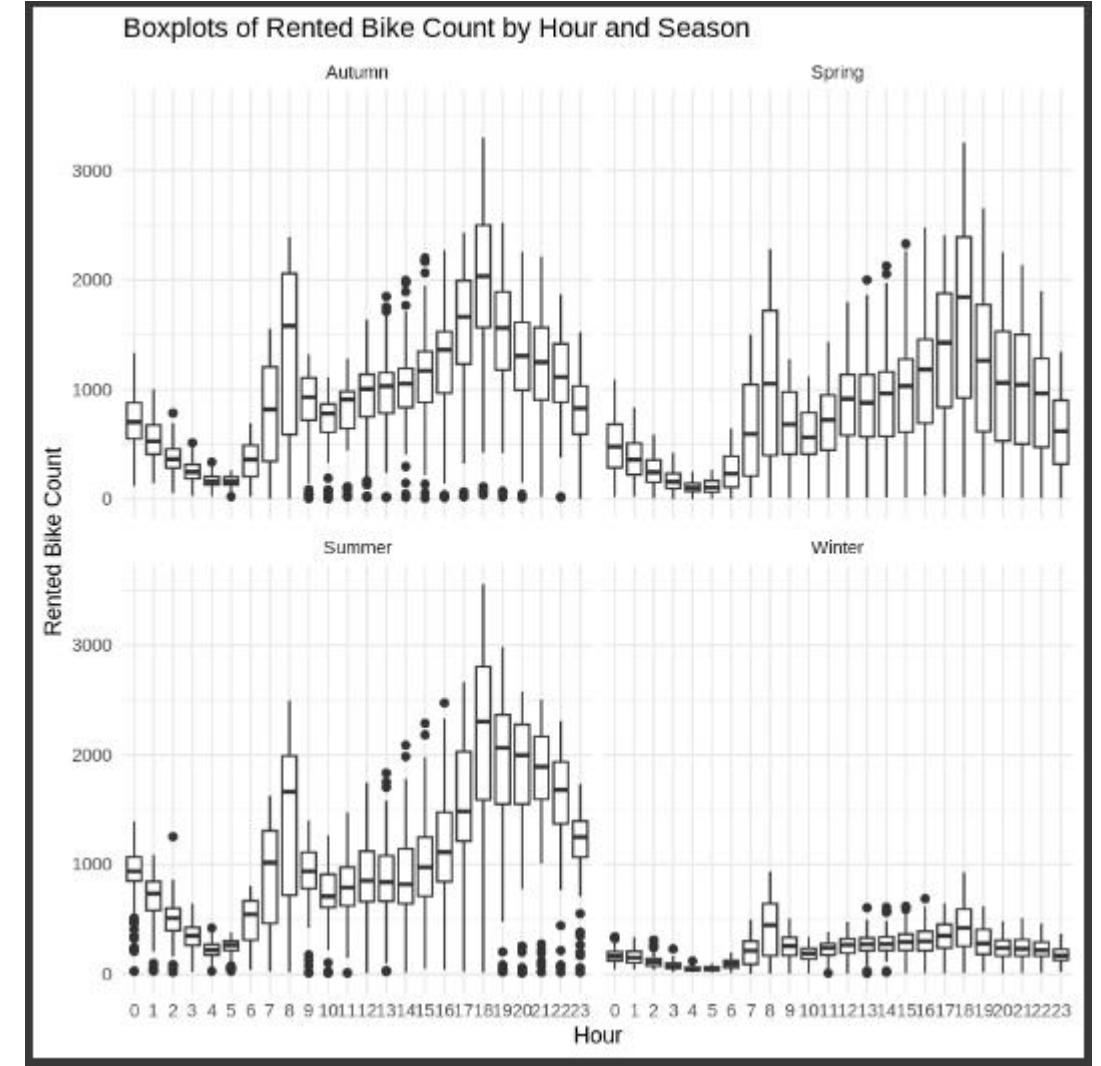
Histogram with Kernel Density Curve of Rented Bike Count



```
# 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

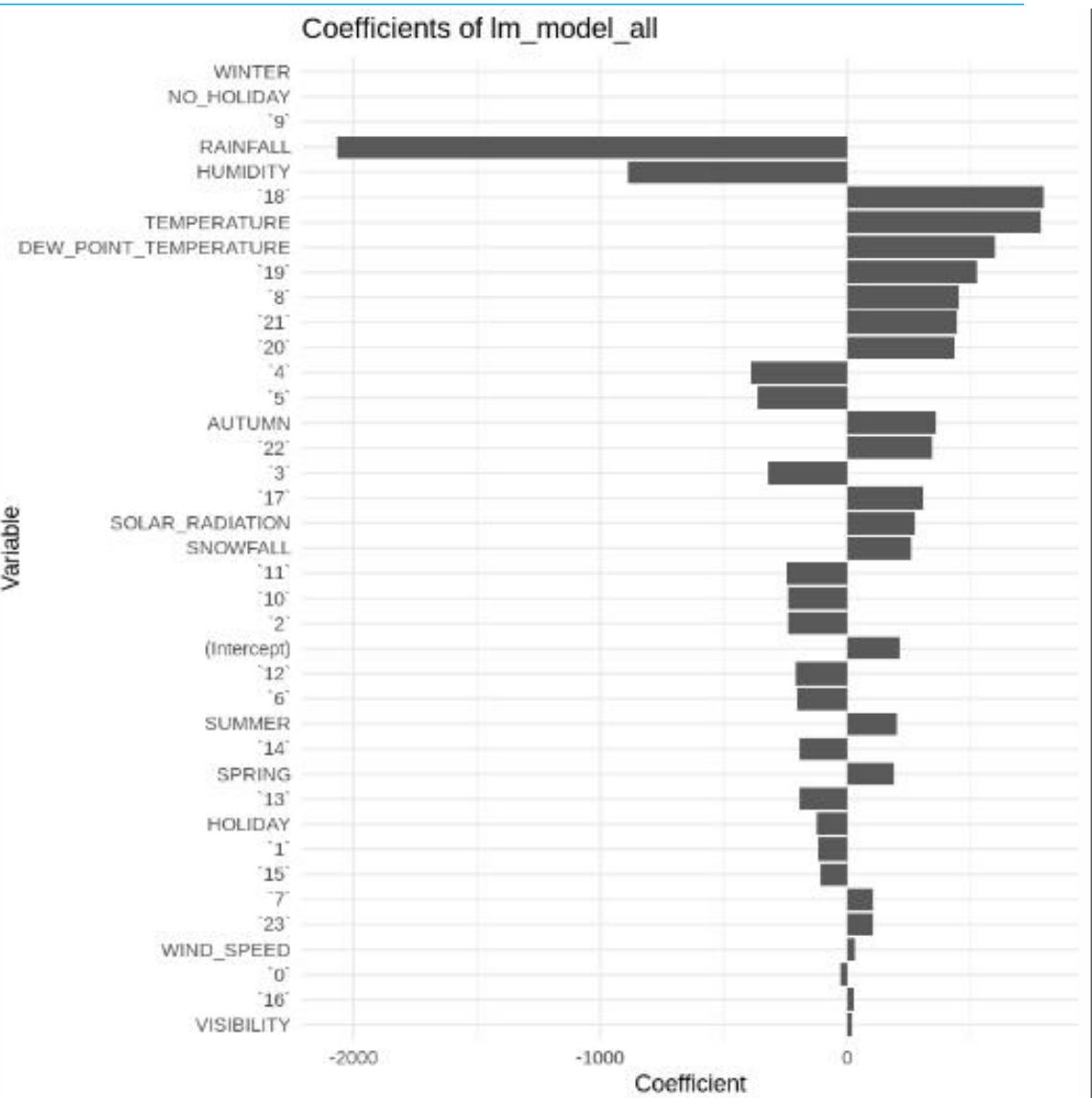
```
# Create boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS
ggplot(seoul_bike_sharing, aes(x = factor(HOUR), y = RENTED_BIKE_COUNT)) +
  geom_boxplot() +
  facet_wrap(~ SEASONS) +
  labs(
    title = "Boxplots of Rented Bike Count by Hour and Season",
    x = "Hour",
    y = "Rented Bike Count"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



Predictive analysis

Ranked coefficients

```
# 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

```
# 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)

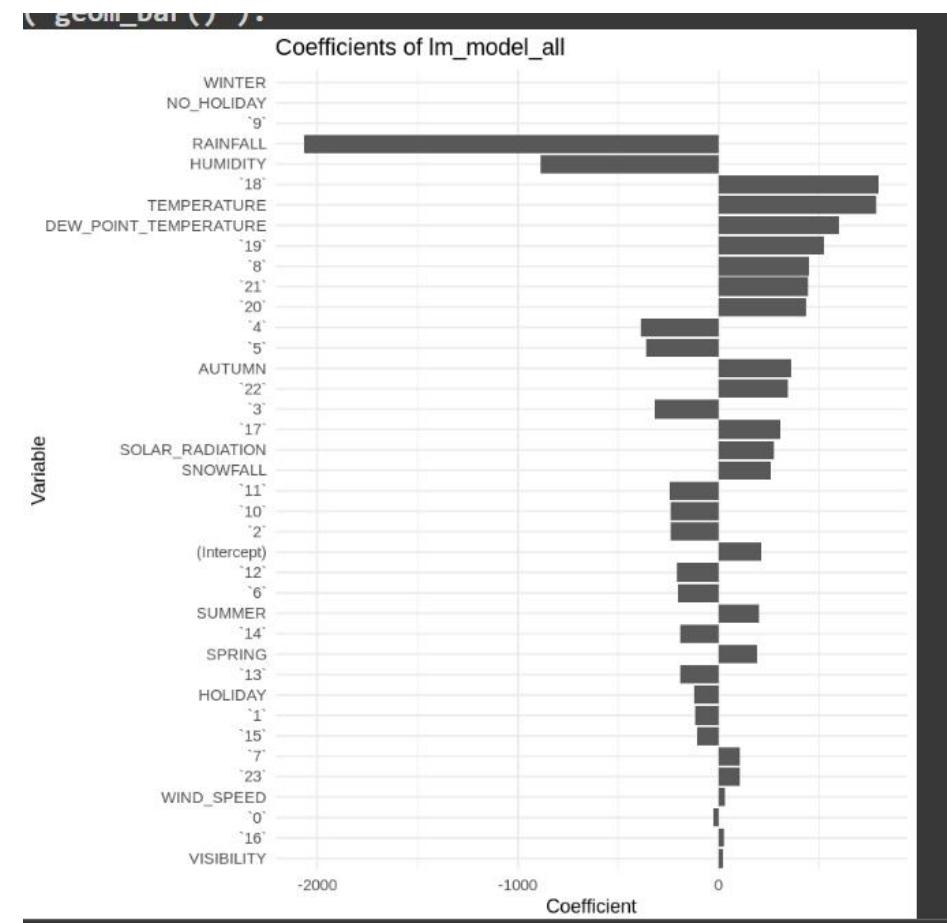
# Print the results
rsq_weather
rmse_weather
rsq_all
rmse_all
```

- 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).

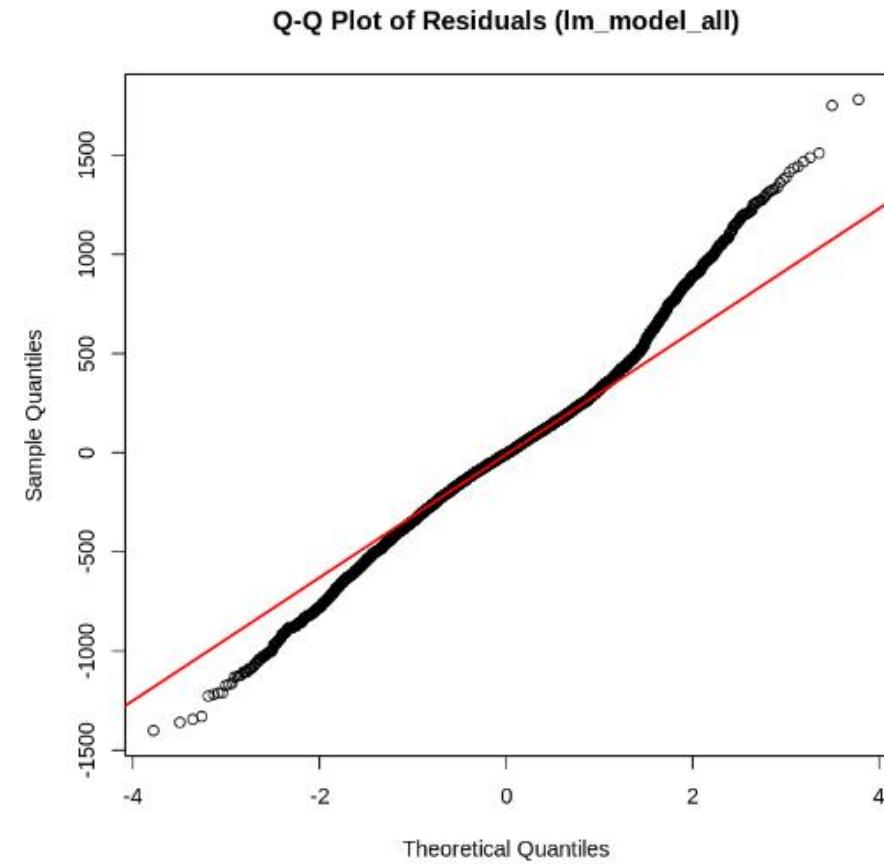
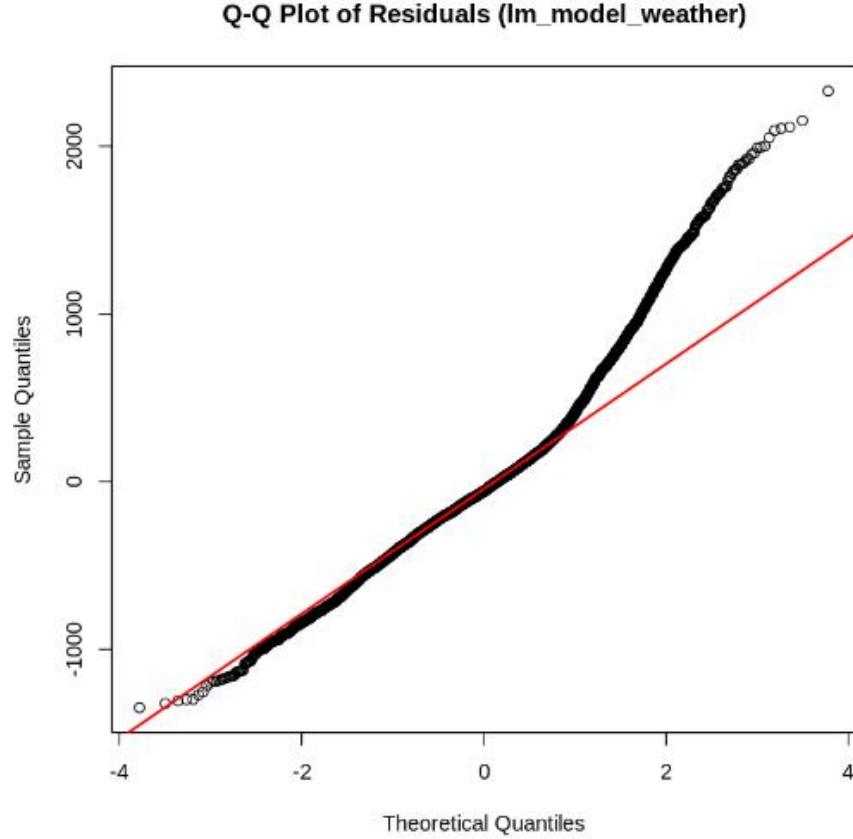
.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.438866
A tibble: 1 × 3		
.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	474.6247
A tibble: 1 × 3		
.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.6690204
A tibble: 1 × 3		
.metric	.estimator	.estimate
<chr>	<chr>	<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



code for Q-Q plot

```
✓ 1s ⏎ # 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

Bike-sharing demand prediction app

New York
Clear

Select a city:
New York

36

ADDING DROP DOWN MENU

Bike-sharing demand prediction app

Select a city:

New York

37

ADDING STATIC TEMPERATURE LINE

Bike-sharing demand prediction app

The screenshot shows a web application interface. At the top, there are three tabs: "Hands-on Lab: Build a bike-share" (active), "Skills Network Editor", and "Bike-sharing demand prediction". The URL in the address bar is "lbmmabuzjek.labs.coursera.org/p/5ccfc8cc/". Below the tabs, there are standard browser controls: back, forward, search, and refresh.

The main content area features a world map with a blue marker indicating New York. A tooltip for New York displays the following information:

- New York
- Clear
- Temperature: 19.04 C
- Visibility: 10000 m
- Humidity: 75 %
- Wind Speed: 2.25 m/s
- Datetime: 2024-07-03 09:00:00

Below the map is a copyright notice: "Leaflet | © OpenStreetMap contributors, CC-BY-SA".

To the right of the map is a modal window titled "Select a city:" with a dropdown menu set to "New York". The window contains a scatter plot titled "Temperature Trend (Next 5 Days)". The x-axis is labeled "Date" and shows a timeline from July 2, 2024, to July 6, 2024. The y-axis is labeled "Temperature (°C)" and ranges from 20 to 32. Blue dots represent daily temperature measurements, with specific values labeled for each point:

Date	Temperature (°C)
2024-07-02	19.04°C
2024-07-03	20.03°C
2024-07-04	22.47°C
2024-07-05	22.81°C
2024-07-06	21.81°C
2024-07-07	21.2°C
2024-07-08	23.41°C
2024-07-09	23.64°C
2024-07-10	25.48°C
2024-07-11	26.64°C
2024-07-12	26.99°C
2024-07-13	26.72°C
2024-07-14	29.1°C
2024-07-15	24.63°C
2024-07-16	24.03°C
2024-07-17	25.33°C
2024-07-18	25°C
2024-07-19	26.27°C
2024-07-20	26.07°C
2024-07-21	26.6°C
2024-07-22	28.28°C
2024-07-23	29.36°C
2024-07-24	29.2°C
2024-07-25	31.38°C

At the bottom of the screen is a Windows taskbar with various icons: Start, Search, File Explorer, Google Chrome, Spotify, and others. The system tray shows the date and time as "03-07-2024 13:40".

ADDING INTERACTIVE BIKE SHARING DEMAND PREDICTION TREND LINE

Bike-sharing demand prediction app

The screenshot shows a web browser window with four tabs open. The active tab is titled "Bike-sharing demand prediction". The main content area displays a world map with a highlighted location for New York. A tooltip provides weather information for New York: Clear, Temperature: 19.04°C, Visibility: 10000 m, Humidity: 75 %, Wind Speed: 2.25 m/s, and Datetime: 2024-07-03 09:00:00. To the right of the map is a temperature trend chart titled "Temperature Trend (Next 5 Days)". The chart plots temperature in degrees Celsius against date from July 2024 to August 2024. Data points are labeled with their respective temperatures. The chart includes a legend for "New York" and a dropdown menu for selecting other cities.

Hands-on Lab: Build a bike-sha x | Skills Network Editor x | Hands-on Lab: Build a bike-sha x | Bike-sharing demand prediction x +

lbmmabuzjek.labs.coursera.org/p/5ccfc8cc/

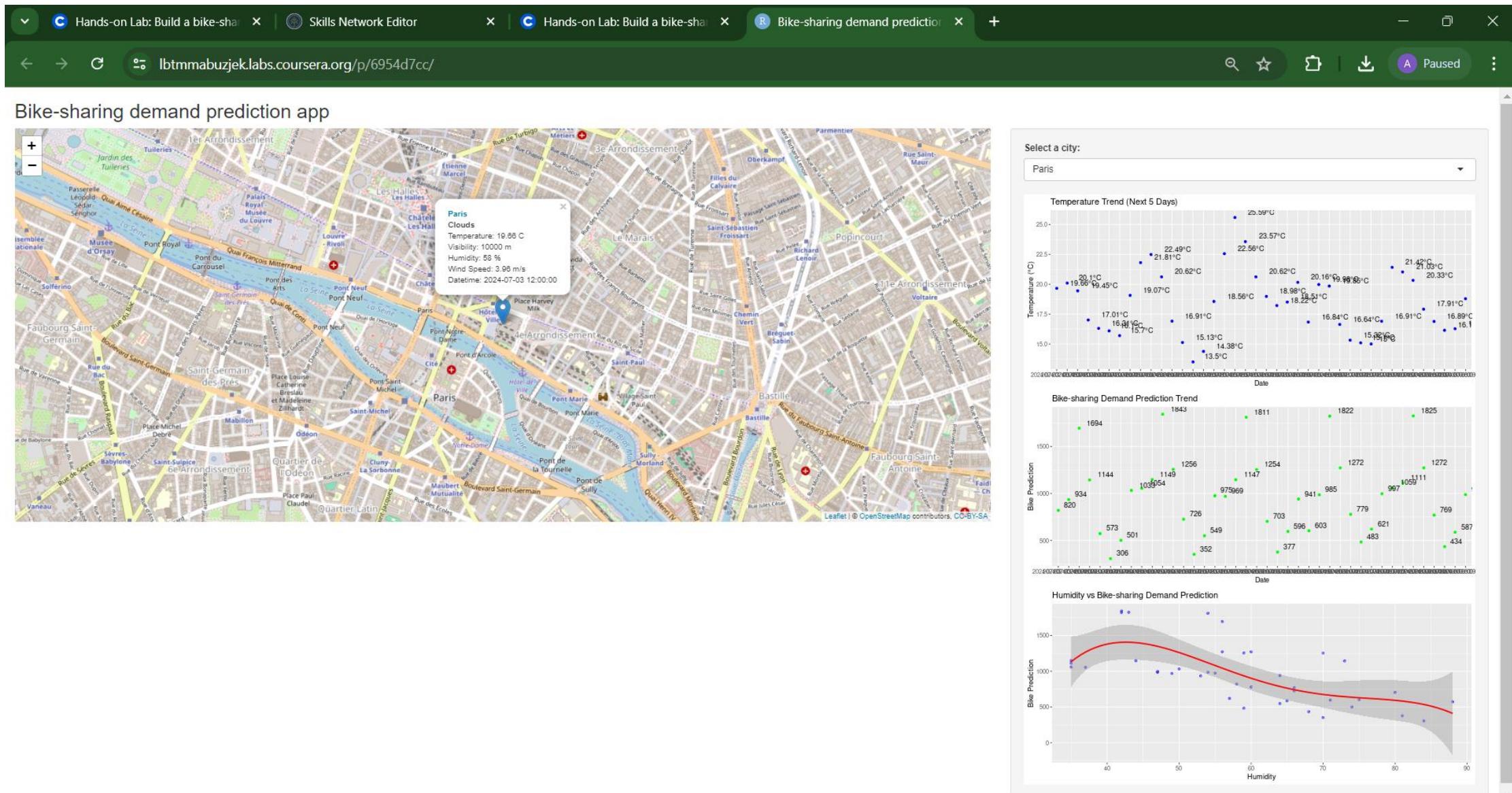
New York
Clear
Temperature: 19.04°C
Visibility: 10000 m
Humidity: 75 %
Wind Speed: 2.25 m/s
Datetime: 2024-07-03 09:00:00

Select a city:
New York

Temperature Trend (Next 5 Days)

Date	Temperature (°C)
2024-07-03	19.04°C
2024-07-04	20.03°C
2024-07-05	21.2°C
2024-07-06	21.81°C
2024-07-07	22.81°C
2024-07-08	22.47°C
2024-07-09	23.64°C
2024-07-10	24.41°C
2024-07-11	25.48°C
2024-07-12	26.64°C
2024-07-13	26.64°C
2024-07-14	26.64°C
2024-07-15	26.64°C
2024-07-16	26.64°C
2024-07-17	26.64°C
2024-07-18	26.64°C
2024-07-19	26.64°C
2024-07-20	26.64°C
2024-07-21	26.64°C
2024-07-22	26.64°C
2024-07-23	26.64°C
2024-07-24	26.64°C
2024-07-25	26.64°C
2024-07-26	26.64°C
2024-07-27	26.64°C
2024-07-28	26.64°C
2024-07-29	26.64°C
2024-07-30	26.64°C
2024-07-31	26.64°C
2024-08-01	26.64°C
2024-08-02	26.64°C
2024-08-03	26.64°C
2024-08-04	26.64°C
2024-08-05	26.64°C
2024-08-06	26.64°C
2024-08-07	26.64°C
2024-08-08	26.64°C
2024-08-09	26.64°C
2024-08-10	26.64°C
2024-08-11	26.64°C
2024-08-12	26.64°C
2024-08-13	26.64°C
2024-08-14	26.64°C
2024-08-15	26.64°C
2024-08-16	26.64°C
2024-08-17	26.64°C
2024-08-18	26.64°C
2024-08-19	26.64°C
2024-08-20	26.64°C
2024-08-21	26.64°C
2024-08-22	26.64°C
2024-08-23	26.64°C
2024-08-24	26.64°C
2024-08-25	26.64°C
2024-08-26	26.64°C
2024-08-27	26.64°C
2024-08-28	26.64°C
2024-08-29	26.64°C
2024-08-30	26.64°C
2024-08-31	26.64°C
2024-09-01	26.64°C
2024-09-02	26.64°C
2024-09-03	26.64°C
2024-09-04	26.64°C
2024-09-05	26.64°C
2024-09-06	26.64°C
2024-09-07	26.64°C
2024-09-08	26.64°C
2024-09-09	26.64°C
2024-09-10	26.64°C
2024-09-11	26.64°C
2024-09-12	26.64°C
2024-09-13	26.64°C
2024-09-14	26.64°C
2024-09-15	26.64°C
2024-09-16	26.64°C
2024-09-17	26.64°C
2024-09-18	26.64°C
2024-09-19	26.64°C
2024-09-20	26.64°C
2024-09-21	26.64°C
2024-09-22	26.64°C
2024-09-23	26.64°C
2024-09-24	26.64°C
2024-09-25	26.64°C
2024-09-26	26.64°C
2024-09-27	26.64°C
2024-09-28	26.64°C
2024-09-29	26.64°C
2024-09-30	26.64°C
2024-10-01	26.64°C
2024-10-02	26.64°C
2024-10-03	26.64°C
2024-10-04	26.64°C
2024-10-05	26.64°C
2024-10-06	26.64°C
2024-10-07	26.64°C
2024-10-08	26.64°C
2024-10-09	26.64°C
2024-10-10	26.64°C
2024-10-11	26.64°C
2024-10-12	26.64°C
2024-10-13	26.64°C
2024-10-14	26.64°C
2024-10-15	26.64°C
2024-10-16	26.64°C
2024-10-17	26.64°C
2024-10-18	26.64°C
2024-10-19	26.64°C
2024-10-20	26.64°C
2024-10-21	26.64°C
2024-10-22	26.64°C
2024-10-23	26.64°C
2024-10-24	26.64°C
2024-10-25	26.64°C
2024-10-26	26.64°C
2024-10-27	26.64°C
2024-10-28	26.64°C
2024-10-29	26.64°C
2024-10-30	26.64°C
2024-10-31	26.64°C
2024-11-01	26.64°C
2024-11-02	26.64°C
2024-11-03	26.64°C
2024-11-04	26.64°C
2024-11-05	26.64°C
2024-11-06	26.64°C
2024-11-07	26.64°C
2024-11-08	26.64°C
2024-11-09	26.64°C
2024-11-10	26.64°C
2024-11-11	26.64°C
2024-11-12	26.64°C
2024-11-13	26.64°C
2024-11-14	26.64°C
2024-11-15	26.64°C
2024-11-16	26.64°C
2024-11-17	26.64°C
2024-11-18	26.64°C
2024-11-19	26.64°C
2024-11-20	26.64°C
2024-11-21	26.64°C
2024-11-22	26.64°C
2024-11-23	26.64°C
2024-11-24	26.64°C
2024-11-25	26.64°C
2024-11-26	26.64°C
2024-11-27	26.64°C
2024-11-28	26.64°C
2024-11-29	26.64°C
2024-11-30	26.64°C
2024-12-01	26.64°C
2024-12-02	26.64°C
2024-12-03	26.64°C
2024-12-04	26.64°C
2024-12-05	26.64°C
2024-12-06	26.64°C
2024-12-07	26.64°C
2024-12-08	26.64°C
2024-12-09	26.64°C
2024-12-10	26.64°C
2024-12-11	26.64°C
2024-12-12	26.64°C
2024-12-13	26.64°C
2024-12-14	26.64°C
2024-12-15	26.64°C
2024-12-16	26.64°C
2024-12-17	26.64°C
2024-12-18	26.64°C
2024-12-19	26.64°C
2024-12-20	26.64°C
2024-12-21	26.64°C
2024-12-22	26.64°C
2024-12-23	26.64°C
2024-12-24	26.64°C
2024-12-25	26.64°C
2024-12-26	26.64°C
2024-12-27	26.64°C
2024-12-28	26.64°C
2024-12-29	26.64°C
2024-12-30	26.64°C
2024-12-31	26.64°C
2025-01-01	26.64°C
2025-01-02	26.64°C
2025-01-03	26.64°C
2025-01-04	26.64°C
2025-01-05	26.64°C
2025-01-06	26.64°C
2025-01-07	26.64°C
2025-01-08	26.64°C
2025-01-09	26.64°C
2025-01-10	26.64°C
2025-01-11	26.64°C
2025-01-12	26.64°C
2025-01-13	26.64°C
2025-01-14	26.64°C
2025-01-15	26.64°C
2025-01-16	26.64°C
2025-01-17	26.64°C
2025-01-18	26.64°C
2025-01-19	26.64°C
2025-01-20	26.64°C
2025-01-21	26.64°C
2025-01-22	26.64°C
2025-01-23	26.64°C
2025-01-24	26.64°C
2025-01-25	26.64°C
2025-01-26	26.64°C
2025-01-27	26.64°C
2025-01-28	26.64°C
2025-01-29	26.64°C
2025-01-30	26.64°C
2025-01-31	26.64°C
2025-02-01	26.64°C
2025-02-02	26.64°C
2025-02-03	26.64°C
2025-02-04	26.64°C
2025-02-05	26.64°C
2025-02-06	26.64°C
2025-02-07	26.64°C
2025-02-08	26.64°C
2025-02-09	26.64°C
2025-02-10	26.64°C
2025-02-11	26.64°C
2025-02-12	26.64°C
2025-02-13	26.64°C
2025-02-14	26.64°C
2025-02-15	26.64°C
2025-02-16	26.64°C
2025-02-17	26.64°C
2025-02-18	26.64°C
2025-02-19	26.64°C
2025-02-20	26.64°C
2025-02-21	26.64°C
2025-02-22	26.64°C
2025-02-23	26.64°C
2025-02-24	26.64°C
2025-02-25	26.64°C
2025-02-26	26.64°C
2025-02-27	26.64°C
2025-02-28	26.64°C
2025-02-29	26.64°C
2025-03-01	26.64°C
2025-03-02	26.64°C
2025-03-03	26.64°C
2025-03-04	26.64°C
2025-03-05	26.64°C
2025-03-06	26.64°C
2025-03-07	26.64°C
2025-03-08	26.64°C
2025-03-09	26.64°C
2025-03-10	26.64°C
2025-03-11	26.64°C
2025-03-12	26.64°C
2025-03-13	26.64°C
2025-03-14	26.64°C
2025-03-15	26.64°C
2025-03-16	26.64°C
2025-03-17	26.64°C
2025-03-18	26.64°C
2025-03-19	26.64°C
2025-03-20	26.64°C
2025-03-21	26.64°C
2025-03-22	26.64°C
2025-03-23	26.64°C
2025-03-24	26.64°C
2025-03-25	26.64°C
2025-03-26	26.64°C
2025-03-27	26.64°C
2025-03-28	26.64°C
2025-03-29	26.64°C
2025-03-30	26.64°C
2025-03-31	26.64°C
2025-04-01	26.64°C
2025-04-02	26.64°C
2025-04-03	26.64°C
2025-04-04	26.64°C
2025-04-05	26.64°C
2025-04-06	26.64°C
2025-04-07	26.64°C
2025-04-08	26.64°C
2025-04-09	26.64°C
2025-04-10	26.64°C
2025-04-11	26.64°C
2025-04-12	26.64°C
2025-04-13	26.64°C
2025-04-14	26.64°C
2025-04-15	26.64°C
2025-04-16	26.64°C
2025-04-17	26.64°C
2025-04-18	26.64°C
2025-04-19	26.64°C
2025-04-20	26.64°C
2025-04-21	26.64°C
2025-04-22	26.64°C
2025-04-23	26.64°C
2025-04-24	26.64°C
2025-04-25	26.64°C
2025-04-26	26.64°C
2025-04-27	26.64°C
2025-04-28	26.64°C
2025-04-29	26.64°C
2025-04-30	26.64°C
2025-05-01	26.64°C
2025-05-02	26.64°C
2025-05-03	26.64°C
2025-05-04	26.64°C
2025-05-05	26.64°C
2025-05-06	26.64°C
2025-05-07	26.64°C
2025-05-08	26.64°C
2025-05-09	26.64°C
2025-05-10	26.64°C
2025-05-11	26.64°C
2025-05-12	26.64°C
2025-05-13	26.64°C
2025-05-14	26.64°C
2025-05-15	26.64°C
2025-05-16	26.64°C
2025-05-17	26.64°C
2025-05-18	26.64°C
2025-05-19	26.64°C
2025-05-20	26.64°C
2025-05-21	26.64°C
2025-05-22	26.64°C
2025-05-23	26.64°C
2025-05-24	26.64°C
2025-05-25	26.64°C
2025-05-26	26.64°C
2025-05-27	26.64°C
2025-05-28	26.64°C
2025-05-29	26.64°C
2025-05-30	26.64°C
2025-05-31	26.64°C
2025-06-01	26.64°C
2025-06-02	26.64°C
2025-06-03	26.64°C
2025-06-04	26.64°C
2025-06-05	26.64°C
2025-06-06	26.64°C
2025-06-07	26.64°C
2025-06-08	26.64°C
2025-06-09	26.64°C
2025-06-10	26.64°C
2025-06-11	26.64°C
2025-06-12	26.64°C
2025-06-13	26.64°C
2025-06-14	26.64°C
2025-06-15	26.64°C
2025-06-16	26.64°C
2025-06-17	26.64°C
2025-06-18	26.64°C
2025-06-19	26.64°C
2025-06-20	2

ADDING A STATIC HUMIDITY AND BIKE SHARING DEMAND PREDICTION CORRELATION PLOT

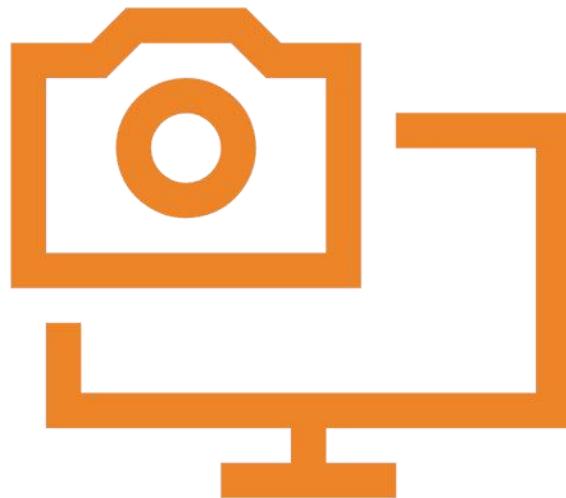


CONCLUSION



- 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

The screenshot shows the RStudio interface with the following components:

- Header Bar:** Includes back, forward, search, and other navigation icons.
- Title Bar:** Shows the URL coursera.org/learn/data-science-with-r-capstone-project/ungradedLab/4b74C/hands-on-lab-build-a-bike-sharing-demand-prediction-app-using-coursera-labs/lab?path=%2F.
- RStudio Logo:** coursera
- File Menu:** File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, Help.
- Code Editor:** Displays the `model_prediction.R` file with the following content:9 # and make predictions
10 source("model_prediction.R")
11
12 # Function to generate test weather data
13 test_weather_data_generation <- function() {
14 city_weather_bike_df <- generate_city_weather_bike_data()
15 stopifnot(length(city_weather_bike_df) > 0)
16 print(city_weather_bike_df)
17 return(city_weather_bike_df)
18 }
19
20 # Create a RShiny server
21 shinyServer(function(input, output) {
22 # Test generate_city_weather_bike_data() function
23 city_weather_bike_df <- test_weather_data_generation()
24
25 # Create another data frame called cities_max_bike with each row contains city location info and max bike
26 # prediction for the city
27 cities_max_bike <- city_weather_bike_df %>%
28 group_by(CITY_ASCII) %>%
29 summarise(MAX_BIKE_PREDICTION = max(BIKE_PREDICTION),
30 LAT = first(LAT),
31 LNG = first(LNG),
32 LABEL = first(LABEL),
33 DETAILED_LABEL = first(DETAILED_LABEL))
34
35 # Define color factor
36 color_levels <- colorFactor(c("green", "yellow", "red"), levels = c("small", "medium", "large"))
37
38 # Render leaflet map based on dropdown selection
39 observeEvent(input\$city_dropdown, {
40 if (input\$city_dropdown == "All") {
41 output\$city_bike_map <- renderLeaflet({
42 leaflet(data = cities_max_bike) %>%
43 addTiles() %>%
44 addCircleMarkers(
45 radius = ~ifelse(MAX_BIKE_PREDICTION <= 50, 6,
46 ifelse(MAX_BIKE_PREDICTION <= 100, 10, 12),
47 color = ~color_levels(ifelse(MAX_BIKE_PREDICTION <= 50, "small",
48 ifelse(MAX_BIKE_PREDICTION <= 100, "medium", "large"))),
49 stroke = FALSE,
50 fillOpacity = 0.5,
51 label = ~as.character(LABEL),
52 popup = ~as.character(LABEL)
53) %>%
54 setView(lng = cities_max_bike\$LNG[1], lat = cities_max_bike\$LAT[1], zoom = 4)
55 }
56 })
57 }
58 <function>(input, output) :
59
- Environment Tab:** Shows the Global Environment and Functions pane.
- Files Tab:** Shows the project structure with files like `model.csv`, `model_prediction.R`, `selected_cities.csv`, `server.R`, and `ui.R`.

Code Snippets

The screenshot shows a code editor interface with a dark theme. On the left, there are several icons for file operations like new, open, save, and search. The main area contains R code for web scraping. The code starts by checking if the rvest library is installed, installing it if necessary, and then loading the library. It defines a URL to the Wikipedia page for bicycle-sharing systems and reads the HTML content into a root node. It then finds all child table nodes and prints a preview of the first table. The output shows the HTML structure of the table, including columns for Country, City / Region, and Name.

```
# Check if need to install rvest library
if(!require("rvest")) install.packages("rvest")

# Load the library
library(rvest)

Loading required package: rvest

[ ] # Define the URL
url <- "https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems"

# Get the root HTML node by calling the `read_html()` method with URL
root_node <- read_html(url)

[ ] # Get all child <table> nodes
table_nodes <- html_nodes(root_node, "table")

# Inspect the tables to find the right one (in this case, it's the first one)
# Print a preview of the first table to confirm
table_nodes[[1]]

{html_node}
<table class="wikitable sortable sticky-header" style="background:#f8f9faff;">
[1] <tbody>\n<tr>\n<th>Country</th>\n<th>City / Region</th>\n<th>Name</th>\n< ...
```

Code snippets

The screenshot shows a Jupyter Notebook interface with the following content:

```
[ ] {html_node}
[+] ↗ {table class="wikitable sortable sticky-header" style="background:#f8f9faff;"}
[1] <tbody>\n<tr>\n<th>Country</th>\n<th>City / Region</th>\n<th>Name</th>\n<th>System</th>\n<th>Operator</th>\n<th>Launched</th>\n<th>Discontinued</th>\n<th>Length:760</th>\n<th>Length:760</th>\n<th>Length:760</th>\n<th>Length:760</th>\n<th>Class :character</th>\n<th>Class :character</th>\n<th>Class :character</th>\n<th>Class :character</th>\n<th>Mode :character</th>\n<th>Mode :character</th>\n<th>Mode :character</th>\n<th>Mode :character</th>
```

Cell 1:

```
# Convert the bike-sharing system table into a dataframe
bike_sharing_df <- html_table(table_nodes[[1]], fill = TRUE)
```

Cell 2:

```
# Summarize the dataframe
summary(bike_sharing_df)
```

Cell 3:

```
Country      City / Region      Name      System
Length:760    Length:760        Length:760    Length:760
Class :character Class :character Class :character Class :character
Mode :character Mode :character Mode :character Mode :character
Operator      Launched        Discontinued
Length:760    Length:760        Length:760
Class :character Class :character Class :character
Mode :character Mode :character Mode :character
```

Cell 4:

```
# Export the dataframe into a csv file
write.csv(bike_sharing_df, "raw_bike_sharing_systems.csv", row.names = FALSE)
```

Code snippets



The screenshot shows a code editor window with a dark theme. At the top, there is a toolbar with various icons: up, down, left, right, copy, paste, find, etc. Below the toolbar, the title "DATA COLLECTION" is displayed in bold capital letters, preceded by a downward arrow icon. The main area contains five code blocks, each starting with a square bracket and a space []. The code is written in R, demonstrating how to collect current weather data from an API.

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

# Load the httr library
library(httr)

[ ] # 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"

[ ] # 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")

[ ] # 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'

[ ] # 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](#) [+ Text](#)

Code snippets

```
▶ # 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 Ripynd

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

+ Code + Text

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

```
# 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 <- "fbfb5094275db52c25ded246628b08ec7"

# 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()
    pressure <- c()
    humidity <- c()
    wind_speed <- c()
    wind_deg <- c()
    forecast_datetime <- c()
    season <- c()
```

```
# 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, result$city)
  weather <- c(weather, result$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

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Module 2-1 Data Wrangling R.ipynb
- Header:** File Edit View Insert Runtime Tools Help Last edited on 2 July
- Toolbar:** Comment Share Gemini
- Code Cell 1:** 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
- Code Cell 2:** [] 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
- Code Cell 3:** [] # Download raw_bike_sharing_systems.csv
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-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/IBMDriverSkillsNetwork-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/IBMDriverSkillsNetwork-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/IBMDriverSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_seoul_bike_sharing.csv"
download.file(url, destfile = "raw_seoul_bike_sharing.csv")

[1] # Define paths for each dataset

Code snippets

```
▶ # 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

Code snippets

```
[ ] #> A tibble: 4 × 2
#>   variable    class
#>   <chr>      <chr>
#> 1 COUNTRY    character
#> 2 CITY       character
#> 3 SYSTEM     character
#> 4 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>
#> 1 4115[22]
#> 2 310[59]
#> 3 500[72]
#> 4 [75]
#> 5 180[76]
#> 6 600[77]
#> 7 [78]
#> 8 initially 800 (later 2500)
#> 9 100 (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
1	Brussels	<chr> <chr> <chr>
2	Limassol (& Agios Dometios, Aglandjia, Dali, Engomi, Latsia,...	3 Gen... 4115[22]
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]  
ref_link <- "[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

```
# 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.   :0.00000  Min.   : 0.0000  Min.   :0.00000  Length:8465
1st Qu.:0.00000  1st Qu.: 0.0000  1st Qu.:0.00000  Class :character
Median :0.01000  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

```
# 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(RENTEDBIKECOUNT, 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")
```

[] # 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

Module 3-1 Exploratory Data Analysis R.ipynb ★

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

+ Code + Text

operational_hours
1 8465

Load RSQLite library
library(RSQLite)

Connect to SQLite database
con <- dbConnect(SQLite(), dbname = "my_database.db")

Task 3: Weather Outlook for Seoul over the next 3 hours
query <- "SELECT *
FROM cities_weather_forecast
WHERE CITY = 'Seoul'
ORDER BY FORECAST_DATETIME
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 2021-04-16 12:00:00

[] # Task 4: Seasons in Seoul Bike Sharing Dataset
query_task4 <- "SELECT SEASONS
FROM seoul_bike_sharing;"
result_task4 <- dbGetQuery(con, query_task4)

Print the result
print(result_task4)

SEASONS
1 Winter
2 Winter
3 Winter
4 Winter

Task 5: Date Range in Seoul Bike Sharing Dataset
query_task5 <- "SELECT MIN(DATE) AS first_date, MAX(DATE) AS last_date
FROM seoul_bike_sharing;"
result_task5 <- dbGetQuery(con, query_task5)

Print the result
print(result_task5)

first_date last_date
1 01/01/2018 31/12/2017

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

+ Code + Text

Code snippets

```
# 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
```

```
# 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)
```

```
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
```

```
# 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,
SORT(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)
```

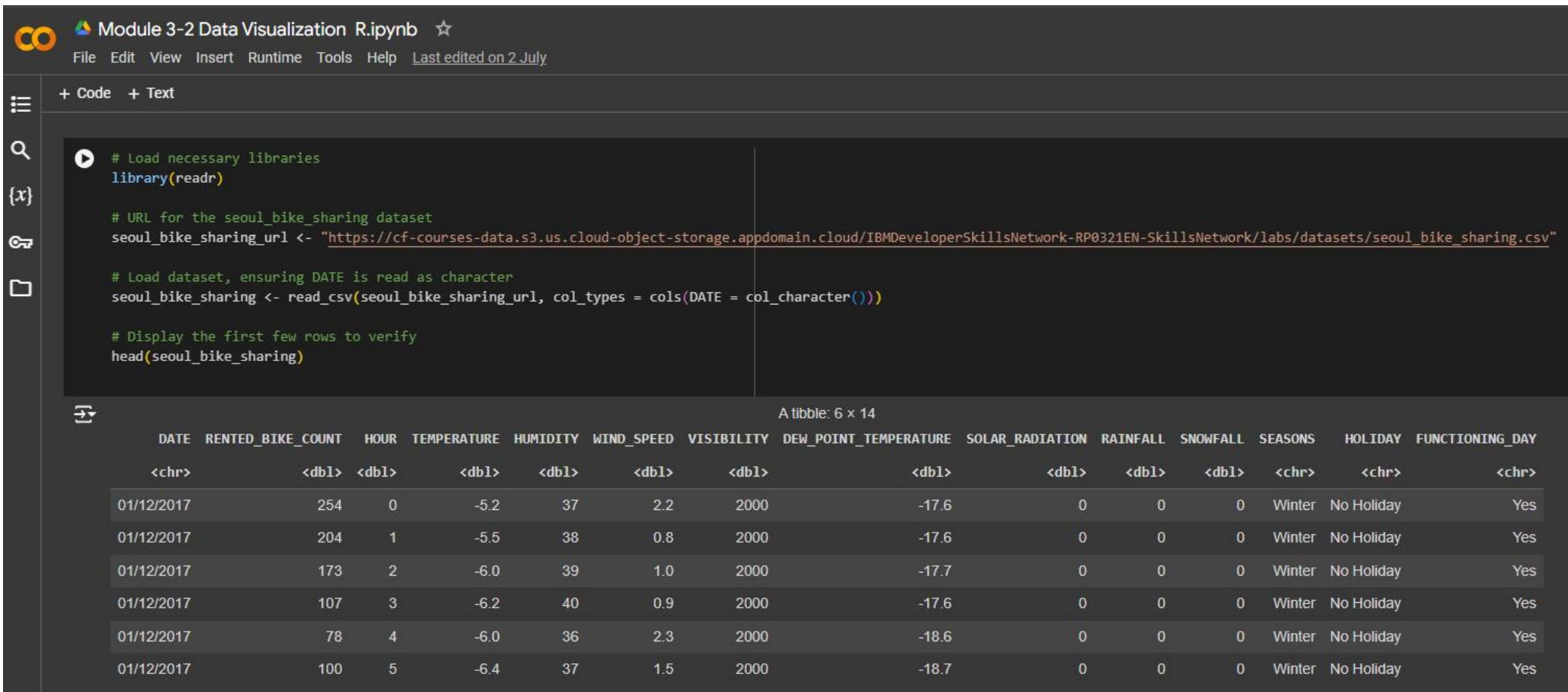
SEASONS	avg_bike_count	min_bike_count	max_bike_count	std dev bike count
1 Autumn	924.1105	2	3298	617.3885
2 Spring	746.2542	2	3251	618.5247
3 Summer	1834.0734	9	3556	690.0884
4 Winter	225.5412	3	937	150.3374

```
[ ] # 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



The screenshot shows a Jupyter Notebook cell with the following R code:

```
# Load necessary libraries
library(readr)

# URL for the seoul_bike_sharing dataset
seoul_bike_sharing_url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/seoul_bike_sharing.csv"

# Load dataset, ensuring DATE is read as character
seoul_bike_sharing <- read_csv(seoul_bike_sharing_url, col_types = cols(DATE = col_character()))

# Display the first few rows to verify
head(seoul_bike_sharing)
```

Below the code, the output shows a table titled "A tibble: 6 × 14" with the following data:

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

```
▶ # 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 ...
$ 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 0.01 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

```
▶ # 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 0.01 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. :0.0000  Min. : 27  Min. : -30.600
1st Qu.:42.00 1st Qu.:0.9000  1st Qu.: 935 1st Qu.: -5.100
Median :57.00  Median :1.5000  Median :1690  Median : 4.700
Mean   :58.15  Mean : 1.726   Mean :1434  Mean  : 3.945
3rd Qu.:74.00 3rd Qu.:2.3000 3rd Qu.:2000 3rd Qu.: 15.200
Max.  :98.00  Max. :7.400   Max. :2000  Max. : 27.200

  SOLAR_RADIATION      RAINFALL      SNOWFALL      SEASONS
Min. :0.00000  Min. : 0.0000  Min. :0.00000  Length:8465
1st Qu.:0.00000 1st Qu.: 0.0000  1st Qu.:0.00000  Class :character
Median :0.01000  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

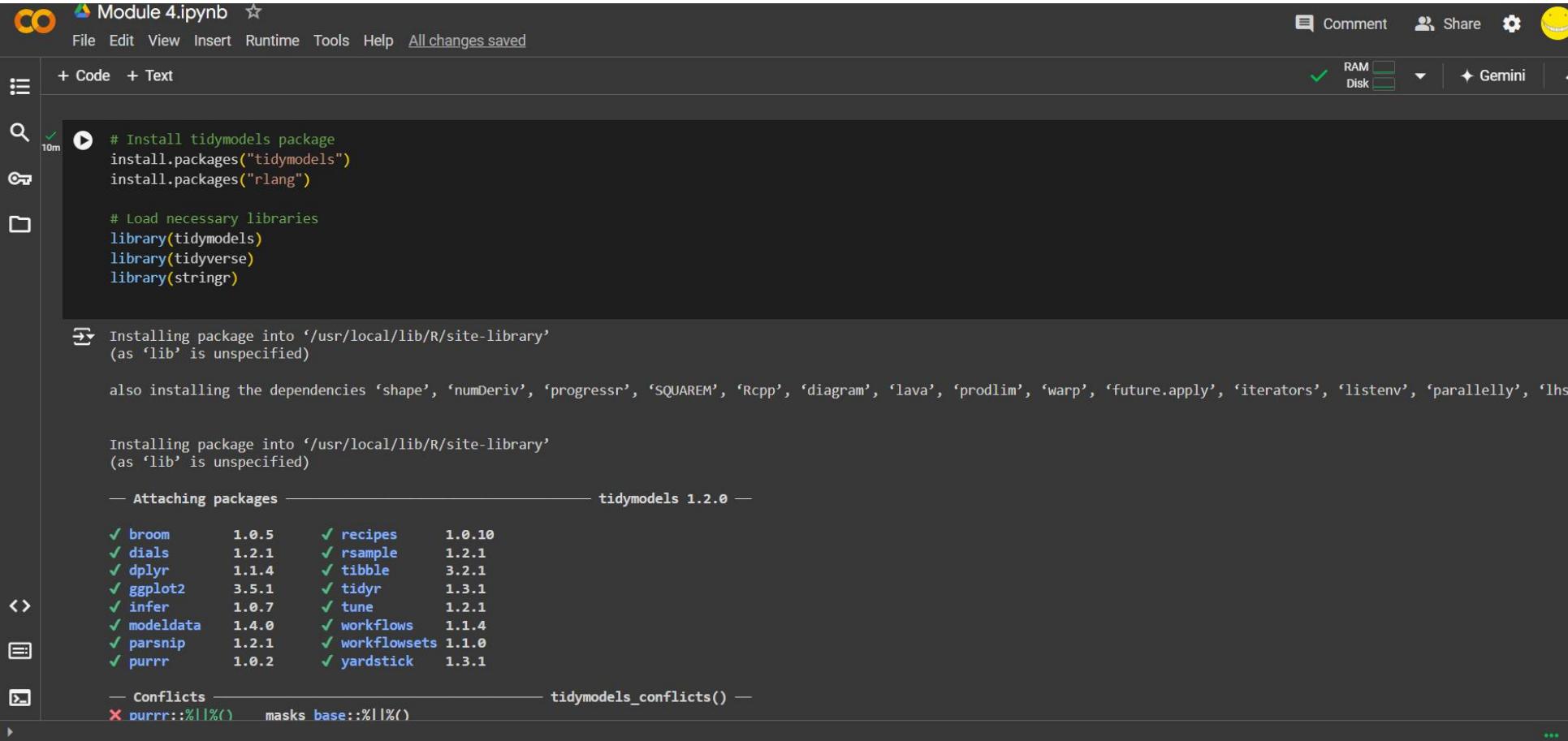
[ ] # Task 7: Calculate expected number of records
expected_records <- 365
expected_records

☒ 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



The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Module 4.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved.
- Code Cell:** Contains R code for installing packages:

```
# Install tidymodels package
install.packages("tidyverse")
install.packages("rlang")

# Load necessary libraries
library(tidyverse)
library(rlang)
library(stringr)
```
- Output:** Shows the output of the R code, including package installation messages:

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

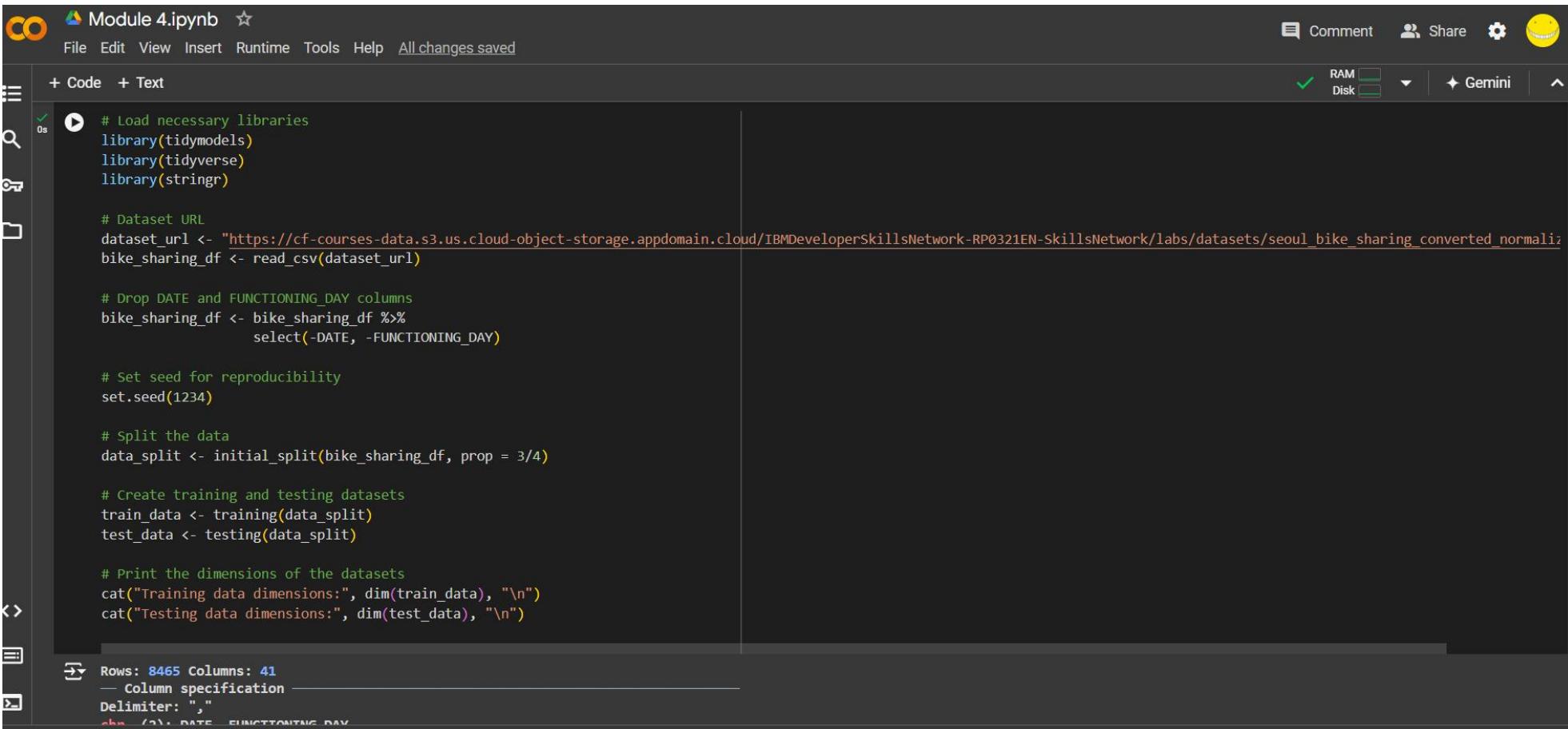
also installing the dependencies ‘shape’, ‘numDeriv’, ‘progressr’, ‘SQUAREM’, ‘Rcpp’, ‘diagram’, ‘lava’, ‘prodlim’, ‘warp’, ‘future.apply’, ‘iterators’, ‘listenv’, ‘parallelly’, ‘lhs’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

— Attaching packages —————— tidyverse 1.2.0 ——————
✓ broom     1.0.5   ✓ recipes    1.0.10
✓ dials      1.2.1   ✓ rsample     1.2.1
✓ dplyr      1.1.4   ✓ tibble     3.2.1
✓ ggplot2    3.5.1   ✓ tidyverse   1.3.1
✓ infer       1.0.7   ✓ tune       1.2.1
✓ modeldata   1.4.0   ✓ workflows  1.1.4
✓ parsnip     1.2.1   ✓ workflowsets 1.1.0
✓ purrr      1.0.2   ✓ yardstick  1.3.1

— Conflicts —————— tidyverse_conflicts() ——————
✗ purrr::%>%() masks base::%>%()
```

Code snippets



The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Module 4.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved
- Code Cell:** Contains R code for data loading, cleaning, splitting, and dimension printing.
- Output Cell:** Displays the output of the last command: "Rows: 8465 Columns: 41".
- Environment:** RAM and Disk status are shown as green.
- Tools:** Comment, Share, Settings, and a smiley face icon.

```
# 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_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)

# Print the dimensions of the datasets
cat("Training data dimensions:", dim(train_data), "\n")
cat("Testing data dimensions:", dim(test_data), "\n")
```

Rows: 8465 Columns: 41
— Column specification —
Delimiter: ","
dbl / (1, DATE, FUNCTIONING_DAY)

Code snippets

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

# Dataset URL
dataset_url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDriverSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/seoul_bike_sharing_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,
RAINFALL + SNOWFALL, data = data)
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

```
# 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 *** 
`^21`        446.58    34.09  12.120 < 2e-16 ***
```

```
# 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

# 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 x ui.R x server.R x
1 # Install and import required libraries
2 require(shiny)
3 require(ggplot2)
4 require(leaflet)
5 require(tidyverse)
6 require(httr)
7 require(scales)
8 # Import model_prediction R which contains methods to call OpenWeather API
9 # and make predictions
10 source("model_prediction.R")
11
12 # Function to generate test weather data
13 test_weather_data_generation <- function() {
14   city_weather_bike_df <- generate_city_weather_bike_data()
15   stopifnot(length(city_weather_bike_df) > 0)
16   print(city_weather_bike_df)
17   return(city_weather_bike_df)
18 }
19
20 # Create a RShiny server
21 shinyServer(function(input, output) {
22   # Test generate_city_weather_bike_data() function
23   city_weather_bike_df <- test_weather_data_generation()
24
25   # Create another data frame called cities_max_bike with each row contains city location info and max bike
26   # prediction for the city
27   cities_max_bike <- city_weather_bike_df %>%
28     group_by(CITY_ASCII) %>%
29     summarise(MAX BIKE PREDICTION = max(BIKE PREDICTION),
30             LAT = first(LAT),
31             LNG = first(LNG),
32             LABEL = first(LABEL),
33             DETAILED_LABEL = first(DETAILED_LABEL))
34
35   # Define color factor
36   color_levels <- colorFactor(c("green", "yellow", "red"), levels = c("small", "medium", "large"))
37
38   # Render leaflet map based on dropdown selection
39   observeEvent(input$city_dropdown, {
40     if (input$city_dropdown == "All") {
41       output$city_bike_map <- renderLeaflet({
42         leaflet(data = cities_max_bike) %>%
43         addTiles() %>%
44         addCircleMarkers(
45           radius = ~ifelse(MAX BIKE PREDICTION <= 50, 6,
46                             ifelse(MAX BIKE PREDICTION <= 100, 10, 12)),
35:24  ↴ <function>(input, output) ↴
```

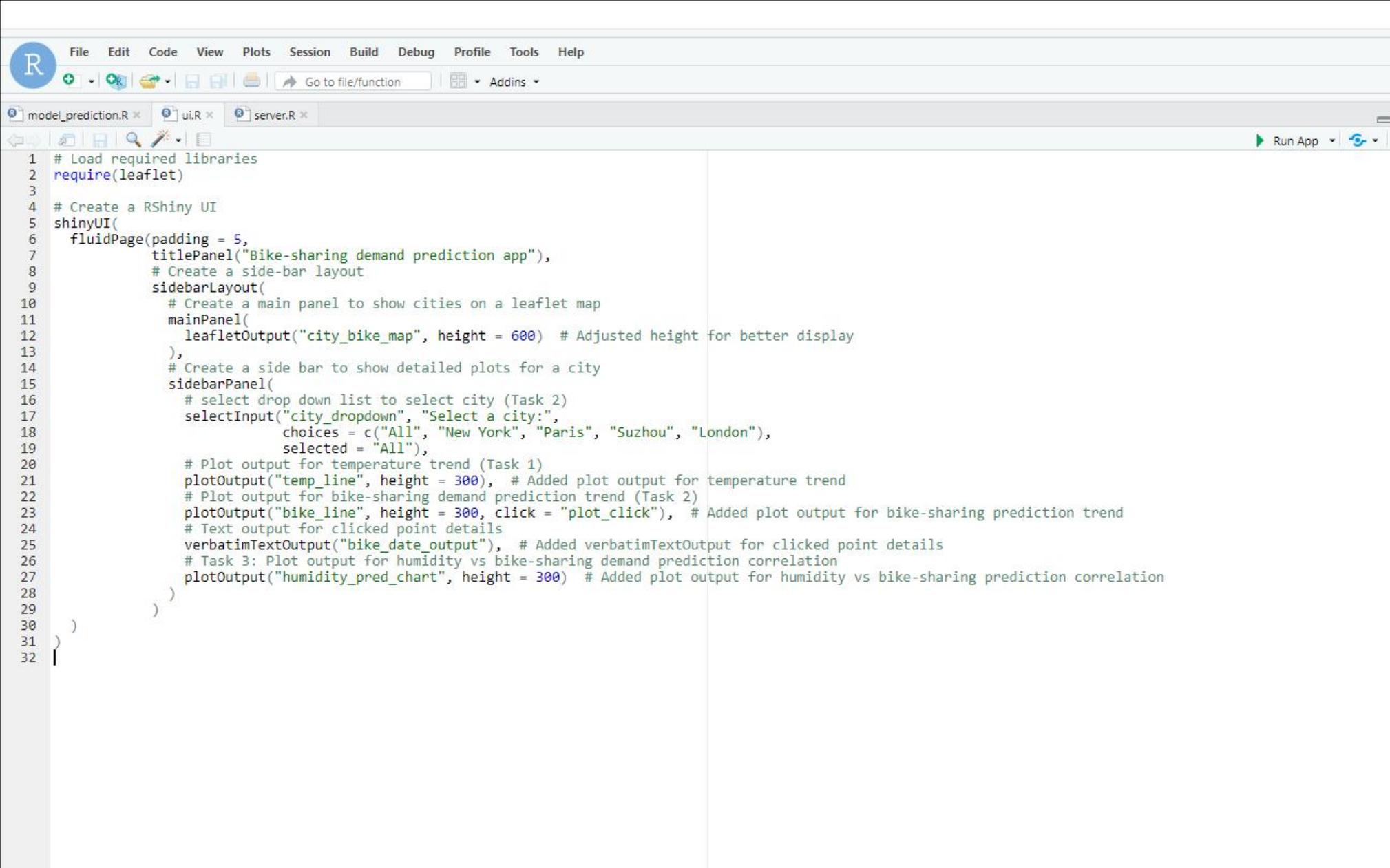
```
model_prediction.R x ui.R x server.R x
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
35:24  ↴ <function>(input, output) ↴
Console
```

Code snippets

The screenshot shows an RStudio interface with three tabs open: model_prediction.R, ui.R, and server.R. The code in model_prediction.R is displayed in the main pane:

```
87 arrange(FORECASTDATETIME) # Ensure data is sorted by FORECASTDATETIME
88
89 p <- ggplot(selected_city_data, aes(x = FORECASTDATETIME, y = BIKE_PREDICTION)) +
90   geom_line(color = "green") +
91   geom_point(color = "green") +
92   geom_text(aes(label = paste0(BIKE_PREDICTION)), vjust = -0.5, hjust = -0.5) +
93   labs(title = "Bike-sharing Demand Prediction Trend",
94        x = "Date", y = "Bike Prediction")
95
96 if (!is.null(input$plot_click)) {
97   click <- input$plot_click
98   near_point <- nearest_data(selected_city_data, click)
99   p <- p +
100     geom_text(data = near_point, aes(label = paste0("Demand: ", BIKE_PREDICTION, "\nDate: ", FORECASTDATETIME)),
101               color = "black", vjust = -1, hjust = -0.5, size = 4, parse = TRUE)
102 }
103
104 p
105 }
106
107
108 # Task 3: Render static humidity and bike-sharing demand prediction correlation plot
109 output$humidity_pred_chart <- renderPlot({
110   if (input$city_dropdown != "All") {
111     selected_city_data <- city_weather_bike_df %>%
112       filter(CITY_ASCII == input$city_dropdown)
113
114     ggplot(selected_city_data, aes(x = HUMIDITY, y = BIKE_PREDICTION)) +
115       geom_point(color = "blue", alpha = 0.5) +
116       geom_smooth(method = "lm", formula = y ~ poly(x, 4), color = "red") +
117       labs(title = "Humidity vs Bike-sharing Demand Prediction",
118            x = "Humidity", y = "Bike Prediction")
119   }
120 })
121
122 # Render text output for clicked point details
123 output$bike_date_output <- renderText({
124   if (!is.null(input$plot_click)) {
125     click <- input$plot_click
126     near_point <- nearest_data(selected_city_data, click)
127     paste("Clicked Demand:", near_point$BIKE_PREDICTION, "\nClicked Date:", near_point$FORECASTDATETIME)
128   }
129 })
130
131 })
132
```

The code uses ggplot2 for data visualization, including line plots and points. It also includes a smooth line for the correlation between humidity and bike prediction. A text output is generated for the clicked point.



The image shows the RStudio IDE interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, Tools, and Help. The left sidebar has icons for file operations like Open, Save, and Print. The main workspace shows three files: model_prediction.R, ui.R, and server.R. The server.R file is open and contains the following R code:

```
1 # Load required libraries
2 require(leaflet)
3
4 # Create a RShiny UI
5 shinyUI(
6   fluidPage(padding = 5,
7     titlePanel("Bike-sharing demand prediction app"),
8     # Create a side-bar layout
9     sidebarLayout(
10       # Create a main panel to show cities on a leaflet map
11       mainPanel(
12         leafletOutput("city_bike_map", height = 600) # Adjusted height for better display
13       ),
14       # Create a side bar to show detailed plots for a city
15       sidebarPanel(
16         # select drop down list to select city (Task 2)
17         selectInput("city_dropdown", "Select a city:",
18           choices = c("All", "New York", "Paris", "Suzhou", "London"),
19           selected = "All"),
20         # Plot output for temperature trend (Task 1)
21         plotOutput("temp_line", height = 300), # Added plot output for temperature trend
22         # Plot output for bike-sharing demand prediction trend (Task 2)
23         plotOutput("bike_line", height = 300, click = "plot_click"), # Added plot output for bike-sharing prediction trend
24         # Text output for clicked point details
25         verbatimTextOutput("bike_date_output"), # Added verbatimTextOutput for clicked point details
26         # Task 3: Plot output for humidity vs bike-sharing demand prediction correlation
27         plotOutput("humidity_pred_chart", height = 300) # Added plot output for humidity vs bike-sharing prediction correlation
28       )
29     )
30   )
31 }
32 }
```