Predicting Bike Rental Count

Sivamani Murugan
07th December 2019

Contents

Introduction

1. Problem Statement

Predicting the Bike Rental Count based on the Historical data. This will help Client to understand the Bike Rental Count pattern and based on our prediction, client make necessary arrangements to have bikes available. The data provided to us contains of various attributes to decide the bike rental count like weather, season, etc.

2. Data

Historical Data File Name: "*Day.csv*"

Table 1.1: Data variables

Variable Name	Explanation
instant	Record Index
dteday	Date
season	Season (1: Springer, 2: summer, 3: fall, 4: winter)
yr	Year (0: 2011, 1: 2012)
mnth	Month (1 to 12)
holiday	Indicates whether it is holiday or not (0: holiday, 1: Not Holiday)
weekday	Day of the week (0 to 6)
workingday	Indicates whether it is working(excluding week end and holiday's) 0: non- working day, 1: working day
weathersit	Indicates weather on the given day 1: Clear, Few clouds, Partly cloudy, Partly cloudy 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered Clouds 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
temp	Normalized Temperature in Celsius (Min = -8, Max = +39)
atemp	Normalized Feeling Temperature in Celsius (Min = -16, Max = +50)
hum	Normalized Humidity (values are divided to Max= 100)
windspeed	Normalized WindSpeed (values are divided to Max=67)
casual	Count of Casual Bike Booking count
registered	Count of Registered Bike Booking count
cnt	Count of total Bike Booking count (casual + registered)

Table 1.2: Sample Data

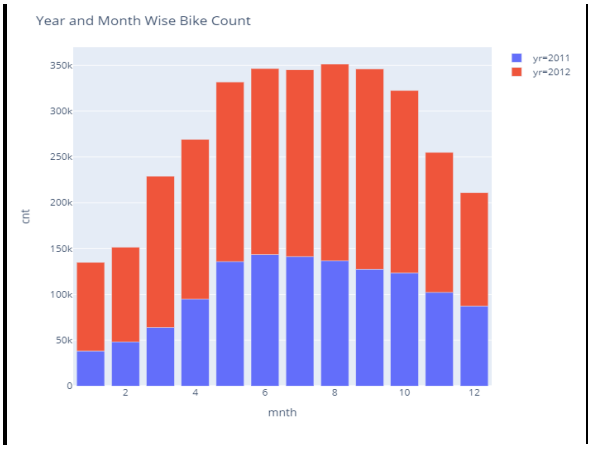
instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	1/1/2011	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	1/2/2011	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
3	1/3/2011	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
4	1/4/2011	1	0	1	0	2	1	1	0.2	0.212122	0.590435	0.160296	108	1454	1562
5	1/5/2011	1	0	1	0	3	1	1	0.226957	0.22927	0.436957	0.1869	82	1518	1600
6	1/6/2011	1	0	1	0	4	1	1	0.204348	0.233209	0.518261	0.0895652	88	1518	1606
7	1/7/2011	1	0	1	0	5	1	2	0.196522	0.208839	0.498696	0.168726	148	1362	1510
8	1/8/2011	1	0	1	0	6	0	2	0.165	0.162254	0.535833	0.266804	68	891	959
9	1/9/2011	1	0	1	0	0	0	1	0.138333	0.116175	0.434167	0.36195	54	768	822
10	1/10/2011	1	0	1	0	1	1	1	0.150833	0.150888	0.482917	0.223267	41	1280	1321

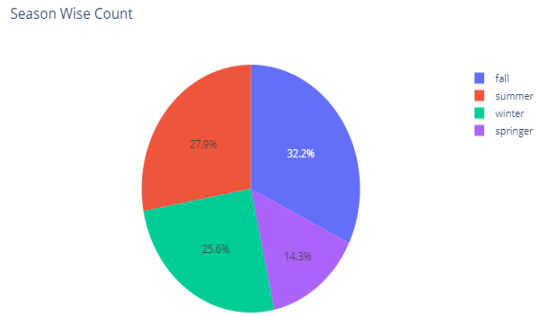
Exploratory Data Analysis

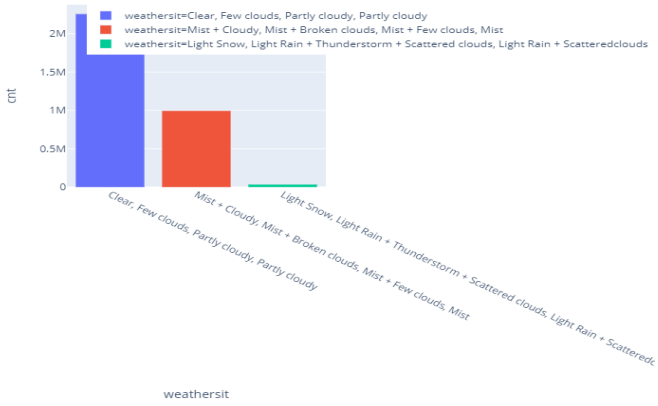
1. Analysis:

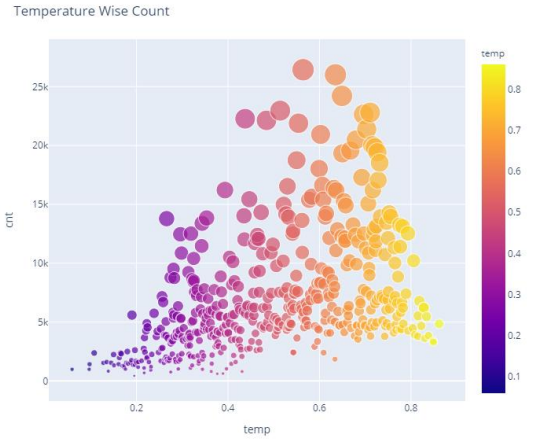
1. The given "day.csv" file has **16 variables with 731 observations**
2. Below are the numerical/continuous variables
    a. *temp*
    b. *atemp*
    c. *hum*
    d. *windspeed*
    e. *casual*
    f. *registered*
    g. *cnt*
3. Below are the categorical variables
    a. season
    b. yr
    c. mnth
    d. holiday
    e. weekday
    f. workingday
    g. weathersit
4. Based on the Understanding "cnt" target variable with respect to other categorical variables, we can understand the pattern of Bike rental.
5. Below the graphs which explains sum of the "cnt" against all the categorical variables

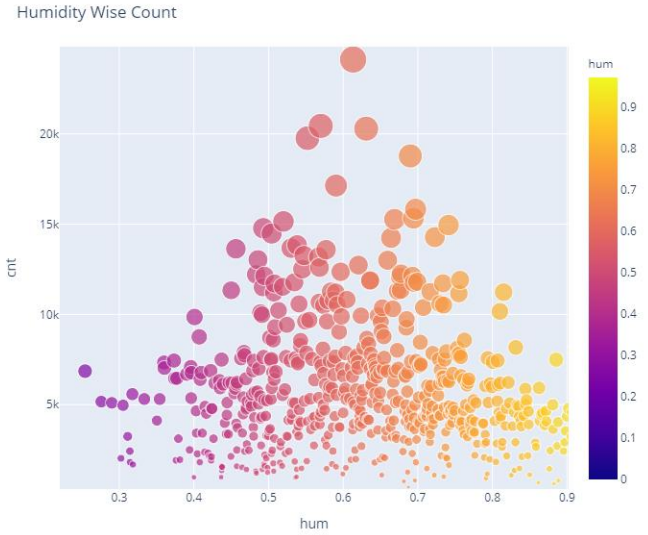
Figure 1.1: Bike Count based on different Independent variables (refer Python/R code)

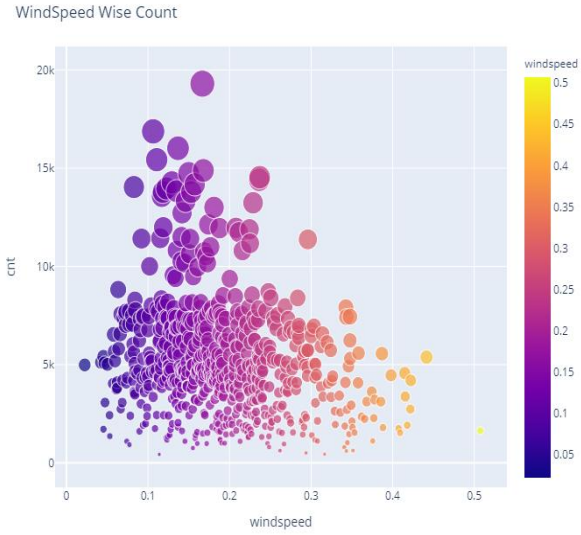












2. Conclusion on Preliminary Analysis

Table 1.3: Preliminary Prediction

Season	Temperature	Humidity	Weather	Wind speed	Bike Count
Fall /summer/Winter	Between 0.4 to 0.7	Between 0.5 to 0.8	Clear/Mist	Between 0.1 to 0.3	High Bookings
Other seasons	Between 0 to 0.3 Or Between 0.8 to 1	Between 0 to 0.3 Or Between 0.9 to 1	Other weather	Between 0.35 to 0.6	Very Low Bookings

3. Pre Processing

Missing Value Analysis

1. Based on the given data, we don't see any Missing Value in any of the given variables

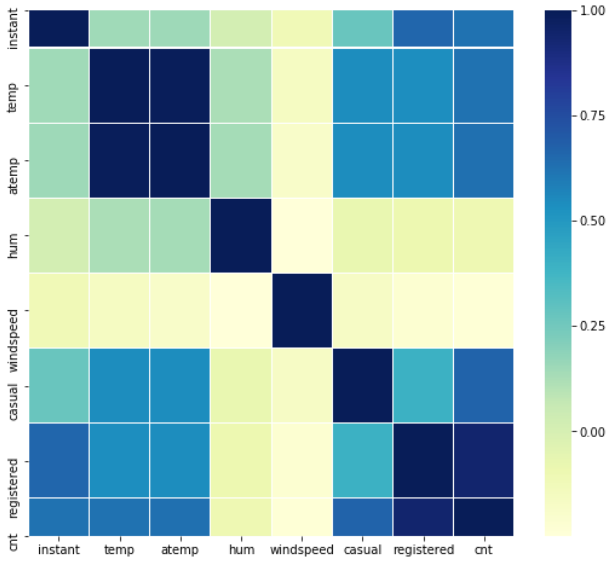
```
#Pre-Processing Technique #1: Check Missing Values in all variables
df_null = df_train.isna().sum().to_frame()
df_null.reset_index(inplace=True)
df_null.rename(columns={0:"MissingValueCount","index":"variable"},inplace=True)
print(df_null.loc[df_null["MissingValueCount"] > 0])
# No Missing Values Found
```

```
Empty DataFrame
Columns: [variable, MissingValueCount]
Index: []
```

Feature Selection

1. For Numerical variables, we can check Co linearity between the variables to understand the dependency
2. Since multiple variables with same information causes model performance, we need to remove one of the variable among two variable which are dependent
3. I have used the Correlation Graph to identify the Multi-Co linearity between Numerical Independent variables

Figure 2.1: Correlation Graph (refer Python/R code)



4. Based on the Correlation graph, we observe that “atemp” and “temp” are collinear to each other.
5. So, we can drop the “atemp” variable from further analysis
6. Also, we can drop “casual” and “registered” variables, since we considered “cnt” as target variable which is sum of “casual” and “registered”
7. For Categorical Test, we can identify Multi-Co linearity based on either ANOVA test or Chi-Square test.
8. We have used ANOVA test to verify the Multi-Co linearity between categorical variables
9. And as per the ANOVA, all the categorical variables carry meaningful information about the Target variable.
10. In addition, we can drop “dteday” variable, since the required information is split across “yr”,”mnth”,”weekday”.
11. Below are the final Variables after Feature Selection

```
'season', 'yr', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'hum',
'windspeed', 'cnt'
```

Outlier Analysis

1. The Outlier analysis on Numerical variables is important to have good data for model prediction.
2. We have “temp”,”hum” and “windspeed” numerical variables and out of which, we see “hum” and “windspeed” has Outliers as below

Figure 2.2: Outlier Data for Numerical variables (refer Python/R code)

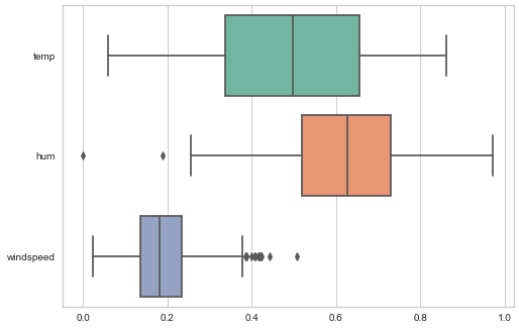


Figure 2.3: Histogram of “hum” with Outlier Data

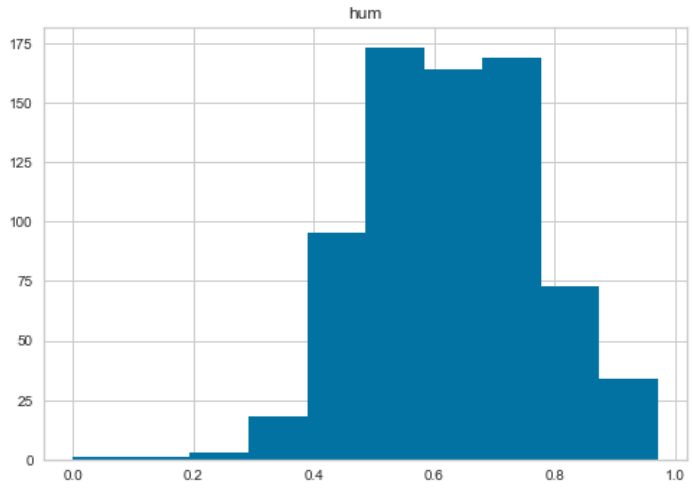


Figure 2.4: Histogram of “hum” without Outlier Data

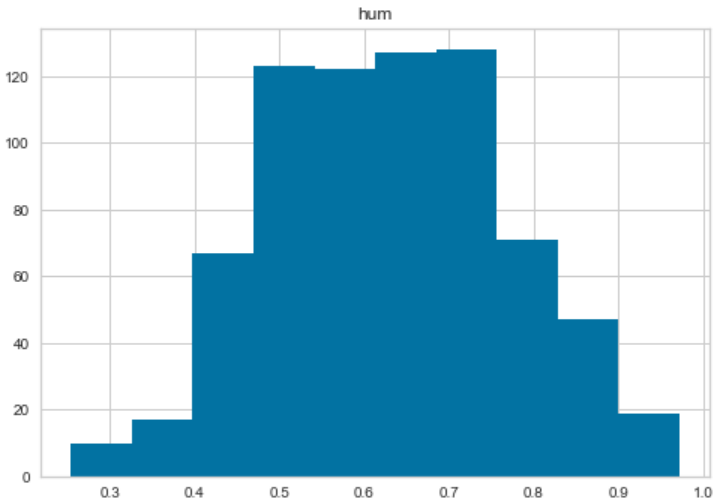


Figure 2.5: Histogram of "windspeed" with Outlier Data

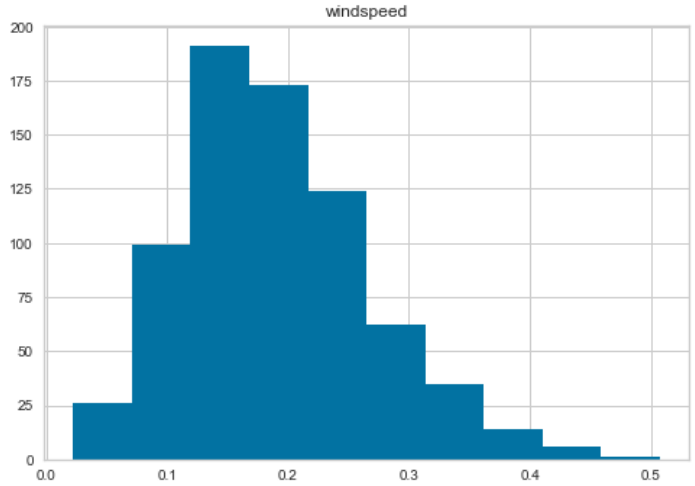
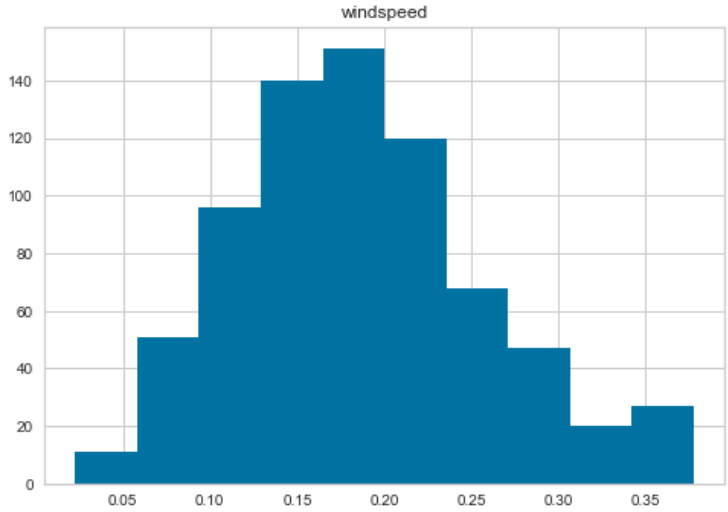


Figure 2.6: Histogram of "windspeed" without Outlier Data



3. To remove the Outlier, we have updated the outlier data with "NA" and imputed the "NA" with Mean of the corresponding columns

Figure 2.7: Python Code to check the Outlier and Update with NA

```
In [385]: # For all numeric columns(other than "ID_Code"), replace the outlier data with NA
numerical_columns = df_train.select_dtypes(include = "float64").columns
# The outlier analysis can be done only on Numeric variables
# Get 75th percentile of data
# Get 25th percentile of data
# Get inter quarter data
# delete the data which is below/above of the lower/upper bound respectively
for col in numerical_columns:
    i75, i25 = np.percentile(df_train.loc[:,col],[75,25])
    iqr = i75 - i25
    lower_bound = i25 - (iqr * 1.5)
    upper_bound = i75 + (iqr * 1.5)
    print("column Name:" + col)
    print("i75:" + str(i75))
    print("i25:" + str(i25))
    print("Lower Bound:" + str(lower_bound))
    print("Upper Bound:" + str(upper_bound))
    print("No. Of Rows before Lower Bound:" + str(df_train.loc[df_train[col]<lower_bound,col].count()))
    print("No. Of Rows after Upper Bound:" + str(df_train.loc[df_train[col]>upper_bound,col].count()))
    df_train.loc[df_train[col]<lower_bound,col] = np.nan
    df_train.loc[df_train[col]>upper_bound,col] = np.nan
```

```
column Name:temp
i75:0.6554165000000001
i25:0.3370835
Lower Bound:-0.14041600000000015
Upper Bound:1.1329160000000003
No. Of Rows before Lower Bound:0
No. Of Rows after Upper Bound:0
column Name:hum
i75:0.7302085
i25:0.52
Lower Bound:0.20468725
Upper Bound:1.0455212500000002
No. Of Rows before Lower Bound:2
No. Of Rows after Upper Bound:0
column Name:windspeed
i75:0.2332145
i25:0.13495
Lower Bound:-0.012446750000000034
Upper Bound:0.38061125
No. Of Rows before Lower Bound:0
No. Of Rows after Upper Bound:13
```

Data Sampling

1. Sampling in necessary to train and test the model to understand the Model performance
2. As a standard practice, we should take **80% of sample** and **20% of test** data.
3. I have used simple sample, to get the sample

4. Model Development and Accuracy Metrics

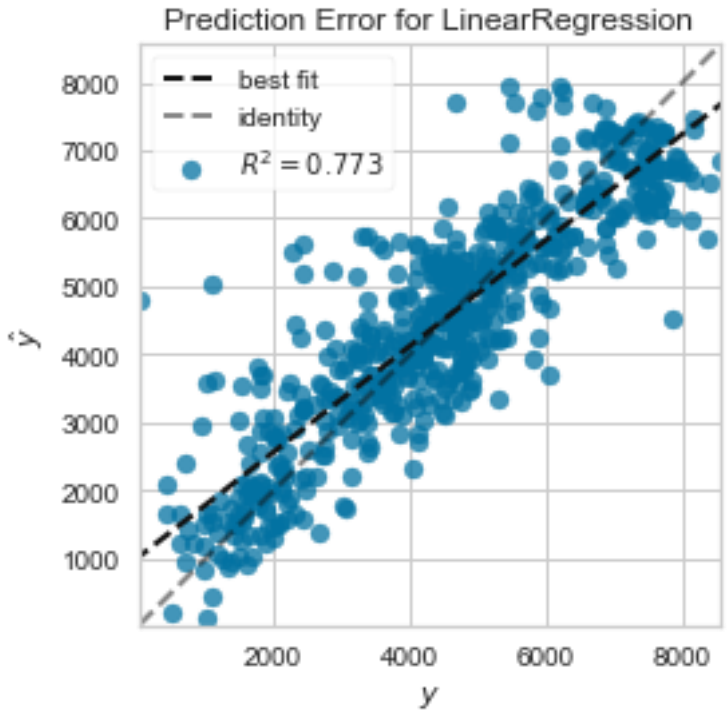
1. Linear Regression Model

a. Train the model with 80% of data with all final list of variables

b. Below are the accuracy metrics

Metrics	Python Values	R Values
R-Squared	77.3%	84.25%
Mean Squared Error(MSE)	838411.88	579230.3
Mean Absolute Percentage deviation(MAPE)	58.31%	18.52%
Min Max Accuracy	NA	84.67%
Correlation Accuracy	NA	90.2%

Figure 4.1: Prediction Error for Linear Regression in Python



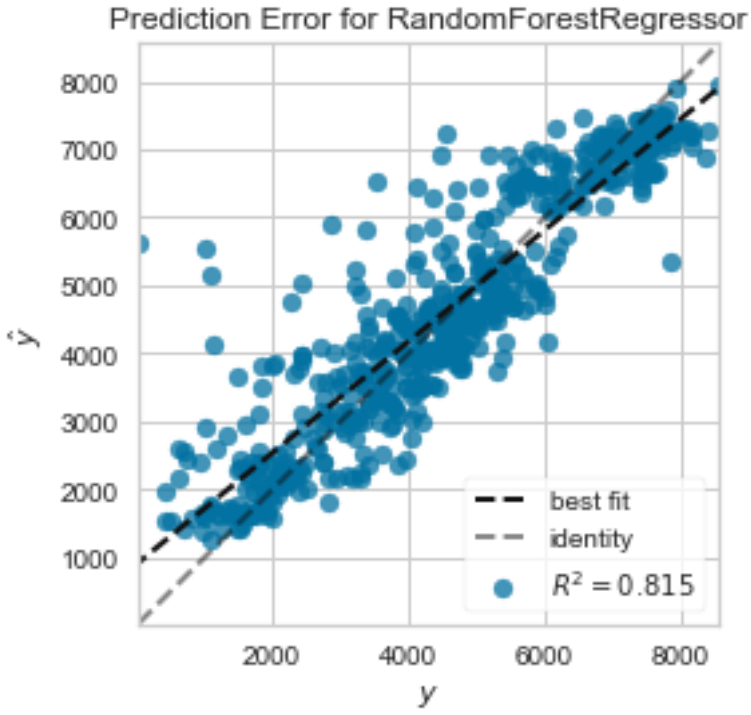
2. Random Forest Regression Model

a. Train the model with 80% of data with all final list of variables

b. Below are the accuracy metrics

Metrics	Python Values	R Values
R-Squared	81.5%	NA
Mean Squared Error(MSE)	684469.02	NA
Mean Absolute Percentage deviation(MAPE)	63.94%	14.44%
Min Max Accuracy	NA	89.21%
Correlation Accuracy	NA	94.3%

Figure 4.1: Prediction Error for Random Forest Regression in Python



3. Finalize the Model

a. Based on the accuracy metrics, in both R and Python, the "**Random Forest**" algorithm performs better when compared to Linear Regression model.

b. Save the model to Disk

5. Test the Sample Data

1. Existing one sample data

a. We can provide the below sample data for the Random Forest Model to check the output

Table 5.1: Sample Value to predict

Season	Year	Month	Holiday	Weekday	Working day	Weather	Temp	Hum	Wind Speed
Springer	2011	January	No	Wednesday	Yes	Clear, Few Clouds	0.226957	0.436957	0.1869

Actual and Predicted Values

Table 5.2: Predicted values using Python and R

Actual Bikes Booked	Python Predicted Bike Count	R Predicted Bike Count
1600	1642	1324

2. Using Test csv file

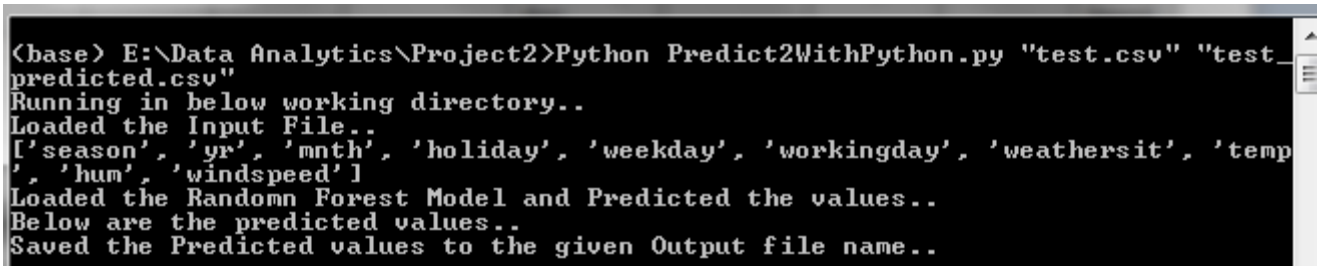
b. I have developed Python and R script to provide input as Test csv file and get the predicted values based on Saved Random Forest Model

c. Below are the steps to predict using saved Random forest model using Python

   a. Place the test csv file in any folder
   b. Place the "rf_final_model_py.sav" model file and "Predict2withPython.py" file in the same folder
   c. From Command Prompt, navigate to the above placed folder
   d. Run the below command

```
Python Predict2withPython.py <input_csv_filename> <output_csv_filename>
```

Figure 5.1: Sample Command for Python Prediction



d. Below are the steps to predict using saved Random forest model using R
    a. Place the test csv file in any folder
    b. Place the "rf_final_model.rds" model file and "Predict2WithR.R" file in the same folder
    c. From Command Prompt, navigate to the above placed folder
    d. Run the below command

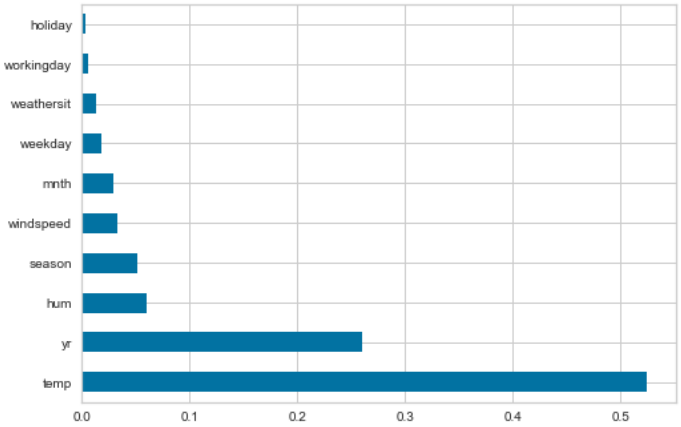
```
RScript --vanilla Predict2withR.R <input_csv_filename> <output_csv_filename>
```

Figure 5.1: Sample Command for R Prediction

```
E:\Data Analytics\Project2\R>RScript --vanilla Predict2WithR.R "test.csv" "test_
prd.csv"
randomForest 4.6-14
Type rfNews() to see new features/changes/bug fixes.
Warning message:
package 'randomForest' was built under R version 3.6.1
[1] "test.csv"
[1] "test_prd.csv"
[1] "Loaded given csv files..:test.csv"
[1] "Loaded the model..:rf_final_model.rds"
[1] "Saved the Predicted values to..test_predicted.csv file"
```

Appendix A – Additional Figures

Figure 6.1: Feature Importance in Random Forest Model



Appendix B – Python Code

a. Refer attached "***Project2WithPython.ipynb***" Jupyter Notebook file for the complete code for model development
b. Refer attached "***Predict2WithPython.py***" python file to Predict Test data from Command Prompt(as mentioned 5.2 "Using Test csv file)

Appendix C – R Code

a. Refer attached "***Project2WithR.R***" R file for the complete code for model development
b. Refer attached "***Predict2WithR.R***" R file to Predict Test data from Command Prompt(as mentioned 5.2 "Using Test csv file)