# Applied Data Science with R Capstone project

**Aleksandr Migunov**

**02/11/2022**

# Outline

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

# Executive Summary

- Collecting and understanding data from multiple sources
- Performing data wrangling and preparation with regular expressions and Tidyverse
- Performing exploratory data analysis with SQL and visualization using Tidyverse and ggplot2
- Performing modelling the data with linear regressions using Tidymodels
- Building an interactive dashboard using R Shiny

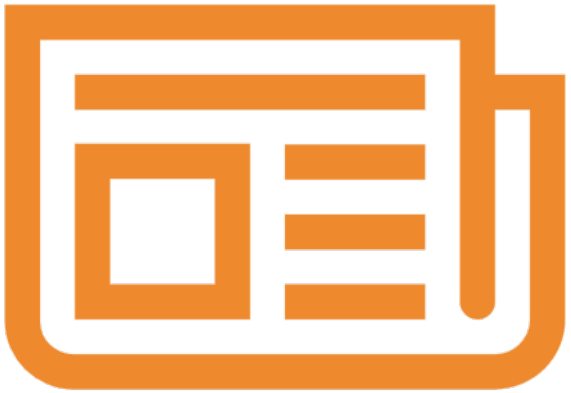
# Introduction

- **The Use Case.** The goal of this project is to analyze how weather would affect bike-sharing demand in urban areas.
- **The Solution of the Use Case:**
- 1) collect and process related weather and bike-sharing demand data from various sources;
- 2) perform exploratory data analysis on the data;

# Introduction

- The Solution of the Use Case:
- 3) build predictive models to predict bike-sharing demand;
- 4) combine the results and connect them to a live dashboard displaying an interactive map and associated visualization of the current weather and the estimated bike demand.

# Methodology



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

# Methodology

# Data collection

- Data sets were collected in two ways: 1) through webscraping, 2) through API calls to OpenWeather API.
- Bike sharing systems HTML table was extracted from a Wiki page and converted into a data frame.
- 5-day weather forecasts was get for a list of cities using the OpenWeather API.
- Also, two datasets were downloaded as csv files from cloud storage.

# Data wrangling

- Data wrangling was done in two ways:
- 1) with regular expressions,
- 2) with dplyr.

# Data wrangling with regular expressions

- Data wrangling with regular expressions included:
- 1) standardization of column names for all collected datasets,
- 2) removal of undesired reference links using regular expressions,
- 3) extraction of the numeric value using regular expressions.

# Data wrangling with dplyr

- Data wrangling with dplyr included:
- 1) detecting and handling missing values,
- 2) creating indicator (dummy) variables for categorical variables,
- 3) normalizing data.

# EDA with SQL

- Performed SQL queries:
- 1. Determine how many records are in the seoul_bike_sharing dataset.
- 2. Determine how many hours had non-zero rented bike count.
- 3. Query the the weather forecast for Seoul over the next 3 hours.
- 4. Find which seasons are included in the seoul bike sharing dataset.

# EDA with SQL

- Performed SQL queries:
- 5. Find the first and last dates in the Seoul Bike Sharing dataset.
- 6. Determine which date and hour had the most bike rentals.
- 7. Determine the average hourly temperature and the average number of bike rentals per hour over each season. List the top ten results by average bike count.

# EDA with SQL

- Performed SQL queries:
- 8. Find the average hourly bike count during each season.
- 9. 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?

# EDA with SQL

- Performed SQL queries:
- 10. 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.
- 11. Find all cities with total bike counts between 15000 and 20000. Return the city and country names, plus the coordinates (LAT, LNG), population, and number of bicycles for each city.

# EDA with data visualization

- Plotted charts:
- 1. Scatter plot of RENTED_BIKE_COUNT vs DATE.
- 2. Scatter plot of RENTED_BIKE_COUNT vs DATE with HOURS added as color.
- 3. Histogram of RENTED_BIKE_COUNT overlaid with a kernel density curve.
- 4. Scatter plots of RENTED_BIKE_COUNT vs TEMPERATURE by SEASONS.

# EDA with data visualization

- Plotted charts:
- 5. Display of four boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS.
- 6. Scatter plot of daily total snowfall vs. DATE.

# Predictive analysis

- Building, evaluating, improving and finding the best performing model:
- 1. Splitting data into training and testing datasets
- 2. Building a linear regression model using only the weather variables
- 3. Building a linear regression model using both weather and date variables
- 4. Evaluating the models and identifying important variables

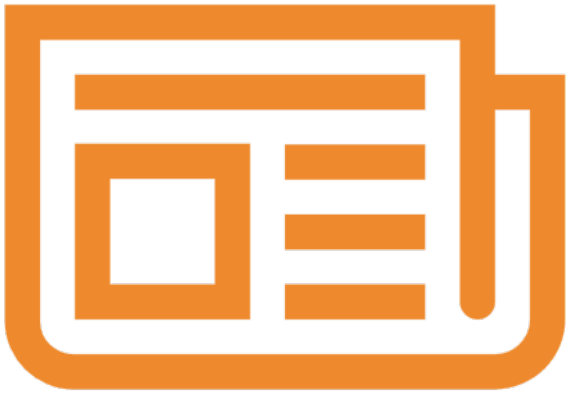
# Predictive analysis

- Building, evaluating, improving and finding the best performing model:
- 5. Adding polynomial terms
- 6. Adding interactions terms
- 7. Adding regularization terms
- 8. Experimenting to search for improved models

# Build a R Shiny dashboard

- Dashboard includes:
- 1) a basic max bike prediction overview map,
- 2) a select input (dropdown) to select a specific city,
- 3) a static temperature trend line,
- 4) an interactive bike-sharing demand prediction trend line,
- 5) a static humidity and bike-sharing demand prediction correlation plot.

# Results



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

# EDA with SQL

# Busiest bike rental times

- Find dates and hours which had the most bike rentals

```
In [7]: # provide your solution here
query = "SELECT DATE, HOUR, RENTED_BIKE_COUNT FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT=(SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)"
sqlQuery(conn, query)
```

A data.frame: 1 × 3

| | DATE | HOUR | RENTED_BIKE_COUNT |
|---|---|---|---|
| | <fct> | <int> | <int> |
| 1 | 2018-06-19 | 18 | 3556 |

- The most busy time was June 19, 2018 at 6 PM.

# Hourly popularity and temperature by seasons

- Find hourly popularity and temperature by season

AVG_RENTALS - average bike rental

AVG_TEMP - average temperatu

```
In [8]: # provide your solution here
query = "SELECT SEASONS, AVG(RENTED_BIKE_COUNT) AS AVG_RENTALS, AVG(TEMPERATURE) AS AVG_TEMP
FROM SEOUL_BIKE_SHARING GROUP BY SEASONS"
sqlQuery(conn, query)
```

A data.frame: 4 × 3

| | SEASONS | AVG_RENTALS | AVG_TEMP |
|---|---|---|---|
| | <fct> | <int> | <dbl> |
| 1 | Autumn | 924 | 13.821167 |
| 2 | Spring | 746 | 13.021389 |
| 3 | Summer | 1034 | 26.587274 |
| 4 | Winter | 225 | -2.540463 |

# Rental Seasonality

- Rental Seasonality

AVG - average, STDDEV - standard deviation

```
In [9]: # provide your solution here
query = "SELECT SEASONS, AVG(RENTED_BIKE_COUNT) AS AVG_COUNT, MIN(RENTED_BIKE_COUNT) AS MIN_COUNT,
MAX(RENTED_BIKE_COUNT) AS MAX_COUNT, STDDEV(RENTED_BIKE_COUNT) AS STDDEV_COUNT
FROM SEOUL_BIKE_SHARING GROUP BY SEASONS"
sqlQuery(conn, query)
```

A data.frame: 4 × 5

| | SEASONS | AVG_COUNT | MIN_COUNT | MAX_COUNT | STDDEV_COUNT |
|---|---|---|---|---|---|
| | <fct> | <int> | <int> | <int> | <dbl> |
| 1 | Autumn | 924 | 2 | 3298 | 617.3885 |
| 2 | Spring | 746 | 2 | 3251 | 618.5247 |
| 3 | Summer | 1034 | 9 | 3556 | 690.0884 |
| 4 | Winter | 225 | 3 | 937 | 150.3374 |

# Weather Seasonality

- Weather Seasonality

HUMID - humidity, VIS - visibility

In [10]:
```
# provide your solution here
query = "SELECT SEASONS, AVG(TEMPERATURE) AS AVG_TEMP, AVG(HUMIDITY) AS AVG_HUMID, AVG(WIND_SPEED) AS AVG_WIND,
AVG(VISIBILITY) AS AVG_VIS, AVG(DEW_POINT_TEMPERATURE) AS AVG_DEW_POINT, AVG(SOLAR_RADIATION) AS AVG_RADIATION,
AVG(RAINFALL) AS AVG_RAIN, AVG(SNOWFALL) AS AVG_SNOW, AVG(RENTED_BIKE_COUNT) AS AVG_COUNT
FROM SEOUL_BIKE_SHARING GROUP BY SEASONS"
sqlQuery(conn, query)
```

A data.frame: 4 × 10

| | SEASONS | AVG_TEMP | AVG_HUMID | AVG_WIND | AVG_VIS | AVG_DEW_POINT | AVG_RADIATION | AVG_RAIN | AVG_SNOW | AVG_COUNT |
|---|---|---|---|---|---|---|---|---|---|---|
| | <fct> | <dbl> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> |
| 1 | Autumn | 13.821167 | 59 | 1.492101 | 1558 | 5.150594 | 0.5227827 | 0.11765617 | 0.06350026 | 924 |
| 2 | Spring | 13.021389 | 58 | 1.857778 | 1240 | 4.091389 | 0.6803009 | 0.18694444 | 0.00000000 | 746 |
| 3 | Summer | 26.587274 | 64 | 1.609420 | 1501 | 18.750136 | 0.7612545 | 0.25348732 | 0.00000000 | 1034 |
| 4 | Winter | -2.540463 | 49 | 1.922685 | 1445 | -12.416667 | 0.2981806 | 0.03282407 | 0.24750000 | 225 |

# Bike-sharing info in Seoul

- Find the total Bike count and city info for Seoul

```
In [11]: # provide your solution here
query = "SELECT W.CITY, W.COUNTRY, W.LAT, W.LNG, W.POPULATION, B.BICYCLES FROM WORLD_CITIES W, BIKE_SHARING_SYSTEMS B
WHERE W.CITY='Seoul' AND W.CITY=B.CITY"
sqlQuery(conn, query)
```

A data.frame: 1 × 6

| | CITY | COUNTRY | LAT | LNG | POPULATION | BICYCLES |
|---|---|---|---|---|---|---|
| | <fct> | <fct> | <dbl> | <dbl> | <int> | <int> |
| 1 | Seoul | Korea, South | 37.58 | 127 | 21794000 | 20000 |

- Seoul population = 21,794,000 people, number of bicycles = 20,000.

# Cities similar to Seoul

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

Similar cities:

Shanghai,

Beijing,

Weifang,

Ningbo,

Zhuzhou

In [12]:
```
# provide your solution here
query = "SELECT W.CITY, W.COUNTRY, W.LAT, W.LNG, W.POPULATION, B.BICYCLES FROM WORLD_CITIES W, BIKE_SHARING_SYSTEMS B
WHERE B.BICYCLES>=15000 AND B.BICYCLES<='20000' AND W.CITY=B.CITY"
sqlQuery(conn, query)
```

A data.frame: 6 × 6

| | CITY | COUNTRY | LAT | LNG | POPULATION | BICYCLES |
|---|---|---|---|---|---|---|
| | <fct> | <fct> | <dbl> | <dbl> | <int> | <int> |
| 1 | Shanghai | China | 31.16 | 121.46 | 22120000 | 19165 |
| 2 | Seoul | Korea, South | 37.58 | 127.00 | 21794000 | 20000 |
| 3 | Beijing | China | 39.90 | 116.39 | 19433000 | 16000 |
| 4 | Weifang | China | 36.71 | 119.10 | 9373000 | 20000 |
| 5 | Ningbo | China | 29.87 | 121.54 | 7639000 | 15000 |
| 6 | Zhuzhou | China | 27.84 | 113.14 | 3855609 | 20000 |

# EDA with Visualization

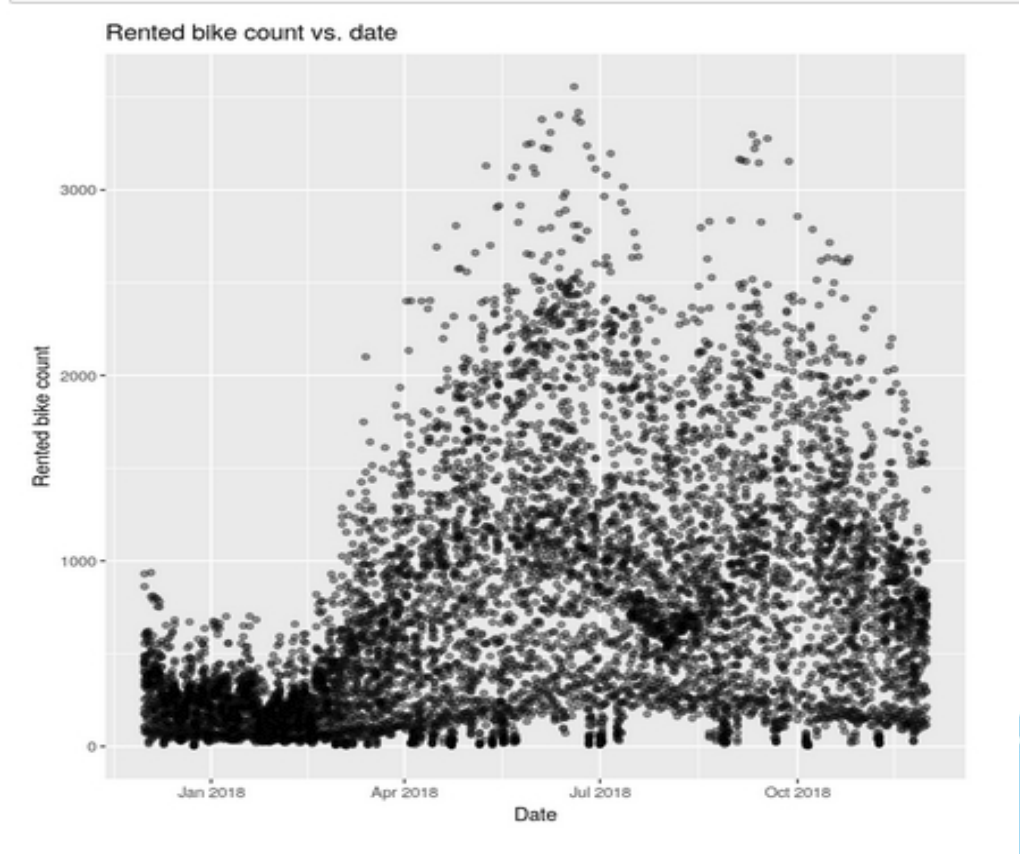
# Bike rental vs. Date

Scatter plot
of RENTED_BIKE_COUNT vs. DATE

x = DATE,

y = RENTED_BIKE_COUNT

This plot has maximum in June and minimum in February.



# Bike rental vs. Datetime

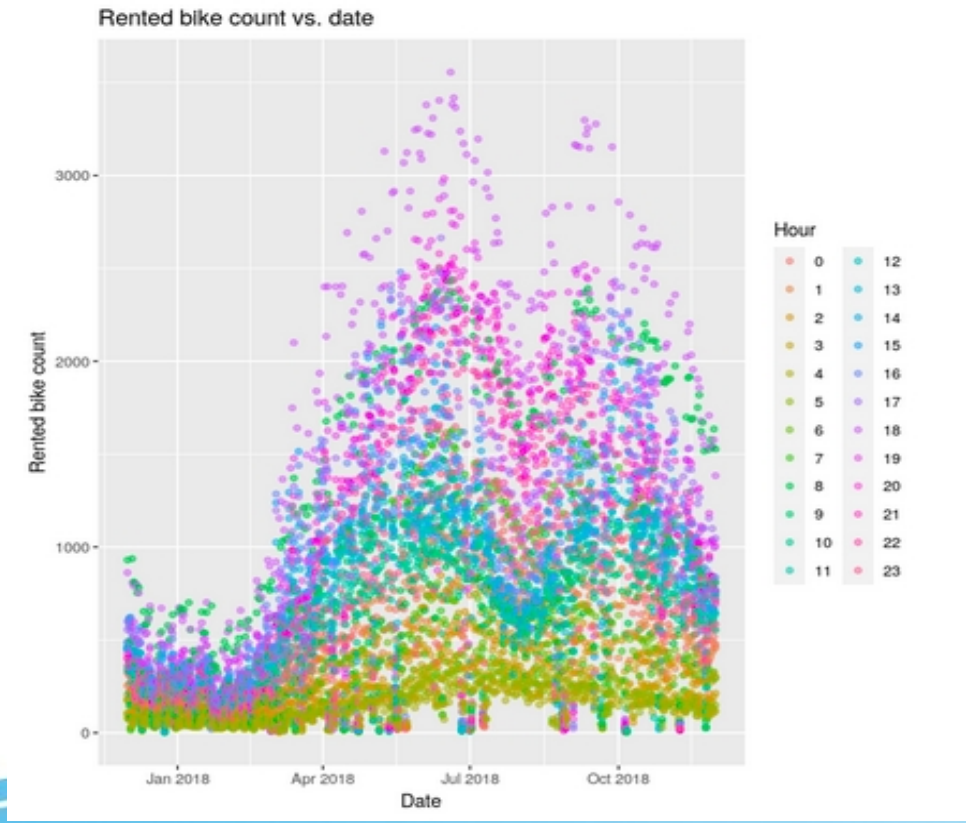
Scatter plot of
the RENTED_BIKE_COUNT vs. TIME
with HOURS as the color

x = DATE,

y = RENTED_BIKE_COUNT,

color = HOURS

The highest number is at 17:00, the lowest number is at 4:00.



# Bike rental histogram

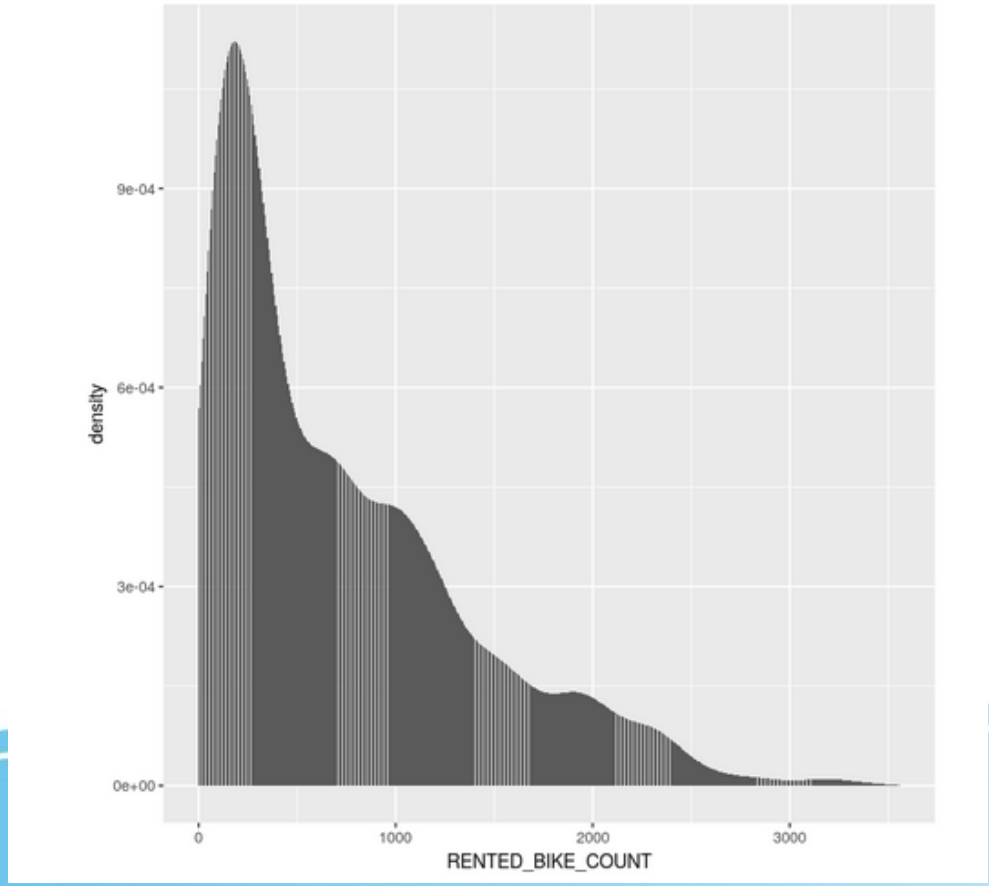
Histogram of RENTED_BIKE_COUNT overlaid with a kernel density curve

x = RENTED_BIKE_COUNT, y = density

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.

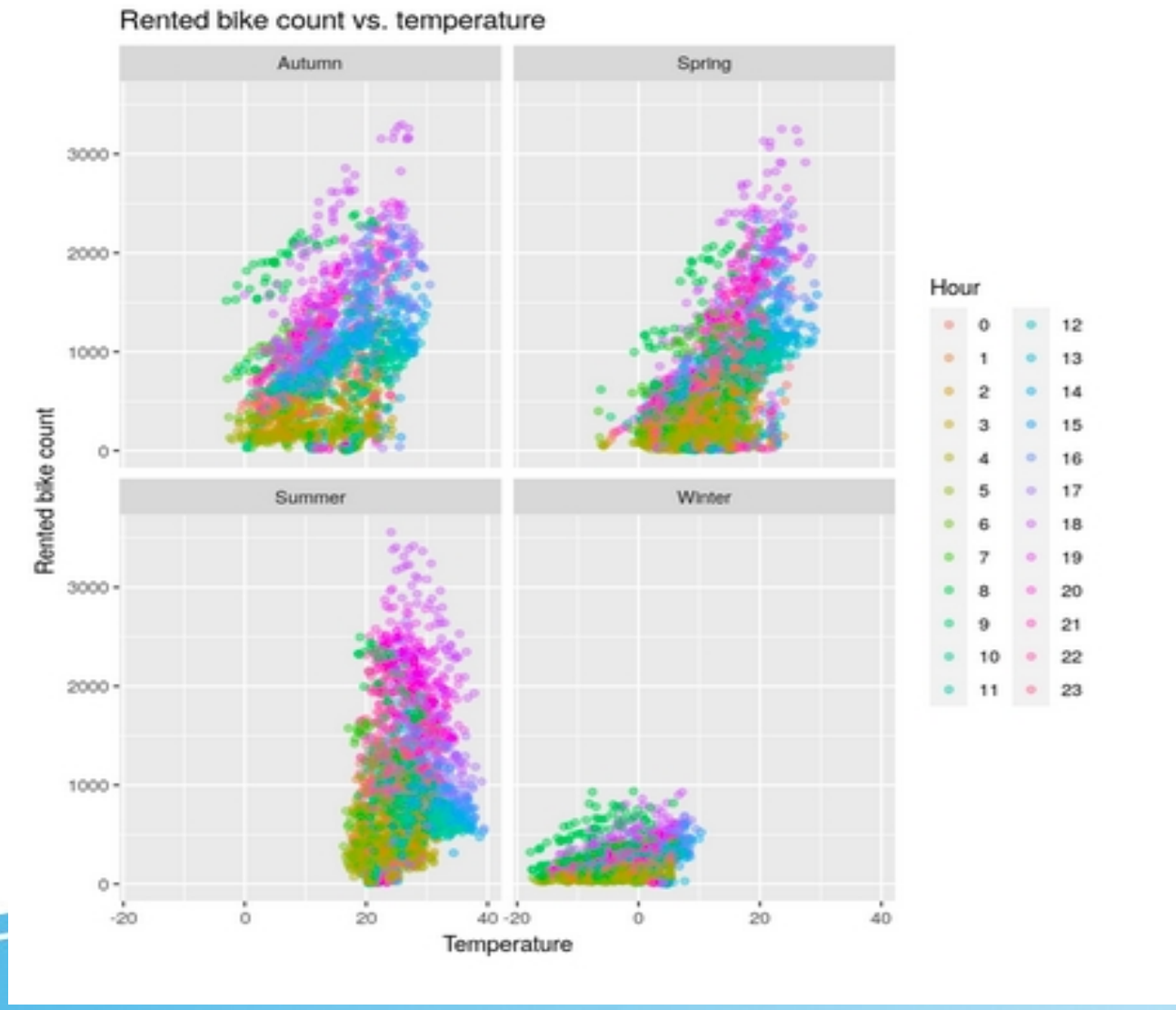
Interestingly, judging from the tail of the distribution, on rare occasions there are many more bikes rented out than usual. the tail of the distribution, on rare occasions there are many more bikes rented out than usual.



# Bike rental vs. Temperature by Seasons

Scatter plot of
the RENTED_BIKE_COUNT vs.
TEMPERATURE with HOURS as the
color grouped by SEASONS

Visually, we can see some
strong correlations as
approximately linear patterns.

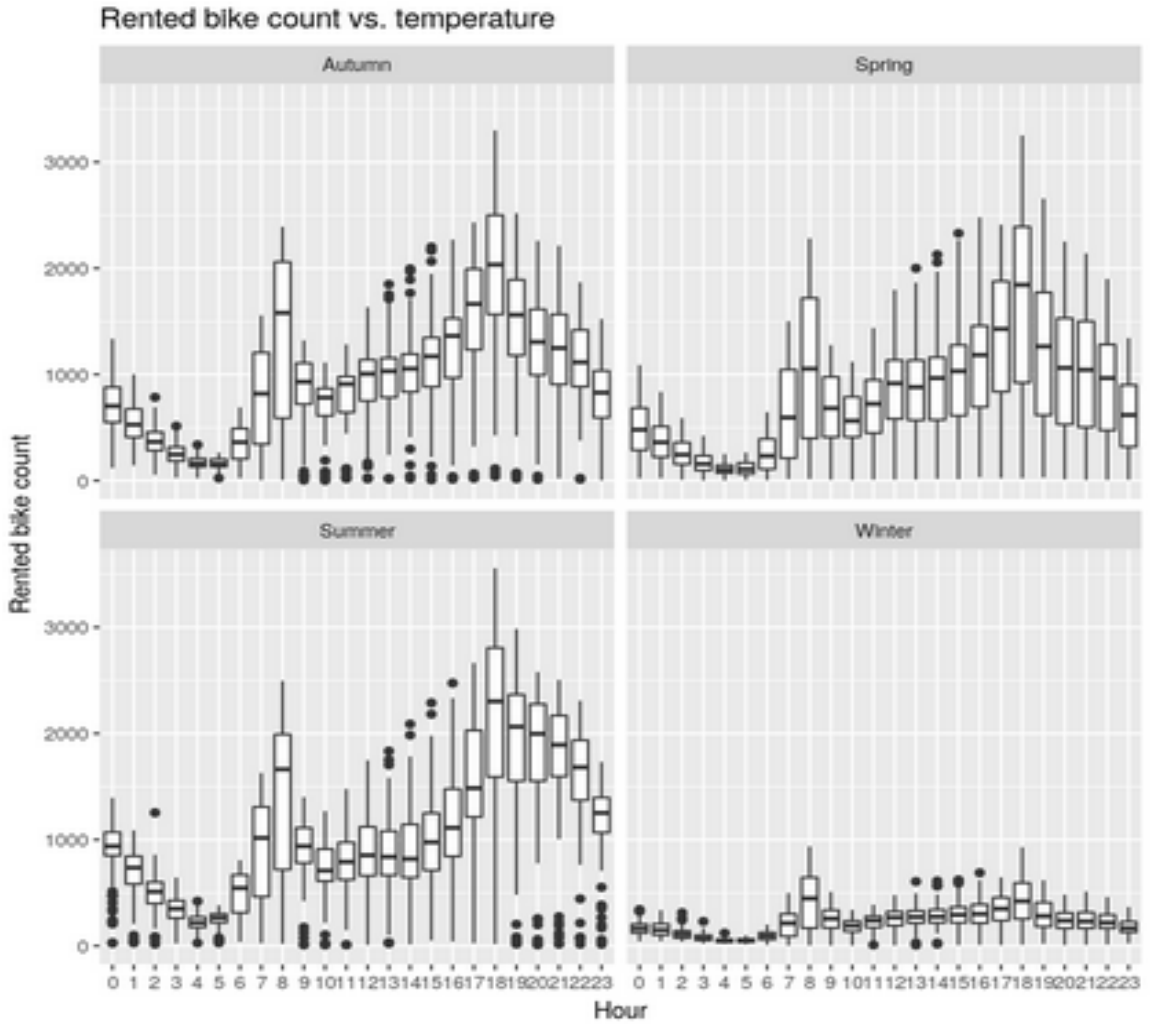


# Bike rental vs. Hours by Seasons

Box plots of
the RENTED_BIKE_COUNT vs.
TEMPERATURE grouped 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.



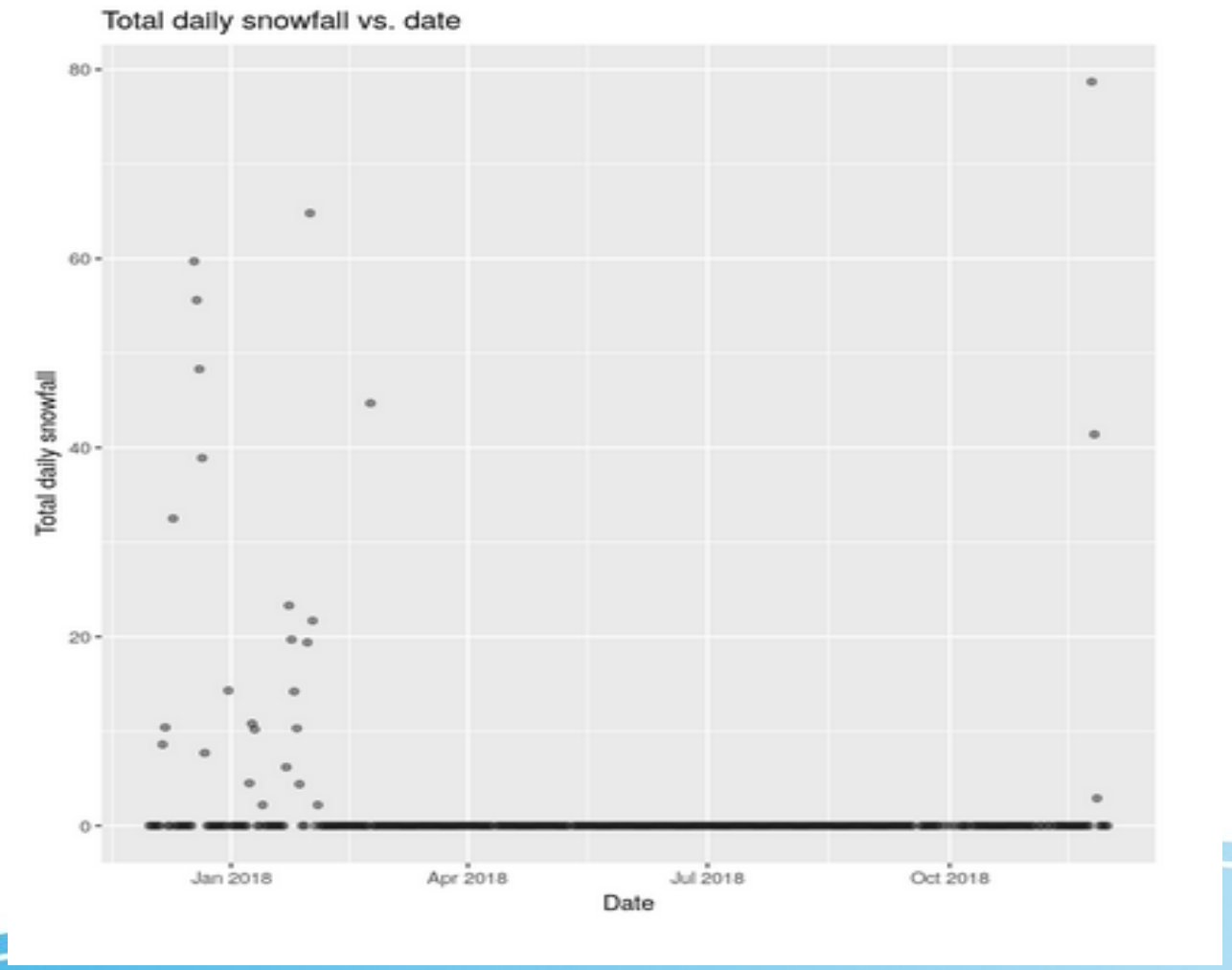
# Daily total snowfall

Show a scatter plot of the daily total snowfall vs. DATE

x = DATE,

y = total daily snowfall

As expected, snowfall is in winter and early spring.



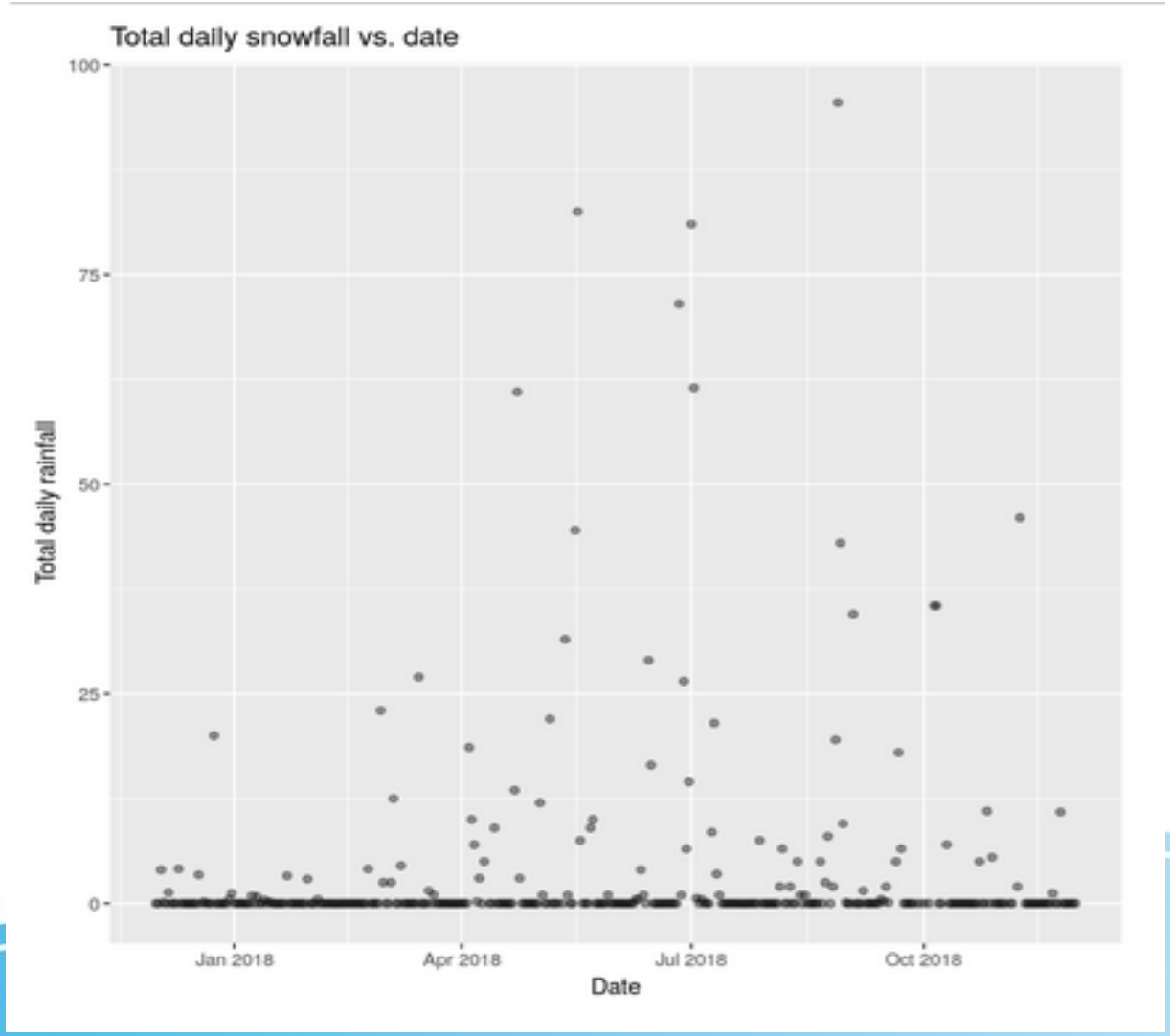
# Daily total rainfall

Show a scatter plot of the daily total rainfall vs. DATE

x = DATE,

y = total daily rainfall

As expected, most rainfall is in summer and autumn.

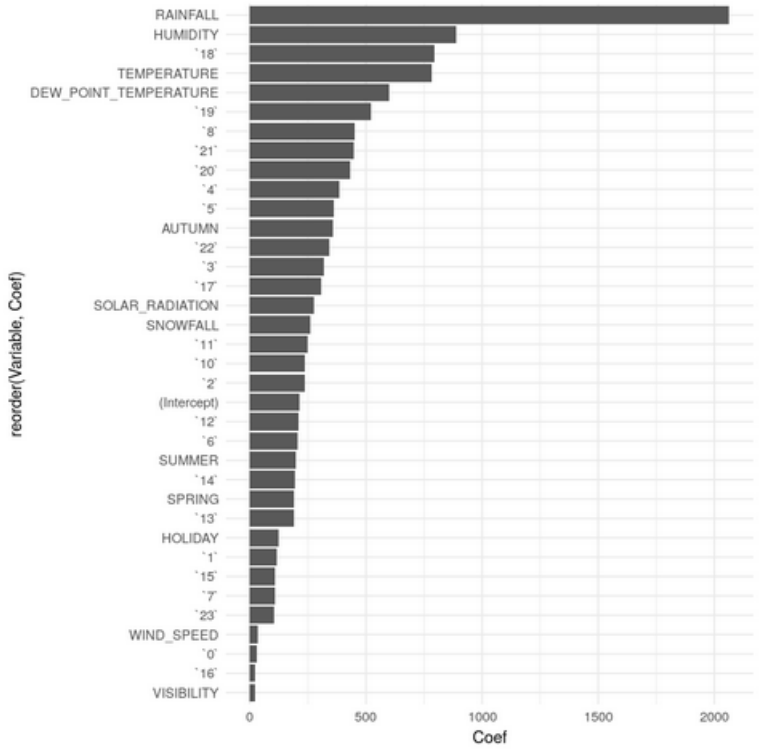


# Predictive analysis

# Ranked coefficients

Screenshot of the ranked coefficients bar chart for the baseline model is on the right.

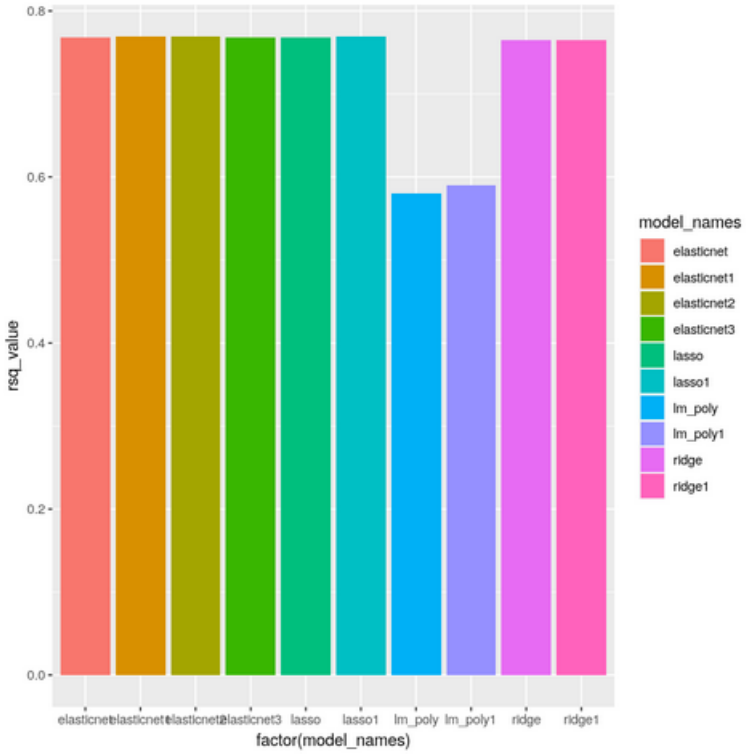
It shows that some coefficients are very important, like rainfall, humidity, and temperature, while others are less important.



# Model evaluation

Built models are: lm_poly (with polynomial terms), lm_poly1 (with polynomial and interaction terms), ridge (with penalty 0.1), ridge1 (with penalty 1), lasso (with penalty 0.1), lasso1 (with penalty 1), elasticnet (with penalty 0.1 and mixture 0.5), elasticnet1 (with penalty 1 and mixture 0.5), elasticnet2 (with penalty 1 and mixture 0.3), elasticnet3 (with penalty 1 and mixture 0.1).

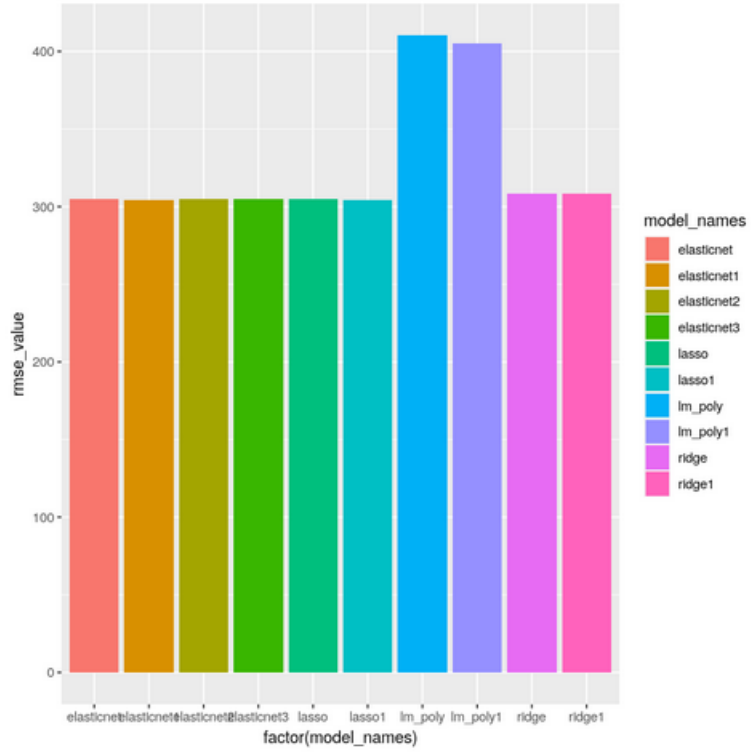
Their R-squared values are shown on the grouped bar chart on the right.



# Model evaluation

Built models are: lm_poly (with polynomial terms), lm_poly1 (with polynomial and interaction terms), ridge (with penalty 0.1), ridge1 (with penalty 1), lasso (with penalty 0.1), lasso1 (with penalty 1), elasticnet (with penalty 0.1 and mixture 0.5), elasticnet1 (with penalty 1 and mixture 0.5), elasticnet2 (with penalty 1 and mixture 0.3), elasticnet3 (with penalty 1 and mixture 0.1).

Their RMSE values are shown on the grouped bar chart on the right.



# Find the best performing model

- Select the best performing model in terms of R-squared is lasso 1.
- It has R-squared =
- 0.7688, and RMSE =
- 304.561.

```
In [47]: # Report the best performed model in terms of rmse and rsq
         ## The best model in terms of rsq is lasso1
         rsq_lasso1
         rmse_lasso1
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.7688183 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 304.561 |

# Find the best performing model

- Select the best performing model in terms of RMSE is elasticnet1.
- It has R-squared =
- 0.7687, and RMSE =
- 304.550.

```
In [48]: ## The best model in terms of rmse is elasticnet1
rsq_elasticnet1
rmse_elasticnet1
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.768742 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 304.5503 |

# Find the best performing model

- Both models have formula:

RENTED_BIKE_COUNT ~ ., poly(TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, SNOWFALL, degree=8)

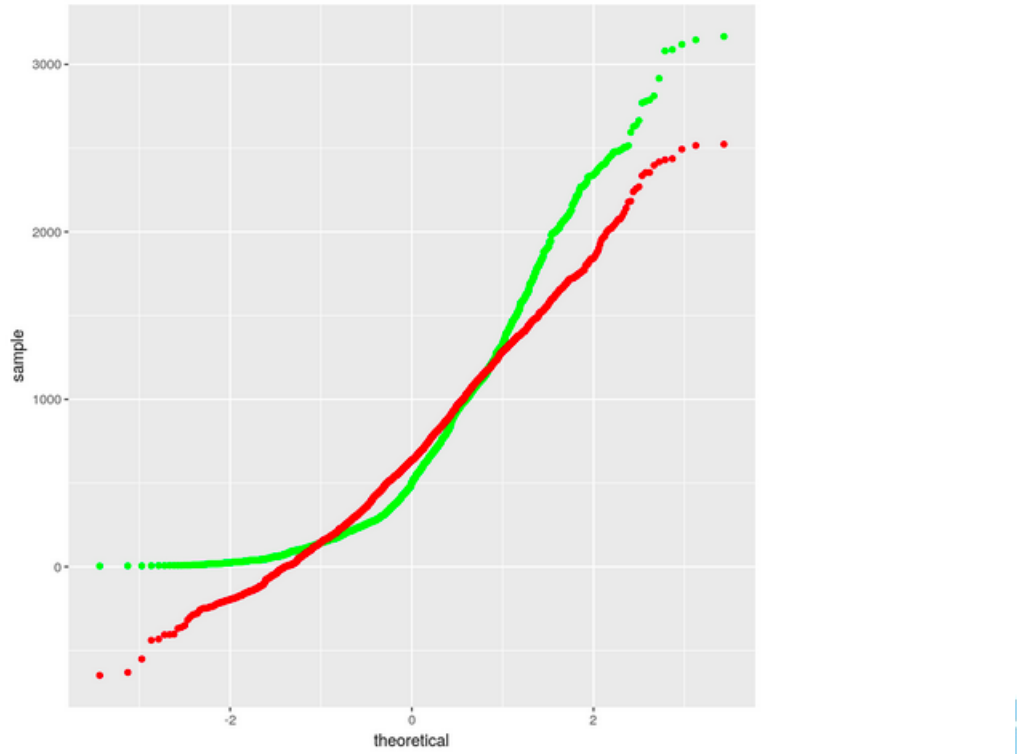
lasso1 has penalty = 1, mixture = 1;

elasticnet1 has penalty = 1, mixture = 0.5

# Q-Q plot of the best model

Q-Q plot of lasso1 (best in terms of R-squared)

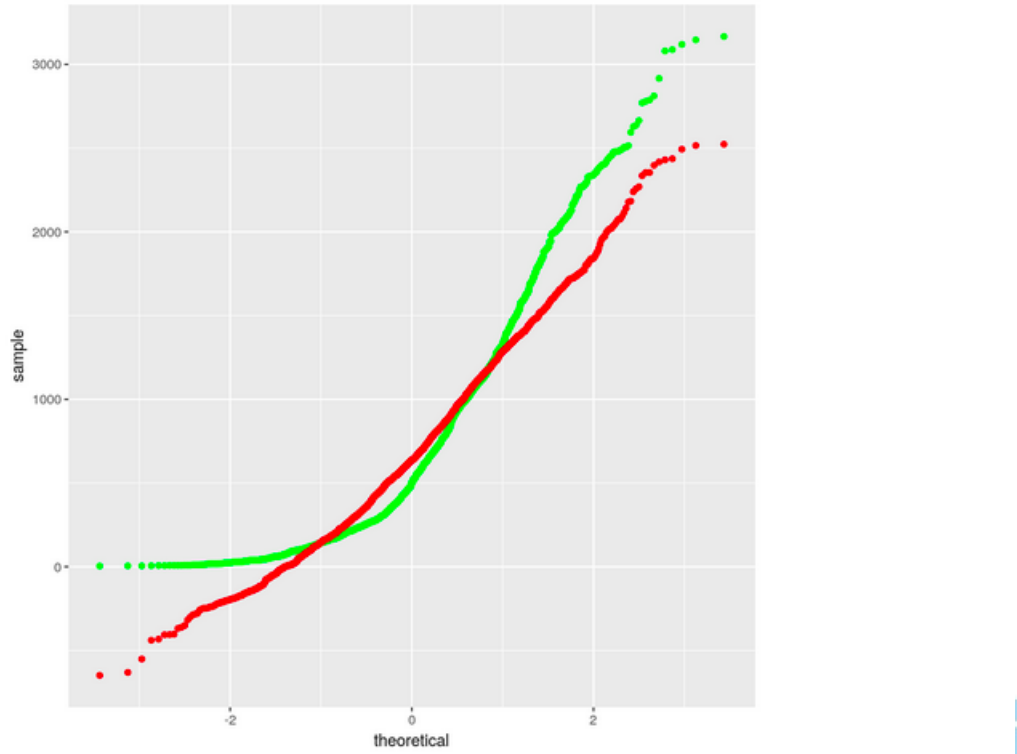
In [48]:
```
# HINT: Use ggplot() +
    # stat_qq(aes(sample=truth), color='green') +
    # stat_qq(aes(sample=prediction), color='red')
ggplot() +
  stat_qq(aes(sample=test_results_lasso1$truth), color='green') +
  stat_qq(aes(sample=test_results_lasso1$.pred), color='red')
```



# Q-Q plot of the best model

Q-Q plot of elasticnet1 (best in terms of RMSE)

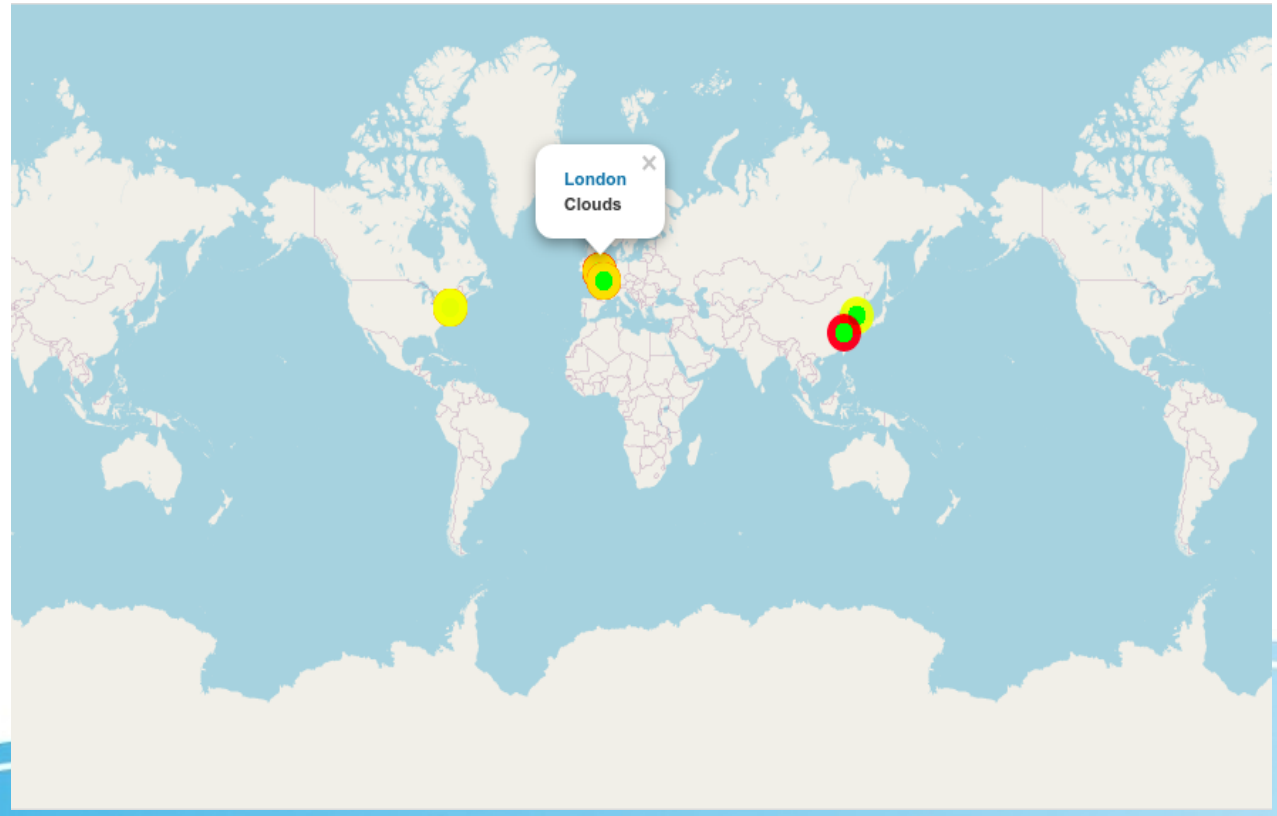
```
In [48]: # HINT: Use ggplot() +
             # stat_qq(aes(sample=truth), color='green') +
             # stat_qq(aes(sample=prediction), color='red')
         ggplot() +
           stat_qq(aes(sample=test_results_lasso1$truth), color='green') +
           stat_qq(aes(sample=test_results_lasso1$.pred), color='red')
```



# Dashboard

# Bike-sharing demand prediction app

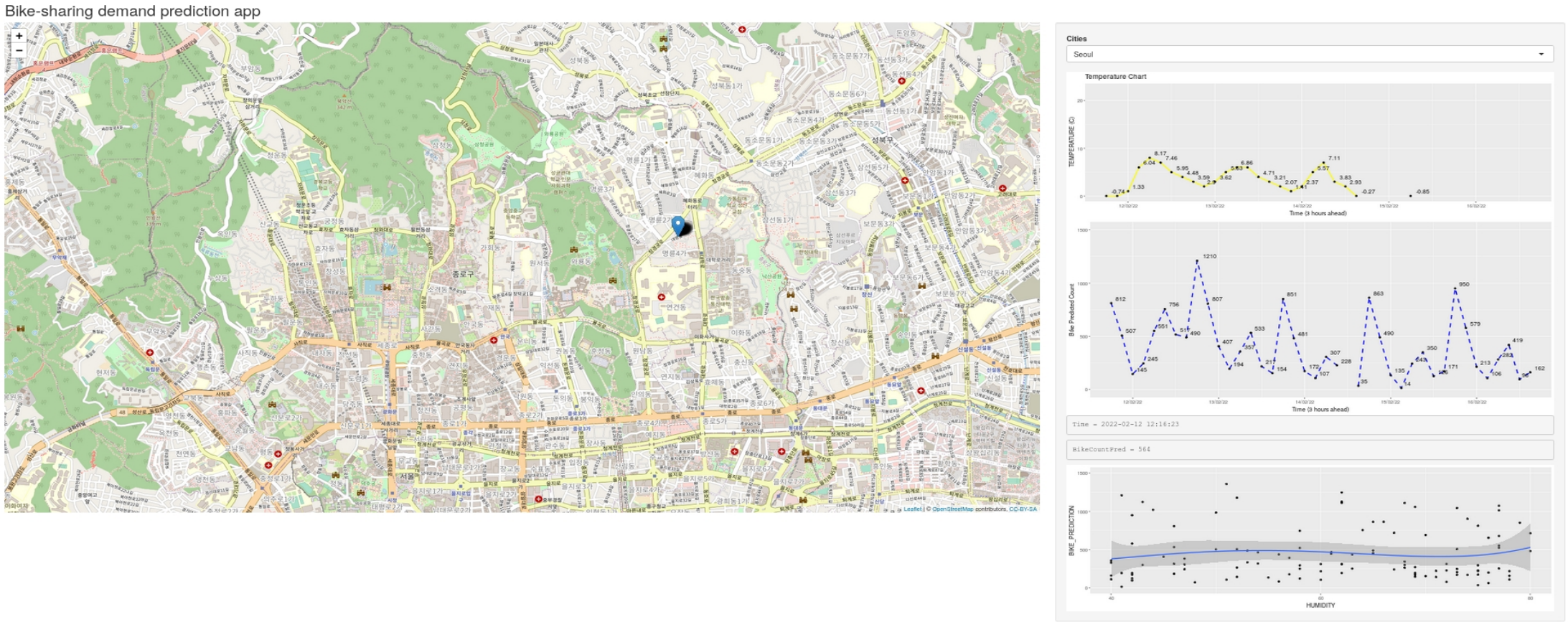
- Screenshot for cities’ max bike-sharing prediction on a map
- Each of the circles
- indicates one of
- the 5 cities,
- color indicates
- small, medium or
- large demand.



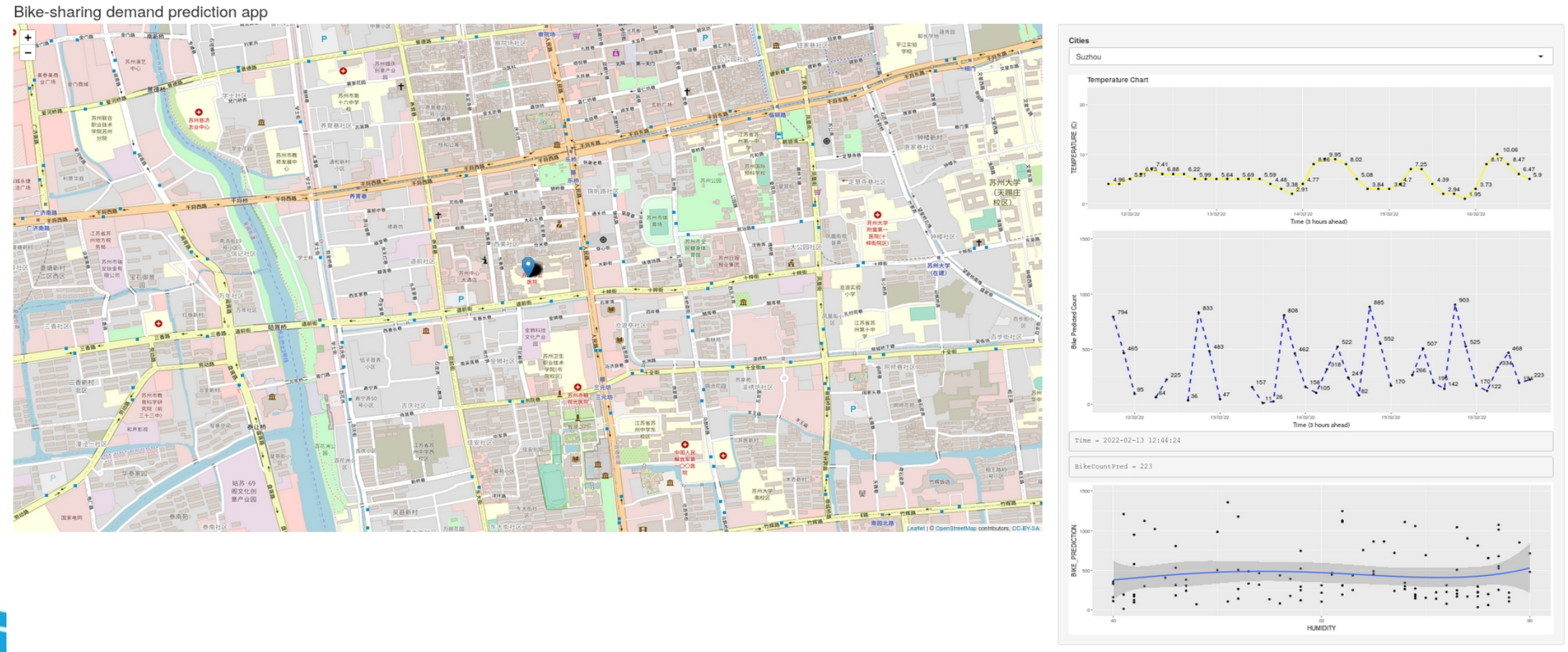
# Bike-sharing demand prediction app

- The following slides show bike-sharing demand prediction app when one of the cities is selected.
- The map on the left is the map of the selected city.
- On the right the top plot is shows temperature vs. time,
- the middle one shows bike predicted count vs. time (which enables to see time and bike predicted count as any given point),
- the bottom one shows bike prediction vs. humidity.

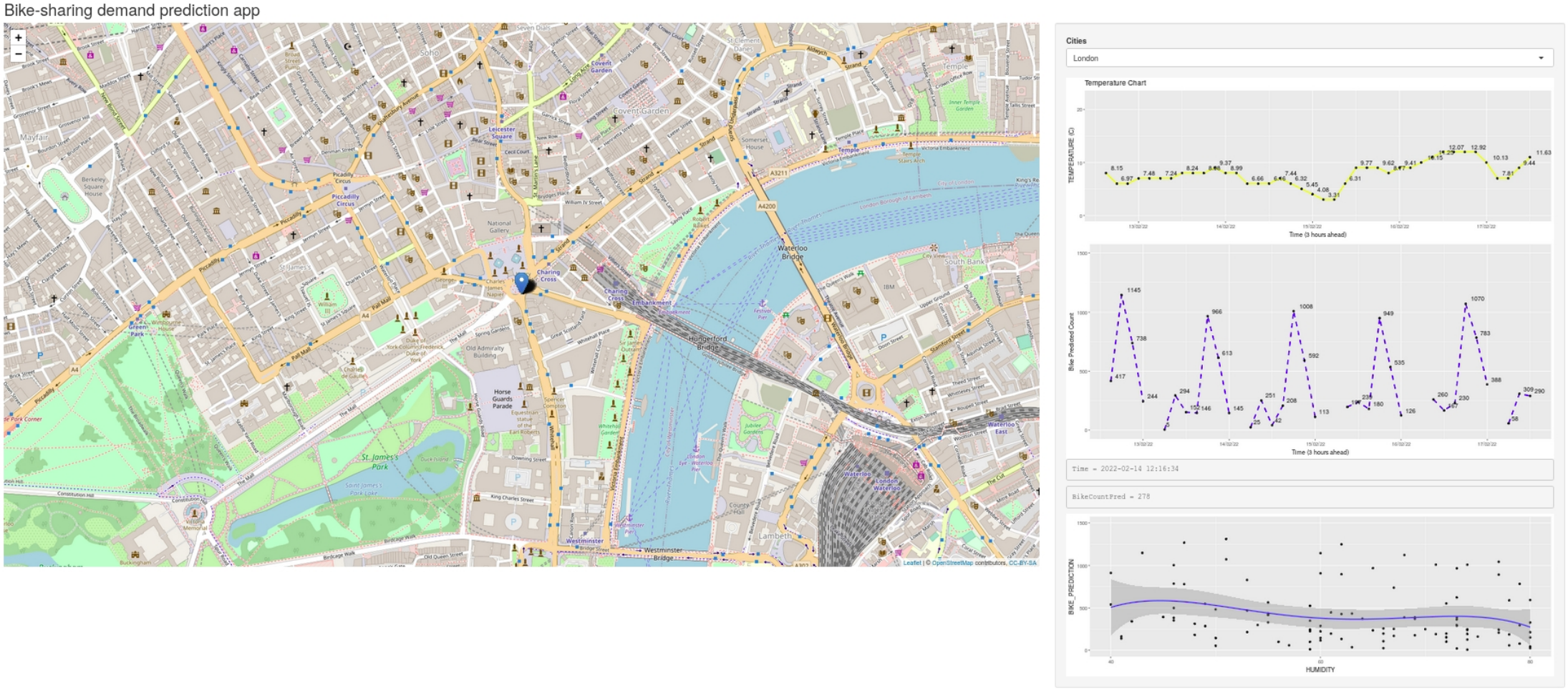
# Bike-sharing demand prediction app (Seoul is selected)



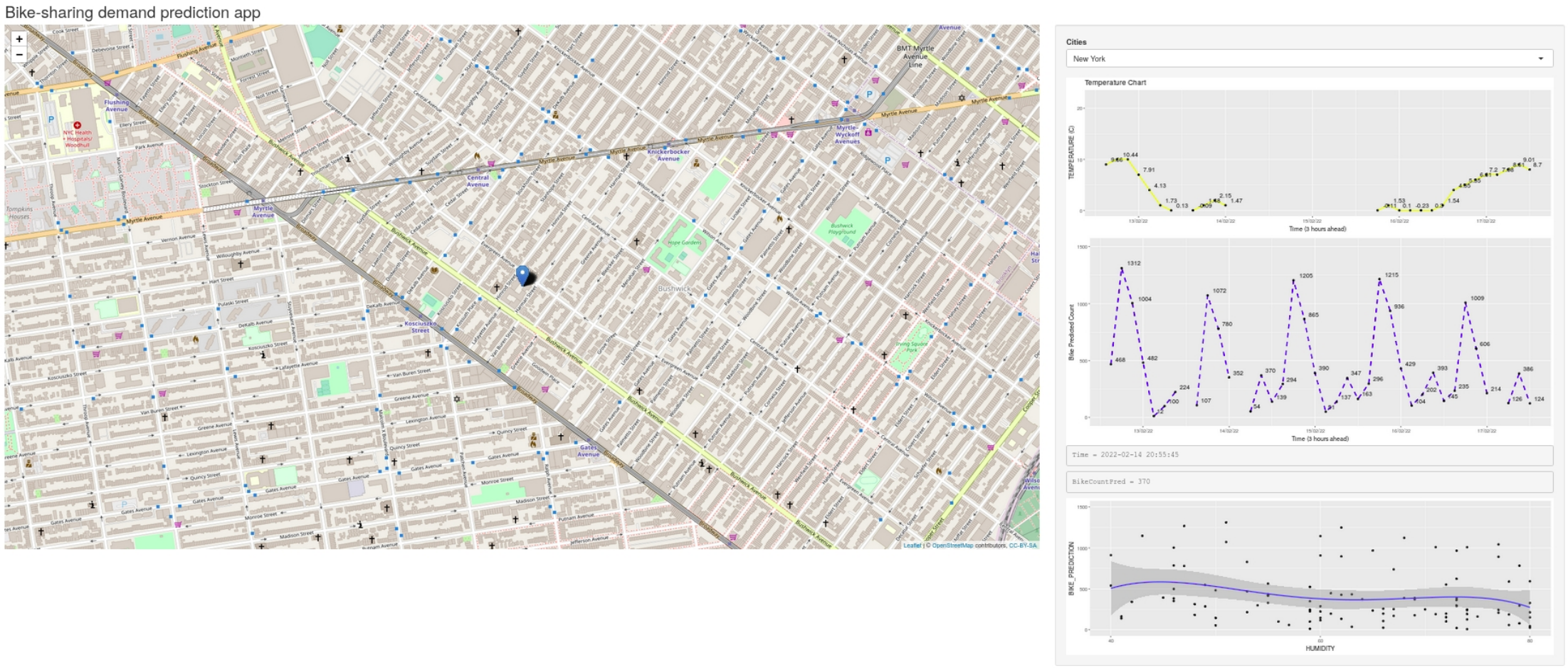
# Bike-sharing demand prediction app (Suzhou is selected)



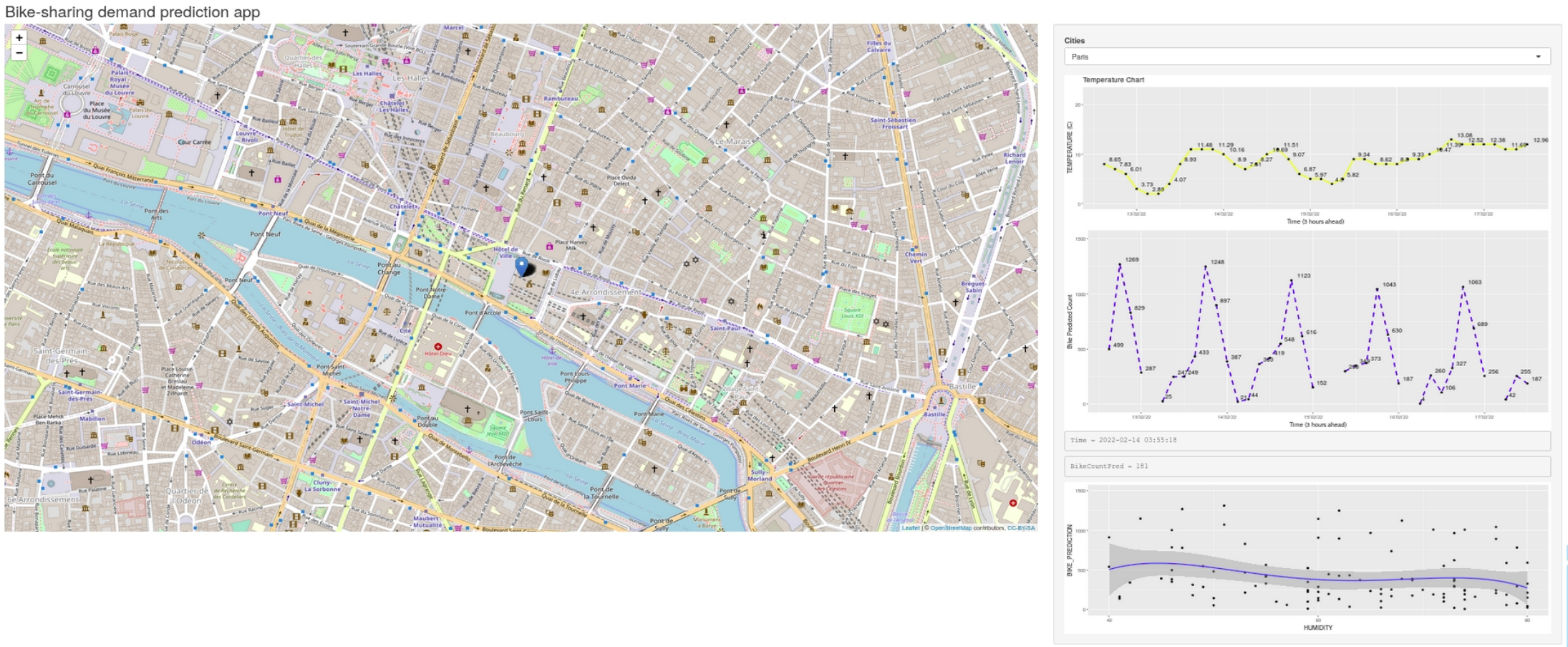
# Bike-sharing demand prediction app (London is selected)



# Bike-sharing demand prediction app (New York is selected)



# Bike-sharing demand prediction app (Paris is selected)



# CONCLUSION

- Data was collected from multiple sources.
- Data wrangling and preparation with regular expressions and Tidyverse were performed.
- Exploratory data analysis with SQL and visualization using Tidyverse and ggplot2 were performed.
- Modeling the data with linear regressions using Tidymodels was performed.

# CONCLUSION

- The best models were found:
- 1) in terms of RMSE the best model is elasticnet1, which has R-squared = 0.7687, and RMSE = 304.550;
- 2) in terms of R-square the best model is lasso 1, which has R-squared = 0.7688, and RMSE = 304.561.

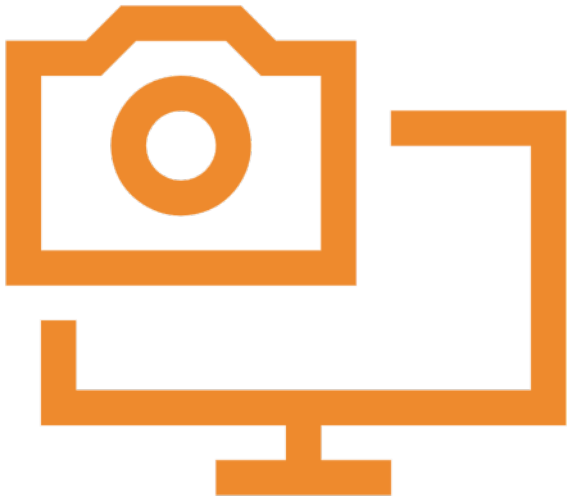
# CONCLUSION

- Both models have the formula:
- RENTED_BIKE_COUNT ~ ., poly(TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, SNOWFALL, degree=8)
- lasso1 has penalty = 1, mixture = 1;
- elasticnet1 has penalty = 1, mixture = 0.5

# CONCLUSION

- Interactive dashboard using R Shiny was built.
- It displays an interactive map and associates visualization of the current weather and the estimated bike demand.

# APPENDIX



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

# Data collection through webscraping

TASK: Extract bike sharing systems HTML table from a Wiki page and convert it into a data frame

In [2]:
```
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)
root_node
```

```
{html_document}
<html class="client-nojs" lang="en" dir="ltr">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
[2] <body class="mediawiki ltr sitedir-ltr mw-hide-empty-elt ns-0 ns-subject  ...
```

# Data collection through webscraping

In [3]:
```
# Convert the bike-sharing system table into a dataframe
table_nodes <- html_nodes(root_node, "table")
table_nodes[[2]]
df <- html_table(table_nodes[[2]], fill = TRUE)
head(df)
```

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

A tibble: 6 × 10

| Country | City | Name | System | Operator | Launched | Discontinued | Stations | Bicycles | Daily ridership |
|---|---|---|---|---|---|---|---|---|---|
| <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> |
| Albania | Tirana[5] | Ecovolis | | | March 2011 | | 8 | 200 | |
| Argentina | Buenos Aires[6][7] | Ecobici | Serttel Brasil[8] | Bike In Baires Consortium.[9] | 2010 | | 400 | 4000 | 21917 |
| Argentina | Mendoza[10] | Metrobici | | | 2014 | | 2 | 40 | |
| Argentina | Rosario | Mi Bici Tu Bici[11] | | | 2 December 2015 | | 47 | 480 | |
| Argentina | San Lorenzo, Santa Fe | Biciudad | Biciudad | | 27 November 2016 | | 8 | 80 | |
| Australia | Melbourne[12] | Melbourne Bike Share | PBSC & 8D | Motivate | June 2010 | 30 November 2019[13] | 53 | 676 | |

# Data collection through webscraping

In [4]:
```
# Summarize the dataframe
summary(df)
```

```
   Country             City               Name              System
 Length:515         Length:515         Length:515         Length:515
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
   Operator           Launched         Discontinued         Stations
 Length:515         Length:515         Length:515         Length:515
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character
   Bicycles         Daily ridership
 Length:515         Length:515
 Class :character   Class :character
 Mode  :character   Mode  :character
```

Export the data frame as a csv file called `raw_bike_sharing_systems.csv`

In [5]:
```
# Export the dataframe into a csv file
write.csv(df, "raw_bike_sharing_system.csv")
```

# Data collection through API

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

In [45]:
```
cities <- c("Seoul", "Washington, D.C.", "Paris", "Suzhou")
cities_weather_df <- get_weather_forecaset_by_cities(cities)
#cities_weather_df
```

```
Response [https://api.openweathermap.org/data/2.5/forecast?q=Seoul&appid=147101b3239a9289dc17ee40f789b7b8&units=metr
ic]
  Date: 2022-02-06 23:20
  Status: 200
  Content-Type: application/json; charset=utf-8
  Size: 15.8 kB

Response [https://api.openweathermap.org/data/2.5/forecast?q=Washington%2C%20D.C.&appid=147101b3239a9289dc17ee40f789
b7b8&units=metric]
  Date: 2022-02-06 23:20
  Status: 200
  Content-Type: application/json; charset=utf-8
  Size: 15.7 kB

Response [https://api.openweathermap.org/data/2.5/forecast?q=Paris&appid=147101b3239a9289dc17ee40f789b7b8&units=metr
ic]
  Date: 2022-02-06 23:20
  Status: 200
  Content-Type: application/json; charset=utf-8
  Size: 15.8 kB

Response [https://api.openweathermap.org/data/2.5/forecast?q=Suzhou&appid=147101b3239a9289dc17ee40f789b7b8&units=met
ric]
  Date: 2022-02-06 23:20
  Status: 200
  Content-Type: application/json; charset=utf-8
  Size: 16.1 kB
```

# Data collection through API

```
In [46]: cities_weather_df
```

A data.frame: 160 × 12

| city | weather | visibility | temp | temp_min | temp_max | pressure | humidity | wind_speed | wind_deg | forecast_datetime | season |
|---|---|---|---|---|---|---|---|---|---|---|---|
| <fct> | <fct> | <int> | <dbl> | <dbl> | <dbl> | <int> | <int> | <dbl> | <int> | <fct> | <fct> |
| Seoul | Clouds | 10000 | -10.47 | -10.47 | -4.11 | 1029 | 44 | 1.11 | 323 | 2022-02-07 00:00:00 | Winter |
| Seoul | Clouds | 10000 | -7.38 | -7.38 | -1.20 | 1029 | 40 | 2.69 | 315 | 2022-02-07 03:00:00 | Winter |
| Seoul | Clouds | 10000 | -3.54 | -3.54 | -0.07 | 1028 | 35 | 3.73 | 306 | 2022-02-07 06:00:00 | Winter |
| Seoul | Clouds | 10000 | -0.64 | -0.64 | -0.64 | 1028 | 39 | 2.85 | 317 | 2022-02-07 09:00:00 | Winter |
| Seoul | Clouds | 10000 | -1.09 | -1.09 | -1.09 | 1029 | 44 | 1.50 | 311 | 2022-02-07 12:00:00 | Winter |
| Seoul | Clouds | 10000 | -1.72 | -1.72 | -1.72 | 1028 | 47 | 1.43 | 303 | 2022-02-07 15:00:00 | Winter |
| Seoul | Clouds | 10000 | -2.58 | -2.58 | -2.58 | 1028 | 47 | 1.01 | 282 | 2022-02-07 18:00:00 | Winter |
| Seoul | Clouds | 10000 | -3.37 | -3.37 | -3.37 | 1028 | 45 | 0.80 | 300 | 2022-02-07 21:00:00 | Winter |

```
In [47]: # Write cities_weather_df to `cities_weather_forecast.csv`
write.csv(cities_weather_df, "cities_weather_forecast.csv", row.names=FALSE)
```

# Data collection through API

**TASK: Download datasets as csv files from cloud storage**

In [48]:
```
# Download several datasets

# Download some general city information such as name and locations
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsN
# download the file
download.file(url, destfile = "raw_worldcities.csv")

# Download a specific hourly Seoul bike sharing demand dataset
url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsN
# download the file
download.file(url, destfile = "raw_seoul_bike_sharing.csv")
```

# Data wrangling with regular expressions

**TASK: Standardize column names for all collected datasets**

In [13]:
```
for (dataset_name in dataset_list){
    # Read dataset
    dataset <- read_csv(dataset_name)
    # Standardized its columns:

    # Convert all column names to uppercase
    names(dataset) <- toupper(names(dataset))
    # Replace any white space separators by underscores, using the str_replace_all function
    str_replace_all(dataset, "\\s+", "_")
    # Save the dataset
    write.csv(dataset, dataset_name, row.names=FALSE)
}
```

# Data wrangling with regular expressions

In [22]:
```
for (dataset_name in dataset_list){
    # Print a summary for each data set to check whether the column names were correctly converted
    #print(summary(dataset))
    print(dataset_name)
    print(names(dataset))
}
```

```
[1] "raw_bike_sharing_systems.csv"
 [1] "CITY"       "CITY_ASCII" "LAT"        "LNG"        "COUNTRY"
 [6] "ISO2"       "ISO3"       "ADMIN_NAME" "CAPITAL"    "POPULATION"
[11] "ID"
[1] "raw_seoul_bike_sharing.csv"
 [1] "CITY"       "CITY_ASCII" "LAT"        "LNG"        "COUNTRY"
 [6] "ISO2"       "ISO3"       "ADMIN_NAME" "CAPITAL"    "POPULATION"
[11] "ID"
[1] "raw_cities_weather_forecast.csv"
 [1] "CITY"       "CITY_ASCII" "LAT"        "LNG"        "COUNTRY"
 [6] "ISO2"       "ISO3"       "ADMIN_NAME" "CAPITAL"    "POPULATION"
[11] "ID"
[1] "raw_worldcities.csv"
 [1] "CITY"       "CITY_ASCII" "LAT"        "LNG"        "COUNTRY"
 [6] "ISO2"       "ISO3"       "ADMIN_NAME" "CAPITAL"    "POPULATION"
[11] "ID"
```

# Data wrangling with regular expressions

## TASK: Remove undesired reference links using regular expressions

In [16]:
```
# remove reference link
remove_ref <- function(strings) {
    #ref_pattern <- "Define a pattern matching a reference link such as [1]"
    ref_pattern <- "\\[[\\w]+\\]"
    # Replace all matched substrings with a white space using str_replace_all()
    result <- str_replace_all(strings, ref_pattern, " ")
    # Trim the reslt if you want
    result <- trimws(result)
    return(result)
}
```

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

In [17]:
```
# sub_bike_sharing_df %>% mutate(column1=remove_ref(column1), ... )
result <- sub_bike_sharing_df %>%
#  select(CITY, SYSTEM, BICYCLES) %>%
  mutate(sub_bike_sharing_df, CITY=remove_ref(CITY), SYSTEM=remove_ref(SYSTEM), BICYCLES=remove_ref(BICYCLES))
```

*TODO:* Use the following code to check whether all reference links are removed:

In [18]:
```
#result<- sub_bike_sharing_df
result %>%
    select(CITY, SYSTEM, BICYCLES) %>%
    filter(find_reference_pattern(CITY) | find_reference_pattern(SYSTEM) | find_reference_pattern(BICYCLES))
```

A tibble: 0 × 3

| CITY | SYSTEM | BICYCLES |
|---|---|---|
| <chr> | <chr> | <chr> |

# Data wrangling with regular expressions

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

In [19]:
```
# Extract the first number
extract_num <- function(columns){
    # Define a digital pattern
    #digitals_pattern <- "Define a pattern matching a digital substring"
    digitals_pattern <- "[0-9]+"
    # Find the first match using str_extract
    first_match <- str_extract(columns, digitals_pattern)
    # Convert the result to numeric using the as.numeric() function
    first_match <- as.numeric(first_match)
}
```

*TODO:* Use the `dplyr::mutate()` function to apply `extract_num` on the `BICYCLES` column

In [20]:
```
# Use the mutate() function on the BICYCLES column
result <- result %>%
#    select(BICYCLES) %>%
    mutate(BICYCLES=extract_num(BICYCLES))
```

*TODO:* Use the summary function to check the descriptive statistics of the numeric `BICYCLES` column

In [21]:
```
#summary(result$BICYCLES)
summary(result$BICYCLES)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
      5     100     350    2022    1400   78000      78
```

*TODO:* Write the cleaned bike-sharing systems dataset into a csv file called `bike_sharing_systems.csv`

In [22]:
```
# Write dataset to `bike_sharing_systems.csv`
write_csv(result, "bike_sharing_systems.csv")
```

# Data wrangling with dplyr

## TASK: Detect and handle missing values

In [9]:
```
# Drop rows with `RENTED_BIKE_COUNT` column == NA
drop_na_rows <- bike_sharing_df %>% drop_na(RENTED_BIKE_COUNT)
dim(drop_na_rows)
head(drop_na_rows)
```

8465 · 14

A tibble: 6 × 14

| DATE | RENTED_BIKE_COUNT | HOUR | TEMPERATURE | HUMIDITY | WIND_SPEED | VISIBILITY | DEW_POINT_TEMPERATURE | SOLAR_RADIATION | RAINFALL | S |
|---|---|---|---|---|---|---|---|---|---|---|
| <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | |
| 01/12/2017 | 254 | 0 | -5.2 | 37 | 2.2 | 2000 | -17.6 | 0 | 0 | |
| 01/12/2017 | 204 | 1 | -5.5 | 38 | 0.8 | 2000 | -17.6 | 0 | 0 | |
| 01/12/2017 | 173 | 2 | -6.0 | 39 | 1.0 | 2000 | -17.7 | 0 | 0 | |
| 01/12/2017 | 107 | 3 | -6.2 | 40 | 0.9 | 2000 | -17.6 | 0 | 0 | |
| 01/12/2017 | 78 | 4 | -6.0 | 36 | 2.3 | 2000 | -18.6 | 0 | 0 | |
| 01/12/2017 | 100 | 5 | -6.4 | 37 | 1.5 | 2000 | -18.7 | 0 | 0 | |

In [10]:
```
# Print the dataset dimension again after those rows are dropped
bike_sharing_df <- drop_na_rows
dim(bike_sharing_df)
```

8465 · 14

# Data wrangling with dplyr

In [18]:
```
# Calculate the summer average temperature
summer <- filter(bike_sharing_df, SEASONS=="Summer")
summer <- summer[!(is.na(summer$TEMPERATURE)),]
summer_mean_temperature <- mean(summer$TEMPERATURE)
summer_mean_temperature
```

```
26.5877105143377
```

In [21]:
```
# Impute missing values for TEMPERATURE column with summer average temperature
bike_sharing_df <- bike_sharing_df %>% replace_na(list(TEMPERATURE=summer_mean_temperature))
```

In [22]:
```
# Print the summary of the dataset again to make sure no missing values in all columns
summary(bike_sharing_df)
```

```
     DATE           RENTED_BIKE_COUNT      HOUR        TEMPERATURE
 Length:8465        Min.   :   2.0    Min.   : 0.00   Min.   :-17.80
 Class :character   1st Qu.: 214.0    1st Qu.: 6.00   1st Qu.:  3.00
 Mode  :character   Median : 542.0    Median :12.00   Median : 13.50
                    Mean   : 729.2    Mean   :11.51   Mean   : 12.77
                    3rd Qu.:1084.0    3rd Qu.:18.00   3rd Qu.: 22.70
                    Max.   :3556.0    Max.   :23.00   Max.   : 39.40
    HUMIDITY       WIND_SPEED      VISIBILITY   DEW_POINT_TEMPERATURE
 Min.   : 0.00   Min.   :0.000   Min.   :  27   Min.   :-30.600
 1st Qu.:42.00   1st Qu.:0.900   1st Qu.: 935   1st Qu.: -5.100
 Median :57.00   Median :1.500   Median :1690   Median :  4.700
 Mean   :58.15   Mean   :1.726   Mean   :1434   Mean   :  3.945
 3rd Qu.:74.00   3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 15.200
 Max.   :98.00   Max.   :7.400   Max.   :2000   Max.   : 27.200
 SOLAR_RADIATION     RAINFALL          SNOWFALL         SEASONS
 Min.   :0.0000   Min.   : 0.0000   Min.   :0.00000   Length:8465
 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.00000   Class :character
 Median :0.0100   Median : 0.0000   Median :0.00000   Mode  :character
 Mean   :0.5679   Mean   : 0.1491   Mean   :0.07769
 3rd Qu.:0.9300   3rd Qu.: 0.0000   3rd Qu.:0.00000
 Max.   :3.5200   Max.   :35.0000   Max.   :8.80000
   HOLIDAY          FUNCTIONING_DAY
 Length:8465        Length:8465
 Class :character   Class :character
 Mode  :character   Mode  :character
```

# Data wrangling with dplyr

**TASK: Create indicator (dummy) variables for categorical variables**

In [30]:
```
# Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.
bike_sharing_df <- bike_sharing_df %>%
  mutate(dummy = 1) %>% # column with single value
  spread(
    key = SEASONS, # column to spread
    value = dummy,
    fill = 0) %>%
  mutate(dummy = 1) %>% # column with single value
  spread(
    key = HOLIDAY, # column to spread
    value = dummy,
    fill = 0) %>%
  mutate(dummy = 1) %>% # column with single value
  spread(
    key = FUNCTIONING_DAY, # column to spread
    value = dummy,
    fill = 0) %>%
  mutate(dummy = 1) %>% # column with single value
  spread(
    key = HOUR, # column to spread
    value = dummy,
    fill = 0)
```

In [31]:
```
# Print the dataset summary again to make sure the indicator columns are created properly
summary(bike_sharing_df)
```

# Data wrangling with dplyr

## TASK: Normalize data

In [35]:
```
# 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`
bike_sharing_df <- bike_sharing_df %>%
  mutate(RENTED_BIKE_COUNT=(RENTED_BIKE_COUNT - min(RENTED_BIKE_COUNT)) /
         (max(RENTED_BIKE_COUNT) - min(RENTED_BIKE_COUNT))) %>%
  mutate(TEMPERATURE=(TEMPERATURE - min(TEMPERATURE)) /
         (max(TEMPERATURE) - min(TEMPERATURE))) %>%
  mutate(HUMIDITY=(HUMIDITY - min(HUMIDITY)) /
         (max(HUMIDITY) - min(HUMIDITY))) %>%
  mutate(WIND_SPEED=(WIND_SPEED - min(WIND_SPEED)) /
         (max(WIND_SPEED) - min(WIND_SPEED))) %>%
  mutate(VISIBILITY=(VISIBILITY - min(VISIBILITY)) /
         (max(VISIBILITY) - min(VISIBILITY))) %>%
  mutate(DEW_POINT_TEMPERATURE=(DEW_POINT_TEMPERATURE - min(DEW_POINT_TEMPERATURE)) /
         (max(DEW_POINT_TEMPERATURE) - min(DEW_POINT_TEMPERATURE))) %>%
  mutate(SOLAR_RADIATION=(SOLAR_RADIATION - min(SOLAR_RADIATION)) /
         (max(SOLAR_RADIATION) - min(SOLAR_RADIATION))) %>%
  mutate(RAINFALL=(RAINFALL - min(RAINFALL)) /
         (max(RAINFALL) - min(RAINFALL))) %>%
  mutate(SNOWFALL=(SNOWFALL - min(SNOWFALL)) /
         (max(SNOWFALL) - min(SNOWFALL)))
```

In [36]:
```
# Print the summary of the dataset again to make sure the numeric columns range between 0 and 1
summary(bike_sharing_df)
```

# EDA with SQL

## Task 1 - Record Count

**Determine how many records are in the seoul_bike_sharing dataset.**

## Solution 1

In [2]:
```
# provide your solution here
query = "SELECT COUNT(*) AS NUM_OF_RECORDS FROM SEOUL_BIKE_SHARING"
sqlQuery(conn, query)
```

A data.frame: 1 × 1

| | NUM_OF_RECORDS |
|---|---|
| | <int> |
| 1 | 8465 |

# EDA with SQL

## Task 2 - Operational Hours

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

## Solution 2

In [3]:
```
# provide your solution here
query = "SELECT COUNT(*) AS NUM_OF_HOURS FROM SEOUL_BIKE_SHARING WHERE RENTED_BIKE_COUNT<>0"
sqlQuery(conn, query)
```

A data.frame: 1 × 1

| | NUM_OF_HOURS |
|---|---|
| | <int> |
| 1 | 8465 |

# EDA with SQL

## Task 3 - Weather Outlook

**Query the the weather forecast for Seoul over the next 3 hours.**

Recall that the records in the CITIES_WEATHER_FORECAST dataset are 3 hours apart, so we just need the first record from the query.

## Solution 3

In [4]:
```
# provide your solution here
query = "SELECT * FROM CITIES_WEATHER_FORECAST
WHERE CITY='Seoul' AND FORECAST_DATETIME=(SELECT MAX(FORECAST_DATETIME) FROM CITIES_WEATHER_FORECAST)"
sqlQuery(conn, query)
```

A data.frame: 1 × 12

| | CITY | WEATHER | VISIBILITY | TEMP | TEMP_MIN | TEMP_MAX | PRESSURE | HUMIDITY | WIND_SPEED | WIND_DEG | SEASON | FORECAST_DATETIME |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <fct> | <fct> | <int> | <dbl> | <dbl> | <dbl> | <int> | <int> | <dbl> | <int> | <fct> | <dttm> |
| 1 | Seoul | Clouds | 10000 | 25.1 | 25.1 | 25.1 | 1020 | 17 | 1.54 | 262 | Spring | 2021-04-21 09:00:00 |

# EDA with SQL

### Task 4 - Seasons

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

### Solution 4

```
In [5]: # provide your solution here
query = "SELECT DISTINCT SEASONS FROM SEOUL_BIKE_SHARING"
sqlQuery(conn, query)
```

A data.frame: 4 × 1

| | SEASONS <fct> |
|---|---|
| 1 | Autumn |
| 2 | Spring |
| 3 | Summer |
| 4 | Winter |

# EDA with SQL

### Task 5 - Date Range

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

### Solution 5

In [6]:
```
# provide your solution here
query = "SELECT MIN(DATE) AS FIRST_DATE, MAX(DATE) AS LAST_DATE FROM SEOUL_BIKE_SHARING"
sqlQuery(conn, query)
```

A data.frame: 1 × 2

| | FIRST_DATE | LAST_DATE |
|---|---|---|
| | <fct> | <fct> |
| 1 | 2017-01-12 | 2018-12-11 |

# EDA with SQL

### Task 6 - Subquery - 'all-time high' ¶

**determine which date and hour had the most bike rentals.**

### Solution 6

In [7]:
```
# provide your solution here
query = "SELECT DATE, HOUR, RENTED_BIKE_COUNT FROM SEOUL_BIKE_SHARING
WHERE RENTED_BIKE_COUNT=(SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)"
sqlQuery(conn, query)
```

A data.frame: 1 × 3

| | DATE | HOUR | RENTED_BIKE_COUNT |
|---|---|---|---|
| | <fct> | <int> | <int> |
| 1 | 2018-06-19 | 18 | 3556 |

# EDA with SQL

## Task 7 - Hourly popularity and temperature by season

**Determine the average hourly temperature and the average number of bike rentals per hour over each season. List the top ten results by average bike count.**

## Solution 7

In [8]:
```
# provide your solution here
query = "SELECT SEASONS, AVG(RENTED_BIKE_COUNT) AS AVG_RENTALS, AVG(TEMPERATURE) AS AVG_TEMP
FROM SEOUL_BIKE_SHARING GROUP BY SEASONS"
sqlQuery(conn, query)
```

A data.frame: 4 × 3

| | SEASONS | AVG_RENTALS | AVG_TEMP |
|---|---|---|---|
| | <fct> | <int> | <dbl> |
| 1 | Autumn | 924 | 13.821167 |
| 2 | Spring | 746 | 13.021389 |
| 3 | Summer | 1034 | 26.587274 |
| 4 | Winter | 225 | -2.540463 |

# EDA with SQL

## Task 8 - Rental Seasonality ¶

**Find the average hourly bike count during each season.**

Also include the minimum, maximum, and standard deviation of the hourly bike count for each season.

### Solution 8

In [9]:
```
# provide your solution here
query = "SELECT SEASONS, AVG(RENTED_BIKE_COUNT) AS AVG_COUNT, MIN(RENTED_BIKE_COUNT) AS MIN_COUNT,
MAX(RENTED_BIKE_COUNT) AS MAX_COUNT, STDDEV(RENTED_BIKE_COUNT) AS STDDEV_COUNT
FROM SEOUL_BIKE_SHARING GROUP BY SEASONS"
sqlQuery(conn, query)
```

A data.frame: 4 × 5

| | SEASONS | AVG_COUNT | MIN_COUNT | MAX_COUNT | STDDEV_COUNT |
|---|---|---|---|---|---|
| | <fct> | <int> | <int> | <int> | <dbl> |
| 1 | Autumn | 924 | 2 | 3298 | 617.3885 |
| 2 | Spring | 746 | 2 | 3251 | 618.5247 |
| 3 | Summer | 1034 | 9 | 3556 | 690.0884 |
| 4 | Winter | 225 | 3 | 937 | 150.3374 |

# EDA with SQL

## 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.

## Solution 9

In [10]:
```
# provide your solution here
query = "SELECT SEASONS, AVG(TEMPERATURE) AS AVG_TEMP, AVG(HUMIDITY) AS AVG_HUMID, AVG(WIND_SPEED) AS AVG_WIND,
AVG(VISIBILITY) AS AVG_VIS, AVG(DEW_POINT_TEMPERATURE) AS AVG_DEW_POINT, AVG(SOLAR_RADIATION) AS AVG_RADIATION,
AVG(RAINFALL) AS AVG_RAIN, AVG(SNOWFALL) AS AVG_SNOW, AVG(RENTED_BIKE_COUNT) AS AVG_COUNT
FROM SEOUL_BIKE_SHARING GROUP BY SEASONS"
sqlQuery(conn, query)
```

A data.frame: 4 × 10

| | SEASONS | AVG_TEMP | AVG_HUMID | AVG_WIND | AVG_VIS | AVG_DEW_POINT | AVG_RADIATION | AVG_RAIN | AVG_SNOW | AVG_COUNT |
|---|---|---|---|---|---|---|---|---|---|---|
| | <fct> | <dbl> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl> | <dbl> | <int> |
| 1 | Autumn | 13.821167 | 59 | 1.492101 | 1558 | 5.150594 | 0.5227827 | 0.11765617 | 0.06350026 | 924 |
| 2 | Spring | 13.021389 | 58 | 1.857778 | 1240 | 4.091389 | 0.6803009 | 0.18694444 | 0.00000000 | 746 |
| 3 | Summer | 26.587274 | 64 | 1.609420 | 1501 | 18.750136 | 0.7612545 | 0.25348732 | 0.00000000 | 1034 |
| 4 | Winter | -2.540463 | 49 | 1.922685 | 1445 | -12.416667 | 0.2981806 | 0.03282407 | 0.24750000 | 225 |

# EDA with SQL

## 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.

## Solution 10

In [11]:
```
# provide your solution here
query = "SELECT W.CITY, W.COUNTRY, W.LAT, W.LNG, W.POPULATION, B.BICYCLES FROM WORLD_CITIES W, BIKE_SHARING_SYSTEMS B
WHERE W.CITY='Seoul' AND W.CITY=B.CITY"
sqlQuery(conn, query)
```

A data.frame: 1 × 6

| | CITY | COUNTRY | LAT | LNG | POPULATION | BICYCLES |
|---|---|---|---|---|---|---|
| | <fct> | <fct> | <dbl> | <dbl> | <int> | <int> |
| 1 | Seoul | Korea, South | 37.58 | 127 | 21794000 | 20000 |

# EDA with SQL

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

**Find all cities with total bike counts between 15000 and 20000. Return the city and country names, plus the coordinates (LAT, LNG), population, and number of bicycles for each city.**

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

### Solution 11

In [12]:
```
# provide your solution here
query = "SELECT W.CITY, W.COUNTRY, W.LAT, W.LNG, W.POPULATION, B.BICYCLES FROM WORLD_CITIES W, BIKE_SHARING_SYSTEMS B
WHERE B.BICYCLES>=15000 AND B.BICYCLES<='20000' AND W.CITY=B.CITY"
sqlQuery(conn, query)
```

A data.frame: 6 × 6

| | CITY | COUNTRY | LAT | LNG | POPULATION | BICYCLES |
|---|---|---|---|---|---|---|
| | <fct> | <fct> | <dbl> | <dbl> | <int> | <int> |
| 1 | Shanghai | China | 31.16 | 121.46 | 22120000 | 19165 |
| 2 | Seoul | Korea, South | 37.58 | 127.00 | 21794000 | 20000 |
| 3 | Beijing | China | 39.90 | 116.39 | 19433000 | 16000 |
| 4 | Weifang | China | 36.71 | 119.10 | 9373000 | 20000 |
| 5 | Ningbo | China | 29.87 | 121.54 | 7639000 | 15000 |
| 6 | Zhuzhou | China | 27.84 | 113.14 | 3855609 | 20000 |

# EDA with visualization

### Task 1 - Load the dataset

Ensure you read `DATE` as type `character`.

### Solution 1

In [95]:
```
# provide your solution here
seoul_bike_sharing <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-R
df <- read.csv(seoul_bike_sharing)
df$DATE <- as.character(df$DATE)
str(df$DATE)
str(df)
```

```
 chr [1:8465] "01/12/2017" "01/12/2017" "01/12/2017" "01/12/2017" ...
'data.frame':   8465 obs. of  14 variables:
 $ DATE                 : chr  "01/12/2017" "01/12/2017" "01/12/2017" "01/12/2017" ...
 $ RENTED_BIKE_COUNT    : int  254 204 173 107 78 100 181 460 930 490 ...
 $ HOUR                 : int  0 1 2 3 4 5 6 7 8 9 ...
 $ TEMPERATURE          : num  -5.2 -5.5 -6 -6.2 -6 -6.4 -6.6 -7.4 -7.6 -6.5 ...
 $ HUMIDITY             : int  37 38 39 40 36 37 35 38 37 27 ...
 $ WIND_SPEED           : num  2.2 0.8 1 0.9 2.3 1.5 1.3 0.9 1.1 0.5 ...
 $ VISIBILITY           : int  2000 2000 2000 2000 2000 2000 2000 2000 2000 1928 ...
 $ DEW_POINT_TEMPERATURE: num  -17.6 -17.6 -17.7 -17.6 -18.6 -18.7 -19.5 -19.3 -19.8 -22.4 ...
 $ SOLAR_RADIATION      : num  0 0 0 0 0 0 0 0 0.01 0.23 ...
 $ RAINFALL             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ SNOWFALL             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ SEASONS              : Factor w/ 4 levels "Autumn","Spring",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ HOLIDAY              : Factor w/ 2 levels "Holiday","No Holiday": 2 2 2 2 2 2 2 2 2 2 ...
 $ FUNCTIONING_DAY      : Factor w/ 1 level "Yes": 1 1 1 1 1 1 1 1 1 1 ...
```

# EDA with visualization

### Task 2 - Recast `DATE` as a date

Use the format of the data, namely "%d/%m/%Y".

### Solution 2 ¶

In [138]:
```
# provide your solution here
df$DATE <- strptime(df$DATE, "%d/%m/%Y")
df$DATE <- format(df$DATE, "%Y-%m-%d")
df$DATE <- as.Date(df$DATE)
head(df)
```

A data.frame: 6 × 14

| | DATE | RENTED_BIKE_COUNT | HOUR | TEMPERATURE | HUMIDITY | WIND_SPEED | VISIBILITY | DEW_POINT_TEMPERATURE | SOLAR_RADIATION | RAINFALL |
|---|---|---|---|---|---|---|---|---|---|---|
| | <date> | <int> | <int> | <dbl> | <int> | <dbl> | <int> | <dbl> | <dbl> | <dbl |
| 1 | 2017-12-01 | 254 | 0 | -5.2 | 37 | 2.2 | 2000 | -17.6 | 0 | |
| 2 | 2017-12-01 | 204 | 1 | -5.5 | 38 | 0.8 | 2000 | -17.6 | 0 | |
| 3 | 2017-12-01 | 173 | 2 | -6.0 | 39 | 1.0 | 2000 | -17.7 | 0 | |
| 4 | 2017-12-01 | 107 | 3 | -6.2 | 40 | 0.9 | 2000 | -17.6 | 0 | |
| 5 | 2017-12-01 | 78 | 4 | -6.0 | 36 | 2.3 | 2000 | -18.6 | 0 | |
| 6 | 2017-12-01 | 100 | 5 | -6.4 | 37 | 1.5 | 2000 | -18.7 | 0 | |

# EDA with visualization

### 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.

### Solution 3

In [3]:
```
library(tidyverse)
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──

✔ ggplot2 3.3.5     ✔ purrr   0.3.4
✔ tibble  3.1.6     ✔ dplyr   1.0.7
✔ tidyr   1.1.4     ✔ stringr 1.4.0
✔ readr   2.1.2     ✔ forcats 0.5.1

── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

In [4]:
```
# provide your solution here
df$HOUR <- as.factor(df$HOUR)
str(df$HOUR)
```

```
 Factor w/ 24 levels "0","1","2","3",..: 1 2 3 4 5 6 7 8 9 10 ...
```

# EDA with visualization

**Task 4 - Dataset Summary**

Use the base R `sumamry()` function to describe the `seoul_bike_sharing` dataset.

**Solution 4**

```
In [8]: # provide your solution here
summary(seoul_bike_sharing)
```

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

 SOLAR_RADIATION     RAINFALL          SNOWFALL         SEASONS
 Min.   :0.0000   Min.   : 0.0000   Min.   :0.00000   Autumn:1937
 1st Qu.:0.0000   1st Qu.: 0.0000   1st Qu.:0.00000   Spring:2160
 Median :0.0100   Median : 0.0000   Median :0.00000   Summer:2208
 Mean   :0.5679   Mean   : 0.1491   Mean   :0.07769   Winter:2160
 3rd Qu.:0.9300   3rd Qu.: 0.0000   3rd Qu.:0.00000
 Max.   :3.5200   Max.   :35.0000   Max.   :8.80000

      HOLIDAY     FUNCTIONING_DAY
 Holiday   : 408   Yes:8465
 No Holiday:8057
```

# EDA with visualization

**Task 5 - Based on the above stats, calculate how many Holidays there are.**

**Solution 5:**

In [9]:
```
# provide your solution here
df %>%
  count(HOLIDAY=='Holiday')
```

A data.frame: 2 × 2

| HOLIDAY == "Holiday" | n |
|---|---|
| <lgl> | <int> |
| FALSE | 8057 |
| TRUE | 408 |

# EDA with visualization

**Task 6 - Calculate the percentage of records that fall on a holiday.**

**Solution 6**

In [10]:
```
# provide your solution here
holidays <- df %>%
  count(HOLIDAY=='Holiday')
no_holidays <- df %>%
  count(HOLIDAY!='Holiday')
percentage <- holidays[2,2] * 100 / (holidays[2,2] + no_holidays[2,2])
percentage
```

4.8198464264619

**Task 7 - Given there is exactly a full year of data, determine how many records we expect to have.**

**Solution 7**

In [29]:
```
365*24
```

8760

# EDA with visualization

**Task 8 - Given the observations for the 'FUNCTIONING_DAY' how many records must there be?** ¶

**Solution 8**

In [14]:
```
# provide your solution here
df %>%
  count(FUNCTIONING_DAY)
```

A data.frame: 1 × 2

| FUNCTIONING_DAY | n |
|---|---|
| <fct> | <int> |
| Yes | 8465 |

# EDA with visualization

**Task 9 - Load the dplyr package, group the data by `SEASONS`, and use the `summarize()` function to calculate the seasonal total rainfall and snowfall.**

**Solution 9**

In [15]:
```r
# provide your solution here
library(dplyr)
by_seasons <- df %>%
  group_by(SEASONS) %>%
  summarize(total_rainfall=sum(RAINFALL), total_snowfall=sum(SNOWFALL))
```

In [16]:
```r
by_seasons
```

A tibble: 4 × 3

| SEASONS | total_rainfall | total_snowfall |
|---|---|---|
| <fct> | <dbl> | <dbl> |
| Autumn | 227.9 | 123.0 |
| Spring | 403.8 | 0.0 |
| Summer | 559.7 | 0.0 |
| Winter | 70.9 | 534.6 |

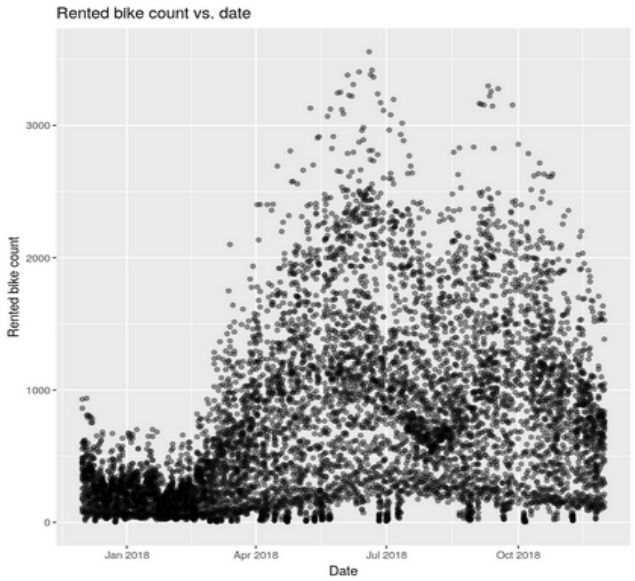
# EDA with visualization

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

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

**Solution 10**

In [18]:
```r
# provide your solution here
df_plot <- df ## %>%
ggplot(df_plot, aes(x = DATE, y = RENTED_BIKE_COUNT)) +
  geom_point(alpha = 0.4) +
  labs(x = "Date", y = "Rented bike count",
       title = "Rented bike count vs. date")
```

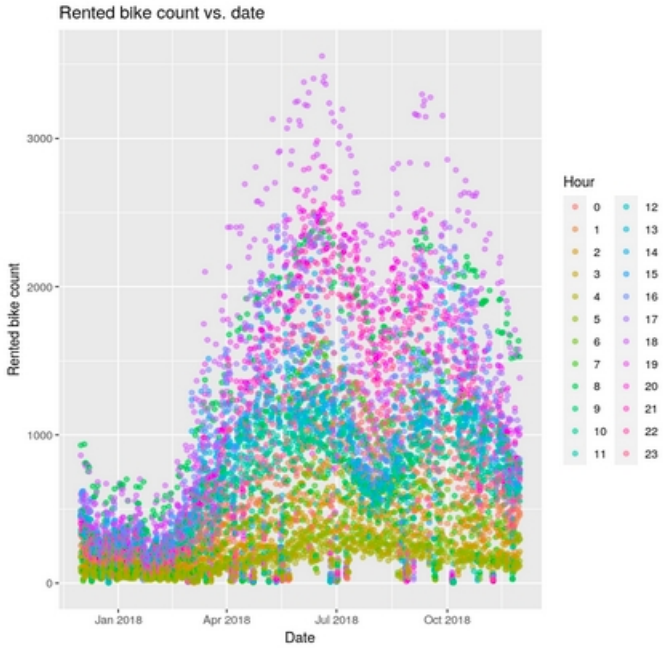


# EDA with visualization

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

**Solution 11**

In [20]:
```r
# provide your solution here
df_plot$HOUR <- as.factor(df_plot$HOUR)
ggplot(df_plot, aes(x = DATE, y = RENTED_BIKE_COUNT, color = HOUR)) +
  geom_point(alpha = 0.5) +
  labs(x = "Date", y = "Rented bike count",
       color = "Hour",
       title = "Rented bike count vs. date")
```



# EDA with visualization

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

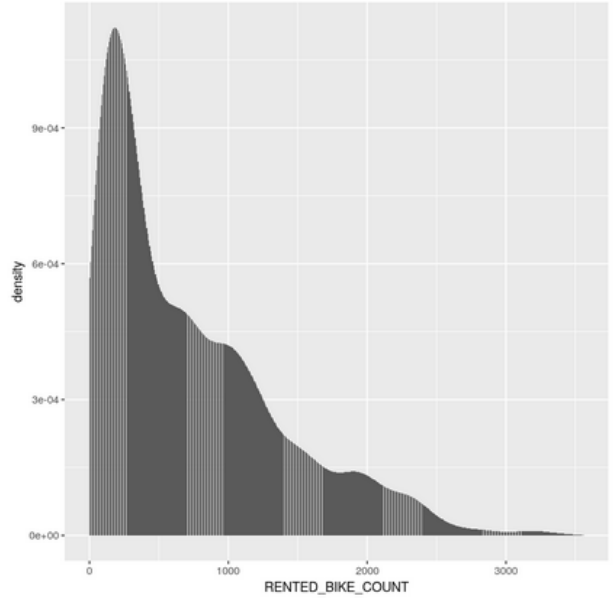
Normalize the histogram so the y axis represents 'density'. This can be done by setting `y=..density..` in the aesthetics of the histogram.

Click here for a hint

Click here for another hint

**Solution 12**

In [21]:
```
#ggplot(df_plot, aes(x = RENTED_BIKE_COUNT, y=..density..)) +
ggplot(df_plot, aes(x = RENTED_BIKE_COUNT)) +
  geom_bar(stat="density")
```



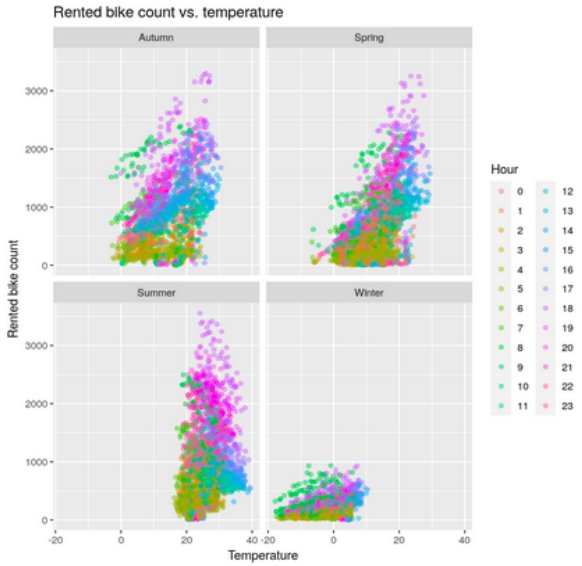
# EDA with visualization

**Task 13 - Use a scatter plot to visualize the correlation between `RENTED_BIKE_COUNT` and `TEMPERATURE` by `SEASONS`.** ¶

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.

**Solution 13**

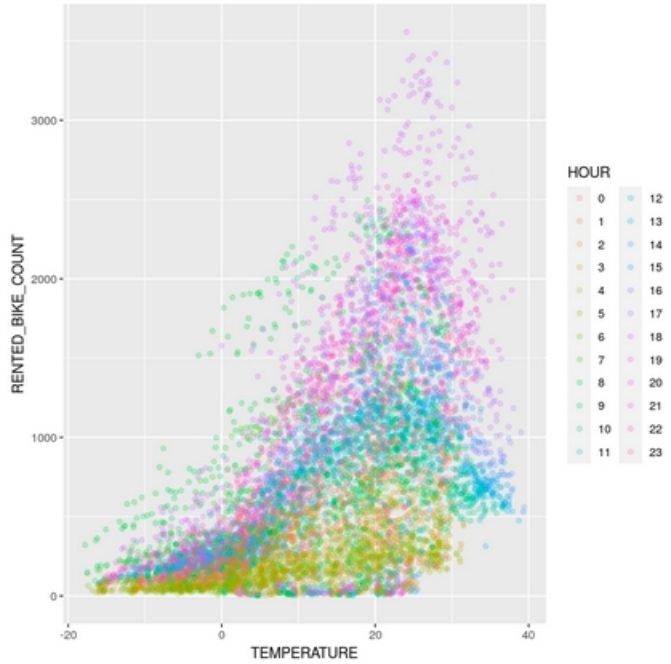
In [22]:
```
# provide your solution here
ggplot(df_plot, aes(x = TEMPERATURE, y = RENTED_BIKE_COUNT, color = HOUR)) +
  geom_point(alpha = 0.4) +
  facet_wrap(~SEASONS) +
  labs(x = "Temperature", y = "Rented bike count",
       color = "Hour",
       title = "Rented bike count vs. temperature")
```



# EDA with visualization

Comparing this plot to the same plot below, but without grouping by `SEASONS`, shows how important seasonality is in explaining bike rental counts.

In [23]:
```
seoul_bike_sharing <- df_plot
ggplot(seoul_bike_sharing) +
   geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=1/5)
```



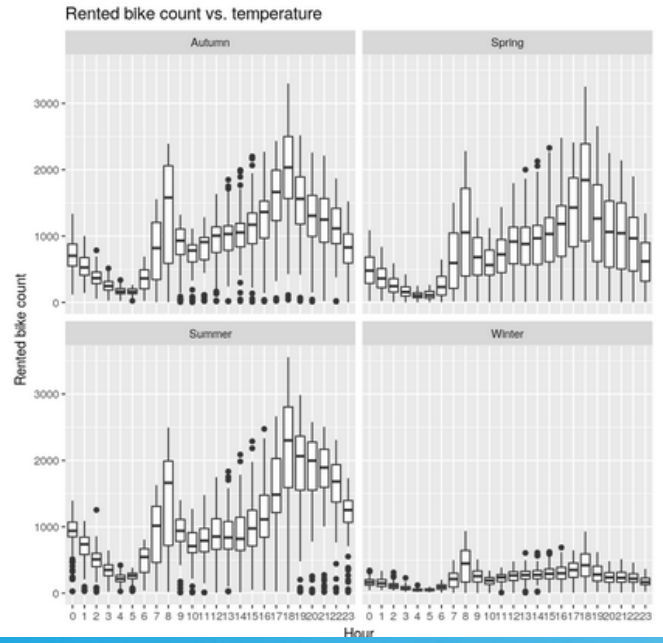
# EDA with visualization

**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.

**Solution 14**

In [24]:
```
# provide your solution here
ggplot(df_plot, aes(x = HOUR, y = RENTED_BIKE_COUNT)) +
  geom_boxplot() +
  facet_wrap(~SEASONS) +
  labs(x = "Hour", y = "Rented bike count",
       color = "Hour",
       title = "Rented bike count vs. temperature")
```



# EDA with visualization

**Task 15 - Group the data by `DATE`, and use the summarize() function to calculate the daily total rainfall and snowfall.** ¶

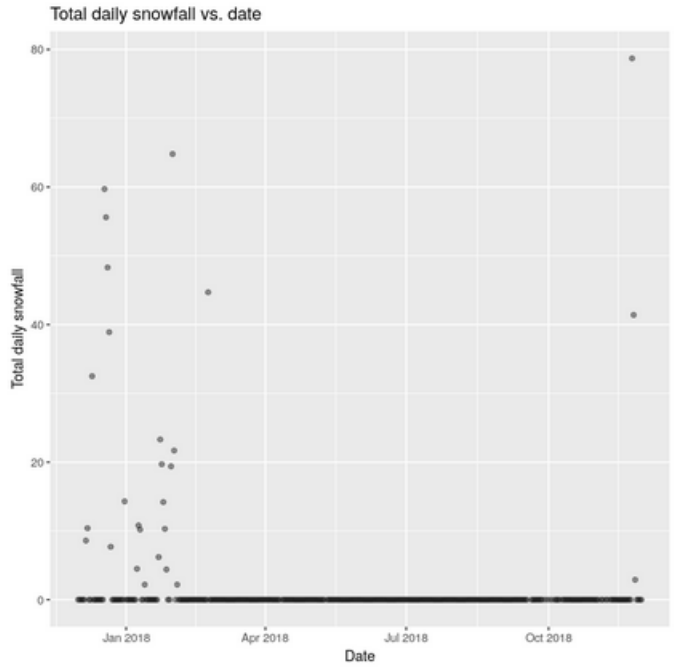
Also, go ahead and plot the results if you wish.

**Solution 15**

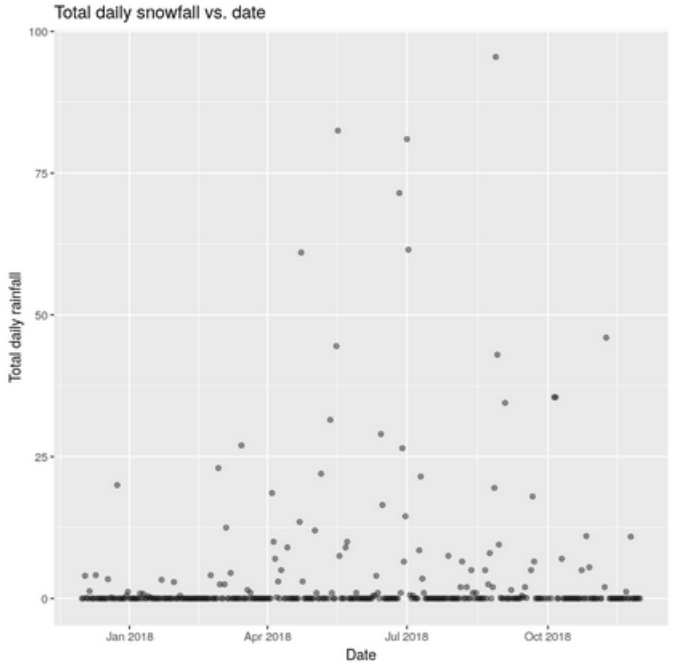
In [25]:
```
# provide your solution here
by_date <- df %>%
  group_by(DATE) %>%
  summarize(daily_rainfall=sum(RAINFALL), daily_snowfall=sum(SNOWFALL))
```

# EDA with visualization

In [26]:
```
ggplot(by_date, aes(x = DATE, y = daily_snowfall)) +
  geom_point(alpha = 0.4) +
  labs(x = "Date", y = "Total daily snowfall",
       title = "Total daily snowfall vs. date")
```




In [27]:
```
ggplot(by_date, aes(x = DATE, y = daily_rainfall)) +
  geom_point(alpha = 0.4) +
  labs(x = "Date", y = "Total daily rainfall",
       title = "Total daily snowfall vs. date")
```



# EDA with visualization

**Task 16 - Determine how many days had snowfall.**

**Solution 16**

In [28]:
```
# provide your solution here
by_date %>%
  count(daily_snowfall>0.0)
```

A tibble: 2 × 2

| daily_snowfall > 0 | n |
|---|---|
| <lgl> | <int> |
| FALSE | 326 |
| TRUE | 27 |

# Predictive analysis

## TASK: Split training and testing data ¶

First, we need to split the full dataset into training and testing datasets.

The training dataset will be used for fitting regression models, and the testing dataset will be used to evaluate the trained models.

*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%.

In [5]:
```
# Use the `initial_split()`, `training()`, and `testing()` functions to split the dataset
# With seed 1234
set.seed(1234)
# prop = 3/4
# train_data
# test_data
bike_sharing_split <- initial_split(bike_sharing_df, prop = 0.75)
train_data <- training(bike_sharing_split)
test_data <- testing(bike_sharing_split)
```

# Predictive analysis

## TASK: Build a linear regression model using weather variables only

Define a linear regression model specification.

In [6]:
```
# Use `linear_reg()` with engine `lm` and mode `regression`
# Pick linear regression
lm_model_weather <- linear_reg() %>%
  # Set engine
  #set_engine(engine = "lm", mode="regression")
  set_engine(engine = "lm") %>%
  set_mode(mode = "regression")

# Print the linear function
lm_model_weather
```

```
Linear Regression Model Specification (regression)

Computational engine: lm
```

Fit a model with the response variable `RENTED_BIKE_COUNT` and predictor variables `TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE + SOLAR_RADIATION + RAINFALL + SNOWFALL`

In [7]:
```
# Fit the model called `lm_model_weather`
# RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE +
# SOLAR_RADIATION + RAINFALL + SNOWFALL,  with the training data
lm_model_weather <- lm_model_weather %>%
  fit(RENTED_BIKE_COUNT ~ TEMPERATURE + HUMIDITY + WIND_SPEED + VISIBILITY + DEW_POINT_TEMPERATURE +
      SOLAR_RADIATION + RAINFALL + SNOWFALL, data = train_data)
```

Print the fit summary for the `lm_model_weather` model.

In [8]:
```
# print(lm_model_weather$fit)
summary(lm_model_weather$fit)
```

# Predictive analysis

**TASK: Build a linear regression model using all variables**

In [9]:
```
# Fit the model called `lm_model_all`
# `RENTED_BIKE_COUNT ~ .` means use all other variables except for the response variable

lm_model_all <- linear_reg() %>%
  # Set engine
  set_engine(engine = "lm") %>%
  set_mode(mode="regression")

lm_model_all <- lm_model_all %>%
  fit(RENTED_BIKE_COUNT ~ ., data = train_data)
```

Print the fit summary for `lm_model_all`.

In [10]:
```
# summary(lm_model_all$fit)
summary(lm_model_all$fit)
```

# Predictive analysis

**TASK: Model evaluation and identification of important variables**

In [12]:
```
# rsq_weather <- rsq(...)
rsq(test_results_weather, truth = truth,
    estimate = .pred)
# rsq_all <- rsq(...)
rsq(test_results_all, truth = truth,
    estimate = .pred)

# rmse_weather <- rmse(...)
rmse(test_results_weather, truth = truth,
     estimate = .pred)
# rmse_all <- rmse(...)
rmse(test_results_all, truth = truth,
     estimate = .pred)
```

A tibble: 1 × 3

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

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 | 474.6247 |

A tibble: 1 × 3

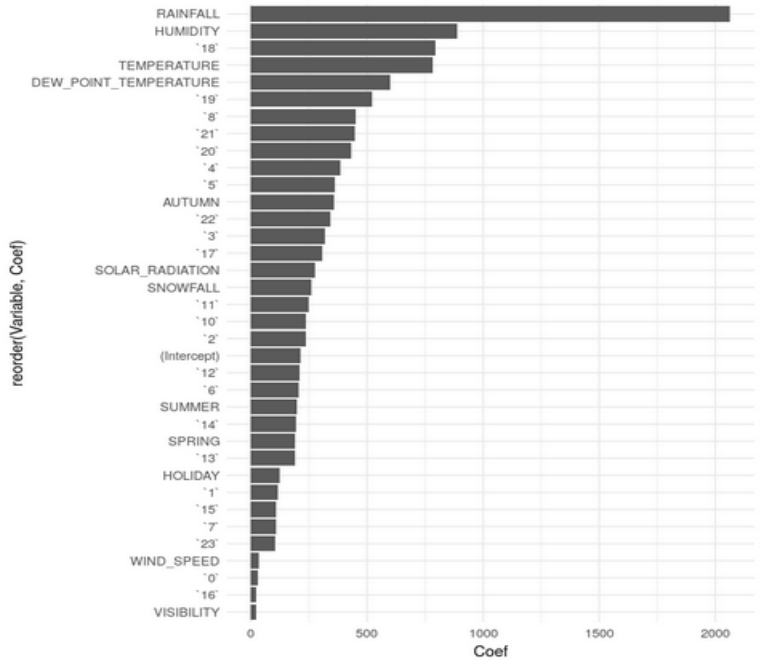
| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 364.4235 |

# Predictive analysis

*TODO:* Sort the coefficient list in descending order and visualize the result using `ggplot` and `geom_bar`

In [14]:
```
# Visualize the list using ggplot and geom_bar
abs_cof_df <- stack(abs(lm_model_all$fit$coefficients))
names(abs_cof_df) <- c("Coef", "Variable")
abs_cof_df <- abs_cof_df %>%
                select(Variable, Coef)
coefs_sorted <- arrange(abs_cof_df, -Coef)
coefs_sorted <- na.omit(coefs_sorted)

# Draw bar charts for the sorted coefficients
ggplot(data=coefs_sorted, aes(x= reorder(Variable,Coef),Coef)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal()
```



# Predictive analysis

## TASK: Add polynomial terms

In [9]:
```
# Fit a linear model with higher order polynomial on some important variables

# #HINT: Use ploy function to build polynomial terms, lm_poly <- RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) +
##poly(HUMIDITY, 4) .....

lm_poly <- lm_spec %>%
  fit(RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) + poly(HUMIDITY, 4) + poly(WIND_SPEED, 4) + poly(VISIBILITY, 8) +
      poly(DEW_POINT_TEMPERATURE, 4) + poly(SOLAR_RADIATION, 6) + poly(RAINFALL, 6) + poly(SNOWFALL, 6),
      data = train_data)
```

In [10]:
```
# Print model summary

# summary(lm_poly$fit)
summary(lm_poly$fit)
```

# Predictive analysis

## TASK: Add polynomial terms

Now, calculate R-squared and RMSE for the test results generated by `lm_ploy` model

In [13]:
```
# Calculate R-squared and RMSE from the test results
rsq(test_results, truth = truth,
    estimate = .pred)
rmse(test_results, truth = truth,
     estimate = .pred)
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.5800577 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 410.2241 |

# Predictive analysis

## TASK: Add interaction terms

*TODO:* Try adding some interaction terms to the previous polynomial models.

In [14]:
```
# Add interaction terms to the poly regression built in previous step

# HINT: You could use `*` operator to create interaction terms such as HUMIDITY*TEMPERATURE and make the formula look
# RENTED_BIKE_COUNT ~ RAINFALL*HUMIDITY ...
lm_poly1 <- lm_spec %>%
  fit(RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 6) + poly(HUMIDITY, 4) + poly(WIND_SPEED, 4) + poly(VISIBILITY, 8) +
      poly(DEW_POINT_TEMPERATURE, 4) + poly(SOLAR_RADIATION, 6) + poly(RAINFALL, 6) + poly(SNOWFALL, 6) +
      HUMIDITY*TEMPERATURE + RAINFALL*HUMIDITY + TEMPERATURE*WIND_SPEED + TEMPERATURE*VISIBILITY +
      TEMPERATURE*DEW_POINT_TEMPERATURE + TEMPERATURE*SOLAR_RADIATION + TEMPERATURE*RAINFALL + TEMPERATURE*SNOWFALL +
      HUMIDITY*WIND_SPEED + HUMIDITY*VISIBILITY + HUMIDITY*DEW_POINT_TEMPERATURE + HUMIDITY*SOLAR_RADIATION +
      HUMIDITY*SNOWFALL + WIND_SPEED*VISIBILITY + WIND_SPEED*DEW_POINT_TEMPERATURE + WIND_SPEED*SOLAR_RADIATION +
      WIND_SPEED*RAINFALL + WIND_SPEED*SNOWFALL + VISIBILITY*DEW_POINT_TEMPERATURE + VISIBILITY*SOLAR_RADIATION +
      VISIBILITY*RAINFALL + VISIBILITY*SNOWFALL + DEW_POINT_TEMPERATURE*SOLAR_RADIATION +
      DEW_POINT_TEMPERATURE*RAINFALL + DEW_POINT_TEMPERATURE*SNOWFALL + SOLAR_RADIATION*RAINFALL +
      SOLAR_RADIATION*SNOWFALL + RAINFALL*SNOWFALL, data = train_data)
```

In [15]:
```
# Print model summary
summary(lm_poly1$fit)
```

# Predictive analysis

## TASK: Add interaction terms

In [17]:
```
# Calculate R-squared and RMSE for the new model to see if performance has improved
rsq(test_results1, truth = truth,
    estimate = .pred)
rmse(test_results1, truth = truth,
     estimate = .pred)
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.5900638 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 405.2858 |

# Predictive analysis

## TASK: Add regularization

*TODO:* Define a linear regression model specification `glmnet_spec` using `glmnet` engine

In [21]:
```
# HINT: Use linear_reg() function with two parameters: penalty and mixture
# - penalty controls the intensity of model regularization
# - mixture controls the tradeoff between L1 and L2 regularizations

# You could manually try different parameter combinations or use grid search to find optimal combinations
glmnet_spec <- linear_reg(penalty = 0.1, mixture = 0) %>%   # ridge
  # Set engine
  set_engine("glmnet")

glmnet_spec1 <- linear_reg(penalty = 0.1, mixture = 1) %>%  # lasso
  # Set engine
  set_engine("glmnet")

glmnet_spec2 <- linear_reg(penalty = 0.1, mixture = 0.5) %>%  # elastic net
  # Set engine
  set_engine("glmnet")
```

# Predictive analysis

## TASK: Add regularization

In [22]:
```
# Fit a glmnet model using the fit() function
ridge_wf <- workflow() %>%
  add_recipe(weather_recipe)
ridge_fit <- ridge_wf %>%
  add_model(glmnet_spec) %>%
  fit(data = train_data)
ridge_fit %>%
  extract_fit_parsnip() %>%
  tidy()
```

In [23]:
```
lasso_wf <- workflow() %>%
  add_recipe(weather_recipe)
lasso_fit <- lasso_wf %>%
  add_model(glmnet_spec1) %>%
  fit(data = train_data)
lasso_fit %>%
  extract_fit_parsnip() %>%
  tidy()
```

In [24]:
```
elasticnet_wf <- workflow() %>%
  add_recipe(weather_recipe)
elasticnet_fit <- elasticnet_wf %>%
  add_model(glmnet_spec2) %>%
  fit(data = train_data)
elasticnet_fit %>%
  extract_fit_parsnip() %>%
  tidy()
```

# Predictive analysis

## TASK: Add regularization

In [28]:
```
# Report rsq and rmse of the `lm_glmnet` model
rsq(test_results_ridge, truth = truth,
    estimate = .pred)
rsq(test_results_lasso, truth = truth,
    estimate = .pred)
rsq(test_results_elasticnet, truth = truth,
    estimate = .pred)

rmse(test_results_ridge, truth = truth,
     estimate = .pred)
rmse(test_results_lasso, truth = truth,
     estimate = .pred)
rmse(test_results_elasticnet, truth = truth,
     estimate = .pred)
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.7645112 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.7685309 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.7685045 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 308.2321 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 304.6527 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 304.6712 |

# Predictive analysis

## TASK: Experiment to search for improved models

*TODO:* Experiment by building and testing at least five different models. For each of your experiments, include polynomial terms, interaction terms, and one of the three regularizations we introduced.

In [29]:
```
# Build at least five different models using polynomial terms, interaction terms, and regularizations.
ridge_spec <- linear_reg(penalty = 1, mixture = 0) %>%   # ridge
  # Set engine
  set_engine("glmnet")

lasso_spec <- linear_reg(penalty = 1, mixture = 1) %>%  # lasso
  # Set engine
  set_engine("glmnet")

elasticnet_spec <- linear_reg(penalty = 1, mixture = 0.5) %>%  # elastic net
  # Set engine
  set_engine("glmnet")

elasticnet_spec1 <- linear_reg(penalty = 1, mixture = 0.3) %>%  # elastic net
  # Set engine
  set_engine("glmnet")

elasticnet_spec2 <- linear_reg(penalty = 1, mixture = 0.1) %>%  # elastic net
  # Set engine
  set_engine("glmnet")

# Save their rmse and rsq values
```

# Predictive analysis

## TASK: Experiment to search for improved models

In [30]:
```
ridge_wf1 <- workflow() %>%
  add_recipe(weather_recipe)
ridge_fit1 <- ridge_wf1 %>%
  add_model(ridge_spec) %>%
  fit(data = train_data)
ridge_fit1 %>%
  extract_fit_parsnip() %>%
  tidy()
```

In [31]:
```
lasso_wf1 <- workflow() %>%
  add_recipe(weather_recipe)
lasso_fit1 <- lasso_wf1 %>%
  add_model(lasso_spec) %>%
  fit(data = train_data)
lasso_fit1 %>%
  extract_fit_parsnip() %>%
  tidy()
```

In [32]:
```
elasticnet_wf1 <- workflow() %>%
  add_recipe(weather_recipe)
elasticnet_fit1 <- elasticnet_wf1 %>%
  add_model(elasticnet_spec) %>%
  fit(data = train_data)
elasticnet_fit1 %>%
  extract_fit_parsnip() %>%
  tidy()
```

In [33]:
```
elasticnet_wf2 <- workflow() %>%
  add_recipe(weather_recipe)
elasticnet_fit2 <- elasticnet_wf2 %>%
  add_model(elasticnet_spec1) %>%
  fit(data = train_data)
elasticnet_fit2 %>%
  extract_fit_parsnip() %>%
  tidy()
```

In [34]:
```
elasticnet_wf3 <- workflow() %>%
  add_recipe(weather_recipe)
elasticnet_fit3 <- elasticnet_wf3 %>%
  add_model(elasticnet_spec2) %>%
  fit(data = train_data)
elasticnet_fit3 %>%
  extract_fit_parsnip() %>%
  tidy()
```

# Predictive analysis

## TASK: Experiment to search for improved models

| model_names | rsq_value | rmse_value |
|---|---|---|
| <fct> | <dbl> | <dbl> |
| lm_poly | 0.5800577 | 410.2241 |
| lm_poly1 | 0.5900638 | 405.2858 |
| ridge | 0.7645112 | 308.2321 |
| lasso | 0.7685309 | 304.6527 |
| elasticnet | 0.7685045 | 304.6712 |
| ridge1 | 0.7645112 | 308.2321 |
| lasso1 | 0.7688183 | 304.5610 |
| elasticnet1 | 0.7687420 | 304.5503 |
| elasticnet2 | 0.7686181 | 304.6139 |
| elasticnet3 | 0.7684817 | 304.6862 |

In [44]:
```
# Report the best performed
#model in terms of rmse and rsq
## The best model in terms
#of rsq is lasso1
rsq_lasso1
rmse_lasso1
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.7688183 |

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 304.561 |

In [45]:
```
## The best model in terms
#of rmse is elasticnet1
rsq_elasticnet1
rmse_elasticnet1
```

A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rsq | standard | 0.768742 |

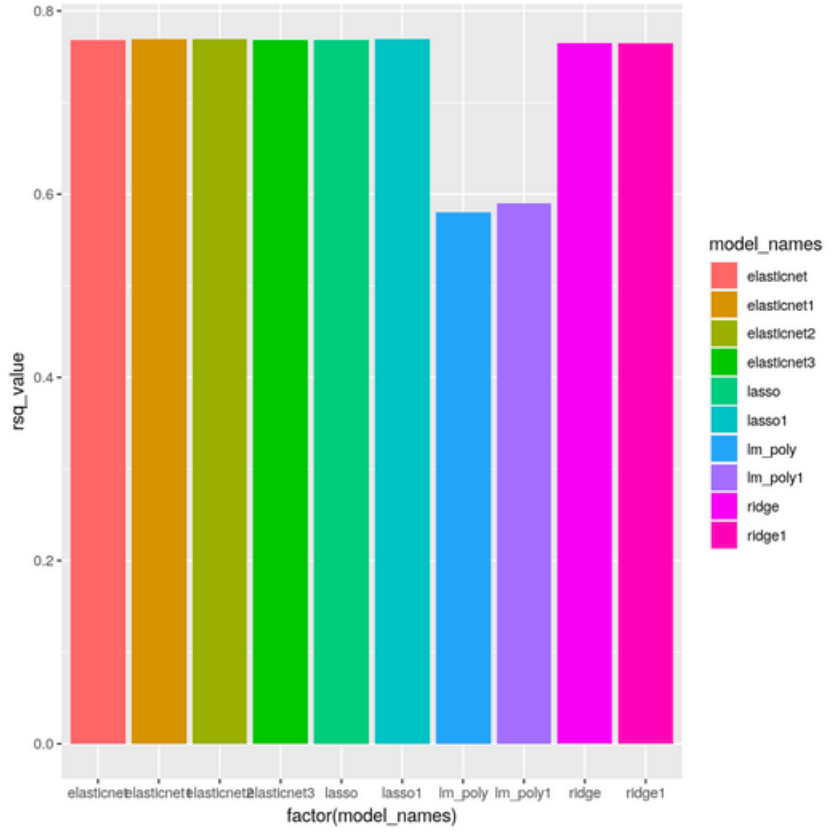
A tibble: 1 × 3

| .metric | .estimator | .estimate |
|---|---|---|
| <chr> | <chr> | <dbl> |
| rmse | standard | 304.5503 |

# Predictive analysis

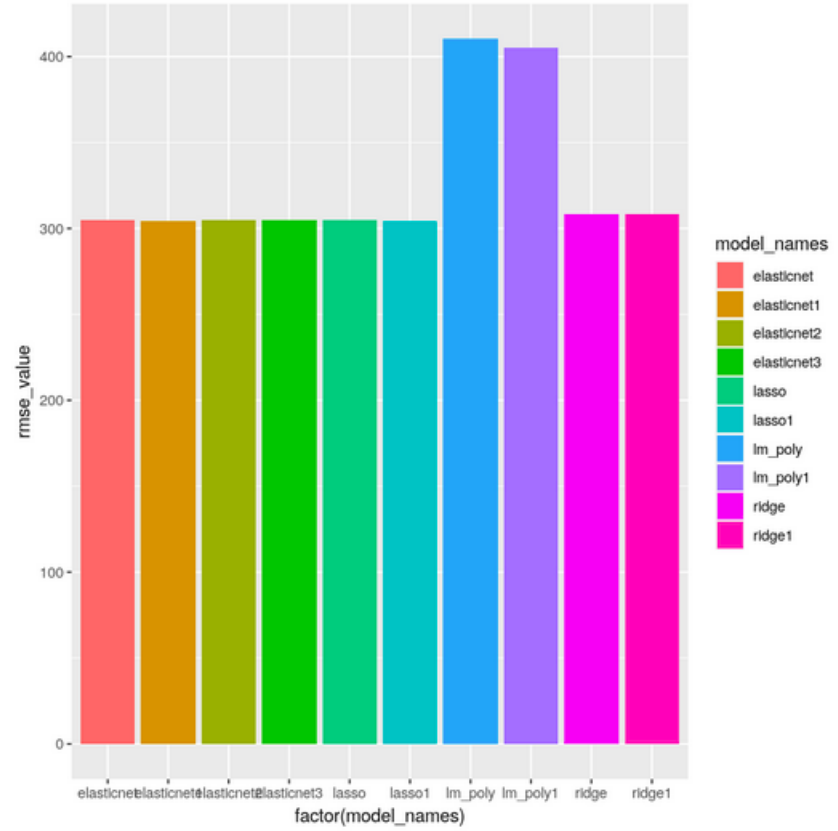
*TODO:* Visualize the saved RMSE and R-squared values using a grouped barchart

In [46]:
```
# HINT: Use ggplot() + geom_bar()
ggplot(comparison_df, aes(fill=model_names, y=rsq_value, x=factor(model_names))) +
    geom_bar(position="dodge", stat="identity")
```



# Predictive analysis

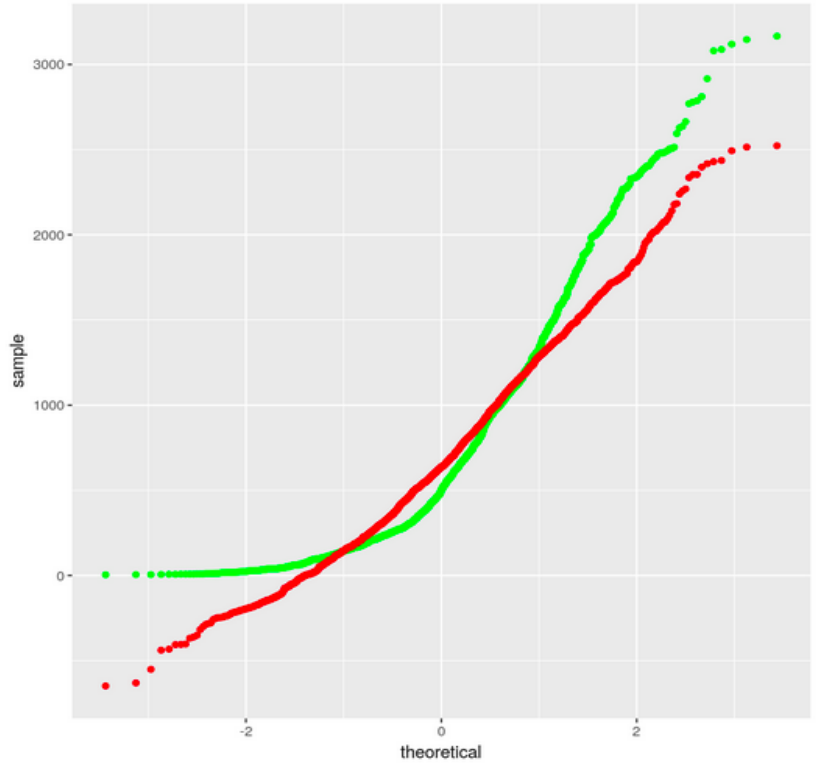
In [47]:
```
ggplot(comparison_df, aes(fill=model_names, y=rmse_value, x=factor(model_names))) +
    geom_bar(position="dodge", stat="identity")
```



# Predictive analysis

*TODO:* Create a Q-Q plot by plotting the distribution difference between the predictions generated by your best model and the true values on the test dataset.

In [48]:
```
# HINT: Use ggplot() +
    # stat_qq(aes(sample=truth), color='green') +
    # stat_qq(aes(sample=prediction), color='red')
ggplot() +
  stat_qq(aes(sample=test_results_lasso1$truth), color='green') +
  stat_qq(aes(sample=test_results_lasso1$.pred), color='red')
```



# Predictive analysis

In [49]:
```
# HINT: Use ggplot() +
    # stat_qq(aes(sample=truth), color='green') +
    # stat_qq(aes(sample=prediction), color='red')
ggplot() +
  stat_qq(aes(sample=test_results_elasticnet1$truth), color='green') +
  stat_qq(aes(sample=test_results_elasticnet1$.pred), color='red')
```