Applied Data Science with R Capstone project

<Syed Bokhari>

<09-06-2022>

Table Of Contents

Outline

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

Executive Summary

Biking is a cheap alternate transportation mode that is being increasingly adopted by people in urban settings to avoid traffic congestion. It is also a good way to keep fit. Several cities such as Seoul have adopted biking systems that allow people to rent bikes for their daily use. The purpose of this study is to investigate the effect of weather and related factors on Bike Rentals (sometimes referred to as Rental Bike Count).

Understanding the impact of weather and related variables on biking behavior will allow planners to predict biking demand and make decisions regarding the maintenance, upgradation and expansion of bike fleets. It might also offer insights on how to encourage biking behavior among people.

This project uses publicly available data on Seoul's bike usage as well as data on Biking Systems and World cities. It uses OpenWeatherAPI for forecasting weather and creating related sets.

The project makes use of a baseline Regression Model to evaluate the impact of weather on Bike Rentals. It then improves the baseline model

Executive Summary

with additional features, polynomial terms and Generalized Linear Models with varying levels of regularization to improve the model.

It subsequently uses the refined model to make predictions about future bike demand in 5 cities around the world. The project uses R-Shiny to model this demand on a Leaflet Map. In addition, selecting each of these cities displays a map of the city along with:

- Forecasted Temperature Plot
- Forecasted Biking Demand Plot
- Regression plot of Humidity's Impact on Predicted Biking Demand

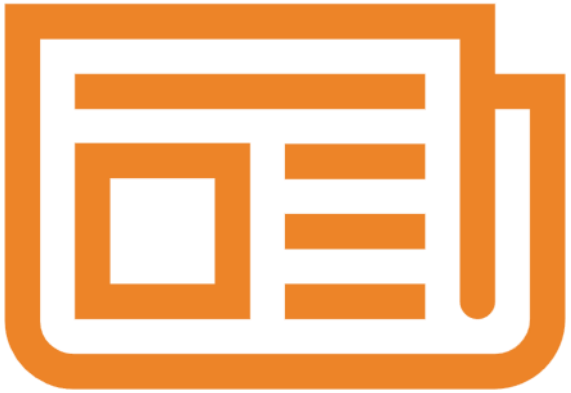
The analysis clearly showed that there was a significant impact on Bike Rentals due to Temperature which varied with the Seasons. Other weather related variables such as Rainfall etc., also impacted Bike Rentals. Hour of the day was another significant variable that influenced Bike Rentals. Consequently, much of the variation in biking demand could thus be explained by weather related changes.

Introduction

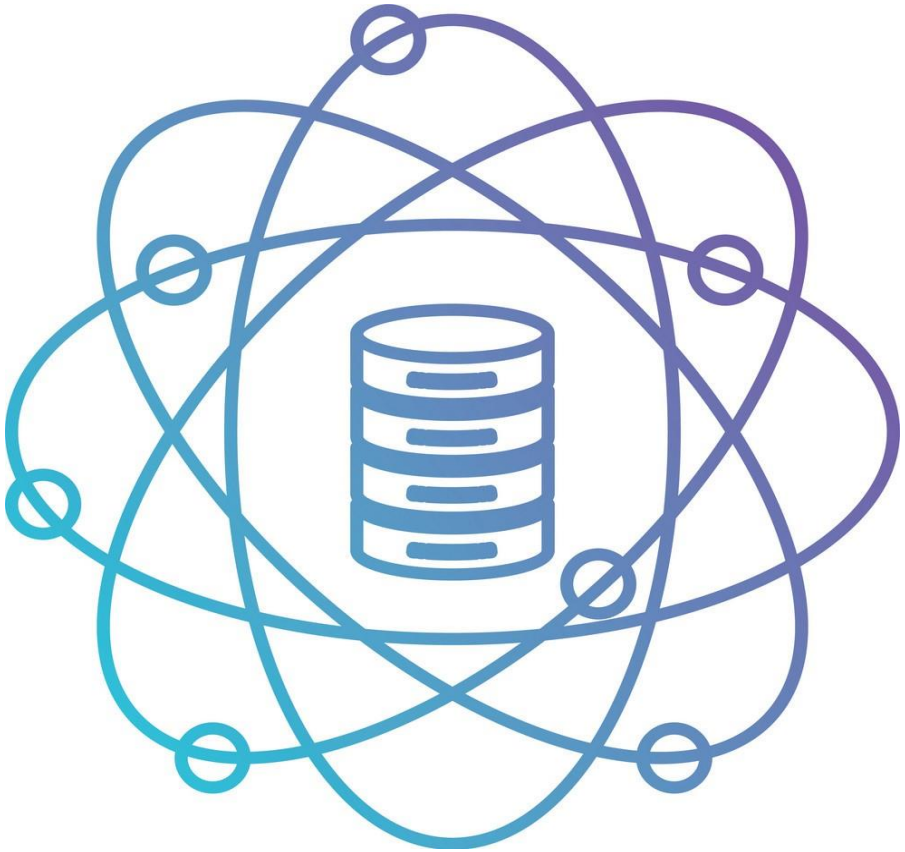
Congestion and fitness concerns have prompted the adoption of biking as an alternate means of transportation in many urban areas. Many major cities have adopted bike sharing systems to assist bikers in this endeavor. However, demand for biking tends to vary. Some of this variation may be due to weather related factors, as in extreme conditions people may prefer alternative transportation means to biking.

To understand how weather impacts biking behavior, this project makes use of data from the city of Seoul from 2017 to 2018. The system in use there involves Bike Rentals. Data for these Rentals combined with the data for prevailing weather conditions and other factors is used to determine how these variables may impact the demand for Bike Rentals. The predictive analytics is done by making use of Linear Regression models. Over the course of the project, baseline results are improved with the addition of more complex features. Forecasts about weather then are used to predict Biking demand for the next few days in several major cities that share characteristics with 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 an R Shiny dashboard app



Methodology

Details the manner in which various activities such as Data Collection, Data Wrangling, Exploratory Data Analysis (EDA) etc. were performed.

Data Collection

- For the Datasets, two data sets had to be acquired from the web and then converted into CSV format through processing while two other datasets were acquired from IBM Cloud Storage Service in CSV format.
- The Bike Sharing Systems dataset, referred to as Bike Systems dataset, was extracted from Wikipedia using Webscraping.
- The original link for the webpage is: https://en.wikipedia.org/wiki/List_of_bicycle-sharing_systems
- For webscraping, the rvest library was used to read html from the wiki page directly.
- Afterwards, the html data was parsed for html table tags. Each such table was printed, and the Bike Sharing Systems table was identified and saved.

Data Collection

- The bike sharing table was then read into a CSV file creating the Bike Systems dataset.
- The second dataset, City Weather Forecast was obtained from OpenWeatherAPI, an open source API from where weather forecasts for global cities for the next 5 days at 3 hour intervals can be obtained.
- To get this information, the httr library was used alongside an API key from OpenWeatherAPI to obtain a JSON object as a response to an HTTP GET request which used the API key as a query parameter.
- The JSON object was examined and the relevant fields pertaining to weather forecasting for the city of interest was recorded from the JSON object.
- These fields were then combined into a dataframe containing the weather forecasting information for the city of interest.

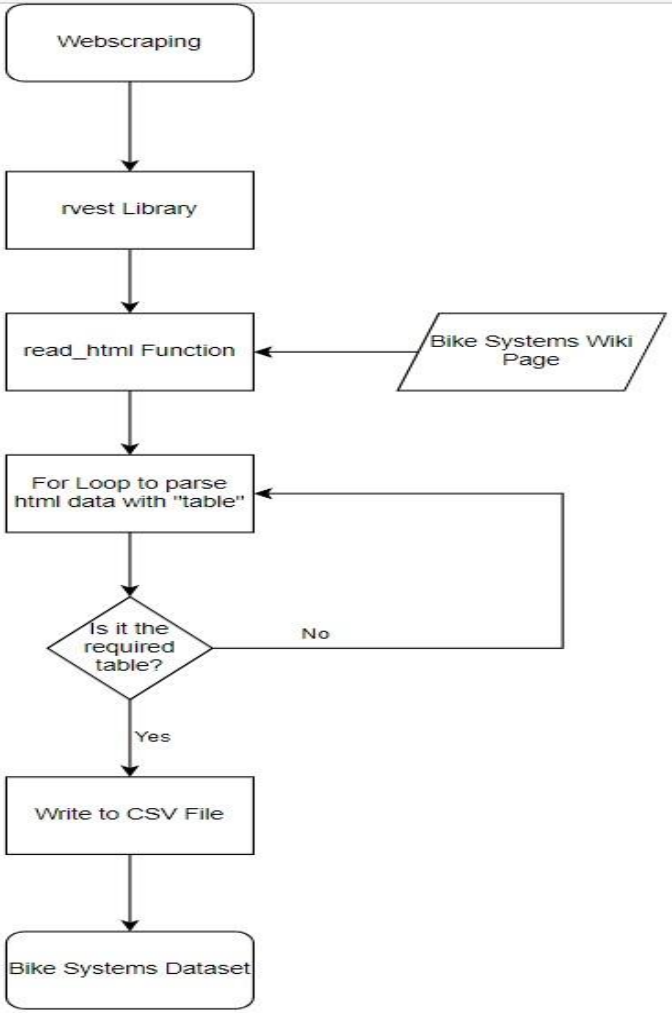
Data Collection

- All the steps done previously were then combined into a single function where the input was a list of city names and the output was a single dataframe containing the city names and the weather forecasts for the cities for different weather variables.
- The function used the API key to GET weather forecast information for one city at a time in JSON format; separated the relevant information from the JSON object into relevant fields and stored them.
- It looped through the list of cities and repeated the above process for each city.
- Afterwards, the data for all the cities was combined into a single dataframe.
- This dataframe was then written as a CSV file to create City Weather Forecast dataset.

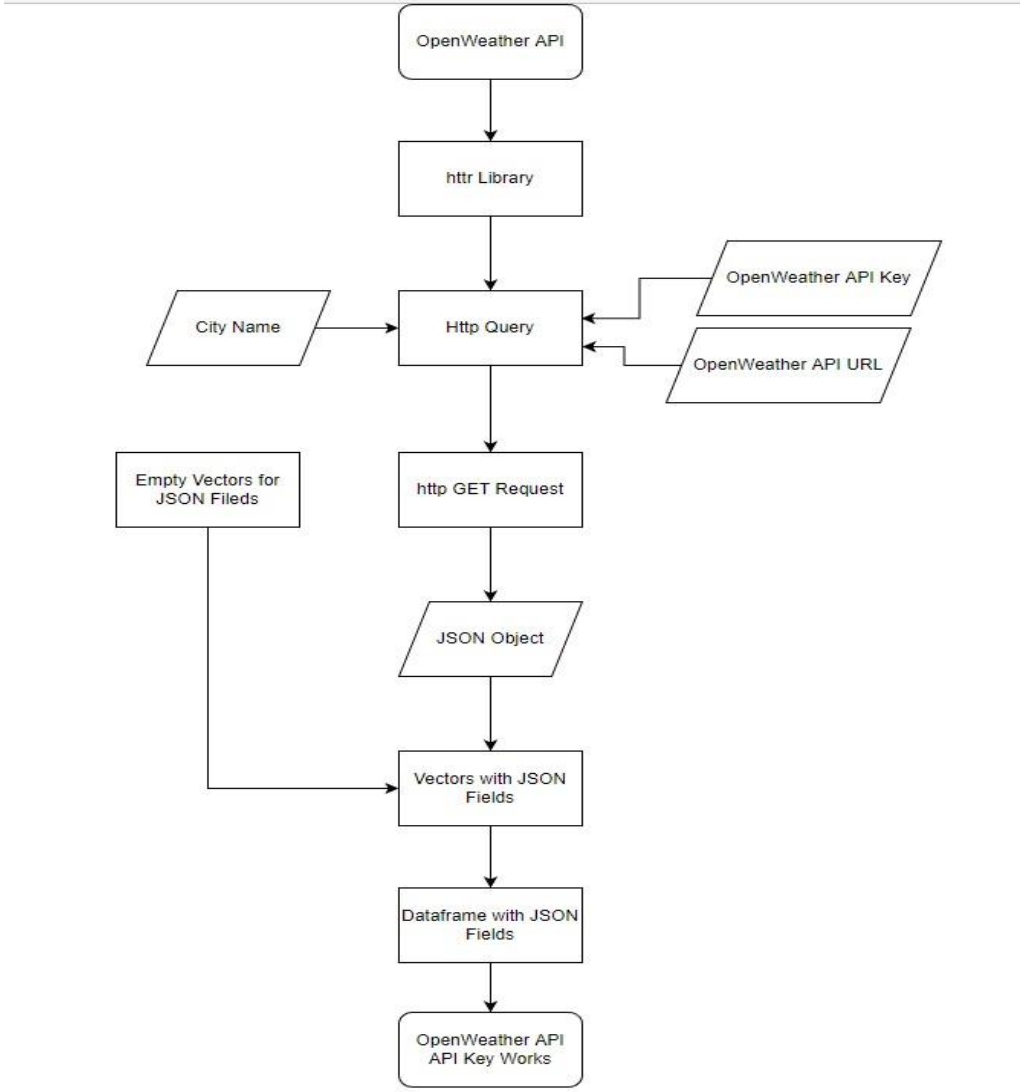
Data Collection

- The other two datasets, World Cities and Seoul Bike Sharing, referred to as Bike Sharing dataset, were present in CSV format already.
- Both sets were downloaded from IBM Cloud Storage Service as CSV files.
- All 4 collected datasets were in raw form upon collection and needed to be processed further.

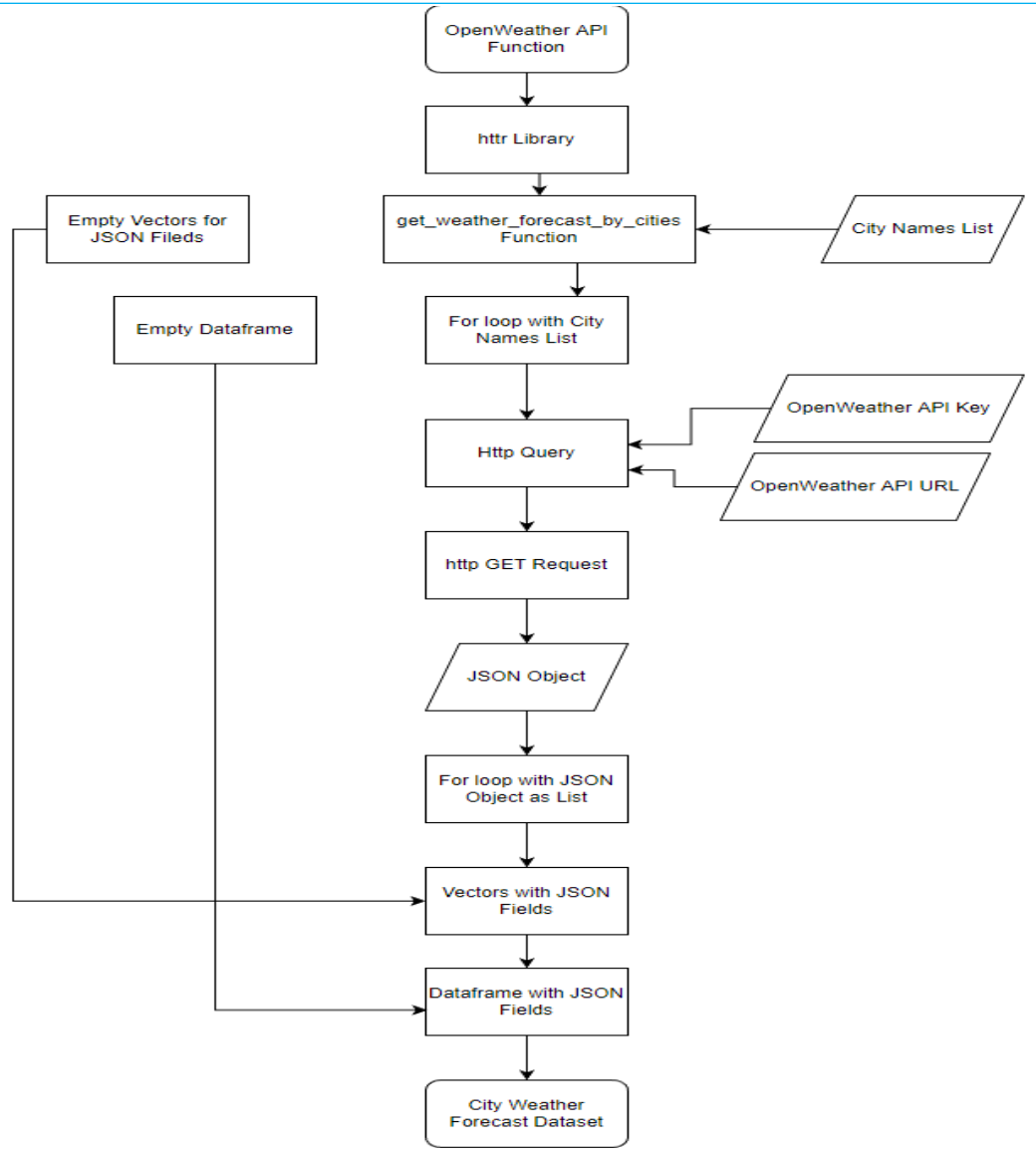
Data Collection – Webscraping



Data Collection- OpenWeather API



Data Collection- OpenWeather API



Data Wrangling

- The datasets acquired in the data collection phase were raw and needed to be processed to ensure no irregularities in the data caused problems during subsequent phases.
- Firstly, the names of all the columns in every dataset were standardized. This was done by reading in the datasets in CSV formats into dataframes and then changing the names using the names() function.
- The Bike Systems dataset was webscraped and so had lots of features that were not needed for the project.
- The first problem had to do with column types. These were auto-chosen to be of character type. This was true in many cases but in some cases certain numeric columns were also cast as character type.

Data Wrangling

- A search of the columns that were supposed to be numeric with the grepl function and regular expressions indicated that they had some characters in some rows which led to them being cast as character type. This is common for webscraped datasets where input validation standards vary a lot.
- The data in the other columns was examined, the pattern of characters appearing in the data was identified as reference links and subsequently, some regular expressions were used to identify the frequency of these characters in the columns.
- Subsequently, a function was designed using regular expressions to replace the reference links in these columns using stringr functions. The columns were thus cleaned of reference links.
- For the numeric columns cast as character data, again Regular Expressions were used to extract only the numeric data which was cast from character to numeric.

Data Wrangling

- The resulting columns were then ‘mutated’ using the dplyr package in the original dataframe as numeric type columns.
- The resulting dataframe was then written as a CSV file as cleaned Bike Systems dataset.
- The Bike Sharing dataset was checked. It had some columns with NA values. These needed to be dealt with.
- NA values were in the Rented Bike Count and Temperature columns.
- As Rented Bike Count was seen as the response variable, NA values in it did not help in later phases. Therefore, all rows with missing values in Rented Bike Count were dropped. There were 295 such rows which represented only 3% of the total rows.

Data Wrangling

- There were NA values in Temperature column also. But Temperature is a predictor variable. Therefore, its missing values were imputed so that other data in the row was not lost through dropping.
- As only 9 values of Temperature were missing and all of them were for the Summer season, the mean value of Temperature for Summer was imputed for the missing values.
- Next, there were some columns in the dataset that had categorical variables which only took a range of values. As predictive analytics with such columns was difficult, it was decided to split them into indicator variables.
- Indicator or dummy variables record a 1 for a specific value of the categorical variable and 0 for all others. This will be repeated for every specific value of the categorical variable.

Data Wrangling

- There were four such columns in the dataset: HOUR, SEASONS, HOLIDAY. FUNCTIONING DAY. Of these, HOUR was read as a numerical column. It was converted into a factor or categorical variable.
- Subsequently, the function dummy_cols from fastDummies library was used to convert HOURS, SEASONS and HOLIDAY columns into Dummy or Indicator variables. As FUNCTIONING DAY had a single value, it was left alone.
- Next, the numerical columns had values which covered very different ranges. This could create problems as large values in a column could reduce model accuracy by biasing the model.
- One solution to the problem of large values is normalization which converts all columns into values with a single range.

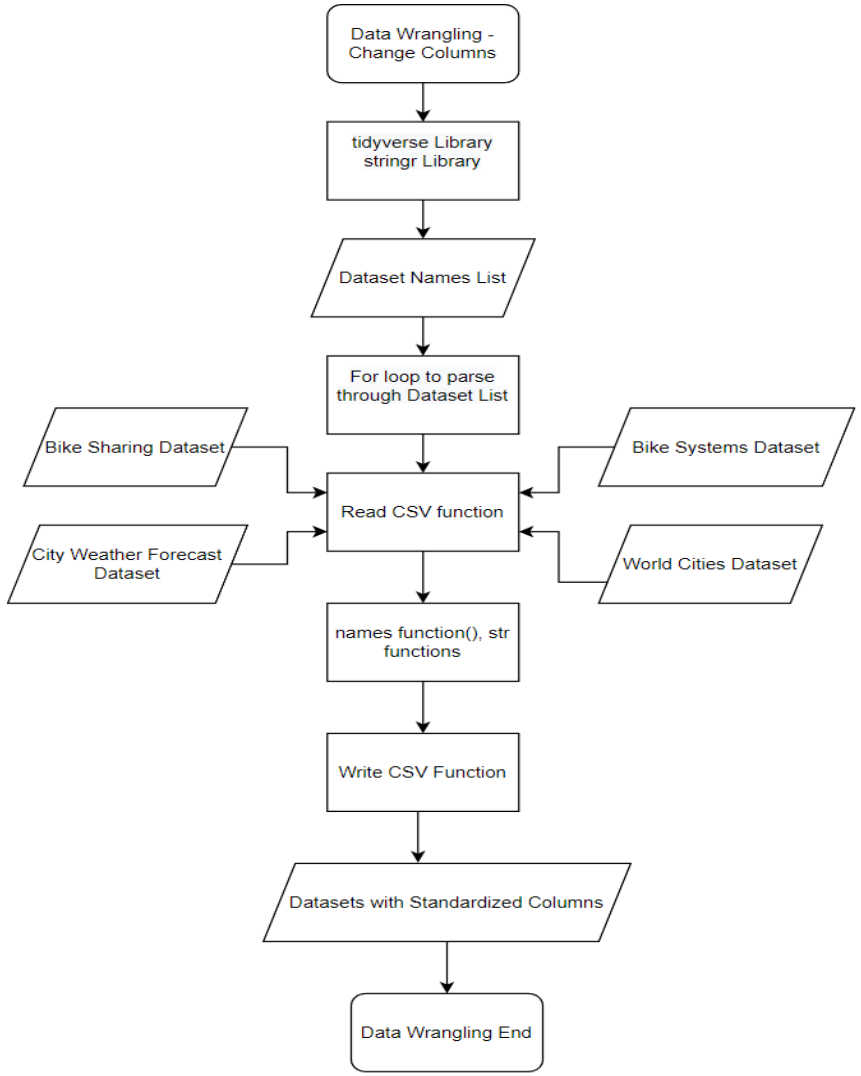
Data Wrangling

- There are different types of normalization. Min-max scaling is one of these methods which was attempted with the columns containing numerical values.
- The numerical columns are: RENTED BIKE COUNT, TEMPERATURE, HUMIDITY, VISIBILITY, WIND SPEED, DEW POINT TEMPERATURE,SOLAR RADIATION, RAINFALL and SNOWFALL.
- All these columns could be transformed using several methods. Three separate methods were used to see the relative efficacy of each method. All three yielded the same result.
- The mutate function in dplyr was used to convert the columns using the formula for min-max scaling in the original dataframe.
- This method required lengthy coding and is thus not recommended.

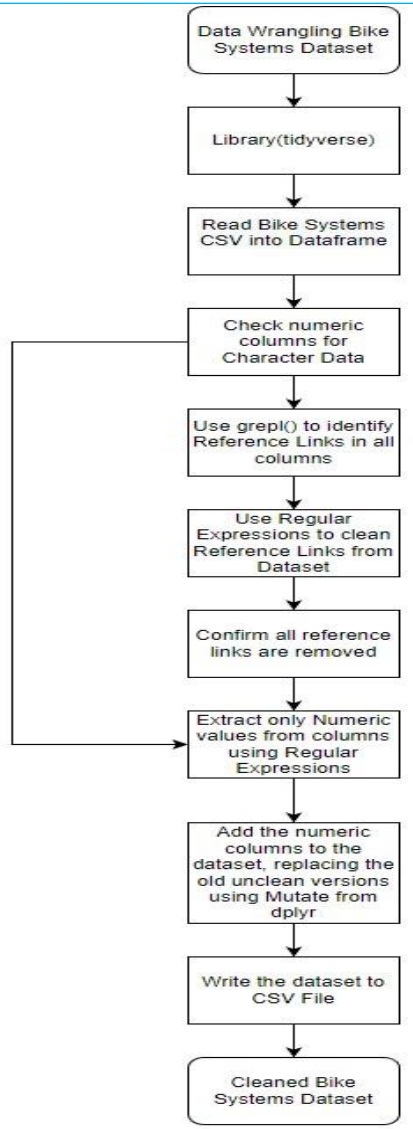
Data Wrangling

- A list of the columns was created and a for loop was run where each of these columns was transformed using the min max formula.
- This was a relatively less code heavy method and may be used.
- The caret package was used to install the caret library. It provides a preprocess function where method range corresponds to min max scaling. The predict function generates the min maxed values from the results of preprocess.
- This requires some knowledge of caret's functions but can be used.
- The normalized dataset had many new columns. These column names were standardized again. The resulting dataset was saved as a CSV file.

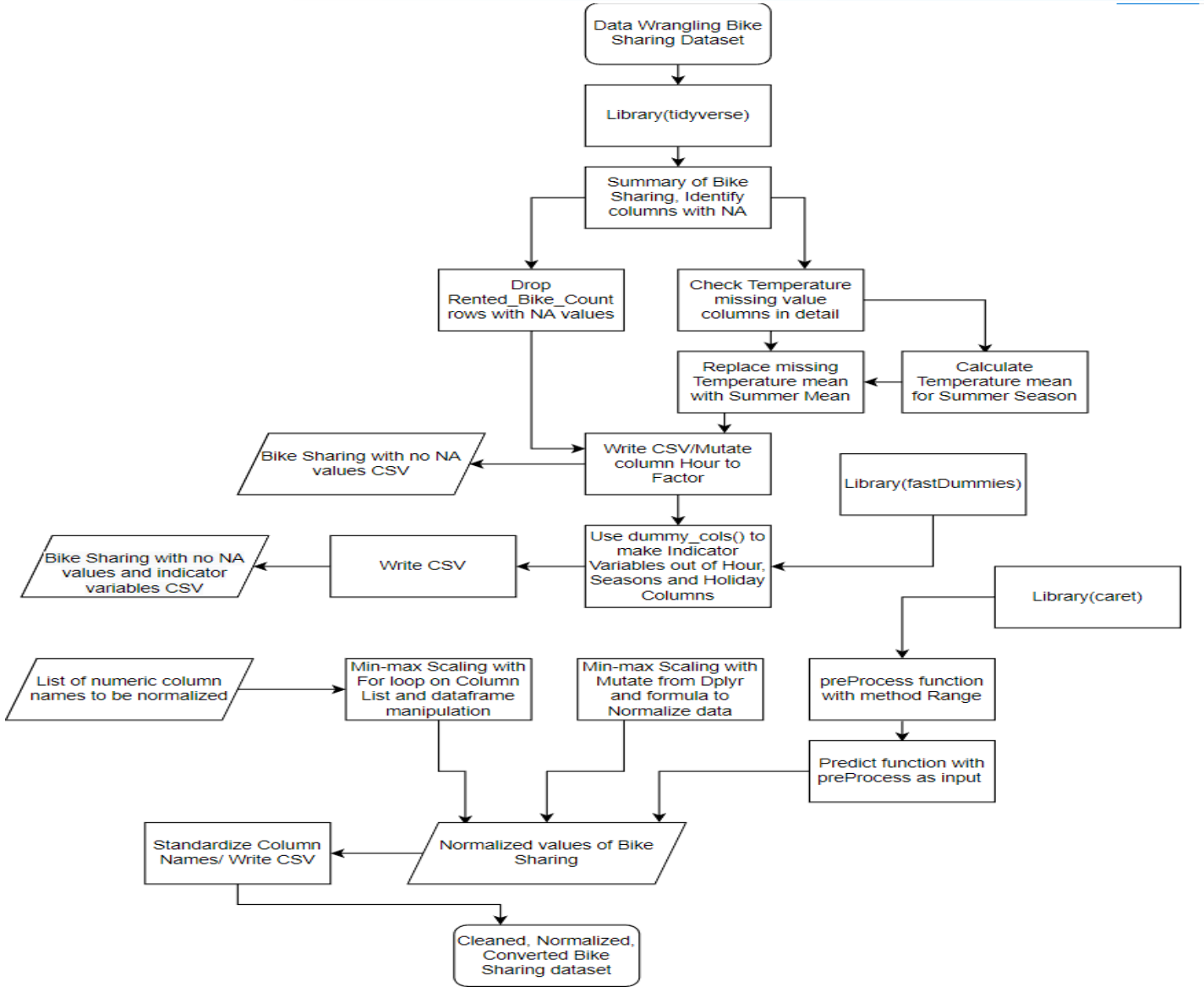
Data Wrangling – Change Columns



Data Wrangling–Clean Bike Systems



Data Wrangling–Clean Bike Sharing



EDA with SQL

- The 1st query determined the total number of rows from the Bike Sharing dataset.
- The 2nd determined how many entries in that dataset had non-zero values for Hour of the day.
- The 3rd query determined what the weather of Seoul was forecasted to be in the next 3 hours from the City Weather Forecast dataset.
- The 4th query determined the Distinct values of Seasons in the Seasons column of the Bike Sharing dataset.
- The 5th query involved determining the First Date and the Last Date in the Bike Sharing dataset.
- The 6th query determined which Hour of the Day and Date had most Bike Rentals by making use of a Subquery, both using the Bike Sharing dataset.

EDA with SQL

- The 7th Query determined the average Temperature and average number of Bike Rentals over each Season. It also listed the top 10 results by Bike Count. Both of these queries used the Bike Sharing dataset.
- The 8th query determined several statistical measures such as Minimum, Maximum and Standard Deviation alongside Average Bike Count for each Season from the Bike Sharing dataset.
- The 9th Query determined averages of various Weather related variables (Temperature, Visibility, Humidity, Wind Speed, Dew-Point Temperature, Solar Radiation, Snowfall, Rainfall) alongside Average Bike Count for each Season from the Bike Sharing dataset. The results were ranked by Average Bike Count in Descending order.

EDA with SQL

- The 10th query made use of the implicit join technique to join the Bike Sharing dataset with the World Cities dataset in order to determine Geographical Information of Seoul (Country, Latitude, Longitude, Population) from the World Cities dataset and the Number of Bikes in Seoul from the Bike Sharing dataset.
- The 11th query also made use of the implicit join technique to join Bike Sharing dataset with World Cities dataset to determine the Geographical Information (City, Country, Latitude, Longitude, Population) and Bicycle Count of all those cities which had between 15,000 and 20,000 bicycles.

EDA with data visualization

- Several charts were plotted to help in Exploratory Data Analysis(EDA).
- The 1st chart was plotted between Date and Bike Rentals. This clearly showed variations in Bike Rentals on different dates.
- The 2nd chart added Hours to the first plot by varying Fill color of points. This revealed that there was considerable time variation alongside date variation of Bike Rentals.
- The 3rd chart investigates Temperature by plotting it alongside Bike Rentals. Here too a relationship was observed. Rentals fell at extremes of Temperature and had a peak too.
- The 4th chart split Temperature according to Seasons. This revealed clear variations in Temperature by Season.

EDA with data visualization

- The 5th chart plots the distribution of Rentals with hour of the day using a Box Plot divided by Seasons. This revealed a surprising regularity to Rentals throughout a day in nearly every season.
- The 6th plot is a Histogram of Bike Rentals with a Kernel Density Curve. It reveals some important points about the distribution of Bike Rentals.

Predictive Analysis

- With the data cleaned and wrangled, through Exploratory data analysis some basic relationships were observed.
- These indicated linear relationships between the Response Variable, Bike Rentals and some predictor variables, such as Temperature.
- Due to these linear relationships, a Linear Regression (LM) model was used as a baseline model to model the data with Bike Rentals as the Response Variable and Weather related Variables as predictor variables.
- The data was split into a training and testing set. The training set was used to train the LM baseline model and then the testing set was used for its evaluation.
- The LM baseline model made predictions with testing set's data which was compared to the actual values in the testing set.

Predictive Analysis

- The results were evaluated using measures such as R-Squared (RSQ) and Root Mean Square Error (RMSE).
- A ranking of the effects of each predictor variable in the baseline model was visualized for better manipulation.
- The baseline model seemed promising and additional features were added to the baseline LM model to improve its performance further.
- First, polynomial terms were added to the baseline LM model to capture non-linear behavior of some of the predictor variables. The ranking was used here.
- Again the model was trained with the training set and the results evaluated using the testing set. The measures RSQ and RMSE were re-calculated.
- The results showed some improvement which was a good sign.

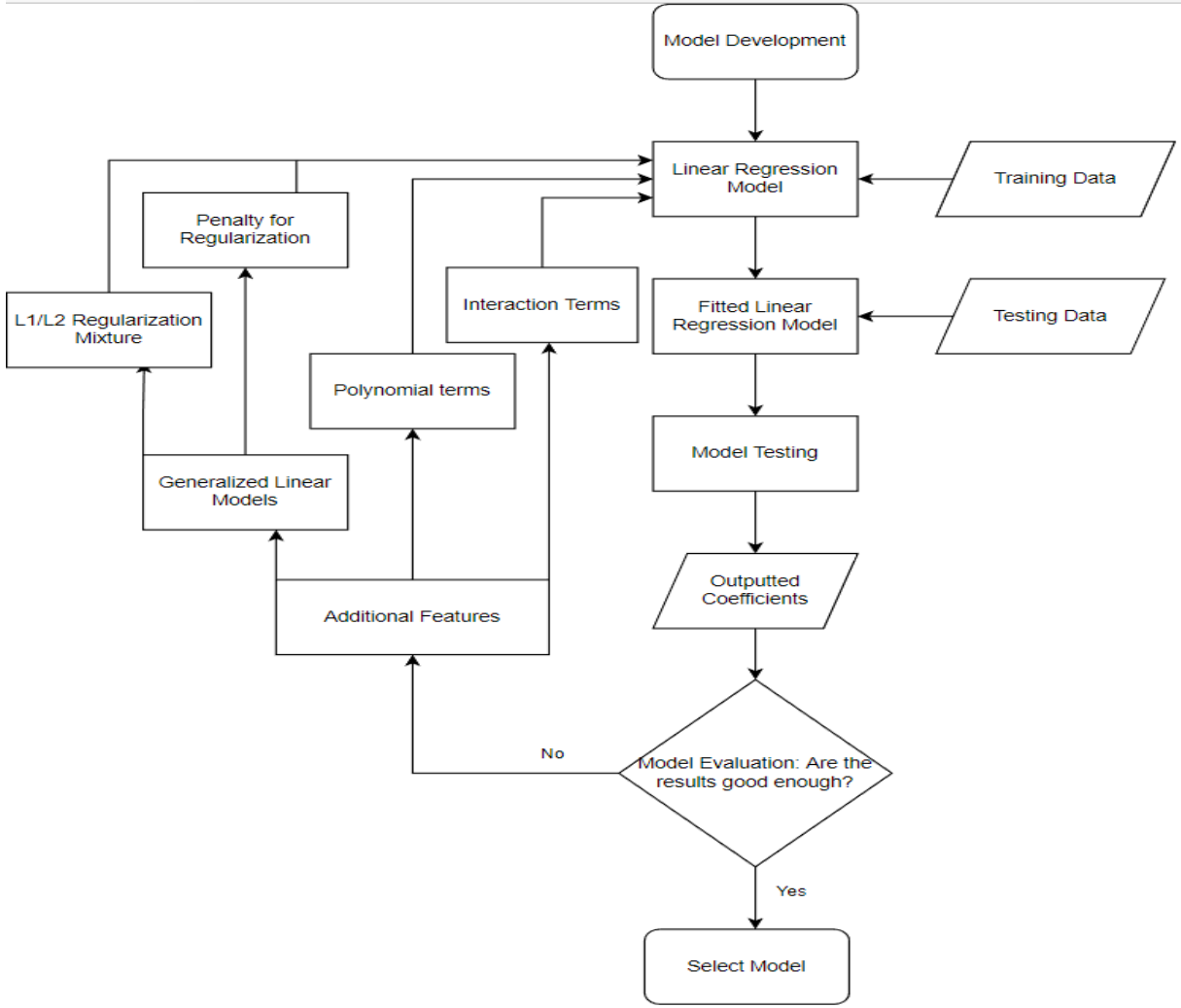
Predictive Analysis

- Next, interaction terms were added to capture the effect of those variables which interacted with each other. Again, the ranking provided a guide as to which variables were important in this regard.
- The resulting LM model was again trained and evaluated with the training and testing sets respectively.
- The results showed some improvement in the measures RSQ and RMSE.
- Next, Generalized Linear Models (GLM) were introduced. These models evaluated the predictor variables somewhat differently from the normal Linear Regression to produce more accurate results.
- Normal LM models tend to overfit the training data, thereby producing excellent results that don’t repeat with testing data. GLM model’s minimize overfitting and reduce model accuracy degradation.

Predictive Analysis

- The resulting GLM model was trained and evaluated with the training and testing sets respectively.
- The results showed some improvement in RSQ and RMSE.
- Next, regularization was introduced and different mixtures of L1 and L2 regularization were added to the GLM model. This was to further prevent overfitting of the model to the training data. More complexity can cause overfitting. Different mixtures of L1 and L2 introduce different amounts of penalty in the model.
- The regularized GLM models with different values of Penalty and Mixture of L1 and L2 were trained and evaluated with the training and testing sets respectively.
- The model with the best RSQ and RMSE was identified and chosen.
- GLMs were used with Linear Regression to evaluate the predictor variables.

Predictive Analysis- Flowchart



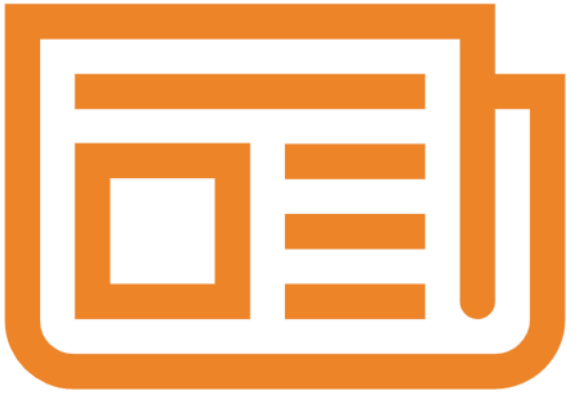
R Shiny dashboard

- The R Shiny Dashboard includes a global map where circular markers indicate the level of Predicted Bike Demand for 5 cities. These markers may be clicked to obtain a Popup with weather conditions for the cities.
- The dashboard has a Dropdown list where any of the 5 cities can be selected in addition to the All option which is the option for the Global Map.
- Selecting a city changes the view to the specific city's map. There is a marker in the city center that can be clicked to display a Popup with detailed information about the city's weather conditions.
- In addition to the City map, there is also a Temperature plot for the city below the drop-down list which displays the forecasted temperature for the city. The data for forecasting is taken from OpenWeather API.

R Shiny dashboard

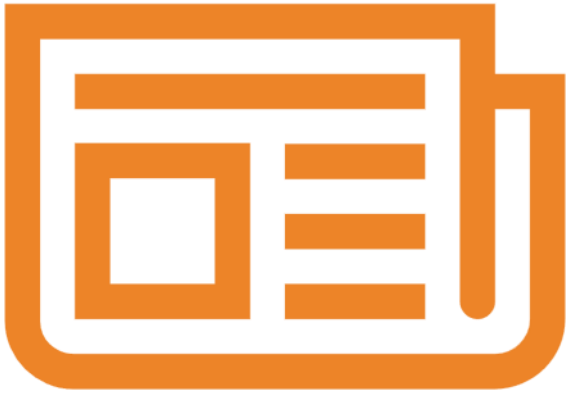
- Below the Temperature plot is a Predicted Bike Demand plot which uses a regression based model of weather and other data to estimate “predicted” bike demand for the next 5 days.
- There is a Text Box underneath the Predicted Bike Demand plot. When a user clicks on the plot, the predicted demand and date and time for that point gets displayed in the text box.
- Underneath the text box, there is a Regression Plot which plots the effect of Humidity on the Predicted Bike Demand by making use of a simple Linear Regression model.
- For details of the dashboard, check the Dashboard section in the presentation.

Results



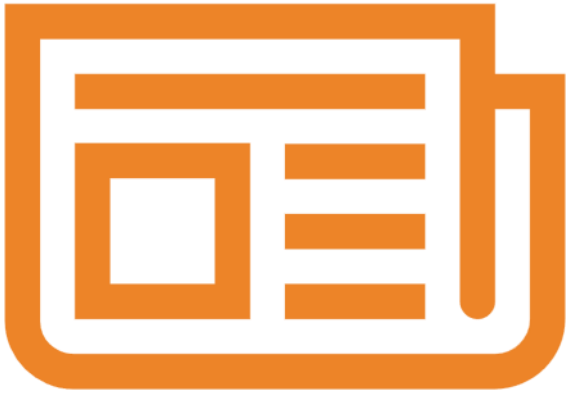
- Exploratory data analysis indicated that there were some clear relationships between the response variable, Rented Bike Count and some of the other variables.
- Temperature was clearly an important variable in explaining the variation between values of Rented Bike Count. Extremes of Temperature reduced Rented Bike Count with the optimal value of Temperature being 20 and 30 °C for Rented Bike Count.
- Temperature variations were dependent on Seasons and as a result there was significant difference in Rented Bike Count for each Season. Winter performed worst in this regard while Summer was the best performer.
- Time of day also impacted Bike Rentals. Peak Bike Rentals occurred at 6 pm when the workday ended and the second peak occurred at 8 am when the workday started.

Results



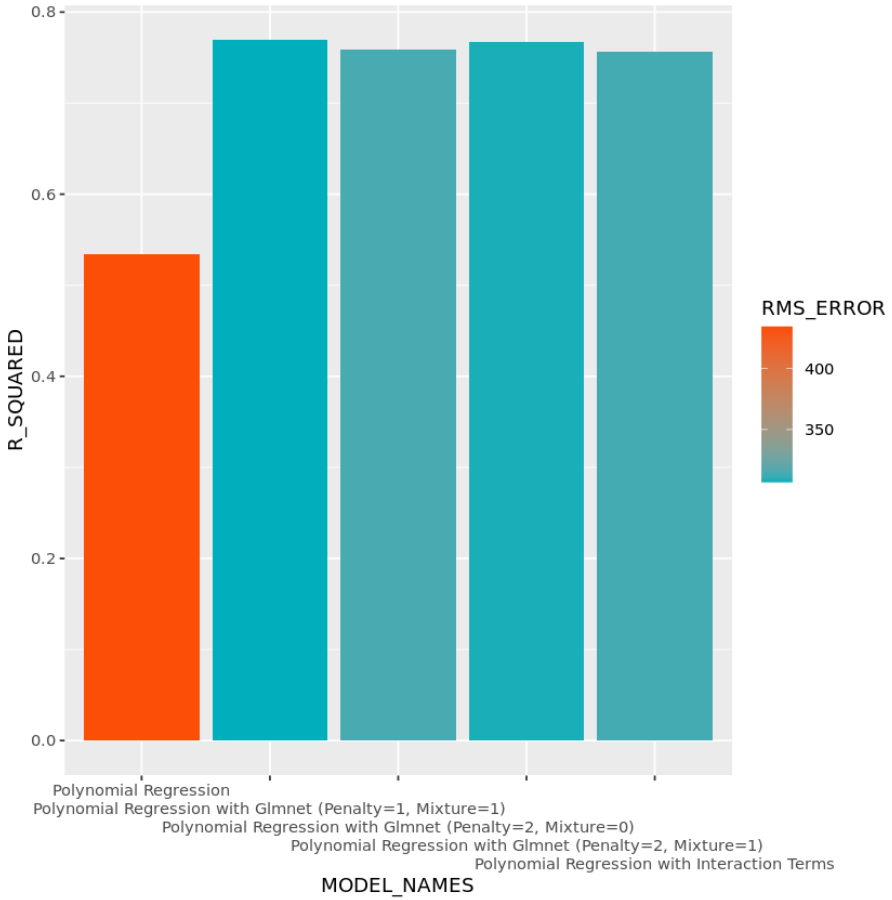
- Predictive Analysis was conducted with the Bike Sharing dataset. Initially, a baseline Linear Regression model was used with weather variables as predictor variables and Rented Bike Count as the response variable.
- The data from Bike Sharing dataset was split into a training set and a testing set.
- The baseline Linear Regression Model was trained with the training set and then it was used with the testing set to predict the values of the Response Variable. The resulting values had an R-Squared value of 0.42 and Root Mean Square Error of 491.5.
- The baseline Linear Regression Model was improved by adding all variables from the Bike Sharing dataset as features. Again, the improved model was trained with the training set and then evaluated with the testing set. The resulting values had an R-Squared value of 0.65 and Root Mean Square Error of 382.3.

Results



- Subsequently, all the coefficients of the model were ranked according to the magnitude of their value. This would aid subsequent efforts to improve the model.
- Polynomial terms were now added to the Linear Regression model to capture some behavior of non-linear variables. Interaction terms were added also. The new model was trained with the training set and evaluated with the testing set. The resulting values had an R-Squared value of 0.757 and a Root Mean Square value of 313.84.
- Finally, Generalized Linear Models were added instead of simple Linear Regression Model. These models also incorporated Regularization that allows for improved results. Several combinations of L1 and L2 regularizations were tested and the best model had an R-Squared of 0.768 and Root Mean Square Error of 306.93.

Results



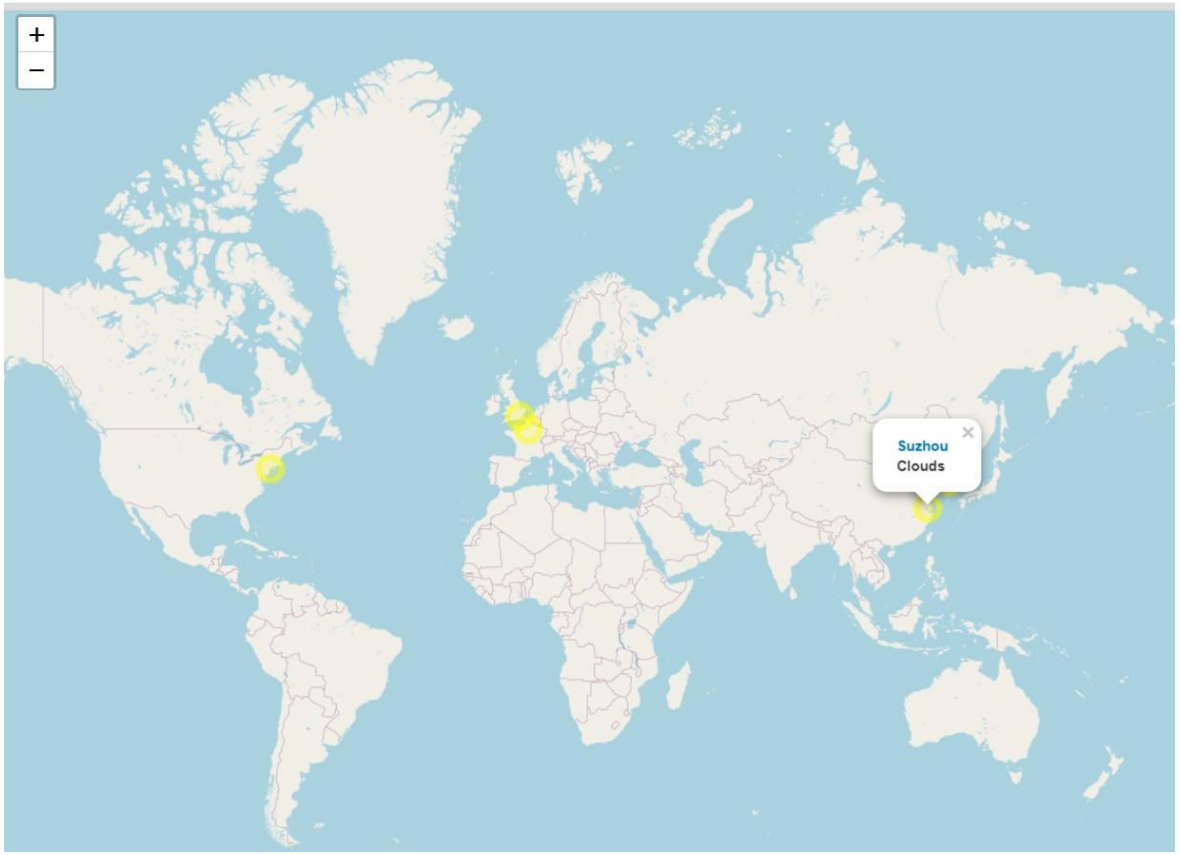
A comparison of the R-Squared and RMS values of the various models created with the data is presented in the accompanying bar chart. The second bar represents the model with the best coefficients.

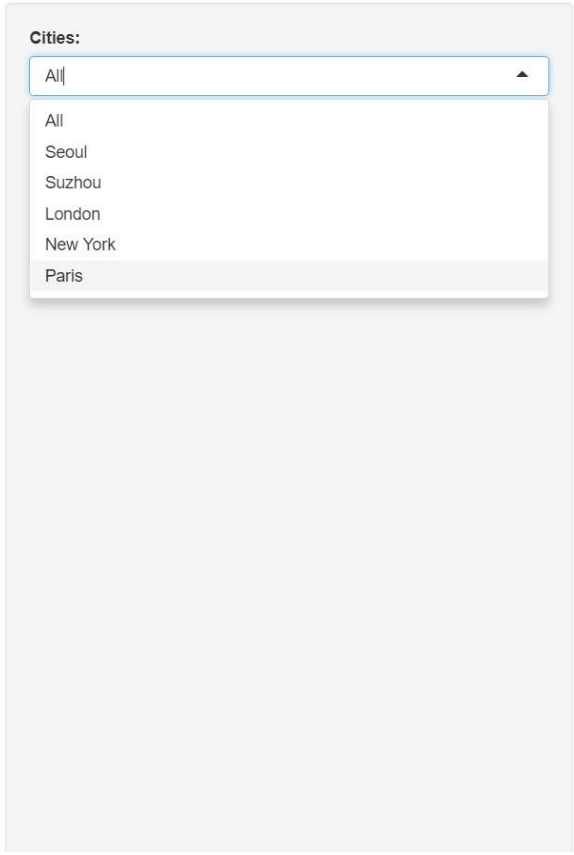
Results

An R-Shiny Dashboard was created with multiple maps and features for those maps. Some demo shots of the R-Shiny Dashboard are provided.

Results

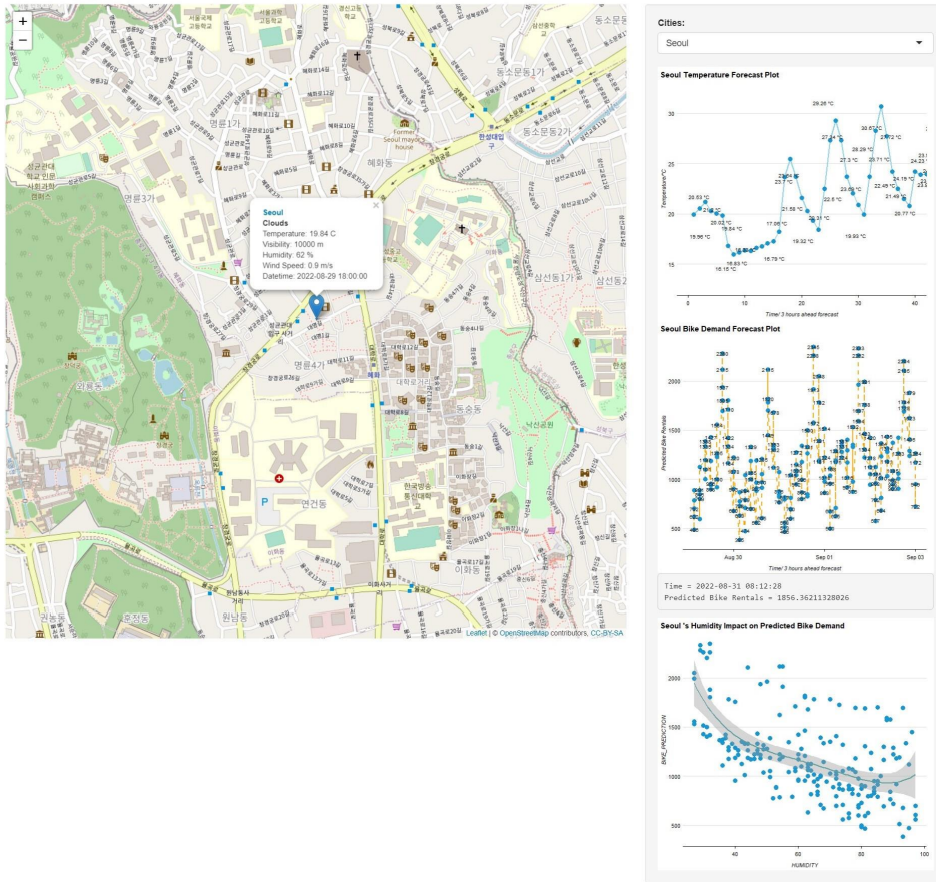
Bike-sharing demand prediction app



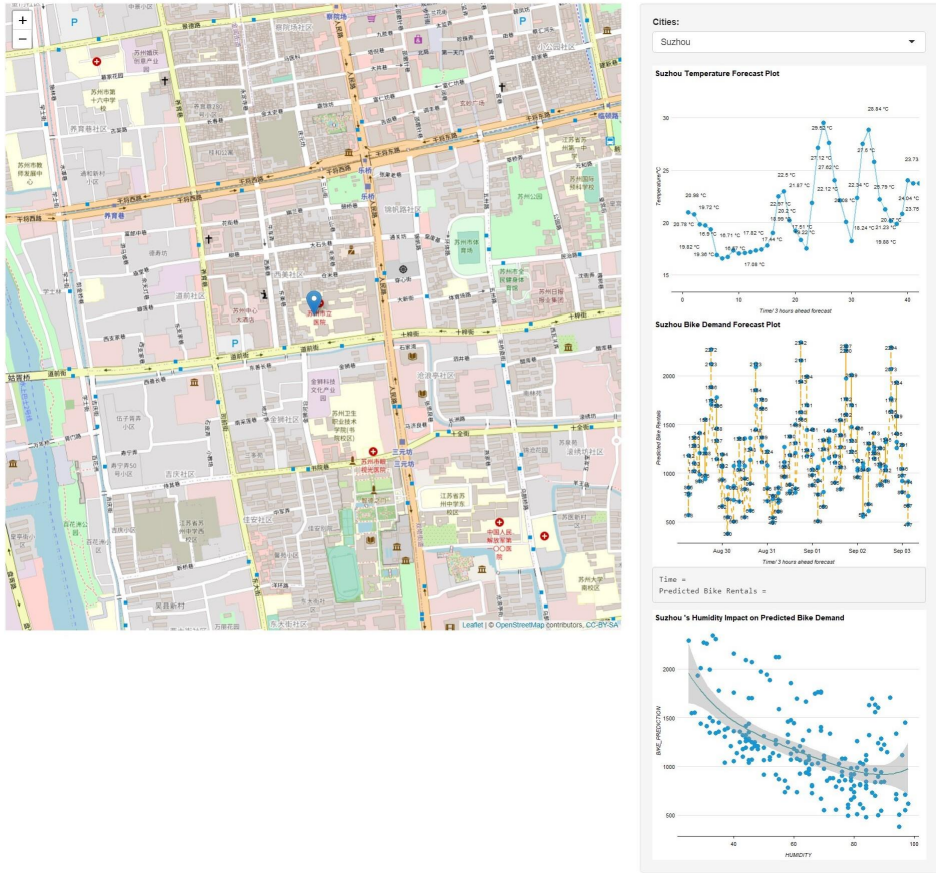


Results

Bike-sharing demand prediction app



Bike-sharing demand prediction app



EDA with SQL

Exploratory Data Analysis (EDA) with SQL is conducted. SQL is used to query various datasets so that relationships between different variables can be established which may then be used to conduct further Predictive Analysis. Query results are used to look at patterns and study relationships in some detail.

Busiest bike rental times

- Find dates and hours which had the most bike rentals

DATE	HOUR
<chr>	<int>
19/06/2018	18

From the query result, it is clear that the busiest date for Bike Rentals in Seoul according to the dataset was 19th June, 2018 and the busiest hour was 18 or 6 pm in the evening. Incidentally, according to the data set, this was one of the most popular hours for Bike Rentals in general. The other such hour was 8 am in the morning.

Hourly popularity and temperature by seasons

- Find hourly popularity and temperature by season

SEASONS	AVERAGE_TEMPERATURE	AVERAGE_BIKE_RENTALS
<chr>	<dbl>	<dbl>
Autumn	13.821580	924.1105
Spring	13.021685	746.2542
Summer	26.587711	1034.0734
Winter	-2.540463	225.5412

From the result, it is clear that there is some sort of a relationship between Temperature and Bike Rentals, as lower temperatures in Winter show decreased Bike Rentals while higher temperatures in other seasons show increased Bike Rentals. But the nature of relationship is not clear. Between both Spring and Autumn, the temperature variation is small but the change in Bike Rentals is huge. While between Autumn and Summer, the temperature variation is huge but the change in average Bike Rentals is small.

Hourly popularity and temperature by seasons

- The query on Seasonality in the previous slide clearly indicates how seasonal phenomenon affect Average Number of Bike Rentals. The highest rentals are in Summer, the second highest in Autumn, the third highest in Spring and the lowest number in Winter. Interestingly, the gap between Summer and Winter is 4.5 times while that between Spring and Winter is 3.3 times. In other words, only 7.6% of the Total Bike Rentals were in Winter, making it very unpopular for bike rentals.

Rental Seasonality

- Rental Seasonality
- The query on Seasonality below clearly indicates how seasonal phenomenon affect Average Number of Bike Rentals. The highest Average Rentals are in Summer, the second highest in Autumn, the third highest in Spring and the lowest amount in Winter. The same holds true for Maximum Rentals. This holds true for Standard Deviation also except that Autumn and Spring have about the same value. Interestingly, while Summer has the highest Minimum Rentals, Winter has the second highest Minimum Rentals while Spring and Autumn are tied for the last place. So there is some factor that keeps Minimum Rentals in Winter high.

SEASONS	AVERAGE_BIKE_RENTALS	MINIMUM_BIKE_RENTALS	MAXIMUM_BIKE_RENTALS	STANDARD_DEVIATION_BIKE_RENTALS
<chr>	<dbl>	<int>	<int>	<dbl>
Autumn	924.1105	2	3298	617.3885
Spring	746.2542	2	3251	618.5247
Summer	1034.0734	9	3556	690.0884
Winter	225.5412	3	937	150.3374

Weather Seasonality

- Weather Seasonality
- Weather phenomenon due to seasons may explain significant part of the variation in Average Bike Rentals. Humidity may be a factor here as changes in it by season track changes in Average Bike Rental somewhat. There is little variation there between Spring and Autumn to explain the change in Rentals though. Dew-point Temperature may also explain some of the changes as its changes track changes in Rentals by Season quite clearly. Lots of Snowfall might be the reason for unpopularity of bikes in Winter. Solar Radiation might also be a factor although it too fails to adequately explain the relationship between difference in Rentals in Autumn and Spring. Rainfall also tracks changes in seasons and Rentals closely and may explain the variation in Rentals.

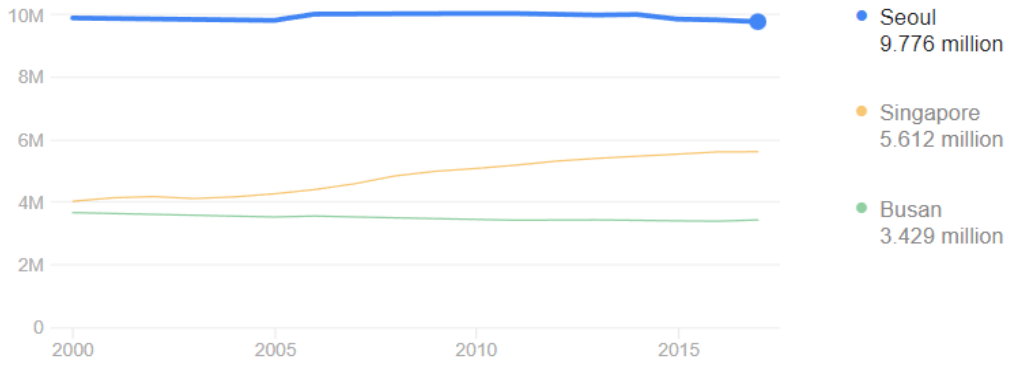
SEASONS	avg(TEMPERATURE)	avg(HUMIDITY)	avg(VISIBILITY)	avg(WIND_SPEED)	avg(DEW_POINT_TEMPERATURE)	avg(SOLAR_RADIATION)	avg(RAINFALL)	avg(SNOWFALL)	AVERAGE_BIKE_RENTALS	RANK
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<int>
Summer	26.587711	64.98143	1501.745	1.609420	18.750136	0.7612545	0.25348732	0.00000000	1034.0734	1
Autumn	13.821580	59.04491	1558.174	1.492101	5.150594	0.5227827	0.11765617	0.06350026	924.1105	2
Spring	13.021685	58.75833	1240.912	1.857778	4.091389	0.6803009	0.18694444	0.00000000	746.2542	3
Winter	-2.540463	49.74491	1445.987	1.922685	-12.416667	0.2981806	0.03282407	0.24750000	225.5412	4

Bike-sharing info in Seoul

- Find the total Bike count and city info for Seoul

CITY	COUNTRY	LAT	LNG	POPULATION	BICYCLES
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>
Seoul	Korea, South	37.5833	127	21794000	37500

9.776 million (2017)



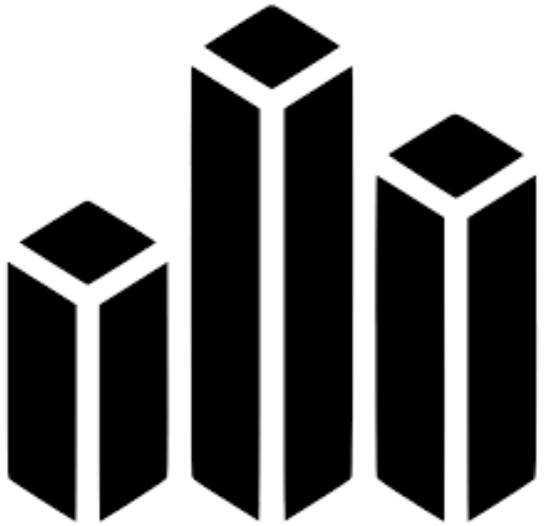
The Geographical Information for the city of Seoul is presented here including its Latitude, Longitude and Population as well as Bike Count.
Interestingly, the Population for Seoul in the dataset appears to be much larger than the Population listed in other sources. (See the result from Google below the Query Result)

Cities similar to Seoul

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

CITY	COUNTRY	LAT	LNG	POPULATION	BICYCLES
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>
Beijing	China	39.9050	116.3914	19433000	16000
Ningbo	China	29.8750	121.5492	7639000	15000
Shanghai	China	31.1667	121.4667	22120000	19165
Weifang	China	36.7167	119.1000	9373000	20000
Zhuzhou	China	27.8407	113.1469	3855609	20000

All cities possessing between 15,000 to 20,000 Bicycles are listed here along with their Geographical Co-ordinates and Population as these are the cities closest Seoul in Bike Scale.
Interestingly, all of these cities are located in China.



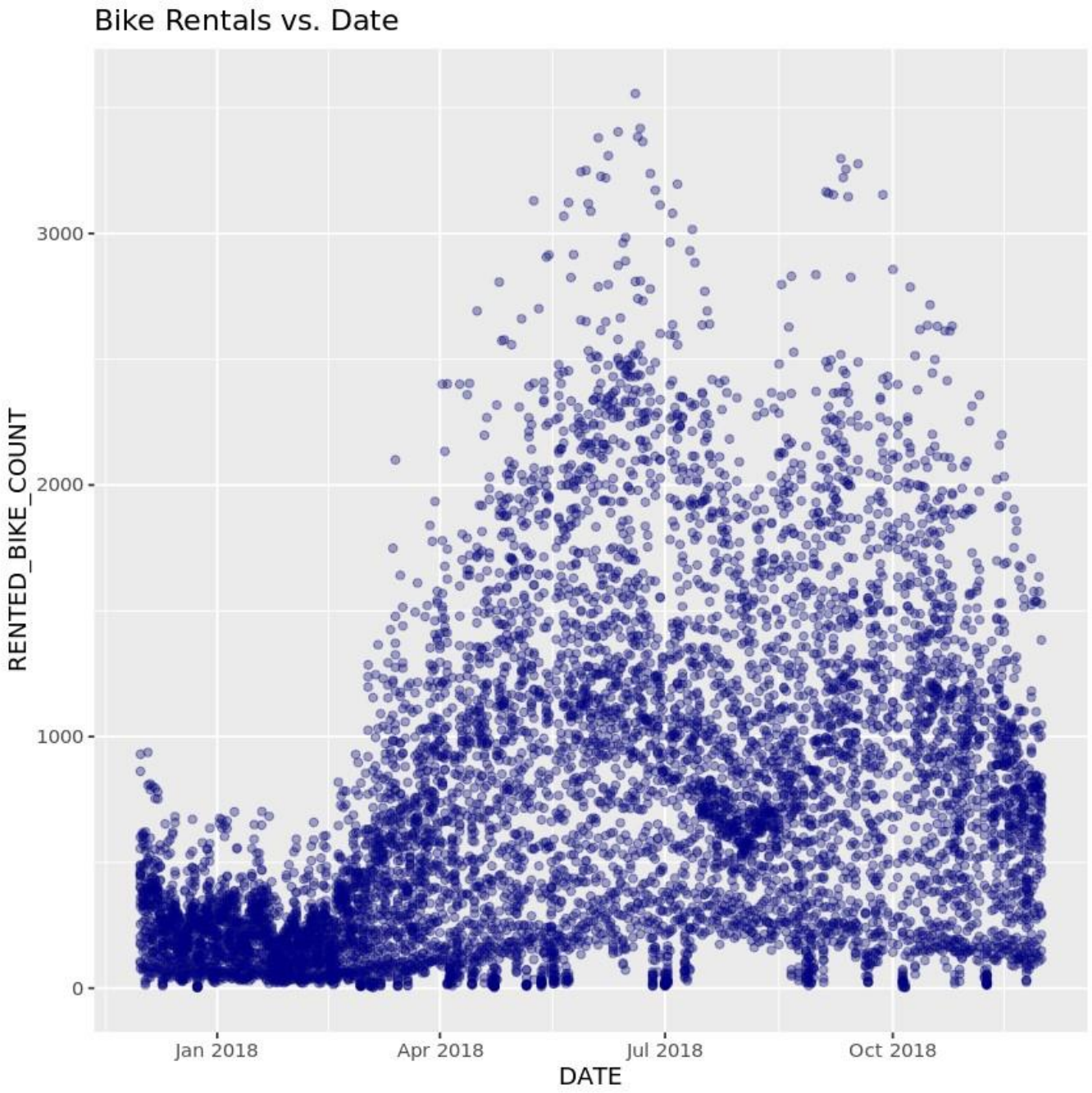
EDA with Visualization

Exploratory Data Analysis (EDA) continues. Various visualizations are displayed by utilizing the graphing capabilities of the ggplot2 package in R. The visualizations shed light on the relationship between various weather variables and Bike Rentals. Time related variables and their relationship to Bike Rentals is considered as well.

Bike rental vs. Date

The plot clearly shows a seasonal trend to bike rentals. The number of rentals is lesser in the months from December to April. But then a rising trend in rentals begins around March and reaches a peak in July. The trend starts to fall somewhat then starting from October and reaches the low level in December.

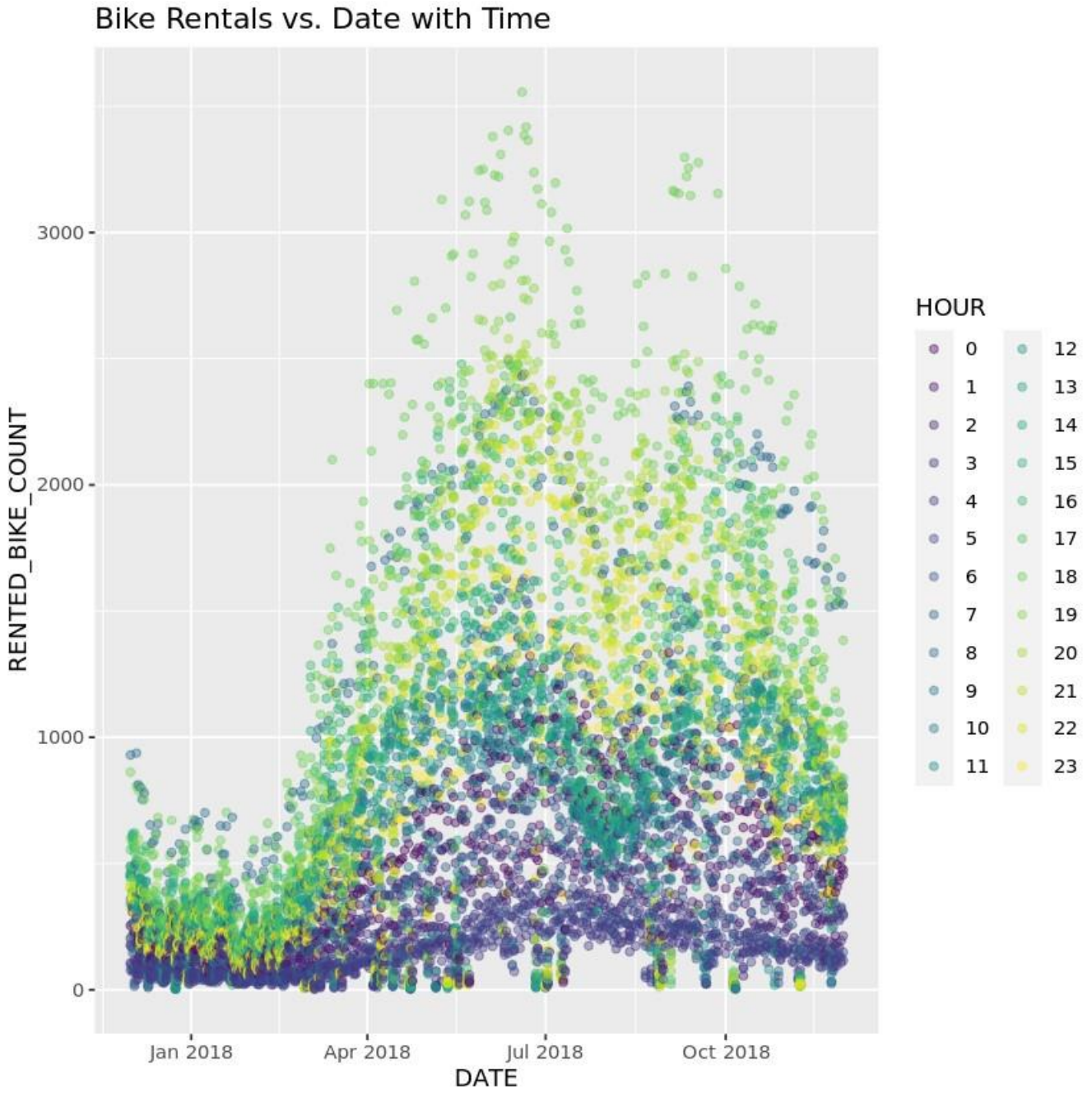
These months roughly correspond to seasons. It appears that Bike Rentals are lowest in Winter and highest in Summer. This might be because people would choose alternative transport during the cold winter weather but prefer to enjoy the better weather such as sunlight particularly in the summer months.



Bike rental vs. Date and Time

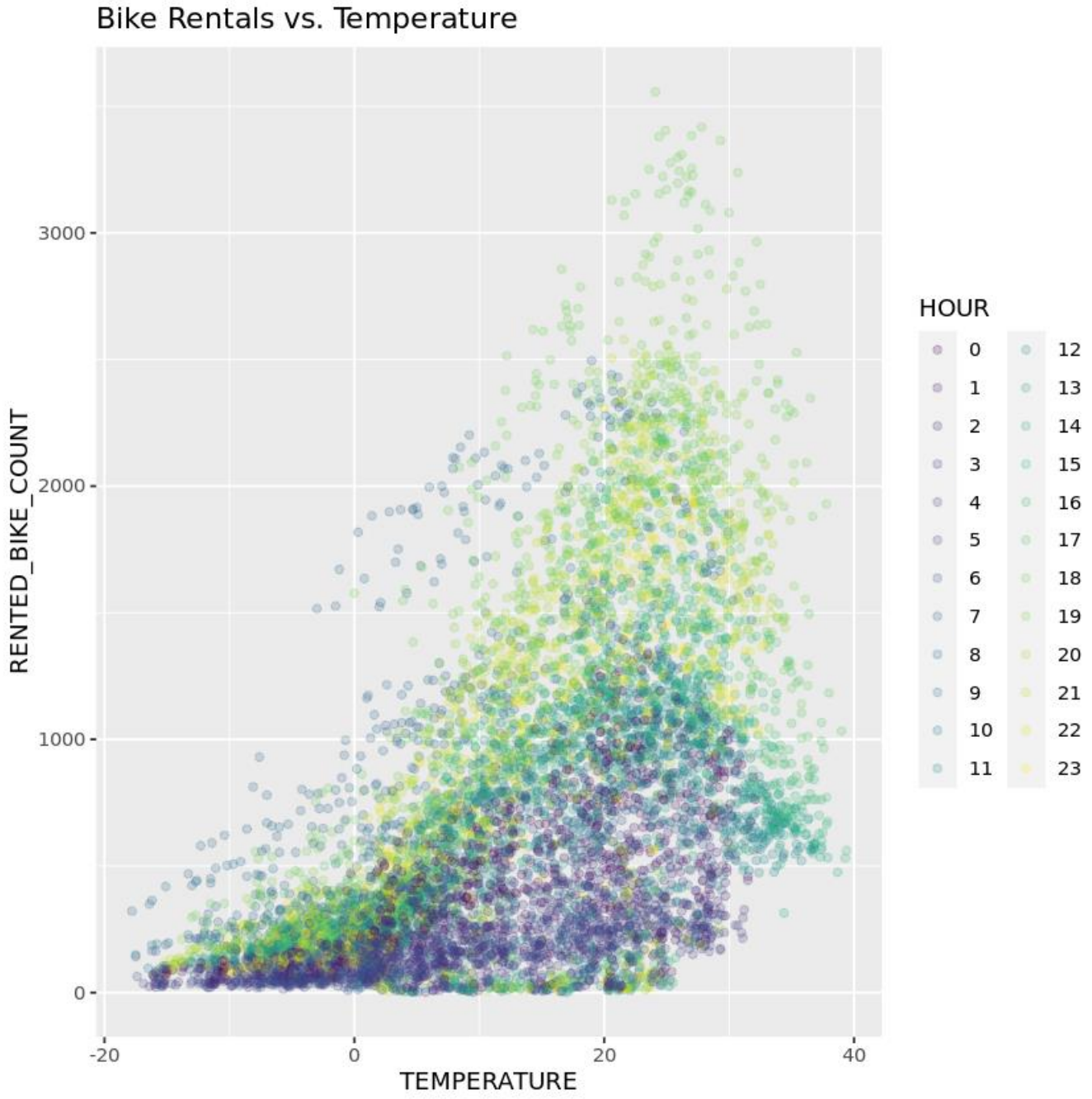
Time also plays a role here. It can be observed from the plot that the rising trend in Bike Rentals from March onward is driven more and more by Rentals in the daylight hours but rentals at night contribute to the rise also. In fact, there appear to be rentals throughout the day and night from May to November.

This too corresponds closely to the Seasons. People appear to enjoy renting bikes in the summer even after the sun goes down because the weather may still be pleasant. Night rentals are common throughout the year which is an interesting observation that may require more details to explain.



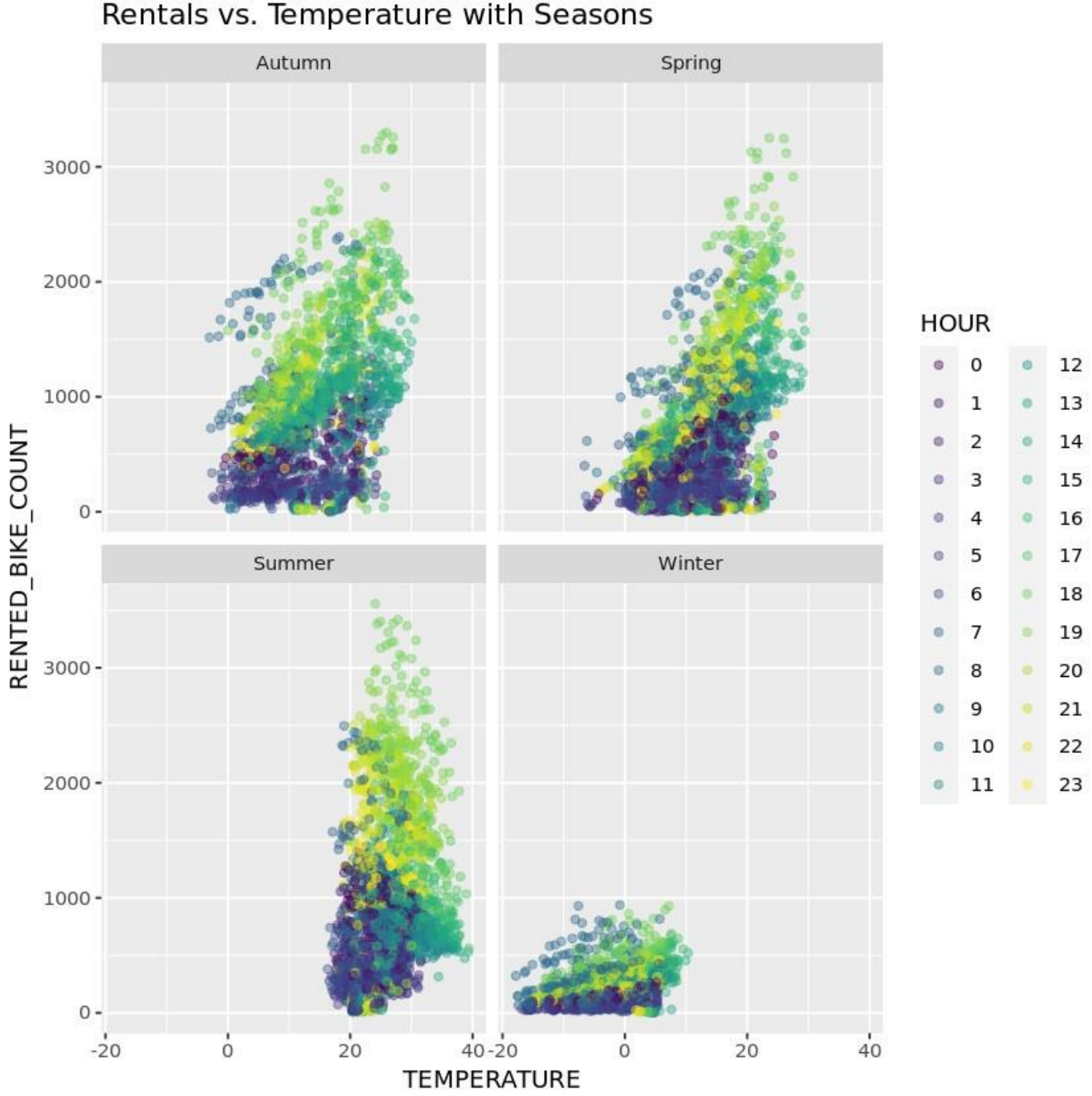
Bike Rentals vs. Temperature

Another important variable to consider for Bike Rentals is Temperature. There is considerable variation in Temperature over the year due to Seasonal changes. Demand for Bike Rentals varies significantly over the year due to Temperature variation as people would prefer alternate transportation if the Temperature is too low or high. The plot of Temperature vs. Bike Rentals confirms this. Observe that at extremes of low and high Temperature demand is lower. At very low Temperature demand is much lower while at high Temperature it is lesser. Peak demand is at temperature between 20 and 30 degrees. Timing also varies as at higher Temperature Rentals are in demand during morning hours.



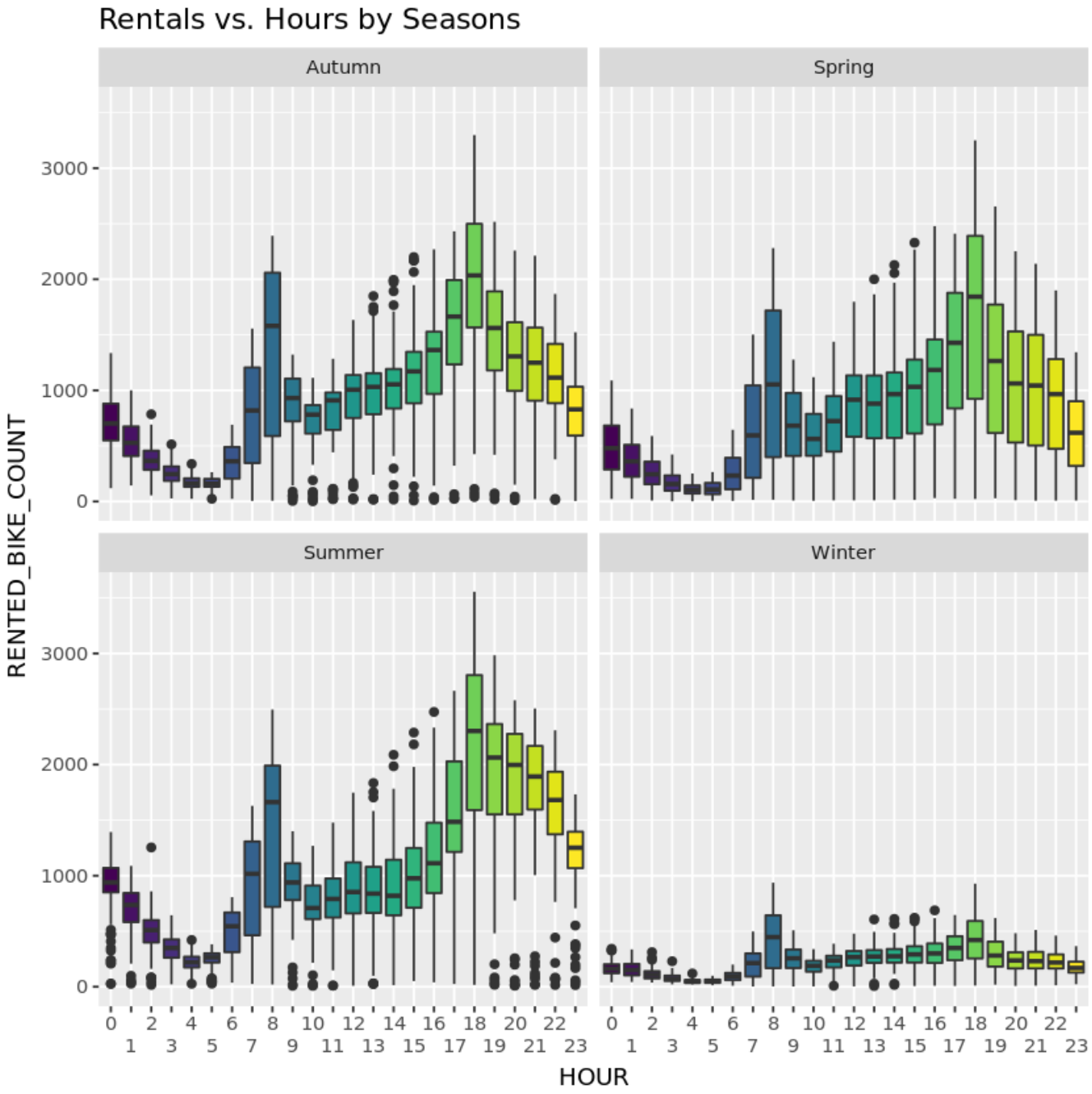
Rentals vs. Temperature with Seasons

As noted in the previous plot, Temperature seems to have a significant impact on Bike Rentals. This impact becomes more evident when temperature is considered over the 4 seasons. It can be seen that demand is much lesser in Winter due to lower temperatures. In Summer it is high and primarily driven by rentals in the evening. In Autumn and Spring also, the overall demand is decent. But the rental timings vary somewhat. There is more demand for Rentals during daylight hours around and after noon in Autumn than in Spring, despite similar temperature. This indicates that variables other than temperature may explain differences between Autumn and Spring.



Rentals vs. Hours by Seasons

To understand the impact of hour of the day on Bike Rentals box plots based on season were plotted. Interestingly, the overall distribution of variables remained the same over each season. The peak demand was at 18 or 6 pm and the second peak was at 8 am. This is sensible as work days don't change over the seasons and at 8 am people need to get to work and at 6pm they need to get off of work so demand is higher during those times. In winter though these two demand peaks are same. These may be people who use bikes for commuting throughout the year. Also, in every season there is some demand for bikes even at night hours.



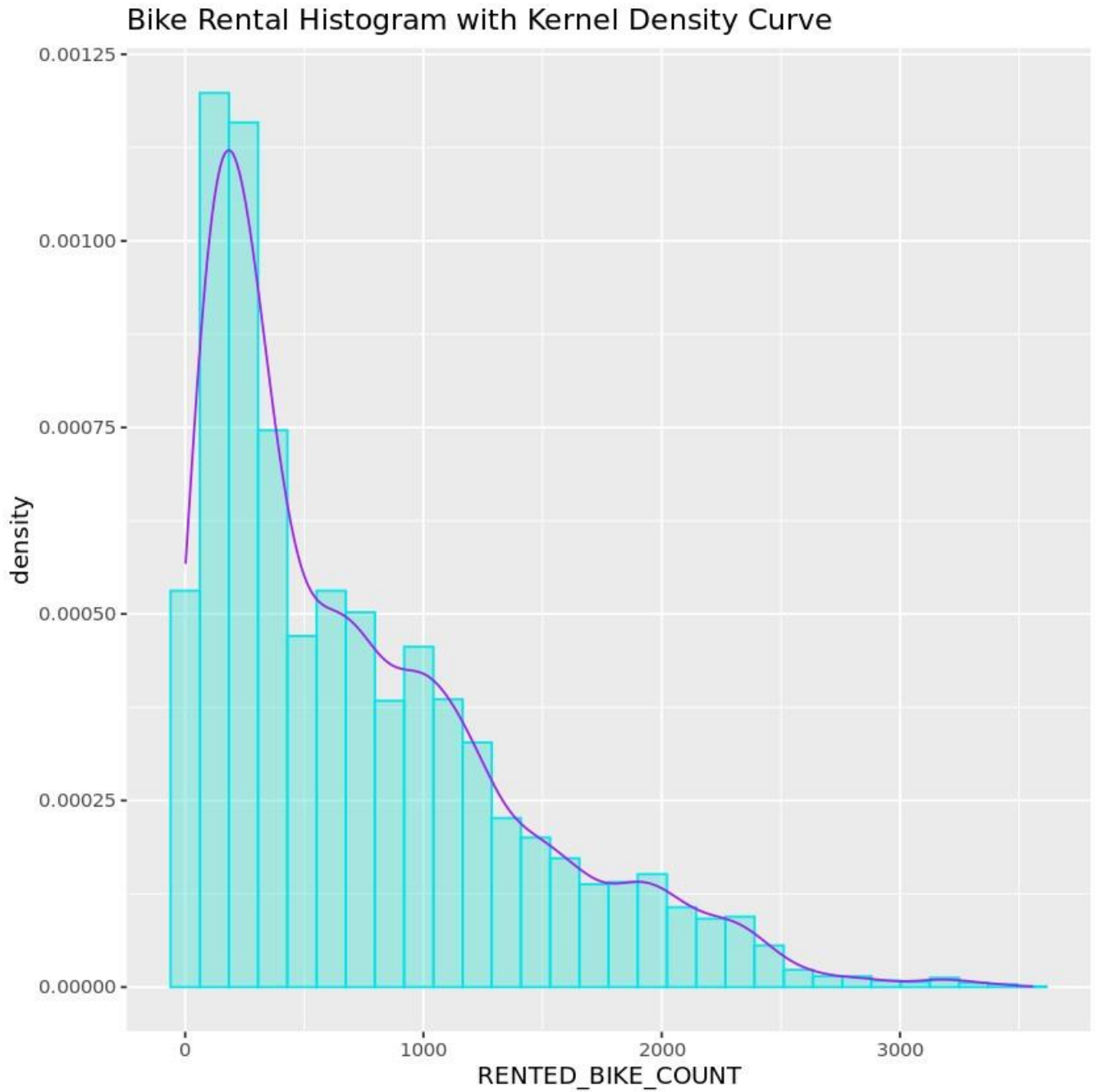
Bike rental histogram

The histogram represents the frequency with which different numbers of bikes are rented out at a time according to the data.

From the histogram and the accompanying curve, it can be seen that the highest number of bikes rented out at a time or mode is in the 125-300 range.

However, the sudden bumps in the number of rented bikes at around 650 and 900 rentals, as well as at around 1900 and 3200 indicate that there are certain factors which may have modes for sub-groups of the data.

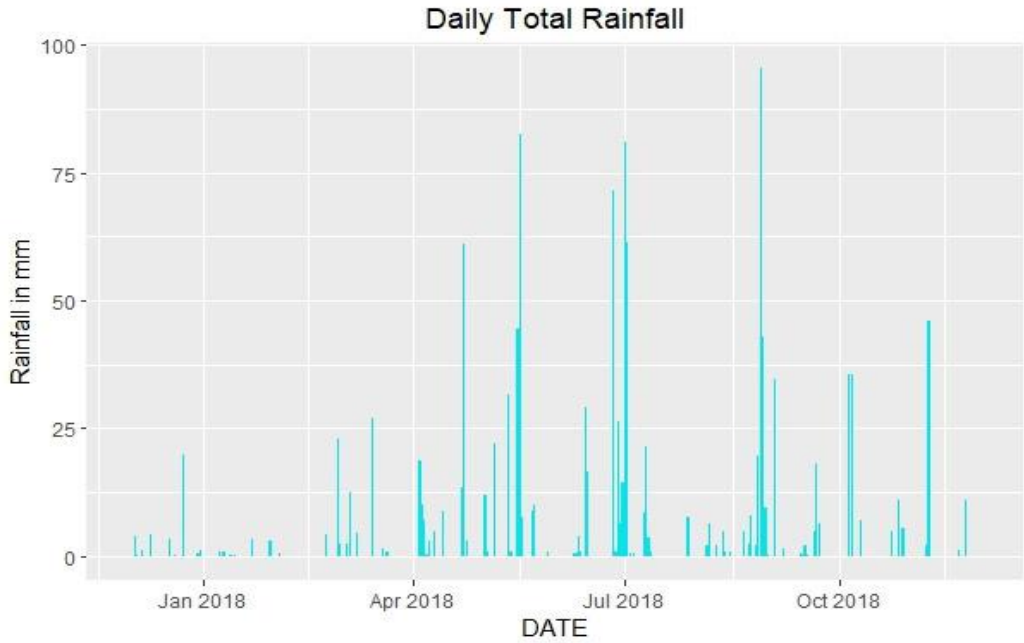
The tail of the distribution is long, indicating that there are many times when bikes much larger than usual amounts are rented out. The bump at 3200 is quite interesting in this regard.

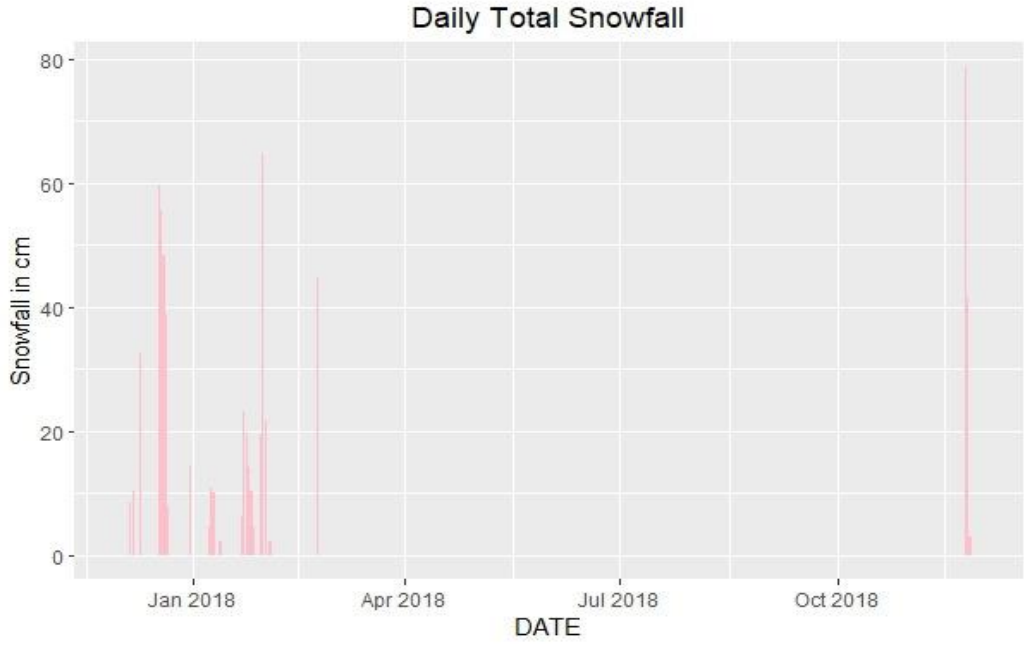


Daily total rainfall and snowfall

Show bar charts calculating the daily total rainfall and snowfall:

As is clear from the plots, Rainfall occurs intermittently throughout the year. It occurs in highest amounts from May to September though. It is lowest during months when Snowfall occurs. This is not surprising. In Winter months, when Snowfall occurs intermittently, Rainfall is low because rain clouds turn into snow clouds due to low temperatures. Given the low Bike Rentals in Winter Season, it is tempting to assume that Snowfall may be the reason for these lower Bike Rentals.



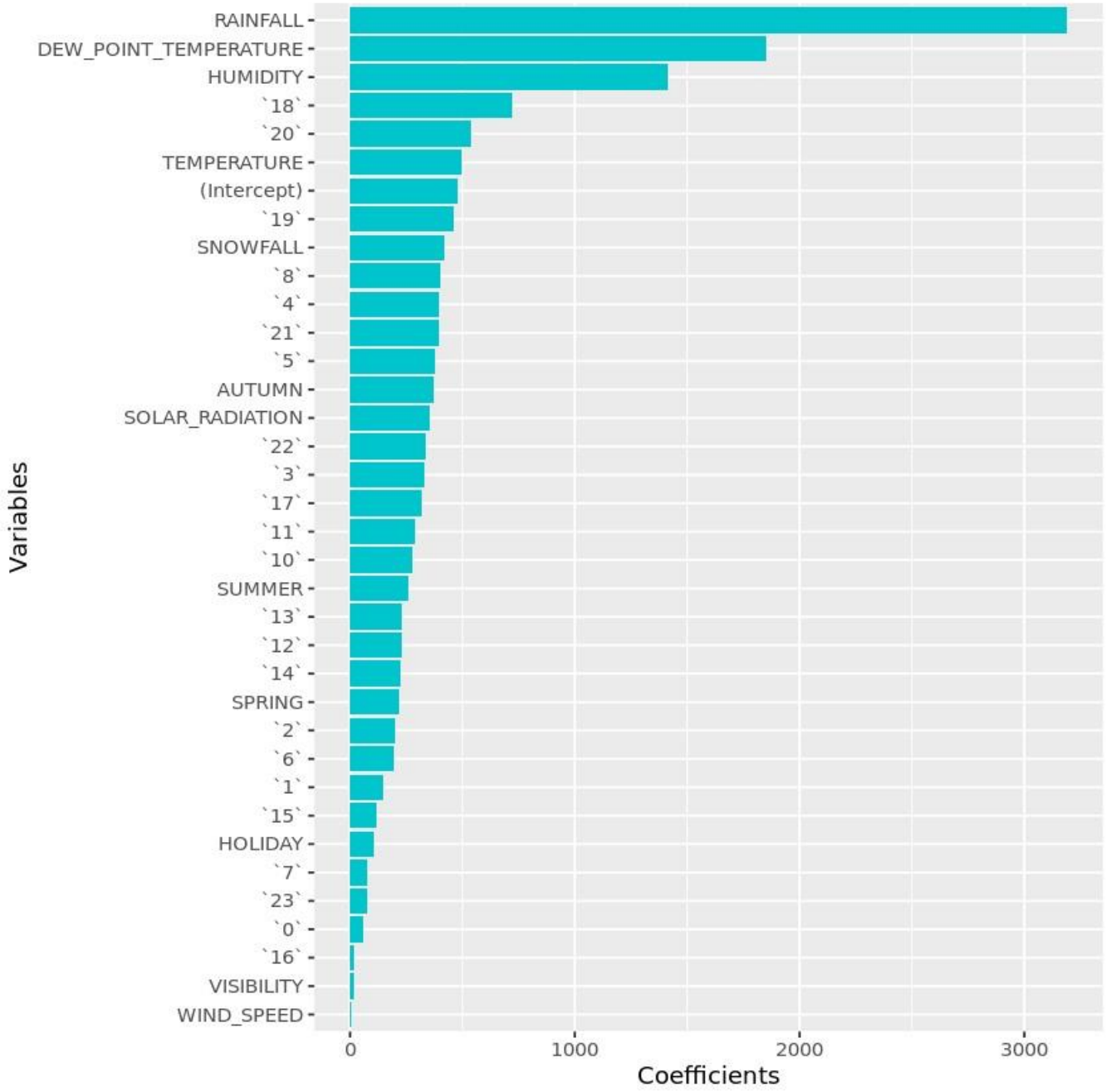


Predictive analysis

Predictive analysis makes use of statistical modeling and other techniques to make use of available data so that estimates about future values of variables of interest can be made. It yields better insights with large amounts of data and with models that are better suited for making estimations from the data.

Ranked coefficients

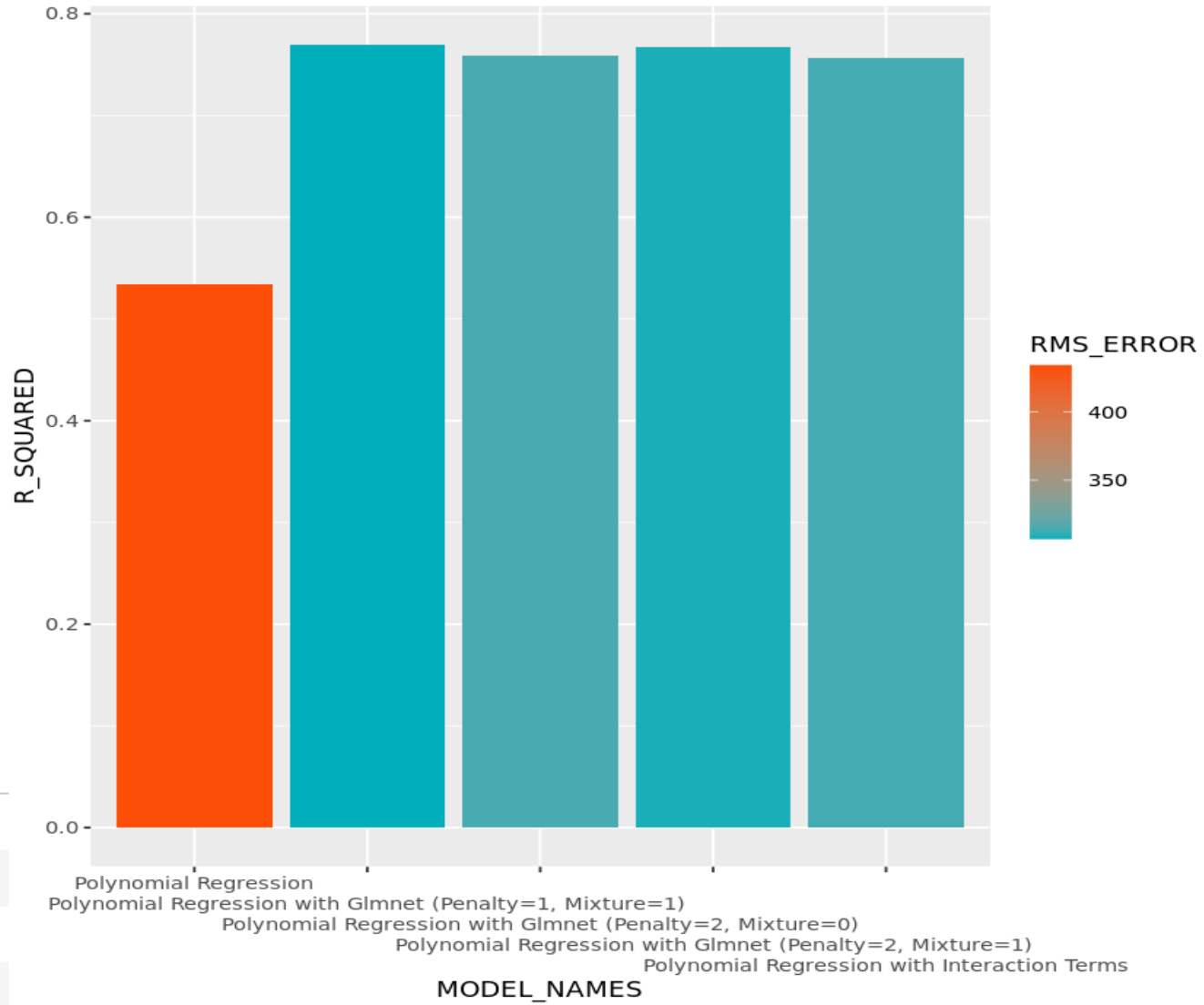
The coefficients are ranked in accordance with their explanatory power for the response variable Bike Count.



Model evaluation

The models evaluated along with their details and R-Squared and Root Mean Square Error coefficients are given below. A grouped bar chart of their values is given as well.

MODEL_NAMES	R_SQUARED	RMS_ERROR
<fct>	<dbl>	<dbl>
Polynomial Regression	0.534	434.34
Polynomial Regression with Interaction Terms	0.757	313.84
Polynomial Regression with Glmnet (Penalty=1, Mixture=1)	0.769	306.93
Polynomial Regression with Glmnet (Penalty=2, Mixture=1)	0.767	308.13
Polynomial Regression with Glmnet (Penalty=2, Mixture=0)	0.759	314.30



The best performing model

- The formula for the best performing model is:
- RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 16) + poly(HUMIDITY, 12) + poly(DEW_POINT_TEMPERATURE,12) + poly(SOLAR_RADIATION,14) + VISIBILITY + RAINFALL*HUMIDITY + TEMPERATURE*RAINFALL + TEMPERATURE*HUMIDITY + DEW_POINT_TEMPERATURE*HUMIDITY + TEMPERATURE*SOLAR_RADIATION + poly(RAINFALL,12) + `18` + `20` + `19` + `8` + `4` + `21` + `5` + AUTUMN + `22` + `3` + `18`*TEMPERATURE + `18`*SOLAR_RADIATION + HOLIDAY + `20`*TEMPERATURE + SUMMER + SPRING + `17` + `11` + `10`

The best performing model

[15]:
```
glmnet_spec <- linear_reg(penalty=1, mixture=1) %>%
               set_engine(engine="glmnet")
```

Fit a glmnet model called `lm_glmnet` using the `fit()` function. For the formula part, keep the polynominal and interaction terms you used in the previous task.

[16]:
```
# Fit a glmnet model using the fit() function
lm_glmnet <- glmnet_spec %>% fit(RENTED_BIKE_COUNT ~ poly(TEMPERATURE, 16) + poly(HUMIDITY, 12) + poly(DEW_POINT_TEMPERATURE,12) +
                                 poly(SOLAR_RADIATION,14) + VISIBILITY + RAINFALL*HUMIDITY + TEMPERATURE*RAINFALL + TEMPERATURE*HUMIDITY +
                                 DEW_POINT_TEMPERATURE*HUMIDITY + TEMPERATURE*SOLAR_RADIATION + poly(RAINFALL,12) + `18` + `20` + `19` + `8` +
                                 `4` + `21` + `5` + AUTUMN + `22` + `3` + `18`*TEMPERATURE + `18`*SOLAR_RADIATION + HOLIDAY + `20`*TEMPERATURE +
                                 SUMMER + SPRING + `17` + `11` + `10`, data=train_data)
```

[17]:
```
# Report rsq and rmse of the `lm_glmnet` model
glmnet_test <- lm_glmnet %>% predict(new_data=test_data) %>% mutate(truth=test_data$RENTED_BIKE_COUNT)
glmnet_test[glmnet_test<0] <- 0
```

[18]:
```
rsq_glmnet <- rsq(glmnet_test,truth=truth,estimate=.pred)
rmse_glmnet <- rmse(glmnet_test,truth=truth,estimate=.pred)
rsq_glmnet
```

A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rsq	standard	0.7688285

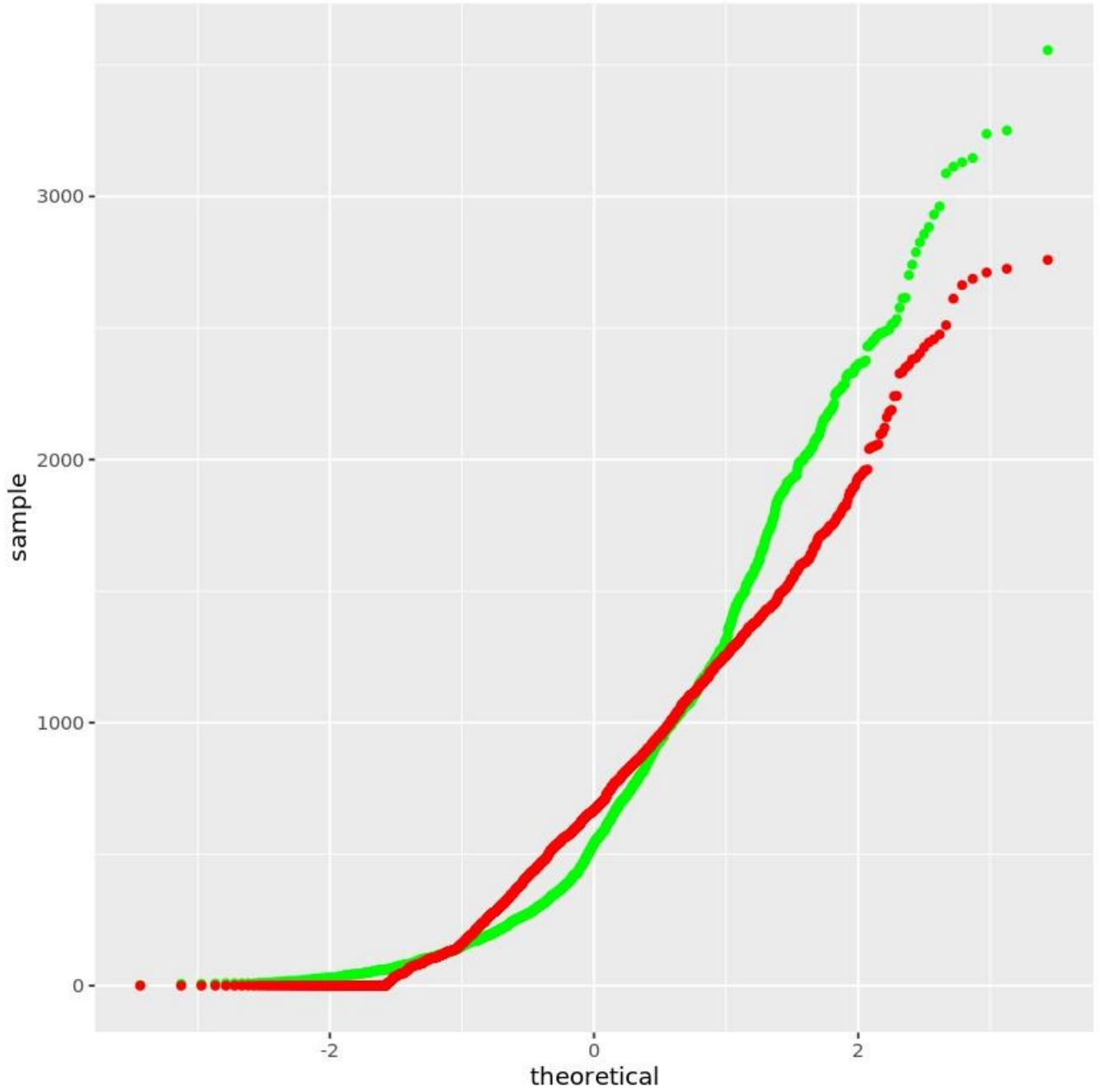
[19]:
```
rmse_glmnet
```

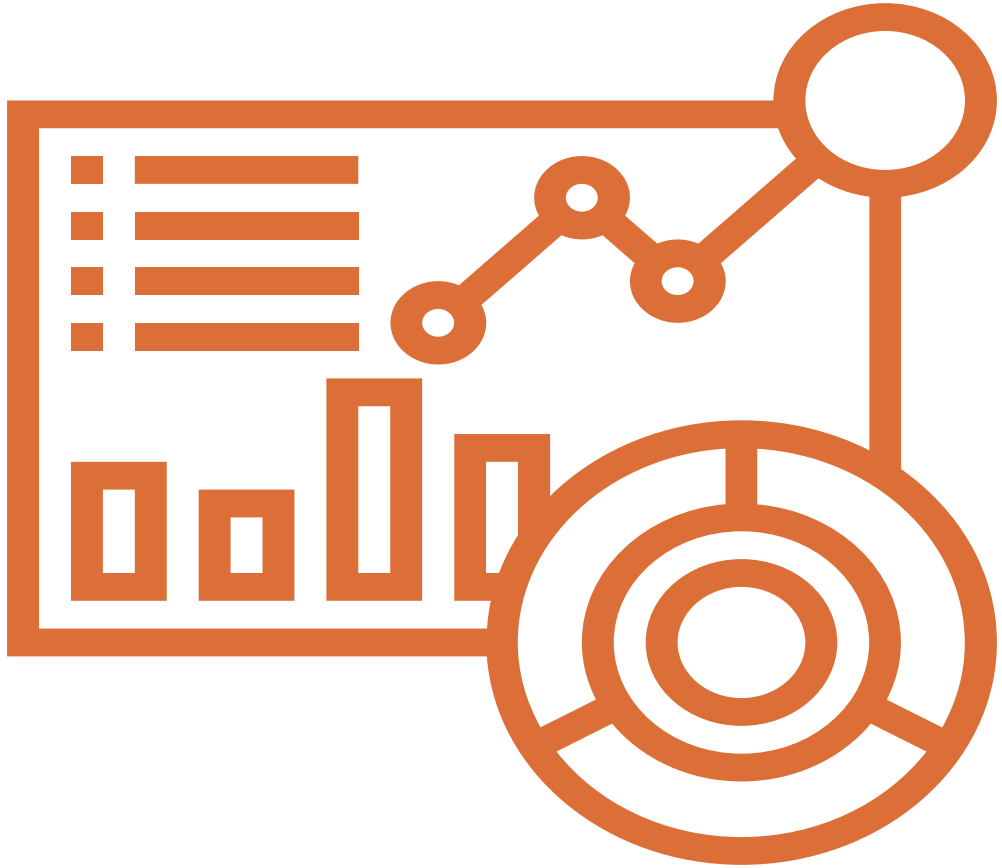
A tibble: 1 × 3

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
rmse	standard	306.9332

Q-Q plot of the best model

The Q-Q Plot of the values obtained by estimation on test dataset by making use of the best model.





Dashboard

An R-Shiny based Dashboard that displays the Temperature Forecast, Predicted Bike Demand Forecast for the next 5 days and a correlation plot between Humidity and Biking Demand for London, Paris, New York, Suzhou and Seoul.

Dashboard Global Bike Demand View

Bike-sharing demand prediction app

Cities:

All

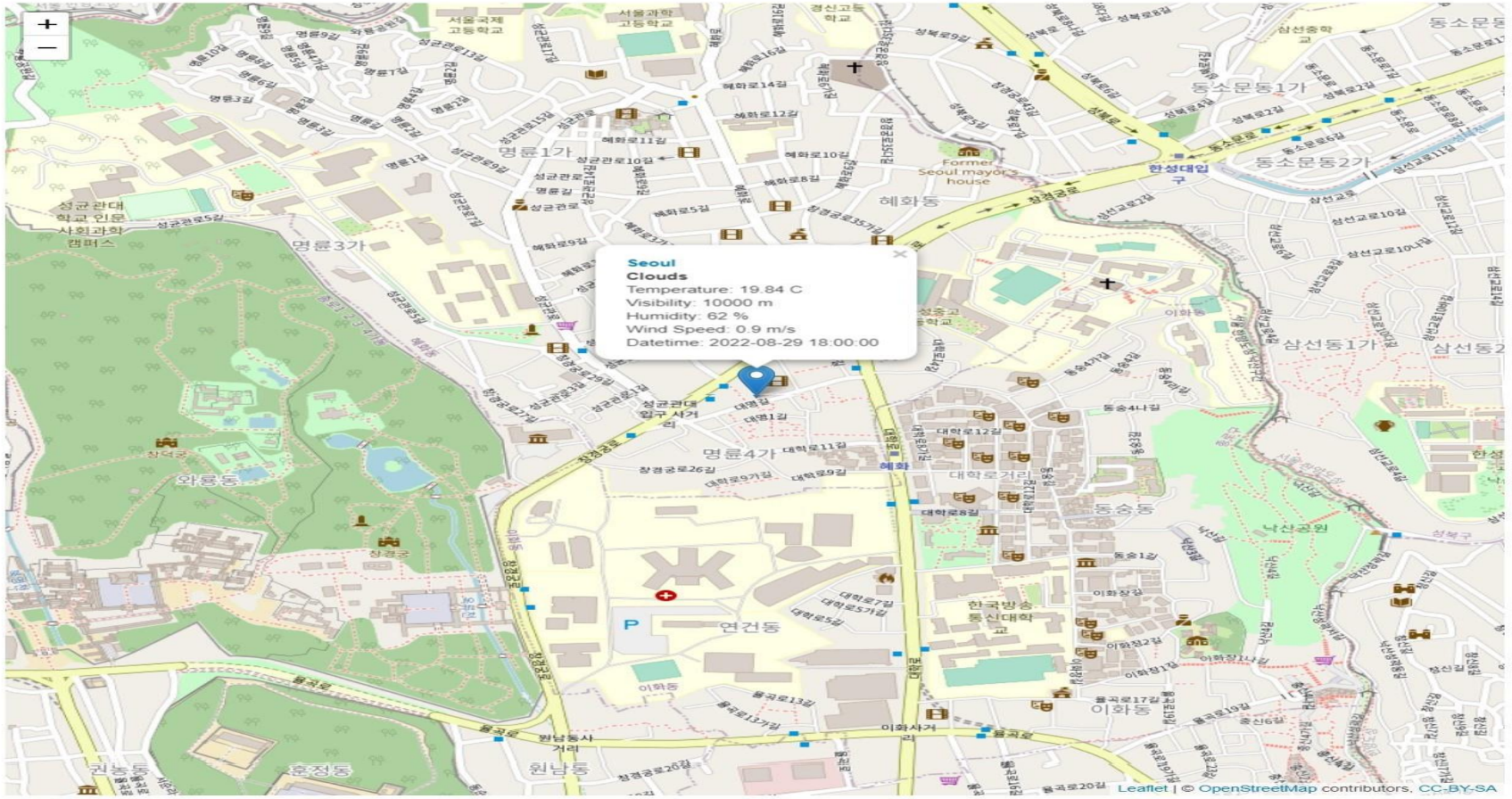
All
Seoul
Suzhou
London
New York
Paris

Dashboard Global Bike Demand View

- A map of Predicted Bike Demand for the cities of London, New York, Paris, Suzhou and Seoul.
- The map displays the predicted bike demand through color and radius of markers. A green color with small radius indicates low demand, yellow color with medium radius indicates normal demand and red color with large radius indicates large demand.
- The Cities drop-down list allows users to choose between the different cities mentioned above.
- Clicking on a City Marker on the map displays a Popup with the name of the City and its Current Weather. Only one city Popup may be displayed at a time.

Dashboard City View - Seoul

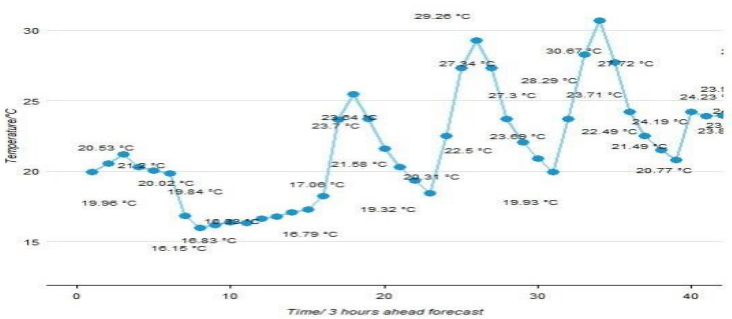
Bike-sharing demand prediction app



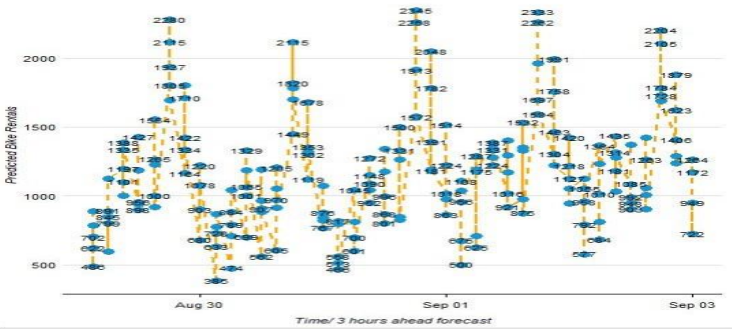
Cities:

Seoul

Seoul Temperature Forecast Plot

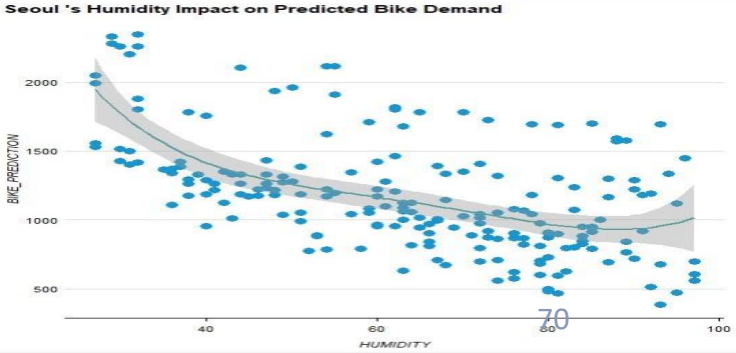


Seoul Bike Demand Forecast Plot




```
Time = 2022-08-31 08:12:28
Predicted Bike Rentals = 1856.36211328026
```

Seoul 's Humidity Impact on Predicted Bike Demand

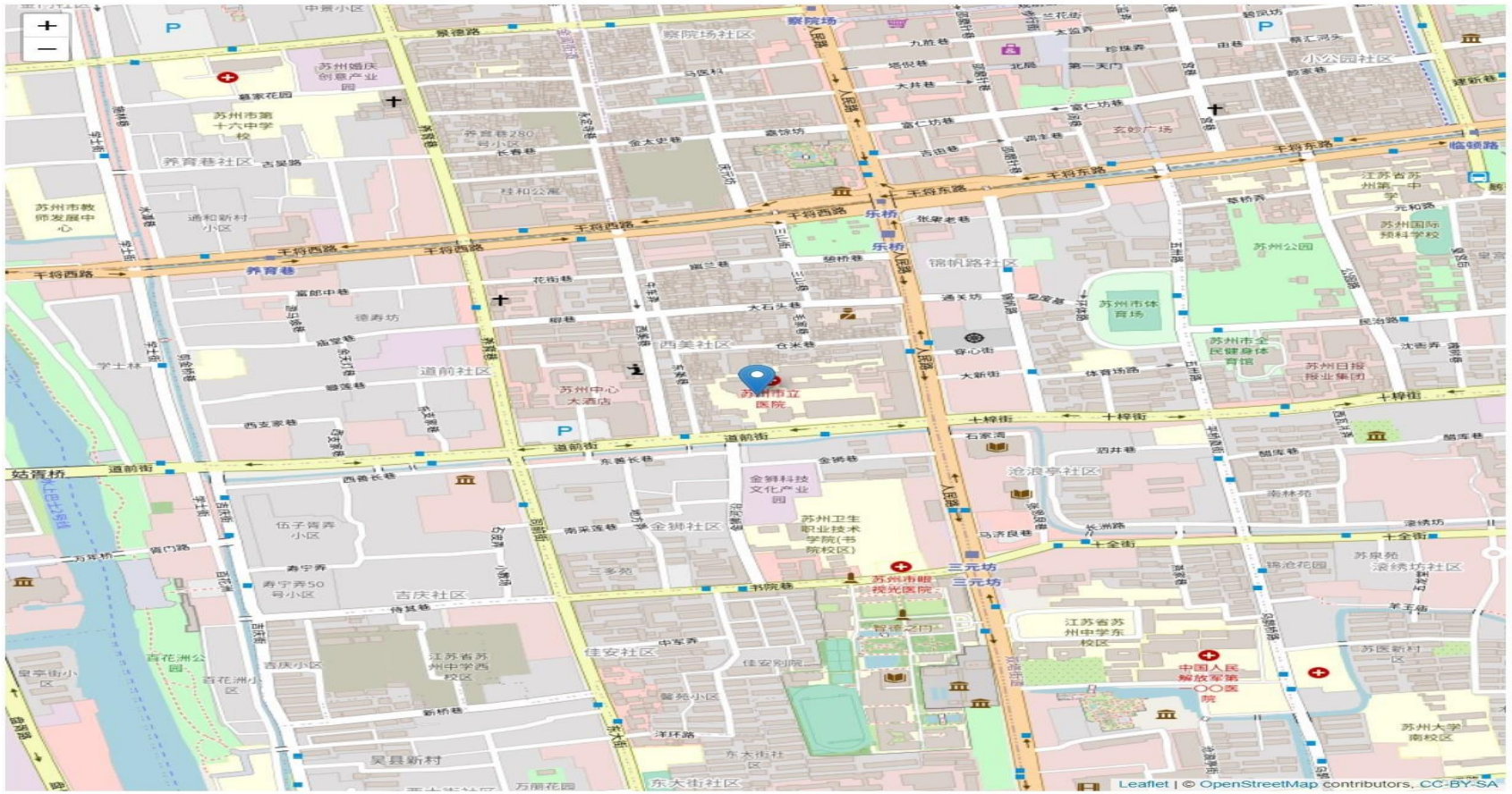


Dashboard City View - Seoul

- The city view of the dashboard displays a map of the selected city. Clicking on the map opens a Popup with details about weather conditions in the city as well as time information.
- There is a Temperature Forecast plot to the right below the Cities: dropdown list. The temperature values on the plot are labelled and end with a °C.
- There is also a Bike Demand Forecast plot which is right below the Temperature Plot, which displays predicted Bike Demand for the next 5 days. Users can click on the plot and the Text Box below displays the chart values for the point that was clicked.
- Below the Bike Demand Forecast Plot, there is a regression plot displaying the impact of Humidity on Forecasted Bike Demand.

Dashboard City View - Suzhou

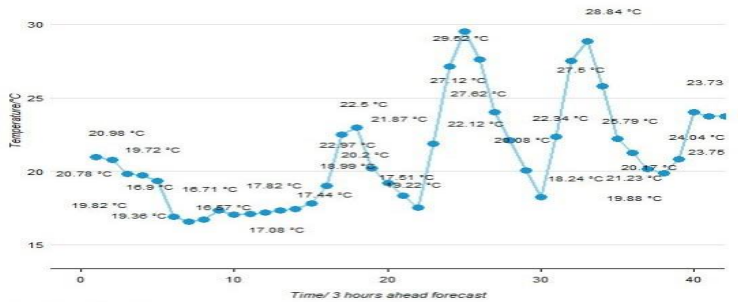
Bike-sharing demand prediction app



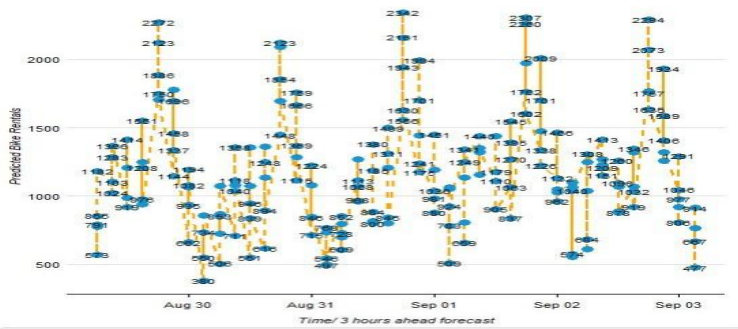
Cities:

Suzhou

Suzhou Temperature Forecast Plot

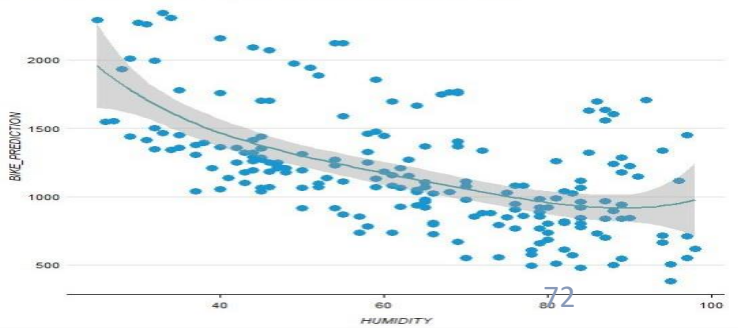


Suzhou Bike Demand Forecast Plot



Time =
Predicted Bike Rentals =

Suzhou 's Humidity Impact on Predicted Bike Demand



Dashboard City View - Suzhou

- Another city from the dropdown list, Suzhou instead of Seoul is selected. The map and the plots change accordingly.
- The city details are not visible in this view because the blue marker was not clicked. The details Popup appears when the marker is clicked.
- The plots reflect the change in the underlying data as Suzhou's weather conditions are different from Seoul.
- All cities in the dropdown list have their own map displayed and the plots reflect the data of the selected city.
- The Text Box below Bike Demand Forecast is empty because no point on the graph was clicked in this view.

CONCLUSION

In this project, the impact of weather on biking demand was investigated. Data for weather and biking demand, in the form of Bike Rentals for city of Seoul was used. This data was modeled through Linear Regression techniques and these techniques were further refined. The best results were obtained from a Generalized Linear Model (GLM) which made use of Regularization and lots of features including polynomial terms, simple variables and interaction terms. It managed to explain about 77 % of the variation in Bike Rentals (R-Squared of 0.768 and Root Mean Square Error of 306.93).

These results indicated a clear relationship between weather and Bike Demand. In particular, there was a large variation in Bike Demand between seasons. Summer saw the highest demand overall while Winter had the least demand of all the seasons. Temperature played a major factor in this. Extremes of Temperature discouraged bike use. Other variables, such as Humidity, Rainfall and Snowfall played their role also. A nonweather variable that had a significant impact was Hour of the day. Bike Demand peaked during the evening rush hour and then the morning rush hour (6pm and 8 am respectively).

CONCLUSION

The findings presented important information about the impact of weather on bike demand. These findings were then used to create a Map-based Dashboard for 5 cities using R-Shiny. The Dashboard presented the option to choose any of the cities and displayed the selected city's map and plots of forecasted Temperature, Predicted Bike Demand for the next 5 days and impact of predicted Humidity on predicted Bike Demand.

The findings are useful for city authorities managing biking systems and may help them in assisting with decisions about maintaining, upgrading and expanding bike fleets. In addition, they may interest researchers considering transportation related issues. They may also interest climate researchers also as climate change might impact bike demand through weather changes. Furthermore, additional research may be conducted to augment these findings with similar research for other cities, impact of other variables such as fuel prices on bike demand and impact of pollution as well as context and culture on bike demand (biking may be seen as a lower status activity in some cultures or a symbol of a lower class in some contexts with attendant demerits).

APPENDIX

- Includes relevant assets such as R code snippets, SQL queries, charts, output etc. created during this project.

Appendix: Webscraping-1

•[1]:
```
# Check if need to install rvest` library
require("rvest")
library(rvest)
```

```
Loading required package: rvest
Loading required package: xml2
```

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

TODO: Get the root HTML node

[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
bike_page <- read_html(url)
```

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.

Appendix: Webscraping-2

[7]:
```r
# Convert the bike-sharing system table into a dataframe
bike_page_tables <- html_nodes(bike_page, "table")

for (table in bike_page_tables)  {
    print(table)
    }

bike_df <- html_table(bike_page_tables[2], fill= TRUE)
```

```
{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:inherit">
[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">
[1] <tbody>\n<tr>\n<th scope="row" class="navbox-group" style="width:1%">East ...
{html_node}
<table class="nowraplinks navbox-subgroup" style="border-spacing:0">
[1] <tbody>\n<tr>\n<th scope="row" class="navbox-group" style="width:1%">Cana ...
```

Summarize the bike sharing system data frame

[11]:
```r
# Summarize the dataframe
summary(bike_df)
```

```
     Length Class      Mode
[1,] 10     data.frame list
```

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

[16]:
```r
# Export the dataframe into a csv file
write.csv(bike_df,"Bike_Data_Frame.csv",row.names=FALSE)
```

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

Appendix: Open Weather API- 1

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

First import `httr` library

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

library(httr)
```

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

[60]:
```
# URL for Current Weather API
current_weather_url <- 'https://api.openweathermap.org/data/2.5/weather'
forecast_url <- 'https://api.openweathermap.org/data/2.5/forecast'
```

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

[114]:
```
# need to be replaced by your real API key
city_name12 <- c("Seoul")
your_api_key <- [redacted]
# 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 = city_name12, appid = your_api_key, units="metric")
```

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

[188]:
```
response <- GET(forecast_url, query=current_query)
```

If we check the response type, we can see it is in JSON format

[6]:
```
http_type(response)
```

'application/json'

Appendix: Open Weather API- 2

[14]:
```r
# Create some empty vectors to hold data temporarily
weather <- c()
visibility <- c()
temp <- c()
temp_min <- c()
temp_max <- c()
pressure <- c()
humidity <- c()
wind_speed <- c()
wind_deg <- c()
```

Now assign the values in the `json_result` list into different vectors

[15]:
```r
# $weather is also a list with one element, its $main element indicates the weather status such as clear or rain
weather <- c(weather, json_result$weather[[1]]$main)
# Get Visibility
visibility <- c(visibility, json_result$visibility)
# Get current temperature
temp <- c(temp, json_result$main$temp)
# Get min temperature
temp_min <- c(temp_min, json_result$main$temp_min)
# Get max temperature
temp_max <- c(temp_max, json_result$main$temp_max)
# Get pressure
pressure <- c(pressure, json_result$main$pressure)
# Get humidity
humidity <- c(humidity, json_result$main$humidity)
# Get wind speed
wind_speed <- c(wind_speed, json_result$wind$speed)
# Get wind direction
wind_deg <- c(wind_deg, json_result$wind$deg)
```

Combine all vectors as columns of a data frame

Appendix: Open Weather API- 3

[16]:
```
# Combine all vectors
weather_data_frame <- data.frame(weather=weather,
                                 visibility=visibility,
                                 temp=temp,
                                 temp_min=temp_min,
                                 temp_max=temp_max,
                                 pressure=pressure,
                                 humidity=humidity,
                                 wind_speed=wind_speed,
                                 wind_deg=wind_deg)
```

[17]:
```
# Check the generated data frame
print(weather_data_frame)
```

```
  weather visibility  temp temp_min temp_max pressure humidity wind_speed
1   Clear      10000 21.51    20.66    22.69     1007       78       2.06
  wind_deg
1      160
```

Appendix: Open Weather API- 4

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

Now you should be familiar with the usage of OpenWeather API. Next, you need to complete a task to get 5-day weather forecasts for a list of cities

TODO: Write a function to return a data frame containing 5-day weather forecasts for a list of cities

[191]:
```r
# Create some empty vectors to hold data temporarily

# City name column
city <- c()
# Weather column, rainy or cloudy, etc
weather <- c()
# Sky visibility column
visibility <- c()
# Current temperature column
temp <- c()
# Max temperature column
temp_min <- c()
# Min temperature column
temp_max <- c()
# Pressure column
pressure <- c()
# Humidity column
humidity <- c()
# Wind speed column
wind_speed <- c()
# Wind direction column
wind_deg <- c()
# Forecast timestamp
forecast_datetime <- c()
# Season column
# Note that for season, you can hard code a season value from levels Spring, Summer, Autumn, and Winter based on your current month.
season <- c()
```

Appendix: Open Weather API- 5

[231]:

```
# Get forecast data for a given city list
get_weather_forecaset_by_cities <- function(city_names){
    df <- data.frame()
    for (citi in city_names){
        your_api_key <- [redacted]
        # Forecast API URL
        forecast_url <- 'https://api.openweathermap.org/data/2.5/forecast'
        # Create query parameters
        forecast_query <- list(q = citi, appid = your_api_key, units="metric")

        # Make HTTP GET call for the given city
        forecast_response <- GET(forecast_url, query=forecast_query)
        json_result <- content(forecast_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_result$list

        # Loop the json result

        for(result in results) {
                city <- c(city, json_result$city$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)
              season <- c(season,"Summer")

        }

        # Add the R Lists into a data frame

    }
```

Appendix: Open Weather API- 6

```
        # 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)

                #print(length(weather))

    # 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`

[232]:
```
cities <- c("Seoul", "Washington, D.C.", "Paris", "Suzhou")
cities_weather_df <- get_weather_forecaset_by_cities(cities)
```

[233]:
```
# Write cities_weather_df to `cities_weather_forecast.csv`
write.csv(cities_weather_df, "cities_weather_forecast.csv", row.names=FALSE)
```

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

HTTP request in R

Appendix: Data Wrangling- Regex-1

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.

[2]:
```
dataset_list <- c('Bike_Data_Frame.csv', 'raw_seoul_bike_sharing.csv', 'cities_weather_forecast.csv', 'raw_worldcities.csv')
```

TODO: Write a `for` loop to iterate over the above datasets and convert their column names

[4]:
```
for (dataset_name in dataset_list){
    # Read dataset
    dataset <- read_csv(dataset_name, col_types = cols())
    # Standardized its columns:
    names(dataset) <- str_replace(str_replace(toupper(names(dataset))," ","_"),"\\.","_")
    # Convert all column names to uppercase

    # Replace any white space separators by underscores, using the str_replace_all function

    # Save the dataset

    write.csv(dataset, dataset_name, row.names=FALSE)
}
```

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

[5]:
```
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, col_types = cols())
    print(names(dataset))
}
```

Appendix: Data Wrangling- Regex-2

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

[5]:
```
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, col_types = cols())
    print(names(dataset))
}
```

```
 [1] "COUNTRY"         "CITY"            "NAME"            "SYSTEM"
 [5] "OPERATOR"        "LAUNCHED"        "DISCONTINUED"    "STATIONS"
 [9] "BICYCLES"        "DAILY_RIDERSHIP"
 [1] "DATE"                 "RENTED_BIKE_COUNT"     "HOUR"
 [4] "TEMPERATURE"          "HUMIDITY"              "WIND_SPEED"
 [7] "VISIBILITY"           "DEW_POINT_TEMPERATURE" "SOLAR_RADIATION"
[10] "RAINFALL"             "SNOWFALL"              "SEASONS"
[13] "HOLIDAY"              "FUNCTIONING_DAY"
 [1] "CITY"             "WEATHER"          "VISIBILITY"
 [4] "TEMP"             "TEMP_MIN"         "TEMP_MAX"
 [7] "PRESSURE"         "HUMIDITY"         "WIND_SPEED"
[10] "WIND_DEG"         "FORECAST_DATETIME" "SEASON"
 [1] "CITY"       "CITY_ASCII" "LAT"        "LNG"        "COUNTRY"
 [6] "ISO2"       "ISO3"       "ADMIN_NAME" "CAPITAL"    "POPULATION"
[11] "ID"
```

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.

[6]:
```
# First load the dataset
bike_sharing_df <- read_csv("Bike_Data_Frame.csv", col_types=cols())
```

[7]:
```
# Print its head
head(bike_sharing_df)
```

A tibble: 6 × 10

Appendix: Data Wrangling- Regex-3

[7]:
```
# Print its head
head(bike_sharing_df)
```

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):

- `COUNTRY` : Country name
- `CITY` : City name
- `SYSTEM` : Bike-sharing system name
- `BICYCLES` : Total number of bikes in the system

[52]:
```
# Select the four columns
sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)
```

Appendix: Data Wrangling- Regex-4

[52]:
```
# Select the four columns
sub_bike_sharing_df <- bike_sharing_df %>% select(COUNTRY, CITY, SYSTEM, BICYCLES)
```

Let's see the types of the selected columns

[46]:
```
bike_sharing_df %>%
    summarize_all(class) %>%
    gather(variable, class)
```

A tibble: 10 × 2

variable	class
<chr>	<chr>
COUNTRY	character
CITY	character
NAME	character
SYSTEM	character
OPERATOR	character
LAUNCHED	character
DISCONTINUED	character
STATIONS	character
BICYCLES	character
DAILY_RIDERSHIP	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.

Appendix: Data Wrangling- Regex-5

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.

[12]:
```
# 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)
```

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

[13]:
```
sub_bike_sharing_df %>%
    select(BICYCLES) %>%
    filter(find_character(BICYCLES)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

BICYCLES
<chr>
1790 (2019)[21]
4200 (2021)
4115[25]
7270 (regular) 2395 (electric)[38]
310[65]
500[75]
[78]
180[79]
600[82]
initially 800 (later 2500)

Appendix: Data Wrangling- Regex-6

[18]:
```
bike_sharing_df %>%
      select(STATIONS) %>%
      filter(find_character(STATIONS)) %>%
      slice(0:10)
```

A spec_tbl_df: 10 × 1

STATIONS <chr>
dockless
dockless
dockless
70 (2021)[20]
305 (2021)
initially 150
10 (22)
24[121]
dockless
350 & dockless

[19]:
```
bike_sharing_df %>%
      select(DAILY_RIDERSHIP) %>%
      filter(find_character(DAILY_RIDERSHIP)) %>%
      slice(0:10)
```

A spec_tbl_df: 10 × 1

DAILY_RIDERSHIP <chr>

Appendix: Data Wrangling- Regex-7

[19]:
```
bike_sharing_df %>%
      select(DAILY_RIDERSHIP) %>%
      filter(find_character(DAILY_RIDERSHIP)) %>%
      slice(0:10)
```

A spec_tbl_df: 10 × 1

DAILY_RIDERSHIP
<chr>
2800[18]
7010[23]
4,700[74][non-primary source needed]
376[97]
380[97]
675[97]
3640[97]
105000 (2017[111]
150000 [111]
28093[249]

As you can see, many rows have non-numeric characters, such as `32 (including 6 rollers) [162]` and `1000[253]`. This is actually very common for a table scraped from Wiki when no input validation is enforced.

Later, you will use regular expressions to clean them up.

Next, let's take a look at the other columns, namely `COUNTRY`, `CITY`, and `SYSTEM`, to see if they contain any undesired reference links, such as in `Melbourne[12]`.

Appendix: Data Wrangling- Regex-8

[55]:
```
# 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)
```

[65]:
```
# Check whether the COUNTRY column has any reference links
bike_sharing_df %>%
    select(COUNTRY) %>%
    filter(find_reference_pattern(COUNTRY)) %>%
    slice(0:10)
```

A spec_tbl_df: 0 × 1

COUNTRY
<chr>

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

[64]:
```
# Check whether the CITY column has any reference links
bike_sharing_df %>%
    select(CITY) %>%
    filter(find_reference_pattern(CITY)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

CITY

Appendix: Data Wrangling- Regex-9

[64]:
```
# Check whether the CITY column has any reference links
bike_sharing_df %>%
    select(CITY) %>%
    filter(find_reference_pattern(CITY)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

CITY
<chr>
Tirana[5]
Buenos Aires[6][7]
Mendoza[10]
Melbourne[12]
Melbourne[12]
Brisbane[14][15]
Lower Austria[16]
Different locations[19]
Brussels[24]
Namur[26]

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

[63]:
```
# Check whether the System column has any reference links
bike_sharing_df %>%
    select(SYSTEM) %>%
    filter(find_reference_pattern(SYSTEM)) %>%
    slice(0:10)
```

A spec_tbl_df: 8 × 1

Appendix: Data Wrangling-Regex-10

[63]:
```
# Check whether the System column has any reference links
bike_sharing_df %>%
    select(SYSTEM) %>%
    filter(find_reference_pattern(SYSTEM)) %>%
    slice(0:10)
```

A spec_tbl_df: 8 × 1

SYSTEM
<chr>
Serttel Brasil[8]
EasyBike[64]
4 Gen.[72]
3 Gen. SmooveKey[120]
3 Gen. Smoove[147][148][149][145]
3 Gen. Smoove[185]
3 Gen. Smoove[187]
3 Gen. Smoove[189]

[60]:
```
bike_sharing_df %>%
    select(NAME) %>%
    filter(find_reference_pattern(NAME)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

NAME
<chr>

Appendix: Data Wrangling-Regex-11

A spec_tbl_df: 10 × 1

NAME
<chr>
Mi Bici Tu Bici[11]
Citybike Wien[17]
Velo Antwerp[22]
Rekola[95]
BatumVelo[119]
Velocity [122]
Call a Bike [123]
LIDL-BIKES - Call a Bike [123]
Donkey Republic Berlin [124]
Mobike [125]

[62]:
```
bike_sharing_df %>%
    select(OPERATOR) %>%
    filter(find_reference_pattern(OPERATOR)) %>%
    slice(0:10)
```

A spec_tbl_df: 7 × 1

OPERATOR
<chr>
Bike In Baires Consortium.[9]
The Metropolitan Area of Aburra Valley[59]
Smovengo[113]

Appendix: Data Wrangling-Regex-12

The Metropolitan Area of Aburra Valley[59]
Smovengo[113]
Batumi Avtotransporti[121]
Freie Lastenräder [127]
DVB [134]
Russ Outdoor[211]

[67]:
```
bike_sharing_df %>%
    select(LAUNCHED) %>%
    filter(find_reference_pattern(LAUNCHED)) %>%
    slice(0:10)
```

A spec_tbl_df: 10 × 1

LAUNCHED
<chr>
2014 [66]
2015 [67]
2018 [68]
2015 [69]
2017 [70]
2018 [71]
2013 [73]
9 October 2016[76]
2014 [80]
2018 [81]

Appendix: Data Wrangling-Regex-13

2014 [80]
2018 [81]

[68]:
```
bike_sharing_df %>%
    select(DISCONTINUED) %>%
    filter(find_reference_pattern(DISCONTINUED)) %>%
    slice(0:10)
```

A spec_tbl_df: 8 × 1

DISCONTINUED
<chr>
30 November 2019[13]
2020[41]
2020[77]
October 2012 [85][86][87] Copenhagen municipality changed its mind and a new version was introduced in late 2013.[88]
2010 [93]
2019[94]
September 2020[281][282]
1997[326]

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.

Appendix: Data Wrangling-Regex-14

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`

[76]:
```
# remove reference link
remove_ref <- function(strings) {
    ref_pattern <- "\\[[A-z0-9]+\\]"
    # Replace all matched substrings with a white space using str_replace_all()
    result <- str_replace_all(strings,ref_pattern," ")
    # Trim the reslt if you want
    result <- trimws(result)
     return(result)
}
```

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

[106]:
```
result <- bike_sharing_df %>% mutate(CITY=remove_ref(CITY),SYSTEM=remove_ref(SYSTEM),BICYCLES=remove_ref(BICYCLES),OPERATOR=remove_ref(OPERATOR),
                          NAME=remove_ref(NAME),DISCONTINUED=remove_ref(DISCONTINUED),LAUNCHED=remove_ref(LAUNCHED),
                          STATIONS=remove_ref(STATIONS),DAILY_RIDERSHIP=remove_ref(DAILY_RIDERSHIP))
```

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

[105]:
```
 result %>%
    select(CITY, SYSTEM, BICYCLES, NAME, OPERATOR, DISCONTINUED, LAUNCHED, DAILY_RIDERSHIP, STATIONS) %>%
    filter(find_reference_pattern(CITY) | find_reference_pattern(SYSTEM) | find_reference_pattern(BICYCLES) | find_reference_pattern(NAME) |
           find_reference_pattern(OPERATOR) | find_reference_pattern(DISCONTINUED) | find_reference_pattern(LAUNCHED)  |
           find_reference_pattern(DAILY_RIDERSHIP) | find_reference_pattern(STATIONS))
```

TASK: Extract the numeric value using regular expressions

Appendix: Data Wrangling-Regex-15

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]`.

[90]:
```
# Extract the first number
extract_num <- function(columns){
    # Define a digital pattern
    digitals_pattern <- "[0-9]+"
    # Find the first match using str_extract
    result <- str_extract(columns,digitals_pattern)
    # Convert the result to numeric using the as.numeric() function
    result <- as.numeric(result)
}
```

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

[107]:
```
# Use the mutate() function on the BICYCLES column
result <- result %>% mutate(BICYCLES=extract_num(BICYCLES),STATIONS=extract_num(STATIONS),DAILY_RIDERSHIP=extract_num(DAILY_RIDERSHIP))
```

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

[108]:
```
summary(result$BICYCLES)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
    5.00    93.75   300.00  1958.09  1257.50 78000.00       84
```

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

[109]:
```
# Write dataset to `bike_sharing_systems.csv`
 write.csv(result,"bike_sharing_systems.csv",row.names=FALSE)
```

Appendix: Data Wrangling- Dplyr-1

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

[2]:
```
bike_sharing_df <- read_csv("raw_seoul_bike_sharing.csv", col_types = cols())
```

[ ]:
```
# Or you may read it from here again
# url <- "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-RP0321EN-SkillsNetwork/labs/datasets/raw_seoul_b
# Notice some column names in the raw datasets are not standalized if you haven't done them properly in the previous lab
```

First take a quick look at the dataset:

[42]:
```
summary(bike_sharing_df)
dim(bike_sharing_df)
```

```
     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
```

Appendix: Data Wrangling- Dplyr-2

```
    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!

Appendix: Data Wrangling- Dplyr-3

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

[36]:
```
# Drop rows with `RENTED_BIKE_COUNT` column == NA
bike_sharing_df <-  bike_sharing_df %>%
                    drop_na(c("RENTED_BIKE_COUNT"))
```

[9]:
```
# Print the dataset dimension again after those rows are dropped
summary(bike_sharing_df)
dim(bike_sharing_df)
```

```
     DATE           RENTED_BIKE_COUNT      HOUR        TEMPERATURE
 Length:8465        Min.   :   2.0    Min.   : 0.00   Min.   :-17.80
 Class :character   1st Qu.: 214.0    1st Qu.: 6.00   1st Qu.:  3.00
 Mode  :character   Median : 542.0    Median :12.00   Median : 13.40
                    Mean   : 729.2    Mean   :11.51   Mean   : 12.75
                    3rd Qu.:1084.0    3rd Qu.:18.00   3rd Qu.: 22.60
                    Max.   :3556.0    Max.   :23.00   Max.   : 39.40
                                                      NA's   :11
```

Appendix: Data Wrangling- Dplyr-4

[9]:
```
# Print the dataset dimension again after those rows are dropped
summary(bike_sharing_df)
dim(bike_sharing_df)
```

```
     DATE           RENTED_BIKE_COUNT      HOUR        TEMPERATURE
 Length:8465        Min.   :   2.0    Min.   : 0.00   Min.   :-17.80
 Class :character   1st Qu.: 214.0    1st Qu.: 6.00   1st Qu.:  3.00
 Mode  :character   Median : 542.0    Median :12.00   Median : 13.40
                    Mean   : 729.2    Mean   :11.51   Mean   : 12.75
                    3rd Qu.:1084.0    3rd Qu.:18.00   3rd Qu.: 22.60
                    Max.   :3556.0    Max.   :23.00   Max.   : 39.40
                                                      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.: 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
```

```
8465
14
```

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

Appendix: Data Wrangling- Dplyr-5

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.

[10]:
```
bike_sharing_df %>%
                filter(is.na(TEMPERATURE))
```

A tibble: 11 × 14

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	SNOWFA
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl
07/06/2018	3221	18	NA	57	2.7	1217	16.4	0.96	0.0	
12/06/2018	1246	14	NA	45	2.2	1961	12.7	1.39	0.0	
13/06/2018	2664	17	NA	57	3.3	919	16.4	0.87	0.0	
17/06/2018	2330	17	NA	58	3.3	865	16.7	0.66	0.0	
20/06/2018	2741	19	NA	61	2.7	1236	17.5	0.60	0.0	
30/06/2018	1144	13	NA	87	1.7	390	23.2	0.71	3.5	

Appendix: Data Wrangling- Dplyr-6

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

A tibble: 11 × 14

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	SNOWFA
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<db
07/06/2018	3221	18	NA	57	2.7	1217	16.4	0.96	0.0	
12/06/2018	1246	14	NA	45	2.2	1961	12.7	1.39	0.0	
13/06/2018	2664	17	NA	57	3.3	919	16.4	0.87	0.0	
17/06/2018	2330	17	NA	58	3.3	865	16.7	0.66	0.0	
20/06/2018	2741	19	NA	61	2.7	1236	17.5	0.60	0.0	
30/06/2018	1144	13	NA	87	1.7	390	23.2	0.71	3.5	
05/07/2018	827	10	NA	75	1.1	1028	20.8	1.22	0.0	
11/07/2018	634	9	NA	96	0.6	450	24.9	0.41	0.0	
12/07/2018	593	6	NA	93	1.1	852	24.3	0.01	0.0	
21/07/2018	347	4	NA	77	1.2	1203	21.2	0.00	0.0	
21/08/2018	1277	23	NA	75	0.1	1892	20.8	0.00	0.0	

Appendix: Data Wrangling- Dplyr-7

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.

[37]:
```
# Calculate the summer average temperature
summer_mean <-  bike_sharing_df %>%
                group_by(SEASONS) %>%
                summarize(mean=mean(TEMPERATURE, na.rm =TRUE)) %>%
                filter(SEASONS=="Summer")
summer_mean <- as.numeric(summer_mean[2])
summer_mean
```

26.5877105143377

[38]:
```
# Impute missing values for TEMPERATURE column with summer average temperature
bike_sharing_df <- bike_sharing_df %>%
          mutate(TEMPERATURE=ifelse(is.na(TEMPERATURE),summer_mean,TEMPERATURE))
```

[13]:
```
# Print the summary of the dataset again to make sure no missing values in all columns
summary(bike_sharing_df)
```

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

Appendix: Data Wrangling- Dplyr-8

```
 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
```

[49]:
```
# Save the dataset as `seoul_bike_sharing.csv`
write.csv(bike_sharing_df,'seoul_bike_sharing.csv',row.names=FALSE)
```

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:

[3]:
```
# Using mutate() function to convert HOUR column into character type
bike_sharing_df <- bike_sharing_df %>% mutate(HOUR=as.character(HOUR))
```

`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.

Appendix: Data Wrangling- Dplyr-9

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.

[10]:
```
# Convert SEASONS, HOLIDAY, FUNCTIONING_DAY, and HOUR columns into indicator columns.
bike_df <- bike_sharing_df %>%
           mutate_if(sapply(bike_sharing_df,is.character), as.factor)
```

[11]:
```
install.packages("fastDummies")
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
```

[12]:
```
library(fastDummies)
```

[40]:
```
bike_df <- dummy_cols(bike_df,"HOUR")
bike_df <- dummy_cols(bike_df,"SEASONS")
bike_df <- dummy_cols(bike_df,"HOLIDAY")
```

[41]:
```
# Print the dataset summary again to make sure the indicator columns are created properly
summary(bike_df)
```

```
         DATE       RENTED_BIKE_COUNT      HOUR        TEMPERATURE
 01/01/2018:  24   Min.   :   2.0    Min.   : 0.00   Min.   :-17.80
 01/02/2018:  24   1st Qu.: 214.0    1st Qu.: 6.00   1st Qu.:  3.00
 01/03/2018:  24   Median : 542.0    Median :12.00   Median : 13.50
 01/04/2018:  24   Mean   : 729.2    Mean   :11.51   Mean   : 12.77
 01/05/2018:  24   3rd Qu.:1084.0    3rd Qu.:18.00   3rd Qu.: 22.70
 01/06/2018:  24   Max.   :3556.0    Max.   :23.00   Max.   : 39.40
 (Other)   :8321
    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
```

Appendix: Data Wrangling-Dplyr-10

```
    HUMIDITY       WIND_SPEED      VISIBILITY   DEW_POINT_TEMPERATURE
Min.   : 0.00   Min.   :0.000   Min.   :  27   Min.   :-30.600
1st Qu.:42.00   1st Qu.:0.900   1st Qu.: 935   1st Qu.: -5.100
Median :57.00   Median :1.500   Median :1690   Median :  4.700
Mean   :58.15   Mean   :1.726   Mean   :1434   Mean   :  3.945
3rd Qu.:74.00   3rd Qu.:2.300   3rd Qu.:2000   3rd Qu.: 15.200
Max.   :98.00   Max.   :7.400   Max.   :2000   Max.   : 27.200

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

      HOLIDAY     FUNCTIONING_DAY     HOUR_0            HOUR_1
Holiday   : 408   Yes:8465        Min.   :0.00000   Min.   :0.00000
No Holiday:8057                   1st Qu.:0.00000   1st Qu.:0.00000
                                  Median :0.00000   Median :0.00000
                                  Mean   :0.04158   Mean   :0.04158
                                  3rd Qu.:0.00000   3rd Qu.:0.00000
                                  Max.   :1.00000   Max.   :1.00000

    HOUR_2            HOUR_3            HOUR_4            HOUR_5
Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
Median :0.00000   Median :0.00000   Median :0.00000   Median :0.00000
Mean   :0.04158   Mean   :0.04158   Mean   :0.04158   Mean   :0.04158
3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000
Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000

    HOUR_6            HOUR_7           HOUR_8           HOUR_9
Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000
Mean   :0.04158   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417
3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Appendix: Data Wrangling-Dplyr-11

```
    HOUR_6            HOUR_7           HOUR_8           HOUR_9      
Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000  
Mean   :0.04158   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417  
3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

   HOUR_10          HOUR_11          HOUR_12          HOUR_13      
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   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417  
3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

   HOUR_14          HOUR_15          HOUR_16          HOUR_17      
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   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417  
3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

   HOUR_18          HOUR_19          HOUR_20          HOUR_21      
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   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417  
3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

   HOUR_22          HOUR_23       SEASONS_Autumn   SEASONS_Spring  
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   :0.0417   Mean   :0.0417   Mean   :0.2288   Mean   :0.2552  
3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:1.0000  
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
```

Appendix: Data Wrangling-Dplyr-12

```
    HOUR_22          HOUR_23       SEASONS_Autumn   SEASONS_Spring
 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   :0.0417   Mean   :0.0417   Mean   :0.2288   Mean   :0.2552
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:1.0000
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000

 SEASONS_Summer   SEASONS_Winter   HOLIDAY_Holiday  HOLIDAY_No 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.:1.0000
 Median :0.0000   Median :0.0000   Median :0.0000   Median :1.0000
 Mean   :0.2608   Mean   :0.2552   Mean   :0.0482   Mean   :0.9518
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:1.0000
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

[75]:
```
# Save the dataset as `seoul_bike_sharing_converted.csv`
write_csv(bike_df, "seoul_bike_sharing_converted.csv")
```

[ ]:

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.

Appendix: Data Wrangling-Dplyr-13

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

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

[17]:
```
bike_df %>%
  mutate(TEMPERATURE = (TEMPERATURE - min(TEMPERATURE)) / (max(TEMPERATURE) - min(TEMPERATURE)),
         RAINFALL= (RAINFALL - min(RAINFALL)) / (max(RAINFALL) - min(RAINFALL)),
         WIND_SPEED= (WIND_SPEED - min(WIND_SPEED)) / (max(WIND_SPEED) - min(WIND_SPEED)),
         SNOWFALL= (SNOWFALL - min(SNOWFALL)) / (max(SNOWFALL) - min(SNOWFALL)),
         HUMIDITY= (HUMIDITY - min(HUMIDITY)) / (max(HUMIDITY) - min(HUMIDITY)),
         VISIBILITY= (VISIBILITY - min(VISIBILITY)) / (max(VISIBILITY) - min(VISIBILITY)),
         RENTED_BIKE_COUNT= (RENTED_BIKE_COUNT - min(RENTED_BIKE_COUNT)) / (max(RENTED_BIKE_COUNT) - min(RENTED_BIKE_COUNT)),
         DEW_POINT_TEMPERATURE= (DEW_POINT_TEMPERATURE - min(DEW_POINT_TEMPERATURE)) / (max(DEW_POINT_TEMPERATURE) - min(DEW_POINT_TEMPERATURE)),
         SOLAR_RADIATION= (SOLAR_RADIATION - min(SOLAR_RADIATION)) / (max(SOLAR_RADIATION) - min(SOLAR_RADIATION))) %>% head()
```

A tibble: 6 × 44

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	⋯	HOUI
<fct>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	⋯	<ir
01/12/2017	0.07090602	0	0.2202797	0.3775510	0.2972973	1	0.2249135	0	0	⋯	
01/12/2017	0.05683737	1	0.2150350	0.3877551	0.1081081	1	0.2249135	0	0	⋯	
01/12/2017	0.04811480	2	0.2062937	0.3979592	0.1351351	1	0.2231834	0	0	⋯	
01/12/2017	0.02954418	3	0.2027972	0.4081633	0.1216216	1	0.2249135	0	0	⋯	
01/12/2017	0.02138436	4	0.2062937	0.3673469	0.3108108	1	0.2076125	0	0	⋯	
01/12/2017	0.02757456	5	0.1993007	0.3775510	0.2027027	1	0.2058824	0	0	⋯	

[31]:
```
# Use the `mutate()` function to apply min-max normalization on columns
list_df<-list("RENTED_BIKE_COUNT", "TEMPERATURE", "HUMIDITY", "WIND_SPEED", "VISIBILITY", "DEW_POINT_TEMPERATURE", "SOLAR_RADIATION", "RAINFALL",
         "SNOWFALL")
```

Appendix: Data Wrangling-Dplyr-14

[18]:
```
# Use the `mutate()` function to apply min-max normalization on columns
 list_df<-list("RENTED_BIKE_COUNT", "TEMPERATURE", "HUMIDITY", "WIND_SPEED", "VISIBILITY", "DEW_POINT_TEMPERATURE", "SOLAR_RADIATION", "RAINFALL",
          "SNOWFALL")

for (item in list_df) {
      mini=min(bike_df[item])
      maxi=max(bike_df[item])
      bike_df[item]=(bike_df[item]-mini)/(maxi-mini)
    }
head(bike_df)
```

A tibble: 6 × 44

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	...	HOUI
<fct>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	...	<ir
01/12/2017	0.07090602	0	0.2202797	0.3775510	0.2972973	1	0.2249135	0	0	...	
01/12/2017	0.05683737	1	0.2150350	0.3877551	0.1081081	1	0.2249135	0	0	...	
01/12/2017	0.04811480	2	0.2062937	0.3979592	0.1351351	1	0.2231834	0	0	...	
01/12/2017	0.02954418	3	0.2027972	0.4081633	0.1216216	1	0.2249135	0	0	...	
01/12/2017	0.02138436	4	0.2062937	0.3673469	0.3108108	1	0.2076125	0	0	...	
01/12/2017	0.02757456	5	0.1993007	0.3775510	0.2027027	1	0.2058824	0	0	...	

[21]:
```
install.packages("caret")
library(caret)
```

```
also installing the dependencies ‘proxy’, ‘e1071’

Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
Loading required package: lattice

Attaching package: ‘caret’
```

Appendix: Data Wrangling-Dplyr-15

```
Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift
```

[22]:
```
process <- preProcess(bike_df, method=c("range"))

norm_scale <- predict(process, bike_df)

head(norm_scale)
```

A tibble: 6 × 44

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	···	HOUI
<fct>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	···	<dl
01/12/2017	0.07090602	0	0.2202797	0.3775510	0.2972973	1	0.2249135	0	0	···	
01/12/2017	0.05683737	1	0.2150350	0.3877551	0.1081081	1	0.2249135	0	0	···	
01/12/2017	0.04811480	2	0.2062937	0.3979592	0.1351351	1	0.2231834	0	0	···	
01/12/2017	0.02954418	3	0.2027972	0.4081633	0.1216216	1	0.2249135	0	0	···	
01/12/2017	0.02138436	4	0.2062937	0.3673469	0.3108108	1	0.2076125	0	0	···	
01/12/2017	0.02757456	5	0.1993007	0.3775510	0.2027027	1	0.2058824	0	0	···	

[32]:
```
# Print the summary of the dataset again to make sure the numeric columns range between 0 and 1
summary(bike_df)
```

```
      DATE      RENTED_BIKE_COUNT      HOUR       TEMPERATURE
 01/01/2018:  24   Min.   :0.00000   10     : 353   Min.   :0.0000
 01/02/2018:  24   1st Qu.:0.05965   11     : 353   1st Qu.:0.3636
 01/03/2018:  24   Median :0.15194   12     : 353   Median :0.5472
```

Appendix: Data Wrangling-Dplyr-16

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

```
        DATE      RENTED_BIKE_COUNT      HOUR       TEMPERATURE
 01/01/2018:  24   Min.   :0.00000   10     : 353   Min.   :0.0000
 01/02/2018:  24   1st Qu.:0.05965   11     : 353   1st Qu.:0.3636
 01/03/2018:  24   Median :0.15194   12     : 353   Median :0.5472
 01/04/2018:  24   Mean   :0.20460   13     : 353   Mean   :0.5345
 01/05/2018:  24   3rd Qu.:0.30445   14     : 353   3rd Qu.:0.7080
 01/06/2018:  24   Max.   :1.00000   15     : 353   Max.   :1.0000
 (Other)   :8321                     (Other):6347
    HUMIDITY        WIND_SPEED       VISIBILITY     DEW_POINT_TEMPERATURE
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.4286   1st Qu.:0.1216   1st Qu.:0.4602   1st Qu.:0.4412
 Median :0.5816   Median :0.2027   Median :0.8429   Median :0.6107
 Mean   :0.5933   Mean   :0.2332   Mean   :0.7131   Mean   :0.5977
 3rd Qu.:0.7551   3rd Qu.:0.3108   3rd Qu.:1.0000   3rd Qu.:0.7924
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000

 SOLAR_RADIATION       RAINFALL           SNOWFALL          SEASONS
 Min.   :0.000000   Min.   :0.000000   Min.   :0.000000   Autumn:1937
 1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.000000   Spring:2160
 Median :0.002841   Median :0.000000   Median :0.000000   Summer:2208
 Mean   :0.161326   Mean   :0.004261   Mean   :0.008828   Winter:2160
 3rd Qu.:0.264205   3rd Qu.:0.000000   3rd Qu.:0.000000
 Max.   :1.000000   Max.   :1.000000   Max.   :1.000000

       HOLIDAY     FUNCTIONING_DAY     HOUR_0            HOUR_1
 Holiday   : 408   Yes:8465        Min.   :0.00000   Min.   :0.00000
 No Holiday:8057                   1st Qu.:0.00000   1st Qu.:0.00000
                                   Median :0.00000   Median :0.00000
                                   Mean   :0.04158   Mean   :0.04158
                                   3rd Qu.:0.00000   3rd Qu.:0.00000
                                   Max.   :1.00000   Max.   :1.00000

    HOUR_10          HOUR_11          HOUR_12          HOUR_13
 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   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
```

Appendix: Data Wrangling-Dplyr-17

```
    HOUR_10          HOUR_11          HOUR_12          HOUR_13      
 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   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417  
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

    HOUR_14          HOUR_15          HOUR_16          HOUR_17      
 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   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.0417  
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

    HOUR_18          HOUR_19           HOUR_2           HOUR_20      
 Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000  
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000  
 Median :0.0000   Median :0.0000   Median :0.00000   Median :0.0000  
 Mean   :0.0417   Mean   :0.0417   Mean   :0.04158   Mean   :0.0417  
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.0000  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000  

    HOUR_21          HOUR_22          HOUR_23           HOUR_3       
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000  
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000  
 Median :0.0000   Median :0.0000   Median :0.0000   Median :0.00000  
 Mean   :0.0417   Mean   :0.0417   Mean   :0.0417   Mean   :0.04158  
 3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000  

     HOUR_4           HOUR_5           HOUR_6           HOUR_7       
 Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.0000  
 1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000  
 Median :0.00000   Median :0.00000   Median :0.00000   Median :0.0000  
 Mean   :0.04158   Mean   :0.04158   Mean   :0.04158   Mean   :0.0417  
 3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.0000  
 Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.0000  
```

Appendix: Data Wrangling-Dplyr-18

```
    HOUR_4           HOUR_5           HOUR_6           HOUR_7      
Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.0000  
1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000  
Median :0.00000   Median :0.00000   Median :0.00000   Median :0.0000  
Mean   :0.04158   Mean   :0.04158   Mean   :0.04158   Mean   :0.0417  
3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.0000  
Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.0000  

    HOUR_8           HOUR_9       SEASONS_Autumn   SEASONS_Spring  
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   :0.0417   Mean   :0.0417   Mean   :0.2288   Mean   :0.2552  
3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:1.0000  
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  

SEASONS_Summer   SEASONS_Winter   HOLIDAY_Holiday  HOLIDAY_No 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.:1.0000  
Median :0.0000   Median :0.0000   Median :0.0000   Median :1.0000  
Mean   :0.2608   Mean   :0.2552   Mean   :0.0482   Mean   :0.9518  
3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:1.0000  
Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000  
```

[33]:
```
# Save the dataset as `seoul_bike_sharing_converted_normalized.csv`
write_csv(bike_df, "seoul_bike_sharing_converted_normalized.csv")
```

Appendix: Data Wrangling-Dplyr-19

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:

[23]:
```r
# 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, col_types = cols())
    # Standardized its columns:
    # Convert all columns names to uppercase
    names(dataset) <- toupper(names(dataset))
    # Replace any white space separators by underscore, using str_replace_all function
    names(dataset) <- str_replace_all(names(dataset), " ", "_")
    # Save the dataset back
    write.csv(dataset, dataset_name, row.names=FALSE)
}
```

Appendix: EDA- SQL- 1

[3]:
```
conn <- dbConnect(RSQLite::SQLite(), "FinalDB.sqlite")
```

[6]:
```
bike_df <- read.csv('bike_sharing_systems.csv')
forecast_df <- read.csv('cities_weather_forecast.csv')
cities_df <- read.csv('raw_worldcities.csv')
seoul_bike <- read.csv('seoul_bike_sharing_converted_normalized.csv')
```

[11]:
```
dbWriteTable(conn, "WORLD_CITIES", cities_df, overwrite = TRUE, header = TRUE)
dbWriteTable(conn, "BIKE_SHARING_SYSTEMS", bike_df, overwrite = TRUE, header = TRUE)
dbWriteTable(conn, "CITIES_WEATHER_FORECAST", forecast_df, overwrite = TRUE, header = TRUE)
dbWriteTable(conn, "SEOUL_BIKE_SHARING", seoul_bike, overwrite = TRUE, header = TRUE)
```

Download the following csv files:

- WORLD_CITIES
- BIKE_SHARING_SYSTEMS
- CITIES_WEATHER_FORECAST
- SEOUL_BIKE_SHARING

and load the csv's into 4 tables as mentioned below

- SEOUL_BIKE_SHARING
- CITIES_WEATHER_FORECAST
- BIKE_SHARING_SYSTEMS
- WORLD_CITIES

> Hint : Use the read_csv() function and dbWriteTable() functions

Appendix: EDA- SQL- 2

Task 1 - Record Count

Determine how many records are in the seoul_bike_sharing dataset.

Solution 1

[12]:
```
# provide your solution here
dbGetQuery(conn, "SELECT COUNT() FROM SEOUL_BIKE_SHARING")
```

A data.frame: 1 × 1

COUNT()
<int>
8465

Task 2 - Operational Hours

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

Solution 2

[14]:
```
# provide your solution here
dbGetQuery(conn,"SELECT COUNT(HOUR) FROM SEOUL_BIKE_SHARING WHERE RENTED_BIKE_COUNT != 0")
```

A data.frame: 1 × 1

COUNT(HOUR)
<int>
8462

Appendix: EDA- SQL- 3

Task 3 - Weather Outlook

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

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

Solution 3

16]:
```
# provide your solution here
dbGetQuery(conn,"SELECT * FROM CITIES_WEATHER_FORECAST WHERE FORECAST_DATETIME = '2022-08-26 03:00:00' AND city ='Seoul'")
```

A data.frame: 1 × 12

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>	<chr>	<chr>
Seoul	Clouds	10000	27.67	27.67	27.67	1006	51	1.06	233	2022-08-26 03:00:00	Summer

Appendix: EDA- SQL- 4

Task 4 - Seasons

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

Solution 4

[17]:
```
# provide your solution here
dbGetQuery(conn,"SELECT DISTINCT SEASONS FROM SEOUL_BIKE_SHARING")
```

A data.frame: 4 × 1

SEASONS <chr>
Winter
Spring
Summer
Autumn

Appendix: EDA- SQL- 5

Task 5 - Date Range

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

Solution 5

[34]:
```
# provide your solution here
dbGetQuery(conn,"SELECT DATE AS FIRST_DATE FROM SEOUL_BIKE_SHARING LIMIT 1")
dbGetQuery(conn,"SELECT DATE AS LAST_DATE FROM SEOUL_BIKE_SHARING LIMIT 1 OFFSET 8464")
```

A data.frame:

1 × 1

FIRST_DATE
<chr>
01/12/2017

A data.frame:

1 × 1

LAST_DATE
<chr>
30/11/2018

Appendix: EDA- SQL- 6

Task 6 - Subquery - 'all-time high'

determine which date and hour had the most bike rentals.

Solution 6

[35]:
```
# provide your solution here
dbGetQuery(conn, "SELECT DATE, HOUR FROM SEOUL_BIKE_SHARING WHERE RENTED_BIKE_COUNT=(SELECT MAX(RENTED_BIKE_COUNT) FROM SEOUL_BIKE_SHARING)")
```

A data.frame: 1 × 2

DATE	HOUR
<chr>	<int>
19/06/2018	18

Appendix: EDA- SQL- 7

Task 7 - Hourly popularity and temperature by season

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

Solution 7

[42]:
```
seoul_bike_denorm <- read.csv("seoul_bike_sharing_converted.csv")
```

[43]:
```
dbWriteTable(conn,"SEOUL_BIKE_SHARING_DENORM",seoul_bike_denorm, overwrite = TRUE, header = TRUE)
```

[62]:
```
# provide your solution here
dbGetQuery(conn,"SELECT SEASONS, avg(TEMPERATURE) AS AVERAGE_TEMPERATURE, avg(RENTED_BIKE_COUNT)
                 AS AVERAGE_BIKE_RENTALS FROM SEOUL_BIKE_SHARING_DENORM GROUP BY SEASONS")
dbGetQuery(conn,"SELECT * FROM SEOUL_BIKE_SHARING_DENORM ORDER BY RENTED_BIKE_COUNT DESC LIMIT 10")
```

A data.frame: 4 × 3

SEASONS	AVERAGE_TEMPERATURE	AVERAGE_BIKE_RENTALS
<chr>	<dbl>	<dbl>
Autumn	13.821580	924.1105
Spring	13.021685	746.2542
Summer	26.587711	1034.0734
Winter	-2.540463	225.5412

A data.frame: 10

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	⋯	HOUI
<chr>	<int>	<int>	<dbl>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	⋯	<ir
19/06/2018	3556	18	24.1	57	2.9	1301	15.0	0.56	0	⋯	
21/06/2018	3418	18	27.8	43	3.0	1933	14.0	1.35	0	⋯	

Appendix: EDA- SQL- 8

```
dbGetQuery(conn,"SELECT * FROM SEOUL_BIKE_SHARING_DENORM ORDER BY RENTED_BIKE_COUNT DESC LIMIT 10")
```

A data.frame: 4 × 3

SEASONS	AVERAGE_TEMPERATURE	AVERAGE_BIKE_RENTALS
<chr>	<dbl>	<dbl>
Autumn	13.821580	924.1105
Spring	13.021685	746.2542
Summer	26.587711	1034.0734
Winter	-2.540463	225.5412

A data.frame: 10

DATE	RENTED_BIKE_COUNT	HOUR	TEMPERATURE	HUMIDITY	WIND_SPEED	VISIBILITY	DEW_POINT_TEMPERATURE	SOLAR_RADIATION	RAINFALL	⋯	HOUI
<chr>	<int>	<int>	<dbl>	<int>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	⋯	<ir
19/06/2018	3556	18	24.1	57	2.9	1301	15.0	0.56	0	⋯	
21/06/2018	3418	18	27.8	43	3.0	1933	14.0	1.35	0	⋯	
12/06/2018	3404	18	24.9	53	3.6	2000	14.6	1.28	0	⋯	
20/06/2018	3384	18	27.0	55	3.1	1246	17.1	1.26	0	⋯	
04/06/2018	3380	18	24.4	48	1.9	1998	12.6	0.56	0	⋯	
22/06/2018	3365	18	29.3	27	3.4	1977	8.3	1.24	0	⋯	
08/06/2018	3309	18	26.2	54	2.2	1183	16.1	0.88	0	⋯	
10/09/2018	3298	18	25.9	42	1.1	2000	11.9	0.48	0	⋯	
17/09/2018	3277	18	25.3	56	2.8	1992	15.8	0.54	0	⋯	
12/09/2018	3256	18	27.0	44	1.4	2000	13.6	0.62	0	⋯	

Appendix: EDA- SQL- 9

Task 8 - Rental Seasonality

Find the average hourly bike count during each season.

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

> Hint : Use the SQRT(AVG(col*col) - AVG(col)*AVG(col)) function where col refers to your column name for finding the standard deviation

Solution 8

[49]:
```
# provide your solution here
dbGetQuery(conn,"SELECT SEASONS, avg(RENTED_BIKE_COUNT) AS AVERAGE_BIKE_RENTALS, min(RENTED_BIKE_COUNT) AS MINIMUM_BIKE_RENTALS, max(RENTED_BIKE_COUNT)
                 AS MAXIMUM_BIKE_RENTALS, sqrt((avg(RENTED_BIKE_COUNT*RENTED_BIKE_COUNT))-(avg(RENTED_BIKE_COUNT)*avg(RENTED_BIKE_COUNT))) AS
                 STANDARD_DEVIATION_BIKE_RENTALS
                 FROM SEOUL_BIKE_SHARING_DENORM
                 GROUP BY SEASONS")
```

A data.frame: 4 × 5

SEASONS	AVERAGE_BIKE_RENTALS	MINIMUM_BIKE_RENTALS	MAXIMUM_BIKE_RENTALS	STANDARD_DEVIATION_BIKE_RENTALS
<chr>	<dbl>	<int>	<int>	<dbl>
Autumn	924.1105	2	3298	617.3885
Spring	746.2542	2	3251	618.5247
Summer	1034.0734	9	3556	690.0884
Winter	225.5412	3	937	150.3374

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

Appendix: EDA- SQL- 10

Task 9 - Weather Seasonality

Consider the weather over each season. On average, what were the TEMPERATURE, HUMIDITY, WIND_SPEED, VISIBILITY, DEW_POINT_TEMPERATURE, SOLAR_RADIATION, RAINFALL, and SNOWFALL per season?

Include the average bike count as well , and rank the results by average bike count so you can see if it is correlated with the weather at all.

Solution 9

[53]:
```
# provide your solution here
average_weather <-dbGetQuery(conn,"SELECT SEASONS, avg(TEMPERATURE), avg(HUMIDITY),
                 avg(VISIBILITY),
                 avg(WIND_SPEED), avg(DEW_POINT_TEMPERATURE),
                 avg(SOLAR_RADIATION), avg(RAINFALL), avg(SNOWFALL),
                 avg(RENTED_BIKE_COUNT) AS AVERAGE_BIKE_RENTALS
                 FROM SEOUL_BIKE_SHARING_DENORM
                 GROUP BY SEASONS
                 ORDER BY AVERAGE_BIKE_RENTALS DESC")

average_weather$RANK <- rank(-average_weather$AVERAGE_BIKE_RENTALS, ties.method = "min")
average_weather
```

A data.frame: 4 × 11

SEASONS	avg(TEMPERATURE)	avg(HUMIDITY)	avg(VISIBILITY)	avg(WIND_SPEED)	avg(DEW_POINT_TEMPERATURE)	avg(SOLAR_RADIATION)	avg(RAINFALL)	avg(SN
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
Summer	26.587711	64.98143	1501.745	1.609420	18.750136	0.7612545	0.25348732	0
Autumn	13.821580	59.04491	1558.174	1.492101	5.150594	0.5227827	0.11765617	0
Spring	13.021685	58.75833	1240.912	1.857778	4.091389	0.6803009	0.18694444	0
Winter	-2.540463	49.74491	1445.987	1.922685	-12.416667	0.2981806	0.03282407	0

Appendix: EDA- SQL- 11

Task 10 - Total Bike Count and City Info for Seoul

Use an implicit join across the WORLD_CITIES and the BIKE_SHARING_SYSTEMS tables to determine the total number of bikes avaialble in Seoul, plus the following city information about Seoul: CITY, COUNTRY, LAT, LON, POPULATION, in a single view.

Notice that in this case, the CITY column will work for the WORLD_CITIES table, but in general you would have to use the CITY_ASCII column.

Solution 10

[58]:
```
# provide your solution here
dbGetQuery(conn, "SELECT b.CITY, w.COUNTRY, w.LAT, w.LNG, w.POPULATION, b.BICYCLES FROM WORLD_CITIES w, BIKE_SHARING_SYSTEMS b
                  WHERE w.CITY=b.CITY AND w.CITY='Seoul'")
```

A data.frame: 1 × 6

CITY	COUNTRY	LAT	LNG	POPULATION	BICYCLES
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>
Seoul	Korea, South	37.5833	127	21794000	37500

Appendix: EDA- SQL- 12

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

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

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

Solution 11

[59]:
```
# provide your solution here
dbGetQuery(conn, "SELECT b.CITY, w.COUNTRY, w.LAT, w.LNG, w.POPULATION, b.BICYCLES FROM WORLD_CITIES w, BIKE_SHARING_SYSTEMS b
                  WHERE w.CITY=b.CITY AND b.BICYCLES BETWEEN 15000 AND 20000")
```

A data.frame: 5 × 6

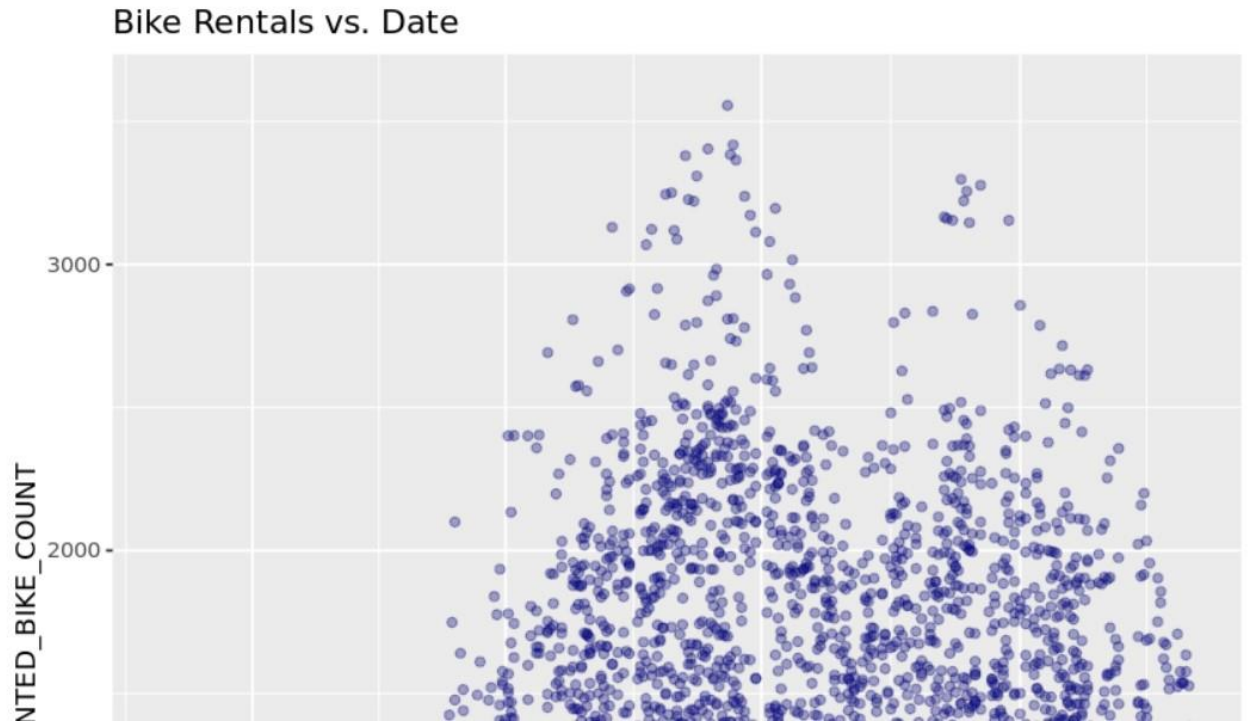
CITY	COUNTRY	LAT	LNG	POPULATION	BICYCLES
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<int>
Beijing	China	39.9050	116.3914	19433000	16000
Ningbo	China	29.8750	121.5492	7639000	15000
Shanghai	China	31.1667	121.4667	22120000	19165
Weifang	China	36.7167	119.1000	9373000	20000
Zhuzhou	China	27.8407	113.1469	3855609	20000

[63]:
```
close(conn)
```

Appendix:EDA-Data Visualization-1

Solution 10

```
•[24]: # provide your solution here
ggplot(seoul_bike_df, aes(x=DATE, y=RENTED_BIKE_COUNT)) +  geom_jitter(alpha=1/20)+ geom_point(alpha=0.3, color="navyblue") +
                    ggtitle("Bike Rentals vs. Date")
```

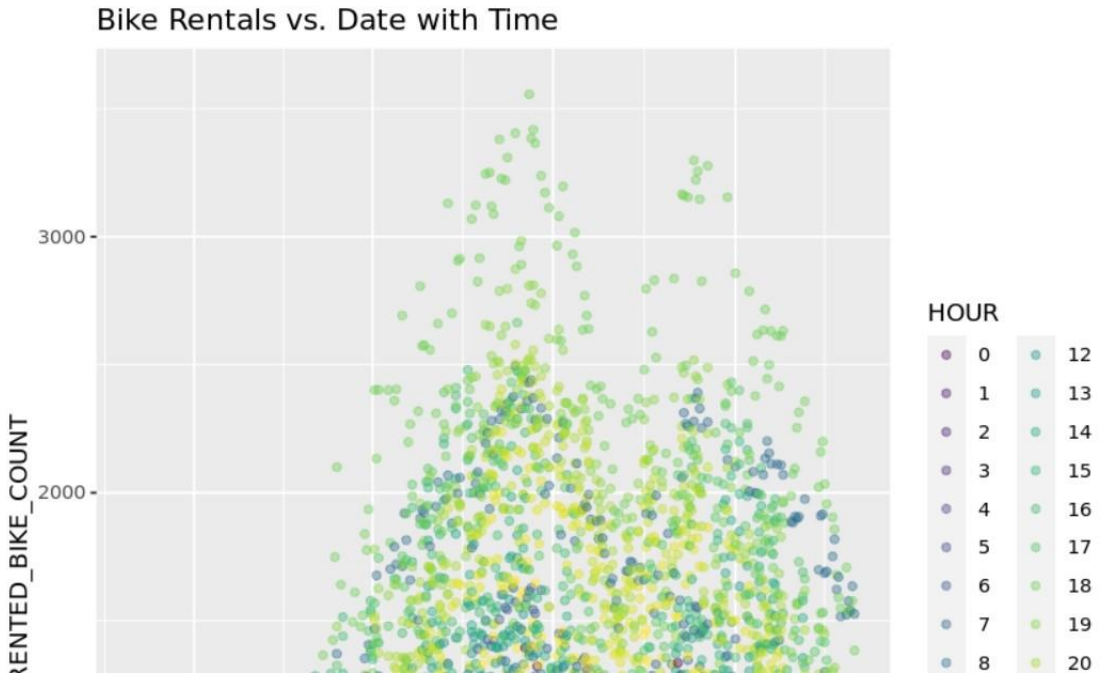


Appendix:EDA-Data Visualization-2

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

Solution 11

[17]:
```
# provide your solution here
ggplot(seoul_bike_df, aes(x=DATE, y=RENTED_BIKE_COUNT, colour=HOUR)) + geom_point(alpha=0.4) +ggtitle("Bike Rentals vs. Date with Time")
```



Appendix:EDA-Data Visualization-3

Distributions

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

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

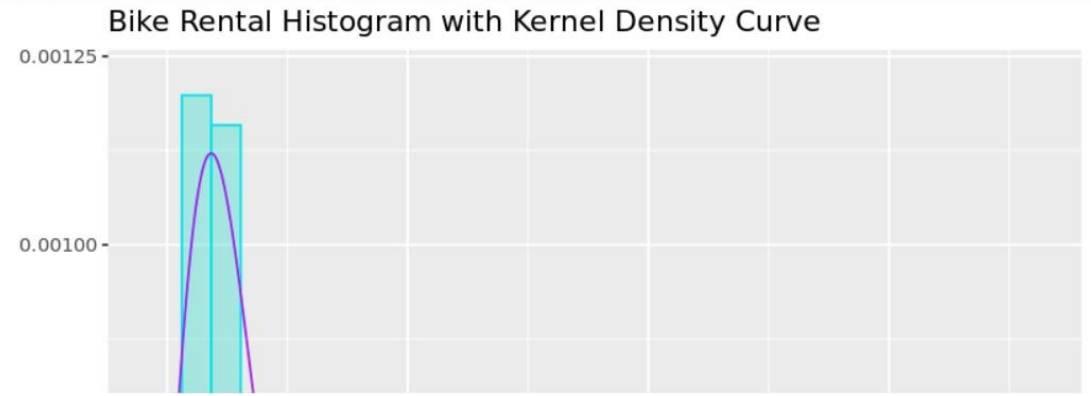
▶ Click here for a hint

▶ Click here for another hint

Solution 12

[25]:
```
# provide your solution here
ggplot(seoul_bike_df, aes(x=RENTED_BIKE_COUNT)) + geom_histogram(aes(y=..density..),alpha=0.4,color="turquoise2",fill="turquoise") +
                    geom_density(color="purple") + ggtitle("Bike Rental Histogram with Kernel Density Curve")
```

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



Appendix:EDA-Data Visualization-4

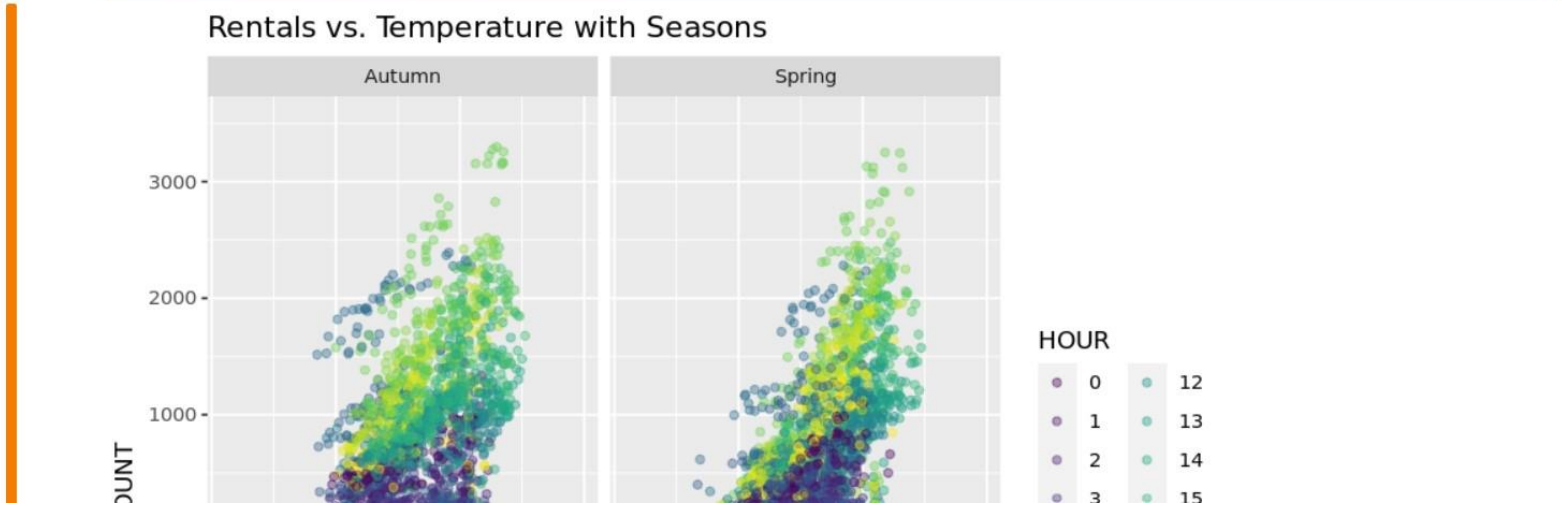
Correlation between two variables (scatter plot)

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

Start with `RENTED_BIKE_COUNT` vs. `TEMPERATURE`, then generate four plots corresponding to the `SEASONS` by adding a `facet_wrap()` layer. Also, make use of colour and opacity to emphasize any patterns that emerge. Use `HOUR` as the color.

Solution 13

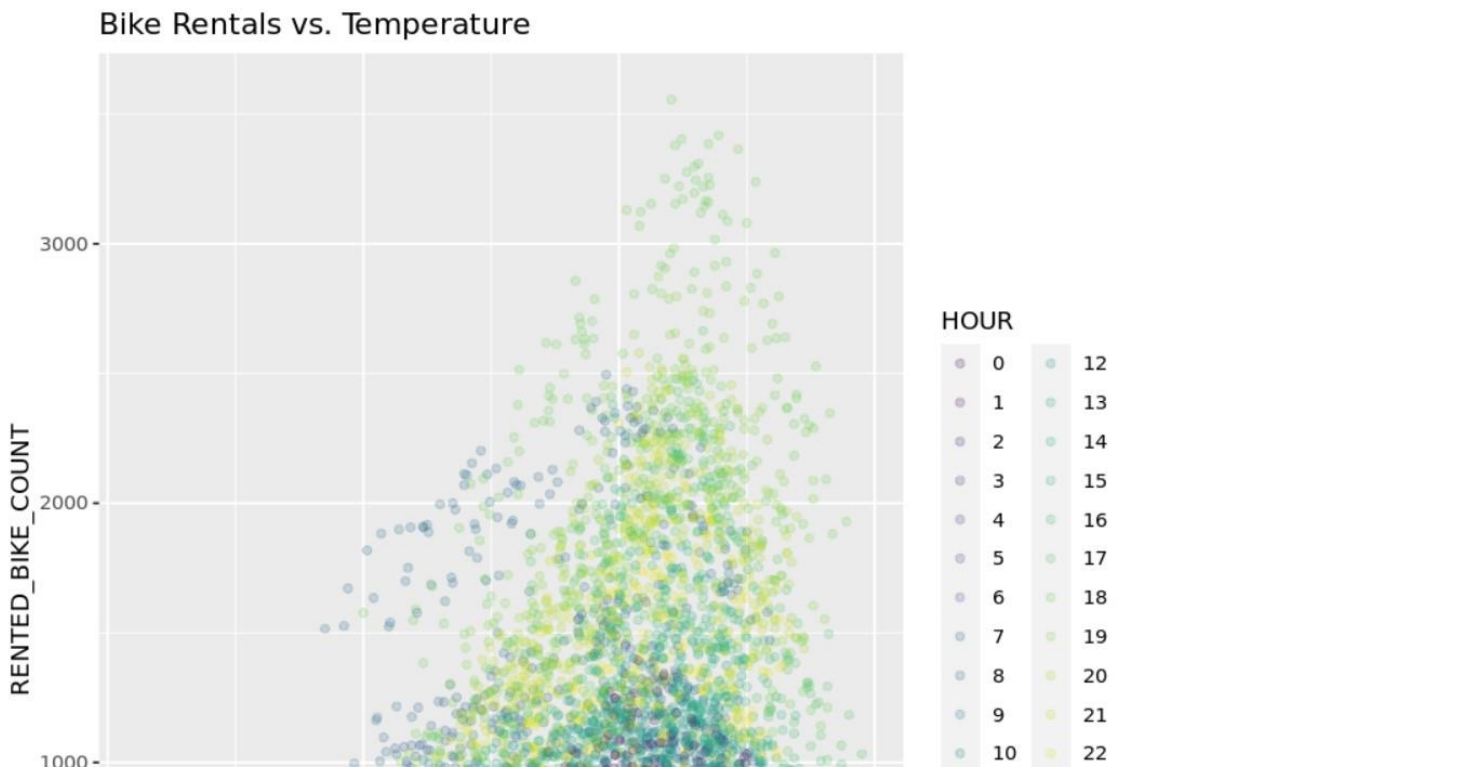
```
[17]:
# provide your solution here
ggplot(seoul_bike_df, aes(x=TEMPERATURE, y=RENTED_BIKE_COUNT, color=HOUR)) + geom_point(alpha=0.4) +facet_wrap(~SEASONS) +
       ggtitle("Rentals vs. Temperature with Seasons")
```



Appendix:EDA-Data Visualization-5

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

[16]:
```
ggplot(seoul_bike_df) +
   geom_point(aes(x=TEMPERATURE,y=RENTED_BIKE_COUNT,colour=HOUR),alpha=1/5) + ggtitle("Bike Rentals vs. Temperature")
```



Appendix:EDA-Data Visualization-6

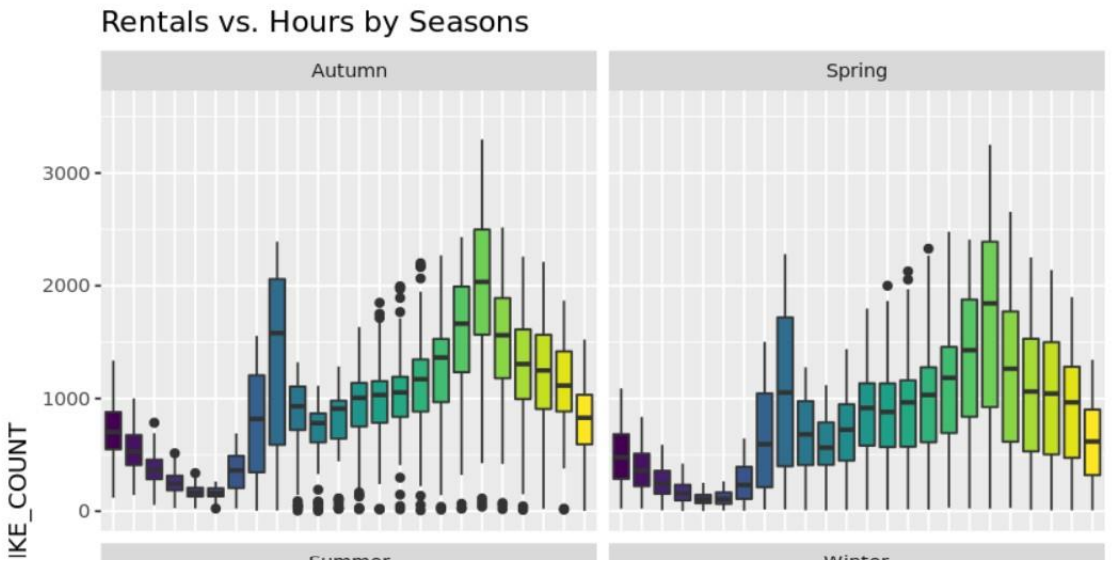
Outliers (boxplot)

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

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

Solution 14

•[18]:
```
# provide your solution here
ggplot(seoul_bike_df, aes(x=HOUR, y=RENTED_BIKE_COUNT)) +geom_boxplot(aes(fill=HOUR)) + facet_wrap(~SEASONS) +guides(fill="none") +
ggtitle("Rentals vs. Hours by Seasons") +
scale_x_discrete(guide=guide_axis(n.dodge=2))
```



Appendix:EDA-Data Visualization-7

```
rain_snow_fall <- seoul_bike_df %>%
                  group_by(DATE) %>%
                  summarize(Daily_Rainfall=sum(RAINFALL), Daily_Snowfall=sum(SNOWFALL))

ggplot(rain_snow_fall, aes(x=DATE,y=Daily_Rainfall)) +
geom_col(position="identity", fill="turquoise2") +
ggtitle("Daily Total Rainfall") + ylab("Rainfall in mm") +
theme(plot.title = element_text(hjust = 0.5))

ggplot(rain_snow_fall, aes(x=DATE,y=Daily_Snowfall)) +
geom_col(position="identity", fill="pink") +
ggtitle("Daily Total Snowfall") +ylab("Snowfall in cm")+
theme(plot.title = element_text(hjust = 0.5))
```