Applied Data Science with R Capstone project

Tianyuan Zhu

8/4/2022

Outline

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

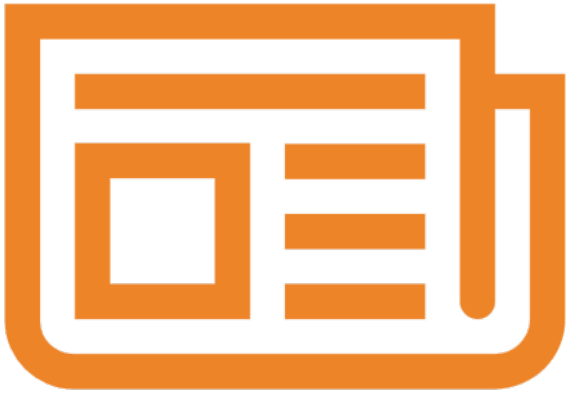
Executive Summary

- This report focuses on the analysis about the how the weather impacts the bike-sharing demand in Seoul.
- Data Collection
  - Weather Data: OpenWeather APIs
  - Bike-sharing Data: Web scrape a Wiki Page
- Data Wrangling
  - Deal with undesired reference links, missing value, indicator variables, normalization.
- Exploratory Data Analysis
  - SQL
  - Data visualization with different types of plots
- Develop and find regression model with best performance
- Create interactive dashboard for data analysis visualization

Introduction

- A bicycle-sharing system, bike share program, public bicycle scheme, or public bike share (PBS) scheme, is a shared transport service in which bicycles are made available for shared use to individuals on a short-term basis for a price or free.
- Objective: Investigate how weather would affect bike-sharing demand in urban areas, specifically, Seoul.

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

- The data collection process is shown in the following flowchart (see Appendix for screenshots of codes and output):

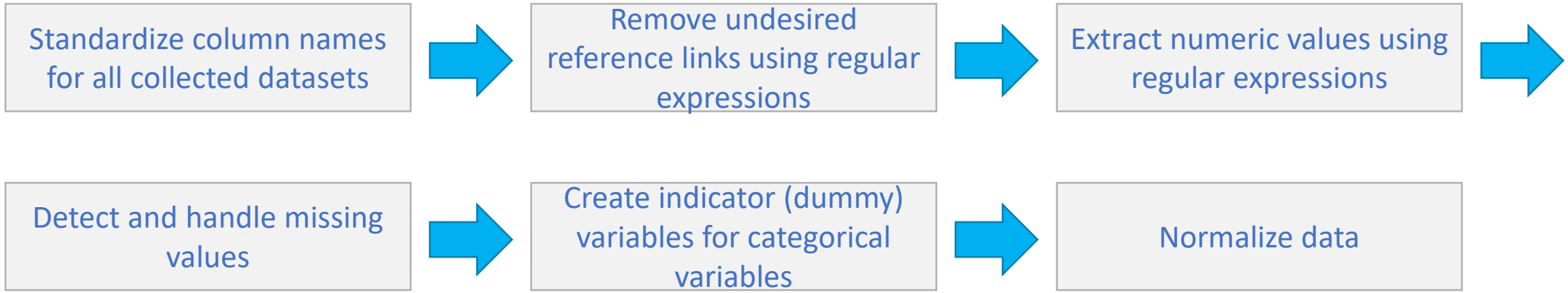
Web scrape a Global Bike-Sharing Systems Wiki Page: Extract bike sharing system data and convert the data to a data frame

↓

OpenWeather APIs Calls: Get the current weather data for a city and 5-day weather forecast for a list of cities

Data wrangling

- The data wrangling process is shown in the following flowchart (see Appendix for screenshots of codes and output):



EDA with SQL

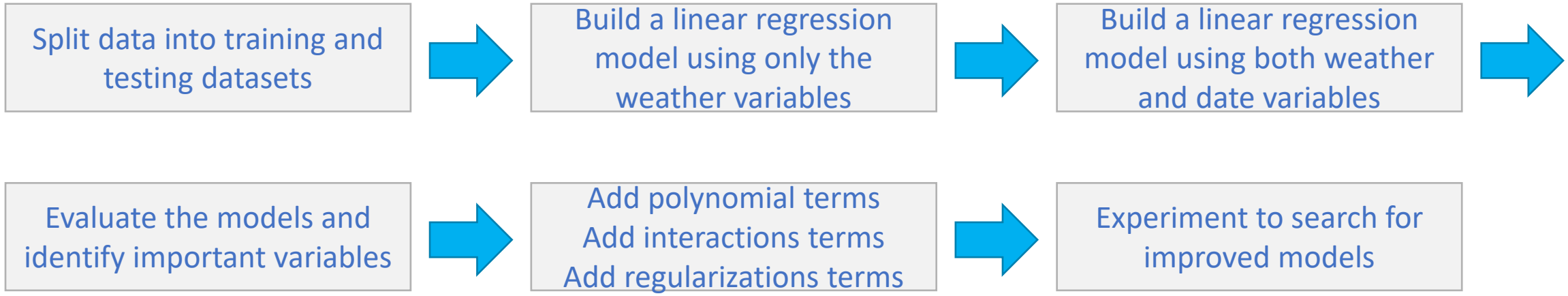
- Summary of performed SQL queries (see Appendix for screenshots of codes and output):
  - Determine how many records are in the seoul_bike_sharing dataset.
  - Determine how many hours had non-zero rented bike count.
  - Query the the weather forecast for Seoul over the next 3 hours.
  - Find which seasons are included in the seoul bike sharing dataset.
  - Find the first and last dates in the Seoul Bike Sharing dataset.
  - Determine which date and hour had the most bike rentals.
  - 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.
  - Find the average hourly bike count during each season.
  - 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?
  - Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes available in Seoul, plus the following city information about Seoul: CITY, COUNTRY, LAT, LON, POPULATION, in a single view.
  - 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

- Summary of charts (see Appendix for screenshots of codes and output):
  - A scatter plot of RENTED_BIKE_COUNT vs DATE
  - The same plot of the RENTED_BIKE_COUNT time series, but add HOURS as the colour
  - A histogram overlaid with a kernel density curve
  - A scatter plot to visualize the correlation between RENTED_BIKE_COUNT and TEMPERATURE by SEASONS
  - A display of four boxplots of RENTED_BIKE_COUNT vs. HOUR grouped by SEASONS
  - Group the data by DATE, and use the summarize() function to calculate the daily total rainfall and snowfall

Predictive analysis

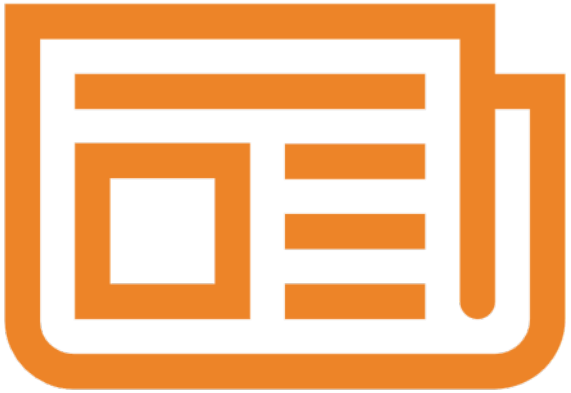
- The process of building, evaluating, improving and finding the best performing model is shown in the following flowchart:



Build a R Shiny dashboard

- Summary of plots and interactions in the dashboard:
  - A basic max bike prediction overview map
  - A select input (dropdown menu) to select a specific city
  - Users can select a specific city to show its detailed weather label
  - Detailed plots for a city:
    - A static temperature trend line
    - An interactive bike-sharing demand prediction trend line
    - 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

- The busiest bike rental time is on 2018-06-19 around 18:00.

2.6.1 Solution 6

In [14]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT DATE, HOUR, RENTED_BIKE_COUNT FROM SEOUL_DATA WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) F
```

A data.frame: 1 × 3

DATE	HOUR	RENTED_BIKE_COUNT
<chr>	<dbl>	<dbl>
2018-06-19	18	3556

Hourly popularity and temperature by seasons

- The top 10 most popular seasons and hours are shown as follows:

2.7 Task 7 - Hourly popularity and temperature by season

2.7.0.1 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.

2.7.1 Solution 7

In [21]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT SEASONS, HOUR, AVG(TEMPERATURE), AVG(RENTED_BIKE_COUNT) FROM SEOUL_DATA GROUP BY SEASONS, HOUR ORDER BY
```

A data.frame: 10 × 4

SEASONS	HOUR	AVG(TEMPERATURE)	AVG(RENTED_BIKE_COUNT)
<chr>	<dbl>	<dbl>	<dbl>
Summer	18	29.38791	2135.141
Autumn	18	16.03185	1983.333
Summer	19	28.27378	1889.250
Summer	20	27.06630	1801.924
Summer	21	26.27826	1754.065
Spring	18	15.97222	1689.311
Summer	22	25.69891	1567.870
Autumn	17	17.27778	1562.877
Summer	17	30.07691	1526.293
Autumn	19	15.06346	1515.568

Rental Seasonality

- The average, minimum, maximum and standard deviation of hourly bike count for each season are shown as follows:

2.8 Task 8 - Rental Seasonality ¶

2.8.0.1 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.

2.8.1 Solution 8

In [23]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT SEASONS, AVG(RENTED_BIKE_COUNT), MIN(RENTED_BIKE_COUNT), MAX(RENTED_BIKE_COUNT), STDEV(RENTED_BIKE_COUNT
```

A data.frame: 4 × 5

SEASONS	AVG(RENTED_BIKE_COUNT)	MIN(RENTED_BIKE_COUNT)	MAX(RENTED_BIKE_COUNT)	STDEV(RENTED_BIKE_COUNT)
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
Autumn	924.1105	2	3298	617.5479
Spring	746.2542	2	3251	618.6680
Summer	1034.0734	9	3556	690.2448
Winter	225.5412	3	937	150.3722

Weather Seasonality

- The average temperature, humidity, wind speed, visibility, dew point temperature, solar radiation, rainfall and snowfall for each season are shown as follows:

2.9 Task 9 - Weather Seasonality

2.9.0.1 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.

2.9.1 Solution 9

In [17]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT SEASONS, AVG(RENTED_BIKE_COUNT), AVG(TEMPERATURE), AVG(HUMIDITY), AVG(WIND_SPEED), AVG(VISIBILITY), AVG(
```

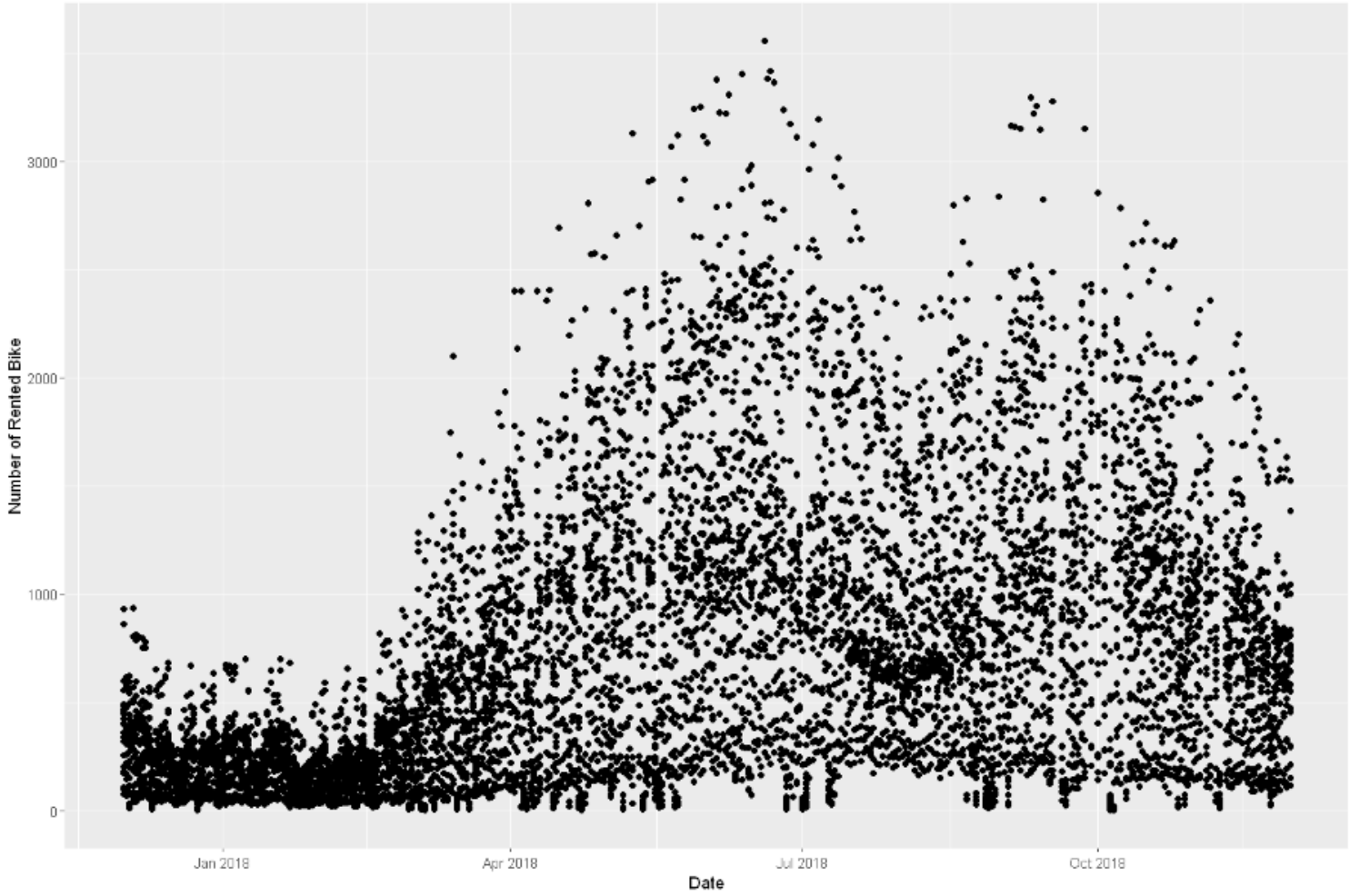
A data.frame: 4 × 10

SEASONS	AVG(RENTED_BIKE_COUNT)	AVG(TEMPERATURE)	AVG(HUMIDITY)	AVG(WIND_SPEED)	AVG(VISIBILITY)	AVG(DEW_POINT_TEMPERATURE)	AVG(SOL
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
Summer	1034.0734	26.587711	64.98143	1.609420	1501.745	18.750136	
Autumn	924.1105	13.821580	59.04491	1.492101	1558.174	5.150594	
Spring	746.2542	13.021685	58.75833	1.857778	1240.912	4.091389	
Winter	225.5412	-2.540463	49.74491	1.922685	1445.987	-12.416667	

EDA with Visualization

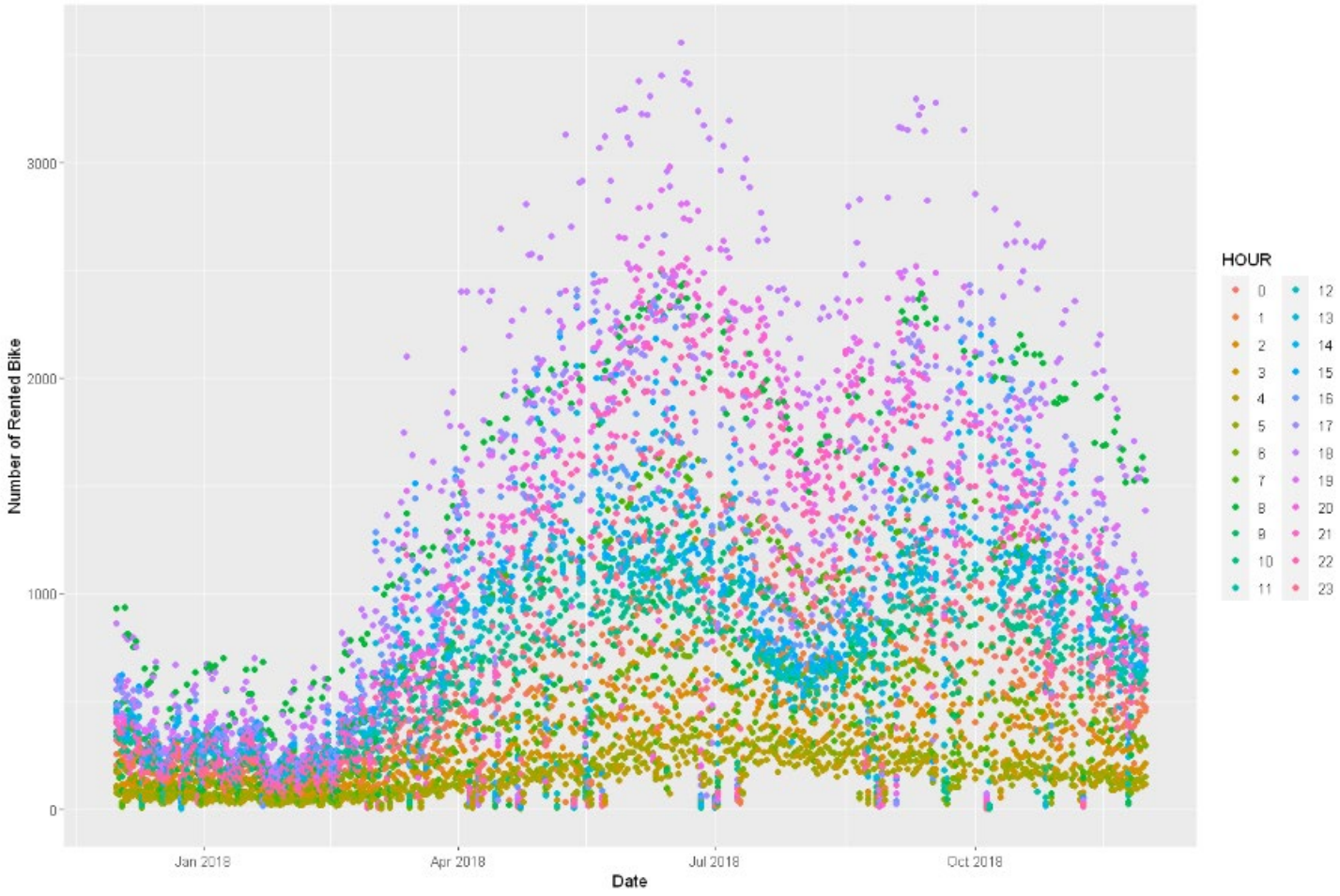
Bike rental vs. Date

In 2018, the bike rental increases in spring and fall and decreases in summer and winter.



Bike rental vs. Datetime

In 2018, the busiest time for bike rental is in spring and fall during 18:00-19:00.

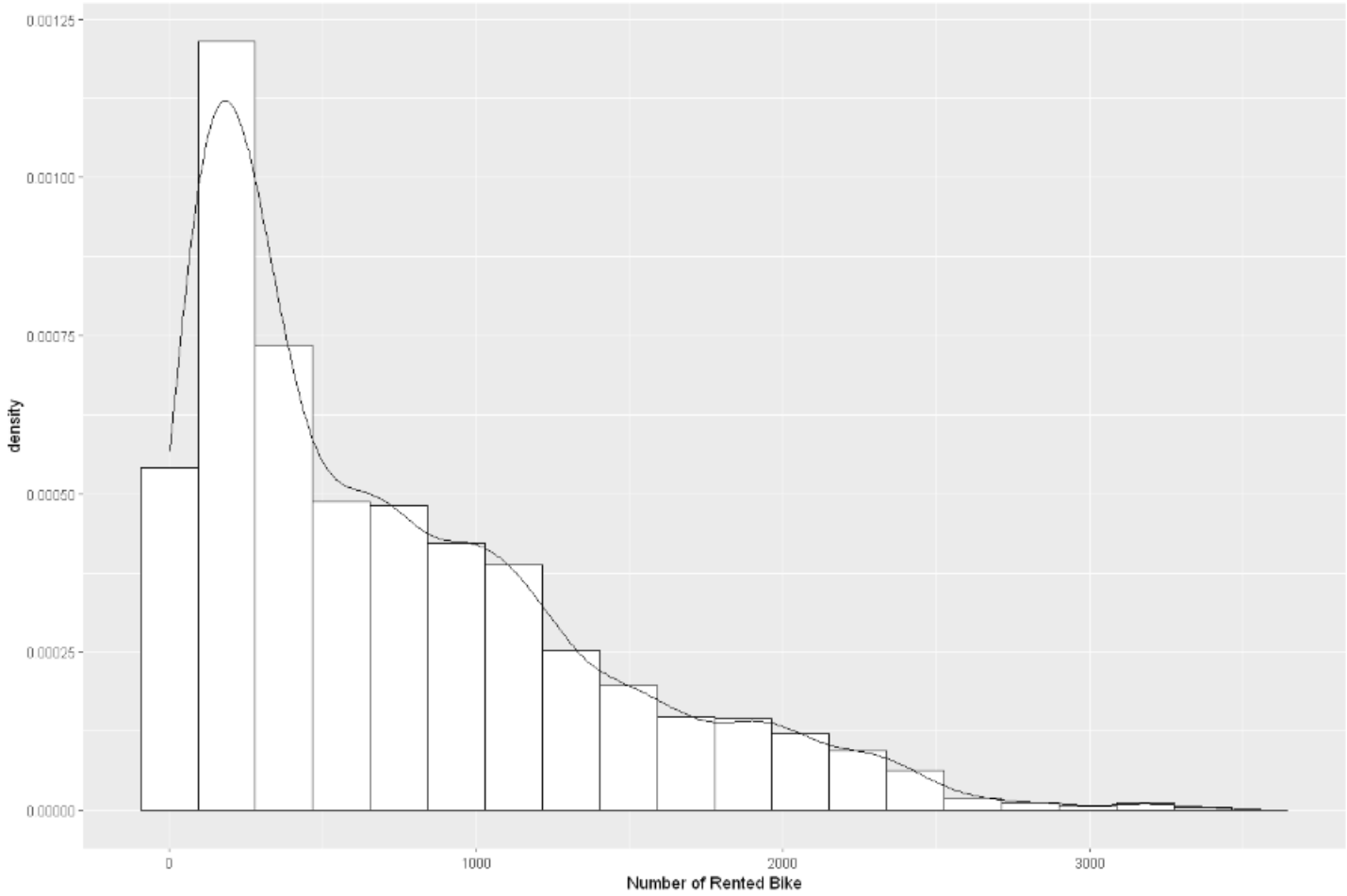


Bike rental histogram

We can see from the histogram that most of the time there are relatively few bikes rented. Indeed, the 'mode', or most frequent amount of bikes rented, is about 250.

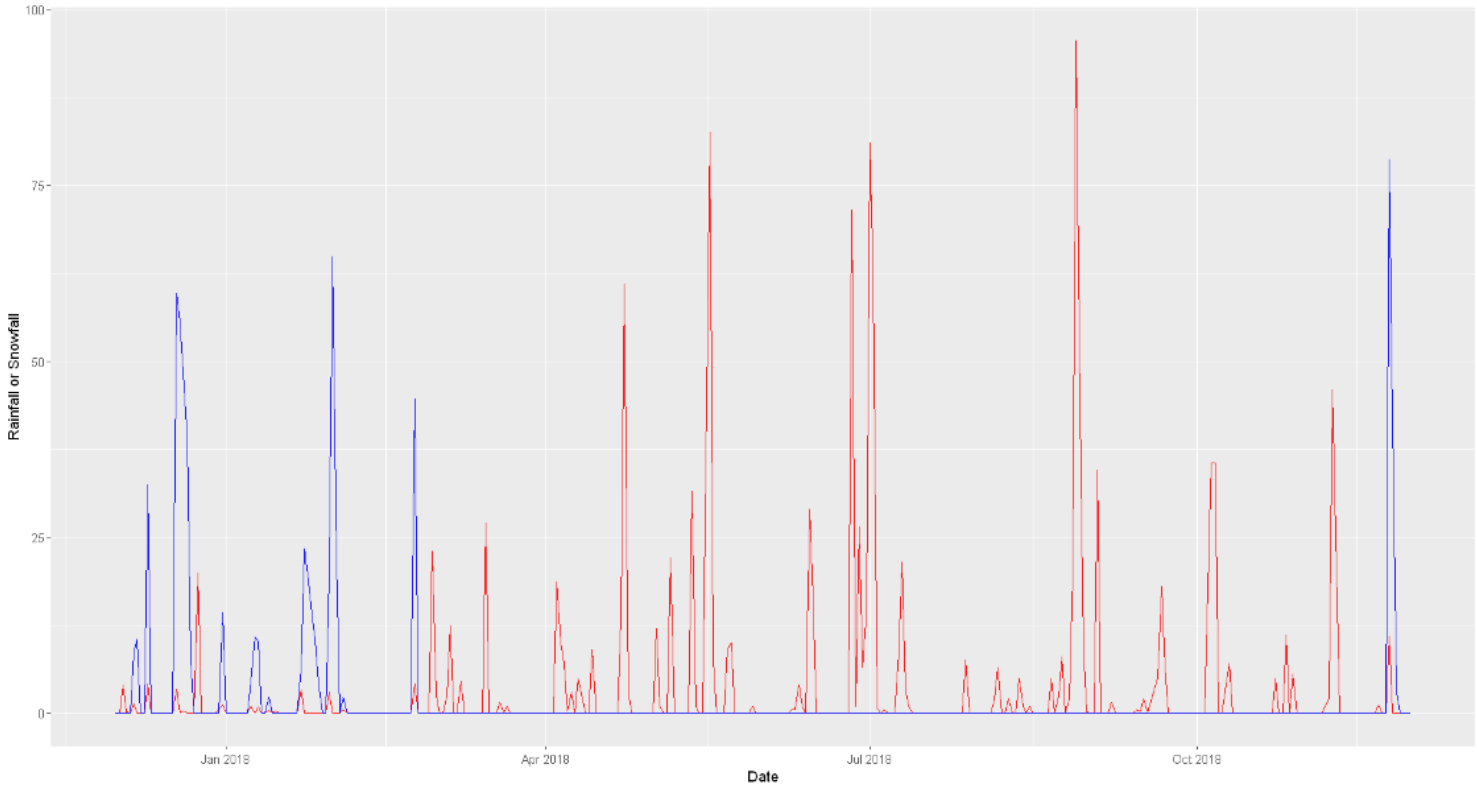
Judging by the 'bumps' at about 700, 900, and 1900, and 3200 bikes, it looks like there may be other modes hiding within subgroups of the data.

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



Daily total rainfall and snowfall

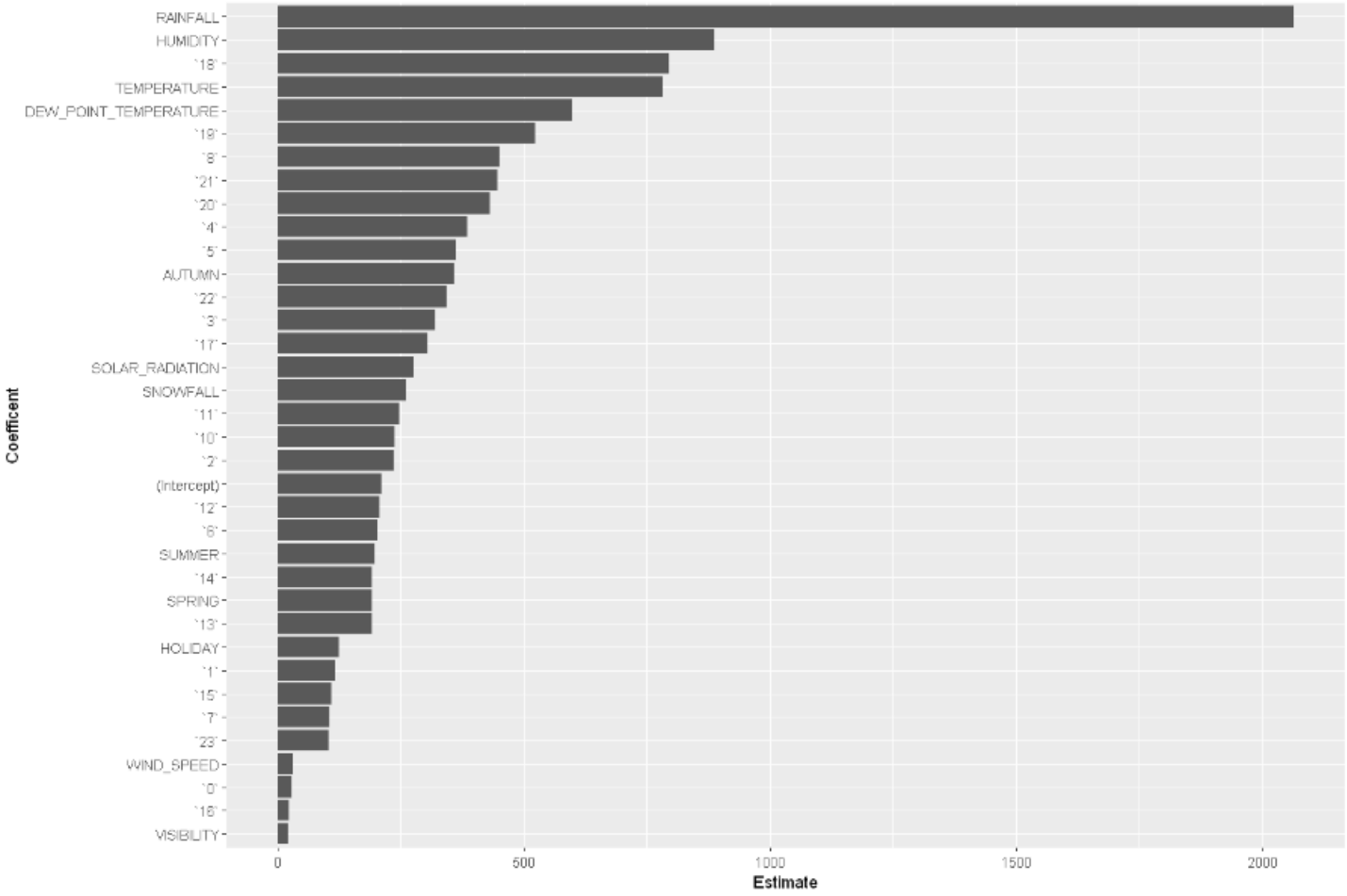
The highest total daily rainfall exists in summer while the highest total daily snowfall exists in winter.



Predictive analysis

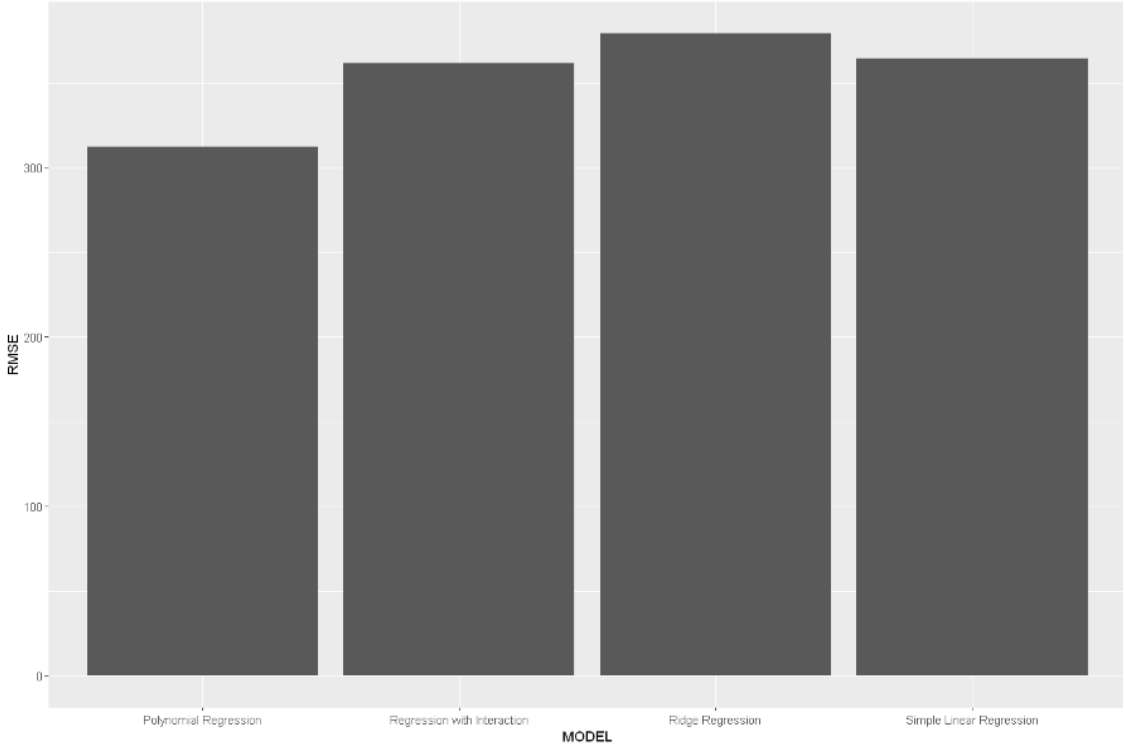
Ranked coefficients

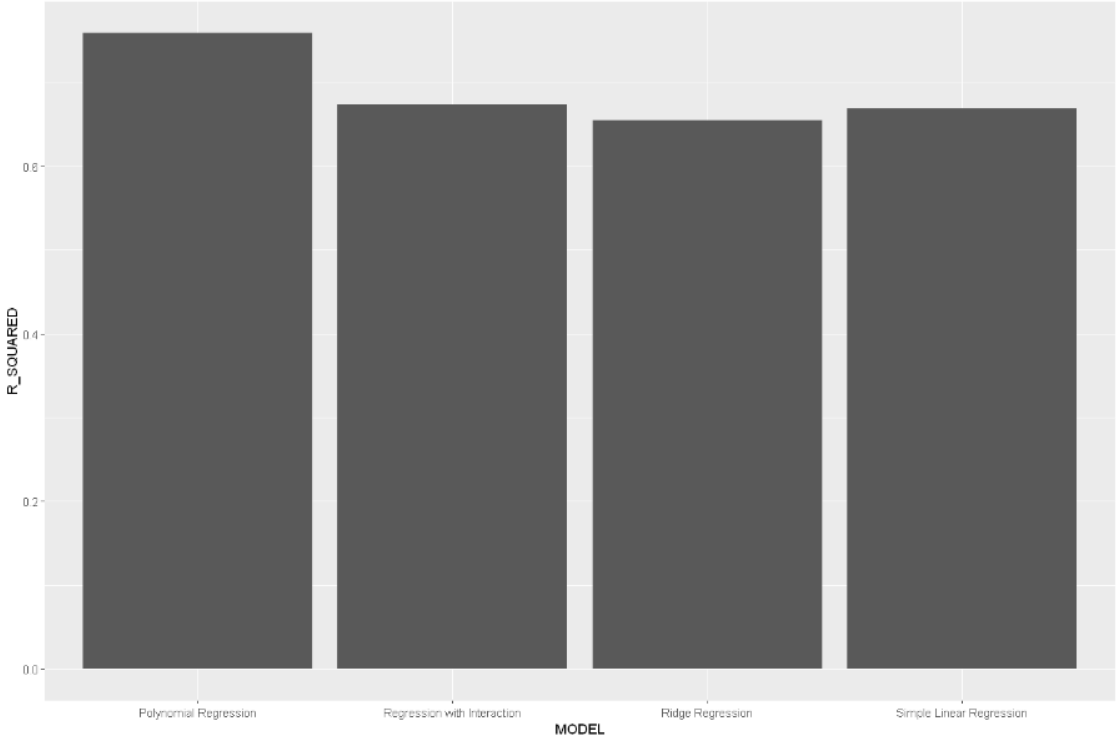
The most relevant variables for bike rental prediction are rainfall, humidity, hour of 18:00, temperature and dew point temperature. Rainfall and humidity are relevant as people are less likely to ride bikes when it rains. 18:00 may be the busiest hour since its peak hour when people get back from work. Hence, riding a bike can be a better choice due to the heavy traffic on the road.



Model evaluation

The polynomial regression model has the best performance.





Find the best performing model

- Formula: RENTED_BIKE_COUNT ~ poly(RAINFALL, 8, raw=TRUE) + poly(HUMIDITY, 4, raw=TRUE) + `18` + poly(TEMPERATURE, 4, raw=TRUE) + poly(DEW_POINT_TEMPERATURE, 3, raw=TRUE) + `19`+ `8` + `21` + `20`+ `4` + `5`+ AUTUMN + `22` + `3` + `17` + SOLAR_RADIATION + SNOWFALL + `11`+ `10` + `2` + `12` + `8` + SUMMER +`14` + SPRING + `13` + HOLIDAY + `1`+ `15` + `7` + `23` + WIND_SPEED +`0` + `16` + VISIBILITY
- Performance:

In [20]:
```
# Calculate R-squared and RMSE from the test results
rsq_poly <- rsq(test_poly_results, truth = truth, estimate = .pred)
rsq_poly
rmse_poly <- rmse(test_poly_results, truth = truth, estimate = .pred)
rmse_poly
```

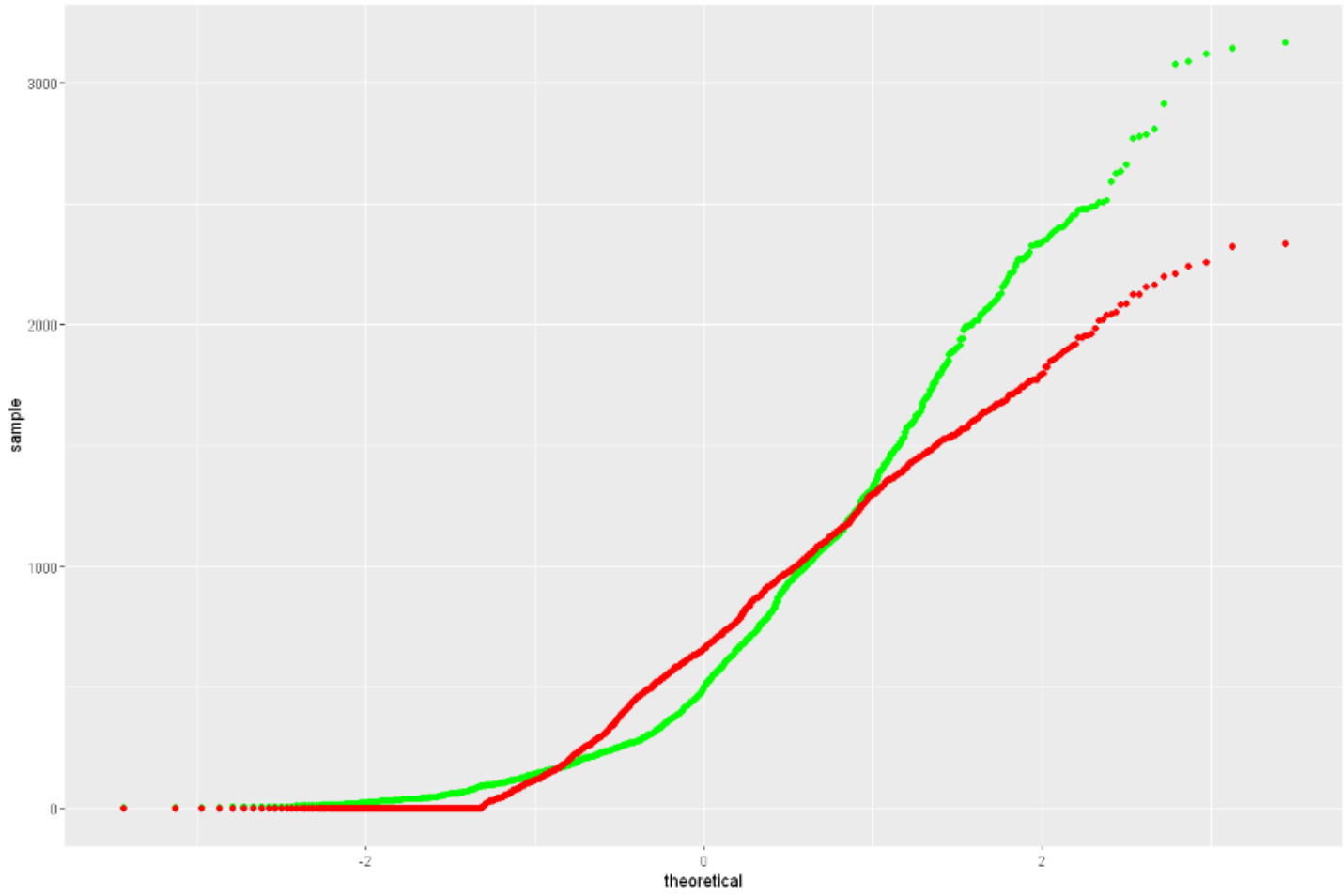
A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.7589189

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	312.3996

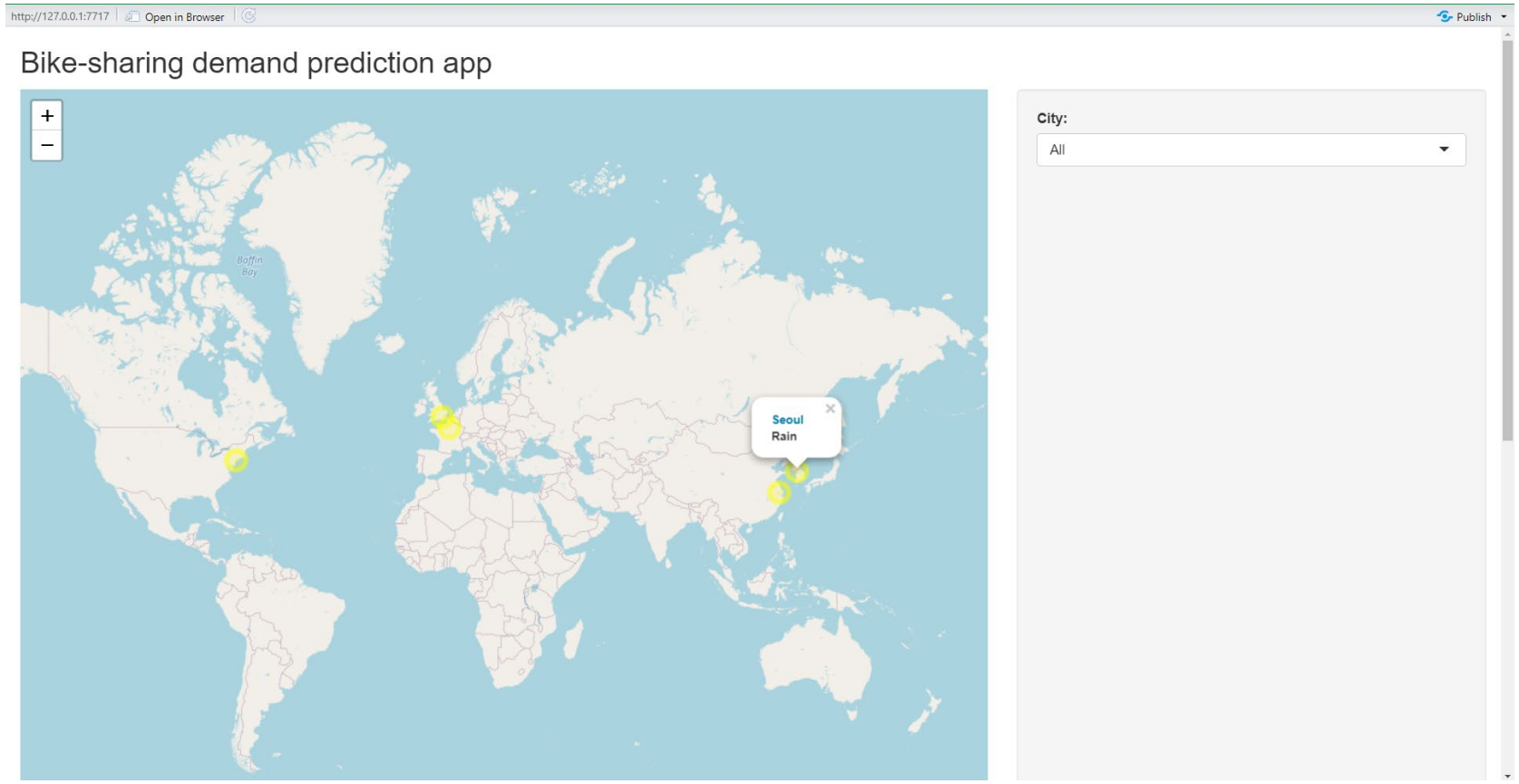
Q-Q plot of the best model



Dashboard

Overview of Bike-sharing prediction

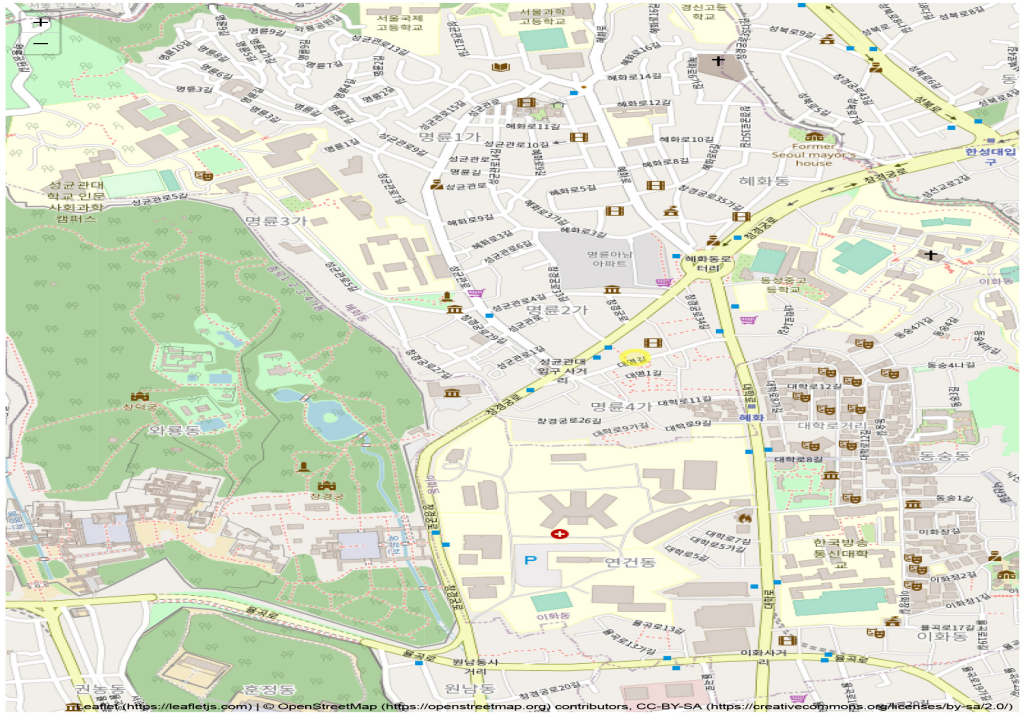
- Each circle marker represents a city, size and color of the circle shows the maximum bike demand prediction level for the city (small: size 6 and green color, medium: size 10 and yellow color, large: size 12 and red color). Click the circle shows the city name and the weather.

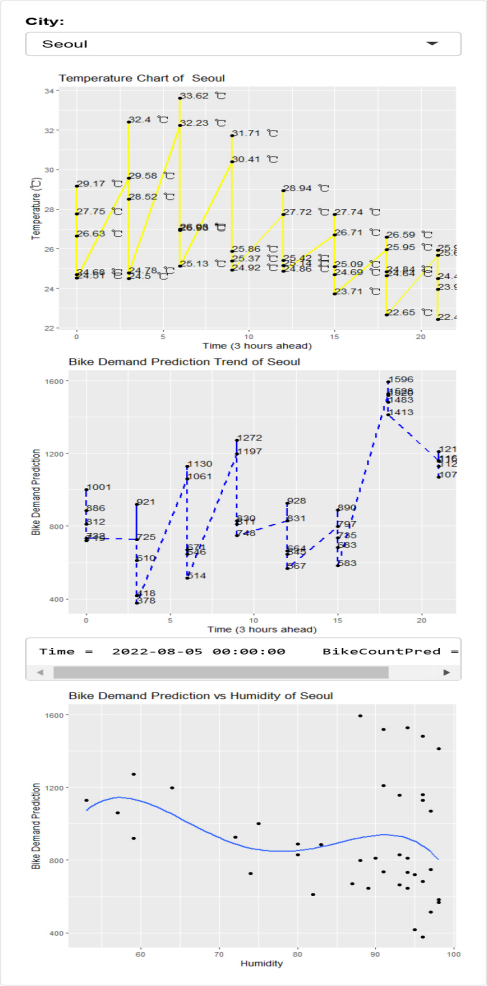


Detailed information of Seoul

The plots on the right shows the temperature trend, biked demand prediction trend and bike demand prediction vs. humidity for Seoul.

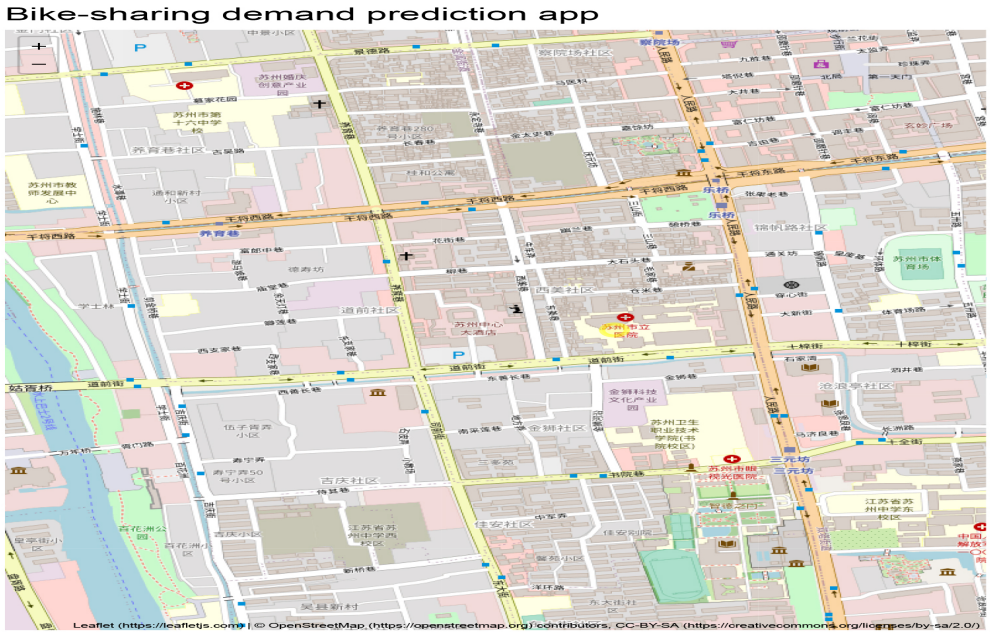
Bike-sharing demand prediction app

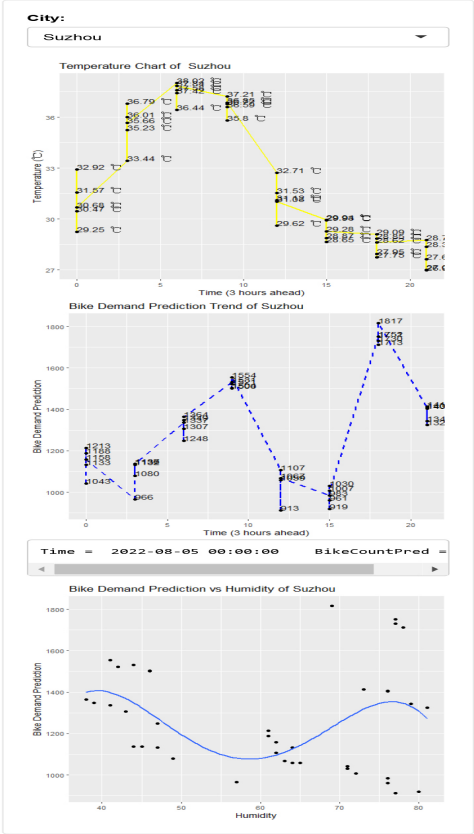




Detailed Information of Suzhou

The plots on the right shows the temperature trend, biked demand prediction trend and bike demand prediction vs. humidity for Suzhou.





CONCLUSION

- Demand for bike-sharing is high during spring and fall and low in summer and winter.
- The busiest time for bike-sharing is around 18:00.
- The most relevant variables for bike-sharing demand prediction are rainfall, humidity, temperature, dew point temperature.
- The relationships between the variables mentioned above and the bike-sharing demand are polynomial.

APPENDIX

- Data Collection: OpenWeatherAPI
- Data Collection: Webscrping
- Data Wrangling
- SQL Queries
- EDA Plots

Data Collection: OpenWeatherAPI

Next, let's create a list to hold URL parameters for current weather API

In [5]:
```
# need to be replaced by your real API key
your_api_key <- "ba0bdfb3d96e412900a0f51c48d2419c"
# Input `q` is the city name
# Input `appid` is your API KEY,
# Input `units` are preferred units such as Metric or Imperial
current_query <- list(q = "Seoul", appid = your_api_key, units="metric")
```

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

In [6]:
```
response <- GET(current_weather_url, query=current_query)
```

In [9]:
```
class(json_result)
```

'list'

Now let's print the JSON result.

In [10]:
```
json_result
```

re and transmit data objects. To read the JSON HTTP

$coord

In [12]:
```
# $weather is also a list with one element, its $main element indicates the weather status such as clear or rai
weather <- c(weather, json_result$weather[[1]]$main)
# Get Visibility
visibility <- c(visibility, json_result$visibility)
# Get current temperature
temp <- c(temp, json_result$main$temp)
# Get min temperature
temp_min <- c(temp_min, json_result$main$temp_min)
# Get max temperature
temp_max <- c(temp_max, json_result$main$temp_max)
# Get pressure
pressure <- c(pressure, json_result$main$pressure)
# Get humidity
humidity <- c(humidity, json_result$main$humidity)
# Get wind speed
wind_speed <- c(wind_speed, json_result$wind$speed)
# Get wind direction
wind_deg <- c(wind_deg, json_result$wind$deg)
```

Combine all vectors as columns of a data frame

In [14]:
```
# Check the generated data frame
print(weather_data_frame)
```

```
  weather visibility  temp temp_min temp_max pressure humidity wind_speed
1   Clear      10000 30.08    28.66    30.76     1011       66       2.06
  wind_deg
1       50
```

In [16]:
```
# Get forecast data for a given city list
get_weather_forecaset_by_cities <- function(city_names){
    df <- data.frame()
    for (city_name in city_names){
        # Forecast API URL
        forecast_url <- 'https://api.openweathermap.org/data/2.5/forecast'
        # Create query parameters
        forecast_query <- list(q = city_name, appid = "ba0bdfb3d96e412900a0f51c48d2419c", units="metric")
        # Make HTTP GET call for the given city
        response <- GET(forecast_url, query=forecast_query)
        # Note that the 5-day forecast JSON result is a list of lists. You can print the reponse to check the results
        json_result <- content(response, as="parsed")
        results <- json_result$list
        # Loop the json result
        for(result in results) {
            city <- c(city, city_name)
            }

            # Add the R Lists into a data frame
            df <- data.frame(city=city,
                             weather=weather,
                             visibility=visibility,
                             temp=temp,
                             temp_min=temp_min,
                             temp_max=temp_max,
                             pressure=pressure,
                             humidity=humidity,
                             wind_speed=wind_speed,
                             wind_deg=wind_deg,
                             forecast_datetime=forecast_datetime,
                             season=season)

    }

    # Return a data frame
    return(df)

}
```

Complete and call `get_weather_forecaset_by_cities` function with a list of cities, and write the data frame into a csv file called `cities_weather_forecast.csv`

In [17]:
```
cities <- c("Seoul", "Washington, D.C.", "Paris", "Suzhou")
cities_weather_df <- get_weather_forecaset_by_cities(cities)
cities_weather_df
str(cities_weather_df)
```

city	weather	visibility	temp	temp_min	temp_max	pressure	humidity	wind_speed	wind_deg	forecast_datetime	season
<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<int>	<int>	<dbl>	<int>	<dbl>	<chr>
Seoul	Clear	10000	30.08	30.08	30.65	1011	66	1.96	82	1659139200	Summer
Seoul	Clouds	10000	31.74	31.74	35.07	1011	58	2.98	110	1659150000	Summer
Seoul	Clouds	10000	34.19	34.19	36.25	1009	48	2.45	123	1659160800	Summer
Seoul	Rain	10000	28.94	28.94	28.94	1009	74	2.43	198	1659171600	Summer
Seoul	Rain	10000	27.66	27.66	27.66	1010	80	1.40	50	1659182400	Summer
Seoul	Clouds	10000	26.85	26.85	26.85	1009	82	1.59	77	1659193200	Summer

Data Collection: Webscrping

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

TODO: Get the root HTML node

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  ...
```

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

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

table_nodes <- html_nodes(root_node, "table")

In [3]:
```
# Convert the bike-sharing system table into a dataframe
table_node <- html_nodes(root_node, "table")
table_node
for (node in table_node){
 print(node)
}
```

```
{xml_nodeset (5)}
[1] <table class="box-Copy_edit plainlinks metadata ambox ambox-style ambox-C ...
[2] <table class="wikitable sortable" style="text-align:left"><tbody>\n<tr>\n ...
[3] <table class="nowraplinks mw-collapsible autocollapse navbox-inner" style ...
[4] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbod ...
[5] <table class="nowraplinks navbox-subgroup" style="border-spacing:0"><tbod ...

{html_node}
<table class="box-Copy_edit plainlinks metadata ambox ambox-style ambox-Copy_edit" role="presentation">
[1] <tbody><tr>\n<td class="mbox-image"><div class="mbox-image-div"><img alt= ...
{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 ...
{html_node}
<table class="nowraplinks mw-collapsible autocollapse navbox-inner" style="border-spacing:0;background:transparent;color:inheri
t">
[1] <tbody>\n<tr><th scope="col" class="navbox-title" colspan="2">\n<style da ...
{html_node}
<table class="nowraplinks navbox-subgroup" style="border-spacing:0">
...
...
```

In [4]:
```
bike_sharing_df<-html_table(table_node[[2]], fill=TRUE)
bike_sharing_df
```

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	
Australia	Melbourne[12]	oBike	4 Gen. oBike		July 2017	July 2018	dockless	1250	
Australia	Brisbane[14][15]	CityCycle	3 Gen. Cyclocity	JCDecaux	September 2010		150	2000	

Data Wrangling

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

In [5]:
```
# Drop rows with `RENTED_BIKE_COUNT` column == NA
bike_sharing_df<-bike_sharing_df %>% drop_na(RENTED_BIKE_COUNT)
```

In [6]:
```
# Print the dataset dimension again after those rows are droppe
dim(bike_sharing_df)
```

8465 · 14

In [7]:
```
bike_sharing_df %>%
                filter(is.na(TEMPERATURE))
```

A tibble: 11 × 14

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WI
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	
07/06/2018	3.8792535	0.9389561	NA	-0.06022153	
12/06/2018	0.8046113	0.3611369	NA	-0.64954264	
13/06/2018	3.0121265	0.7945013	NA	-0.06022153	
17/06/2018	2.4921617	0.7945013	NA	-0.01111144	

In [8]:
```
# Calculate the summer average temperature
mean_temp<-bike_sharing_df %>% group_by(SEASONS) %>% summariz
mean_temp
class(mean_temp)
mean_temp_summer<-mean_temp$avg_temp[mean_temp$SEASONS == 'Su
mean_temp_summer
```

A tibble: 4 × 2

SEASONS	avg_temp
<chr>	<dbl>
Autumn	0.07993194
Spring	0.01296048
Summer	1.14878089
Winter	-1.28998603

'tbl_df' · 'tbl' · 'data.frame'

1.14878088540798

In [9]:
```
# Impute missing values for TEMPERATURE column with summer average temperature
bike_sharing_df<-bike_sharing_df %>% replace_na(list(TEMPERATURE= mean_temp_summer))
```

In [14]:
```
# Print the dataset summary again to make sure the indicator columns are created properly
summary(bike_sharing_df)
```

```
     DATE           RENTED_BIKE_COUNT  TEMPERATURE          HUMIDITY
 Length:8465        Min.   :-1.1320   Min.   :-2.567596   Min.   :-2.859497
 Class :character   1st Qu.:-0.8020   1st Qu.:-0.826109   1st Qu.:-0.796873
 Mode  :character   Median :-0.2914   Median : 0.053008   Median :-0.060221
                    Mean   : 0.0000   Mean   :-0.007919   Mean   :-0.003883
                    3rd Qu.: 0.5524   3rd Qu.: 0.823281   3rd Qu.: 0.774650
                    Max.   : 4.4008   Max.   : 2.221494   Max.   : 1.953292
   WIND_SPEED         VISIBILITY        DEW_POINT_TEMPERATURE
 Min.   :-1.66449   Min.   :-2.317654   Min.   :-2.654888
 1st Qu.:-0.79601   1st Qu.:-0.824966   1st Qu.:-0.702416
 Median :-0.21703   Median : 0.416200   Median : 0.047946
 Mean   : 0.00094   Mean   :-0.004853   Mean   :-0.009863
 3rd Qu.: 0.55495   3rd Qu.: 0.925818   3rd Qu.: 0.851904
 Max.   : 5.47630   Max.   : 0.925818   Max.   : 1.770715
 SOLAR_RADIATION        RAINFALL           SNOWFALL         FUNCTIONING_DAY
 Min.   :-0.655094   Min.   :-0.131792   Min.   :-0.17188   Length:8465
 1st Qu.:-0.655094   1st Qu.:-0.131792   1st Qu.:-0.17188   Class :character
```

In [16]:
```
# Use the `mutate()` function to apply min-max normalization on columns
# `RENTED_BIKE_COUNT`, `TEMPERATURE`, `HUMIDITY`, `WIND_SPEED`, `VISIBILITY`, `DEW_POINT_TEMPERATURE`, `SOLAR_RADIATION`, `RAINFA
bike_sharing_df<-bike_sharing_df %>% mutate(RENTED_BIKE_COUNT = (RENTED_BIKE_COUNT - min(RENTED_BIKE_COUNT)) / (max(RENTED_BIKE_C
bike_sharing_df<-bike_sharing_df %>% mutate(TEMPERATURE = (TEMPERATURE - min(TEMPERATURE)) / (max(TEMPERATURE) - min(TEMPERATURE)
bike_sharing_df<-bike_sharing_df %>% mutate(HUMIDITY = (HUMIDITY - min(HUMIDITY)) / (max(HUMIDITY) - min(HUMIDITY)))
bike_sharing_df<-bike_sharing_df %>% mutate(WIND_SPEED = (WIND_SPEED - min(WIND_SPEED)) / (max(WIND_SPEED) - min(WIND_SPEED)))
bike_sharing_df<-bike_sharing_df %>% mutate(VISIBILITY = (VISIBILITY - min(VISIBILITY)) / (max(VISIBILITY) - min(VISIBILITY)))
bike_sharing_df<-bike_sharing_df %>% mutate(DEW_POINT_TEMPERATURE = (DEW_POINT_TEMPERATURE - min(DEW_POINT_TEMPERATURE)) / (max(D
bike_sharing_df<-bike_sharing_df %>% mutate(SOLAR_RADIATION = (SOLAR_RADIATION - min(SOLAR_RADIATION)) / (max(SOLAR_RADIATION) -
bike_sharing_df<-bike_sharing_df %>% mutate(RAINFALL = (RAINFALL - min(RAINFALL)) / (max(RAINFALL) - min(RAINFALL)))
bike_sharing_df<-bike_sharing_df %>% mutate(SNOWFALL = (SNOWFALL - min(SNOWFALL)) / (max(SNOWFALL) - min(SNOWFALL)))
```

In [17]:
```
# Print the summary of the dataset again to make sure the numeric columns range between 0 and 1
summary(bike_sharing_df)
```

```
     DATE           RENTED_BIKE_COUNT  TEMPERATURE        HUMIDITY
 Length:8465        Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
 Class :character   1st Qu.:0.05965   1st Qu.:0.3636   1st Qu.:0.4286
 Mode  :character   Median :0.15194   Median :0.5472   Median :0.5816
                    Mean   :0.20460   Mean   :0.5345   Mean   :0.5933
                    3rd Qu.:0.30445   3rd Qu.:0.7080   3rd Qu.:0.7551
                    Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
   WIND_SPEED       VISIBILITY     DEW_POINT_TEMPERATURE SOLAR_RADIATION
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000        Min.   :0.000000
 1st Qu.:0.1216   1st Qu.:0.4602   1st Qu.:0.4412        1st Qu.:0.000000
 Median :0.2027   Median :0.8429   Median :0.6107        Median :0.002841
 Mean   :0.2332   Mean   :0.7131   Mean   :0.5977        Mean   :0.161326
 3rd Qu.:0.3108   3rd Qu.:1.0000   3rd Qu.:0.7924        3rd Qu.:0.264205
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000        Max.   :1.000000
    RAINFALL           SNOWFALL        FUNCTIONING_DAY         Autumn
 Min.   :0.000000   Min.   :0.000000   Length:8465        Min.   :0.0000
 1st Qu.:0.000000   1st Qu.:0.000000   Class :character   1st Qu.:0.0000
 Median :0.000000   Median :0.000000   Mode  :character   Median :0.0000
 Mean   :0.004261   Mean   :0.008828                      Mean   :0.2288
 3rd Qu.:0.000000   3rd Qu.:0.000000                      3rd Qu.:0.0000
 Max.   :1.000000   Max.   :1.000000                      Max.   :1.0000
     Spring           Summer           Winter          Holiday
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
 Median :0.0000   Median :0.0000   Median :0.0000   Median :0.0000
 Mean   :0.2552   Mean   :0.2608   Mean   :0.2552   Mean   :0.0482
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.0000
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

set again to make sure no missing values in all columns

```
_COUNT      HOUR            TEMPERATURE
1320   Min.   :-1.661230   Min.   :-2.567596
8020   1st Qu.:-0.794501   1st Qu.:-0.826109
2914   Median : 0.072227   Median : 0.053008
0000   Mean   : 0.001015   Mean   :-0.007919
5524   3rd Qu.: 0.938956   3rd Qu.: 0.823281
4008   Max.   : 1.661230   Max.   : 2.221494
ED         VISIBILITY
```

ion to convert HOUR column into character type

```
>%
haracter)
```

```
t.default(HOUR):
y the caller; using TRUE"
```

CTIONING_DAY, and HOUR columns into indicator columns.

```
f %>%
n with single value

spread

f %>%
n with single value

spread

f %>%
n with single value

ead
```

SQL Queries

2.1 Task 1 - Record Count

2.1.0.1 Determine how many records are in the seoul_bike_sharing dataset.

2.1.1 Solution 1

2.3 Task 3 - Weather Outlook

2.3.0.1 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.

2.4 Task 4 - Seasons

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

In

2.4.1 Solution 4

In [12]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT DISTINCT SEASONS FROM SEOUL_DATA')
```

A data.frame: 4 × 1

SEASONS
<chr>
Winter
Spring
Summer
Autumn

In

2.5 Task 5 - Date Range

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

2.5.1 Solution 5

In [13]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT MIN(DATE) AS FIRST_DATE, MAX(DATE) AS LAST_DATE FROM SEOUL_DATA')
```

A data.frame: 1 × 2

FIRST_DATE	LAST_DATE
<chr>	<chr>
2017-12-01	2018-11-30

ON	RAINFALL	SNOV
>l>	<dbl>	
0	0	

2.6 Task 6 - Subquery - 'all-time high'

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

2.6.1 Solution 6

In [14]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT DATE, HOUR, RENTED_BIKE_COUNT FROM SEOUL_DATA WHERE RENTED_BIKE_COUNT = (SELECT MAX(RENTED_BIKE_COUNT) F
```

27.06630	1801.924
26.27826	1754.065
15.97222	1689.311
25.69891	1567.870
17.27778	1562.877

2.9 Task 9 - Weather Seasonality

2.9.0.1 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.

2.9.1 Solution 9

In [17]:
```
# provide your solution here
dbGetQuery(conn, 'SELECT SEASONS, AVG(RENTED_BIKE_COUNT), AVG(TEMPERATURE), AVG(HUMIDITY), AVG(WIND_SPEED), AVG(VISIBILITY), AVG(
```

A data.frame: 4 × 10

SEASONS	AVG(RENTED_BIKE_COUNT)	AVG(TEMPERATURE)	AVG(HUMIDITY)	AVG(WIND_SPEED)	AVG(VISIBILITY)	AVG(DEW_POINT_TEMPERATURE)	AVG(SOL
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
Summer	1034.0734	26.587711	64.98143	1.609420	1501.745	18.750136	
Autumn	924.1105	13.821580	59.04491	1.492101	1558.174	5.150594	
Spring	746.2542	13.021685	58.75833	1.857778	1240.912	4.091389	
Winter	225.5412	-2.540463	49.74491	1.922685	1445.987	-12.416667	

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

2.11.0.1 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.

2.11.1 Solution 11

In [19]:
```
# provide your solution here
```

In [20]:
```
close(conn)
```

```
Error in UseMethod("close"): no applicable method for 'close' applied to an object of class "c('SQLiteConnection', 'DBIConnecti
on', 'DBIObject')"
Traceback:

1. close(conn)
```

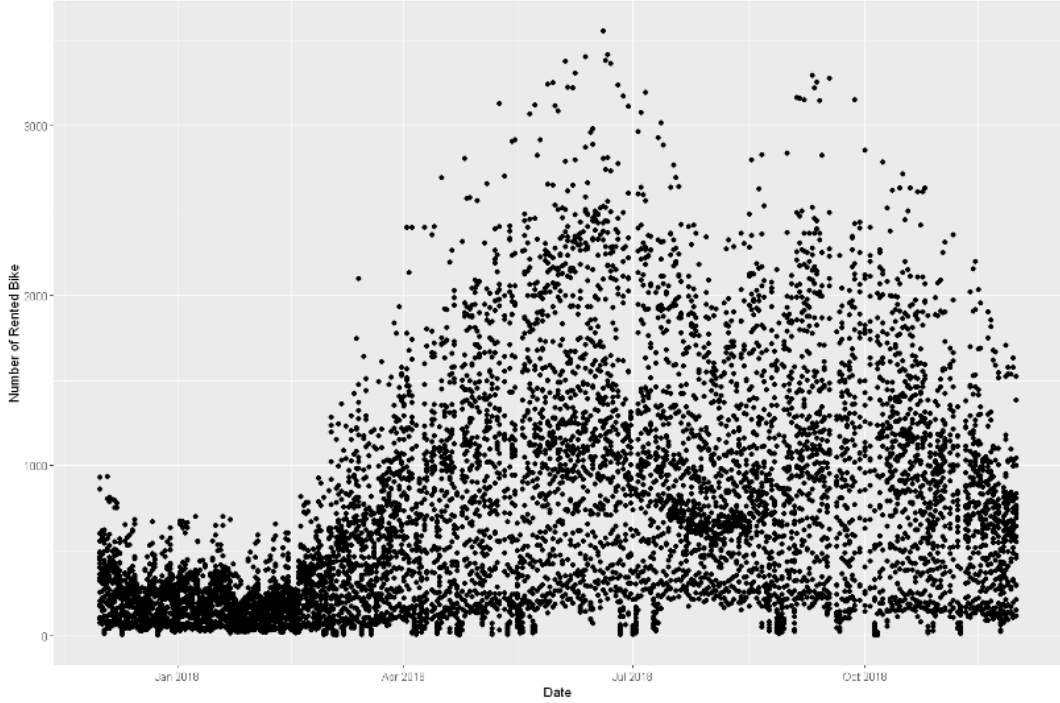
EDA Plots

2.4.2 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.

2.4.3 Solution 10

In [16]:
```
# provide your solution here
seoul_bike_sharing_df %>% ggplot(aes(x=DATE, y=RENTED_BIKE_COUNT),alpha=1/5)+geom_point()+labs(x="Date", y="Number of Rented Bike
```

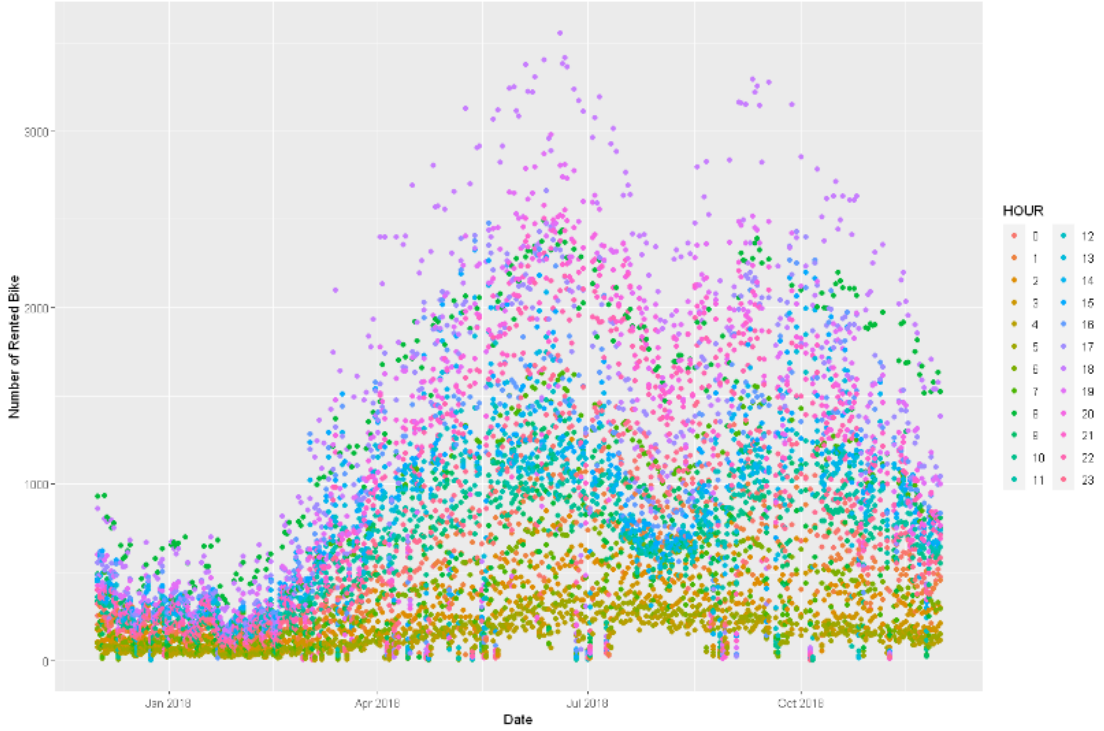


EDA Plots

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

2.4.8 Solution 11

In [17]:
```
# provide your solution here
seoul_bike_sharing_df %>% ggplot(aes(x=DATE, y=RENTED_BIKE_COUNT, colour=HOUR),alpha=1/5)+geom_point()+labs(x="Date", y="Number
```



EDA Plots

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

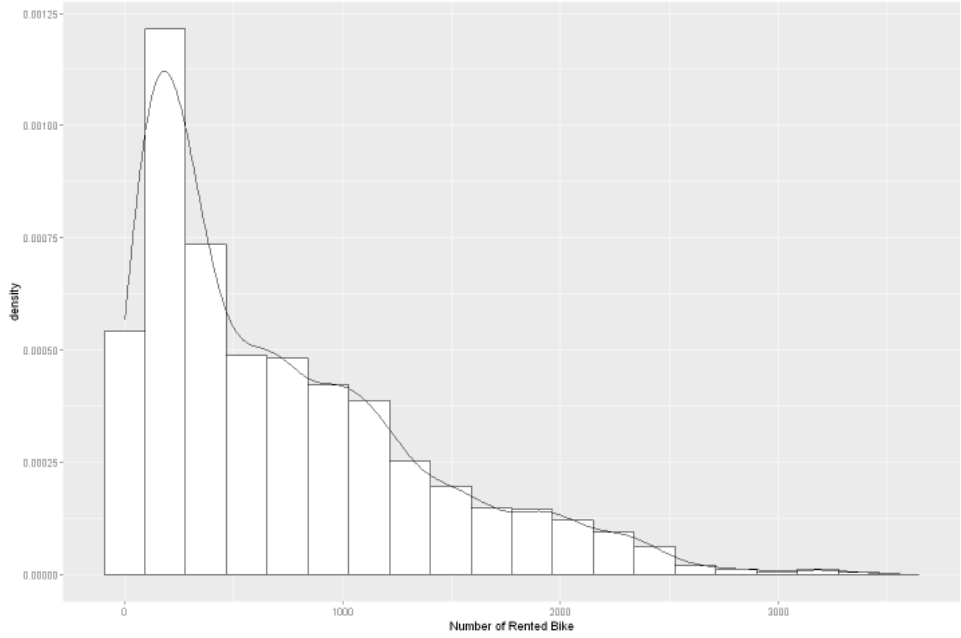
Normalize the histogram so the y axis represents 'density'. This can be done by setting `y=..density..` in the aesthetics of the histogram.

Click here for a hint

Click here for another hint

2.5.2 Solution 12

In [31]:
```
# provide your solution here
seoul_bike_sharing_df %>% ggplot(aes(x =RENTED_BIKE_COUNT))+geom_histogram(aes(y = ..density..), color="black", fill="white", bir
```



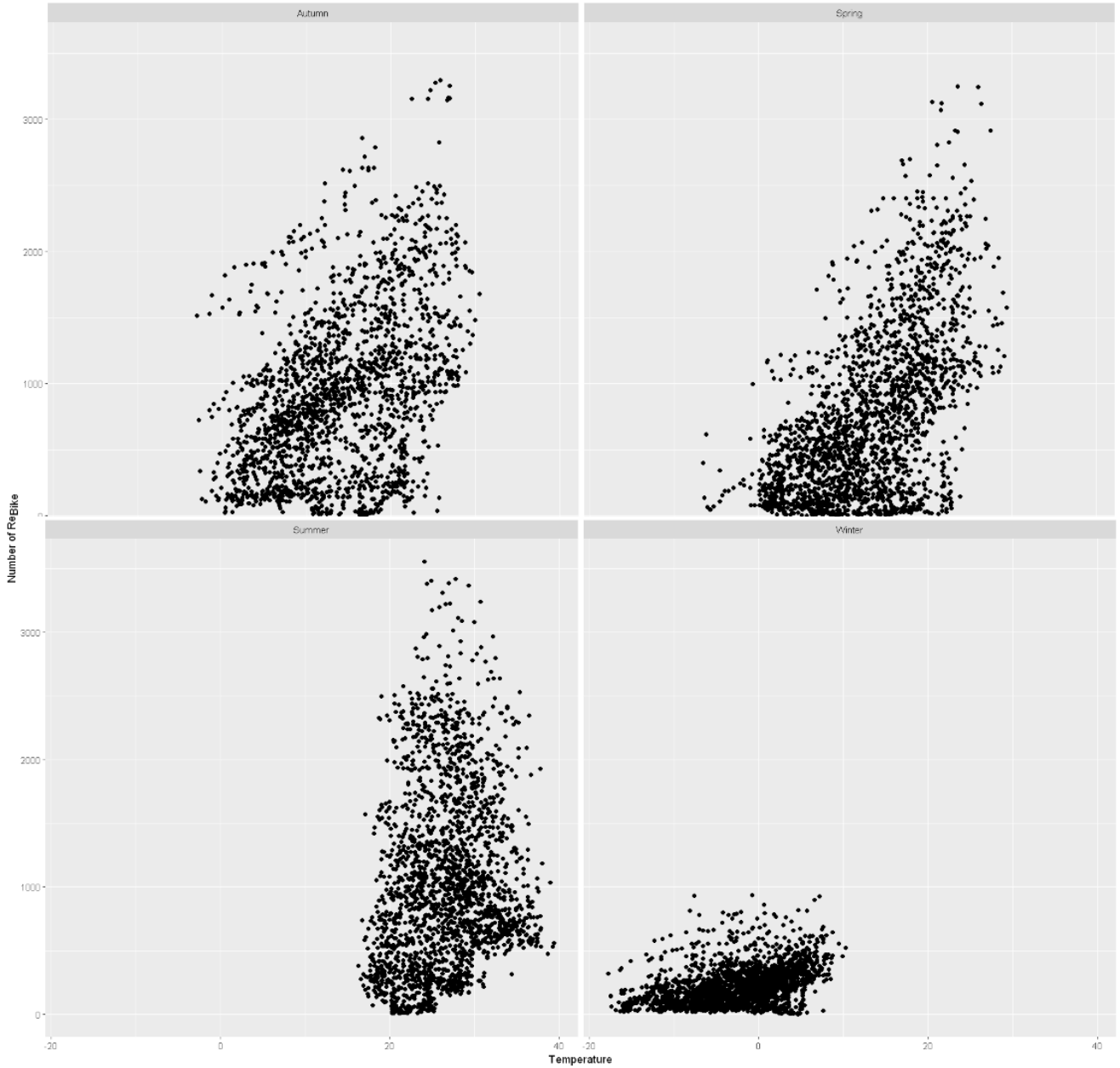
EDA Plots

2.6.1 Task 13 - Use a scatter plot to visualize the correlation betw `SEASONS`.

Start with `RENTED_BIKE_COUNT` vs. `TEMPERATURE`, then generate four plots correspond use of colour and opacity to emphasize any patterns that emerge. Use `HOUR` as the colo

2.6.2 Solution 13

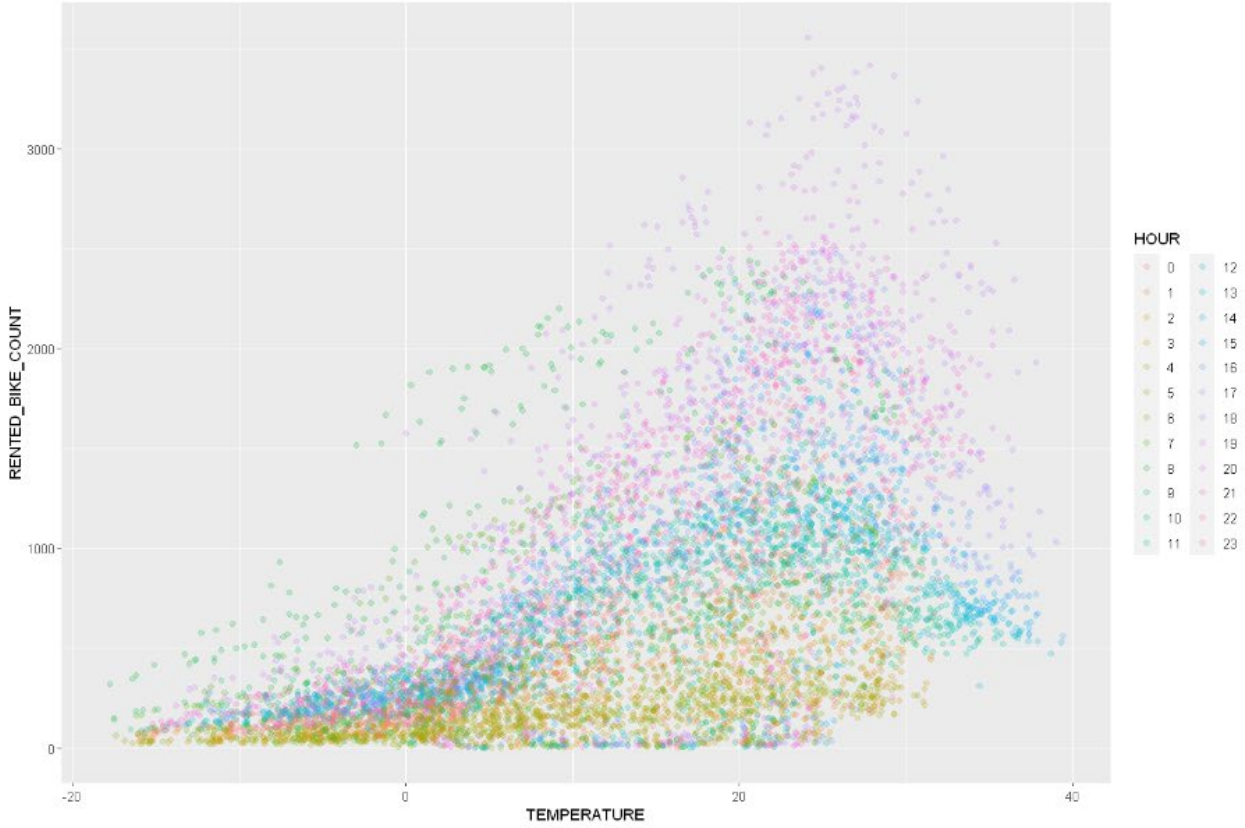
In [34]:
```
# provide your solution here
options(repr.plot.width=15, repr.plot.height=15)
seoul_bike_sharing_df %>% ggplot(aes(x=TEMPERATURE, y=RENTED_BIKE_COUNT)
```



EDA Plots

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

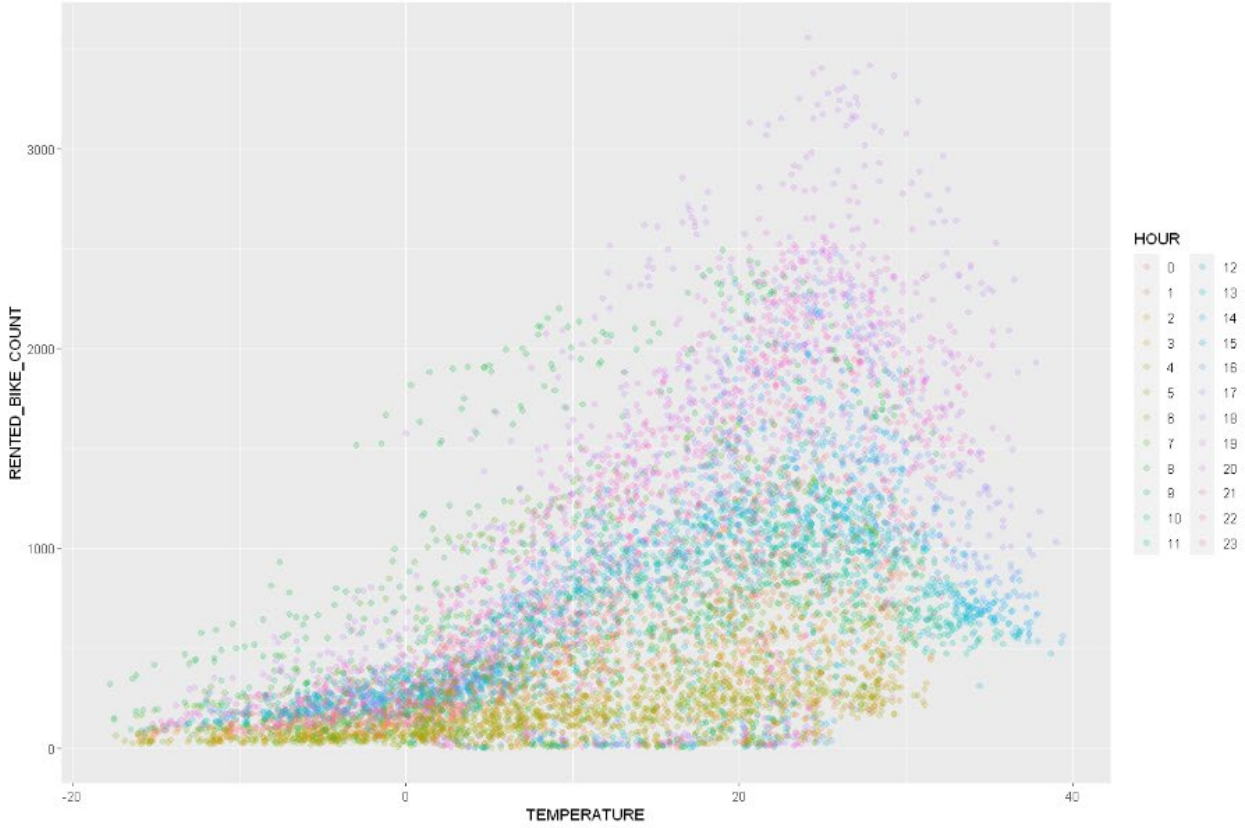
In [20]:
```
ggplot(seoul_bike_sharing_df) +
   geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=1/5)
```



EDA Plots

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

In [20]:
```
ggplot(seoul_bike_sharing_df) +
   geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=1/5)
```



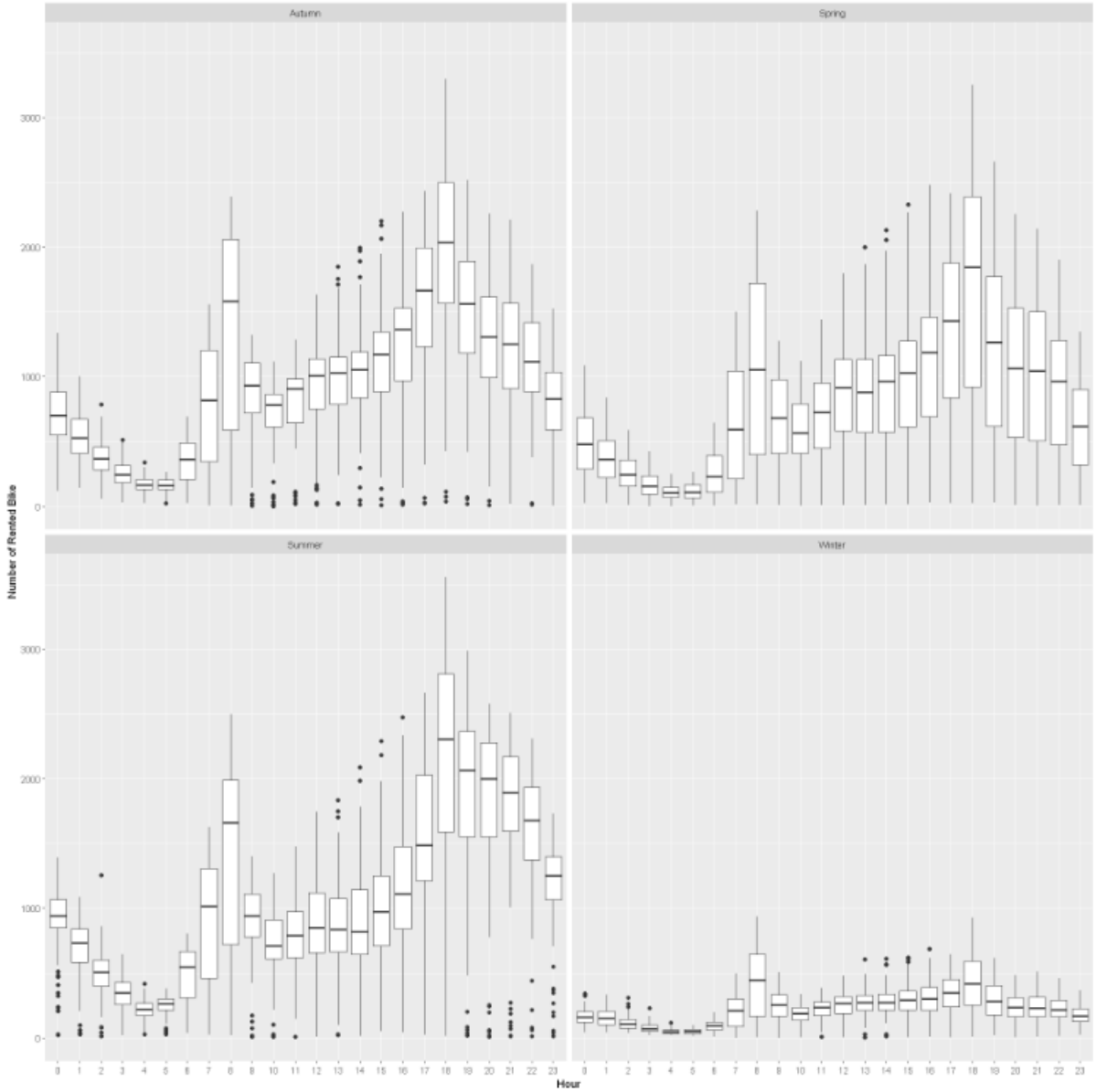
EDA Plots

2.7.1 Task 14 - Create a display of four boxplots of `RENTED_BIKE_COUNT` vs

Use `facet_wrap` to generate four plots corresponding to the seasons.

2.7.2 Solution 14

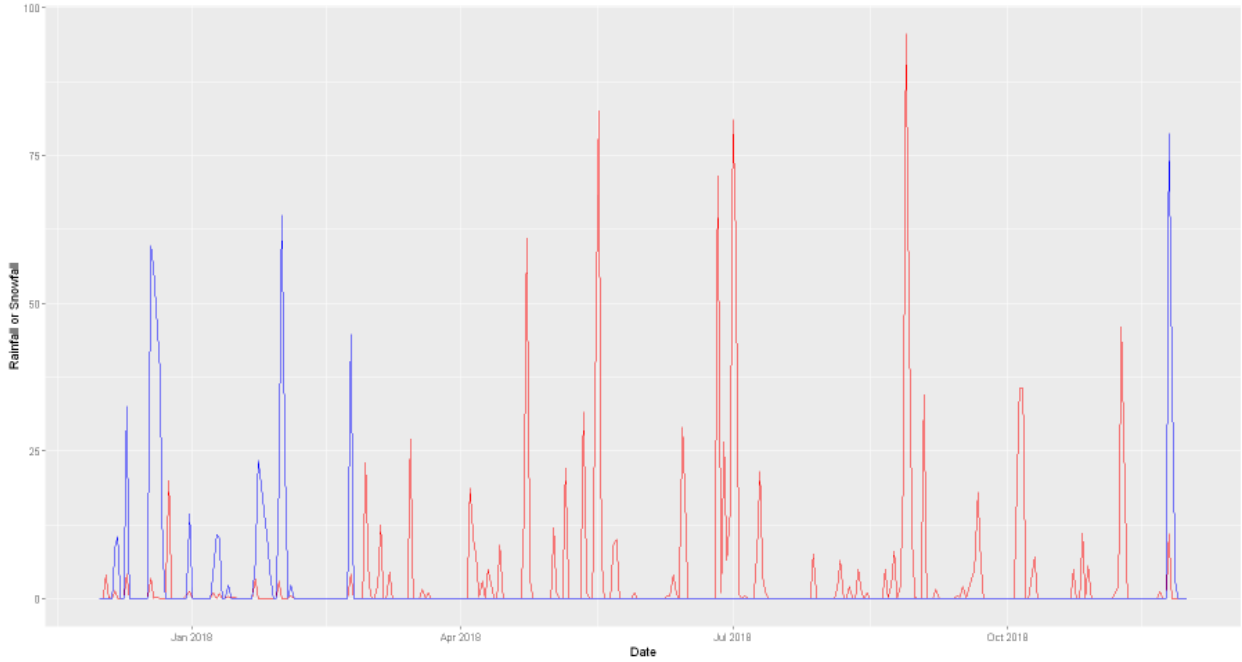
In [35]:
```
# provide your solution here
seoul_bike_sharing_df %>% ggplot(aes(x=HOUR, y=RENTED_BIKE_COUNT))+geom_boxplot()+l
```



EDA Plots

2.7.4 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.

2.7.5 Solution 15

In [43]:
```
# provide your solution here
options(repr.plot.width=15, repr.plot.height=8)
seoul_bike_sharing_df %>%
  group_by(DATE) %>%
  summarize(sum_rainfall=sum(RAINFALL), sum_snowfall=sum(SNOWFALL)) %>%
  ggplot(aes(x=DATE)) +geom_line(aes(y=sum_rainfall), color="red")+geom_line(aes(y=sum_snowfall), color="blue")+labs(x="Date", y=
```