Project Name - Bike Renting

Candidate Name: Ajay Jadhav

Date: 29th June 2020

Contents

Chapter 1

1. Introduction

1.1 Problem Statement

The objective of this Case is to Predication of bike rental count on daily based on the environmental and seasonal settings.

1.2 Data

Dataset has 16 variables out of which 15 variables are independent and 1 ('cnt') is dependent variable which is a target variable. And we have to prepare a model to predict the count of bikes on daily basis based on environmental and seasonal conditions. In the dataset target variable is continuous in nature so this problem comes under Regression problem.

Table 1.1 Sample data

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

The data attributes in the dataset:

1) instant: Record index

2) dteday: Date

3) season: Season (1:springer, 2:summer, 3:fall, 4:winter)

4) yr: Year (0: 2011, 1:2012)

5) mnth: Month (1 to 12)

6) hr: Hour (0 to 23)

7) holiday: weather day is holiday or not (extracted fromHoliday Schedule)

8) weekday: Day of the week

9) workingday: If day is neither weekend nor holiday is 1, otherwise is 0.

10) weathersit: (extracted fromFreemeteo)

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

11) temp: Normalized temperature in Celsius. The values are derived via

(t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (only in hourly scale)

12) atemp: Normalized feeling temperature in Celsius. The values are derived via

(t-t_min)/(t_maxt_min), t_min=-16, t_max=+50 (only in hourly scale)

13) hum: Normalized humidity. The values are divided to 100 (max)

14) windspeed: Normalized wind speed. The values are divided to 67 (max)

15) casual: count of casual users

16) registered: count of registered users

17) cnt: count of total rental bikes including both casual and registered

Chapter 2

2. Methodology

2.1 Data Pre -Processing

Data Pre – Processing also known as Exploratory Data Analysis is an initial and most important step in any Data science project way before we step in to Machine Learning. We derive Useful Information needed for the model to run effectively Therefore it's extremely important process and take approximately up to 70% of the time estimated for the project.
Following are the Data Pre – Processing techniques used:

2.1.1 Exploratory Data Analysis and Data Cleaning

First, we import Data in to our Environments, Explore the data by checking its Dimensions, Data Structures, Variable Names, Summary, Rename Data variables for easy understanding of the data.

Table 2.1 describe dataset

```
df.describe()
```

	instant	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	73
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	1.395349	0.495385	0.474354	0.627894	0.190486	84
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	0.544894	0.183051	0.162961	0.142429	0.077498	68
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.000000	0.059130	0.079070	0.000000	0.022392	
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	1.000000	0.337083	0.337842	0.520000	0.134950	31
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	1.000000	0.498333	0.486733	0.626667	0.180975	71
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	2.000000	0.655417	0.608602	0.730209	0.233214	109
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	3.000000	0.861667	0.840896	0.972500	0.507463	341

Table 2.2 data types and data columns

```
df.dtypes
instant          int64
dteday          object
season           int64
yr               int64
mnth             int64
holiday          int64
weekday          int64
workingday       int64
weathersit       int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
dtype: object
```

```
df.columns
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
```

2.1.2 Missing Value Analysis

Missing Values occurs when there's data value stored in any column or data set that may happen due to some error while storing at very first place or may have been missed. Although it doesn't matter a much but if there's more than % of the data missing value then the entire column is ignored. Here in our data set there is no missing value

Table 2.3 Missing Values

	Missing_value	percent_missing
count	0	0.0
windspeed	0	0.0
humidity	0	0.0
atemp	0	0.0
temperature	0	0.0
weather	0	0.0
workingday	0	0.0
weekday	0	0.0
holiday	0	0.0
month	0	0.0
year	0	0.0
season	0	0.0

2.1.3 Outlier Analysis

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of distribution. The easiest way to detect outliers is to create a graph. Plots such as Box plots, Scatterplots and Histograms can help to detect outliers. Alternatively, we can use mean and standard deviation to list out the outliers. Interquartile Range and Quartiles can also be used to detect outliers.

In our case we used Box plots for detecting outliers and almost all the variables do not have outliers except "windspeed" and "humidity".

Boxplot stat method is one of the methods to remove outliers by treating those as missing values in first place and impute them with mean, median or KNN imputation method or in case the outliers are more then we delete such observations.

Table 2.4 Variable with Outliers

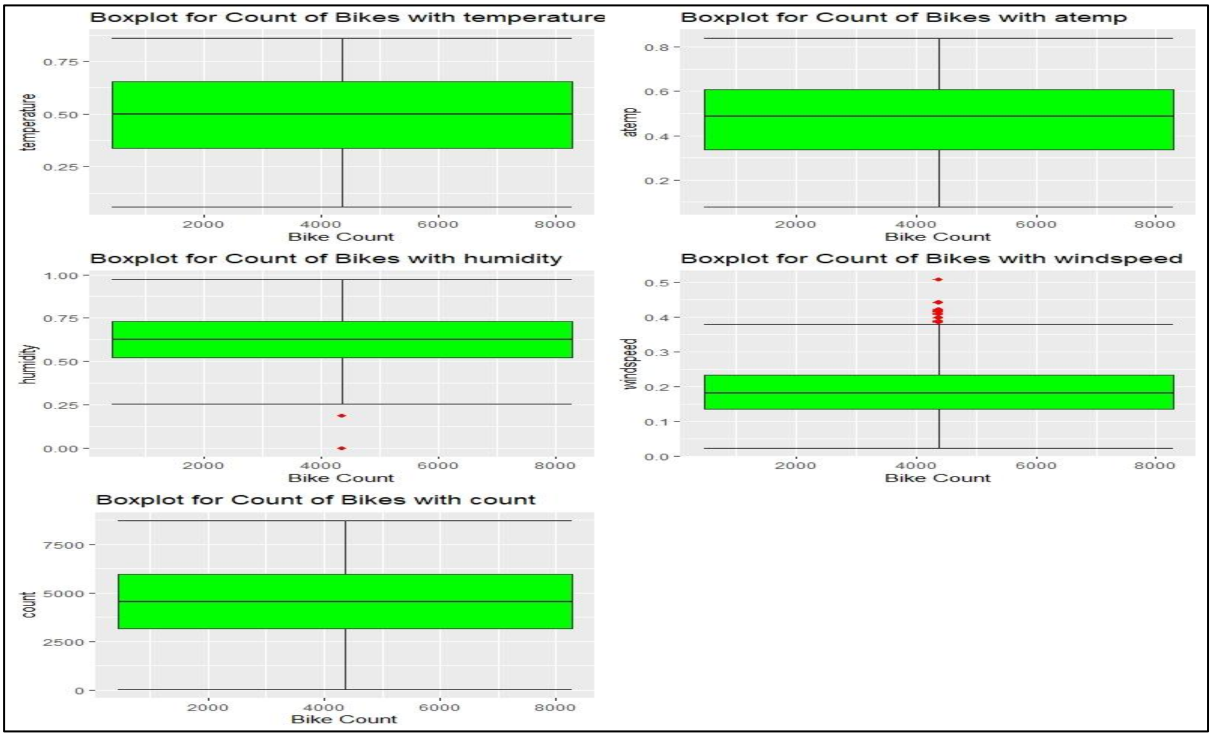
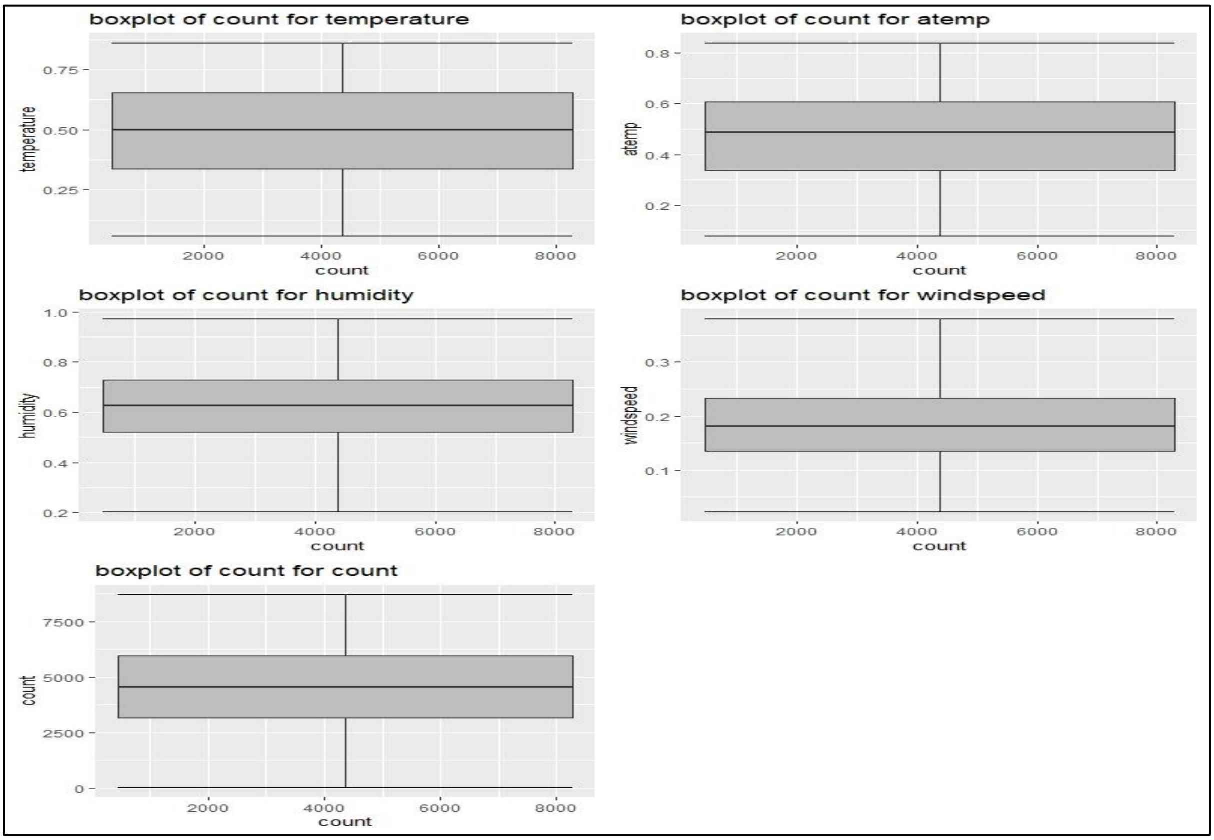


Table 2.5 Variable after removing Outliers

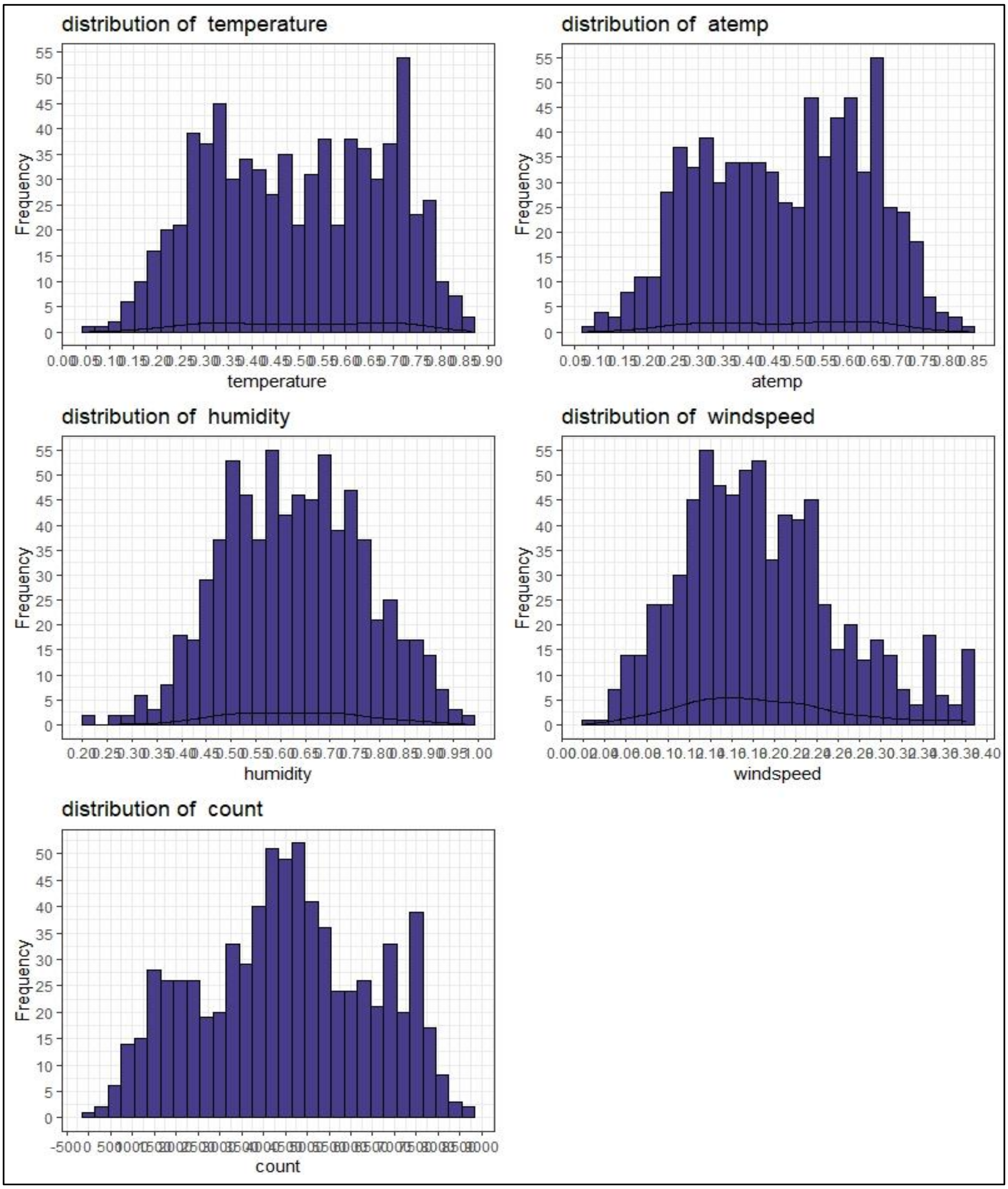


2.1.4 Data understanding using visualization

2.1.4.1 Distribution of continuous variables

For checking of Distribution of each continuous variable we plot histogram in both R and Python and check the distribution using the plots. For our model according to the plots we conclude that our model is Normally distributed.

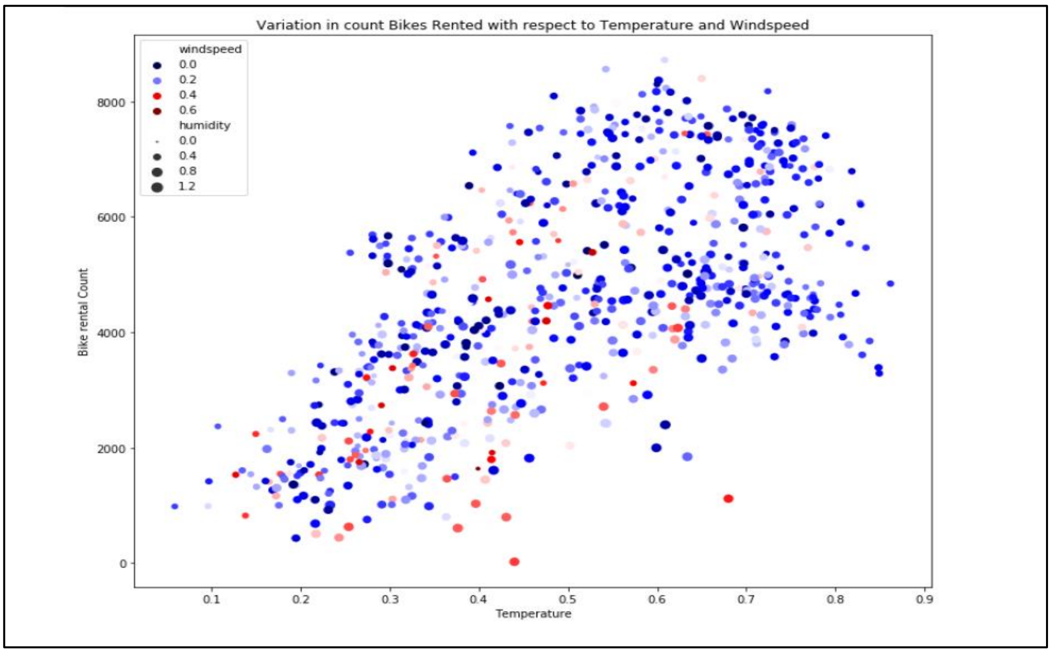
<u>Table 2.6 Distribution of continuous variables</u>



2.1.4.2 Distribution of categorical variables wrt target variable

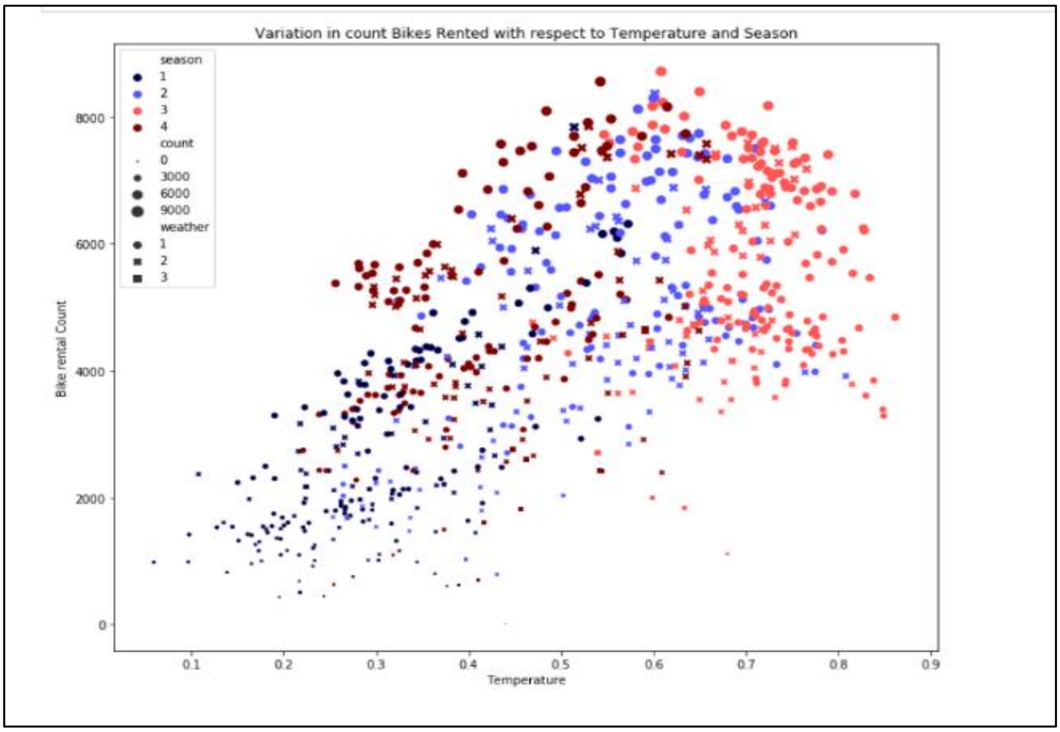
Checking the impact of each continuous variables on target variable using scatterplot

<u>Table 2.7 Variation in bike rented with respect to Temperature and windspeed</u>



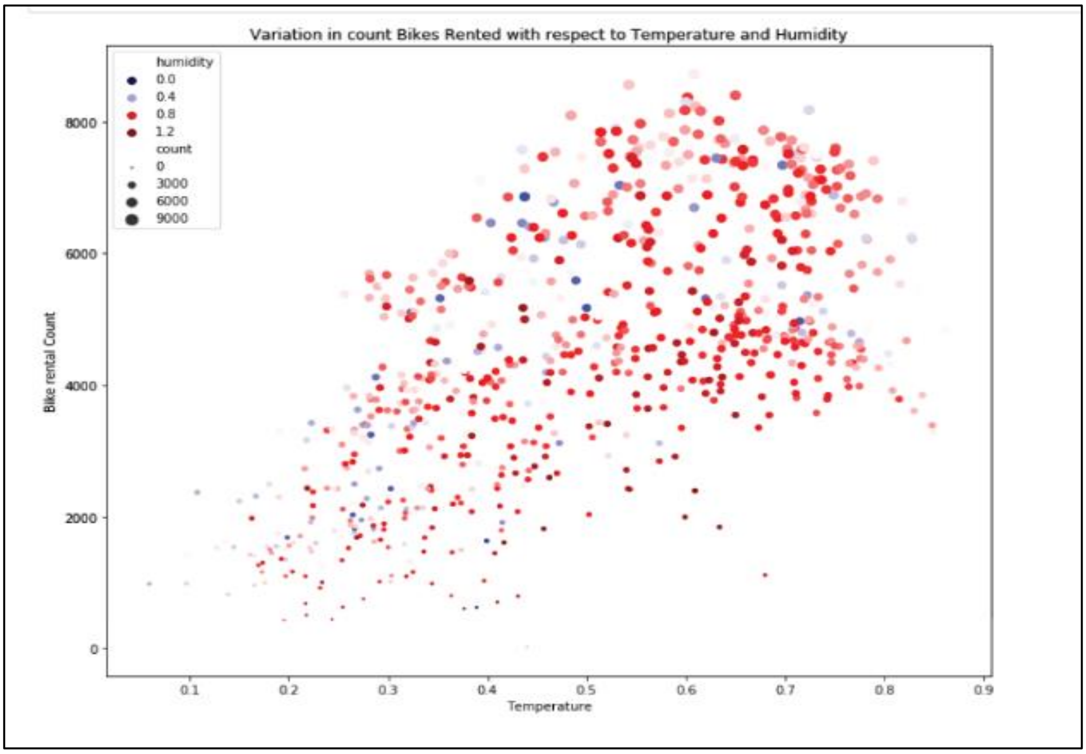
Bikes rented with respect to Temperature and windspeed: According the Scatterplot plotted above the maximum no of bikes rented is found when the temperature is between scale 0.5 to 0.8 and humidity below 0.8 and windspeed below 0.2

<u>Table 2.8 Variation in bike rented with respect to Temperature and Season</u>



Bikes rented with respect to Temperature and Season: According the Scatterplot plotted above the maximum no of bikes rented is found during Season 1,2 and 4 when the temperature is between scale 0.5 to 0.8 and weather is 1 and 2 even in it weather 1 is more

<u>Table 2.9 Variation in bike rented with respect to Temperature and Humidity</u>

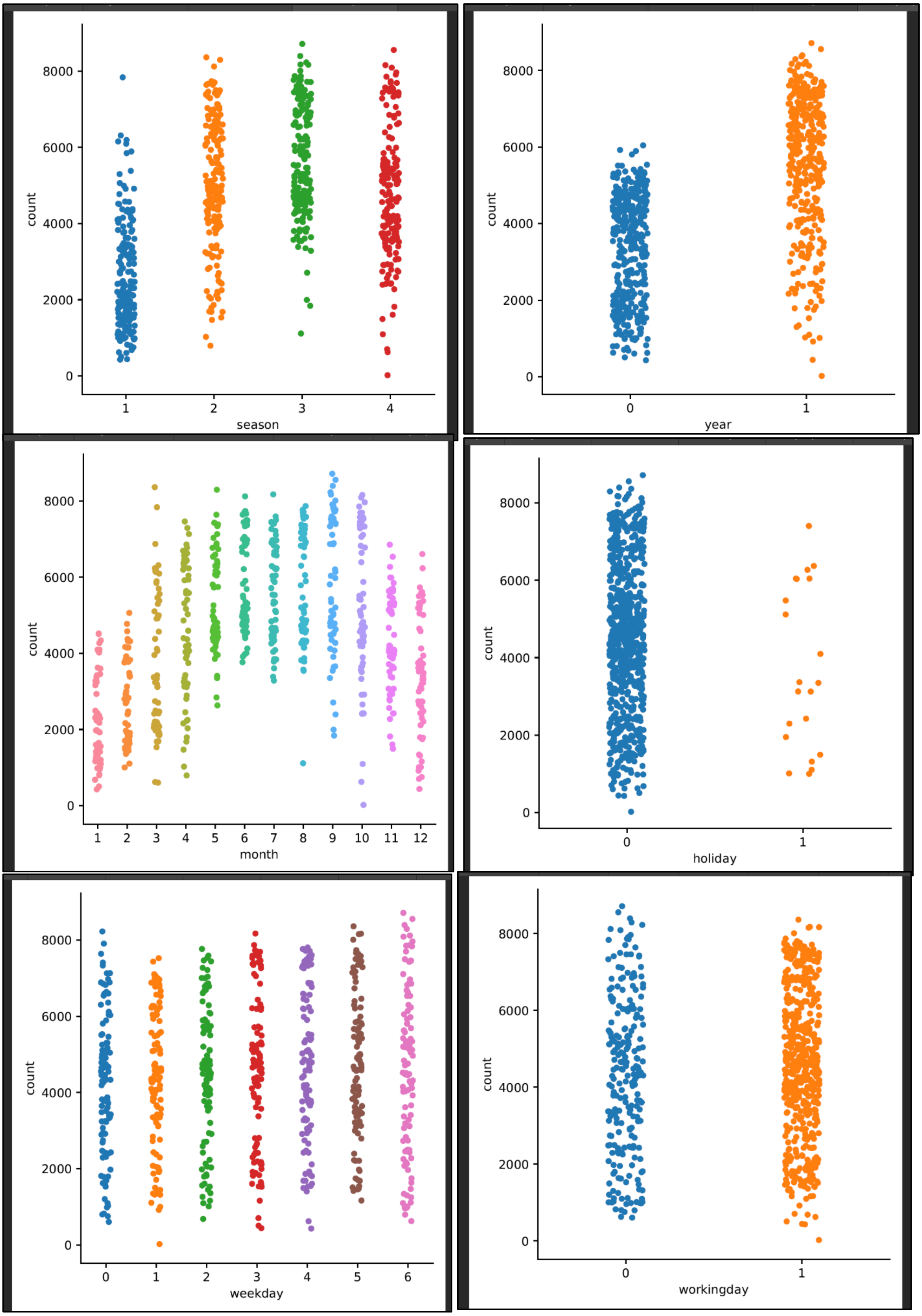


Bikes rented with respect to Temperature and Humidity: According the Scatterplot plotted above the maximum no of bikes rented is found when the temperature is between scale 0.5 to 0.8 and humidity below 0.8.

2.1.4.3 Distribution of continuous variables wrt target variable

1.) season vs count: Summer, Fall and Winter has more count as compared to Spring season with almost between 4000 to 8000 bike count on daily basis
2.) year vs count: the count of bike rented increased in the year 2012 as compared to 2011
3.) month vs count: the count goes on increases gradually from march and reaches maximum up to October and slightly decreases in November and December.
4.) holiday vs count: the count of bike rented is much high on holidays as compared to working day
5.) week day vs count: bike count is maximum on day 5 and 6 as per the weekdays
6.) working day vs count: the count is slightly increased on week ends as compared to working days
7.) weather vs count: the count of bike rented is maximum on days having clear weather with few or partly cloudy as compared to days with mist combined with clouds and least
in bad weather

Table 2.10 Distribution of continuous variables wrt target variable

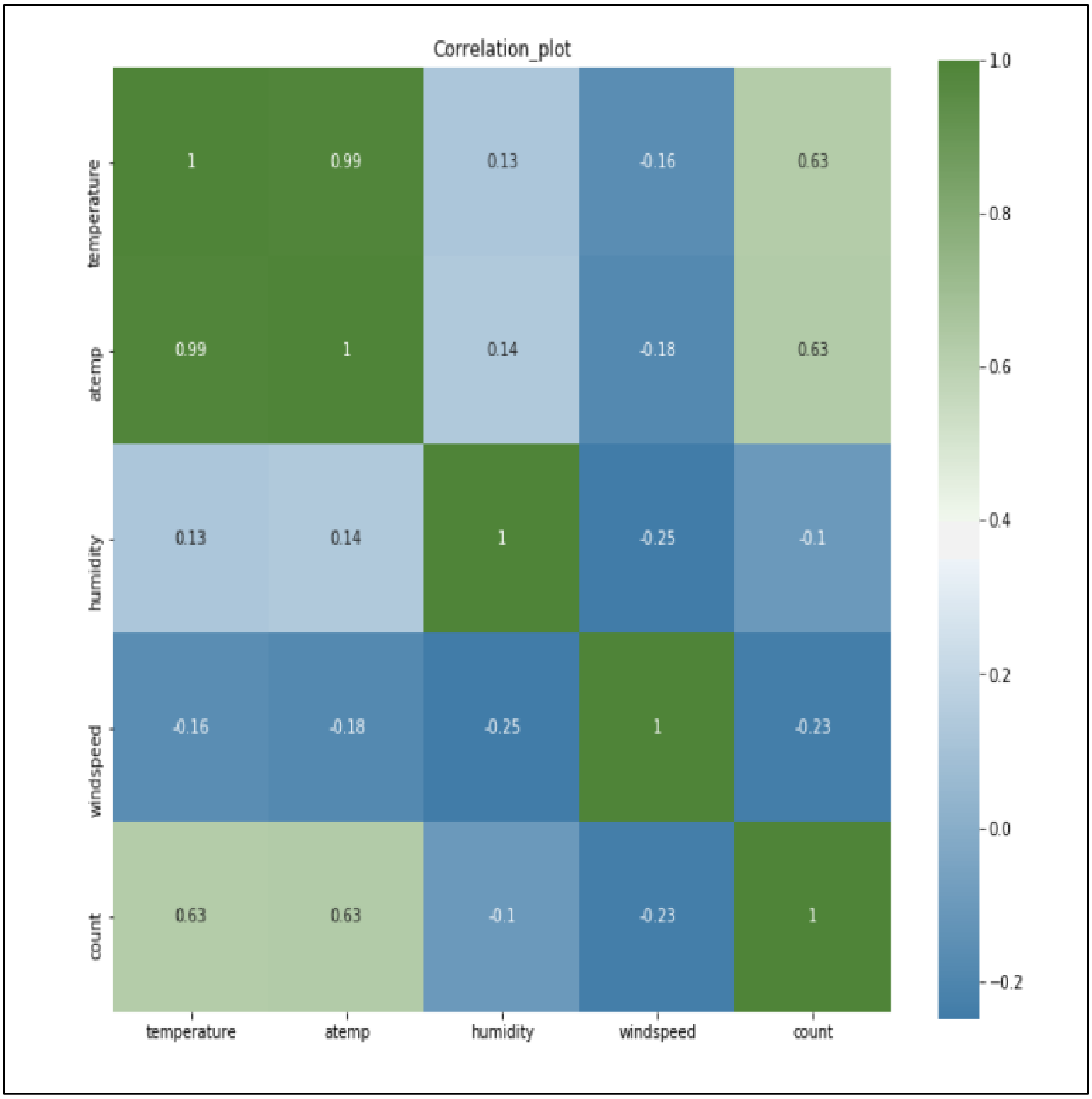


2.1.5 Feature Selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

Correlation Analysis for continuous variables and ANOVA (Analysis of Variance) for categorical variables.

<u>Table 2.11 Correlation plot for all continuous variables.</u>



From Correlation Analysis we Conclude that Variable “temperature” and “atemp” has high correlation so we drop the “atemp” variable.

Table 2.12 Result of ANOVA Test

```
            df        sum_sq       mean_sq           F        PR(>F)
season     1.0  4.517974e+08  4.517974e+08  143.967653  2.133997e-30
Residual  729.0  2.287738e+09  3.138187e+06         NaN           NaN
            df        sum_sq       mean_sq           F        PR(>F)
year       1.0  8.798289e+08  8.798289e+08  344.890586  2.483540e-63
Residual  729.0  1.859706e+09  2.551038e+06         NaN           NaN
            df        sum_sq       mean_sq          F        PR(>F)
month      1.0  2.147445e+08  2.147445e+08  62.004625  1.243112e-14
Residual  729.0  2.524791e+09  3.463362e+06        NaN           NaN
            df        sum_sq       mean_sq         F    PR(>F)
holiday    1.0  1.279749e+07  1.279749e+07  3.421441  0.064759
Residual  729.0  2.726738e+09  3.740381e+06       NaN       NaN
            df        sum_sq       mean_sq         F    PR(>F)
weekday    1.0  1.246109e+07  1.246109e+07  3.331091  0.068391
Residual  729.0  2.727074e+09  3.740843e+06       NaN       NaN
              df        sum_sq       mean_sq         F    PR(>F)
workingday   1.0  1.024604e+07  1.024604e+07  2.736742  0.098495
Residual    729.0  2.729289e+09  3.743881e+06       NaN       NaN
            df        sum_sq       mean_sq          F        PR(>F)
weather    1.0  2.422888e+08  2.422888e+08  70.729298  2.150976e-16
Residual  729.0  2.497247e+09  3.425578e+06        NaN           NaN
```

From ANOVA Analysis we conclude that in variables “holiday”, “weekday” and “workingday” have value of pr>0.05 so we drop these variables.

2.1.5 Feature Selection

Table 2.13 Sample Data after Feature Selection

	season	year	month	weather	temperature	humidity	windspeed	count
0	1	0	1	2	0.344167	0.805833	0.160446	985.0
1	1	0	1	2	0.363478	0.696087	0.248539	801.0
2	1	0	1	1	0.196364	0.437273	0.248309	1349.0
3	1	0	1	1	0.200000	0.590435	0.160296	1562.0
4	1	0	1	1	0.226957	0.436957	0.186900	1600.0

2.1.6 Feature Scaling

Feature scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data pre-processing step. As the given data for continuous variables is already Normalised form, so we don't need to scale the data. We can check the normality of data by using histogram plot and by summarizing the variables.

Table 2.14 Histogram plot that shows Normalised data

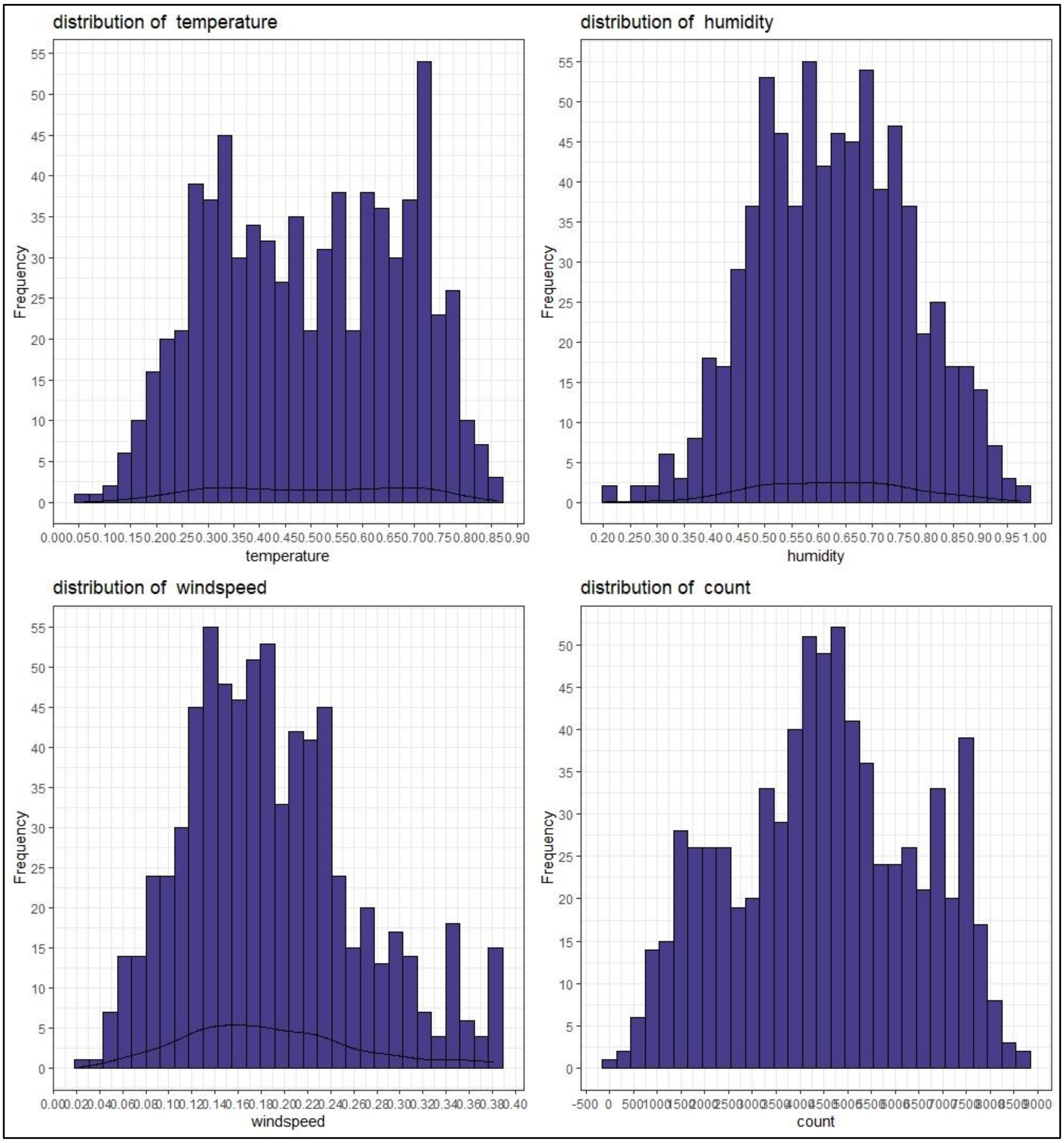


Table 2.15 summary of all variables

```
df.describe()
```

	season	year	month	weather	temperature	humidity	windspeed	count
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000
mean	2.496580	0.500684	6.519836	1.395349	0.495385	0.627894	0.190486	4504.348837
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.142429	0.077498	1937.211452
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.000000	0.022392	22.000000
25%	2.000000	0.000000	4.000000	1.000000	0.337083	0.520000	0.134950	3152.000000
50%	3.000000	1.000000	7.000000	1.000000	0.498333	0.626667	0.180975	4548.000000
75%	3.000000	1.000000	10.000000	2.000000	0.655417	0.730209	0.233214	5956.000000
max	4.000000	1.000000	12.000000	3.000000	0.861667	0.972500	0.507463	8714.000000

2.2 Predictive Modelling / Model Development

Model development is an iterative process, in which many models are derived, tested and built upon until a model fitting the desired criteria is built. Subsequent modelling work may need to begin the search at the same place as the original model building began, rather than where it finished. Here we have tried with different model and will choose the model which will provide the most accurate values.

2.2.1 Decision Tree

Decision Tree is a supervised machine learning algorithm, which is used to predict the data for classification and regression. It accepts both continuous and categorical variables. is a flowchart-like structure in which each internal node represents a “test” on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. Extremely easy to understand by the business users.

2.2.2 Random Forest

Random forest is a supervised learning algorithm. The "forest" it builds, is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result. In random forest we can go for multiple trees as higher trees will give higher no of accuracy.

2.2.3 Linear Regression

linear regression is useful for finding relationship between two continuous variables. One is predictor or independent variable and other is response or dependent variable. It looks for statistical relationship but not deterministic relationship. Relationship between two variables is said to be deterministic if one variable can be accurately expressed by the other.

Chapter 3

Conclusion

In this project the data Underwent EDA and then by applying Various machine Learning Algorithms on the Data set and by checking performances and selecting our final best model on observation of the results obtained.

3.1 Model Evaluation

3.1.1 R-squared, MAPE, RMSE

We have applied three algorithms on our dataset and calculated R-Squared, MAPE (Mean Absolute Percentage Error) and RMSE (Root Mean Square Error) values for Evaluating our models.

R-squared is the degree to which input variable explain the variation of the output. In simple words R-squared tells how much variance of dependent variable explained by the independent variable. Value of R-squared varies between 0-1, where 0 means independent variable unable to explain the target variable and 1 means target variable is completely explained by the independent variable. Higher values of R-Squared indicates better fit for model.

MAPE (Mean Absolute Percentage Error) is a prediction accuracy of a forecasting method. It measures accuracy in terms of percentage lower values which indicates a better fit for a model.

RSME (Root Mean Square Error) is a standard deviation of the errors which occur when a prediction is made on a dataset. Lower values of RSME indicates better fit.

Here, the result of each model in Python and R:

Table 3.1 Final Result in Python:

Final Result

```
1 result
```

	Model Name	MAPE_Train	MAPE_Test	R^2_train	R^2_test	RMSE_train	RMSE_test
0	Decision Tree	62.260133	36.948093	0.677563	0.646470	1080.381858	1226.219619
1	Random Forest	15.367102	20.222679	0.980750	0.885327	263.978800	698.370361
2	Linear Regression	43.781407	20.042233	0.837646	0.838132	766.631216	829.727248

Table 3.2 Final Result in R:

Model	MAPE_Train	MAPE_Test	Rsquare_Train	Rsquare_Test	Rmse_Train	Rmse_Test
Decision Tree for Regression	56.09993361	21.96538199	0.787120187	0.804121361	889.2950721	877.9629156
Random Forest	26.04445367	15.27593835	0.965882729	0.891083483	367.8864509	654.770512
Linear Regression	44.6054817	18.03292572	0.840414513	0.831046592	769.9727273	811.082504

3.2 Model Selection

From the above observations of **MAPE, R-Squared and RSME** values we conclude that the Random Forest is the best model for implementation as it has minimum value for MAPE, Maximum value for **R-Squared** and also lowest value for RSME. By Random Forest we are able to explain up to 88% to the target variable on test data. Also the MAPE values of Test and Train data do not differ a lot and this states that there is no case of overfitting in this model.

Chapter 4

4. Appendix A – R and Python Codes

4.1.1 Python Code

Project 1 - Bike Renting ¶

The objective is to predict the count of bikes rented on daily basis based on the environmental and seasonal conditions.

In [1]:

```
# Load Libraries
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from fancyimpute import KNN
from random import randrange, uniform
import seaborn as sns
from sklearn.metrics import r2_score
from scipy import stats
from matplotlib import pyplot
```

```
Using TensorFlow backend.
```

Data Pre-Processing

In [2]:

```
# Set working directory
os.chdir("F:\\DATA SCIENCE\\# Project\\1")

# Confirm working directory
os.getcwd()
```

Out[2]:

```
'F:\\DATA SCIENCE\\# Project\\1'
```

In [3]:

```
# Loading dataset
df = pd.read_csv('day.csv')
df.head()
```

Out[3]:

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	
0	1	2011-01-01	1	0	1	0	6	0	2	0.344167	0.3
1	2	2011-01-02	1	0	1	0	0	0	2	0.363478	0.3
2	3	2011-01-03	1	0	1	0	1	1	1	0.196364	0.1
3	4	2011-01-04	1	0	1	0	2	1	1	0.200000	0.2
4	5	2011-01-05	1	0	1	0	3	1	1	0.226957	0.2

Exploratory Data Analysis

In [4]:

```
df.shape
```

Out[4]:

```
(731, 16)
```

In [5]:

```
df.describe()
```

Out[5]:

	instant	season	yr	mnth	holiday	weekday	workingday	w
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	73
mean	366.000000	2.496580	0.500684	6.519836	0.028728	2.997264	0.683995	
std	211.165812	1.110807	0.500342	3.451913	0.167155	2.004787	0.465233	
min	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	
25%	183.500000	2.000000	0.000000	4.000000	0.000000	1.000000	0.000000	
50%	366.000000	3.000000	1.000000	7.000000	0.000000	3.000000	1.000000	
75%	548.500000	3.000000	1.000000	10.000000	0.000000	5.000000	1.000000	
max	731.000000	4.000000	1.000000	12.000000	1.000000	6.000000	1.000000	

In [3]:

In [6]:

```
df.dtypes
```

Out[6]:

```
instant         int64
dteday         object
season          int64
yr              int64
mnth            int64
holiday         int64
weekday         int64
workingday      int64
weathersit      int64
temp          float64
atemp         float64
hum           float64
windspeed     float64
casual          int64
registered      int64
cnt             int64
dtype: object
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['instant', 'dteday', 'season', 'yr', 'mnth', 'holiday', 'weekday',
       'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed',
       'casual', 'registered', 'cnt'],
      dtype='object')
```

In [8]:

```
df.nunique()
```

Out[8]:

```
instant       731
dteday        731
season          4
yr              2
mnth           12
holiday         2
weekday         7
workingday      2
weathersit      3
temp          499
atemp         690
hum           595
windspeed     650
casual        606
registered    679
cnt           696
dtype: int64
```

In [9]:

```
# Dropping Reductant variables
# Drop "instant" variable as it's just an index
df = df.drop(['instant'], axis = 1)

# Drop "dteday" Variable as the prediction is to be done on Seasonal basis
df = df.drop(['dteday'], axis = 1)

# Drop " casual" and "registered" variable as cnt is sum of these both
df = df.drop(['casual', 'registered'], axis = 1)
```

In [10]:

```
# Rename the Variables for better understanding
df = df.rename(columns={'yr':'year', 'mnth':'month', 'weathersit':'weather', 'temp':'t
df.columns
```

Out[10]:

```
Index(['season', 'year', 'month', 'holiday', 'weekday', 'workingday',
       'weather', 'temperature', 'atemp', 'humidity', 'windspeed', 'count'],
      dtype='object')
```

In [11]:

```
df.dtypes
```

Out[11]:

```
season           int64
year             int64
month            int64
holiday          int64
weekday          int64
workingday       int64
weather          int64
temperature    float64
atemp          float64
humidity       float64
windspeed      float64
count            int64
dtype: object
```

In [12]:

```
# Sepearting categorical and continous Variables

# categorical variables
cat_names = ['season', 'year', 'month', 'holiday', 'weekday', 'workingday', 'weather']

# continous variables
cnames = ['temperature', 'atemp', 'humidity', 'windspeed', 'count']

```

In [13]:

```
#cnames = pd.Series(['temperature', 'atemp', 'humidity', 'windspeed', 'count'])
#cnames
```

In [14]:

```
for i in cnames:
    print(df.loc[:,i].describe())
```

```
count    731.000000
mean       0.495385
std        0.183051
min        0.059130
25%        0.337083
50%        0.498333
75%        0.655417
max        0.861667
Name: temperature, dtype: float64
count    731.000000
mean       0.474354
std        0.162961
min        0.079070
25%        0.337842
50%        0.486733
75%        0.608602
max        0.840896
Name: atemp, dtype: float64
count    731.000000
mean       0.627894
std        0.142429
min        0.000000
25%        0.520000
50%        0.626667
75%        0.730209
max        0.972500
Name: humidity, dtype: float64
count    731.000000
mean       0.190486
std        0.077498
min        0.022392
25%        0.134950
50%        0.180975
75%        0.233214
max        0.507463
Name: windspeed, dtype: float64
count     731.000000
mean     4504.348837
std      1937.211452
min        22.000000
25%      3152.000000
50%      4548.000000
75%      5956.000000
max      8714.000000
Name: count, dtype: float64
```

Missing Value Analysis

In [224

```
# checking the missing values

Missing_value = df.isnull().sum().sort_values(ascending = False)
percent_missing = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([Missing_value, percent_missing], axis =1, keys =['Missing_va
missing_data
```

Out[15]:

	Missing_value	percent_missing
count	0	0.0
windspeed	0	0.0
humidity	0	0.0
atemp	0	0.0
temperature	0	0.0
weather	0	0.0
workingday	0	0.0
weekday	0	0.0
holiday	0	0.0
month	0	0.0
year	0	0.0
season	0	0.0

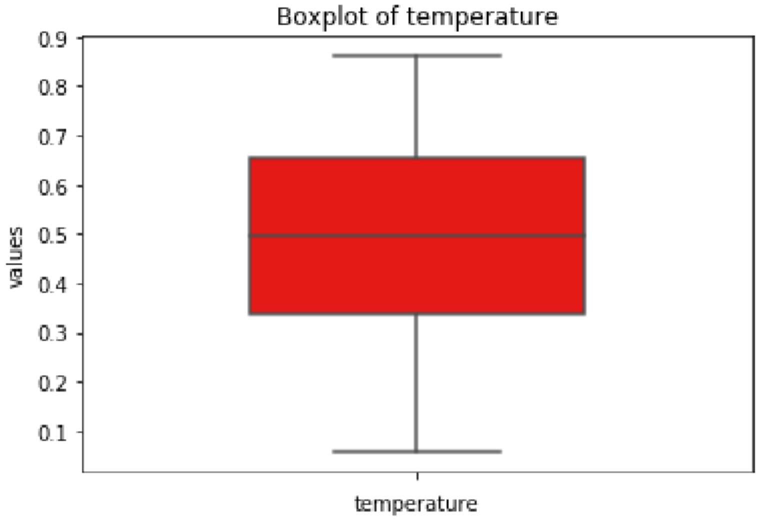
DATA DOES NOT HAVE ANY MISSING VALUES

Outliers Analysis

```
for i in cnames:
    print(i)
    sns.boxplot(y=df[i], color='r', width=0.5, saturation=0.80,  fliersize=10)
    plt.xlabel(i)
    plt.ylabel('values')
    plt.title('Boxplot of ' +i)
    plt.show()
```

In [225

temperature



atemp

From above Boxplots "humidity" and "windspeed" have outliers

```
# Calculate Lower fence, Upper fence and iqr

for i in cnames:
    print(i)
    q75,q25 = np.percentile(df.loc[:,i],[75,25])
    iqr = q75-q25
    minimum = q25-(iqr*1.5)
    maximum = q75+(iqr*1.5)
    print("MIN = "+str(minimum))
    print("MAX = "+str(maximum))
    print("IQR = "+str(iqr))

# Replace outliers with NA
df.loc[df[i]<minimum,i]=np.nan
df.loc[df[i]>maximum,i]=np.nan
```

```
temperature
MIN = -0.14041600000000015
MAX = 1.1329160000000003
IQR = 0.31833300000000001
atemp
MIN = -0.06829675000000018
MAX = 1.0147412500000002
IQR = 0.2707595000000001
humidity
MIN = 0.20468725
MAX = 1.0455212500000002
IQR = 0.21020850000000002
windspeed
MIN = -0.012446750000000034
MAX = 0.38061125
IQR = 0.0982645
count
MIN = -1054.0
MAX = 10162.0
IQR = 2804.0
```

In [226

In [18]:

```python
# Imputation of NA with median
df['humidity']=df['humidity'].fillna(df['humidity'].median())
df['windspeed']=df['windspeed'].fillna(df['windspeed'].median())
```

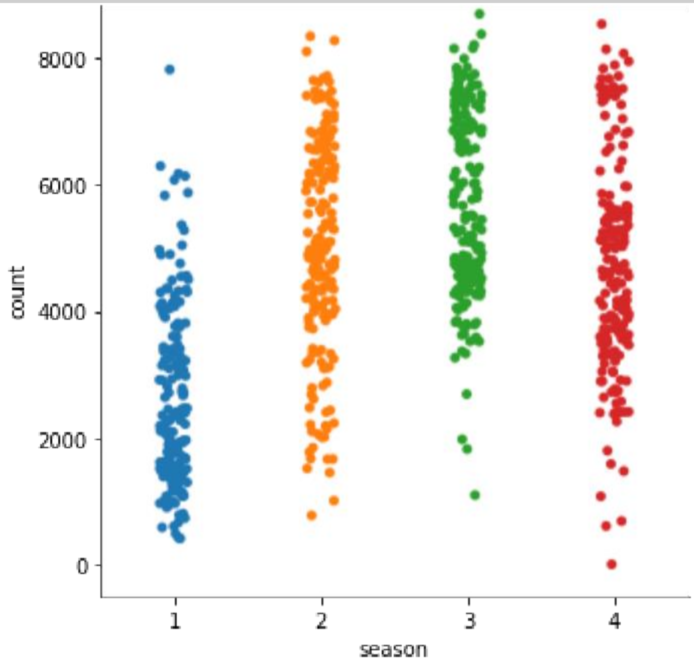
```python
# Checking NA in Data
print(df.isnull().sum())
```

```
season         0
year           0
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
dtype: int64
```

Understanding Data through Visualization

In [20]:

```python
for i in cat_names:
    sns.catplot(x=i,y='count', data=df)
    visuals = str(i)+'.pdf'
    plt.savefig(visuals)
```

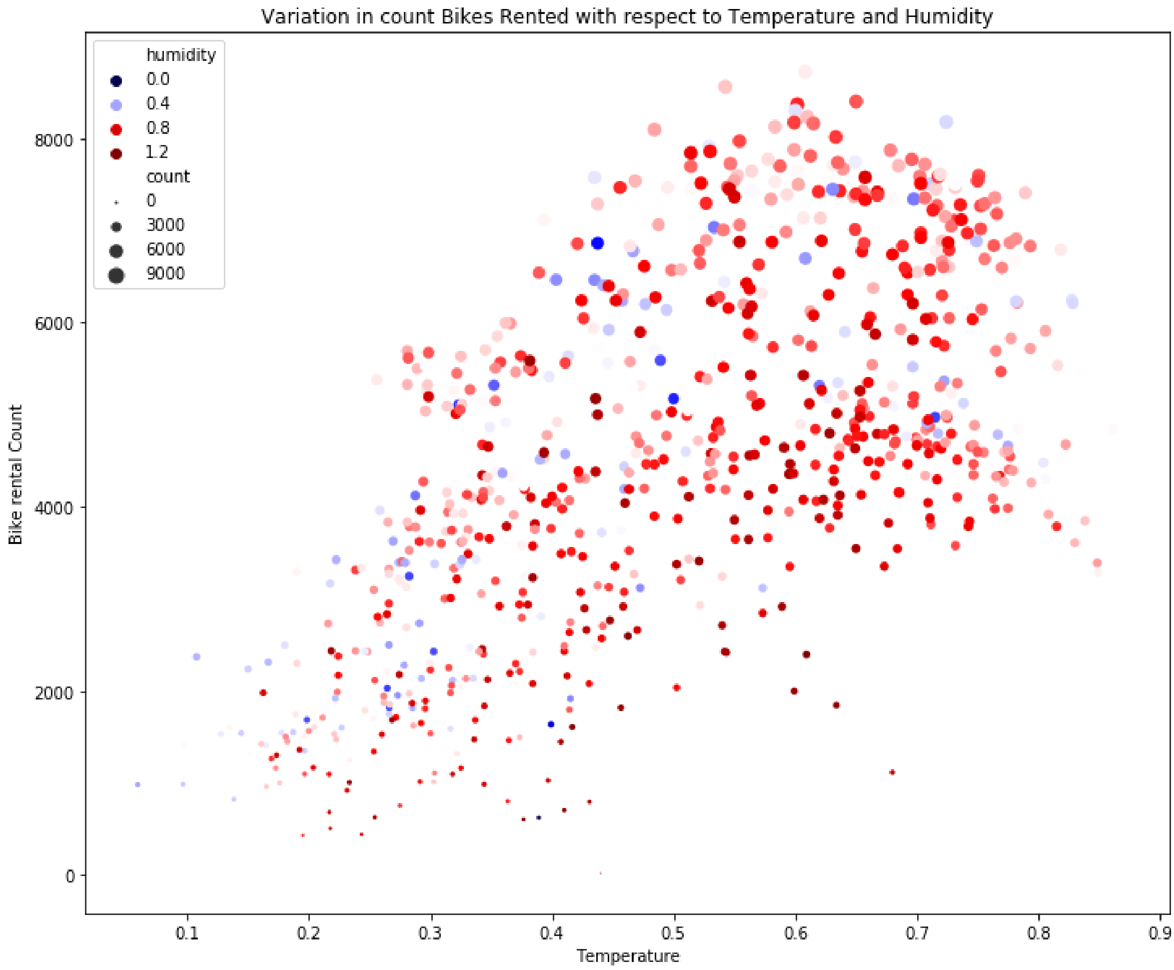


1.) season vs count : Summer, Fall and Winter has more count as compared to Spring season with almost between 4000 to 8000 bike count on daily basis 2.) year vs count : the count of bike rented increased in the

In [227
year 2012 as compared to 2011 3.) month vs count : the count goes on increases gradually from march and reaches maximum up to october and slightly decreases in november and december. 4.) holiday vs coount : the count of bike rented is much high on holidays as compared to working day 5.) week day vs count : bike count is maximum on day 5 and 6 as per the weekdays 6.) working day vs count : the count is slightly increased on week ends as compared to working days 7.) weather vs count : the count of bike rented is maximum on days having clear weather with few or partly cloudy as compared to days with mist combined with clouds and least in bad weather

In [21]:

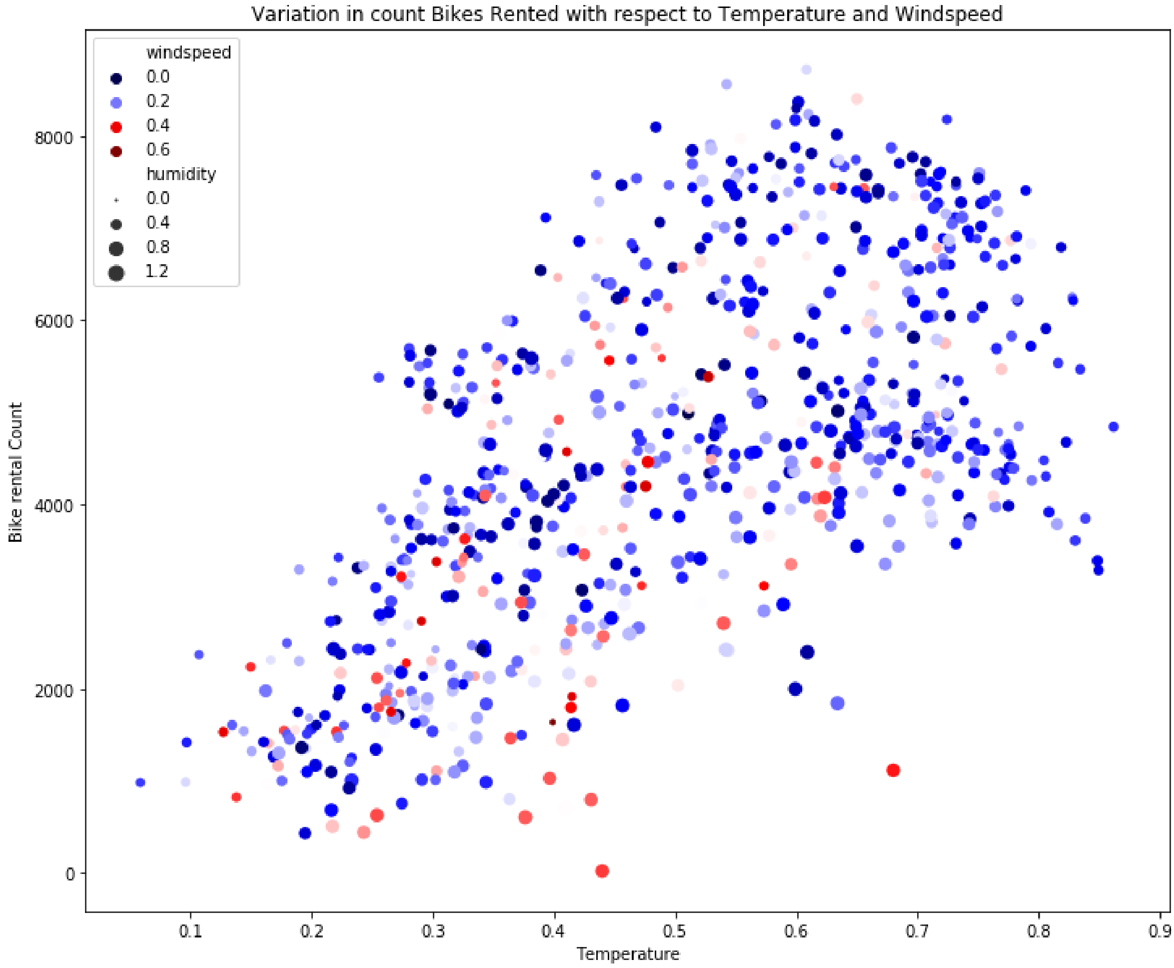
```
f, ax = plt.subplots(figsize=(12,10))
sns.scatterplot(x='temperature', y='count', hue='humidity', size='count', palette='sei
                sizes=(1,80), linewidth=0, data=df, ax=ax)
plt.title("Variation in count Bikes Rented with respect to Temperature and Humidity")
plt.ylabel("Bike rental Count")
plt.xlabel("Temperature")
plt.savefig("temp_humidity_vscount.pdf")
```



Accoring the Scatterplot ploted above the maximum no of bikes rented is found when the temperature is between scale 0.5 to 0.8 and humidity below 0.8

In [228

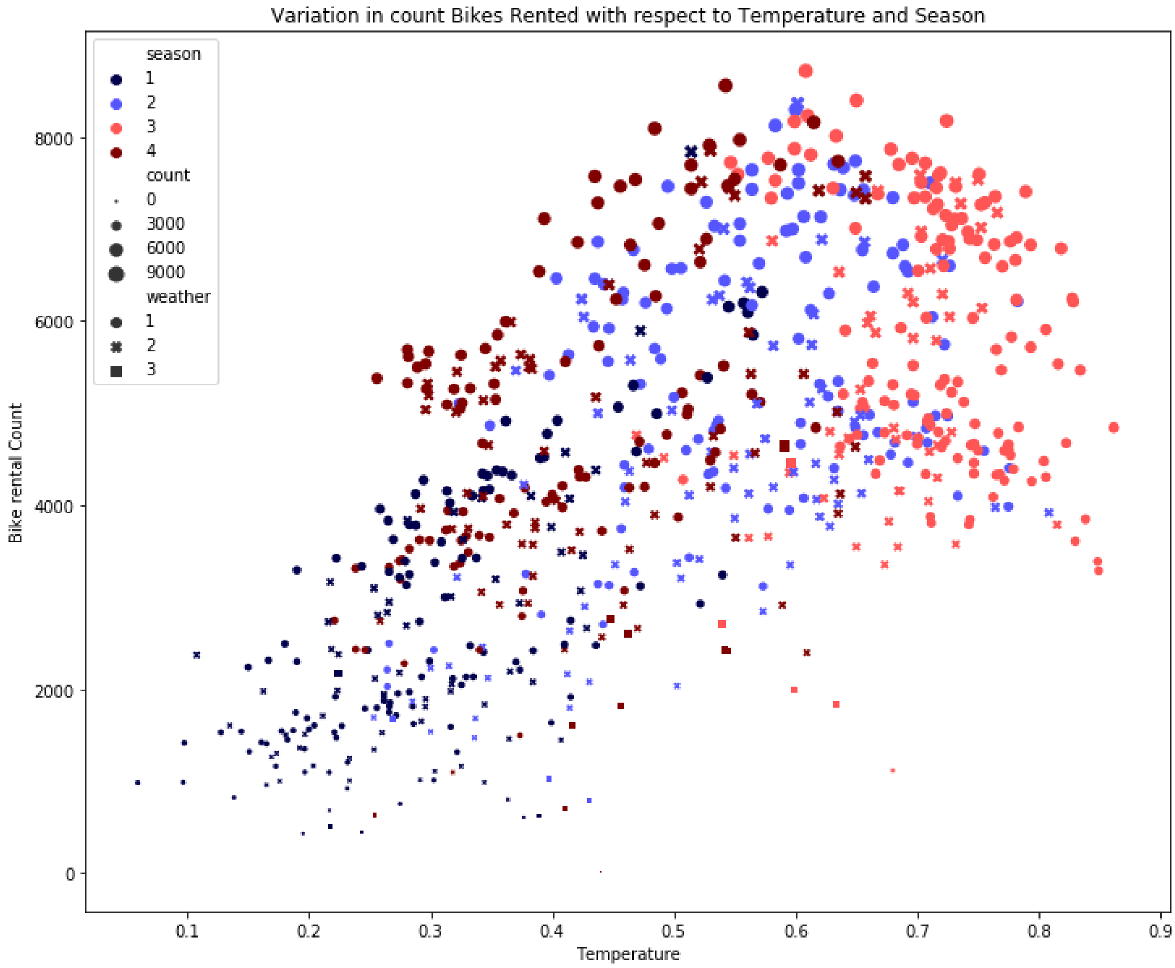
```
f, ax = plt.subplots(figsize=(12,10))
sns.scatterplot(x='temperature', y='count', hue='windspeed', size='humidity', palette=
                sizes=(1,80), linewidth=0, data=df, ax=ax)
plt.title("Variation in count Bikes Rented with respect to Temperature and Windspeed")
plt.ylabel("Bike rental Count")
plt.xlabel("Temperature")
plt.savefig("temp_wind_vscount.pdf")
```



Accoring the Scatterplot ploted above the maximum no of bikes rented is found when the temperature is between scale 0.5 to 0.8 and humidity below 0.8 and windspeed below 0.2

```
f, ax = plt.subplots(figsize=(12,10))
sns.scatterplot(x='temperature', y='count', hue='season', size='count', style='weather
                sizes=(1,80), linewidth=0, data=df, ax=ax)
plt.title("Variation in count Bikes Rented with respect to Temperature and Season")
plt.ylabel("Bike rental Count")
plt.xlabel("Temperature")
plt.savefig("temp_season_vscount.pdf")
```

In [229



Accoring the Scatterplot ploted above the maximum no of bikes rented is found during Season 1,2 and 4 when the temperature is between scale 0.5 to 0.8 and weather is 1 and 2 even in it weather 1 is more

Feature Selection

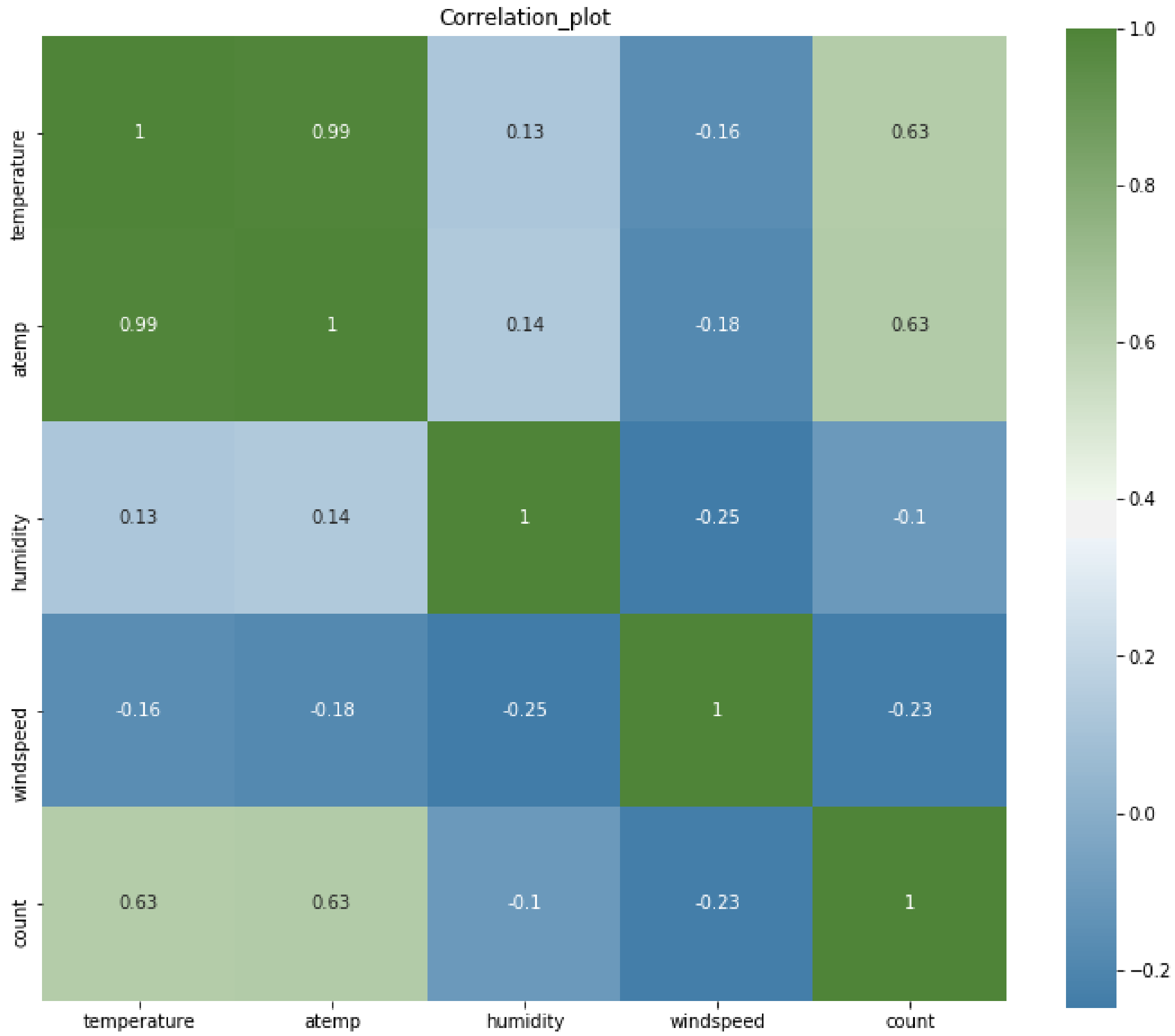
```
# Correlation analysis for numeric variables

#extracting numeric variables
df_corr = df.loc[:,cnames]

#generating correlation matrix
corr_matrix = df_corr.corr()
print(corr_matrix)
```

```
             temperature     atemp  humidity  windspeed     count
temperature     1.000000  0.991702  0.126963  -0.157944  0.627494
atemp           0.991702  1.000000  0.139988  -0.183643  0.631066
humidity        0.126963  0.139988  1.000000  -0.248489 -0.100659
windspeed      -0.157944 -0.183643 -0.248489   1.000000 -0.234545
count           0.627494  0.631066 -0.100659  -0.234545  1.000000
```

```
#correlation plot
f, ax= plt.subplots(figsize=(12,10))
sns.heatmap(corr_matrix,mask=np.zeros_like(corr_matrix,dtype=np.bool),
            cmap=sns.diverging_palette(240,120,as_cmap=True), square=True, ax=ax, anno
plt.title('Correlation_plot')
```

Out[25]:

```
Text(0.5, 1, 'Correlation_plot')
```



According to correlation plot we find that temperature and atemp are highly correlated and hence need to drop atemp variable

```
#Anova test

import statsmodels.api as sm
from statsmodels.formula.api import ols

label = 'count'
for i in cat_names:
    frame = label +' ~ '+i
    model = ols(frame, data=df).fit()
    anova = sm.stats.anova_lm(model, type=2)
    print(anova)
```

```
             df        sum_sq        mean_sq           F        PR(>F)
season 1.0 4.517974e+08 4.517974e+08 143.967653  2.133997e-30 Residual
729.0  2.287738e+09  3.138187e+06          NaN          NaN
             df        sum_sq        mean_sq           F    PR(>F) year
1.0  8.798289e+08   8.798289e+08   344.890586   2.483540e-63  Residual
729.0  1.859706e+09  2.551038e+06          NaN          NaN
             df       sum_sq       mean_sq       F       PR(>F) month
1.0 2.147445e+08 2.147445e+08  62.004625  1.243112e-14 Residual  729.0
2.524791e+09  3.463362e+06        NaN                            NaN
             df      sum_sq      mean_sq      F    PR(>F) holiday
1.0 1.279749e+07 1.279749e+07 3.421441 0.064759 Residual  729.0
2.726738e+09  3.740381e+06        NaN        NaN
             df      sum_sq      mean_sq      F    PR(>F) weekday
1.0 1.246109e+07 1.246109e+07 3.331091 0.068391 Residual  729.0
2.727074e+09  3.740843e+06        NaN        NaN
               df     sum_sq      mean_sq      F    PR(>F) workingday
1.0  1.024604e+07  1.024604e+07  2.736742  0.098495  Residual  729.0
2.729289e+09 3.743881e+06 NaN NaN
             df       sum_sq       mean_sq       F       PR(>F) weather
1.0 2.422888e+08 2.422888e+08 70.729298 2.150976e-16 Residual  729.0
2.497247e+09  3.425578e+06        NaN                            NaN
```

According to anova test we find the variables "weekday", "workingday" and "holiday" have pr > 0.05 so we need to drop them

In [27]:

```
# Droping the reductant variables
df = df.drop(['atemp', 'holiday', 'weekday', 'workingday'], axis=1)
df.shape
```

Out[27]:

(731, 8)

```
df.head()
```

Out[28]:

	season	year	month	weather	temperature	humidity	windspeed	count
0	1	0	1	2	0.344167	0.805833	0.160446	985.0
1	1	0	1	2	0.363478	0.696087	0.248539	801.0
2	1	0	1	1	0.196364	0.437273	0.248309	1349.0
3	1	0	1	1	0.200000	0.590435	0.160296	1562.0
4	1	0	1	1	0.226957	0.436957	0.186900	1600.0

In [29]:

```
df.describe()
```

Out[29]:

	season	year	month	weather	temperature	humidity	windspeed	
count	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	731.000000	
mean	2.496580	0.500684	6.519836	1.395349	0.495385	0.627894	0.190486	4
std	1.110807	0.500342	3.451913	0.544894	0.183051	0.142429	0.077498	1
min	1.000000	0.000000	1.000000	1.000000	0.059130	0.000000	0.022392	
25%	2.000000	0.000000	4.000000	1.000000	0.337083	0.520000	0.134950	3
50%	3.000000	1.000000	7.000000	1.000000	0.498333	0.626667	0.180975	4
75%	3.000000	1.000000	10.000000	2.000000	0.655417	0.730209	0.233214	5
max	4.000000	1.000000	12.000000	3.000000	0.861667	0.972500	0.507463	8

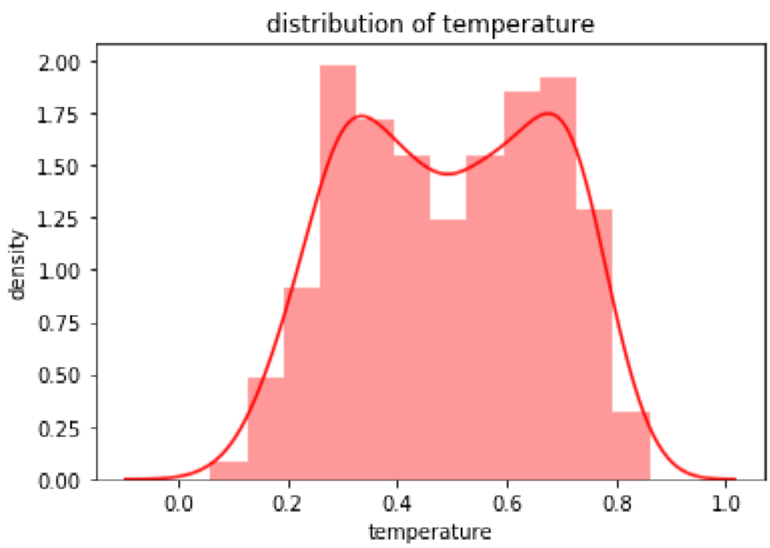
In [30]:

```
# Updating Numeric and categorical variables
# categorical variables
cat_names=['season', 'year', 'month', 'weather']

# Numerical variables
cnames=['temperature', 'humidity', 'windspeed', 'count']
```

Feature Scaling

```
# Distribution plot to check weather the data is normalised or not

for i in cnames:
    print(i)
    sns.distplot(df[i], bins='auto', color='Red')
    plt.title('distribution of '+i)
    plt.ylabel('density')
    plt.show()
```

temperature



humidity

According to distribution plot all data is Normalized

Train-Test-Split

In [32]:

```
df1=df
#df=df1
```

In [33]:

```
df = pd.get_dummies(df,columns=cat_names)
df.shape
```

Out[33]:

```
(731, 25)
```

```
df.head()
```

Out[34]:

	temperature	humidity	windspeed	count	season_1	season_2	season_3	season_4	year_
0	0.344167	0.805833	0.160446	985.0	1	0	0	0	
1	0.363478	0.696087	0.248539	801.0	1	0	0	0	
2	0.196364	0.437273	0.248309	1349.0	1	0	0	0	
3	0.200000	0.590435	0.160296	1562.0	1	0	0	0	
4	0.226957	0.436957	0.186900	1600.0	1	0	0	0	

5 rows × 25 columns

In [40]:

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import metrics
```

In [36]:

```
# Error Metrics
def MAPE(y_true, y_prediction):
    mape = np.mean(np.abs(y_true - y_prediction)/y_true)*100
    return mape
```

In [37]:

```
# Split data into predictor and Target
X = df.drop(['count'], axis=1)
```

```
Y = df['count']
```

In [38]:

```
# Dividing data into train-test
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = .20, random_state=
```

Decision Tree

```
# import libraries
from sklearn.tree import DecisionTreeRegressor

DT_model = DecisionTreeRegressor(max_depth=2).fit(X_train,Y_train)

# prediction on train data
DT_train = DT_model.predict(X_train)

# Prediction on test data
DT_test = DT_model.predict(X_test)

# performance on train data
MAPE_train = MAPE(Y_train, DT_train)

# performance on test data
MAPE_test = MAPE(Y_test, DT_test)

# r2 value for train data
r2_train = r2_score(Y_train, DT_train)

# r2 value for test data
r2_test = r2_score(Y_test, DT_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(Y_train, DT_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(Y_test, DT_test))

print('Mean Absolute Percentage Error for train data = '+str(MAPE_train))
print('Mean Absolute Percentage Error for test data = '+str(MAPE_test))
print('R^2_score for train data = '+str(r2_train))
print('R^2_score for test data = '+str(r2_test))
print("RMSE for train data="+str(RMSE_train))
print("RMSE for test data="+str(RMSE_test))
```

```
Mean Absolute Percentage Error for train data = 62.26013293672567
Mean Absolute Percentage Error for test data = 36.94809301452646
R^2_score for train data = 0.6775629218593628
R^2_score for test data = 0.6464697716428666
RMSE for train data=1080.3818579492188
RMSE for test data=1226.2196190864843
```

In [43]:

```
DT1 = {'Model Name': ['Decision Tree'], 'MAPE_Train':[MAPE_train], 'MAPE_Test':[MAPE_t
       'R^2_train':[r2_train], 'R^2_test':[r2_test], 'RMSE_train':[RMSE_train], 'RMSE_
result1 = pd.DataFrame(DT1)
```

Random Forest

```
# Import Libraries
from sklearn.ensemble import RandomForestRegressor

RF_model = RandomForestRegressor(n_estimators=100).fit(X_train,Y_train)

# prediction on train data
RF_train = RF_model.predict(X_train)

# Prediction on test data
RF_test = RF_model.predict(X_test)

# performance on train data
MAPE_train = MAPE(Y_train, RF_train)

# performance on test data
MAPE_test = MAPE(Y_test, RF_test)

# r2 value for train data
r2_train = r2_score(Y_train, RF_train)

# r2 value for test data
r2_test = r2_score(Y_test, RF_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(Y_train,RF_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(Y_test,RF_test))

print('Mean Absolute Percentage Error for train data = '+str(MAPE_train))
print('Mean Absolute Percentage Error for test data = '+str(MAPE_test))
print('R^2_score for train data = '+str(r2_train))
print('R^2_score for test data = '+str(r2_test))
print("RMSE for train data="+str (RMSE_train))
print("RMSE for test data="+str(RMSE_test))
```

```
Mean Absolute Percentage Error for train data = 15.367102161378016
Mean Absolute Percentage Error for test data = 20.22267871052817
R^2_score for train data = 0.9807500985169355
R^2_score for test data = 0.8853268793110791
RMSE for train data=263.97879975165034
RMSE for test data=698.3703605557162
```

In [45]:

```
RF1 = {'Model Name': ['Random Forest'], 'MAPE_Train':[MAPE_train], 'MAPE_Test':[MAPE_t
       'R^2_train':[r2_train], 'R^2_test':[r2_test], 'RMSE_train':[RMSE_train], 'RMSE_
result2 = pd.DataFrame(RF1)
```

In [46]:

```
result = result1.append(result2)
```

Linear Regression Model

```
#import Libraries
import statsmodels.api as sm

LR_model = sm.OLS(Y_train,X_train).fit()
print(LR_model.summary())
```

```
                            OLS Regression Results
==========================================================================
====
Dep. Variable:                  count   R-squared:
0.838
Model:                            OLS   Adj. R-squared:
0.832
Method:                 Least Squares   F-statistic:                     1
45.2
Date:                Mon, 29 Jun 2020   Prob (F-statistic):          4.07e
-207
Time:                        14:54:04   Log-Likelihood:                -47
07.6
No. Observations:                 584   AIC:                             9
457.
Df Residuals:                     563   BIC:                             9
549.
Df Model:                          20
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
temperature  4861.4866    470.566     10.331      0.000    3937.208     578
5.765
humidity    -2046.1090    349.646     -5.852      0.000   -2732.879    -135
9.339
windspeed   -3183.7171    471.041     -6.759      0.000   -4108.929    -225
8.506
season_1      -95.6917    147.485     -0.649      0.517    -385.381      19
3.997
season_2      797.5360    147.540      5.406      0.000     507.739     108
7.333
season_3      802.4655    167.561      4.789      0.000     473.344     113
1.587
season_4     1448.3627    167.244      8.660      0.000    1119.865     177
6.860
year_0        510.7429    150.333      3.397      0.001     215.461      80
6.024
year_1       2441.9296    149.063     16.382      0.000    2149.142     273
4.717
month_1         0.3954    194.760      0.002      0.998    -382.149      38
2.940
month_2        90.7510    184.500      0.492      0.623    -271.641      45
3.143
```

```
month_3       535.4258    139.833      3.829      0.000     260.768      81
0.083
month_4       268.2218    171.503      1.564      0.118     -68.641      60
5.085
month_5       655.5705    180.429      3.633      0.000     301.175     100
9.966
month_6       229.3232    177.314      1.293      0.196    -118.954      57
7.601
month_7      -239.9952    217.756     -1.102      0.271    -667.709      18
7.719
month_8       268.0775    203.921      1.315      0.189    -132.462      66
8.617
month_9       900.1744    171.404      5.252      0.000     563.505     123
6.844
month_10      439.6648    185.085      2.375      0.018      76.124      80
3.206
month_11     -149.3217    192.040     -0.778      0.437    -526.524      22
7.881
month_12      -45.6150    166.007     -0.275      0.784    -371.684      28
0.454
weather_1    1673.8316     89.662     18.668      0.000    1497.718     184
9.945
weather_2    1359.3541    109.137     12.456      0.000    1144.989     157
3.719
weather_3     -80.5132    217.874     -0.370      0.712    -508.458      34
7.432
==========================================================================
====
Omnibus:                       99.544   Durbin-Watson:
1.896
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             26
6.265
Skew:                          -0.852   Prob(JB):                   1.52
e-58
Kurtosis:                       5.836   Cond. No.                   1.82
e+16
==========================================================================
====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is cor rectly specified.
[2] The smallest eigenvalue is 3.58e-30. This might indicate that there ar e
strong multicollinearity problems or that the design matrix is singular.
```

```
# prediction on train data
LR_train = LR_model.predict(X_train)

# Prediction on test data
LR_test = LR_model.predict(X_test)

# performance on train data
MAPE_train = MAPE(Y_train, LR_train)

# performance on test data
MAPE_test = MAPE(Y_test, LR_test)

# r2 value for train data
r2_train = r2_score(Y_train, LR_train)

# r2 value for test data
r2_test = r2_score(Y_test, LR_test)

# RMSE value for train data
RMSE_train = np.sqrt(metrics.mean_squared_error(Y_train,LR_train))

# RMSE value for test data
RMSE_test = np.sqrt(metrics.mean_squared_error(Y_test,LR_test))

print('Mean Absolute Percentage Error for train data = '+str(MAPE_train))
print('Mean Absolute Percentage Error for test data = '+str(MAPE_test))
print('R^2_score for train data = '+str(r2_train))
print('R^2_score for test data = '+str(r2_test))
print("RMSE for train data="+str (RMSE_train))
print("RMSE for test data="+str(RMSE_test))
```

```
Mean Absolute Percentage Error for train data =
43.78140724487472 Mean Absolute Percentage Error
for test data = 20.042233295431416 R^2_score for
train data = 0.8376458444602071
R^2_score for test data =
0.838132097806852 RMSE for
train
data=766.6312156108335
RMSE for test data=829.7272484647344
```

In [49]:

```
LR1 = {'Model Name': ['Linear Regression'], 'MAPE_Train':[MAPE_train], 'MAPE_Test':[
       'R^2_train':[r2_train], 'R^2_test':[r2_test], 'RMSE_train':[RMSE_train], 'RMS
result3 = pd.DataFrame(LR1)
```

In [50]:

```
result = result.append(result3)
```

In [51]:

```
result = result.reset_index(drop=True)
```

Final Result

```
result
```

Out[52]:

	Model Name	MAPE_Train	MAPE_Test	R^2_train	R^2_test	RMSE_train	RMSE_test
0	Decision Tree	62.260133	36.948093	0.677563	0.646470	1080.381858	1226.219619
1	Random Forest	15.367102	20.222679	0.980750	0.885327	263.978800	698.370361
2	Linear Regression	43.781407	20.042233	0.837646	0.838132	766.631216	829.727248

According to the results of all the models, observing all MAPE and R^2 values we conclude that Random Forest has minimum MAPE value (20.042) and its R^2 value is maximum (0.83) and RSME value is (698.37). therefore selected as best model among above.

4.1.2 R Code

```
#clear Environment

rm(list = ls())

# set Working Directory

setwd("F:/DATA SCIENCE/# Project/1")

# confirm working directory

getwd()

#Load Data

df = read.csv('day.csv')

###################################### Exploratory Data Analysis

class(df)
head(df)
dim(df)
names(df)
str(df)
summary(df)

#remove "instant" variable as its just index and "dteday" as we need to predict count on
#seasonal basis not on date basis and also drop "casual" and "registered" as count is sum of
this two

df = subset(df,select = -c(instant,dteday,casual, registered))
names(df)

############################ Rename the variables for better understanding

names(df)[2]="year"
names(df)[3]="month"
names(df)[7]="weather"
names(df)[8]="temperature"
names(df)[10]="humidity"
names(df)[12]="count"

names(df)

############################ Seperating categorical and numerical variables

#categorical variables

cat_names=c('season','year','month','holiday','weekday','workingday','weather')
```

```
#numerical variables

cnames=c('temperature', 'atemp','humidity', 'windspeed','count')

################################ Data Pre-processing

#checking for missing values

sum(is.na(df))

#--> there's no missing value in the Data

df1=df
#df=df1

###############################  Outlier Analysis

library(ggplot2)

for (i in 1:length(cnames)) {
  assign(paste0("as",i), ggplot(aes_string(y = (cnames[i]), x = "count"), data = subset(df))+
           stat_boxplot(geom = "errorbar", width = 0.5)+
           geom_boxplot(outlier.colour = "red", fill = "green", outlier.shape = 18,
                        outlier.size = 2, notch = FALSE)+
           theme(legend.position = "bottom")+
           labs(y=cnames[i],x="Bike Count")+
           ggtitle(paste("Boxplot for Count of Bikes with", cnames[i])))

}

gridExtra::grid.arrange(as1,as2,as3,as4,as5,ncol=2)

#--> According to the outlier Analysis varibles "windspeed" and "humidity" shows outliers

# removing outliers by capping upper fence and lower fence values

for(i in cnames){
  print(i)
  #Quartiles
  Q1 = quantile(df[,i],0.25)
  Q3 = quantile(df[,i],0.75)

  #Inter quartile range
  IQR = Q3-Q1

  # Upperfence and Lower fence values
  UL = Q3 + (1.5*IQR(df[,i]))
  LL = Q1 - (1.5*IQR(df[,i]))
```

```
  # No of outliers and inliers in variables
  No_outliers = length(df[df[,i] > UL,i])
  No_inliers = length(df[df[,i] < LL,i])

  # Capping with upper and inner fence values
  df[df[,i] > UL,i] = UL
  df[df[,i] < LL,i] = LL

}

# plotting boxplots after removing outiers

for(i in 1:length(cnames))
{
  assign(paste0("zx",i),ggplot(aes_string(y=(cnames[i]),x = 'count'), data=subset(df))+
         stat_boxplot(geom = "errorbar",width = 0.5) +
         geom_boxplot(outlier.color = "red",fill="green",
                      outlier.shape = 18,outlier.size = 1,notch = FALSE)+
         theme(legend.position = "bottom")+
         labs(y = cnames[i],x='count')+
         ggtitle(paste("boxplot of count for",cnames[i])))
}

gridExtra::grid.arrange(zx1,zx2,zx3,zx4,zx5,ncol = 2)

#--> from this boxplots the variables are free from outliers

########################## understanding Data using visualization

# checking impact of categorical variables on count

for(i in 1:length(cat_names))
{
  assign(paste0("b",i),ggplot(aes_string(y='count',x = (cat_names[i])),
                              data=subset(df))+
         geom_bar(stat = "identity",fill = "DarkSlateBlue") +
         # labs(title = "Scatter Plot of count vs", x = (cnames[i]), y = "count")+
         ggtitle(paste("Number of bikes rented with respect to",cat_names[i])))+
    theme(axis.text.x = element_text( color="black", size=8))+
    theme(plot.title = element_text(face = "bold"))
}

# using library(gridExtra)
gridExtra::grid.arrange(b1,b2,b3,b4,ncol = 2)
gridExtra::grid.arrange(b5,b6,b7,ncol = 2)

#--> From barplot we can observe below points

#1.) season vs count : Summer, Fall and Winter has more count as compared to Spring season
with
```

```
#    almost between 4000 to 8000 bike count on daily basis
#2.) year vs count : the count of bike rented increased in the year 2012 as compared to 2011
#3.) month vs count : the count goes on increases gradually from march and reaches maximum up to
#    october and slightly decreases in november and december.
#4.) holiday vs count : the count of bike rented is much high on holidays as compared to working day
#5.) week day vs count : bike count is maximum on day 5 and 6 as per the weekdays
#6.) working day vs count : the count is slightly increased on week ends as compared to working days
#7.) weather vs count : the count of bike rented is maximum on days having clear weather with few or
#    partly cloudy as compared to days with mist combined with clouds and least in bad weather

####################################################################################################
# Bikes rented with respect to Working day

ggplot(df, aes(x= reorder(weekday, -count), y = count))+
  geom_bar(stat = "identity", fill = "blue")+
  labs(title = "No. of Bikes rented vs weekdays", x = "days of week")+
  theme(panel.background = element_rect("pink"))+
  theme(plot.title = element_text(face = "bold"))
#--> maximum bike rented on day 5 and 4 and least on day 0

# Bikes rented with respect to temp and humidity

ggplot(df,aes(temperature,count)) +
  geom_point(aes(color=humidity),alpha=0.5) +
  labs(title = "Bikes rented with respect to variation in temperature and humidity", x = "temperature")+ theme_bw()
#--> maximum bike rented between temp 0.50 to 0.75 and humidity 0.50 to 0.80


# Bikes rented with respect to temp and windspeed

ggplot(df, aes(x = temperature, y = count))+
  geom_point(aes(color=windspeed))+
  labs(title = "Bikes rented with respect to temperature and windspeed", x = "temperature")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
#--> maximum bike rented temperature between 0.50 to 0.75 and windspeed below 0.2

# Bikes rented with respect to temp and season

ggplot(df, aes(x = temperature, y = count))+
  geom_point(aes(color=season))+
  labs(title = "Bikes rented with respect to temperature and season", x = "temperature")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
```

```
  theme_bw()
#--> maximum bike rented between temperature 0.5 to 0.75 and for season 2 and 3

############################ Feature Selection
df1 = df
df = df1

# correlation analysis by plotting correlation plot

library(corrgram)
corrgram(df[,cnames],order = F,upper.panel = panel.pie,
         text.panel = panel.txt,main="Correlation plot for numeric variables")

#-->According to correlation analysis temp and atemp variables are highly correlated
therefore we drop atemp variable

# Anova analysis for categorical variable with target numeric variable

for(i in cat_names){
  print(i)
  Anova_result= summary(aov(formula = count~ df[,i],df))
  print(Anova_result)
}

#--> According to Anova analysis variables "holiday","weekday" and "workingday" have p
value > 0.05
#therefore we drop them as well.

# Dimension reduction

df = subset(df,select = -c(atemp,holiday,weekday,workingday))


# our data after dimension reduction

summary(df)

head(df)

# updating  continous and categorical variables after dimension reduction

# Continuous variable
cnames= c('temperature','humidity', 'windspeed', 'count')

# Categorical variables
cat_names = c('season', 'year', 'month','weather')

######################################## Feature Scaling

#checking distribution of each continuous variables
```

```
for(i in 1:length(cnames))
{
  assign(paste0("h",i),ggplot(aes_string(x=(cnames[i])),
                             data=subset(df))+
           geom_histogram(fill="darkslateblue",colour = "black")+geom_density()+
           scale_y_continuous(breaks =scales::pretty_breaks(15))+
           scale_x_continuous(breaks = scales::pretty_breaks(15))+
           theme_bw()+xlab(cnames[i])+ylab("Frequency")+
           ggtitle(paste("distribution of ",cnames[i])))
}

gridExtra::grid.arrange(h1,h2,h3,h4,ncol = 2)

#--> According to distribution plot all data is Normalized

# saving the pre_processed data
write.csv(df, "bike_rental_data.csv", row.names = FALSE)

####################################### Model Development

# cleaning R Environment
library(DataCombine)
rmExcept("df")

# copy data
df=df1
df1=df

# Function for Error metrics to calculate performance of model
mape = function(y,y1){
  mean(abs((y-y1)/y))*100
}

# Function for r2 to calculate the goodness of fit of model
rsquare=function(y,y1){
  cor(y,y1)^2
}

# Function for RMSE value
rmse = function(y,y1){
  difference = y - y1
  root_mean_square = sqrt(mean(difference^2))
  print(root_mean_square)
}

# calling Categorical varaibles
cat_names= c("season","year","month","weather")

# creating dummy variables using dummies library
```

```
library(dummies)
df = dummy.data.frame(df,cat_names)

dim(df)
head(df)
#--> hence dummy data set is created

# Dividing data into train and test sets

library(caTools)
set.seed(123)
sample = sample.split(df, SplitRatio = 0.80)
train1 = subset(df, sample == TRUE)
test1 = subset(df, sample == FALSE)

############################## Decision Tree for Regression
# Model Development on train data
library(rpart)

DT_model = rpart(count~., train1,method = "anova")
DT_model

# Prediction on train data
DT_train= predict(DT_model,train1[-25])

# Prediction on test data
DT_test= predict(DT_model,test1[-25])

# MAPE For train data
DT_MAPE_Train = mape(train1[,25],DT_train)#56.09
#--> DT_MAPE_Train = 56.09

# MAPE For train data test data
DT_MAPE_Test = mape(test1[,25],DT_test)
#--> DT_MAPE_Test = 21.96

# Rsquare  For train data
DT_r2_train = rsquare(train1[,25],DT_train)
#--> DT_r2_train = 0.7871

# Rsquare For test data
DT_r2_test = rsquare(test1[,25],DT_test)
#--> DT_r2_test = 0.8041

# rmse For train data
DT_rmse_train = rmse(train1[,25],DT_train)
#--> DT_rmse_train = 889.295

# rmse For test data
DT_rmse_test = rmse(test1[,25],DT_test)
```

```
#--> DT_rmse_test = 877.962

###################################### Random Forest for Regression

# Model Development on Train data using randomForest library
RF_model= randomForest(count~.,train1,ntree=100,method="anova")

RF_model = randomForest::randomForest(count~.,train1,ntree=100, method="anova")

# Prediction on train data
RF_train= predict(RF_model,train1[-25])

# Prediction on test data
RF_test = predict(RF_model,test1[-25])

# MAPE For train data
RF_MAPE_Train = mape(train1[,25],RF_train)
#--> RF_MAPE_Train = 26.04

# MAPE For test data
RF_MAPE_Test = mape(test1[,25],RF_test)
#--> RF_MAPE_Test = 15.27

# Rsquare  For train data
RF_r2_train=rsquare(train1[,25],RF_train)
#--> RF_r2_train = 0.9658

# Rsquare For test data
RF_r2_test=rsquare(test1[,25],RF_test)
#--> RF_r2_test = 0.891

# rmse For train data
RF_rmse_train = rmse(train1[,25],RF_train)
#--> RF_rmse_train = 367.886

# rmse For test data
RF_rmse_test = rmse(test1[,25],RF_test)
#--> RF_rmse_test = 654.77

###################################### Linear Regression model

# Recalling numeric Variables to check variance inflation factor for Multicollinearity
cnames= c("temperature","humidity","windspeed")
numeric_data= df[,cnames]

# VIF test  using usdm library
library(usdm)
vifcor(numeric_data,th=0.6)
#--> value of VIF of all variables are almost 1 so there's no multicollinearity issue.
```

```
# Linear Regression model development
LR_model = lm(count ~.,data = train1)
summary(LR_model)

# prediction on train data
LR_train = predict(LR_model,train1[,-25])

# prediction on test data
LR_test= predict(LR_model,test1[-25])

# MAPE For train data
LR_MAPE_Train = mape(train1[,25],LR_train)
#--> LR_MAPE_Train = 44.605

# MAPE For test data
LR_MAPE_Test = mape(test1[,25],LR_test)
#--> LR_MAPE_Test = 18.032

# Rsquare For train data
LR_r2_train = rsquare(train1[,25],LR_train)
#--> LR_r2_train = 0.84

# Rsquare For test data
LR_r2_test = rsquare(test1[,25],LR_test)
#--> LR_r2_test = 0.83

# rmse For train data
LR_rmse_train = rmse(train1[,25],LR_train)
#--> LR_rmse_train = 769.97

# rmse For test data
LR_rmse_test = rmse(test1[,25],LR_test)
#--> LR_rmse_test = 811.08

############################################### Results

Model = c('Decision Tree for Regression', 'Random Forest', 'Linear Regression')

MAPE_Train = c(DT_MAPE_Train, RF_MAPE_Train, LR_MAPE_Train)

MAPE_Test = c(DT_MAPE_Test, RF_MAPE_Test, LR_MAPE_Test)

Rsquare_Train = c(DT_r2_train, RF_r2_train, LR_r2_train)

Rsquare_Test = c(DT_r2_test, RF_r2_test, LR_r2_test)

Rmse_Train = c(DT_rmse_train, RF_rmse_train, LR_rmse_train)

Rmse_Test = c(DT_rmse_test, RF_rmse_test, LR_rmse_test)
```

```
Final_results = data.frame(Model,MAPE_Train,MAPE_Test,Rsquare_Train,
                           Rsquare_Test,Rmse_Train,Rmse_Test)

Final_results
# Saving the Final Output
write.csv(Final_results, "Final_results.csv", row.names = FALSE)

# From above results Random Forest model have optimum values and hence best model.
```

Chapter 5

5. Appendix B – References

1. edWisor Learning: https://learning.edwisor.com/
2. Coursera - Machine Learning: https://www.coursera.org/learn/machine-learning/
3. https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052