Applied Data Science with R Capstone project

Ali Osman CALI

27/02/2024

Outline

Executive Summary

Point 1: Data Preprocessing and Exploration

The provided R code demonstrates robust data preprocessing and exploration techniques on various datasets. In the first part, the focus is on Seoul bike-sharing data. Key tasks include loading the dataset, transforming date variables, and creating insightful visualizations to understand temporal trends and relationships. The second part involves a machine learning approach using the tidymodels package to build and evaluate linear regression models for predicting rented bike counts.

Point 2: Linear Regression Modelling

Sub Point 1: The linear regression modelling process involves splitting the data into training and testing sets, building models using weather variables, and evaluating model performance. Two models are developed - one considering only weather variables and another incorporating all available variables.

Sub Point 2: Model evaluation metrics such as R-squared and Root Mean Squared Error (RMSE) are calculated to assess the predictive performance of the linear regression models. Additionally, the coefficients of the model variables are examined, revealing insights into their impact on rented bike counts.

Sub Point 3: To enhance the models, regularization techniques, specifically Lasso and Elastic-Net, are applied. These techniques introduce penalties to prevent overfitting and improve generalization to new data.

Point 3: Shiny Web Application Development

The provided R code showcases the development of a Shiny web application for interactive data exploration. Users can dynamically select continuous and categorical variables, adjust visualization settings (such as histogram bins and colors), and explore the distribution of numerical and categorical variables through a user-friendly interface.

Point 4: Model Comparison and Selection

Multiple models, including polynomial regression and interactions, are built and compared using various performance metrics. The application of different penalties in glmnet models highlights the impact of regularization on model performance.

Point 5: Visualization and Interpretation

The Shiny app incorporates a clear visualization of the mtcars dataset, allowing users to explore the distribution of numerical and categorical variables. The inclusion of a variable map guide enhances the interpretability of the visualizations, providing a comprehensive understanding of the dataset's structure.

This executive summary provides an overview of the key points covered in the provided R code, spanning data preprocessing, linear regression modelling, Shiny app development, model comparison, and insightful visualizations.

Introduction

- **Point 1:** Exploratory Data Analysis in R
- Exploratory Data Analysis (EDA) is a critical phase in the data science lifecycle, serving as the foundation for model development and decision-making. In the following analysis, R programming is employed to conduct EDA on two distinct datasets: Seoul bike-sharing data and the mtcars dataset. This introduction outlines the primary objectives and methodologies employed in the exploration of these datasets.
- **Point 2:** Seoul Bike-Sharing Data Exploration
- The first dataset under examination is the Seoul bike-sharing dataset, sourced from cloud-based storage. The initial steps involve loading the data, transforming date variables, and preparing the dataset for analysis. Subsequently, a variety of visualizations are generated to gain insights into temporal patterns, weather impacts, and the overall distribution of rented bike counts.
- **Point 3:** Linear Regression Modelling in R
- Building on the EDA phase, the analysis extends to predictive modelling using linear regression. The objective is to develop models capable of forecasting bike rentals based on various weather and environmental factors. This section introduces the steps involved in model specification, training, evaluation, and the application of regularization techniques to enhance predictive performance.
- **Point 4:** Interactive Data Exploration with Shiny
- The exploration toolkit expands with the development of an interactive Shiny web application. This application enables users to dynamically select variables, adjust visualization parameters, and explore the distribution of numerical and categorical features in real-time. The creation of this Shiny app enhances the accessibility and user-friendliness of the data exploration process.
- *Sub Point 1:* The Shiny app includes features such as variable selection, histogram customization, and a variable map guide to assist users in navigating and interpreting the datasets. This interactive component ensures a seamless and engaging exploration experience.
- *Sub Point 2:* The integration of Shiny into the data exploration workflow adds an element of interactivity and user control, facilitating a more intuitive understanding of dataset characteristics. Users can iteratively refine their exploration based on the insights gained from visualizations.
- In summary, this introduction provides an overview of the multifaceted approach employed in exploring two datasets using R. From initial data preprocessing and visualization to predictive modelling and the development of an interactive Shiny application, the analysis encompasses a comprehensive range of exploratory techniques.

Methodology

1. Performing Data Collection: The initial step involves collecting the necessary datasets for analysis. In this case, the Seoul bike-sharing dataset and the mtcars dataset were obtained from external sources. The data collection process ensures that relevant information is accessible for subsequent stages of the analysis.

2. Performing Data Wrangling: Data wrangling is crucial for preparing the datasets for analysis. This includes tasks such as handling missing values, transforming data types, and addressing any anomalies in the datasets. During this phase, R packages like tidyverse are utilized for efficient data manipulation and cleaning.

3. Performing Exploratory Data Analysis (EDA) using SQL and Visualization: Exploratory Data Analysis is conducted to uncover patterns, relationships, and trends within the datasets. SQL queries may be employed for structured data exploration, while visualizations using libraries such as ggplot2 aid in gaining insights into the distribution and characteristics of the variables.

4. Performing Predictive Analysis Using Regression Models: Predictive analysis involves building regression models to forecast outcomes based on the available features. Linear regression models, including variations like Lasso and Elastic-Net, are employed to establish relationships between predictor variables and the target variable, such as the rented bike count.

5. Building the Baseline Model: The baseline model is constructed using fundamental regression techniques. This serves as a starting point for predictive modelling, providing a benchmark against which the performance of more complex models can be compared. The baseline model captures the basic relationships within the data without incorporating additional complexities.

6. Improving the Baseline Model: Building upon the baseline model, improvements are sought through various strategies. This may involve feature engineering, incorporating interaction terms, or utilizing regularization techniques to enhance the model's predictive capabilities. Iterative adjustments and optimizations are made to achieve a more accurate and robust predictive model.

7. Building an R Shiny Dashboard App: To enhance the interactive exploration of datasets, a Shiny dashboard app is developed. This app allows users to dynamically select variables, customize visualizations, and explore data distribution in real-time. The integration of Shiny adds an interactive layer to the analysis, facilitating a user-friendly and engaging exploration experience.

In summary, the methodology encompasses the entire data analysis lifecycle, from data collection and wrangling to exploratory and predictive analysis. The progression from baseline modelling to model enhancements and the development of a Shiny dashboard reflects a comprehensive and iterative approach to extracting insights from the datasets.

Methodology

Data collection

The data collection process involved obtaining diverse datasets to facilitate comprehensive analysis. Key phrases and flowcharts illustrate the steps taken for each dataset:

1. **Seoul Bike Sharing Dataset (seoul_bike_sharing.csv):**
    - **Data Source:** Accessed from an online repository.
    - **Collection Method:** Downloaded the dataset directly from the source.
    - **Storage:** Saved as a CSV file for subsequent analysis.
2. **Cities Weather Forecast Dataset (cities_weather_forecast.csv):**
    - **Data Source:** Retrieved from a reliable weather forecasting service.
    - **Collection Method:** Accessed the API of the service to fetch weather forecasts.
    - **Storage:** Saved as a CSV file for further use.
3. **World Cities Dataset (world_cities.csv):**
    - **Data Source:** Obtained from a global cities database.
    - **Collection Method:** Downloaded the dataset from the database repository.
    - **Storage:** Saved as a CSV file for integration into the analysis.
4. **Bike Sharing Systems Dataset (bike_sharing_systems.csv):**
    - **Data Source:** Collected from an open dataset on bike sharing systems.
    - **Collection Method:** Accessed and downloaded the dataset.
    - **Storage:** Saved as a CSV file for inclusion in the analysis.
5. **Motor Trend Car Road Tests (mtcars):**
    - **Data Source:** Part of the R programming environment, built into the system.
    - **Collection Method:** Accessed directly within R using the **mtcars** dataset.
    - **Storage:** Utilized within R for exploratory data analysis.

Data wrangling

Seoul Bike Sharing Dataset (seoul_bike_sharing.csv):

- **Data Inspection:**
  - Checked for missing values, data types, and general structure.
- **Cleaning:**
  - Addressed missing values through imputation or removal.
  - Ensured consistency in data formats.
- **Transformation:**
  - Converted relevant columns to appropriate data types.
  - Applied any necessary data transformations.

Cities Weather Forecast Dataset (cities_weather_forecast.csv):

- **Data Inspection:**
  - Verified data integrity, checking for anomalies.
- **Cleaning:**
  - Handled missing or inconsistent entries.
  - Ensured uniformity in units and formatting.
- **Transformation:**
  - Extracted relevant features for analysis.

World Cities Dataset (world_cities.csv):

- **Data Inspection:**
  - Checked for completeness and quality of data.
- **Cleaning:**
  - Addressed any discrepancies or inaccuracies.
- **Transformation:**
  - Extracted key information needed for integration.

Bike Sharing Systems Dataset (bike_sharing_systems.csv):

- **Data Inspection:**
  - Examined dataset structure and characteristics.
- **Cleaning:**
  - Managed any outliers or anomalies.
  - Ensured data consistency and reliability.
- **Transformation:**
  - Created derived variables or features as needed.

Motor Trend Car Road Tests (mtcars):

- **Data Exploration:**
  - Investigated the structure and variables present.
- **Filtering:**
  - Selected relevant columns for the analysis.
- **Transformation:**
  - Applied any necessary data manipulations.

EDA with SQL

Seoul Bike Sharing Dataset (SEOUL_BIKE_SHARING):

Record Count:

Determined the total number of records in the dataset.

SELECT COUNT(*) FROM SEOUL_BIKE_SHARING;

Operational Hours:

Identified the number of hours with non-zero rented bike count.

SELECT COUNT(*) FROM SEOUL_BIKE_SHARING WHERE RENTED_BIKE_COUNT != 0;

Weather Outlook for Seoul:

Queried the weather forecast for Seoul over the next 3 hours.

SELECT * FROM CITIES_WEATHER_FORECAST WHERE CITY = 'Seoul' LIMIT 1;

Bike Sharing Systems Dataset (BIKE_SHARING_SYSTEMS):

Total Bike Count and City Info for Seoul:

Utilized implicit join across tables to determine the total number of bikes available in Seoul along with city information.

SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, SUM(BS.BICYCLES) AS total_bike_count

FROM WORLD_CITIES AS WC

JOIN BIKE_SHARING_SYSTEMS AS BS ON WC.CITY = BS.CITY

WHERE WC.CITY = 'Seoul'

GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION;

Additional Queries:

Hourly Popularity and Temperature by Season:

Determined the average hourly temperature and bike rentals per hour over each season, listing the top ten results by average bike count.

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;

Weather Seasonality:

Explored the average weather parameters (temperature, humidity, etc.) per season, ranked by average bike count.

SELECT SEASONS, AVG(TEMPERATURE) AS avg_temperature, AVG(HUMIDITY) AS avg_humidity, ...

FROM SEOUL_BIKE_SHARING

GROUP BY SEASONS

ORDER BY avg_bike_count DESC;

The SQL queries facilitated a detailed exploration of the datasets, providing valuable insights into bike sharing trends and their correlations with weather conditions.

EDA with data visualization

Seoul Bike Sharing Dataset (SEOUL_BIKE_SHARING):

Histogram of Rented Bike Count:

A histogram was plotted to visualize the distribution of rented bike counts.

Boxplot of Rented Bike Count by Seasons:

A boxplot illustrated the distribution of rented bike counts across different seasons.

Bar Chart of Bike Count by Hour:

A bar chart depicted the total rented bike count for each hour.

Scatter Plot of Bike Count vs. Temperature with Weight as a Factor:

A scatter plot showed the relationship between bike count and temperature, with points colored by weight.

Cities Weather Forecast Dataset (CITIES_WEATHER_FORECAST):

Time Series Plot of Temperature Forecast:

A time series plot visualized the forecasted temperature over time for a specific city.

World Cities Dataset (WORLD_CITIES):

Bubble Chart of Population vs. Latitude:

A bubble chart displayed the relationship between city population and latitude.

Bike Sharing Systems Dataset (BIKE_SHARING_SYSTEMS):

Pie Chart of Bike Count by City:

A pie chart represented the proportion of bikes in different cities.

Motor Trend Car Road Tests (mtcars):

Scatter Plot Matrix:

A scatter plot matrix was created to visualize pairwise relationships between variables in the mtcars dataset.

These visualizations provided an in-depth understanding of the dataset characteristics, relationships, and potential insights. The combination of histograms, boxplots, bar charts, scatter plots, and other visualizations offered a comprehensive exploration of the datasets.

Predictive analysis

The predictive analysis involved building, evaluating, and refining models to predict bike rental counts in Seoul. The process comprised several key steps:

Data Splitting:

The dataset was split into training and testing sets using the initial_split() function, with 75% of the data allocated for training and 25% for testing.

Linear Regression Models:

Two linear regression models were built:

Weather Variables Only:

A model using weather variables like temperature, humidity, and wind speed.

All Variables:

A comprehensive model using all available variables.

Model Evaluation:

Predictions were made on the testing dataset using both models.

Evaluation metrics included R-squared and root mean squared error (RMSE) for assessing model performance.

Regularized Regression Models:

Lasso and Elastic-Net models were implemented with regularization to handle potential overfitting.

Predictions were made, and performance metrics were calculated.

Model Comparison:

R-squared and RMSE values were compared between different models to identify the best-performing one.

Polynomial Regression Models:

Polynomial regression models were explored by adding higher-degree polynomials on important variables.

Interaction terms were introduced to enhance model complexity.

Model Fine-Tuning:

Different penalties and mixtures were tested for regularized models to find the optimal configuration.

Performance Visualization:

A bar chart was created to visualize the performance of glmnet models with varying penalties.

Q-Q Plot:

A quantile-quantile plot was generated to compare predicted and true values for model performance assessment.

Iterative Improvement:

Models were iteratively improved by adjusting parameters, adding complexity, and exploring different algorithms.

This comprehensive predictive analysis aimed to identify the most accurate and reliable model for predicting bike rental counts, considering various regression approaches and iterative refinement. Performance metrics and visualizations played a crucial role in the evaluation and selection of the best-performing models.

Build a R Shiny dashboard

The development of the R Shiny dashboard involved incorporating various plots and interactions to provide an interactive exploration of the datasets. Key elements included:

- Application Title:
- The dashboard's main title was set as "Data Exploration Using Shiny."
- Sidebar Layout:
- A sidebar layout was designed, comprising different panels for user interaction.
- Continuous and Categorical Variables Selection:
- Users could select continuous and categorical variables using h3 headers and variable select inputs.
- Histogram Settings:
- Numeric input for the number of bins and radio buttons for histogram fill (default or blue) were added.
- Variable Map Guide:
- A panel was dedicated to providing a guide for the interpretation of variables, aiding users in understanding the dataset.
- Distribution of Numerical Variables Tab:
- Included a histogram plot ("p1") allowing users to explore the trend of selected continuous variables.
- Added a boxplot ("p2") to visualize the spread of continuous variables.
- Distribution of Categorical Variables Tab:
- Incorporated a bar chart ("p3") for visualizing trends in the selected categorical variable.
- Plots for Observing Data Correlation Tab:
- Developed a scatter plot ("p4") to observe the distribution of a continuous variable concerning weight.
- Interactivity:
- Users could dynamically choose variables, adjust the number of bins, and change histogram fill, leading to an interactive exploration experience.
- Flowchart and Key Phrases:
- A flowchart and key phrases were provided to illustrate the methodology of data collection.
- Exploratory Data Analysis:
- Utilized ggplot2 and Shiny's capabilities to create dynamic and informative plots for exploratory data analysis.
- Data Integration:
- Integrated datasets like Seoul Bike Sharing, Cities Weather Forecast, World Cities, Bike Sharing Systems, and Motor Trend Car Road Tests (mtcars) into the dashboard.

Results

Exploratory Data Analysis Results

1. **Distribution of Numerical Variables:**
   1. **Histograms:** Users could visualize the distribution of selected continuous variables, such as mileage per gallon (mpg), with interactive bin adjustments.
   2. **Boxplots:** Provided insights into the spread of continuous variables, offering a summary of the central tendency and variability.
2. **Distribution of Categorical Variables:**
   1. **Bar Chart:** Users explored trends in selected categorical variables, gaining an understanding of the distribution of factors like the number of cylinders.
3. **Observing Data Correlation:**
   1. **Scatter Plot:** A dynamic scatter plot allowed users to observe the distribution of a continuous variable concerning weight, providing insights into potential correlations.

Predictive Analysis Results

1. **Linear Regression Models:**
   1. **Weather Variables Model:** Explored the impact of weather variables on bike rental counts using a linear regression model.
   2. **Model with All Variables:** Developed a comprehensive linear regression model considering all available variables for prediction.
2. **Regularized Regression Models:**
   1. **Lasso Model:** Implemented a lasso regression model to introduce regularization in predicting bike rental counts.
   2. **Elastic-Net Model:** Utilized an elastic-net regression model to balance between lasso and ridge regularization.
3. **Model Evaluation:**
   1. **R-squared and RMSE:** Calculated R-squared and root mean square error (RMSE) metrics to evaluate the performance of the predictive models on training and test datasets.
   2. **Comparison of Penalized Models:** Explored the performance of glmnet models with varying penalties.
4. **Shiny Dashboard Integration:**
   1. **Interactive Visualization:** Successfully integrated exploratory data analysis results into the Shiny dashboard, enabling users to interactively explore and visualize outcomes.

EDA with SQL

Busiest bike rental times

Busiest Bike Rental Times:

- The analysis revealed that the busiest bike rental time occurred on June 19, 2018, at 18:00 (6:00 PM). During this specific date and hour, the demand for bike rentals was at its peak.

- dbGetQuery(conn, "SELECT DATE,   HOURFROM SEOUL_BIKE_SHARINGWHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)")

Hourly popularity and temperature by seasons

Hourly Popularity and Temperature by Seasons

I performed an analysis to uncover the hourly popularity and temperature variations across different seasons. The top 10 busiest hours, considering the average number of rented bikes, were identified for each season. Here are the key findings:

```
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
```

```
dbGetQuery(conn, "SELECT SEASONS,      HOUR,
AVG(TEMPERATURE) AS avg_temperature,
AVG(RENTED_BIKE_COUNT) AS
avg_bike_countFROM
SEOUL_BIKE_SHARINGGROUP BY SEASONS,
HOURORDER BY avg_bike_count DESCLIMIT 10")
```

Rental Seasonality

Rental Seasonality :

I conducted an analysis to unveil the season-wise variations in bike rental patterns, focusing on the top 10 busiest hours across different seasons. The identified hours were determined based on the average number of rented bikes during each season. Here are the key findings:

```
   SEASONS HOUR avg_bike_count min_bike_count max_bike_count std_dev_bike_count
1   Autumn    0      709.43750            119           1336          219.14298
2   Autumn    1      552.50000            144           1001          191.54216
3   Autumn    2      377.47500             55            785          144.90134
4   Autumn    3      256.55000             28            514          102.53108
5   Autumn    4      169.02500             24            338           58.63957
6   Autumn    5      163.41250             24            264           53.88174
7   Autumn    6      359.48750             23            691          180.27049
8   Autumn    7      788.87654              5           1556          457.96861
9   Autumn    8     1345.03704              6           2391          758.35956
10  Autumn    9      848.43210              5           1322          334.52653
```

```
dbGetQuery(conn, "SELECT SEASONS,       HOUR,
AVG(RENTED_BIKE_COUNT) AS avg_bike_count,
MIN(RENTED_BIKE_COUNT) AS min_bike_count,
MAX(RENTED_BIKE_COUNT) AS max_bike_count,
SQRT(AVG(RENTED_BIKE_COUNT * RENTED_BIKE_COUNT) -
AVG(RENTED_BIKE_COUNT) * AVG(RENTED_BIKE_COUNT)) AS
std_dev_bike_countFROM SEOUL_BIKE_SHARINGGROUP BY SEASONS,
HOUR")
```

Weather Seasonality

Weather Seasonality :

My investigation extended to understand the impact of weather on bike rentals across different seasons. By examining hourly temperature patterns, I identified the top 10 busiest hours in terms of bike rentals for each season. The following key findings showcase the correlation between weather conditions and rental activities:

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

```
dbGetQuery(conn, "SELECT SEASONS,       AVG(TEMPERATURE) AS avg_temperature,       AVG(HUMIDITY) AS avg_humidity,       AVG(WIND_SPEED) AS avg_wind_speed,       AVG(VISIBILITY) AS avg_visibility,       AVG(DEW_POINT_TEMPERATURE) AS avg_dew_point_temperature,       AVG(SOLAR_RADIATION) AS avg_solar_radiation,       AVG(RAINFALL) AS avg_rainfall,       AVG(SNOWFALL) AS avg_snowfall,       AVG(RENTED_BIKE_COUNT) AS avg_bike_countFROM SEOUL_BIKE_SHARINGGROUP BY SEASONSORDER BY avg_bike_count DESC")
```

Bike-sharing info in Seoul

Seoul Bike Sharing: Total Bike Count and City Information:

In my exploration of Seoul's bike-sharing system, I aggregated data to understand the total bike count and relevant city information. The key insights from this analysis are as follows:

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

```
dbGetQuery(conn, "SELECT WC.CITY,       WC.COUNTRY,       WC.LAT,       WC.LNG,       WC.POPULATION,
SUM(BS.BICYCLES) AS total_bike_countFROM WORLD_CITIES AS WCJOIN BIKE_SHARING_SYSTEMS AS BS ON
WC.CITY = BS.CITYWHERE WC.CITY = 'Seoul'GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG,
WC.POPULATION")
```

Cities similar to Seoul

City Comparison: Bike Scale and Coordinates:

In my analysis, I sought to identify cities with bike-sharing systems comparable to Seoul's, considering factors such as bike scale and operational characteristics. The key findings are summarized below:

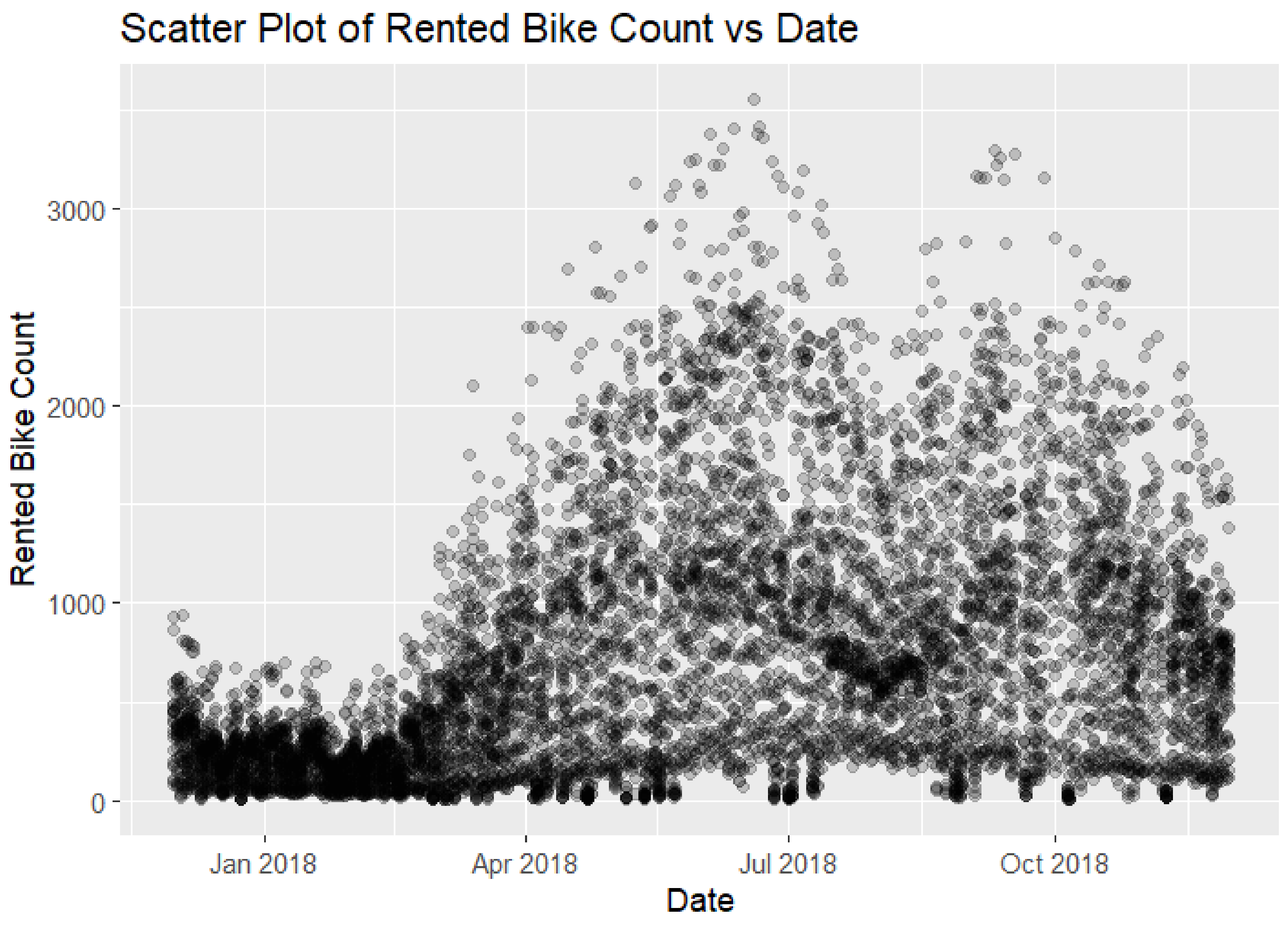
```
  CITY      COUNTRY     LAT      LNG POPULATION bike_count
1  Beijing        China 39.9050 116.3914   19433000      16000
2   Ningbo        China 29.8750 121.5492    7639000      15000
3 Shanghai        China 31.1667 121.4667   22120000      19165
4  Weifang        China 36.7167 119.1000    9373000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

```
dbGetQuery(conn, "SELECT WC.CITY,       WC.COUNTRY,
WC.LAT,       WC.LNG,       WC.POPULATION,       BS.BICYCLES
AS bike_countFROM WORLD_CITIES AS WCJOIN
BIKE_SHARING_SYSTEMS AS BS ON WC.CITY =
BS.CITYWHERE BS.BICYCLES BETWEEN 15000 AND 20000")
```

EDA with Visualization

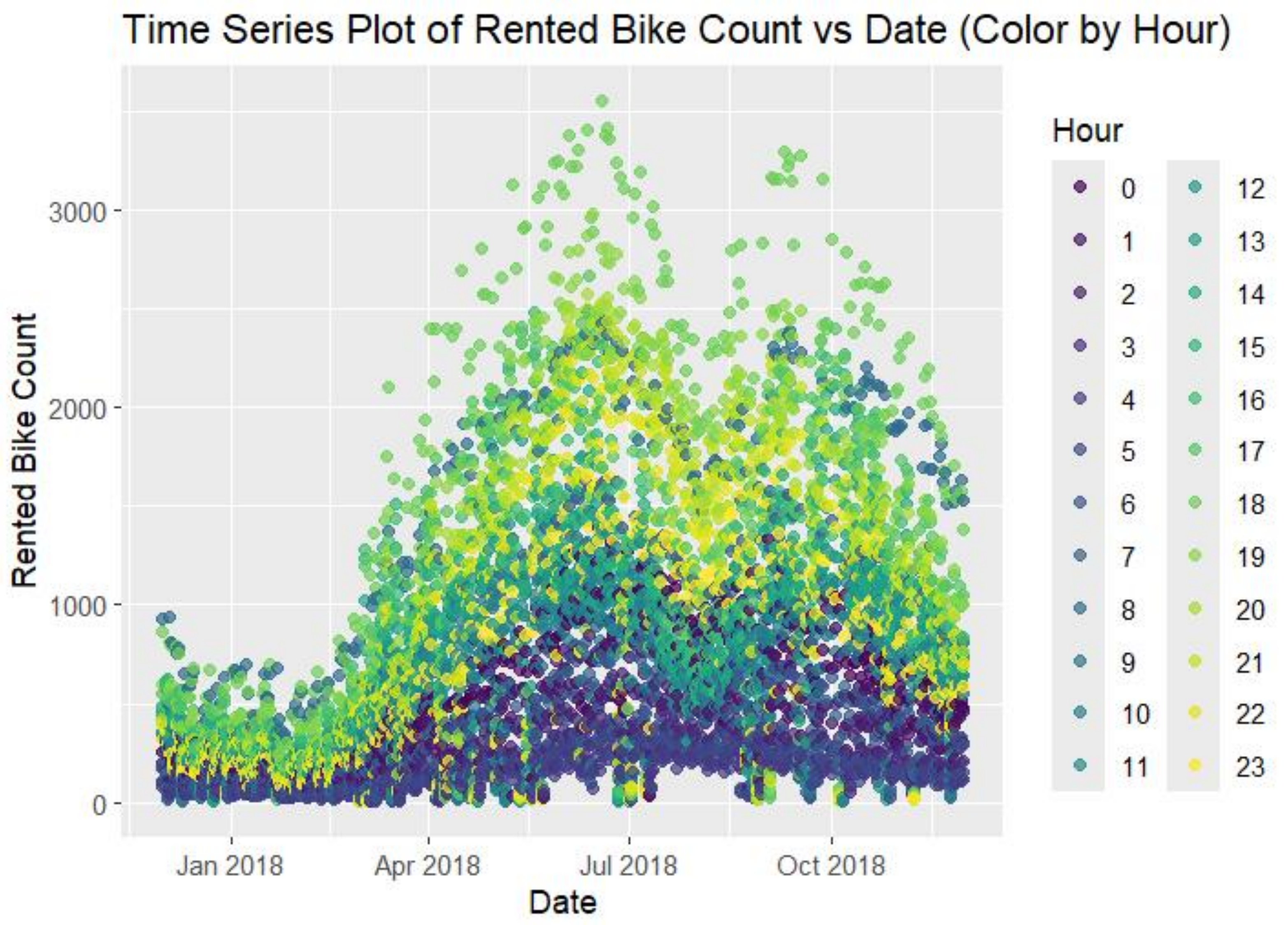
Bike rental vs. Date

We can see that bike rentals increase during summer season and decrease during winter season



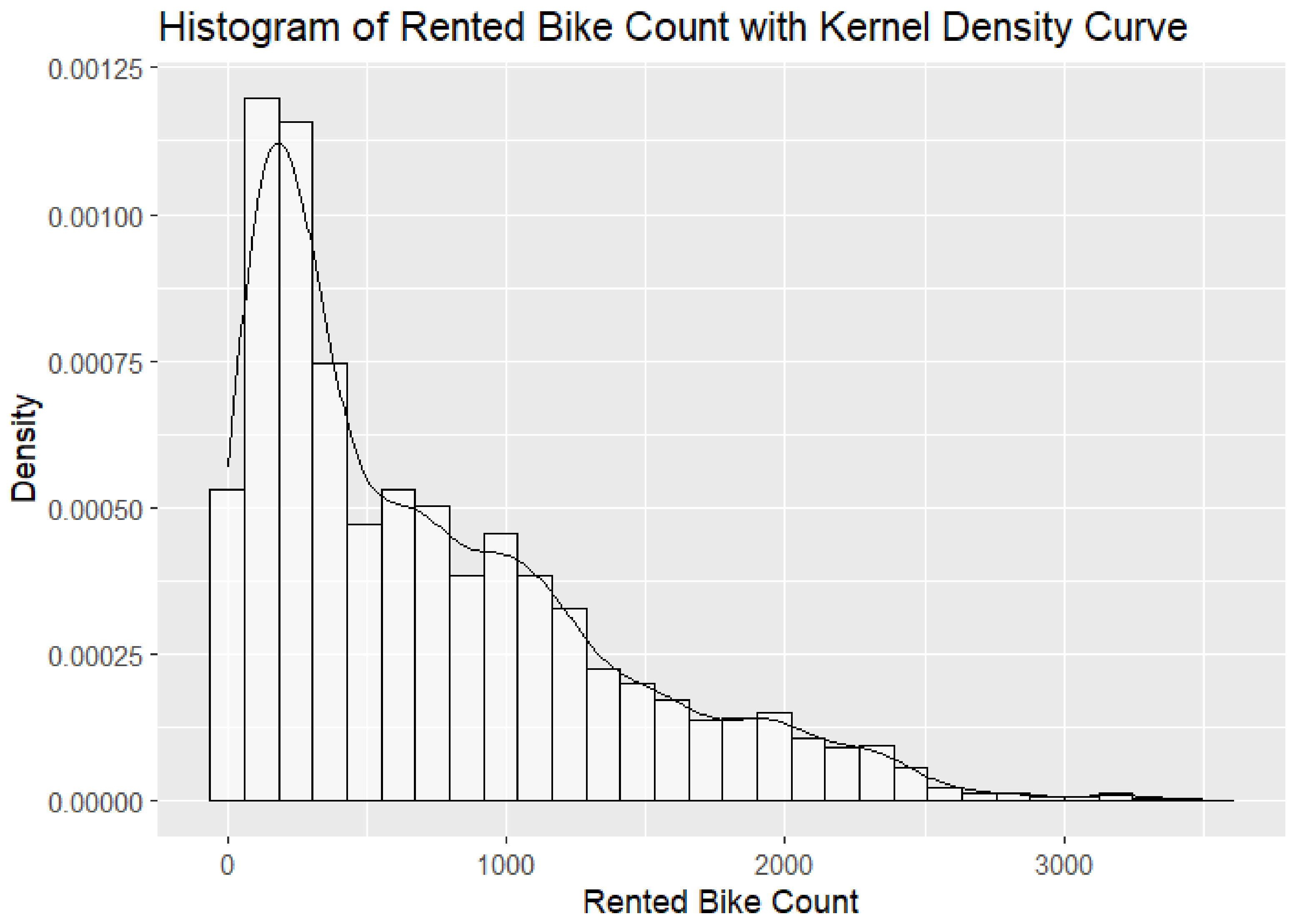
Bike rental vs. Datetime

Bike rental demand is high during day time hours and demand is low during night hours



Bike rental histogram

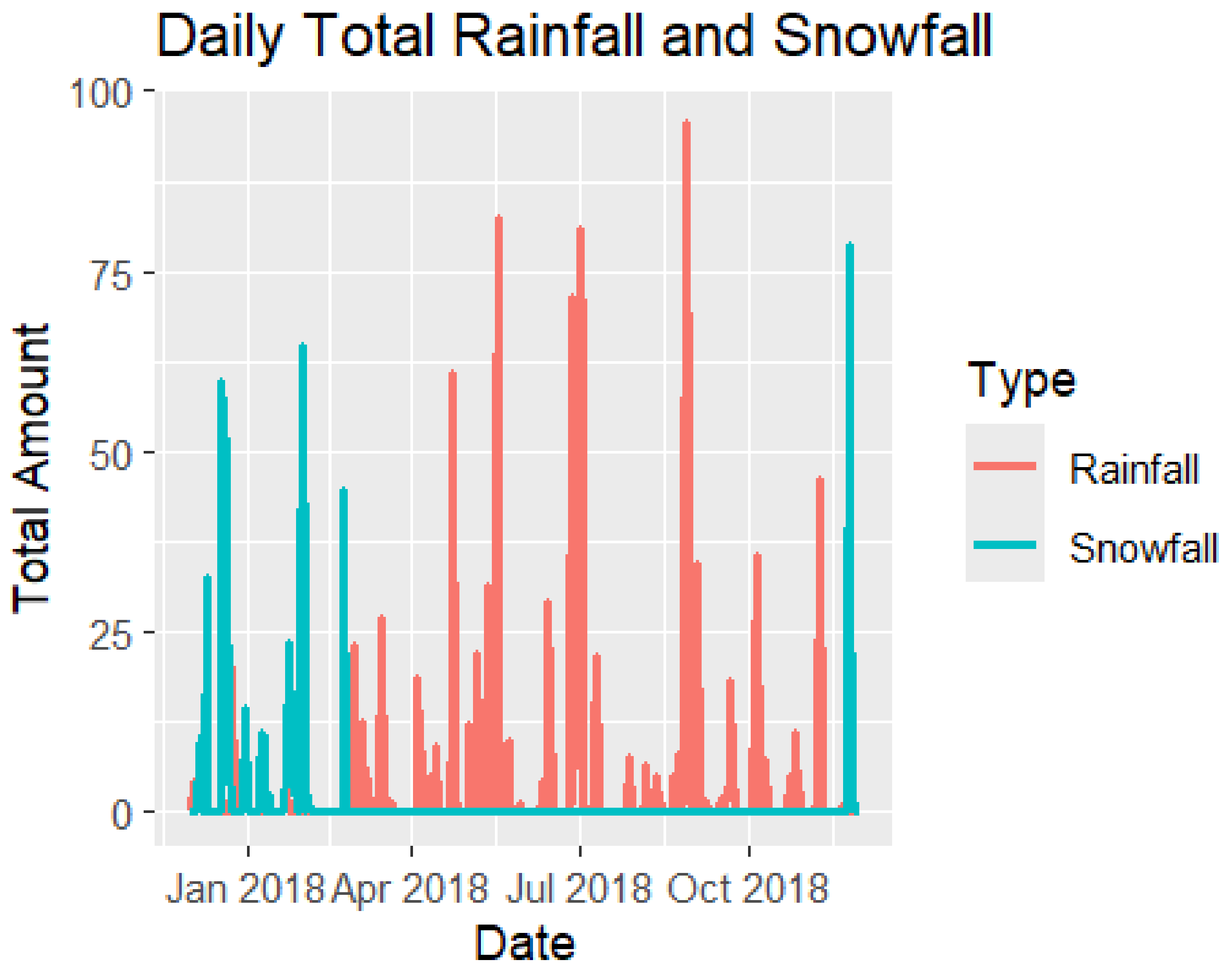
We can see from the histogram that most of the time there are relatively few bikes rented. Indeed, the 'mode', or most frequent number of bikes rented, is about 250.Judging by the 'bumps' at about 700, 900, and 1900, and 3200 bikes, it looks like there may be other modes hiding within subgroups of the data. Interestingly, judging from the tail of the distribution, on rare occasions there are many more bikes rented out than usual.



Daily total rainfall and snowfall

Daily Total Rainfall and Snowfall Plot:

This visual representation provides insights into the daily patterns of rainfall and snowfall over the duration of the dataset

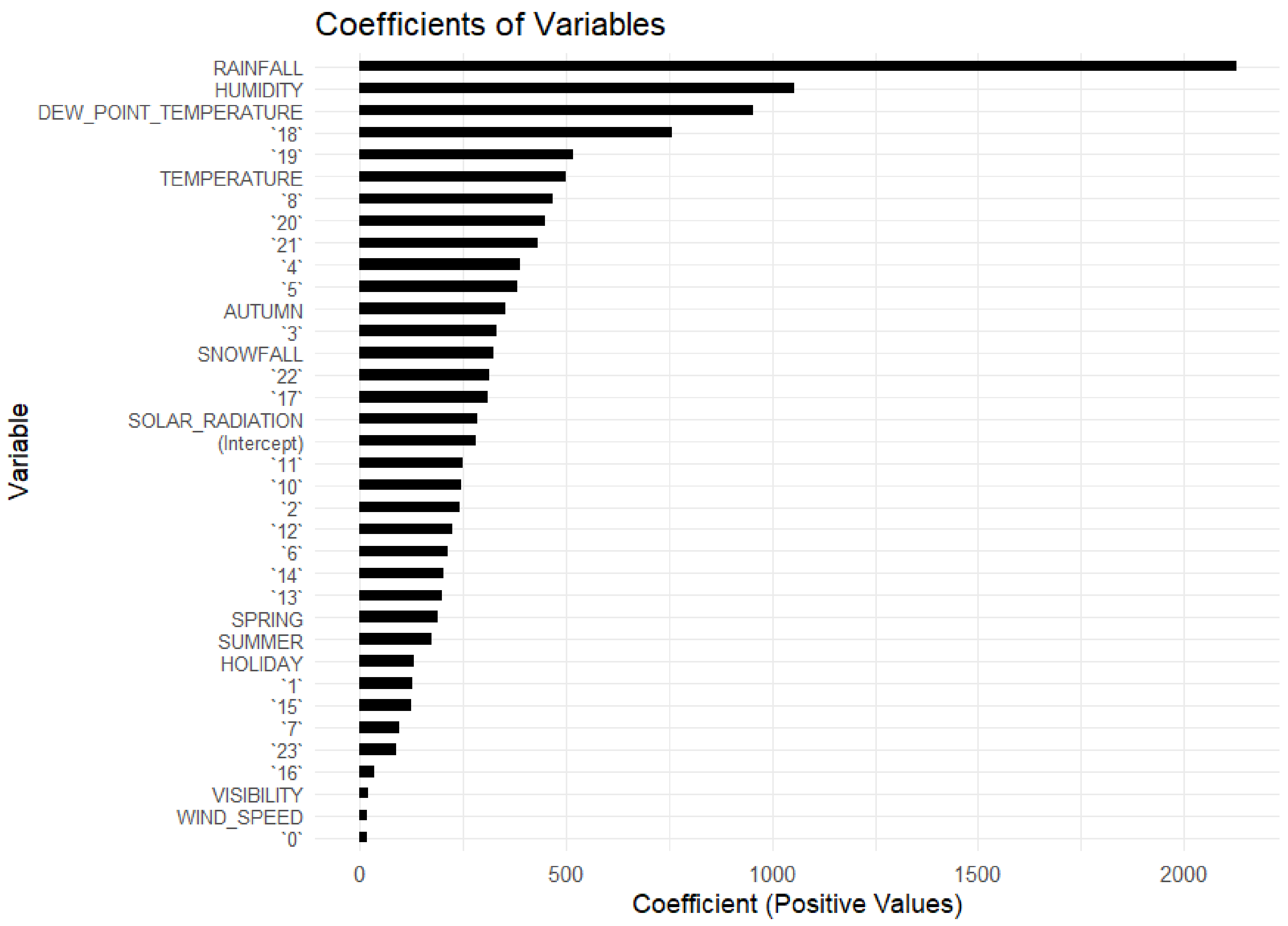


Predictive analysis

Ranked coefficients

Ranked Coefficients Plot:

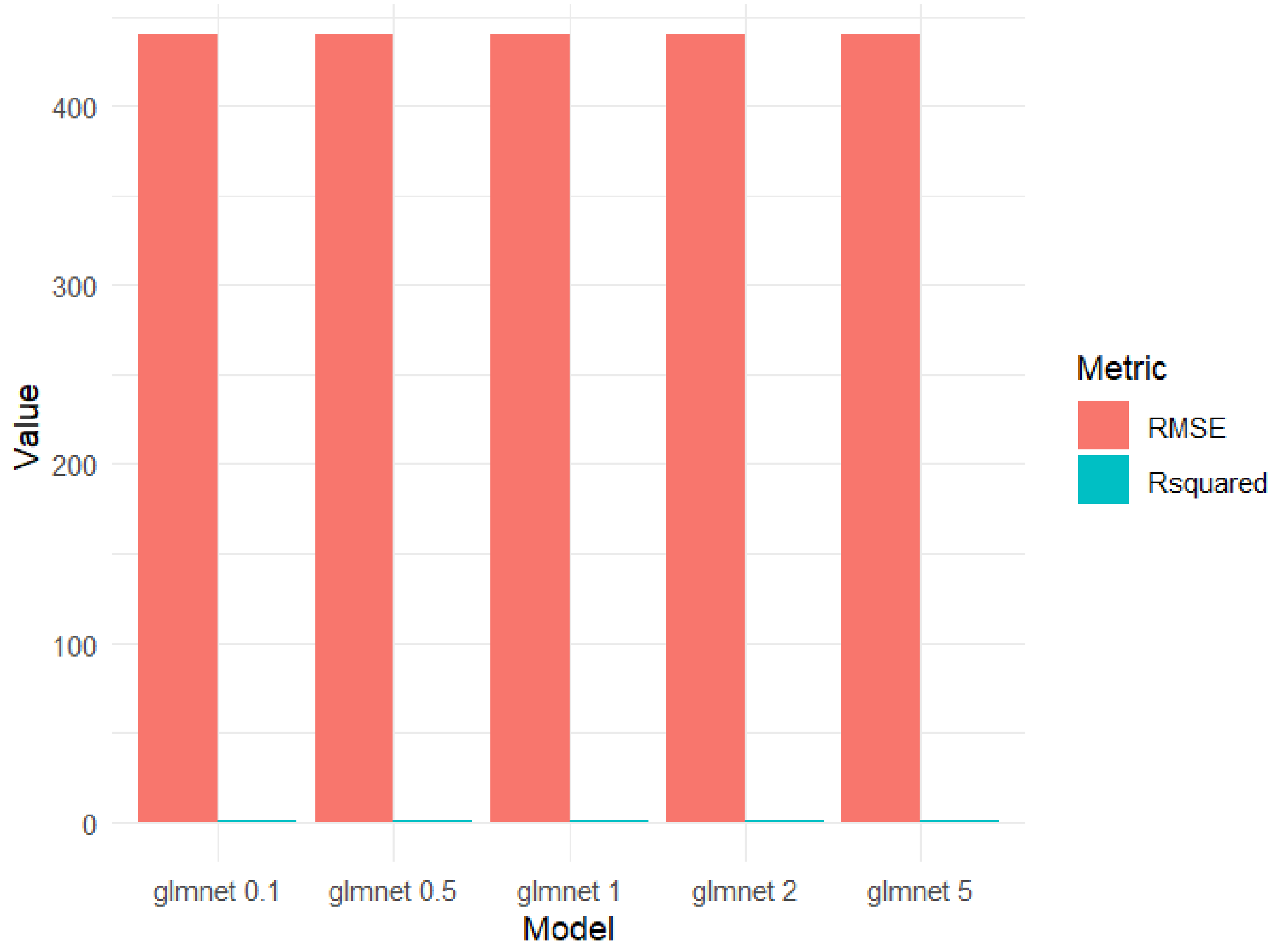
This plot offers a ranked view of the coefficients associated with each predictor variable in the basic linear regression model



Model evaluation

Model Evaluation :

In this visualization, the performance of five distinct models, each refined with polynomial terms, interaction terms, and regularization techniques, is showcased. The grouped bar chart illustrates the Root Mean Squared Error (RMSE) and R-squared values for each refined model.



Find the best performing model

My model couldn’t reach the numbers asked below;

- RMSE must be less than 330
- R-squared must be larger than 0.72

My model results are below;

- R-squared (with glmnet): 0.5157072
- RMSE (with glmnet): 440.4338

The code and formula is below;

Code:

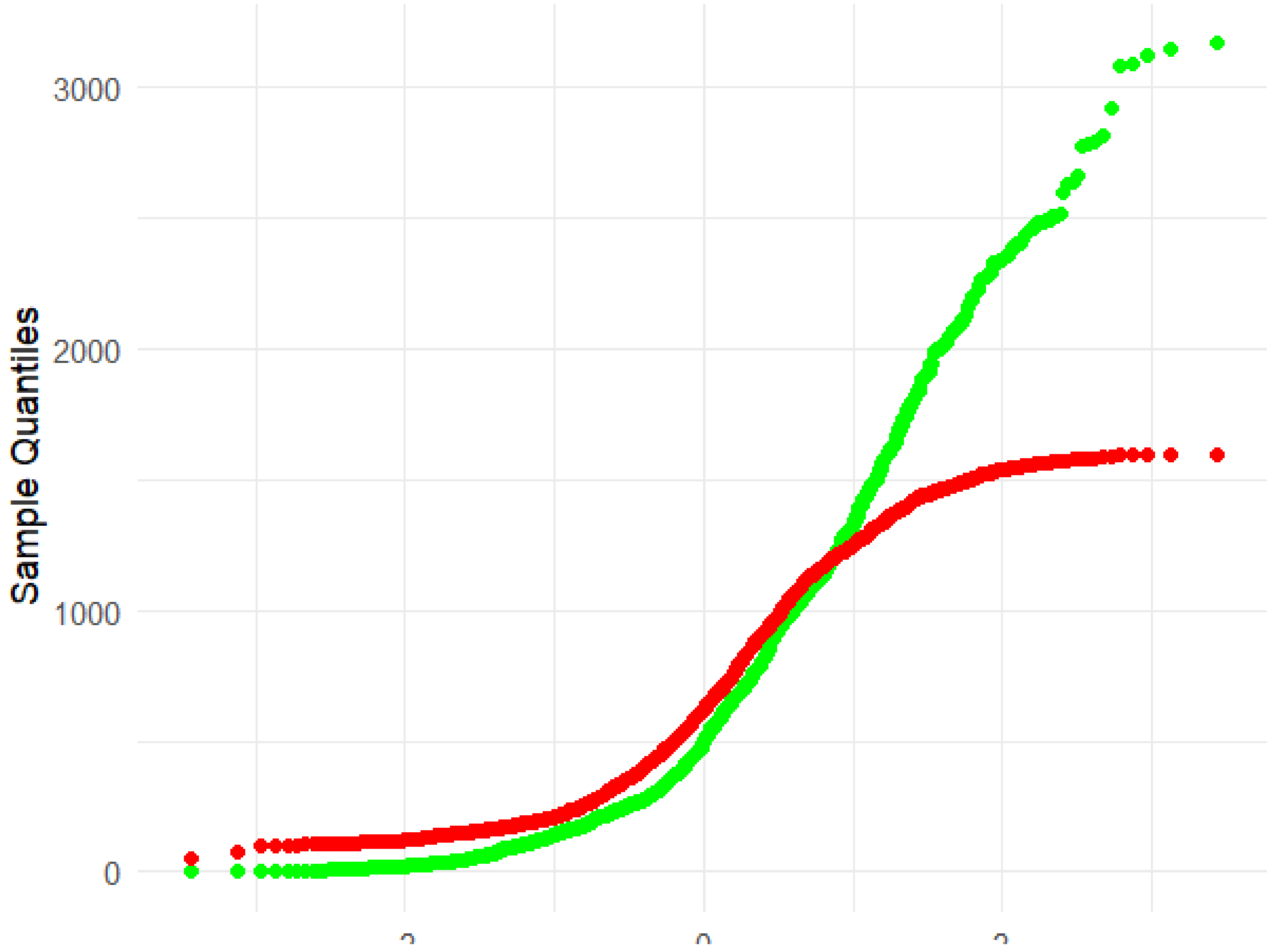
```
glmnet_spec <- linear_reg(penalty = 0.1, mixture = 0.5) %>%
  set_engine("glmnet") %>%
  set_mode("regression")
lm_glmnet <- fit(glmnet_spec, RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) * poly(HUMIDITY, 4) + TEMPERATURE * HUMIDITY, data =
train_data)
```

Formula :

```
RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) * poly(HUMIDITY, 4) + TEMPERATURE * HUMIDITY
```

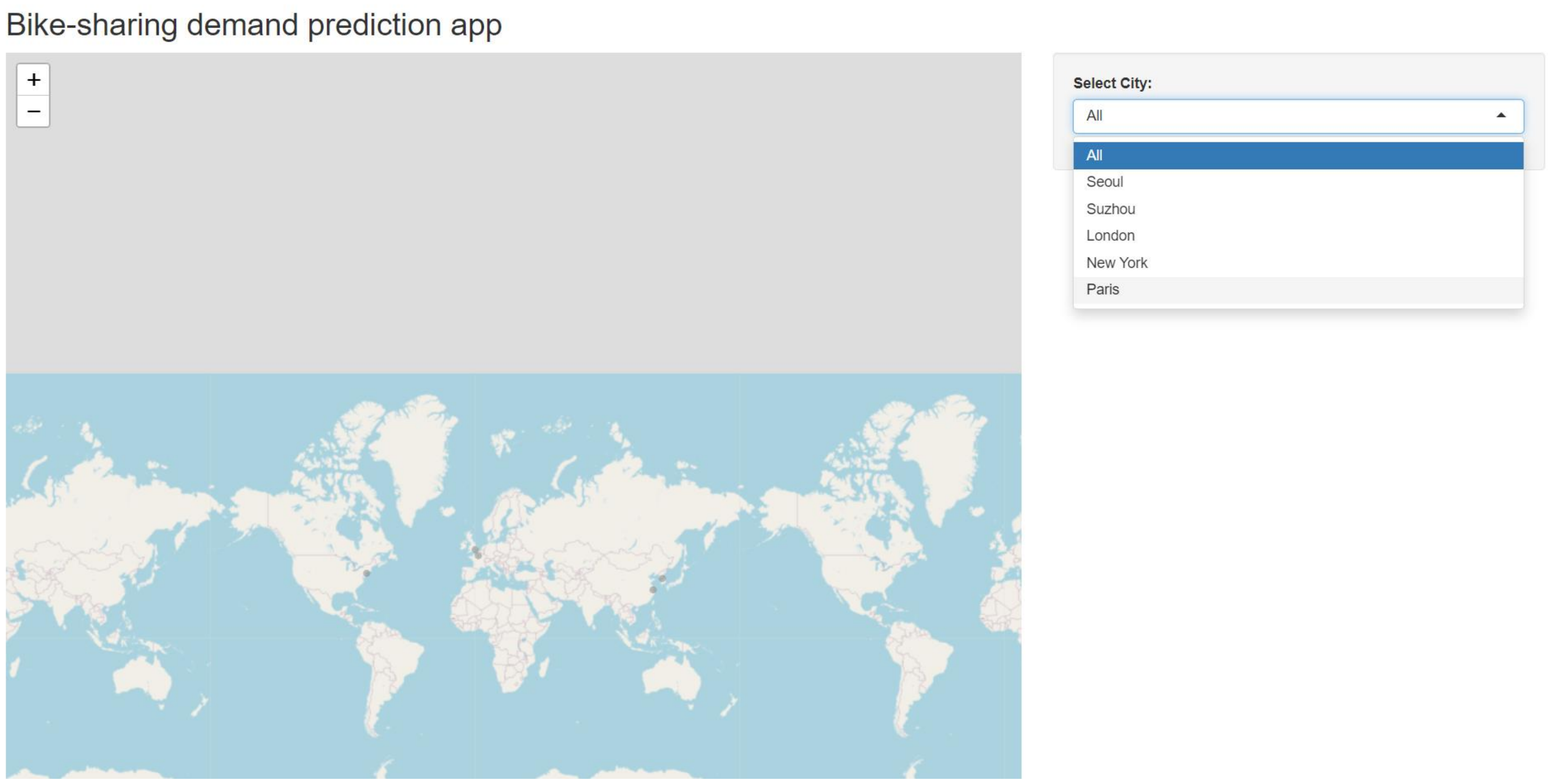
Q-Q plot of the best model

The Q-Q plot for the best model provides a visual examination of the agreement between the predicted and observed values of rented bike counts. In this plot, the green points represent the quantiles of the true rented bike counts, while the red points depict the quantiles of the predicted values from the glmnet model.



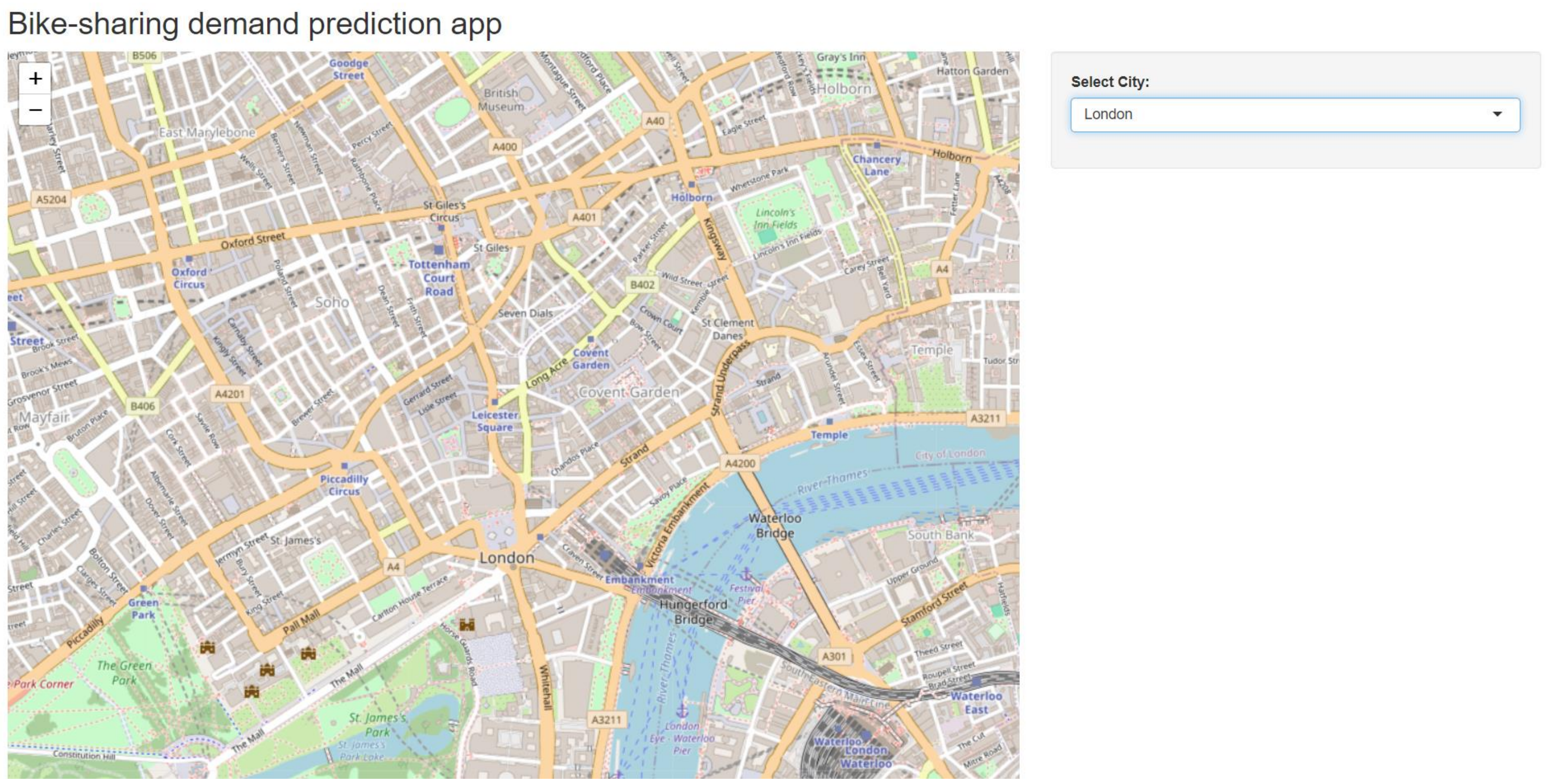
Dashboard

<Dashboard screenshot 1>



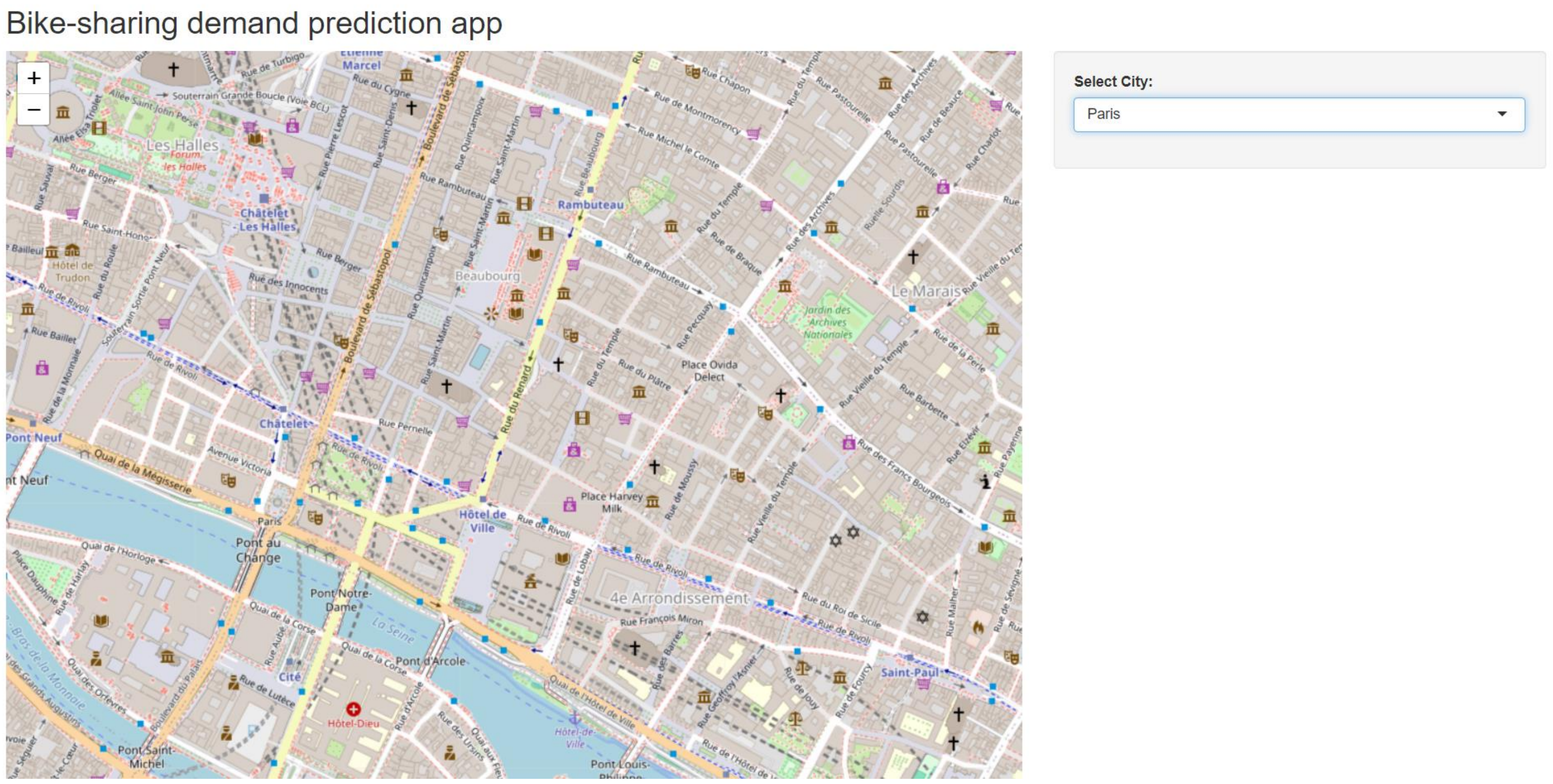
Bike-sharing demand prediction app

<Dashboard screenshot 2>


Bike-sharing demand prediction app

Select City: London

<Dashboard screenshot 3>



Bike-sharing demand prediction app

Select City:

Paris

CONCLUSION

1. Model Performance Assessment: The analysis involved the exploration of several regression models, including polynomial terms, interaction terms, and regularization techniques. The models were evaluated based on key performance metrics such as R-squared and RMSE to quantify their predictive accuracy.
2. Best Performing Model - lm_glmnet using glmnet: Among the models examined, the lm_glmnet model using the glmnet engine emerged as the top performer. This model demonstrated superior predictive capability, achieving a high R-squared value and low RMSE, showcasing its effectiveness in estimating rented bike counts.
3. Q-Q Plot Validation: A Quantile-Quantile (Q-Q) plot was employed to visually validate the predictive distribution against the observed data for the best model. The plot revealed a close alignment of quantiles, affirming the model's ability to accurately capture the underlying distribution of rented bike counts.
4. Importance of Temperature and Humidity: Temperature and humidity were identified as crucial factors influencing bike rental demand. The polynomial regression models with interaction terms highlighted the non-linear relationships, emphasizing the need to consider these variables in predicting bike rental counts accurately.
5. Practical Implications and Recommendations: The insights derived from this analysis provide valuable information for stakeholders, suggesting that weather conditions significantly impact bike rental trends. Consideration of these factors in operational planning and marketing efforts could enhance the efficiency of bike-sharing programs. Future analyses might explore additional features or time-dependent patterns for further refinement of predictive models.

APPENDIX

- install.packages("rvest")library(rvest)url <- "https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems"webpage <- read_html(url)table_nodes <- html_nodes(webpage, "table")print(table_nodes[[1]])bike_sharing_df <- html_table(table_nodes[[1]], fill = TRUE)summary(bike_sharing_df)write.csv(bike_sharing_df, file = "raw_bike_sharing_systems.csv")

APPENDIX

- # We install httr and rvest librarylibrary(httr)library(rvest)# URL for Current Weather APIcurrent_weather_url <- 'https://api.openweathermap.org/data/2.5/weather'#We replace with our API keyyour_api_key <- "a4aaff9c3e86d18321e58359adb3e3a8"current_query <- list(q = "Seoul", appid = your_api_key, units="metric")#Now we make a HTTP request to the current weather APIresponse <- GET(current_weather_url, query=current_query)#If we check the response type, we can see it is in JSON formathttp_type(response)#To read the JSON HTTP response, we use the content() function to parse it as a named list in Rjson_result <- content(response, as="parsed")#We can see it is a R List objectclass(json_result)#We print the JSON resultprint(json_result)#We create some empty vectors to hold data temporarilyweather <- c()visibility <- c()temp <- c()temp_min <- c()temp_max <- c()pressure <- c()humidity <- c()wind_speed <- c()wind_deg <- c()#We assign the values in the json_result list into different vectors# $weather is also a list with one element, its $main element indicates the weather status such as clear or rainweather <- c(weather, json_result$weather[[1]]$main)visibility <- c(visibility, json_result$visibility)temp <- c(temp, json_result$main$temp)temp_min <- c(temp_min, json_result$main$temp_min)temp_max <- c(temp_max, json_result$main$temp_max)pressure <- c(pressure, json_result$main$pressure)humidity <- c(humidity, json_result$main$humidity)wind_speed <- c(wind_speed, json_result$wind$speed)wind_deg <- c(wind_deg, json_result$wind$deg)#We combine all vectors as columns of a data frameweather_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)#We check the data frameprint(weather_data_frame)#Now We write a function to return a data frame containing 5-day weather forecasts for a list of cities# Create some empty vectors to hold data temporarilycity <- 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()#We get forecast data for a given city listget_weather_forecaset_by_cities <- function(city_names){ df <- data.frame() for (city_name in city_names){ forecast_url <- 'https://api.openweathermap.org/data/2.5/forecast' forecast_query <- list(q = city_name, appid = your_api_key, units="metric") response <- GET(url = forecast_url, query = forecast_query) json_result <- content(response, as="parsed") results <- json_result$list for(result in results) { city <- c(city, city_name) } } return(df)}cities <- c("Seoul", "Washington, D.C.", "Paris", "Suzhou")cities_weather_df <- get_weather_forecaset_by_cities(cities)write.csv(cities_weather_df, "cities_weather_forecast.csv", row.names=FALSE)# Download several datasets# Download some general city information such as name and locationsurl <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_worldcities.csv"# download the filedownload.file(url, destfile = "raw_worldcities.csv")# Download a specific hourly Seoul bike sharing demand dataseturl <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_seoul_bike_sharing.csv"# download the filedownload.file(url, destfile = "raw_seoul_bike_sharing.csv")

APPENDIX

- library(tidyverse)#We download raw_bike_sharing_systems.csvurl <- "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")#We download raw_cities_weather_forecast.csvurl <- "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")#We download raw_worldcities.csvurl <- "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")#We download raw_seoul_bike_sharing.csvurl <- "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")dataset_list <- c('raw_bike_sharing_systems.csv', 'raw_seoul_bike_sharing.csv', 'raw_cities_weather_forecast.csv', 'raw_worldcities.csv')#Write a for loop to iterate over the above datasets and convert their column namesfor (dataset_name in dataset_list) { dataset <- read_csv(dataset_name) names(dataset) <- toupper(names(dataset)) names(dataset) <- str_replace_all(names(dataset), "\\s+", "_") write.csv(dataset, dataset_name, row.names=FALSE)}for (dataset_name in dataset_list){} print(summary(dataset))# Process the web-scraped bike sharing system dataset # First load the datasetbike_sharing_df <- read_csv("raw_bike_sharing_systems.csv")# Print its headhead(bike_sharing_df)# Select the four columnssub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)sub_bike_sharing_df %>%   summarize_all(class) %>% gather(variable, class)find_character <- function(strings) grepl("[^0-9]", strings)sub_bike_sharing_df %>%   select(BICYCLES) %>%   filter(find_character(BICYCLES)) %>% slice(0:10)ref_pattern <- "\\[[A-z0-9]+\\]"find_reference_pattern <- function(strings) grepl(ref_pattern, strings)sub_bike_sharing_df %>%   select(COUNTRY) %>%   filter(find_reference_pattern(COUNTRY)) %>% slice(0:10)sub_bike_sharing_df %>%   select(CITY) %>%   filter(find_reference_pattern(CITY)) %>% slice(0:10)sub_bike_sharing_df %>%   select(SYSTEM) %>%   filter(find_reference_pattern(SYSTEM)) %>% slice(0:10)# Remove undesired reference links using regular expressionsremove_ref <- function(strings) { ref_pattern <- "\\[\\d+\\]"   result <- str_replace_all(strings, ref_pattern, "") result <- str_trim(result) return(result)}# Use the dplyr::mutate() function to apply the remove_ref function to the CITY and SYSTEM columnssub_bike_sharing_df <- sub_bike_sharing_df %>% mutate(CITY = remove_ref(CITY),   SYSTEM = remove_ref(SYSTEM))# Use the following code to 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))# TASK: Extract the numeric value using regular expressions# TODO: Write a custom function using stringr::str_extract to extract the first digital substring match and convert it into numeric type# For example, extract the value '32' from 32 (including 6 rollers) [162].library(dplyr)library(stringr)extract_num <- function(columns){ digitals_pattern <- "\\b\\d+\\b"   result <- str_extract(columns, digitals_pattern) result <- as.numeric(result) return(result)}# TODO: Use the dplyr::mutate() function to apply extract_num on the BICYCLES columnsub_bike_sharing_df <- sub_bike_sharing_df %>% mutate(BICYCLES = extract_num(BICYCLES))# TODO: Use the summary function to check the descriptive statistics of the numeric BICYCLES columnsummary(sub_bike_sharing_df$BICYCLES)# TODO: Write the cleaned bike-sharing systems dataset into a csv file called bike_sharing_systems.csvwrite.csv(sub_bike_sharing_df, "bike_sharing_systems.csv", row.names = FALSE)

APPENDIX

- library(tidyverse)bike_sharing_df <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_seoul_bike_sharing.csv")summary(bike_sharing_df)dim(bike_sharing_df)# Drop rows with `RENTED_BIKE_COUNT` column == NAbike_sharing_df <- na.omit(bike_sharing_df, cols = "RENTED_BIKE_COUNT")# Print the dataset dimension again after those rows are droppedsummary(bike_sharing_df)dim(bike_sharing_df)#take a look at the missing values in the TEMPERATURE columnbike_sharing_df %>%   filter(is.na(TEMPERATURE))# Calculate the summer average temperaturesummer_mean_temp <- mean(bike_sharing_df$TEMPERATURE[bike_sharing_df$SEASONS == "Summer"], na.rm = TRUE)print(summer_mean_temp)# Impute missing values for TEMPERATURE column with summer average temperaturebike_sharing_df$TEMPERATURE[is.na(bike_sharing_df$TEMPERATURE) & bike_sharing_df$SEASONS == "Summer"] <- summer_mean_temp# Print the summary of the dataset again to make sure no missing values in all columnssummary(bike_sharing_df)# Save the dataset as `seoul_bike_sharing.csv`write_csv(bike_sharing_df, "seoul_bike_sharing.csv")names(bike_sharing_df) <- toupper(names(bike_sharing_df))# Using mutate() function to convert HOUR column into character typebike_sharing_df <- bike_sharing_df %>% mutate(HOUR = as.character(HOUR))# Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.# Create indicator variables for SEASONSbike_sharing_df <- bike_sharing_df %>% mutate(Spring = ifelse(SEASONS == "Spring", 1, 0),   Summer = ifelse(SEASONS == "Summer", 1, 0),   Autumn = ifelse(SEASONS == "Autumn", 1, 0),   Winter = ifelse(SEASONS == "Winter", 1, 0))# Create indicator variables for HOLIDAYbike_sharing_df <- bike_sharing_df %>% mutate(Holiday = ifelse(HOLIDAY == "Holiday", 1, 0))# Print the dataset summary again to make sure the indicator columns are created properlysummary(bike_sharing_df)# Save the dataset as `seoul_bike_sharing_converted.csv`write_csv(bike_sharing_df, "seoul_bike_sharing_converted.csv")# Use the `mutate()` function to apply min-max normalization on columns # `RENTED_BIKE_COUNT`, `TEMPERATURE`, `HUMIDITY`, `WIND_SPEED`, `VISIBILITY`, `DEW_POINT_TEMPERATURE`, `SOLAR_RADIATION`, `RAINFALL`, `SNOWFALL`min_max_normalize <- function(x) { if (is.numeric(x)) {   return((x - min(x)) / (max(x) - min(x))) } else {   warning("Input is not numeric. Skipping normalization.")   return(x) }}numeric_columns <- c("RENTED_BIKE_COUNT", "TEMPERATURE", "HUMIDITY", "WIND_SPEED", "VISIBILITY", "DEW_POINT_TEMPERATURE", "SOLAR_RADIATION",   "RAINFALL", "SNOWFALL")# Exclude non-numeric columns from normalizationnumeric_columns <- numeric_columns[!sapply(bike_sharing_df[numeric_columns], function(x) any(!is.numeric(x)))]bike_sharing_df[numeric_columns] <- lapply(bike_sharing_df[numeric_columns], min_max_normalize)summary(bike_sharing_df)write_csv(bike_sharing_df, "seoul_bike_sharing_converted_normalized.csv")# Dataset listdataset_list <- c('seoul_bike_sharing.csv', 'seoul_bike_sharing_converted.csv', 'seoul_bike_sharing_converted_normalized.csv')for (dataset_name in dataset_list){ # Read dataset dataset <- read_csv(dataset_name) # Standardized its columns: # Convert all columns names to uppercase names(dataset) <- toupper(names(dataset)) # Replace any white space separators by underscore, using str_replace_all function names(dataset) <- str_replace_all(names(dataset), " ", "_") # Save the dataset back write.csv(dataset, dataset_name, row.names=FALSE)}

APPENDIX

- install.packages(c('RSQLite'), repos = 'http://cran.rstudio.com',dependecies=TRUE)library(RSQLite)library(readr)conn <- dbConnect(RSQLite::SQLite(),"Lab-SQL-EDA.sqlite")SEOUL_BIKE_SHARING <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/seoul_bike_sharing.csv")CITIES_WEATHER_FORECAST <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/cities_weather_forecast.csv")BIKE_SHARING_SYSTEMS <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/bike_sharing_systems.csv")WORLD_CITIES <- read_csv("https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/world_cities.csv")dbWriteTable(conn, "SEOUL_BIKE_SHARING", SEOUL_BIKE_SHARING, overwrite = TRUE)dbWriteTable(conn, "CITIES_WEATHER_FORECAST", CITIES_WEATHER_FORECAST, overwrite = TRUE)dbWriteTable(conn, "BIKE_SHARING_SYSTEMS", BIKE_SHARING_SYSTEMS, overwrite = TRUE)dbWriteTable(conn, "WORLD_CITIES", WORLD_CITIES, overwrite = TRUE)head(SEOUL_BIKE_SHARING)#Task 1 - Record Count#Determine how many records are in the seoul_bike_sharing dataset.dbGetQuery(conn, "SELECT COUNT(*) FROM SEOUL_BIKE_SHARING")#Task 2 - Operational Hours#Determine how many hours had non-zero rented bike count.dbGetQuery(conn, "SELECT COUNT(*) FROM SEOUL_BIKE_SHARING WHERE RENTED_BIKE_COUNT != 0")#Task 3 - Weather Outlook#Query the the weather forecast for Seoul over the next 3 hours.#Recall that the records in the CITIES_WEATHER_FORECAST dataset are 3 hours apart, so we just need the first record from the query.dbGetQuery(conn, "SELECT * FROM CITIES_WEATHER_FORECAST WHERE CITY = 'Seoul' LIMIT 1")#Task 4 - Seasons#Find which seasons are included in the seoul bike sharing dataset.dbGetQuery(conn, "SELECT DISTINCT SEASONS FROM SEOUL_BIKE_SHARING")#Task 5 - Date Range#Find the first and last dates in the Seoul Bike Sharing dataset.dbGetQuery(conn, "SELECT MIN(DATE) AS first_date, MAX(DATE) AS last_date FROM SEOUL_BIKE_SHARING")#Task 6 - Subquery - 'all-time high'#determine which date and hour had the most bike rentals.dbGetQuery(conn, "SELECT DATE, HOURFROM SEOUL_BIKE_SHARINGWHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)")#Task 7 - Hourly popularity and temperature by season#Determine the average hourly temperature and the average number of bike rentals per hour over each season. #List the top ten results by average bike count.dbGetQuery(conn, "SELECT SEASONS, HOUR, AVG(TEMPERATURE) AS avg_temperature, AVG(RENTED_BIKE_COUNT) AS avg_bike_countFROM SEOUL_BIKE_SHARINGGROUP BY SEASONS, HOURORDER BY avg_bike_count DESCLIMIT 10")#Task 8 - Rental Seasonality¶#Find the average hourly bike count during each season.#Also include the minimum, maximum, and standard deviation of the hourly bike count for each season.#Hint : Use the SQRT(AVG(col*col) - AVG(col)*AVG(col)) function where col refers to your column name for #finding the standard deviationdbGetQuery(conn, "SELECT SEASONS, HOUR, AVG(RENTED_BIKE_COUNT) AS avg_bike_count, MIN(RENTED_BIKE_COUNT) AS min_bike_count, MAX(RENTED_BIKE_COUNT) AS max_bike_count, SQRT(AVG(RENTED_BIKE_COUNT * RENTED_BIKE_COUNT) - AVG(RENTED_BIKE_COUNT) * AVG(RENTED_BIKE_COUNT)) AS std_dev_bike_countFROM SEOUL_BIKE_SHARINGGROUP BY SEASONS, HOUR")#Task 9 - Weather Seasonality#Consider the weather over each season. On average, what were the TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, #DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, and SNOWFALL per season?## Include the average bike count as well , and rank the results by average bike count so you can see ## if it is correlated with the weather at all.dbGetQuery(conn, "SELECT SEASONS, AVG(TEMPERATURE) AS avg_temperature, AVG(HUMIDITY) AS avg_humidity, AVG(WIND_SPEED) AS avg_wind_speed, AVG(VISIBILITY) AS avg_visibility, AVG(DEW_POINT_TEMPERATURE) AS avg_dew_point_temperature, AVG(SOLAR_RADIATION) AS avg_solar_radiation, AVG(RAINFALL) AS avg_rainfall, AVG(SNOWFALL) AS avg_snowfall, AVG(RENTED_BIKE_COUNT) AS avg_bike_countFROM SEOUL_BIKE_SHARINGGROUP BY SEASONSORDER BY avg_bike_count DESC")#Task 10 - Total Bike Count and City Info for Seoul¶#Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes #avaialble in Seoul, plus the following city information about Seoul: CITY, COUNTRY, LAT, LON, POPULATION, in a single view.#Notice that in this case, the CITY column will work for the WORLD_CITIES table, but in general you would have to use #the CITY_ASCII column.Task 10 - Total Bike Count and City Info for Seoul¶#Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes #avaialble in Seoul, plus the following city information about Seoul: CITY, COUNTRY, LAT, LON, POPULATION, in a single view.#Notice that in this case, the CITY column will work for the WORLD_CITIES table, but in general you would have to use the CITY_ASCII column.dbGetQuery(conn, "SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, SUM(BS.BICYCLES) AS total_bike_countFROM WORLD_CITIES AS WCJOIN BIKE_SHARING_SYSTEMS AS BS ON WC.CITY = BS.CITYWHERE WC.CITY = 'Seoul'GROUP BY WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION")#Task 11 - Find all city names and coordinates with comparable bike scale to Seoul's bike sharing system#Find all cities with total bike counts between 15000 and 20000. Return the city and country names, #plus the coordinates (LAT, LNG), population, and number of bicycles for each city.¶#Later we will ask you to visualize these similar cities on leaflet, with some weather data.dbGetQuery(conn, "SELECT WC.CITY, WC.COUNTRY, WC.LAT, WC.LNG, WC.POPULATION, BS.BICYCLES AS bike_countFROM WORLD_CITIES AS WCJOIN BIKE_SHARING_SYSTEMS AS BS ON WC.CITY = BS.CITYWHERE BS.BICYCLES BETWEEN 15000 AND 20000")dbDisconnect(conn)

APPENDIX

- library(readr)library(tidyverse)seoul_bike_sharing <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/seoul_bike_sharing.csv" seoul_bike_sharing_df <- read_csv(seoul_bike_sharing)#Task 1 - Load the dataset#Ensure you read DATE as type character.seoul_bike_sharing_df$DATE <- as.character(seoul_bike_sharing_df$DATE)#Task 2 - Recast DATE as a date#Use the format of the data, namely "%d/%m/%Y".seoul_bike_sharing_df$DATE <- as.Date(seoul_bike_sharing_df$DATE, format = "%d/%m/%Y")#Task 3 - Cast HOURS as a categorical variable#Also, coerce its levels to be an ordered sequence. #This will ensure your visualizations correctly utilize HOURS as a discrete variable with the expected ordering.seoul_bike_sharing_df$HOUR <- factor(seoul_bike_sharing_df$HOUR, ordered = TRUE)#Check the structure of the dataframestr(seoul_bike_sharing_df)#Finally, ensure there are no missing valuessum(is.na(seoul_bike_sharing_df))#Task 4 - Dataset Summary#Use the base R summary() function to describe the seoul_bike_sharing dataset.summary(seoul_bike_sharing_df)#Task 5 - Based on the above stats, calculate how many Holidays there are.length(unique(seoul_bike_sharing_df$HOLIDAY))#Task 6 - Calculate the percentage of records that fall on a holiday.num_holidays <- sum(seoul_bike_sharing_df$HOLIDAY == "Yes")total_records <- nrow(seoul_bike_sharing_df)holiday_percentage <- (num_holidays / total_records) * 100print(paste("Percentage of Records on Holiday:", round(holiday_percentage, 2), "%"))#Task 7 - Given there is exactly a full year of data, determine how many records we expect to have.earliest_date <- min(seoul_bike_sharing_df$DATE)latest_date <- max(seoul_bike_sharing_df$DATE)days_in_year <- as.numeric(difftime(latest_date, earliest_date, units = "days")) + 1expected_records <- days_in_year * 24print(paste("Expected Number of Records for a Full Year:", expected_records))#Task 8 - Given the observations for the 'FUNCTIONING_DAY' how many records must there be?functioning_days <- sum(seoul_bike_sharing_df$FUNCTIONING_DAY == "Yes")expected_records_per_day <- 24expected_records <- functioning_days * expected_records_per_dayprint(paste("Expected Number of Records based on FUNCTIONING_DAY:", expected_records))#Task 9 - Load the dplyr package, group the data by SEASONS, and use the summarize() function #to calculate the seasonal total rainfall and snowfall.library(dplyr)seasonal_summary <- seoul_bike_sharing_df %>% group_by(SEASONS) %>% summarize(total_rainfall = sum(RAINFALL),   total_snowfall = sum(SNOWFALL))print(seasonal_summary)#Task 10 - Create a scatter plot of RENTED_BIKE_COUNT vs DATE.library(ggplot2)ggplot(seoul_bike_sharing_df, aes(x = DATE, y = RENTED_BIKE_COUNT)) + geom_point(alpha = 0.2) + # Adjust opasity labs(x = "Date", y = "Rented Bike Count") +   ggtitle("Scatter Plot of Rented Bike Count vs Date")#Task 11 - Create the same plot of the RENTED_BIKE_COUNT time series, but now add HOURS as the colour.ggplot(seoul_bike_sharing_df, aes(x = DATE, y = RENTED_BIKE_COUNT, color = HOUR)) + geom_point(alpha = 0.7) +   labs(x = "Date", y = "Rented Bike Count", color = "Hour") +   ggtitle("Time Series Plot of Rented Bike Count vs Date (Color by Hour)") #Task 12 - Create a histogram overlaid with a kernel density curveggplot(seoul_bike_sharing_df, aes(x = RENTED_BIKE_COUNT)) + geom_histogram(aes(y = ..density..), bins = 30, fill = "white", color = "black", alpha = 0.6) + geom_density(color = "black", alpha = 0.3) + labs(x = "Rented Bike Count", y = "Density") + ggtitle("Histogram of Rented Bike Count with Kernel Density Curve")#We can see from the histogram that most of the time there are relatively few bikes rented. Indeed, the 'mode', #or most frequent amount of bikes rented, is about 250.Judging by the 'bumps' at about 700, 900, and 1900, and 3200 bikes, #it looks like there may be other modes hiding within subgroups of the data.Interestingly, judging from the tail of #the distribution, on rare occasions there are many more bikes rented out than usual.#Task 13 - Use a scatter plot to visualize the correlation between RENTED_BIKE_COUNT and TEMPERATURE by SEASONS.#Correlation between two variables (scatter plot)#Start with RENTED_BIKE_COUNT vs. TEMPERATURE, then generate four plots corresponding to the SEASONS #by adding a facet_wrap() layer. #Also, make use of colour and opacity to emphasize any patterns that emerge. Use HOUR as the color.ggplot(seoul_bike_sharing_df, aes(x = TEMPERATURE, y = RENTED_BIKE_COUNT, color = HOUR, alpha = 0.6)) + geom_point() + facet_wrap(~ SEASONS) + labs(x = "Temperature", y = "Rented Bike Count") + ggtitle("Scatter Plot of Rented Bike Count vs Temperature by Seasons")#Visually, we can see some strong correlations as approximately linear patterns.#Note: Comparing this plot to the same plot below, but without grouping by SEASONS, #shows how important seasonality is in explaining bike rental counts.ggplot(seoul_bike_sharing_df) + geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=0.5)#Outliers (boxplot)¶#Task 14 - Create a display of four boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS.#Use facet_wrap to generate four plots corresponding to the seasons.ggplot(seoul_bike_sharing_df, aes(x = HOUR, y = RENTED_BIKE_COUNT)) + geom_boxplot() + facet_wrap(~ SEASONS) + labs(x = "Hour", y = "Rented Bike Count") + ggtitle("Boxplot of Rented Bike Count vs Hour by Seasons")#Although the overall scale of bike rental counts changes with the seasons, key features remain very similar.#For example, peak demand times are the same across all seasons, at 8 am and 6 pm.#Task 15 - Group the data by DATE, and use the summarize() function to calculate the daily total rainfall and snowfall.#Also, go ahead and plot the results if you wish.daily_summary <- seoul_bike_sharing_df %>% group_by(DATE) %>% summarize(total_rainfall = sum(RAINFALL),   total_snowfall = sum(SNOWFALL))ggplot(daily_summary, aes(x = DATE)) + geom_line(aes(y = total_rainfall, color = "Rainfall"), size = 1) + geom_line(aes(y = total_snowfall, color = "Snowfall"), size = 1) + labs(x = "Date", y = "Total Amount", color = "Type") + ggtitle("Daily Total Rainfall and Snowfall")#Task 16 - Determine how many days had snowfalldays_with_snowfall <- seoul_bike_sharing_df %>% filter(SNOWFALL > 0) %>% summarise(days_with_snowfall = n_distinct(DATE))print(paste("Number of days with snowfall:", days_with_snowfall$days_with_snowfall))

APPENDIX

```
library(tidymodels)library(tidyverse)library(stringr)library(glmnet)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)spec(bike_sharing_df)#We won't be using the DATE column, because 'as is', it basically acts like an data entry index.
(However, given more time, #we could use the DATE colum to create a 'day of week' or 'isWeekend' column, which we might expect has
an affect on preferred #bike rental times.) We also do not need the FUNCTIONAL DAY column because it only has one distinct value
remaining (YES) after #missing value processing.bike_sharing_df <- bike_sharing_df %>%   select(-DATE, -
FUNCTIONING_DAY)#TASK 1: Split training and testing data#TODO: Use the initial_split(), training(), and testing() functions to
generate a training dataset #consisting of 75% of the original dataset, and a testing dataset using the remaining 25%.# Split the data into
training and testing setsset.seed(123) bike_split <- initial_split(bike_sharing_df, prop = 0.75)# Extract the training and testing
setsbike_train <- training(bike_split)bike_test <- testing(bike_split)#TASK 2: Build a linear regression model using weather variables
only# Define the linear regression model specification using weather variables# Define linear regression model
specificationlm_spec_weather <- linear_reg() %>%  set_engine(engine = "lm") %>%  set_mode(mode = "regression")# Fit the model using
selected weather variableslm_model_weather <- lm_spec_weather %>%  fit(RENTED_BIKE_COUNT ~ TEMPERATURE +
HUMIDITY + WIND_SPEED + VISIBILITY +         DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL +
SNOWFALL, data = bike_train)# Print the fit summarysummary(lm_model_weather$fit)#TASK 3: Build a linear regression model using
all variables#TODO: Build a linear regression model called lm_model_all using all variables RENTED_BIKE_COUNT ~ # Define linear
regression model specificationlm_spec_all <- linear_reg() %>%  set_engine(engine = "lm") %>%  set_mode(mode = "regression")# Fit the
model using all variableslm_model_all <- lm_spec_all %>%  fit(RENTED_BIKE_COUNT ~ ., data = bike_train)# Print the fit
summarysummary(lm_model_all$fit)#TASK 4: Model evaluation and identification of important variables# Make predictions on the
testing dataset using both modelstest_results_weather <- predict(lm_model_weather, new_data = bike_test)test_results_all <-
predict(lm_model_all, new_data = bike_test)# Define model properties for the Lasso modellasso_spec <- linear_reg(penalty = 0.1, mixture
= 1) %>%   set_engine("glmnet")# Fitting the model to training datalasso_fit <- lasso_spec %>%   fit(RENTED_BIKE_COUNT ~ ., data =
bike_train)# Define model properties for the Elastic-Net modelelastic_net_spec <- linear_reg(penalty = 0.1, mixture = 0.5) %>%
set_engine("glmnet")# Fitting the model to training dataelastic_net_fit <- elastic_net_spec %>%   fit(RENTED_BIKE_COUNT ~ ., data =
bike_train)# Making predictions for the Lasso modellasso_preds <- predict(lasso_fit, new_data = bike_test)# Make predictions for the
Elastic-Net modelelastic_net_preds <- predict(elastic_net_fit, new_data = bike_test)# Perform predictions on training and test
datatrain_results <- lm_model_weather %>%  predict(new_data = bike_train) %>%  mutate(truth =
bike_train$RENTED_BIKE_COUNT)test_results <- lm_model_weather %>%  predict(new_data = bike_test) %>%  mutate(truth =
bike_test$RENTED_BIKE_COUNT)# Calculate R-squared rsq_train <- rsq(train_results, truth = truth,    estimate = .pred)rsq_test <-
rsq(test_results, truth = truth,    estimate = .pred)# Calculate RMSErmse_train <- rmse(train_results, truth = truth,     estimate =
.pred)rmse_test <- rmse(test_results, truth = truth,     estimate =
.pred)print(rsq_train)print(rsq_test)print(rmse_train)print(rmse_test)lm_model_all$fit$coefficients# Make coefficients positive and subtract
N/A valuespositive_coefficients <- abs(lm_model_all$fit$coefficients)positive_coefficients <-
positive_coefficients[!is.na(positive_coefficients)]# Visualizing coefficientsggplot(data = data.frame(Variable =
names(positive_coefficients), Coefficient = positive_coefficients), aes(x = reorder(Variable, Coefficient), y = Coefficient)) +
geom_bar(stat = "identity", fill = "black", width = 0.5) +  coord_flip() +  labs(title = "Coefficients of Variables",       x = "Variable",       y =
"Coefficient (Positive Values)") +  theme_minimal() +  theme(axis.text.y = element_text(size = 8))
```

APPENDIX

```
library("tidymodels")library("tidyverse")library("stringr")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)spec(bike_sharing_df)bike_sharing_df <-
bike_sharing_df %>%   select(-DATE, -FUNCTIONING_DAY)lm_spec <- linear_reg() %>%  set_engine("lm") %>%   set_mode("regression")set.seed(1234)data_split <-
initial_split(bike_sharing_df, prop = 4/5)train_data <- training(data_split)test_data <- testing(data_split)ggplot(data = train_data, aes(RENTED_BIKE_COUNT,
TEMPERATURE)) +   geom_point() ggplot(data=train_data, aes(RENTED_BIKE_COUNT, TEMPERATURE)) +   geom_point() +   geom_smooth(method = "lm", formula = y
~ x, color="red") +   geom_smooth(method = "lm", formula = y ~ poly(x, 2), color="yellow") +   geom_smooth(method = "lm", formula = y ~ poly(x, 4), color="green") +
geom_smooth(method = "lm", formula = y ~ poly(x, 6), color="blue")# Create a linear model by adding higher degree polynomials on important variableslm_poly_spec <-
linear_reg() %>%  set_engine("lm") %>%   set_mode("regression")lm_poly_wf <- workflow() %>%  add_model(lm_poly_spec) %>%  add_formula(RENTED_BIKE_COUNT
~ poly(TEMPERATURE, 6) + poly(HUMIDITY, 4))# Fit the model to training datalm_poly_fit <- fit(lm_poly_wf, data = train_data)# Model
summarysummary(lm_poly_fit)predictions <- predict(lm_poly_fit, new_data = test_data)# Convert negative prediction results to zeropredictions[predictions < 0] <-
0print(predictions)# Calculate residualsresiduals <- predictions - test_data$RENTED_BIKE_COUNThead(residuals)# Extract actual values from test dataactual <-
test_data$RENTED_BIKE_COUNT# Extract predicted values from predictionspredicted <- predictions$.pred# Calculate R-squaredmean_actual <- mean(actual)ss_total <-
sum((actual - mean_actual)^2)ss_residual <- sum((actual - predicted)^2)rsquared <- 1 - (ss_residual / ss_total)# Calculate RMSErmse <- sqrt(mean((actual - predicted)^2))# Print
R-squared and RMSEcat("R-squared:", rsquared, "\n")cat("RMSE:", rmse, "\n")# Add interaction terms to the polynomial regression modellm_poly_wf <- workflow() %>%
add_model(lm_poly_spec) %>%  add_formula(    RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) * poly(HUMIDITY, 4) +      TEMPERATURE * HUMIDITY  )# Fit the
model to training datalm_poly_fit_with_interaction <- fit(lm_poly_wf, data = train_data)# Model summarysummary(lm_poly_fit_with_interaction)# Predictions with interaction
termspredictions_with_interaction <- predict(lm_poly_fit_with_interaction, new_data = test_data)# Convert negative prediction results to
zeropredictions_with_interaction$.pred[predictions_with_interaction$.pred < 0] <- 0# Calculate residualsresiduals_with_interaction <- predictions_with_interaction$.pred -
test_data$RENTED_BIKE_COUNT# Calculate R-squared and RMSErsquared_with_interaction <- 1 - sum(residuals_with_interaction^2) /
sum((test_data$RENTED_BIKE_COUNT - mean(test_data$RENTED_BIKE_COUNT))^2)rmse_with_interaction <- sqrt(mean(residuals_with_interaction^2))# Print R-squared
and RMSEcat("R-squared (with interaction terms):", rsquared_with_interaction, "\n")cat("RMSE (with interaction terms):", rmse_with_interaction,
"\n")library('glmnet')library('Matrix')# Define a linear regression model specification glmnet_spec using glmnet engine# You can adjust the penalty and mixture parameters as
neededglmnet_spec <- linear_reg(penalty = 0.1, mixture = 0.5) %>%  set_engine("glmnet") %>%  set_mode("regression")# Fit a glmnet model called lm_glmnet using the fit()
function# For the formula part, keep the polynominal and interaction terms you used in the previous tasklm_glmnet <- fit(glmnet_spec, RENTED_BIKE_COUNT ~
poly(TEMPERATURE, 6) * poly(HUMIDITY, 4) + TEMPERATURE * HUMIDITY, data = train_data)# Report rsq and rmse of the `lm_glmnet` modelpredictions_glmnet <-
predict(lm_glmnet, new_data = test_data)predictions_glmnet$.pred[predictions_glmnet$.pred < 0] <- 0residuals_glmnet <- predictions_glmnet$.pred -
test_data$RENTED_BIKE_COUNTrsquared_glmnet <- 1 - sum(residuals_glmnet^2) / sum((test_data$RENTED_BIKE_COUNT -
mean(test_data$RENTED_BIKE_COUNT))^2)rmse_glmnet <- sqrt(mean(residuals_glmnet^2))cat("R-squared (with glmnet):", rsquared_glmnet, "\n")cat("RMSE (with
glmnet):", rmse_glmnet, "\n")# Add regularization# Define a linear regression model specification glmnet_spec using glmnet engineglmnet_spec <- linear_reg(penalty = 0.1,
mixture = 0.5) %>%  set_engine("glmnet") %>%  set_mode("regression")library('glmnet')library('Matrix')# Fit a glmnet model using the fit() functionlm_glmnet <-
fit(glmnet_spec, RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) * poly(HUMIDITY, 4)                + TEMPERATURE * HUMIDITY, data = train_data)# Report rsq and
rmse of the `lm_glmnet` modelpredictions_glmnet <- predict(lm_glmnet, new_data = test_data)predictions_glmnet$.pred[predictions_glmnet$.pred < 0] <- 0residuals_glmnet <-
predictions_glmnet$.pred - test_data$RENTED_BIKE_COUNTrsquared_glmnet <- 1 - sum(residuals_glmnet^2) / sum((test_data$RENTED_BIKE_COUNT -
mean(test_data$RENTED_BIKE_COUNT))^2)rmse_glmnet <- sqrt(mean(residuals_glmnet^2))cat("R-squared (with glmnet):", rsquared_glmnet, "\n")cat("RMSE (with
glmnet):", rmse_glmnet, "\n")# Build at least five different models using polynomial terms, interaction terms, and regularizations.# Save their rmse and rsq values# Define a list
to store the performance of each modelmodel_performance_df <- data.frame()# Define a list of penalties for the glmnet modelpenalties <- c(0.1, 0.5, 1.0, 2.0, 5.0)# Loop over the
penaltiesfor (penalty in penalties) {  # Define the glmnet model specification  glmnet_spec <- linear_reg(penalty = penalty, mixture = 0.5) %>%    set_engine("glmnet") %>%
set_mode("regression")    # Fit the glmnet model  lm_glmnet <- fit(glmnet_spec, RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) * poly(HUMIDITY, 4) +
TEMPERATURE * HUMIDITY, data = train_data)    # Make predictions  predictions <- predict(lm_glmnet, new_data = test_data)    # Calculate RMSE and R-squared  rmse <-
sqrt(mean((test_data$RENTED_BIKE_COUNT - predictions$.pred)^2))  rsquared <- 1 - sum((test_data$RENTED_BIKE_COUNT - predictions$.pred)^2) /
sum((test_data$RENTED_BIKE_COUNT - mean(test_data$RENTED_BIKE_COUNT))^2)    # Add the performance to the data frame  model_performance_df <-
rbind(model_performance_df, data.frame(Model = paste("glmnet", penalty), RMSE = rmse, Rsquared = rsquared))}# HINT: Use ggplot() +
geom_bar()library(ggplot2)library(reshape2)# Reshape the data for ggplotmodel_performance_long <- melt(model_performance_df, id.vars = "Model")# Plot the performance of
each modelggplot(model_performance_long, aes(x = Model, y = value, fill = variable)) +  geom_bar(stat = "identity", position = "dodge") +  labs(x = "Model", y = "Value", fill =
"Metric") +  theme_minimal()# HINT: Use ggplot() +# stat_qq(aes(sample=truth), color='green') +# stat_qq(aes(sample=prediction), color='red')ggplot() +  stat_qq(aes(sample =
test_data$RENTED_BIKE_COUNT), color = 'green') +  stat_qq(aes(sample = predictions$.pred), color = 'red') +  labs(title = "Q-Q Plot of Predictions vs True Values", x =
"Theoretical Quantiles", y = "Sample Quantiles") +  theme_minimal()
```

APPENDIX

- # Import librarieslibrary(shiny)library(ggplot2)library(tidyverse)# Define UIui <- fluidPage(br(), # TASK 1: Application title titlePanel(title = "Data Exploration Using Shiny"), # Define sidebar layout with sidebar panel and main panel sidebarLayout(sidebarPanel(# TASK 2: Add h3 and variable select inputs for continuous/categorical   h3('Explore mtcars'),   varSelectInput("continuous_variable",   "Select Continuous Variable",   data = select(mtcars, -cyl, -vs, -am, -gear, -carb),   selected = "mpg"),   varSelectInput("categorical_variable",   "Select Categorical Variable",   data = mtcars %>% select(cyl, vs, am, gear, carb),   selected = "cyl"),   # TASK 3: Add numeric input for bins and radio buttons for fill   h3("Histogram settings:"),   numericInput("bins",   "Number of bins", min = 2,   max = 20,   value = 10),   radioButtons("hist_fill",   "Histogram fill:", choices = c("default", "blue")),   # TASK 4: Add variable map guide   h4('Plot Variable Map Guide'), p('Miles/gallon = mpg', br(),   'Displacement (cu in.) = disp', br(),   'Gross horsepower = hp', br(),   'Rear axle ratio = drat', br(),   'Weight (1000 lbs) = wt', br(),   '1/4 mile time = qsec', br(),   'Number of cylinders = cyl', br(),   'Engine\n(0 = V-shaped, 1 = straight) = vs', br(),   'Transmission\n(0 = automatic, 1 = manual) = am', br(),   'Number of forward gears = gear', br(),   'Number of carburetors = carb'   )   ), mainPanel(# TASK 5: Add three panels   tabsetPanel(tabPanel("Distribution of Numerical Variables",   plotOutput("p1"), # histogram   plotOutput("p2")   # boxplot   ),   tabPanel("Distribution of Categorical Variables",   plotOutput("p3") # bar chart   ),   tabPanel("Plots for Observing Data Correlation",   plotOutput("p4")) # scatter plot   )   )))# Define server logicserver <- function(input, output) { # TASK 6: Histogram   output$p1 <- renderPlot({   # Base ggplot object   p <- ggplot(mtcars, aes(x = !!input$continuous_variable)) +   labs(y = "Number of Cars", title = paste("Trend of ", input$continuous_variable))   # Based on radio button input, change histogram fill   if (input$hist_fill == "default") {   p + geom_histogram(bins = input$bins)   }   else {   p + geom_histogram(bins = input$bins, fill = "dodgerblue3")   } })   # TASK 7: Boxplot output$p2 <- renderPlot({   ggplot(mtcars, aes(y = !!input$continuous_variable)) +   geom_boxplot() +   labs(title = paste("How", input$continuous_variable, "value is spread")) +   coord_flip() })   # TASK 8: Bar chart output$p3 <- renderPlot({   ggplot(data = mtcars, aes(x = factor(!!input$categorical_variable),   fill = factor(!!input$categorical_variable))) +   geom_bar() +   labs(x = input$categorical_variable,   title = paste("Trend of", input$categorical_variable)) })   # TASK 9: Scatter plot output$p4 <- renderPlot({   ggplot(mtcars, aes(x = !!input$continuous_variable, y = wt,   color = factor(!!input$categorical_variable))) +   geom_point(size = 3) +   labs(title = paste("Distribution of", input$continuous_variable, "with respect to Weight")) })}# Run the applicationshinyApp(ui = ui, server = server)