Applied Data Science with R Capstone project

<11 November 2022>

Outline

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

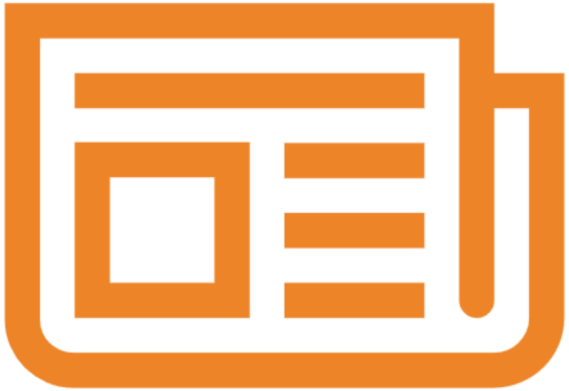
Executive Summary

- This research try to **analyze how weather would affect bike-sharing demand** in urban areas. The dataset used in this analysis is from Seoul Bike Sharing System, Open Weather API, World Cities, and Bike System in the World
- **Weather condition is one of the predictive variables** for number of rented bikes from temperature, rainfall, snowfall, etc.
  - **Most of the rent activity happens in summer**. Supported by weather seasonality, rent seasonality, Summer have the highest number bike and winter have the lowest bike rented.
  - **Average number of bikes rented have a positive correlation with temperature**. Higher the temperature, higher the number of bikes rented.
  - The patterns from the chart shown that there is **high demand of bike in the middle of the years and low demand in the first and last years.**
  - We can try to correlate it with the seasons where the demand of bike is high in summer and autumn season while the demand is low in the winter and some first day in spring season. Therefore **season can influence people decision to rent a bike.**
- This conditions allow bike supply to **consider the weather conditions when try to predict demand** to be able meet the demand yet not create excess supply or minimize cost.
- **To help user easily access the prediction, a dashboard is developed**. It contains Map chart using leaflet, Temperature trend line, Bike-sharing demand prediction trend line, and Bike-sharing demand prediction correlation plot (with prediction line) with interactive dropdown button

Introduction

- Rental bikes are available in many cities around the world
- It is important to provide a reliable supply of rental bikes to optimize availability and accessibility to the public at all times
- Reliable supply means minimizing the number of bikes supplied in order to meet the demand to minimize the cost of the program
- To optimize the supply, it would be helpful to be able to predict the number of bikes required based on various conditions such as weather
- Therefore, it will be a good approach to analyze how weather would affect bike-sharing demand in urban areas

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 from various sources such as:

1. World Cities Data
2. Seoul Bike Sharing Demand Data Set
3. Open Weather API Data - (Get current and forecasted weather)
4. Global Bike Sharing Systems Dataset – (List bike sharing system)

The dataset 1 and 2 are downloadable from given IBM Link. We get the data already in form of csv format.

The 3 dataset obtained using API Key and connect using RODBC() to get the data. We iterate the data and stored into dataframe and converted into csv.

The last, 4 dataset is obtained using webscraping with rvest library. We store the uncleaned data in csv format.

Finally, we store the csv dataset with respect to their names and data.

Data wrangling

To prepare the data sets for analysis, we utilize **library stringr (for regex) & library dplyr**

Library stringr for regex helps to standardize column names for all datasets, cleaning value with specific pattern and either replace, extract or remove the pattern such as reference links, space or numbers

Library dplyr can help data wrangling in fast way and ease with pipeline operations. We can detect and handled missing data, transform categorical variable to dummies, and normalize data to change data range or tailoring to other variables.

EDA with SQL

Here is the step by step for exploring the datasets using SQL. Step divided by established connection to database and querying datasets

1. Establish your Db2 connection
2. Querying Datasets
   a) Record Count
   b) Operational Hours
   c) Weather Outlook
   d) Seasons Breakdown
   e) Data Range
   f) Subquery conditional
   g) Tabulation by seasons
   h) Rental Seasonality
   i) Weather Seasonality
   j) Merge Seoul bike and World Cities dataset and find comparable countries

EDA with data visualization

After previously analyze dataset using MySQL, we implemented ggplot library to help with data visualization to understand data pattern and behavior for descriptive statistics. We need to make sure variables have the suitable type character before jump in to generate table/ plot.

Some of the analysis include:

Descriptive Statistics, Drilling Down, Data Visualization, Data Distribution, Correlation Analysis, and Outliers Detection.

Predictive analysis

In the predictive analysis:

1. We split training and testing data
2. Create regression model with weather variables only
3. Create the baseline model using all variables
4. Identifying important variables
5. Upgrade model with adjustments such as:
    1. Add polynomial terms
    2. Add interaction terms
    3. Add regularization
6. Experiment to search the better model improvements by comparing the model value of RMSE and R Square (iterate process)
7. Additional Q-Q Plot to check prediction with observation data

Build a R Shiny dashboard

In this dashboard consists some charts such as:

1. Map chart using leaflet
2. Temperature trend line
3. Bike-sharing demand prediction trend line
4. Bike-sharing demand prediction correlation plot (with prediction line)

For the interaction, there is a **cities data dropdown list** so user can choose specific country data information. For default it shown all cities data. In all cities mode, only map chart is available while the rest in showing if user selected specific city.

Results

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

EDA with SQL

Busiest bike rental times

A data.frame: 1 × 3

	DATE	HOUR	RENTED_BIKE_COUNT
	<chr>	<int>	<int>
1	19/06/2018	18	3556

Peak of bike rental in Seoul happen in 19/06/2018 in evening at 6 p.m. where the number of rental bikes is about 3556 bicycles.

Hourly popularity and temperature by seasons

A data.frame: 10 × 4

	AVG_TEMP	AVG_RENTED_BIKE	HOUR	SEASONS
	<dbl>	<int>	<int>	<chr>
1	29.38696	2135	18	Summer
2	16.03086	1983	18	Autumn
3	28.27283	1889	19	Summer
4	27.06630	1801	20	Summer
5	26.27826	1754	21	Summer
6	15.97222	1689	18	Spring
7	25.69891	1567	22	Summer
8	17.27778	1562	17	Autumn
9	30.07500	1526	17	Summer
10	15.06049	1515	19	Autumn

From the table we can see that the average rented bike is highest in the summer where 6 of the top 10 highest average rented bikes is in Summer season. Notice there is no winter season in the list.

Rental Seasonality

On average, More bike rented in summer while the lowest in winter. The number of rented bike dropped very high compared to other season which also shown in the deviation from mean.

A data.frame: 4 × 5

	SEASONS	AVG_RENTED_BIKE	MAX_BIKE	MIN_BIKE	STD_BIKE
	<chr>	<int>	<int>	<int>	<dbl>
1	Summer	1034	3556	9	690.0884
2	Autumn	924	3298	2	617.3885
3	Spring	746	3251	2	618.5247
4	Winter	225	937	3	150.3374

Weather Seasonality

A data.frame: 4 × 10

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

Average number of bikes rented have a positive correlation with temperature. Higher the temperature, higher the number of bikes rented.

Bike-sharing info in Seoul

A data.frame: 1 × 6

	BICYCLES	CITY_ASCII	COUNTRY	LAT	LNG	POPULATION
	<int>	<chr>	<chr>	<dbl>	<dbl>	<int>
1	20000	Seoul	Korea, South	37.58	127	21794000

Given the number of population, number of bikes available for rent is quite lower which is about 1:1000 means 1 bike is available for 1000 person.

Cities similar to Seoul

A data.frame: 7 × 6

	BICYCLES	CITY_ASCII	COUNTRY	LAT	LNG	POPULATION
	<int>	<chr>	<chr>	<dbl>	<dbl>	<int>
1	19165	Shanghai	China	31.16	121.46	22120000
2	20000	Seoul	Korea, South	37.58	127.00	21794000
3	16000	Beijing	China	39.90	116.39	19433000
4	20000	Weifang	China	36.71	119.10	9373000
5	15000	Ningbo	China	29.87	121.54	7639000
6	20000	Xi'an	China	34.26	108.90	7135000
7	20000	Zhuzhou	China	27.84	113.14	3855609

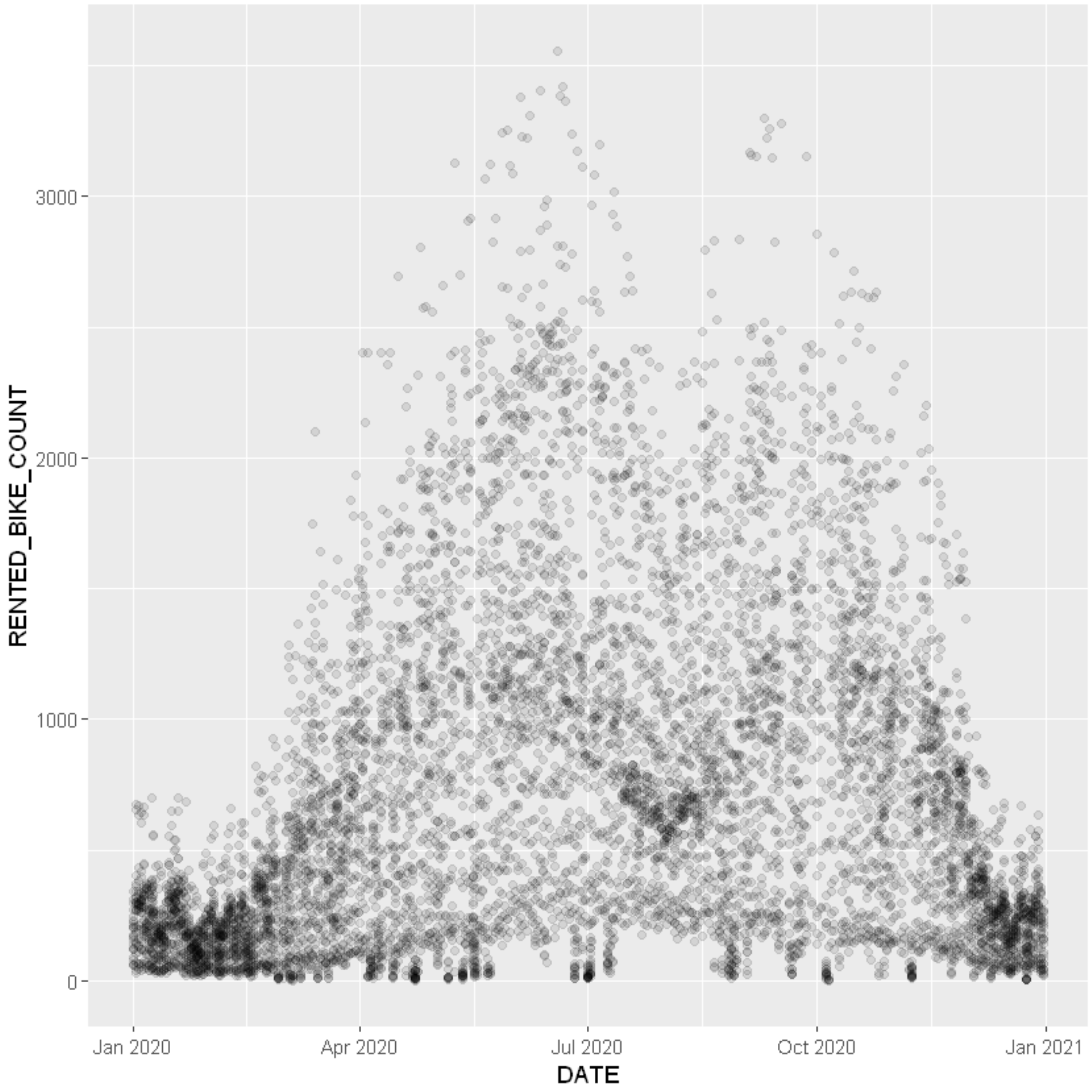
If we want to comparable cities for the Seoul Bike Sharing System we can compare with cities that have the same amount of available bikes supply like Shanghai, Beijing, etc. where it located mostly in China

EDA with Visualization

Bike rental vs. Date

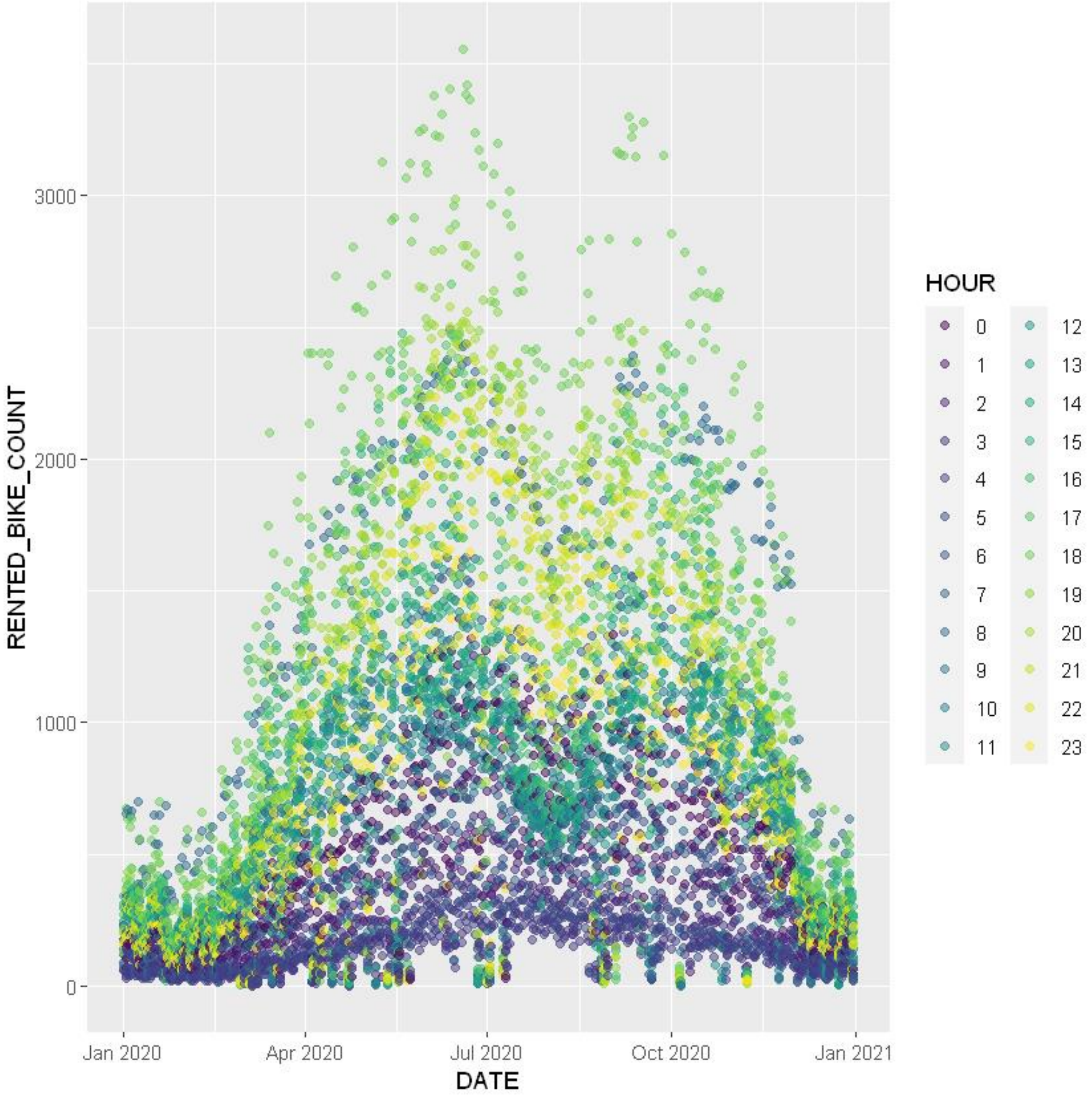
The patterns from the chart shown that there is high demand of bike in the middle of the years and low demand in the first and last years.

We can try to correlate it with the seasons where the demand of bike is high in summer and autumn season while the demand is low in the winter and some first day in spring season. Therefore season can influence people decision to rent a bike.



Bike rental vs. Datetime

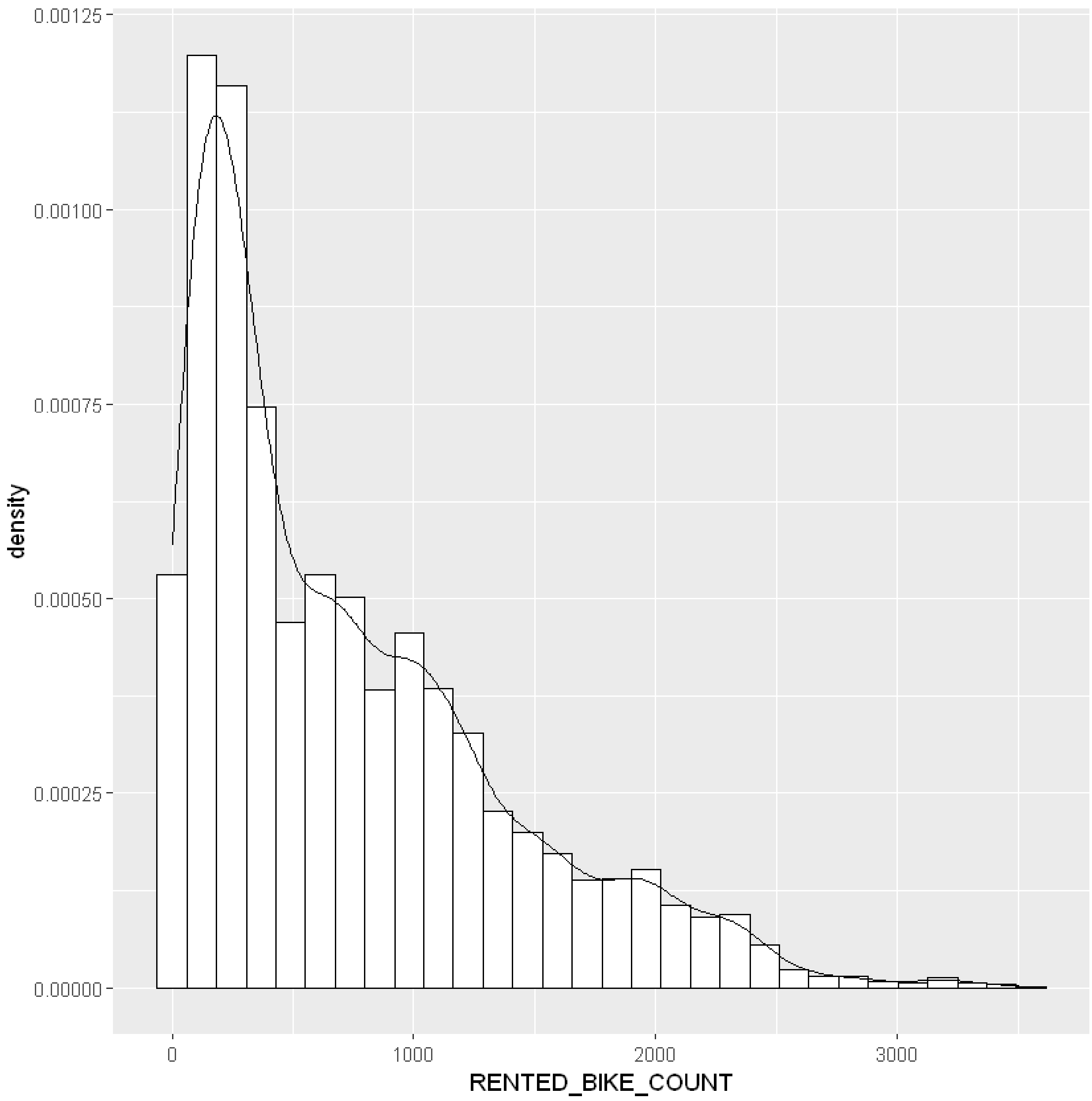
If we break down the data by hour of day, many of rent transaction happened mostly in the daylight hour or near midday. And the number of rent decrease in the evening and eventually very low after midnight hour.



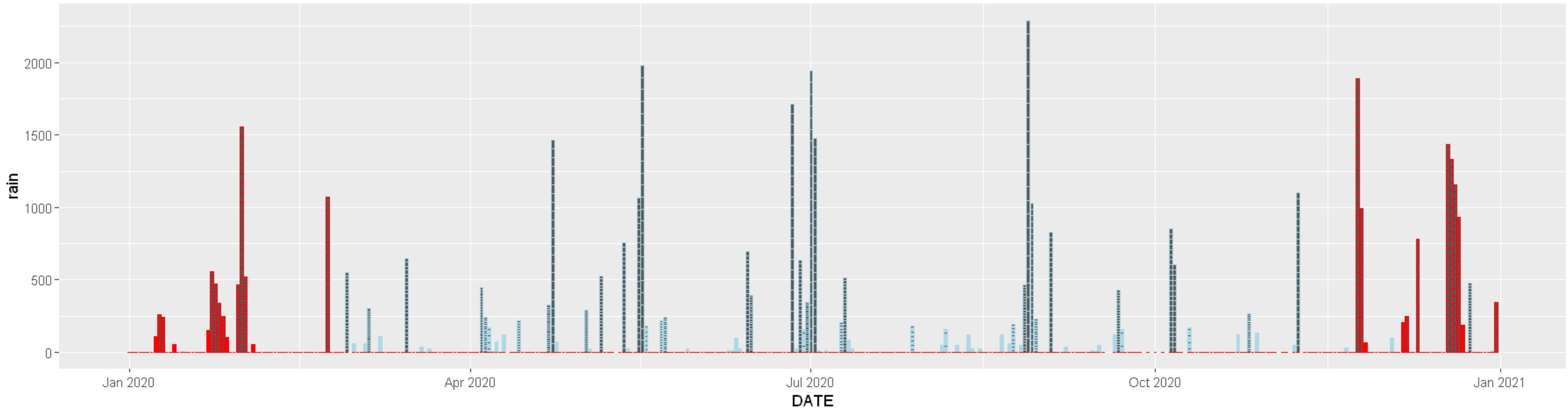
Bike rental histogram

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.

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



While the snowfall (red line) happen mostly in winter, rainfall (blue line) distribute in other season but not specific in one season. There are some day who have a high rainfall and others have lower rainfall. Highest rainfall happen between August and September 2020 and highest snowfall about in December 2020.

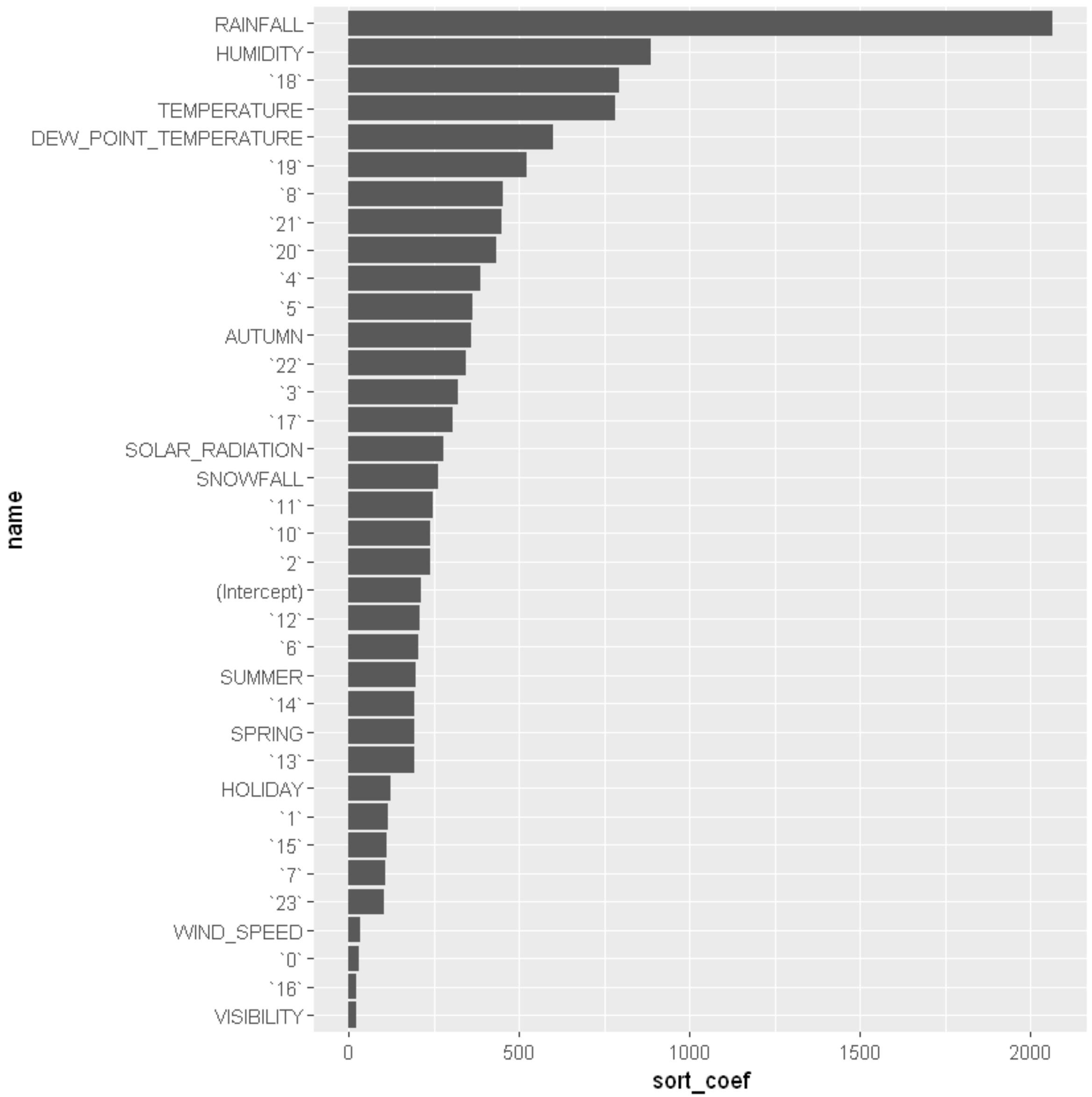
Predictive analysis

Ranked coefficients

From the coefficient of the model, we can find the variable independent that have high correlation with the dependent data or number of rented bikes.

Most of the highest predictors are weather variables such as Rainfall, Humidity, Temperature, and Dew Point Temperature. It means weather condition could influence people decision to rent bikes.

Another significant time is most evening time is highly correlated with higher number of bike rents.



Model evaluation

Here is the result of RMSE and R-Square of each models that created for the estimation

Model Poly

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rsq	standard	0.7069924

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rmse	standard	345.3355

Model Interaction

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rsq	standard	0.6828111

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rmse	standard	356.4426

Model GLM&inter

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rsq	standard	0.6841191

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rmse	standard	356.0645

Model Qubic&inter

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rsq	standard	0.6930281

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rmse	standard	350.6589

Model GLM &Poly&Inter

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rsq	standard	0.7491291

A tibble: 1 × 3

.metric <chr>	.estimator <chr>	.estimate <dbl>
rmse	standard	317.0614

Find the best performing model

The best model will be used to predict the within sample data or test data to check whether the model could accurately predict the test_data observation value.

Model GLM &Poly&Inter

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.7491291

A tibble: 1 × 3

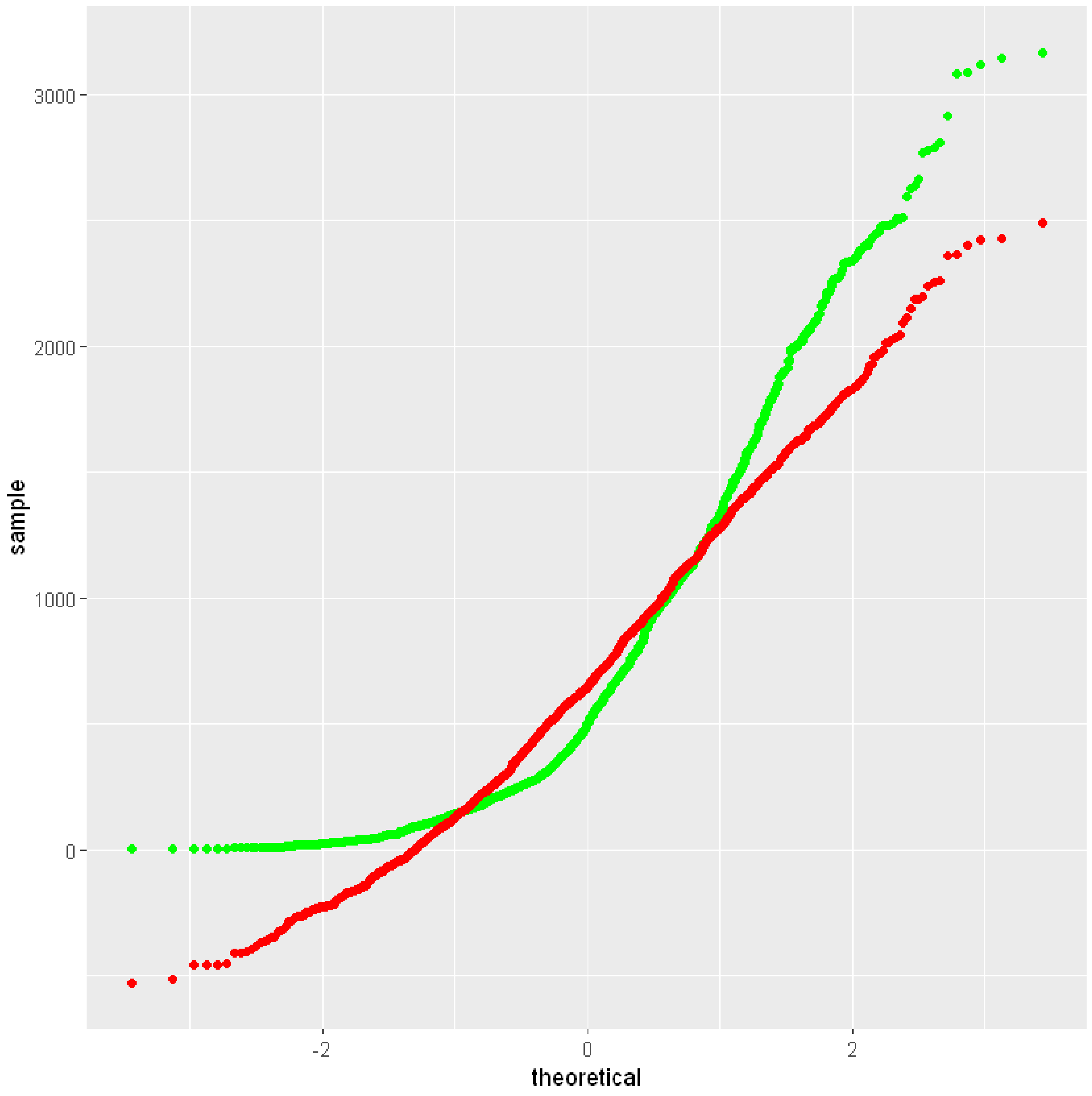
.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	317.0614

Model Equation:

```
lm_2 <- spec_2 %>%
  fit(RENTED_BIKE_COUNT ~ poly(RAINFALL,4) + poly(HUMIDITY,4) + poly(TEMPERATURE,4) + poly(DEW_POINT_TEMPERATURE,4) +
    poly(RAINFALL,3) + poly(HUMIDITY,3) + poly(TEMPERATURE,3) + poly(DEW_POINT_TEMPERATURE,3) + poly(RAINFALL,2) +
    poly(HUMIDITY,2) + poly(TEMPERATURE,2)+ poly(DEW_POINT_TEMPERATURE,2) + poly(SOLAR_RADIATION,2) + poly(SNOWFALL,2) +
    RAINFALL*HUMIDITY + RAINFALL*TEMPERATURE + RAINFALL*DEW_POINT_TEMPERATURE + RAINFALL*SOLAR_RADIATION +
    HUMIDITY*TEMPERATURE + HUMIDITY*DEW_POINT_TEMPERATURE + HUMIDITY*SOLAR_RADIATION + HUMIDITY*SNOWFALL
    + ., data = train_data)
```

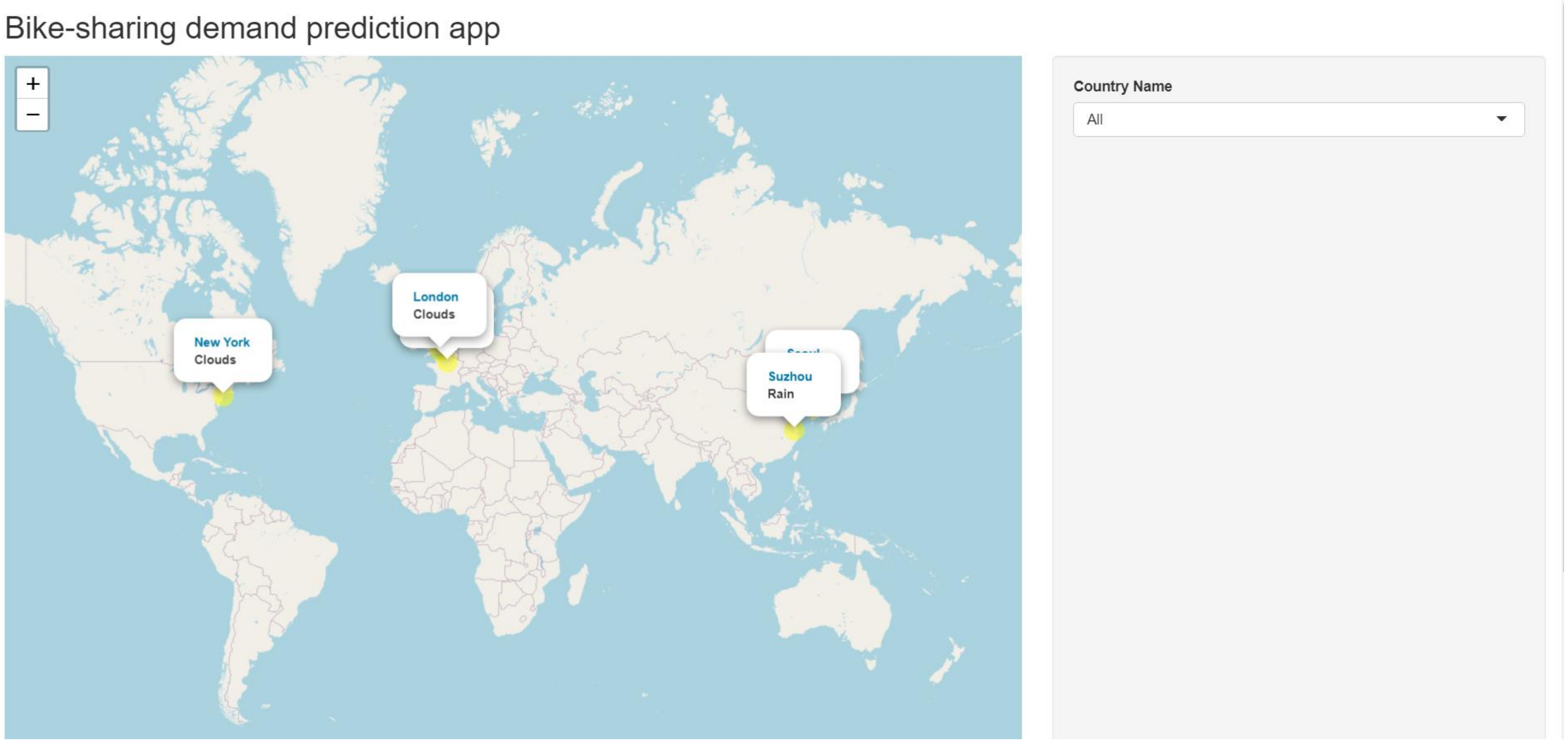
Q-Q plot of the best model

Plot the Q-Q plot of the best model's test results vs the truths



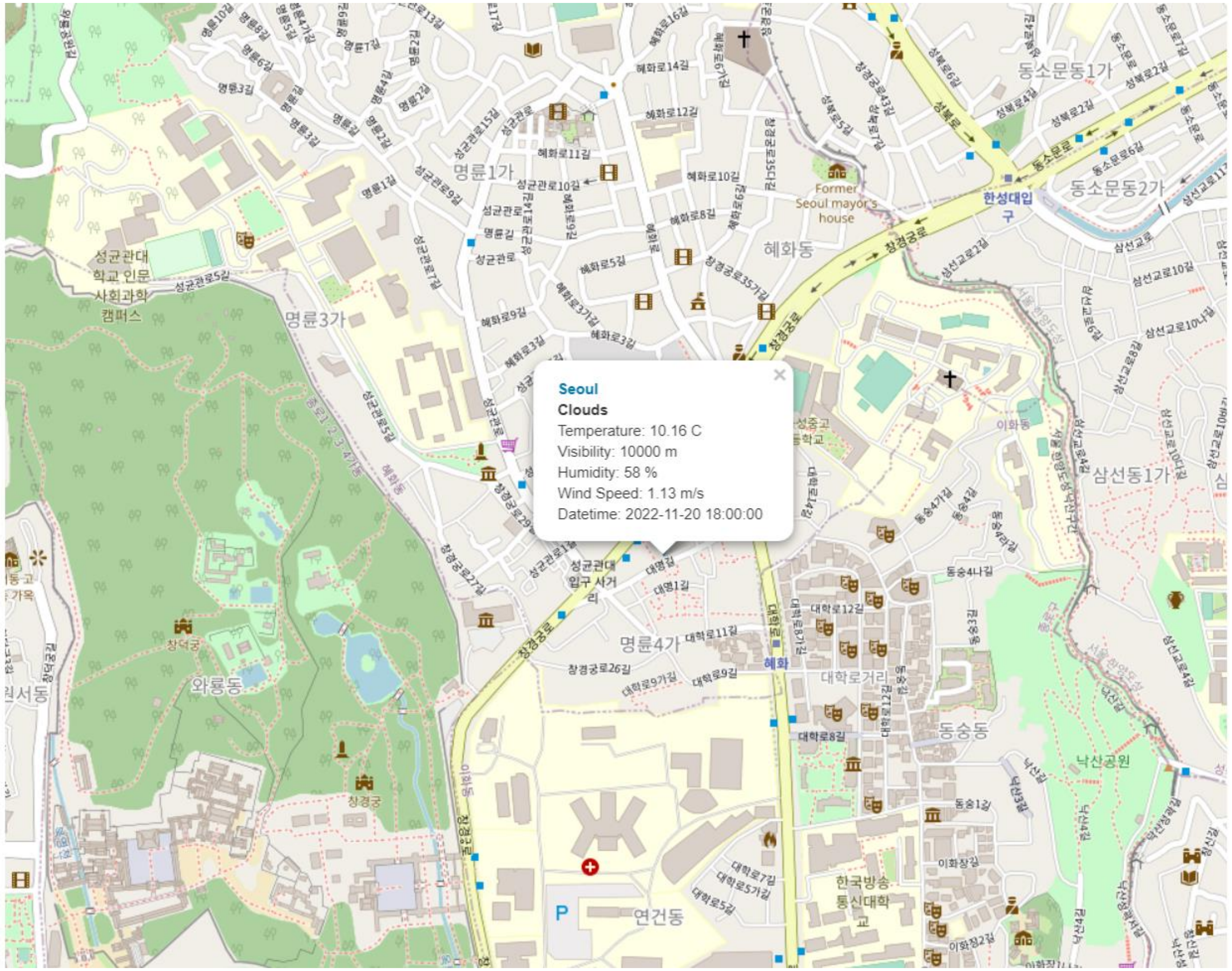
Dashboard

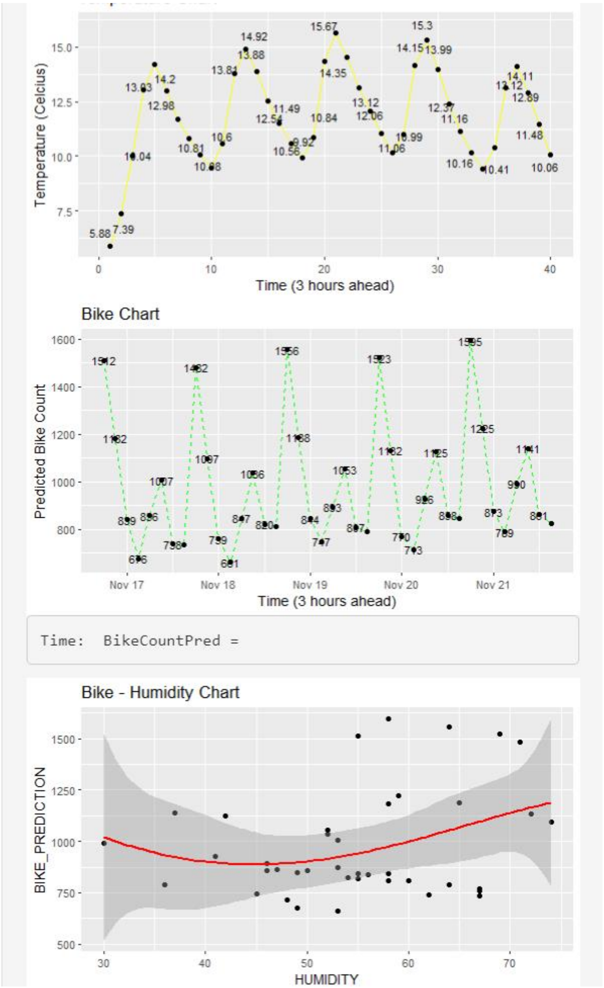
MAIN PAGE OF THE DASHBOARD



This main default page show cities that we can select to get detaild information from and the country name select "All" rather than the name of specific cities itself.

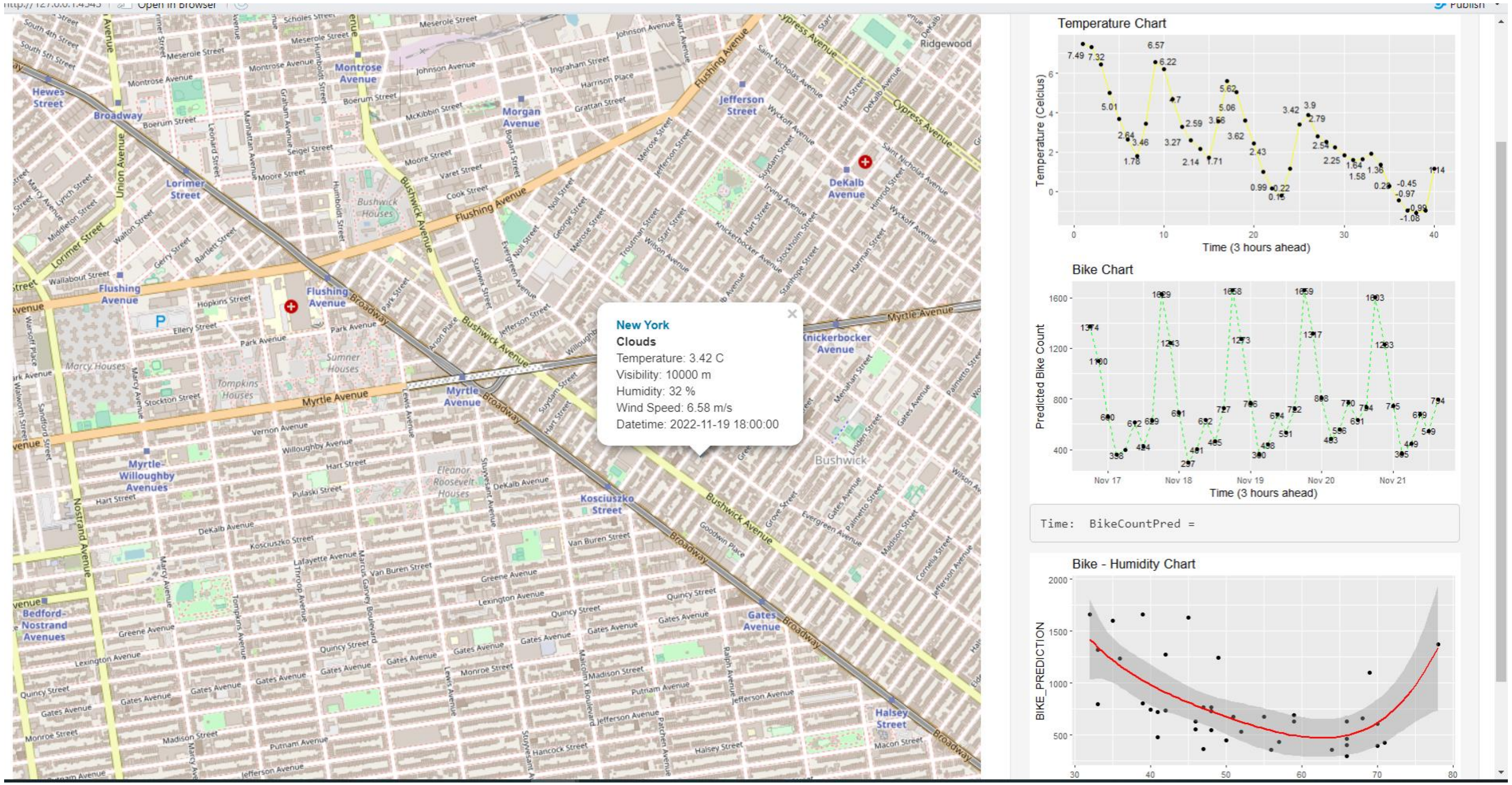
SELECTED CITY: SEOUL





In city specific page, we can get detailed weather conditions all the way to prediction of weather, bike rented and the correlation between

SELECTED CITY: NEW YORK

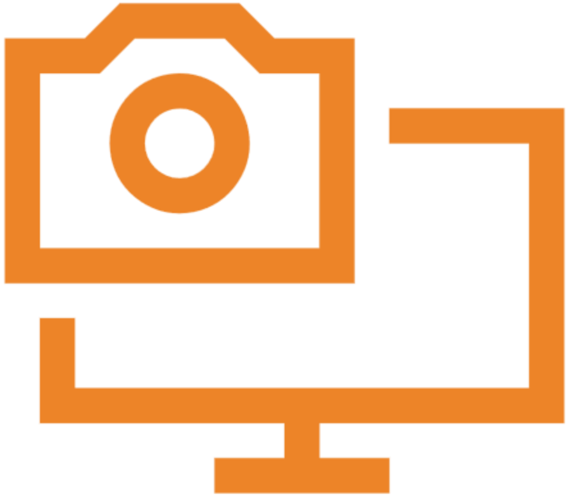


In city specific page, we can get detailed weather conditions all the way to prediction of weather, bike rented and the correlation between

CONCLUSION

- **Most of the rent activity happens in summer**. Supported by weather seasonality, rent seasonality, Summer have the highest number bike and winter have the lowest bike rented.
- **Average number of bikes rented have a positive correlation with temperature**. Higher the temperature, higher the number of bikes rented.
- The patterns from the chart shown that there is **high demand of bike in the middle of the years and low demand in the first and last years.**
- We can try to correlate it with the seasons where the demand of bike is high in summer and autumn season while the demand is low in the winter and some first day in spring season. Therefore **season can influence people decision to rent a bike.**

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 (Screenshot Weather API)

1.3 Coding Practice: Get the current weather data for a city using OpenWeather API

First import `httr` library

n [2]:
```
# Check if need to install rvest` library
require("httr")

library(httr)
library(dplyr)
```
executed in 23ms, finished 18:23:05 2022-11-16

The API base URL to get current weather is https://api.openweathermap.org/data/2.5/weather

n [3]:
```
# URL for Current Weather API
current_weather_url <- 'https://api.openweathermap.org/data/2.5/weather'
```
executed in 32ms, finished 17:22:38 2022-11-11

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

n [4]:
```
# need to be replaced by your real API key
your_api_key <- "b90377239a6120278c48a4da5a18e23b"
# 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")
```
executed in 32ms, finished 17:22:39 2022-11-11

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

n [5]:
```
response <- GET(current_weather_url, query=current_query)
```
executed in 553ms, finished 17:22:40 2022-11-11

Data collection (Screenshot Weather API)

```
In [90]:
# 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 = "b90377239a6120278c48a4da5a18e23b", units="metric")
        # Make HTTP GET call for the given city
        response <- GET(forecast_url, query=forecast_query)
        json_result <- content(response, as="parsed")
        # Note that the 5-day forecast JSON result is a list of lists. You can print the reponse to check the results
        #results <- json_list$list
        results <- json_result$list
        # Loop the json result
        for(result in results) {
            city <- c(city, city_name)
            weather <- c(weather, result$weather[[1]]$main)
            visibility <- c(visibility, result$visibility)
            temp <- c(temp, result$main$temp)
            temp_min <- c(temp_min, result$main$temp_min)
            temp_max <- c(temp_max, result$main$temp_max)
            pressure <- c(pressure, result$main$pressure)
            humidity <- c(humidity, result$main$humidity)
            wind_speed <- c(wind_speed, result$wind$speed)
            wind_deg <- c(wind_deg, result$wind$deg)
            forecast_datetime <- c(forecast_datetime, result$dt_txt)
            date <- format(as.Date(result$dt_txt, format="%Y-%m-%d"),"%m")
            season_date <- case_when(
                date == 12 | date == 1 | date == 2 ~ "Winter",
                date == 3 | date == 4 | date == 5 ~ "Spring",
                date == 6 | date == 7 | date ==8 ~ "Summer",
                date == 9 | date == 10 | date == 11 ~ "Autumn",
                )
            season <- c(season, season_date)
        }

        # 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,
                        season = season,
                        forecast_datetime = forecast_datetime


        )
    }

    # 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 [92]:
cities <- c("Seoul", "Washington, D.C.", "Paris", "Suzhou")
cities_weather_df <- get_weather_forecaset_by_cities(cities)
```

executed in 319ms, finished 18:33:37 2022-11-11

```
In [93]:
head(cities_weather_df)
```

executed in 1.66s, finished 18:33:39 2022-11-11

A data.frame: 6 × 12

	city	weather	visibility	temp	temp_min	temp_max	pressure	humidity	wind_speed	wind_deg	season	forecast_datetime
	<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<int>	<int>	<dbl>	<int>	<chr>	<chr>
1	Seoul	Clear	10000	12.75	12.75	16.16	1024	67	0.56	85	Autumn	2022-11-11 12:00:00
2	Seoul	Clear	10000	13.48	13.48	14.94	1024	64	0.92	99	Autumn	2022-11-11 15:00:00
3	Seoul	Clear	10000	13.61	13.61	14.04	1023	65	1.19	100	Autumn	2022-11-11 18:00:00
4	Seoul	Clouds	10000	13.24	13.24	13.24	1021	68	1.48	96	Autumn	2022-11-11 21:00:00
5	Seoul	Clouds	10000	14.51	14.51	14.51	1021	63	1.56	105	Autumn	2022-11-12 00:00:00
6	Seoul	Clear	10000	18.92	18.92	18.92	1018	49	2.18	151	Autumn	2022-11-12 03:00:00

```
In [94]:
# Write cities_weather_df to `cities_weather_forecast.csv`
write.csv(cities_weather_df, "raw_cities_weather_forecast.csv", row.names=FALSE)
```

executed in 35ms, finished 18:33:45 2022-11-11

For more details about HTTP requests with `httr`, please refer to the previous HTTP request notebook here:

[HTTP request in R]

2.1 TASK: Download datasets as csv files from cloud storage

The last task of this lab is straightforward: download some aggregated datasets from cloud storage

```
In [21]:
# 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-SkillsNetwork/I
# 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-SkillsNetwork/I
# download the file
download.file(url, destfile = "raw_seoul_bike_sharing.csv")
```

executed in 6.20s, finished 00:24:25 2022-11-11

Data collection (Screenshot Scraping)

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 [3]:
```r
url <- "https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems"
# Get the root HTML node by calling the `read_html()` method with URL
html_node <- read_html(url)
html_node
```
executed in 1.35s, finished 18:51:36 2022-11-11

```
{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="skin-vector-legacy mediawiki ltr sitedir-ltr mw-hide-empty-e ...
```

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_node, "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")

You can use a `for` loop to print each table, and then you will see that the actual the bike sharing table is the second element `table_nodes[[2]]`.

Next, you need to convert this HTML table into a data frame using the `html_table()` function. You may choose to include `fill = TRUE` argument to fill any empty table rows/columns.

In [16]:
```r
# Convert the bike-sharing system table into a dataframe
table_node <- html_node(html_node, "table")
bike_df_1 <- html_table(table_node, fill = T)
head(bike_df_1)
```
executed in 204ms, finished 18:59:13 2022-11-11

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	

Summarize the bike sharing system data frame

In [17]:
```r
# Summarize the dataframe
summary(bike_df_1)
```
executed in 22ms, finished 18:59:15 2022-11-11

```
   Country             City               Name              System         
 Length:539         Length:539         Length:539         Length:539        
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
   Operator           Launched         Discontinued         Stations        
 Length:539         Length:539         Length:539         Length:539        
 Class :character   Class :character   Class :character   Class :character  
 Mode  :character   Mode  :character   Mode  :character   Mode  :character  
   Bicycles         Daily ridership   
 Length:539         Length:539        
 Class :character   Class :character  
 Mode  :character   Mode  :character  
```

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

In [18]:
```r
# Export the dataframe into a csv file
write.csv(bike_df_1, file = "raw_bike_sharing_systems.csv", row.names = F)
```
executed in 26ms, finished 18:59:23 2022-11-11

For more details about webscraping with `rvest`, please refer to the previous webscraping notebook here:

Webscraping in R

Data wrangling (Screenshot regex)

1.2 TASK: Standardize column names for all collected datasets

In the previous data collection labs, you collected four datasets in csv format:

- `raw_bike_sharing_systems.csv` : A list of active bike-sharing systems across the world
- `raw_cities_weather_forecast.csv` : 5-day weather forecasts for a list of cities, from OpenWeather API
- `raw_worldcities.csv` : A list of major cities' info (such as name, latitude and longitude) across the world
- `raw_seoul_bike_sharing.csv` : Weather information (Temperature, Humidity, Windspeed, Visibility, Dewpoint, Solar radiation, Snowfall, Rainfall), the number of bikes rented per hour, and date information, from Seoul bike-sharing systems

Optional: If you had some difficulties finishing the data collection labs, you may download the datasets directly from the following URLs:

2]:
```
# # Download raw_bike_sharing_systems.csv
# url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork
# download.file(url, destfile = "raw_bike_sharing_systems.csv")

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

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

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

executed in 15.4s, finished 17:19:07 2022-11-11

To improve dataset readbility by both human and computer systems, we first need to standardize the column names of the datasets above using the following naming convention:

- Column names need to be UPPERCASE
- The word separator needs to be an underscore, such as in `COLUMN_NAME`

You can use the following dataset list and the `names()` function to get and set each of their column names, and convert them according to our defined naming convention.

5]:
```
dataset_list <- c('raw_bike_sharing_systems.csv', 'raw_seoul_bike_sharing.csv', 'raw_cities_weather_forecast.csv', 'raw_world
```

executed in 20ms, finished 21:33:40 2022-11-11

```
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
    names(dataset) <- str_replace_all(names(dataset), " ", "_")
    # Save the dataset
    write.csv(dataset, dataset_name, row.names=FALSE)
}
```

executed in 2.27s, finished 21:33:44 2022-11-11

```
Rows: 539 Columns: 10
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (10): Country, City, Name, System, Operator, Launched, Discontinued, Sta...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 8760 Columns: 14
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): DATE, SEASONS, HOLIDAY, FUNCTIONING_DAY
dbl (10): RENTED_BIKE_COUNT, HOUR, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBI...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 160 Columns: 12
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (3): CITY, WEATHER, SEASON
dbl  (8): VISIBILITY, TEMP, TEMP_MIN, TEMP_MAX, PRESSURE, HUMIDITY, WIND_SPE...
dttm (1): FORECAST_DATETIME

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 26569 Columns: 11
-- Column specification --------------------------------------------------------
Delimiter: ","
chr (7): CITY, CITY_ASCII, COUNTRY, ISO2, ISO3, ADMIN_NAME, CAPITAL
dbl (4): LAT, LNG, POPULATION, ID

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

TODO: Read the resulting datasets back and check whether their column names follow the naming convention

Data wrangling (Screenshot regex)

TODO: Read the resulting datasets back and check whether their column names follow the naming convention

```
for(dataset_name in dataset_list){
    # Print a summary for each data set to check whether the column names were correctly converted
    dataset <- read_csv(dataset_name)
    summary(dataset)
}
```

executed in 4.11s, finished 21:33:46 2022-11-11

```
Rows: 539 Columns: 10
-- Column specification ------------------------------------------------------------
Delimiter: ","
chr (10): COUNTRY, CITY, NAME, SYSTEM, OPERATOR, LAUNCHED, DISCONTINUED, STA...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 8760 Columns: 14
-- Column specification ------------------------------------------------------------
Delimiter: ","
chr  (4): DATE, SEASONS, HOLIDAY, FUNCTIONING_DAY
dbl (10): RENTED_BIKE_COUNT, HOUR, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBI...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 160 Columns: 12
-- Column specification ------------------------------------------------------------
Delimiter: ","
chr  (3): CITY, WEATHER, SEASON
dbl  (8): VISIBILITY, TEMP, TEMP_MIN, TEMP_MAX, PRESSURE, HUMIDITY, WIND_SPE...
dttm (1): FORECAST_DATETIME

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 26569 Columns: 11
-- Column specification ------------------------------------------------------------
Delimiter: ","
chr (7): CITY, CITY_ASCII, COUNTRY, ISO2, ISO3, ADMIN_NAME, CAPITAL
dbl (4): LAT, LNG, POPULATION, ID

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

1.3 Process the web-scraped bike sharing system dataset

By now we have standardized all column names. Next, we will focus on cleaning up the values in the web-scraped bike sharing systems dataset.

```
# First load the dataset
bike_sharing_df <- read_csv("raw_bike_sharing_systems.csv")
```

executed in 6.19s, finished 21:33:49 2022-11-11

```
Rows: 539 Columns: 10
-- Column specification ------------------------------------------------------------
Delimiter: ","
chr (10): COUNTRY, CITY, NAME, SYSTEM, OPERATOR, LAUNCHED, DISCONTINUED, STA...

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

```
# Print its head
head(bike_sharing_df)
```

executed in 8.58s, finished 21:33:51 2022-11-11

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	NA	NA	March 2011	NA	8	200	NA
Argentina	Buenos Aires[6][7]	Ecobici	Serttel Brasil[8]	Bike In Baires Consortium[9]	2010	NA	400	4000	21917
Argentina	Mendoza[10]	Metrobici	NA	NA	2014	NA	2	40	NA
Argentina	Rosario	Mi Bici Tu Bici[11]	NA	NA	2 December 2015	NA	47	480	NA
Argentina	San Lorenzo, Santa Fe	Biciudad	Biciudad	NA	27 November 2016	NA	8	80	NA
Australia	Melbourne[12]	Melbourne Bike Share	PBSC & 8D	Motivate	June 2010	30 November 2019[13]	53	676	NA

Even from the first few rows, you can see there is plenty of undesireable embedded textual content, such as the reference link included in `Melbourne[12]`.

In this project, let's only focus on processing the following revelant columns (feel free to process the other columns for more practice):

Data wrangling (Screenshot regex)

In [170]:
```
# Select the four columns
sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)
```
executed in 10.2s, finished 21:33:54 2022-11-11

Let's see the types of the selected columns

In [171]:
```
sub_bike_sharing_df %>%
    summarize_all(class) %>%
    gather(variable, class)
```
executed in 12.1s, finished 21:33:56 2022-11-11

A tibble: 4 × 2

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

They are all interpreted as character columns, but we expect the `BICYCLES` column to be of numeric type. Let's see why it wasn't loaded as a numeric column - possibly some entries contain characters. Let's create a simple function called `find_character` to check that.

In [172]:
```
# grepl searches a string for non-digital characters, and returns TRUE or FALSE
# if it finds any non-digital characters, then the bicyle column is not purely numeric
find_character <- function(strings) grepl("[^0-9]", strings) #this regex means matches anything except 0-9
```
executed in 14.0s, finished 21:33:58 2022-11-11

Let's try to find any elements in the `Bicycles` column containing non-numeric characters.

In [173]:
```
sub_bike_sharing_df %>%
    select(BICYCLES) %>%
    filter(find_character(BICYCLES)) %>%
    slice(0:10)
```
executed in 16.0s, finished 21:34:00 2022-11-11

A tibble: 10 × 1

BICYCLES
<chr>
1790 (2019)[21]
4200 (2021)
4115[25]

In [174]:
```
# Define a 'reference link' character class,
# `[A-z0-9]` means at least one character
# `\\[` and `\\]` means the character is wrapped by [], such as for [12] or [abc]
ref_pattern <- "\\[[A-z0-9]+\\]"
find_reference_pattern <- function(strings) grepl(ref_pattern, strings)
```
executed in 17.8s, finished 21:34:03 2022-11-11

In [175]:
```
# Check whether the COUNTRY column has any reference links
sub_bike_sharing_df %>%
    select(COUNTRY) %>%
    filter(find_reference_pattern(COUNTRY)) %>%
    slice(0:10)
```
executed in 20.7s, finished 21:34:06 2022-11-11

A tibble: 0 ×
1

COUNTRY
<chr>

Ok, looks like the `COUNTRY` column is clean. Let's check the `CITY` column.

In [176]:
```
# Check whether the CITY column has any reference links
sub_bike_sharing_df %>%
    select(CITY) %>%
    filter(find_reference_pattern(CITY)) %>%
    slice(0:10)
```
executed in 22.1s, finished 21:34:09 2022-11-11

A tibble: 10 × 1

CITY
<chr>
Tirana[5]
Buenos Aires[6][7]
Mendoza[10]

Hmm, looks like the `CITY` column has some reference links to be removed. Next, let's check the `SYSTEM` column.

In [177]:
```
# Check whether the System column has any reference links
sub_bike_sharing_df %>%
    select(SYSTEM) %>%
    filter(find_reference_pattern(SYSTEM)) %>%
    slice(0:10)
```
executed in 23.4s, finished 21:34:11 2022-11-11

A tibble: 8 × 1

SYSTEM
<chr>
Serttel Brasil[8]
EasyBike[64]
4 Gen.[72]
3 Gen. SmooveKey[135]
3 Gen. Smoove[162][163][164][160]
3 Gen. Smoove[200]
3 Gen. Smoove[202]
3 Gen. Smoove[204]

So the `SYSTEM` column also has some reference links.

After some preliminary investigations, we identified that the `CITY` and `SYSTEM` columns have some undesired reference links, and the `BICYCLES` column has both reference links and some textual annotations.

Next, you need to use regular expressions to clean up the unexpected reference links and text annotations in numeric values.

Data wrangling (Screenshot regex)

2 TASK: Remove undesired reference links using regular expressions

TODO: Write a custom function using `stringr::str_replace_all` to replace all reference links with an empty character for columns `CITY` and `SYSTEM`

In [178]:
```
# remove reference link
remove_ref <- function(strings) {
    ref_pattern <- "\\[[A-z0-9]+\\]"
    # Replace all matched substrings with a white space using str_replace_all()
    strings <- str_replace_all(strings, ref_pattern, "\t")
    # Trim the reslt if you want
#     string_trim <- trimws(strings, which = c("both"),  whitespace = "[ \t\r\n]")
    # return(result)
    return(strings)
}
```
executed in 21ms, finished 21:34:19 2022-11-11

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

In [179]:
```
# sub_bike_sharing_df %>% mutate(column1=remove_ref(column1), ... )
result <- sub_bike_sharing_df %>% mutate(CITY = remove_ref(CITY),
                                         SYSTEM = remove_ref(SYSTEM), BICYCLES = remove_ref(BICYCLES))
```
executed in 720ms, finished 21:34:21 2022-11-11

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

In [180]:
```
result %>%
    select(CITY, SYSTEM, BICYCLES) %>%
    filter(find_reference_pattern(CITY) | find_reference_pattern(SYSTEM) | find_reference_pattern(BICYCLES))
```
executed in 1.18s, finished 21:34:23 2022-11-11

A tibble: 0 × 3

CITY	SYSTEM	BICYCLES
<chr>	<chr>	<chr>

In [181]:
```
summary(result$BICYCLES)
```
executed in 21ms, finished 21:35:09 2022-11-11

```
   Length     Class      Mode 
      539 character character 
```

3 TASK: Extract the numeric value using regular expressions

TODO: Write a custom function using `stringr::str_extract` to extract the first digital substring match and convert it into numeric type For example, extract the value '32' from `32 (including 6 rollers) [162]`.

In [182]:
```
# Extract the first number
extract_num <- function(columns){
    # Define a digital pattern
    digitals_pattern <- "[0-9]+"
    # Find the first match using str_extract
    columns <- str_extract(columns, digitals_pattern)
    # Convert the result to numeric using the as.numeric() function
    columns <- as.numeric(columns)
}
```
executed in 18ms, finished 21:36:23 2022-11-11

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

In [183]:
```
# Use the mutate() function on the BICYCLES column
result <- result %>% mutate(BICYCLES = extract_num(BICYCLES))
```
executed in 19ms, finished 21:36:34 2022-11-11

In [193]:
```
# ref_pattern <- "[^0-9]"
# find_reference_pattern <- function(strings) grepl(ref_pattern, strings)

# result %>%
#     select(BICYCLES) %>%
#     filter(find_reference_pattern(BICYCLES))
```
executed in 16ms, finished 21:39:55 2022-11-11

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

In [194]:
```
summary(result$BICYCLES)
```
executed in 20ms, finished 21:39:58 2022-11-11

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
      4      75     300    1892    1200   78000      86 
```

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

In [197]:
```
# Write dataset to `bike_sharing_systems.csv`
write.csv(result, file = "bike_sharing_systems.csv", row.names = F)
```
executed in 27ms, finished 21:41:27 2022-11-11

Data wrangling (Screenshot dplyr)

1.1 Lab Overview:

In this lab, you will focus on wrangling the Seoul bike-sharing demand historical dataset. This is the core dataset to build a predictive model later.

It contains the following columns:

- `DATE` : Year-month-day
- `RENTED BIKE COUNT` - Count of bikes rented at each hour
- `HOUR` - Hour of he day
- `TEMPERATURE` - Temperature in Celsius
- `HUMIDITY` - Unit is `%`
- `WINDSPEED` - Unit is `m/s`
- `VISIBILITY` - Multiplied by 10m
- `DEW POINT TEMERATURE` - The temperature to which the air would have to cool down in order to reach saturation, unit is Celsius
- `SOLAR RADIATION` - MJ/m2
- `RAINFALL` - mm
- `SNOWFALL` - cm
- `SEASONS` - Winter, Spring, Summer, Autumn
- `HOLIDAY` - Holiday/No holiday
- `FUNCTIONAL DAY` - NoFunc(Non Functional Hours), Fun(Functional hours)

For this dataset, you will be asked to use `tidyverse` to perform the following data wrangling tasks:

- **TASK: Detect and handle missing values**
- **TASK: Create indicator (dummy) variables for categorical variables**
- **TASK: Normalize data**

Let's start!

First import the necessary library for this data wrangling task:

20]:
```
# Check if you need to install the `tidyverse` library
require("tidyverse")
library(tidyverse)
```
executed in 32ms, finished 13:24:35 2022-11-12

Then load the bike-sharing system data from the csv processed in the previous lab:

21]:
```
bike_sharing_df <- read_csv("raw_seoul_bike_sharing.csv")
```
executed in 109ms, finished 13:24:35 2022-11-12

```
Rows: 8760 Columns: 14
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): DATE, SEASONS, HOLIDAY, FUNCTIONING_DAY
dbl (10): RENTED_BIKE_COUNT, HOUR, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBI...

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

First take a quick look at the dataset.

In [23]:
```
summary(bike_sharing_df)
dim(bike_sharing_df)
```
executed in 31ms, finished 13:24:36 2022-11-12

```
     DATE           RENTED_BIKE_COUNT      HOUR          TEMPERATURE
 Length:8760        Min.   :   2.0     Min.   : 0.00   Min.   :-17.80
 Class :character   1st Qu.: 214.0     1st Qu.: 5.75   1st Qu.:  3.40
 Mode  :character   Median : 542.0     Median :11.50   Median : 13.70
                    Mean   : 729.2     Mean   :11.50   Mean   : 12.87
                    3rd Qu.:1084.0     3rd Qu.:17.25   3rd Qu.: 22.50
                    Max.   :3556.0     Max.   :23.00   Max.   : 39.40
                    NA's   :295                        NA's   :11
    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.: 940   1st Qu.: -4.700
 Median :57.00   Median :1.500   Median :1698   Median :  5.100
 Mean   :58.23   Mean   :1.725   Mean   :1437   Mean   :  4.074
 3rd Qu.:74.00   3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 14.800
 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:8760
 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   :0.5691   Mean   : 0.1487   Mean   :0.07507
 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:8760        Length:8760
 Class :character   Class :character
 Mode  :character   Mode  :character
```

8760 · 14

From the summary, we can observe that:

Columns `RENTED_BIKE_COUNT`, `TEMPERATURE`, `HUMIDITY`, `WIND_SPEED`, `VISIBILITY`, `DEW_POINT_TEMPERATURE`, `SOLAR_RADIATION`, `RAINFALL`, `SNOWFALL` are numerical variables/columns and require normalization. Moreover, `RENTED_BIKE_COUNT` and `TEMPERATURE` have some missing values (NA's) that need to be handled properly.

`SEASONS`, `HOLIDAY`, `FUNCTIONING_DAY` are categorical variables which need to be converted into indicator columns or dummy variables. Also, `HOUR` is read as a numerical variable but it is in fact a categorical variable with levels ranging from 0 to 23.

Now that you have some basic ideas about how to process this bike-sharing demand dataset, let's start working on it!

Data wrangling (Screenshot dplyr)

2 TASK: Detect and handle missing values

The `RENTED_BIKE_COUNT` column has about 295 missing values, and `TEMPERATURE` has about 11 missing values. Those missing values could be caused by not being recorded, or from malfunctioning bike-sharing systems or weather sensor networks. In any cases, the identified missing values have to be properly handled.

Let's first handle missing values in `RENTED_BIKE_COUNT` column:

Considering `RENTED_BIKE_COUNT` is the response variable/dependent variable, i.e., we want to predict the `RENTED_BIKE_COUNT` using other predictor/independent variables later, and we normally can not allow missing values for the response variable, so missing values for response variable must be either dropped or imputed properly.

We can see that `RENTED_BIKE_COUNT` only has about 3% missing values (295 / 8760). As such, you can safely drop any rows whose `RENTED_BIKE_COUNT` has missing values.

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

]:
```r
# Drop rows with `RENTED_BIKE_COUNT` column == NA
bike_sharing_df <- bike_sharing_df %>%
                   drop_na(RENTED_BIKE_COUNT)
```
executed in 24ms, finished 13:24:38 2022-11-12

]:
```r
# Print the dataset dimension again after those rows are dropped
dim(bike_sharing_df)
```
executed in 80ms, finished 13:24:38 2022-11-12

8465 · 14

Now that you have handled missing values in the `RENTED_BIKE_COUNT` variable, let's continue processing missing values for the `TEMPERATURE` column.

Unlike the `RENTED_BIKE_COUNT` variable, `TEMPERATURE` is not a response variable. However, it is still an important predictor variable - as you could imagine, there may be a positve correlation between `TEMPERATURE` and `RENTED_BIKE_COUNT`. For example, in winter time with lower temperatures, people may not want to ride a bike, while in summer with nicer weather, they are more likely to rent a bike.

How do we handle missing values for `TEMPERATURE`? We could simply remove the rows but it's better to impute them because `TEMPERATURE` should be relatively easy and reliable to estimate statistically.

Let's first take a look at the missing values in the TEMPERATURE column.

]:
```r
bike_sharing_df %>%
                filter(is.na(TEMPERATURE))
```
executed in 37ms, finished 13:24:40 2022-11-12

A tibble: 11 × 14

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFA
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dl
07/06/2018	3221	18	NA	57	2.7	1217	16.4	0.96	[illegible]
12/06/2018	1246	14	NA	45	2.2	1961	12.7	1.39	[illegible]
13/06/2018	2664	17	NA	57	3.3	919	16.4	0.87	[illegible]
17/06/2018	2330	17	NA	58	3.3	865	16.7	0.66	[illegible]
20/06/2018	2741	19	NA	61	2.7	1236	17.5	0.60	[illegible]
30/06/2018	1144	13	NA	87	1.7	390	23.2	0.71	[illegible]
05/07/2018	827	10	NA	75	1.1	1028	20.8	1.22	[illegible]
11/07/2018	634	9	NA	96	0.6	450	24.9	0.41	[illegible]
12/07/2018	593	6	NA	93	1.1	852	24.3	0.01	[illegible]
21/07/2018	347	4	NA	77	1.2	1203	21.2	0.00	[illegible]
21/08/2018	1277	23	NA	75	0.1	1892	20.8	0.00	[illegible]

It seems that all of the missing values for `TEMPERATURE` are found in `SEASONS == Summer`, so it is reasonable to impute those missing values with the summer average temperature.

TODO: Impute missing values for the TEMPERATURE column using its mean value.

In [27]:
```r
# Calculate the summer average temperature
mean_summer <- bike_sharing_df %>%
    filter(SEASONS == 'Summer') %>%
        summarize(mean_summer = mean(TEMPERATURE, na.rm = T)) %>%
            pull()

mean_summer
```
executed in 36ms, finished 13:24:41 2022-11-12

26.5877105143377

In [28]:
```r
# Calculate the summer average temperature
mean_summer <- bike_sharing_df %>%
    filter(SEASONS == 'Summer') %>%
        summarize(mean_summer = mean(TEMPERATURE, na.rm = T)) %>%
            pull()

mean_summer
```
executed in 43ms, finished 13:24:41 2022-11-12

26.5877105143377

Data wrangling (Screenshot dplyr)

In [30]:
```r
# Impute missing values for TEMPERATURE column with summer average temperature
bike_sharing_df$TEMPERATURE <- bike_sharing_df$TEMPERATURE %>% replace_na(mean_summer)
```
executed in 24ms, finished 13:24:42 2022-11-12

In [31]:
```r
# Print the summary of the dataset again to make sure no missing values in all columns
summary(bike_sharing_df)
```
executed in 27ms, finished 13:24:42 2022-11-12

```
     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
```

In [32]:
```r
# Save the dataset as `seoul_bike_sharing.csv`
write.csv(bike_sharing_df, file = "seoul_bike_sharing.csv", row.names = F)
```
executed in 195ms, finished 13:24:43 2022-11-12

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

Regression models can not process categorical variables directly, thus we need to convert them into indicator variables.

In the bike-sharing demand dataset, `SEASONS`, `HOLIDAY`, `FUNCTIONING_DAY` are categorical variables. Also, `HOUR` is read as a numerical variable but it is in fact a categorical variable with levels ranged from 0 to 23.

TODO: Convert `HOUR` column from numeric into character first:

In [34]:
```r
# Using mutate() function to convert HOUR column into character type
bike_sharing_df <- bike_sharing_df %>%
    mutate(HOUR = as.character(HOUR))

summary(bike_sharing_df$HOUR)
```
executed in 30ms, finished 13:24:54 2022-11-12

```
   Length     Class      Mode
     8465 character character
```

```
   Length     Class      Mode
     8465 character character
```

In [36]:
```r
unique(bike_sharing_df$SEASONS)
unique(bike_sharing_df$HOLIDAY)
unique(bike_sharing_df$FUNCTIONING_DAY)
unique(bike_sharing_df$HOUR)
```
executed in 40ms, finished 13:29:02 2022-11-12

'Winter' · 'Spring' · 'Summer' · 'Autumn'

'No Holiday' · 'Holiday'

'Yes'

'0' · '1' · '2' · '3' · '4' · '5' · '6' · '7' · '8' · '9' · '10' · '11' · '12' · '13' · '14' · '15' · '16' · '17' · '18' · '19' · '20' · '21' · '22' · '23'

`SEASONS`, `HOLIDAY`, `FUNCTIONING_DAY`, `HOUR` are all character columns now and are ready to be converted into indicator variables.

For example, `SEASONS` has four categorical values: `Spring`, `Summer`, `Autumn`, `Winter`. We thus need to create four indicator/dummy variables `Spring`, `Summer`, `Autumn`, and `Winter` which only have the value 0 or 1.

So, given a data entry with the value `Spring` in the `SEASONS` column, the values for the four new columns `Spring`, `Summer`, `Autumn`, and `Winter` will be set to 1 for `Spring` and 0 for the others:

Spring	Summer	Autumn	Winter
1	0	0	0

TODO: Convert `SEASONS`, `HOLIDAY`, `FUNCTIONING_DAY`, and `HOUR` columns into indicator columns.

Note that if `FUNCTIONING_DAY` only contains one categorical value after missing values removal, then you don't need to convert it to an indicator column.

In [52]:
```r
# Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.
bike_sharing_df_2 <- bike_sharing_df %>%
    mutate(dummy = 1) %>%
        spread(key = SEASONS, value = dummy, fill = 0) %>%
    mutate(dummy = 1) %>%
        spread(key = HOLIDAY, value = dummy, fill = 0) %>%
    mutate(dummy = 1) %>%
        spread(key = HOUR, value = dummy, fill = 0)
```
executed in 130ms, finished 13:50:53 2022-11-12

In [53]:
```r
# Print the dataset summary again to make sure the indicator columns are created properly
summary(bike_sharing_df_2)
```
executed in 64ms, finished 13:50:56 2022-11-12

```
     DATE            RENTED_BIKE_COUNT  TEMPERATURE        HUMIDITY
 Length:8465        Min.   :   2.0    Min.   :-17.80   Min.   : 0.00
 Class :character   1st Qu.: 214.0    1st Qu.:  3.00   1st Qu.:42.00
 Mode  :character   Median : 542.0    Median : 13.50   Median :57.00
```

```
       9
 Min.   :0.0000
 1st Qu.:0.0000
 Median :0.0000
 Mean   :0.0417
 3rd Qu.:0.0000
 Max.   :1.0000
```

In [54]:
```r
# Save the dataset as `seoul_bike_sharing_converted.csv`
# write_csv(dataframe, "seoul_bike_sharing_converted.csv")
write.csv(bike_sharing_df_2, file = "seoul_bike_sharing_converted.csv", row.names = F)
```
executed in 456ms, finished 13:51:34 2022-11-12

Data wrangling (Screenshot dplyr)

4 TASK: Normalize data

Columns `RENTED_BIKE_COUNT`, `TEMPERATURE`, `HUMIDITY`, `WIND_SPEED`, `VISIBILITY`, `DEW_POINT_TEMPERATURE`, `SOLAR_RADIATION`, `RAINFALL`, `SNOWFALL` are numerical variables/columns with different value units and range. Columns with large values may adversely influence (bias) the predictive models and degrade model accuracy. Thus, we need to perform normalization on these numeric columns to transfer them into a similar range.

In this project, you are asked to use Min-max normalization:

Min-max rescales each value in a column by first subtracting the minimum value of the column from each value, and then divides the result by the difference between the maximum and minimum values of the column. So the column gets re-scaled such that the minimum becomes 0 and the maximum becomes 1.

$$x\_new = \frac{x\_old - x\_min}{x\_max - x\_min}$$

TODO: Apply min-max normalization on `RENTED_BIKE_COUNT`, `TEMPERATURE`, `HUMIDITY`, `WIND_SPEED`, `VISIBILITY`, `DEW_POINT_TEMPERATURE`, `SOLAR_RADIATION`, `RAINFALL`, `SNOWFALL`

[64]:
```
# 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_3 <- bike_sharing_df_2 %>%
    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)))
```
executed in 52ms, finished 14:33:59 2022-11-12

[65]:
```
# Print the summary of the dataset again to make sure the numeric columns range between 0 and 1
summary(bike_sharing_df_3)
```
executed in 220ms, finished 14:34:02 2022-11-12

```
     DATE            RENTED_BIKE_COUNT   TEMPERATURE          HUMIDITY
```

```
 3rd Qu.:0.0000
 Max.   :1.0000
```

In [66]:
```
# Save the dataset as `seoul_bike_sharing_converted_normalized.csv`
# write_csv(dataframe, "seoul_bike_sharing_converted_normalized.csv")
write.csv(bike_sharing_df_3, file = "seoul_bike_sharing_converted_normalized.csv", row.names = F)
```
executed in 509ms, finished 14:34:50 2022-11-12

4.1 Standardize the column names again for the new datasets

Since you have added many new indicator variables, you need to standardize their column names again by using the following code:

In [67]:
```
# Dataset list
dataset_list <- c('seoul_bike_sharing.csv', 'seoul_bike_sharing_converted.csv', 'seoul_bike_sharing_converted_normalized.csv'

for (dataset_name in dataset_list){
    # Read dataset
    dataset <- read_csv(dataset_name)
    # Standardized its columns:
    # Convert all columns names to uppercase
    names(dataset) <- toupper(names(dataset))
    # Replace any white space separators by underscore, using str_replace_all function
    names(dataset) <- str_replace_all(names(dataset), " ", "_")
    # Save the dataset back
    write.csv(dataset, dataset_name, row.names=FALSE)
}
```
executed in 1.63s, finished 14:34:59 2022-11-12

```
Rows: 8465 Columns: 14
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): DATE, SEASONS, HOLIDAY, FUNCTIONING_DAY
dbl (10): RENTED_BIKE_COUNT, HOUR, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBI...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 8465 Columns: 41
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): DATE, FUNCTIONING_DAY
dbl (39): RENTED_BIKE_COUNT, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, ...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
Rows: 8465 Columns: 41
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): DATE, FUNCTIONING_DAY
dbl (39): RENTED_BIKE_COUNT, TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, ...

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

EDA with SQL (Screenshot)

2.0.1 Establish your Db2 connection

Load the 'RODBC' library, and use the 'odbcConnect( )' function as you did in the previous lab to establish the connection to your Db2 assets. Provided you successfully loaded your data into the tables in that lab, you are now ready to start running SQL queries using the RODBC library as you did in Course 3.

In [2]:
```
library(RODBC);
```
executed in 36ms, finished 01:56:35 2022-11-13

In [3]:
```
# provide your solution here
dsn_driver <- "{IBM DB2 ODBC DRIVER - C_IBMDB2_clidriver}"
dsn_database <- "bludb"
dsn_hostname <- "824dfd4d-99de-440d-9991-629c01b3832d.bs2io90l08kqb1od8lcg.databases.appdomain.cloud"
dsn_port <- "30119"
dsn_protocol <- "TCPIP"
dsn_uid <- "***************"
dsn_pwd <- "***************"
dsn_security <- "ssl"
```
executed in 31ms, finished 01:56:36 2022-11-13

In [4]:
```
conn_path <- paste("DRIVER=",dsn_driver,
                  ";DATABASE=",dsn_database,
                  ";HOSTNAME=",dsn_hostname,
                  ";PORT=",dsn_port,
                  ";PROTOCOL=",dsn_protocol,
                  ";UID=",dsn_uid,
                  ";PWD=",dsn_pwd,
                  ";SECURITY=",dsn_security, sep="")
conn <- odbcDriverConnect(conn_path)
```
executed in 4.40s, finished 01:56:41 2022-11-13

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

[5]:
```
# provide your solution here
query = "SELECT COUNT(*) FROM SEOUL_BIKE_SHARING"
sqlQuery(conn,query)
```
executed in 264ms, finished 01:56:42 2022-11-13

A data.frame: 1 × 1

	1
	<int>
1	8465

2.2 Task 2 - Operational Hours

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

2.2.1 Solution 2

[6]:
```
# provide your solution here
query <-
    "SELECT count(HOUR)
    FROM SEOUL_BIKE_SHARING
    WHERE RENTED_BIKE_COUNT != 0;"

view <- sqlQuery(conn,query)
view
```
executed in 465ms, finished 01:56:44 2022-11-13

A data.frame: 1 × 1

	1
	<int>
1	8465

EDA with SQL (Screenshot)

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.

2.3.1 Solution 3

```
# provide your solution here
# Getting the first record from the table
query <-
    "SELECT *
    FROM CITIES_WEATHER_FORECAST
    LIMIT 1;"

sqlQuery(conn,query)
```

executed in 524ms, finished 01:56:44 2022-11-13

A data.frame: 1 × 12

	CITY	WEATHER	VISIBILITY	TEMP	TEMP_MIN	TEMP_MAX	PRESSURE	HUMIDITY	WIND_SPEED	WIND_DEG	SEASON	FORECAST_DATETIME
	<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<int>	<int>	<dbl>	<int>	<chr>	<dttm>
1	Seoul	Clear	10000	12.32	10.91	12.32	1015	50	2.18	248	Spring	2021-04-16 12:00:00

2.4 Task 4 - Seasons

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

2.4.1 Solution 4

```
# provide your solution here
query <-
    "SELECT DISTINCT(SEASONS)
    FROM SEOUL_BIKE_SHARING;"

sqlQuery(conn,query)
```

executed in 572ms, finished 01:56:45 2022-11-13

A data.frame: 4 × 1

	SEASONS
	<chr>
1	Autumn
2	Spring
3	Summer
4	Winter

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 [9]:

```
# provide your solution here
query <- "SELECT MIN(DATE), MAX(DATE)
          FROM SEOUL_BIKE_SHARING;"

sqlQuery(conn,query)
```

executed in 615ms, finished 01:56:45 2022-11-13

A data.frame: 1 × 2

	1	2
	<chr>	<chr>
1	01/01/2018	31/12/2017

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 [10]:

```
# 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)
```

executed in 461ms, finished 01:56:46 2022-11-13

A data.frame: 1 × 3

	DATE	HOUR	RENTED_BIKE_COUNT
	<chr>	<int>	<int>
1	19/06/2018	18	3556

EDA with SQL (Screenshot)

2.7.1 Solution 7

In [11]:
```
# provide your solution here
query <- "SELECT AVG(TEMPERATURE) AS AVG_TEMP, AVG(RENTED_BIKE_COUNT) AS AVG_RENTED_BIKE, HOUR, SEASONS
            FROM SEOUL_BIKE_SHARING
            GROUP BY HOUR, SEASONS
            ORDER BY AVG(RENTED_BIKE_COUNT) DESC
            LIMIT 10;"

sqlQuery(conn,query)
```
executed in 464ms, finished 01:56:47 2022-11-13

A data.frame: 10 × 4

	AVG_TEMP	AVG_RENTED_BIKE	HOUR	SEASONS
	<dbl>	<int>	<int>	<chr>
1	29.38696	2135	18	Summer
2	16.03086	1983	18	Autumn
3	28.27283	1889	19	Summer
4	27.06630	1801	20	Summer
5	26.27826	1754	21	Summer
6	15.97222	1689	18	Spring
7	25.69891	1567	22	Summer
8	17.27778	1562	17	Autumn
9	30.07500	1526	17	Summer
10	15.06049	1515	19	Autumn

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 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

n [12]:
```
# provide your solution here
query <- "SELECT SEASONS, AVG(RENTED_BIKE_COUNT) AS AVG_RENTED_BIKE, MAX(RENTED_BIKE_COUNT) AS MAX_BIKE,
           MIN(RENTED_BIKE_COUNT) AS MIN_BIKE, STDDEV(RENTED_BIKE_COUNT) AS STD_BIKE
           FROM SEOUL_BIKE_SHARING
           GROUP BY SEASONS
           ORDER BY AVG(RENTED_BIKE_COUNT) DESC;"

sqlQuery(conn,query)
```
executed in 463ms, finished 01:56:50 2022-11-13

A data.frame: 4 × 5

	SEASONS	AVG_RENTED_BIKE	MAX_BIKE	MIN_BIKE	STD_BIKE
	<chr>	<int>	<int>	<int>	<dbl>
1	Summer	1034	3556	9	690.0884
2	Autumn	924	3298	2	617.3885
3	Spring	746	3251	2	618.5247
4	Winter	225	937	3	150.3374

Let's explore a bit and see what might be the most significant contributing factors in terms of the provided data.

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.

EDA with SQL (Screenshot)

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

13]:

```
# 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,
         AVG(SOLAR_RADIATION) AS AVG_SOLAR, AVG(RAINFALL) AS AVG_RAINFALL,
         AVG(SNOWFALL) AS AVG_SNOW,
         AVG(RENTED_BIKE_COUNT) AS AVG_RENTED_BIKE
         FROM SEOUL_BIKE_SHARING
         GROUP BY SEASONS
         ORDER BY AVG(RENTED_BIKE_COUNT) DESC;"


sqlQuery(conn,query)
```

executed in 473ms, finished 01:56:54 2022-11-13

A data.frame: 4 × 10

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

2.10 Task 10 - Total Bike Count and City Info for Seoul

2.10.0.1 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.

2.10.1 Solution 10

14]:

```
9366.182094
```

executed in 21ms, finished 01:56:57 2022-11-13

9366.182094

In [23]:

```
# provide your solution here
# WE ASSUME THAT TOTAL NUMBER IS AVERAGE RENTED BIKE PER DAY
# SO WE SUM TOTAL BIKE RENTED DIVIDED BY SUM OF DATE

query <- "SELECT B.BICYCLES, W.CITY_ASCII, W.COUNTRY, W.LAT,
         W.LNG, W.POPULATION
         FROM WORLD_CITIES W, BIKE_SHARING_SYSTEMS B
         WHERE W.CITY_ASCII = B.CITY AND W.CITY_ASCII = 'Seoul';"


sqlQuery(conn,query)
```

executed in 465ms, finished 02:09:58 2022-11-13

A data.frame: 1 × 6

	BICYCLES	CITY_ASCII	COUNTRY	LAT	LNG	POPULATION
	<int>	<chr>	<chr>	<dbl>	<dbl>	<int>
1	20000	Seoul	Korea, South	37.58	127	21794000

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 [70]:

```
# provide your solution here
query <- "SELECT B.BICYCLES, W.CITY_ASCII, W.COUNTRY, W.LAT,
         W.LNG, W.POPULATION
         FROM WORLD_CITIES W, BIKE_SHARING_SYSTEMS B
         WHERE (W.CITY_ASCII = B.CITY) AND B.BICYCLES BETWEEN 15000 AND 20000;"

sqlQuery(conn,query)
```

executed in 486ms, finished 03:01:22 2022-11-13

A data.frame: 7 × 6

	BICYCLES	CITY_ASCII	COUNTRY	LAT	LNG	POPULATION
	<int>	<chr>	<chr>	<dbl>	<dbl>	<int>
1	19165	Shanghai	China	31.16	121.46	22120000
2	20000	Seoul	Korea, South	37.58	127.00	21794000
3	16000	Beijing	China	39.90	116.39	19433000
4	20000	Weifang	China	36.71	119.10	9373000

EDA with data visualization (Screnshot)

2.2 Descriptive Statistics

Now you are all set to take a look at some high level statistics of the `seoul_bike_sharing` dataset.

2.2.1 Task 4 - Dataset Summary

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

2.2.2 Solution 4

]:
```
# provide your solution here
summary(seoul_bike_sharing)
```
executed in 51ms, finished 11:13:27 2022-11-14

```
      DATE            RENTED_BIKE_COUNT      HOUR       TEMPERATURE
 Min.    :2020-01-01   Min.   :   2.0    7      : 353   Min.   :-17.80
 1st Qu.:2020-03-30   1st Qu.: 214.0    8      : 353   1st Qu.:  3.00
 Median :2020-06-28   Median : 542.0    9      : 353   Median : 13.50
 Mean   :2020-06-29   Mean   : 729.2    10     : 353   Mean   : 12.77
 3rd Qu.:2020-09-26   3rd Qu.:1084.0    11     : 353   3rd Qu.: 22.70
 Max.   :2020-12-31   Max.   :3556.0    12     : 353   Max.   : 39.40
                                        (Other):6347
    HUMIDITY      WIND_SPEED     VISIBILITY   DEW_POINT_TEMPERATURE
 Min.   : 0.00   Min.   :0.000   Min.   :  27   Min.   :-30.600
 1st Qu.:42.00   1st Qu.:0.900   1st Qu.: 935   1st Qu.: -5.100
 Median :57.00   Median :1.500   Median :1690   Median :  4.700
 Mean   :58.15   Mean   :1.726   Mean   :1434   Mean   :  3.945
 3rd Qu.:74.00   3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 15.200
 Max.   :98.00   Max.   :7.400   Max.   :2000   Max.   : 27.200

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

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

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

2.2.5 Solution 5:

]:
```
# provide your solution here
seoul_bike_sharing %>%
    filter(HOLIDAY == "Holiday") %>%
    summarize(HOLIDAY_DAY = n_distinct(DATE))

# there are 17 days for holiday
```
executed in 48ms, finished 13:51:32 2022-11-14

A tibble: 1 × 1

HOLIDAY_DAY
<int>
17

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

2.2.7 Solution 6

]:
```
# provide your solution here
relative<- seoul_bike_sharing %>%
    group_by(HOLIDAY) %>%
        summarize(n = n()) %>%
            mutate(freq = n/sum(n))

relative
```
executed in 77ms, finished 11:32:01 2022-11-14

A tibble: 2 × 3

HOLIDAY	n	freq
<chr>	<int>	<dbl>
Holiday	408	0.04819846
No Holiday	8057	0.95180154

EDA with data visualization (Screnshot)

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

2.2.9 Solution 7

[ ]:
```
# provide your solution here
#number of hours per day * number of day in a year
total_row <- length(unique(seoul_bike_sharing$HOUR)) * length(unique(seoul_bike_sharing$DATE))

total_row

# Number of holiday within the dataset
total_row * relative[1,3]
```
executed in 66ms, finished 11:33:54 2022-11-14

8472

A data.frame: 1 × 1

freq
<dbl>
408.3374

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

2.2.11 Solution 8

[ ]:
```
# provide your solution here
seoul_bike_sharing %>%
    count(FUNCTIONING_DAY)
```
executed in 60ms, finished 11:34:26 2022-11-14

A spec_tbl_df: 1 × 2

FUNCTIONING_DAY	n
<chr>	<int>
Yes	8465

2.3 Drilling Down

Let's calculate some seasonally aggregated measures to help build some more context.

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

2.3.2 Solution 9

[59]:
```
library(dplyr)
```
executed in 32ms, finished 11:35:12 2022-11-14

[96]:
```
# provide your solution here
seoul_bike_sharing %>%
    group_by(SEASONS) %>%
        summarize(rain = sum(RAINFALL), snow = sum(SNOWFALL))
```
executed in 56ms, finished 13:14:21 2022-11-14

A tibble: 4 × 3

SEASONS	rain	snow
<chr>	<dbl>	<dbl>
Autumn	227.9	123.0
Spring	403.8	0.0
Summer	559.7	0.0
Winter	70.9	534.6

Wow, that seems like a lot of snow.
Now that you have some ideas about what sorts of questions can be answered through descriptive statistics, let's start visualizing the data.

EDA with data visualization (Screnshot)

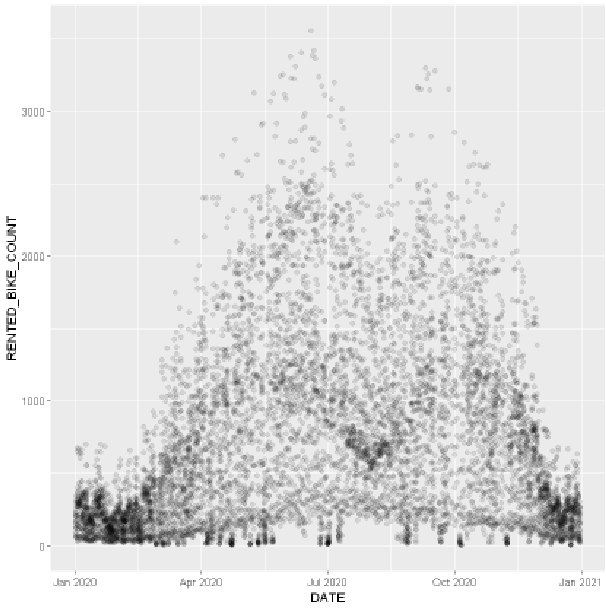
2.4.2 Task 10 - Create a scatter plot of `RENTED_BIKE_COUNT` vs `DATE`.

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

2.4.3 Solution 10

```
5]:  # provide your solution here
     seoul_bike_sharing %>%
         ggplot(aes(y = RENTED_BIKE_COUNT, x = DATE)) +
         geom_point(alpha = 0.1)
```

executed in 1.19s, finished 13:11:08 2022-11-14



2.4.4 Ungraded Task: We can see some patterns emerging here.

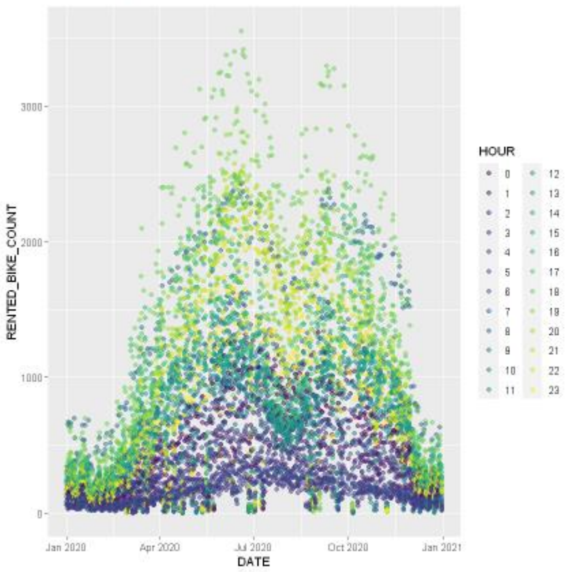
Describe them and keep your findings for your presentation in the final project.

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

```
.04]:  # provide your solution here
       seoul_bike_sharing %>%
           ggplot(aes(y = RENTED_BIKE_COUNT, x = DATE, color = HOUR)) +
           geom_point(alpha = 0.5)
```

executed in 1.90s, finished 13:19:47 2022-11-14



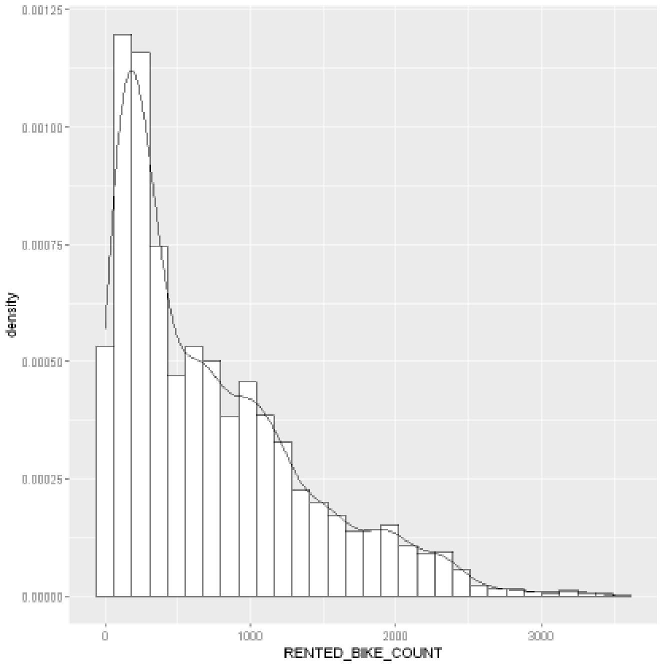
EDA with data visualization (Screnshot)

2.5.2 Solution 12

In [111]:
```
# provide your solution here
seoul_bike_sharing %>%
    ggplot(aes(x = RENTED_BIKE_COUNT)) +
    geom_histogram(aes(y=..density..), col = 1, fill = "white") +
    geom_density(stat = "density")
```

executed in 338ms, finished 13:30:28 2022-11-14

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



2.6 Correlation between two variables (scatter plot)

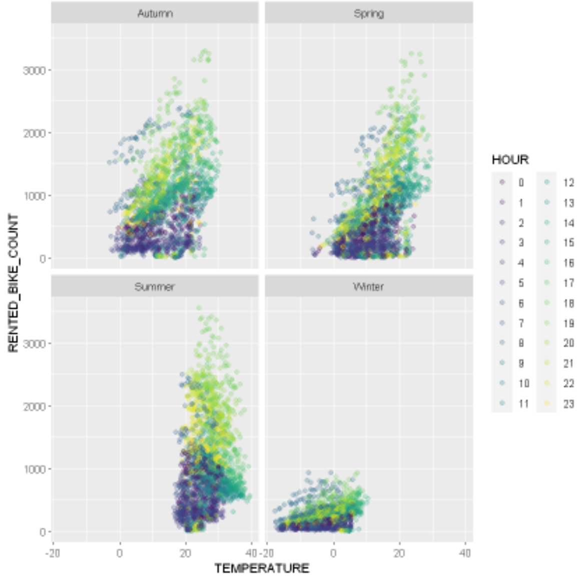
2.6.1 Task 13 - Use a scatter plot to visualize the correlation between `RENTED_BIKI` `SEASONS`.

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

2.6.2 Solution 13

[118]:
```
# provide your solution here
seoul_bike_sharing %>%
    ggplot(aes(y = RENTED_BIKE_COUNT, x = TEMPERATURE, color = HOUR)) +
    geom_point(alpha = 0.25) +
    facet_wrap("SEASONS")
```

executed in 1.95s, finished 13:33:59 2022-11-14



EDA with data visualization (Screnshot)

2.6.3 Ungraded Task: Describe the patterns you see.

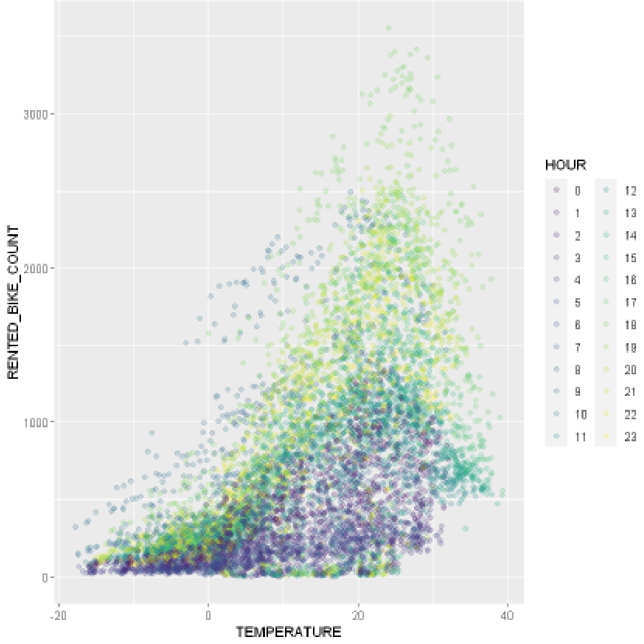
What do these patterns imply about the relationships between these variables? Keep your findings fo

Click here for a solution

Comparing this plot to the same plot below, but without grouping by `SEASONS` , shows how important

```
ggplot(seoul_bike_sharing) +
    geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=1/5)
```

executed in 1.69s, finished 13:35:16 2022-11-14



2.7 Outliers (boxplot)

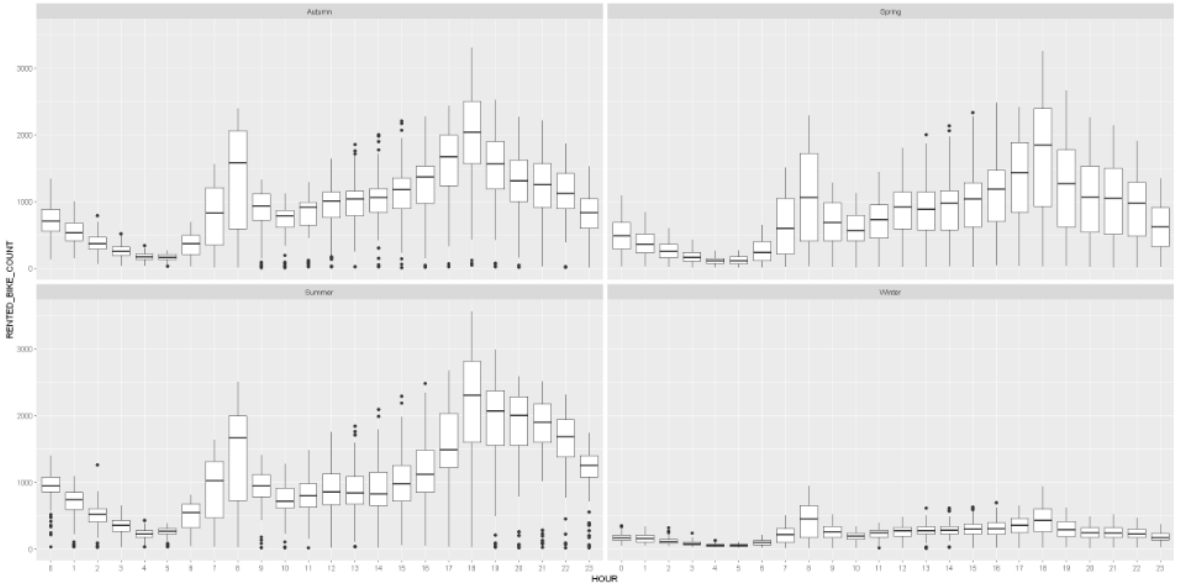
2.7.1 Task 14 - Create a display of four boxplots of `RENTED_BIKE_COUNT` vs. `HOUR` grouped by `SEASONS`.

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

2.7.2 Solution 14

```
# provide your solution here
options(repr.plot.width=20, repr.plot.height=10)
seoul_bike_sharing %>%
    ggplot(aes(y = RENTED_BIKE_COUNT, x = HOUR)) +
        geom_boxplot() +  # boxplot geom
        facet_wrap("SEASONS")      # facet by prediction
```

executed in 1.43s, finished 13:41:27 2022-11-14

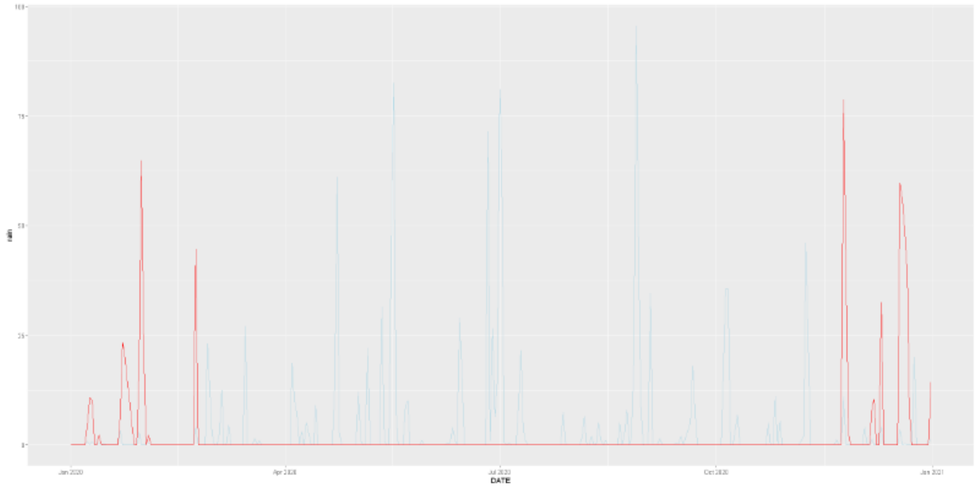


EDA with data visualization (Screnshot)

In [129]:

```r
# provide your solution here
seoul_bike_sharing %>%
    group_by(DATE) %>%
        summarize(rain = sum(RAINFALL), snow = sum(SNOWFALL)) %>%
        ggplot(aes(x = DATE)) +
          geom_line(aes(y = rain), color = "light blue") +
          geom_line(aes(y = snow), color = "red")
```

executed in 402ms, finished 13:45:59 2022-11-14



2.7.6 Task 16 - Determine how many days had snowfall.

2.7.7 Solution 16

In [132]:

```r
# provide your solution here
seoul_bike_sharing %>%
    filter(SNOWFALL != 0) %>%
    summarize(day_snow = n_distinct(DATE))
```

executed in 55ms, finished 13:48:47 2022-11-14

A tibble: 1 × 1

day_snow
<int>
27