BIKE RENTAL COUNT PREDICTION

MOULEESHWARAN.S

22.09.2019

TABLE OF CONTENTS

1.INTRODUCTION

1.1 PROBLEM STATEMENT:
In our project we need to predict bike rental count on daily basis based on the season and environmental settings.

1.2 DATA OVERVIEW:

We have 16 variables and 731 observations. In that 13 variables are independent and 3 dependent variables.

Lets have a look at the data:

In [7]:
```
#lets see first five observations of our data
Bike_Data.head()
```

Out[7]:

	index	date	season	year	month	holiday	weekday	workingday	weather	temperature	atemp	humidity	windspeed	casual	registered	count
0	1	01-01-2011	1	0	1	0	6	0	2	0.344167	0.363625	0.805833	0.160446	331	654	985
1	2	02-01-2011	1	0	1	0	0	0	2	0.363478	0.353739	0.696087	0.248539	131	670	801
2	3	03-01-2011	1	0	1	0	1	1	1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
3	4	04-01-2011	1	0	1	0	2	1	1	0.200000	0.212122	0.590435	0.160296	108	1454	1562
4	5	05-01-2011	1	0	1	0	3	1	1	0.226957	0.229270	0.436957	0.186900	82	1518	1600

In [8]:
```
# lets see last five observations of our data
Bike_Data.tail()
```

Out[8]:

	index	date	season	year	month	holiday	weekday	workingday	weather	temperature	atemp	humidity	windspeed	casual	registered	count
726	727	27-12-2012	1	1	12	0	4	1	2	0.254167	0.226642	0.652917	0.350133	247	1867	2114
727	728	28-12-2012	1	1	12	0	5	1	2	0.253333	0.255046	0.590000	0.155471	644	2451	3095
728	729	29-12-2012	1	1	12	0	6	0	2	0.253333	0.242400	0.752917	0.124383	159	1182	1341
729	730	30-12-2012	1	1	12	0	0	0	1	0.255833	0.231700	0.483333	0.350754	364	1432	1796
730	731	31-12-2012	1	1	12	0	1	1	2	0.215833	0.223487	0.577500	0.154846	439	2290	2729

Here casual,registered and count are our dependent variables

COUNT = CASUAL+REGISTERED

Remaining all are independent variables.

2. METHODOLOGY

2.1 DATA PRE-PROCESSING:

Data preprocessing is a data mining techniques which transforms raw data into an understandable format.data goes through series of steps during preprocessing. They are data cleaning,data visualization,data transformation,data reduction.

2.1.1 DATA EXPLORATION:

We need to check dimensions of the data, data types of the data, summary of the data .so that we can get good understandings about the data and also identify the target variable.

```
# Check names of dataset
Bike_Data.columns

# Rename variables in dataset
Bike_Data = Bike_Data.rename(columns = {'instant':'index','dteday':'date','yr':'year','mnth':'month','weathersit':'weather',
                                        'temp':'temperature','hum':'humidity','cnt':'count'})

Bike_Data.columns
```

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

For our convineance, I changed some shortcut variable name into understandable format.the above picture is self-explanatory

2.1.2 MISSING VALUE ANALYSIS:

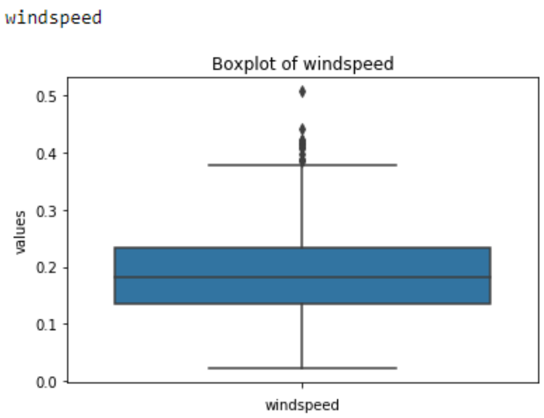
Missing values are the data which is not present in the particular variable or observations. It may happen due to human error, or it may mark as an optional during the survey. If the data set contains missing values which is above 30%, either we need to drop the column or that particular observation.in our dataset we don't have any missing values but in real world problems there is always some missing values. We need to impute those missing values either it is classification or regression problems.

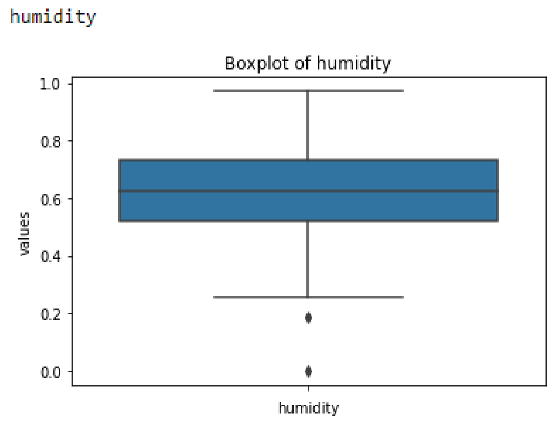
```
season         0
year           0
month          0
holiday        0
weekday        0
workingday     0
weather        0
temperature    0
atemp          0
humidity       0
windspeed      0
count          0
dtype: int64
```

2.1.3 OUTLIER ANALYSIS:

Basically outliers are the values which are lying far away from the remaining variables which may lead biased towards the higher value which results in the performance of our model. So that we need to treat the outliers .

Here outliers are detected using boxplot. We have inliers in humidity and outliers in windspeed other than that we don't have any outliers.so, In our case we saved minimum value to the inliers and maximum values to the outliers.so that we no need to loss the data and also we can increase the performance the of our model. How much data we feed is that much accuracy to our model.



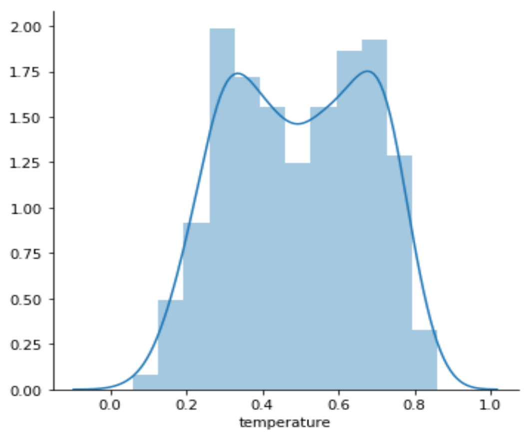


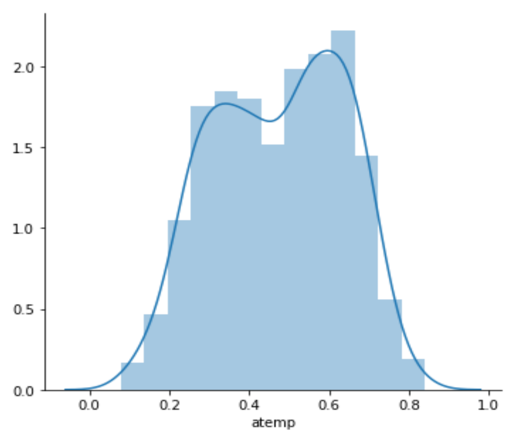
2.1.4 DATA VISUALIZATION:

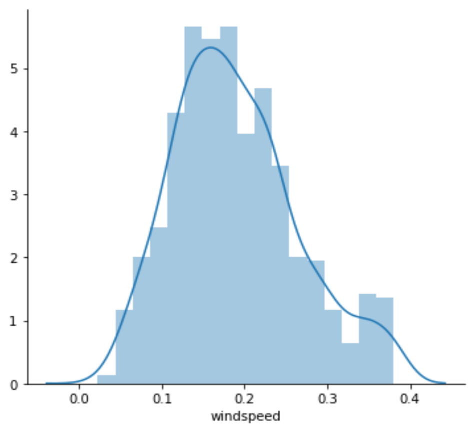
Data visualization is the easy method to understand the data. It will gives clear idea of our data and also impact of dependent variables.

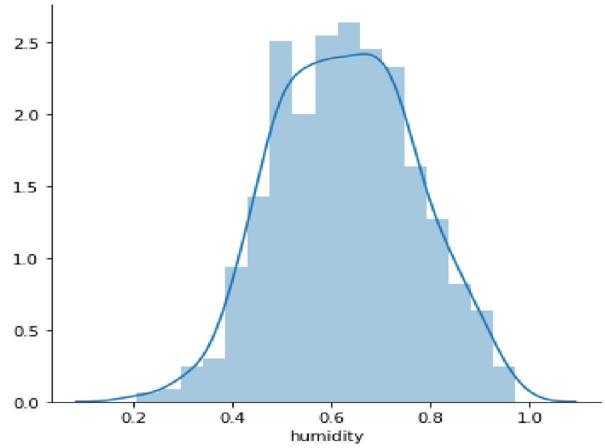
2.1.4a:DISTRIBUTION OF THE NUMERIC VARIABLE:

Distributution plot helps us to know the distribution of the data and makes us easily understandable. So that it used in the bothR and Python languages.here we used histogram for our visualization.







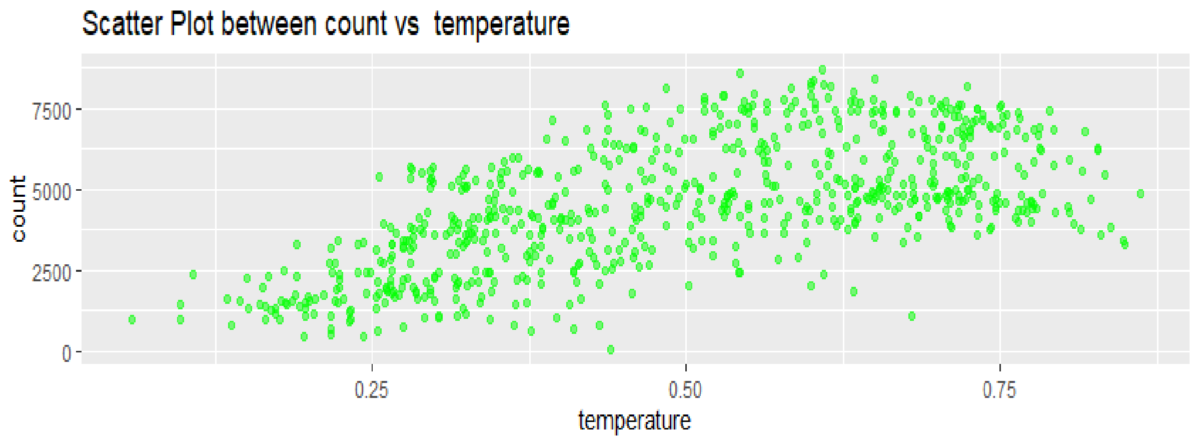


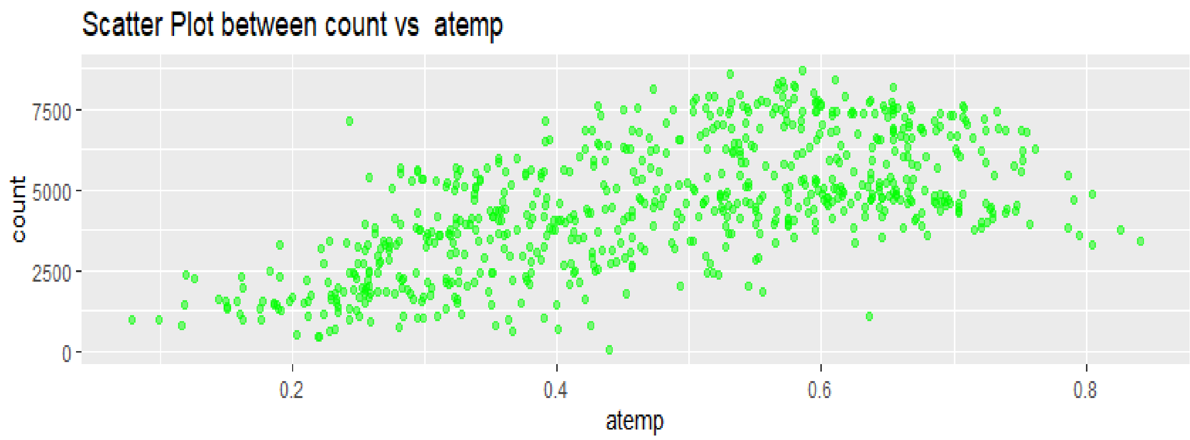
Based on the above graphs, we can clearly says that our temperature and atemp variables are carrying same information. So that we are going to drop any one of the variables for our further analysis.

2.1.4b:DISTRIBUTION OF CONTINUOUS VARIABLES WITH RESPECT TO TARGET VARIABLE:

Here we used scatterplot our visualization. First plot is between temperature and atemp variables. Both the temperature and atemp variables are mostly similar to each other.

From the below plot, we say that the temperature inceases our bike rental count also increases





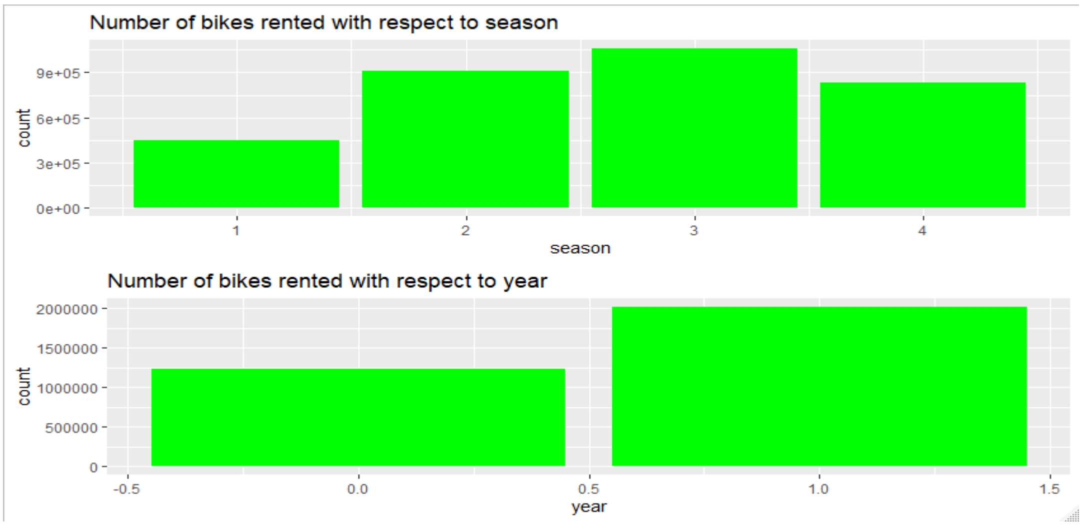
Second plot is between windspeed and humidity with respect to the count.

From the above plot we say that bike Rental count is not affected by humidity and windspeed.

2.1.4c:IMPACT OF THE CATEGORICAL VARIABLE WITH RESPECT TO TARGET VARIABLE:

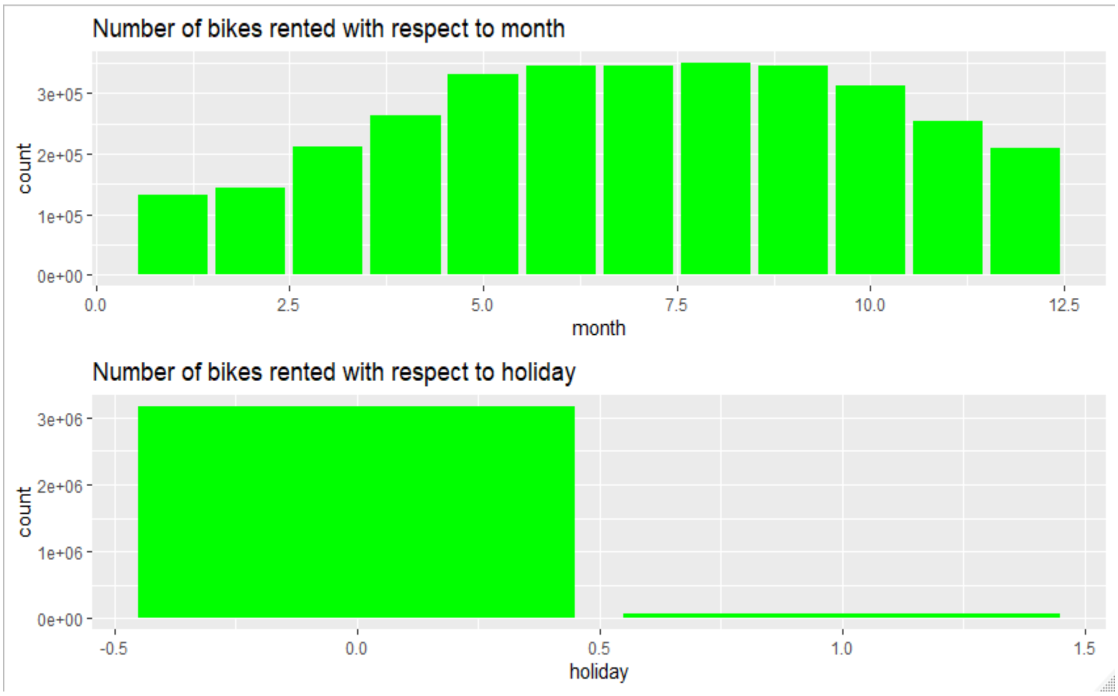
COUNT VS SEASON: Bike rental is higher in the season 3 which is which is fall and low in season 1 which is spring

COUNT VS YEAR : Bike rental is higher in the year 1 which is 2012



COUNT VS MONTH: Bike rental is higher in the month of 8 which is in august and low in 1 which is in January

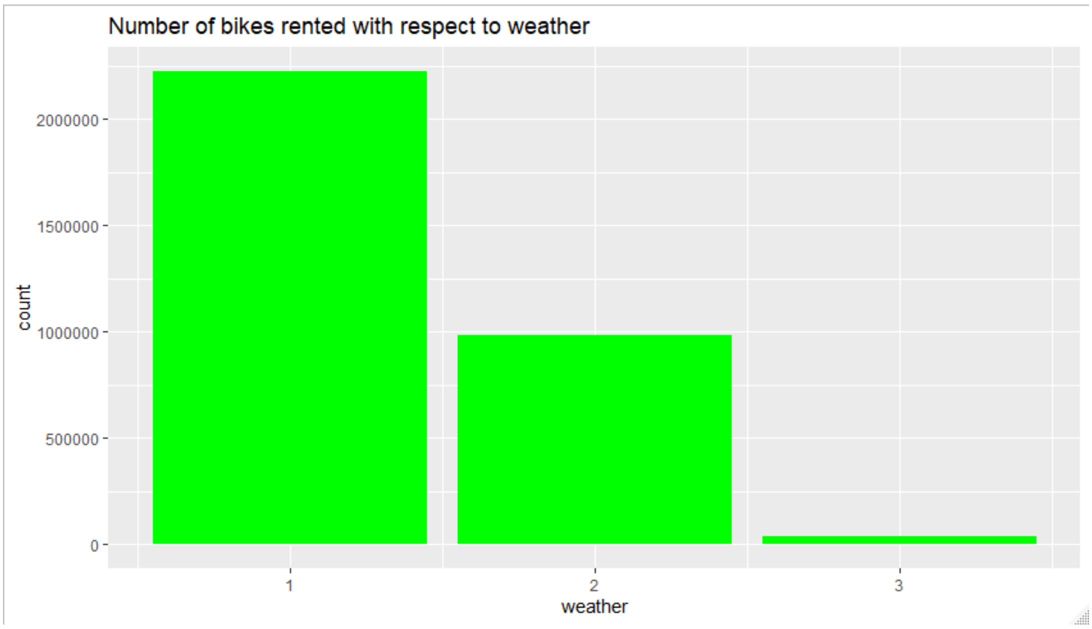
COUNT VS HOLIDAY: Bike rental count is higher in 0 which is holiday and low in workingday



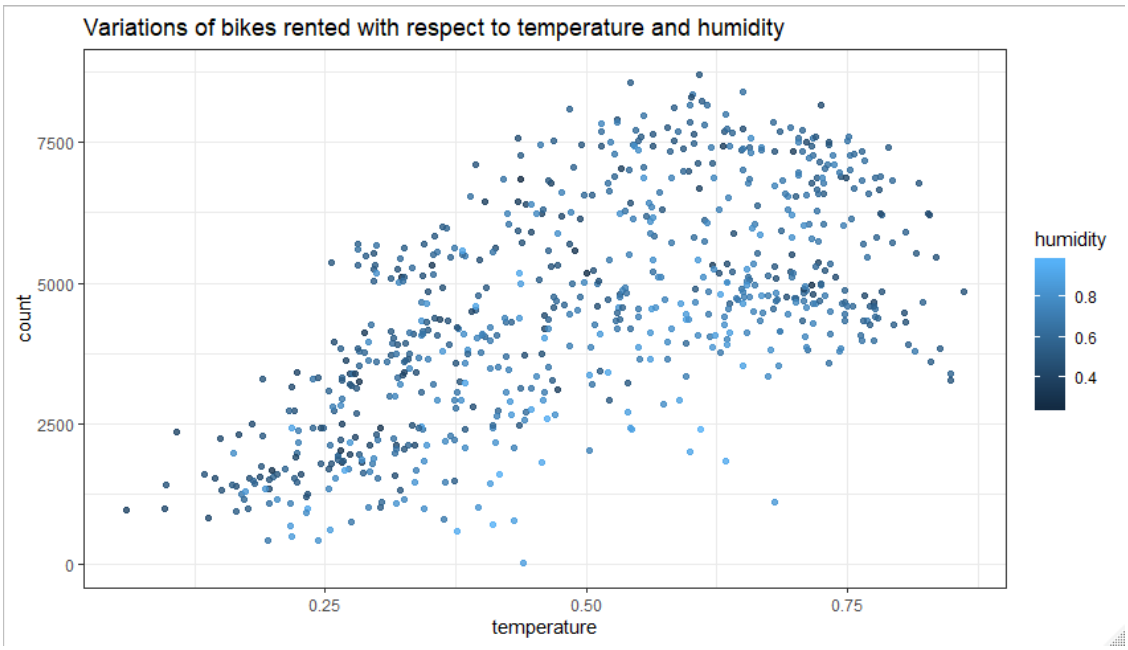
COUNT VS WEEKDAY: Bike rental count is high in 5 which is friday and low in 0 which is sunday

COUNT VS WORKING DAY: Bike rental count is high in 1 which is working day and low in 0 which is holiday

COUNT VS WEATHER: Bike rental count is high in 1 which is in 1 which clear,few clouds,partly cloudy and there is no bikes rental in 4



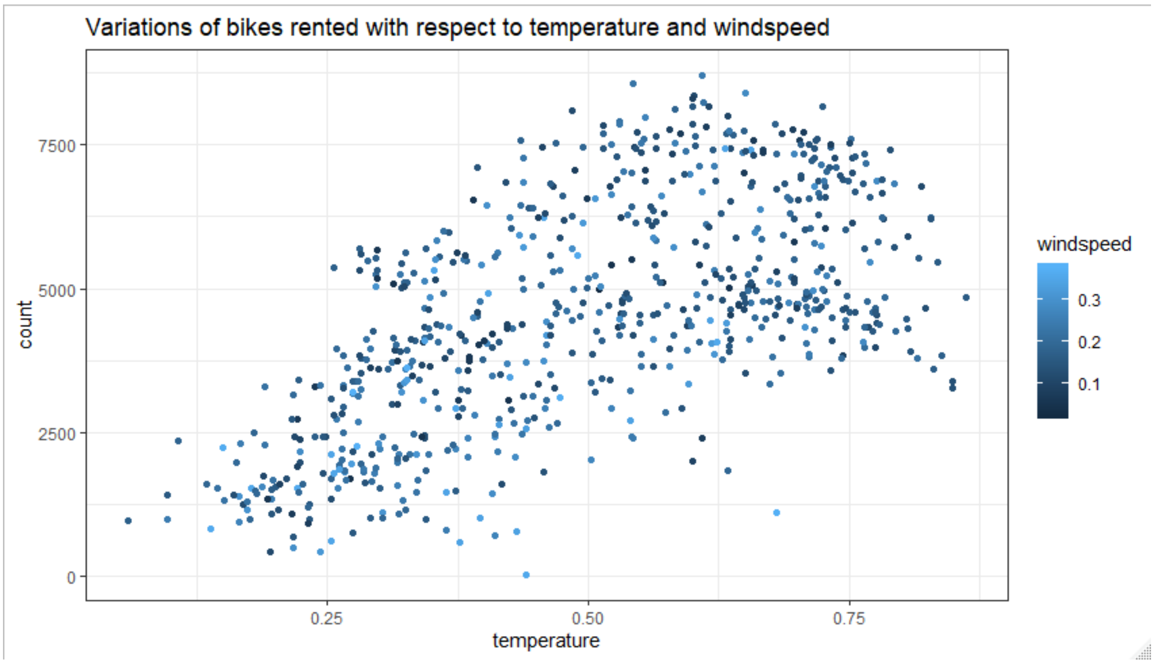
A) BIKE RENTED WITH RESPECTED TO TEMPERATURE AND HUMIDITY:



From the above plot, we can say that bike rental count is higher when the

- temperature is between 0.4 to 0.8
- humidity less than 0.8

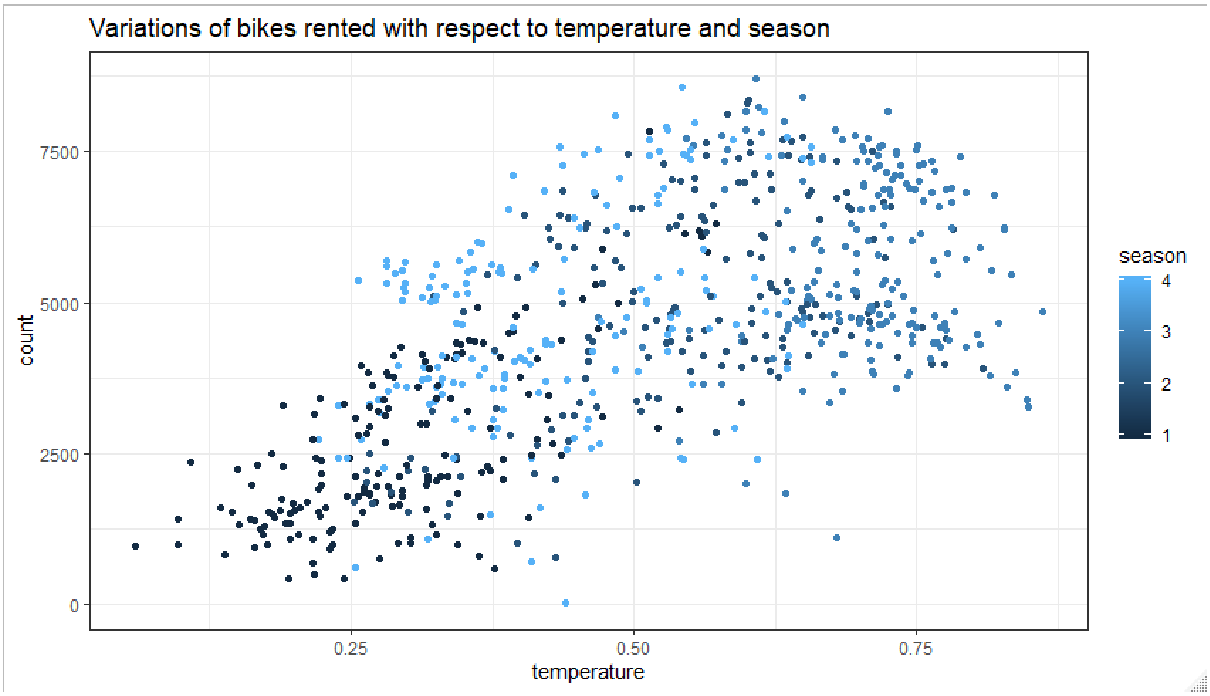
B) BIKE RENTED WITH RESPECT TO TEMPERATURE AND WINDSPEED:



From this above plot,we say that bike rental is higher when the

- temperature is between 0.4 to 0.8
- humidity is less than 0.8
- windspeed is less than 0.2

C) BIKE RENTED WITH RESPECT TO TEMPERATURE AND SEASON



From the plot we say that, bike rental count is higher when the

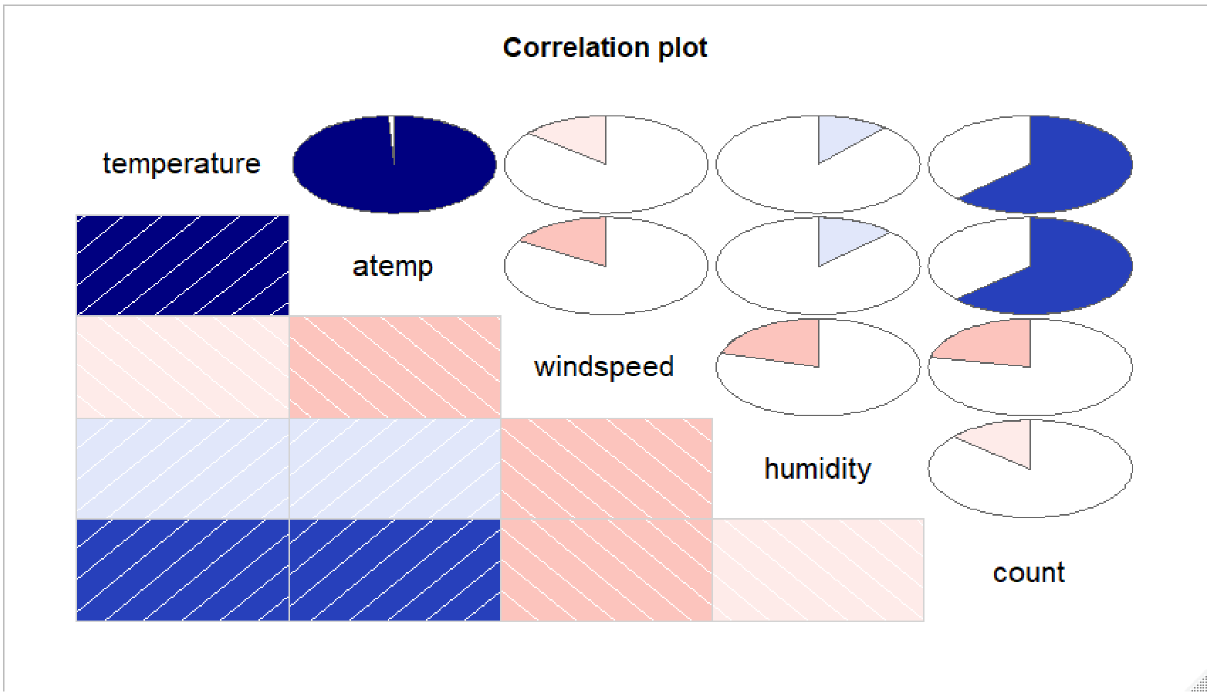
- temperature is between 0.4 to 0.8
- season was 2 and 3
- weather was from 1 and 2

2.1.5 FEATURE SELECTION:

We can use correlation analysis for numerical variables and Analysis of Variance for categorical variables. It shows correlation between the two variables. So that if two variables carrying same information can be removed.

2.1.5a: CORRELATION MATRIX AND PLOT

```
            temperature      atemp   windspeed
temperature   1.0000000  0.9917378 -0.1401690
atemp         0.9917378  1.0000000 -0.1660383
windspeed    -0.1401690 -0.1660383  1.0000000
humidity      0.1141910  0.1265874 -0.2044964
count         0.6258917  0.6292045 -0.2161933
              humidity      count
temperature  0.1141910  0.6258917
atemp        0.1265874  0.6292045
windspeed   -0.2044964 -0.2161933
humidity     1.0000000 -0.1366214
count       -0.1366214  1.0000000
> #correlation plot
```



From the above plot, we say that temperature and atemp variables are carrying same information. So we need to remove atemp variable.

2.1.5b: ANALYSIS OF VARIANCE:

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

From the above diagram, holiday,weekday,and working day these variables has p-value which is higher than 0.05.so that we need to drop these variables.

2.1.5c DIMENSION REDUCTION:

After the feature selection, we have only these 8 variables.they are mentioned in the below diagram.

```python
# Removing the variables which has p-value > 0.05 and correlated variable
Bike_Data = Bike_Data.drop(['atemp', 'holiday','weekday','workingday'], axis=1)
```

```python
# After removing variables lets check dimension of the data
Bike_Data.shape
```

```
(731, 8)
```

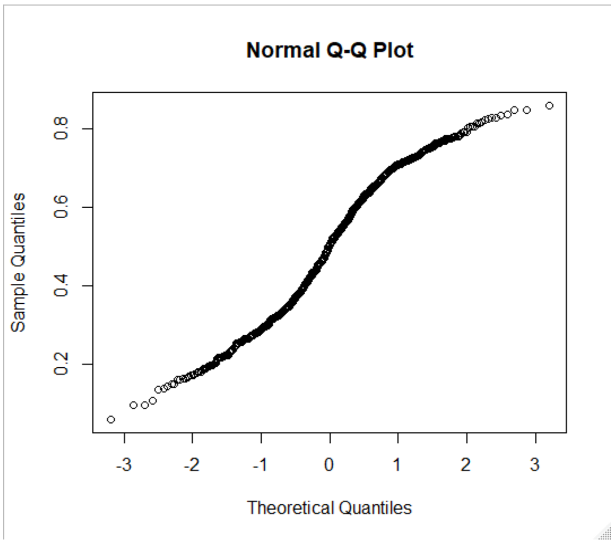
```python
# After removing variables lets check column names of the data
Bike_Data.columns
```

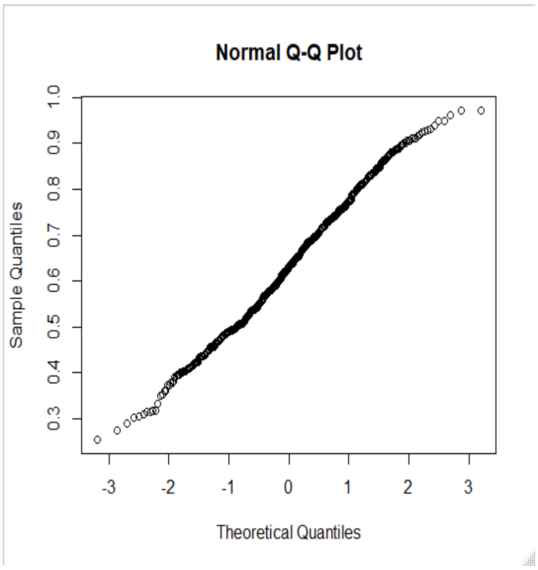
```
Index(['season', 'year', 'month', 'weather', 'temperature', 'humidity',
       'windspeed', 'count'],
      dtype='object')
```

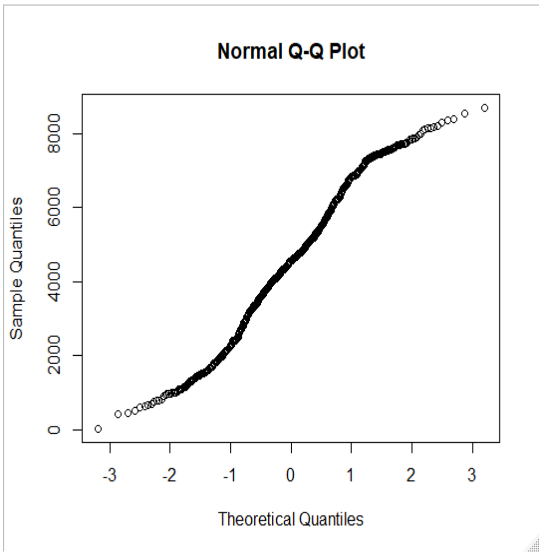
```python
#after removing the variables, we need update numerical and categorical variables

# numerical variable
cnames = ['temperature','humidity', 'windspeed', 'count']

# Categorical variables
catnames = ['season', 'year', 'month','weather']
```

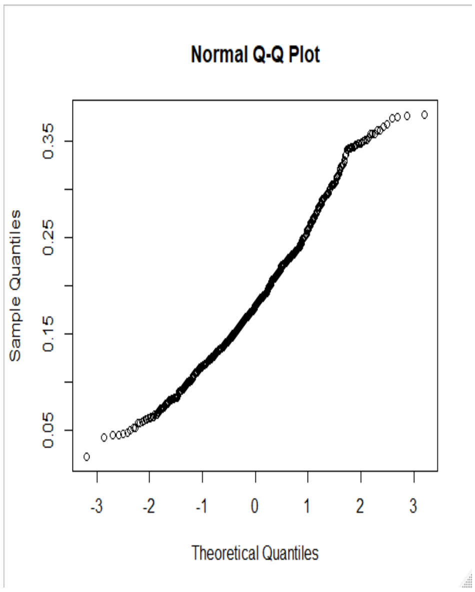
2.1.6 FEATURE SCALING:

In our dataset, all our continuous variables are already normalized. So we don't need to need any scaling methods to scale the data. Though we can use qqplot, summary, distribution of the data to see the normality









Summary of the data after feature selection and dimension reduction.

```
Bike_Data.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 MODEL DEVELOPMENT:

Next we need to split the data into train and test data and build a model using train data to predict the output using test data. Different models to be built and the model which gives more accurate values must be selected.

2.2.1 LINEAR REGRESSION:

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things:

(1) Does a set of predictor variables do a good job in predicting an outcome (dependent) variable?
(2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable?

These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. We trained our model in both R and Python and predicted in these languages using test data.

2.2.2 DECISION TREE:

Decision tree builds regression or classification models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

2.2.3. RANDOM FOREST:

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees, which involves training each decision tree on a different data sample where sampling is done with replacement. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.The higher no of trees in the random forest will give higher no of accuracy, so in random forest we can go for multiple trees. It can handle large no of independent variables without variable deletion and it will give the estimates that what variables are important.

2.3 HYPERPARAMETER TUNING

In statistics, hyperparameter is a parameter from a prior distribution; it captures the prior belief before data is observed. In any machine learning algorithm, these parameters need to be initialized before training a model. Choosing appropriate hyperparameters plays a crucial role in the success of good model. Since it makes a huge impact on the learned model.

2.3.1 TUNING PARAMETERS:

We will explore two different methods for optimizing hyperparameters:

- ✓ Grid Search
- ✓ Random Search

2.3.1A). RANDOM SEARCH:

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. In this search pattern, random combinations of parameters are considered in every iteration. The chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimised parameters without any aliasing.

2.3.1.B) Grid Search

Grid search is a technique which tends to find the right set of hyperparameters for the particular model. Hyperparameters are not the model parameters and it is not possible to find the best set from the training data. In this tuning technique, we simply build a model for every combination of various hyperparameters and evaluate each model. The model which gives the highest accuracy will be selected.

3. MODEL EVALUATION

3.1 EVALUATION METRICS:

In regression problems, we have three important metrics.they are

- ✓ MAPE(Mean Absolute Percentage Error)
- ✓ R-SQUARED
- ✓ RMSE(Root Mean Square Error)

3.1.1 MAPE(Mean Absolute Percentage Error)

MAPE is a measure of prediction accuracy of a forecasting method. It measures accuracy in terms of percentage.Lower value of MAPE indicates better fit.

3.1.2 R-SQUARED

R-squared is basically explains the degree to which input variable explain the variation of the output. In simple words Rsquared tells how much variance of dependent variable explained by the independent variable. It is a measure if goodness of fit in regression line. Higher values of R-square indicate better fit.

3.1.3 RMSE(Root Mean Square Error)

Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors).Residuals are a measure of how far from the regression line data points are, RMSE is a measure of how spread out these residuals are. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit.

PREDICTED MODELS IN PYTHON:

```
Final_Results
```

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Linear Regression	45.480011	17.528674	0.835720	0.847770	776.486700	782.728203
1	Decision Tree	81.433513	29.346978	0.669888	0.725704	1100.706423	1050.678261
2	Random Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
3	Grid Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
4	Random Forest	16.726797	17.056622	0.981439	0.871775	261.001514	718.368952
5	Random Search CV Random Forest	17.740802	17.042779	0.981037	0.874703	263.810977	710.117599
6	Grid search CV Random Forest	21.536863	17.406332	0.964378	0.872677	361.574910	715.836573

PREDICTED OUTPUT IN R:

	Model_name	MAPE_train	MAPE_test	Rsquare_train	Rsquare_test	RMSE_train	RMSE_test
1	Linear regression	0.15497164	0.1829289	0.8311816	0.8671739	789.6785	717.2833
2	Decision tree	0.52210598	0.2438791	0.8119266	0.7986807	833.4855	885.5906
3	Random search CV decision tree	0.52210598	0.2438791	0.8119266	0.7986807	833.4855	885.5906
4	Grid search CV decision tree	0.52210598	0.2438791	0.8119266	0.7986807	833.4855	885.5906
5	Random forest	0.07220796	0.1200109	0.9652279	0.9132608	371.1086	585.0001
6	Random search CV random forest	0.06441047	0.1222848	0.9718028	0.9077133	332.6579	593.6970
7	Grid search CV random forest	0.06441047	0.1222848	0.9718028	0.9077133	332.6579	601.4685

3.2 MODEL SELECTION:

From the predicted output in R and Python, the random forest model can have explained almost 90% of the predictor matches with the target variable.the values of the random forest model is mentioned below.

- **MAPE = 0.83**
- **R-SQUARED =0.87**
- **RMSE = 718.36**

4. R-CODE

```
##################################################
######## BIKE RENTAL COUNT PREDICTION ############
##################################################

#lets clean the R environment
rm(list = ls())

#setting working directory
setwd("D:/Data Science/Assignments/Project")
getwd()

# Load libraries
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced",
      "Information", "MASS", "rpart", "ROSE",
     'sampling', 'DataCombine', 'inTrees',"scales","psych","gplots")
#install.packages(x)
lapply(x, require, character.only = TRUE)
rm(x)

#lets load the data
Bike_Data = read.csv("day.csv")

#explore the data
dim(Bike_Data)
names(Bike_Data)

#rename the shortcut values for our understanding
names(Bike_Data)[1] = 'index'
names(Bike_Data)[2] = 'date'
names(Bike_Data)[4] = 'year'
names(Bike_Data)[5] = 'month'
names(Bike_Data)[9] = 'weather'
names(Bike_Data)[10] = 'temperature'
names(Bike_Data)[12] = 'humidity'
names(Bike_Data)[16] = 'count'

#lets check column names after renamed
names(Bike_Data)

#lets see top 5 observations in the dataset
head(Bike_Data)
```

```
#lets check last 5 observations in our dataset
tail(Bike_Data)

#lets check structure of each variable
str(Bike_Data)

#lets see summary of the dataset
summary(Bike_Data)

#in our dataset we have 16 variables out of which all are independent variable except last
variable
str(Bike_Data['count'])

#in our dataset some vaiables has no usefull information for our prediction
#so it is better to remove those variables.so it helps us to make useful inferences

#lets drop unnecessary variables
Bike_Data = subset(Bike_Data,select = -c(index,date,casual,registered))

#lets divide categorical variables and numerical variables
#numerical variables
cnames = c("temperature",'atemp','windspeed','humidity','count')
#categorical variables
catnames = c('season','year','month','holiday','weekday','workingday','weather')

#Data preprocessing
missing_val = sum(is.na(Bike_Data))
missing_val
#there is no missing values in our dataset

#outlier analysis
for (i in 1:length(cnames))
  {
    assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "count"), data =
subset(Bike_Data))+
          stat_boxplot(geom = "errorbar", width = 0.5) +
          geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                   outlier.size=1, notch=FALSE) +
          theme(legend.position="bottom")+
          labs(y=cnames[i],x="count")+
          ggtitle(paste("Box plot of count for",cnames[i])))
}

#plotting boxplot
library(gridExtra)
gridExtra::grid.arrange(gn1,gn2,ncol=2)
```

```
gridExtra::grid.arrange(gn3,gn4,ncol=2)
gridExtra::grid.arrange(gn5,ncol=1)

#lets remove outliers using boxplot
df = Bike_Data
for(i in cnames){
   print(i)
    outliers = Bike_Data[,i][Bike_Data[,i] %in% boxplot.stats(Bike_Data[,i])$out]
    print(length(outliers))
    Bike_Data = Bike_Data[which(!Bike_Data[,i] %in% outliers),]
  }
#lets plot boxplot after removing outliers
for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "count"), data =
subset(Bike_Data))+
          stat_boxplot(geom = "errorbar", width = 0.5) +
          geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,
                   outlier.size=1, notch=FALSE) +
          theme(legend.position="bottom")+
          labs(y=cnames[i],x="count")+
          ggtitle(paste("Box plot of count for",cnames[i])))
}
#plotting Boxplot after removing outliers
gridExtra::grid.arrange(gn1,gn2,ncol=2)
gridExtra::grid.arrange(gn3,gn4,ncol=2)
gridExtra::grid.arrange(gn5,ncol=1)

#data visualization
#univariate analysis
#lets see distribution of the variables.
for(i in 1:length(cnames))
{
   assign(paste0("h",i),ggplot(aes_string(x=(cnames[i])),
                          data=subset(Bike_Data))+
           geom_histogram(fill="green",colour = "red")+geom_density()+
           scale_y_continuous(breaks =pretty_breaks(n=10))+
           scale_x_continuous(breaks = pretty_breaks(n=10))+
           theme_bw()+xlab(cnames[i])+ylab("Frequency")+
           ggtitle(paste("distribution plot for ",cnames[i])))
}

#lets see distribution plot
gridExtra::grid.arrange(h1,h2,ncol=1)
gridExtra::grid.arrange(h3,h4,ncol=1)
gridExtra::grid.arrange(h5,ncol=1)
```

```
#bivariate analysis
#lets check distribution between target and continuous variables
for(i in 1:length(cnames))
{
  assign(paste0("s",i),ggplot(aes_string(y='count',x = (cnames[i])),
                         data=subset(Bike_Data))+
           geom_point(alpha=0.5,color="green") +
           ggtitle(paste("Scatter Plot between count vs ",cnames[i])))
}
#lets plot between continuous and target variables.
gridExtra::grid.arrange(s1,s2,ncol=1)
gridExtra::grid.arrange(s3,s4,ncol=1)
gridExtra::grid.arrange(s5,ncol=1)

#from the above graphs,we can see that as temperature increases and rental count also increases
#apart from temperature,windspeed and humidity doesnot impact on rental count

#lets check categorical variables
for(i in 1:length(catnames))
{
  assign(paste0("b",i),ggplot(aes_string(y='count',x = (catnames[i])),
                         data=subset(Bike_Data))+
           geom_bar(stat = "identity",fill = "green") +
           ggtitle(paste("Number of bikes rented with respect to",catnames[i])))+
    theme(axis.text.x = element_text( color="red", size=8))+
    theme(plot.title = element_text(face = "old"))
}

#lets plot between categorical and target variables
gridExtra::grid.arrange(b1,b2,ncol=1)
gridExtra::grid.arrange(b3,b4,ncol=1)
gridExtra::grid.arrange(b5,b6,ncol=1)
gridExtra::grid.arrange(b7,ncol=1)

# Based on the plot,we can the observe the below inferences
aggregate(count ~ season ,sum,data = Bike_Data)
#Bike rental count is high in season 3 which is fall  and low in season 1

aggregate(count ~ year ,sum,data = Bike_Data)
#Bike rental count is high in year 1 which is 2012

aggregate(count ~ month,sum,data = Bike_Data)
#Bike rental count is high in the month of august and low in january

aggregate(count ~ holiday ,sum,data = Bike_Data)
#Bike rental count is high on holidays which is 0 and low in working day
```

```
aggregate(count ~ weekday ,sum,data = Bike_Data)
#bike rental count is high in 5 which is friday and low in 0 which is sunday

aggregate(count ~ workingday,sum,data = Bike_Data)
#Bike rental count is high in 1 which is working day

#Bike rental count is high in weather 1 which is Clear, Few clouds, Partly cloudy, Partly
cloudy
#and there is no bike rented on 4
aggregate(count ~ weather,sum,data = Bike_Data)

# Bikes rented with respect to temperature and humidity
ggplot(Bike_Data,aes(temperature,count)) +
  geom_point(aes(color=humidity),alpha=0.8) +
  labs(title = "Variations of bikes rented with respect to temperature and humidity",
       x = "temperature")+ theme_bw()
#based on the below plot we know that bike rental is higher when the
#temperature is between 0.5 to 0.75
#humidity less than 0.6

#Bikes rented with respect to temperature and windspeed
ggplot(Bike_Data, aes(x = temperature, y = count))+
  geom_point(aes(color=windspeed))+
  labs(title = "Variations of bikes rented with respect to temperature and windspeed",
       x = "temperature")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
#based on the below plot we know that bike rental is higher when the
#temperature is between 0.5 to 0.75
#windspeed is less than 0.2

# Bikes rented with respect to temperature and season
ggplot(Bike_Data, aes(x = temperature, y = count))+
  geom_point(aes(color=season))+
  labs(title = "Variations of bikes rented with respect to temperature and season",
       x = "temperature")+
  theme(plot.title = element_text(hjust = 0.5, face = "bold"))+
  theme_bw()
#based on the below plot we know that bike rental is higher when the
#temperature is between 0.5 to 0.75
#season was 2 and 3
```

#**FEATURE SELECTION**

```
#lets find correlation matrix using corrplot and correlation plot using corrgram library
#FOR NUMERICAL VARIABLES
#lets save dataset after outlier analysis
df = Bike_Data
#correlation matrix
correlation_matrix = cor(Bike_Data[,cnames])
correlation_matrix
#correlation plot
corrgram(Bike_Data[,cnames],order = F,upper.panel = panel.pie,
        text.panel = panel.txt,main = 'Correlation plot')
#From the correlation plot,we see that temperature and atemp variables are correlated to each other
#so we need to remove atemp variable.

#lets see annova test for categorical variables
for (i in catnames) {
  print(i)
  anova = summary(aov(formula = count~Bike_Data[,i],Bike_Data))
  print(anova)
}
#based on the anova result, we are going to drop three variables namely,
#HOLIDAY
#WEEKDAY
#WORKINGDAY
#because these variables having the p-value > 0.05

#Dimension reduction
Bike_Data = subset(Bike_Data,select = -c(holiday,weekday,workingday,atemp))

#lets check after dimension reduction
dim(Bike_Data)
head(Bike_Data)

#lets update continous and categorical variables after dimension reduction
cnames = c('temperature','humidity','windspeed','count')
catnames = c('season','year','month','weather')
```

#**FEATURE SCALING**

```
#lets check normality between the varaibles
for (i in cnames){
  print(i)
  normality = qqnorm(Bike_Data[,i])

}
```

```
#already we plotted distrution between these variables,lets recall it
for(i in 1:length(cnames))
{
  assign(paste0("h",i),ggplot(aes_string(x=(cnames[i])),
                    data=subset(Bike_Data))+
         geom_histogram(fill="green",colour = "red")+geom_density()+
         scale_y_continuous(breaks =pretty_breaks(n=10))+
         scale_x_continuous(breaks = pretty_breaks(n=10))+
         theme_bw()+xlab(cnames[i])+ylab("Frequency")+
         ggtitle(paste("distribution plot for ",cnames[i])))
}
gridExtra::grid.arrange(h1,h2,h3,h4,ncol = 2)

#summary of the data
for (i in cnames) {
  print(i)
  print(summary(Bike_Data[,i]))

}
#Based on the above inferences and plots,we can see that the variables are normalised
#as mentioned in problem statement
```

#**MODEL DEVELOPMENT**

```
#lets clean our environment except preprocessed dataset
rmExcept('Bike_Data')

#we can pass categorical variables to regression problems
#lets convert categorical variables into dummy variables
#save our preprocessed data
df = Bike_Data

#create dummies
library(dummies)
catnames = c('season','year','month','weather')
Bike_Data = dummy.data.frame(Bike_Data,catnames)

#we succesfully created dummies,lets check dimension and top 5 observations
dim(Bike_Data)
head(Bike_Data)

#divide the data into train and test
set.seed(1234)
train_index = sample(1:nrow(df), 0.8 * nrow(df))
train_data = Bike_Data[train_index,]
test_data = Bike_Data[-train_index,]
```

#**linear regression**
#check multicollearity
library(usdm)
cnames = c('temperature','humidity','windspeed')
vif(Bike_Data[,cnames])

vifcor(Bike_Data[,cnames], th = 0.9)
#No variable from the 3 input variables has collinearity problem.

#The linear correlation coefficients ranges between:
 # min correlation (humidity ~ temperature): 0.1267216
#max correlation (windspeed ~ humidity): -0.2411599

#---------- VIFs of the remained variables --------
 # Variables VIF
#1 temperature 1.034137
#2 humidity 1.070959
#3 windspeed 1.080362
#lets run regression model
lm_model = lm(count~. ,data = Bike_Data)
#lets check performance of our modedl
summary(lm_model)
#Residual standard error: 787.1 on 710 degrees of freedom
#Multiple R-squared: 0.8394, Adjusted R-squared: 0.8349
#F-statistic: 185.6 on 20 and 710 DF, p-value: < 2.2e-16

Function for Error metrics to calculate the performance of model
#lets build function for MAPE
#calculate MAPE
MAPE = function(y, y1){
 mean(abs((y - y1)/y))
}

# 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))
}

#lets predict for train and test data
Predictions_LR_train = predict(lm_model,train_data[,-25])
Predictions_LR_test = predict(lm_model,test_data[,-25])

```
#let us check performance of our model

#mape calculation
LR_train_mape = MAPE(Predictions_LR_train,train_data[,25])
LR_test_mape = MAPE(test_data[,25],Predictions_LR_test)

#Rsquare calculation
LR_train_r2 = rsquare(train_data[,25],Predictions_LR_train)
LR_test_r2 = rsquare(test_data[,25],Predictions_LR_test)

#rmse calculation
LR_train_rmse = RMSE(train_data[,25],Predictions_LR_train)
LR_test_rmse = RMSE(test_data[,25],Predictions_LR_test)

print(LR_train_mape)#0.16
print(LR_test_mape)#0.17
print(LR_train_r2)#0.825
print(LR_test_r2)#0.893
print(LR_train_rmse)#804.9
print(LR_test_rmse)#648.9
```

#**Decision tree regression**

```
library(rpart)
DT_model = rpart(count ~ ., data = train_data, method = "anova")
DT_model


# Lets predict for train and test data
predictions_DT_train= predict(DT_model,train_data[,-25])
predictions_DT_test= predict(DT_model,test_data[,-25])

# MAPE calculation
DT_train_mape = MAPE(train_data[,25],predictions_DT_train)
DT_test_mape = MAPE(test_data[,25],predictions_DT_test)

# Rsquare calculation
DT_train_r2= rsquare(train_data[,25],predictions_DT_train)
DT_test_r2 = rsquare(test_data[,25],predictions_DT_test)

# RMSE calculation
DT_train_rmse = RMSE(train_data[,25],predictions_DT_train)
DT_test_rmse = RMSE(test_data[,25],predictions_DT_test)

print(DT_train_mape)#0.536
print(DT_test_mape)#0.269
```

```
print(DT_train_r2)#0.806
print(DT_test_r2)#0.834
print(DT_train_rmse)#846.85
print(DT_test_rmse)#805.67
```

#**Random search CV in decision tree**

```
df = Bike_Data
#setting parameters for training using caret library
control = trainControl(method="repeatedcv", number=5, repeats=1,search='random')
maxdepth = c(1:30)
tunegrid = expand.grid(.maxdepth=maxdepth)

# Lets build a model using above parameters on train data
RDT_model = caret::train(count~., data=train_data,
method="rpart2",trControl=control,tuneGrid= tunegrid)
print(RDT_model)

#lets look best parameter
best_parameter = RDT_model$bestTune
print(best_parameter)
#maximum depth = 10

#again build a decsion tree using best parameters
RDT_bestmodel = rpart(count~.,train_data,method = 'anova',maxdepth=10)
print(RDT_bestmodel)

#lets predict for train and test data
predictions_RDT_train = predict(RDT_bestmodel,train_data[1:24])
predictions_RDT_test = predict(RDT_bestmodel,test_data[1:24])

#model performance
# MAPE calculation
RDT_train_mape = MAPE(train_data[,25],predictions_RDT_train)
RDT_test_mape = MAPE(test_data[,25],predictions_RDT_test)

# Rsquare calculation
RDT_train_r2= rsquare(train_data[,25],predictions_RDT_train)
RDT_test_r2 = rsquare(test_data[,25],predictions_RDT_test)

# RMSE calculation
RDT_train_rmse = RMSE(train_data[,25],predictions_RDT_train)
RDT_test_rmse = RMSE(test_data[,25],predictions_RDT_test)

print(RDT_train_mape)#0.522
print(RDT_test_mape)#0.243
print(RDT_train_r2)#0.811
```

```
print(RDT_test_r2)#0.798
print(RDT_train_rmse)#833.48
print(RDT_test_rmse)#885.59
```

#**Grid search CV decision tree**

```
#setting parameters for training using caret library
control = trainControl(method="repeatedcv", number=5, repeats=2,search='grid')
maxdepth = c(6:30)
tunegrid = expand.grid(maxdepth=maxdepth)

# Lets build a model using above parameters on train data
GDT_model = caret::train(count~., data=train_data,
method="rpart2",trControl=control,tuneGrid= tunegrid)
print(GDT_model)

#lets look best parameter
best_parameter = GDT_model$bestTune
print(best_parameter)
#maximum depth = 10

#again build a decsion tree using best parameters
GDT_bestmodel = rpart(count~.,train_data,method = 'anova',maxdepth=10)
print(GDT_bestmodel)

#lets predict for train and test data
predictions_GDT_train = predict(GDT_bestmodel,train_data[1:24])
predictions_GDT_test = predict(GDT_bestmodel,test_data[1:24])

#model performance
# MAPE calculation
GDT_train_mape = MAPE(train_data[,25],predictions_GDT_train)
GDT_test_mape = MAPE(test_data[,25],predictions_GDT_test)

# Rsquare calculation
GDT_train_r2= rsquare(train_data[,25],predictions_GDT_train)
GDT_test_r2 = rsquare(test_data[,25],predictions_GDT_test)

# RMSE calculation
GDT_train_rmse = RMSE(train_data[,25],predictions_GDT_train)
GDT_test_rmse = RMSE(test_data[,25],predictions_GDT_test)

print(GDT_train_mape)#0.522
print(GDT_test_mape)#0.243
print(GDT_train_r2)#0.811
print(GDT_test_r2)#0.798
print(GDT_train_rmse)#833.48
print(GDT_test_rmse)#885.59
```

```
#RANDOM FOREST
#lets build the random forest model
RF_model = randomForest(count~.,data = train_data,n.trees = 500)
print(RF_model)

#lets predict for both train and test data
predictions_RF_train = predict(RF_model,train_data[-25])
predictions_RF_test = predict(RF_model,test_data[-25])

#MAPE calculation
RF_train_mape = MAPE(predictions_RF_train,train_data[,25])
RF_test_mape = MAPE(predictions_RF_test,test_data[,25])

#Rsquare calculation
RF_train_r2 = rsquare(predictions_RF_train,train_data[,25])
RF_test_r2 = rsquare(predictions_RF_test,test_data[,25])

#RMSE calculation
RF_train_rmse = RMSE(train_data[,25],predictions_RF_train)
RF_test_rmse = RMSE(test_data[,25],predictions_RF_test)

print(RF_train_mape)#0.07
print(RF_test_mape)#0.11
print(RF_train_r2)#0.965
print(RF_test_r2)#0.912
print(RF_train_rmse)#371.06
print(RF_test_rmse)#586.72

#Random search CV random forest
#setting parameters for training using caret library
control = trainControl(method="repeatedcv", number=5, repeats=3,search='random')
maxdepth = c(1:30)
tunegrid = expand.grid(maxdepth=maxdepth)

#lets build Random forest model using the above parameters
RRF_model = caret::train(count~.,data=train_data,method
='rf',trcontrol=control,tunegrid=tunegrid)
print(RRF_model)
best_parameter = RRF_model$bestTune
print(best_parameter)
#mtry = 13
#lets again build the random forest by above paremeters
RRF_bestmodel = randomForest(count~.,data = train_data,method = 'rf',mtry =
13,importance = TRUE)
print(RRF_bestmodel)
```

```
#lets predict for both train and test data
prediction_RRF_train = predict(RRF_bestmodel,train_data[-25])
prediction_RRF_test = predict(RRF_bestmodel,test_data[-25])

#MAPE calculation
RRF_train_mape = MAPE(train_data[,25],prediction_RRF_train)
RRF_test_mape = MAPE(test_data[,25],prediction_RRF_test)

#Rsquare calculation
RRF_train_r2 = rsquare(train_data[,25],prediction_RRF_train)
RRF_test_r2 = rsquare(test_data[,25],prediction_RRF_test)

#RMSE calculation
RRF_train_rmse = RMSE(train_data[,25],prediction_RRF_train)
RRF_test_rmse = RMSE(test_data[,25],prediction_RRF_test)

print(RRF_train_mape)#0.241
print(RRF_test_mape)#0.159
print(RRF_train_r2)#0.971
print(RRF_test_r2)#0.907
print(RRF_train_rmse)#333.513
print(RRF_test_rmse)#602.26
```

#**GRID SEARCH CV RANDOM FOREST**

```
#lets set require parameters using caret library
control = trainControl(method="repeatedcv", number=5, repeats=4,search='grid')
maxdepth = c(6:30)
tunegrid = expand.grid(maxdepth=maxdepth)

#lets build Random forest model using the above parameters
GRF_model = caret::train(count~.,data=train_data,method
='rf',trcontrol=control,tunegrid=tunegrid)
print(GRF_model)
best_parameter = GRF_model$bestTune
print(best_parameter)
#mtry = 13
#lets again build the same model using bestparameter
GRF_bestmodel = randomForest(count~.,data = train_data,mtry =13,importance =
TRUE,method='rf')
print(GRF_bestmodel)

#lets predict on train and test data,
predictions_GRF_train = predict(GRF_bestmodel,train_data[-25])
predictions_GRF_test = predict(GRF_bestmodel,test_data[-25])
#MAPE calculation
GRF_train_mape = MAPE(predictions_GRF_train,train_data[,25])
GRF_test_mape = MAPE(predictions_GRF_test,test_data[,25])
```

```
#Rsquare calculation
GRF_train_r2 = rsquare(predictions_GRF_train,train_data[,25])
GRF_test_r2 = rsquare(predictions_GRF_test,test_data[,25])

#RMSE calculation
GRF_train_rmse = RMSE(predictions_GRF_train,train_data[,25])
GRF_test_rmse = RMSE(predictions_GRF_test,test_data[,25])

print(GRF_train_mape)#0.06
print(GRF_test_mape)#0.12
print(GRF_train_r2)#0.972
print(GRF_test_r2)#0.90
print(GRF_train_rmse)#335.18
print(GRF_test_rmse)#597.59

#MODEL SELECTION
Model_name = c('Linear regression',
          'Decision tree','Random search CV decision tree','Grid search CV decision tree',
          'Random forest','Random search CV random forest','Grid search CV random forest')

MAPE_train = c(LR_train_mape,DT_train_mape,RDT_train_mape,GDT_train_mape,
          RF_train_mape,GRF_train_mape,GRF_train_mape)

MAPE_test = c(LR_test_mape,DT_test_mape,RDT_test_mape,GDT_test_mape,
         RF_test_mape,GRF_test_mape,GRF_test_mape)

Rsquare_train = c(LR_train_r2,DT_train_r2,RDT_train_r2,GDT_train_r2,
            RF_train_r2,GRF_train_r2,GRF_train_r2)

Rsquare_test = c(LR_test_r2,DT_test_r2,RDT_test_r2,GDT_test_r2,
           RF_test_r2,GRF_test_r2,GRF_test_r2)

RMSE_train =  c(LR_train_rmse,DT_train_rmse,RDT_train_rmse,GDT_train_rmse,
           RF_train_rmse,GRF_train_rmse,GRF_train_rmse)

RMSE_test = c(LR_test_rmse,DT_test_rmse,RDT_test_rmse,GDT_test_rmse,
         RF_test_rmse,RRF_test_rmse,GRF_test_rmse)

FINAL_RESULTS =
data.frame(Model_name,MAPE_train,MAPE_test,Rsquare_train,Rsquare_test,
                RMSE_train,RMSE_test)
```

	Model_name	MAPE_train	MAPE_test	Rsquare_train	Rsquare_test	RMSE_train	RMSE_test
1	Linear regression	0.15497164	0.1829289	0.8311816	0.8671739	789.6785	717.2833
2	Decision tree	0.52210598	0.2438791	0.8119266	0.7986807	833.4855	885.5906
3	Random search CV decision tree	0.52210598	0.2438791	0.8119266	0.7986807	833.4855	885.5906
4	Grid search CV decision tree	0.52210598	0.2438791	0.8119266	0.7986807	833.4855	885.5906
5	Random forest	0.07220796	0.1200109	0.9652279	0.9132608	371.1086	585.0001
6	Random search CV random forest	0.06441047	0.1222848	0.9718028	0.9077133	332.6579	593.6970
7	Grid search CV random forest	0.06441047	0.1222848	0.9718028	0.9077133	332.6579	601.4685

5. PYTHON CODE

Bike Rental Prediction

```
# Load the required libraries for analysis of data
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Set working directory
os.chdir("D:\Data Science\Assignments\Project")

# lets Check working directory
os.getcwd()

# Load the data
Bike_Data = pd.read_csv("day.csv")
```

Explore the data

```
# Check the dimensions(no of rows and no of columns)
Bike_Data.shape

# Check names of dataset
Bike_Data.columns

# Rename variables in dataset
Bike_Data = Bike_Data.rename(columns = {'instant':'index','dteday':'date','yr':'year','mnth':'month','weath
ersit':'weather',
                        'temp':'temperature','hum':'humidity','cnt':'count'})

Bike_Data.columns


#lets see first five observations of our data
Bike_Data.head()

# lets see last five observations of our data
Bike_Data.tail()

# Lets see the datatypes of the given data
```

```python
Bike_Data.dtypes

# lets Check summary of the dataset
Bike_Data.describe()

# Variable Identification
Bike_Data['count'].dtypes

#lets drop some variables because it doesnot carry any useful information

Bike_Data = Bike_Data.drop(['casual','registered','index','date'],axis=1)

# Lets check dimensions of data after removing some variables
Bike_Data.shape


# Continous Variables
cnames= ['temperature', 'atemp', 'humidity', 'windspeed', 'count']

# Categorical variables-
cat_cnames=['season', 'year', 'month', 'holiday', 'weekday', 'workingday','weather']
```

EDA or Data Preprocessing

```python
# Missing Value anlysis

# to check if there is any missing values
Missing_val = Bike_Data.isnull().sum()
Missing_val
# In our dataset we dont have any missing values.so that we dont need to do any imputation methods

# Outlier Analysis

# Lets save copy of dataset before preprocessing
df = Bike_Data.copy()
Bike_Data = df.copy()

# Using seaborn library, we can viualize the outliers by plotting box plot
for i in cnames:
   print(i)
   sns.boxplot(y=Bike_Data[i])
   plt.xlabel(i)
   plt.ylabel("values")
   plt.title("Boxplot of "+i)
   plt.show()

# From boxplot we can see inliers in humidity and outliers in windspeed
```

```
#Lets detect and remove outliers
for i in cnames:
  print(i)
  # Quartiles and IQR
  q25,q75 = np.percentile(Bike_Data[i],[25,75])
  IQR = q75-q25

  # Lower and upper limits
  Minimum = q25 - (1.5 * IQR)
  print(Minimum)
  Maximum = q75 + (1.5 * IQR)
  print(Maximum)

  Minimum = Bike_Data.loc[Bike_Data[i] < Minimum ,i]
  Maximum = Bike_Data.loc[Bike_Data[i] > Maximum ,i]

#we substituted minimum values for inliers and maximum values for outliers.
#from that we removed all the outliers.

# after replacing the outliers,let us plot boxplot for understanding
for i in cnames:
  print(i)
  sns.boxplot(y=Bike_Data[i])
  plt.xlabel(i)
  plt.ylabel("values")
  plt.title("Boxplot of "+i)
  plt.show()
```

Visualization

```
# Univariate Analysis

# temperature
sns.FacetGrid(Bike_Data , height = 5).map(sns.distplot,'temperature').add_legend()
#normally distributed

# humidity
sns.FacetGrid(Bike_Data , height = 5).map(sns.distplot,'humidity').add_legend()
#normally distributed

# windspeed
sns.FacetGrid(Bike_Data , height = 5).map(sns.distplot,'windspeed').add_legend()
#normally distributed

#atemp
sns.FacetGrid(Bike_Data , height = 5).map(sns.distplot,'atemp').add_legend()
#normally distributed
```

```
# count
sns.FacetGrid(Bike_Data , height = 5).map(sns.distplot,'count').add_legend()
#normally distributed

# Bivariate Analysis -----------------------------------------------------
# Lets check impact of continous variables on target variable

# count vs temperature
sns.violinplot(x='count',y='temperature',data=Bike_Data)

#temperature is directly proportional to each other
#as temperature increases bike rental count also increases

# count vs humidity

sns.violinplot(x='count',y='humidity',data=Bike_Data)

# Apart from humidity,Bike rental count does not get affected

# count vs windspeed

sns.violinplot(x='count',y='windspeed',data=Bike_Data)

# Apart from windspeed, Bike rental count does not get affected

#for categorical variables


# SEASON
print(Bike_Data.groupby(['season'])['count'].sum())
#based on the season, bike rental count is high in season 3 which is fall and low in season 1 which is spring

#lets visualize the count using scatterplot
sns.scatterplot(x='season',y='count',data = Bike_Data)

# YEAR
print(Bike_Data.groupby(['year'])['count'].sum())
#based on the year, bike rental count is high in the year 1 which is 2012

#lets visualize the count using scatterplot
sns.scatterplot(x='year',y='count',data = Bike_Data)

# MONTH
print(Bike_Data.groupby(['month'])['count'].sum())
#Based on the month, Bike rental count is high in 8 which is in august and low in 1 which is in january
```

```
#lets visualize the count using scatterplot
sns.scatterplot(x='month',y='count',data = Bike_Data)

#HOLIDAY
print(Bike_Data.groupby(['holiday'])['count'].sum())
#Based on the holiday, bike rental count is high in 0 which is holiday and low in 1 which is working day

#lets visualize the count using scatterplot
sns.scatterplot(x='holiday',y='count',data = Bike_Data)

# WEAKDAY
print(Bike_Data.groupby(['weekday'])['count'].sum())
#Based on the weakday, bike rental count is high in 5 which is friday and low in 0 which is sunday

#lets visualize the count using scatterplot
sns.scatterplot(x='weekday',y='count',data = Bike_Data)

# WORKINGDAY
print(Bike_Data.groupby(['workingday'])['count'].sum())
#Based on the workingday, Bike rental count is high in 1 which is working day and low in 0 which is hoiday

#lets visualize the count using scatterplot
sns.scatterplot(x='workingday',y='count',data = Bike_Data)

#WEATHER
print(Bike_Data.groupby(['weather'])['count'].sum())
#Based n the weather bike rental count is higher in 1 which clear,few clouds,partly cloudy and there is no bik
es rental in 4

#lets visualize the count using scatterplot
sns.scatterplot(x='weather',y='count',data = Bike_Data)

# Bike rented with respected to tempeature and humidity
f, ax = plt.subplots(figsize=(10, 10))
sns.scatterplot(x="temperature", y="count",
            hue="humidity", size="count",
            palette="rainbow",sizes=(1, 100), linewidth=0,
            data=Bike_Data,ax=ax)
plt.title("Varation in bike rented with respect to temperature and humidity")
plt.ylabel("Bike rental count")
plt.xlabel("temperature")

# based on the below plot we know that bike rental is higher when the
                #temperature is between 0.4 to 0.8
                #humidity less than 0.8

#Bikes rented with respect to temperature and windspeed
```

```
f, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(x="temperature", y="count",
            hue="windspeed", size="humidity",
            palette="rainbow",sizes=(1, 100), linewidth=0,
            data=Bike_Data,ax=ax)
plt.title("Varation in bike rented with respect to  temperature and windspeed")
plt.ylabel("Bike rental count")
plt.xlabel("temperature")

#based on the below plot we know that bike rental is higher when the
                    #temperature is between 0.4 to 0.8
                    #humidity is less than 0.8
                    #windspeed is less than 0.2

# Bikes rented with respect to temperature and season
f, ax = plt.subplots(figsize=(10,10))
sns.scatterplot(x="temperature", y="count",
            hue="season", size="count",style= "weather",
            palette="rainbow",sizes=(1, 100), linewidth=0,
            data=Bike_Data,ax=ax)
plt.title("Varation in bike rented with respect to temperature and season")
plt.ylabel("Bike rental count")
plt.xlabel("Normalized temperature")

#based on the below plot we know that bike rental is higher when the
                    #temperature is between 0.4 to 0.8
                    #season was 2 and 3
                    #weather was from 1 and 2
```

Feature Selection

```
# Lets save dataset after outlier analysis
df =  Bike_Data.copy()
Bike_Data = df.copy()

# Correlation analysis

# Correlation matrix continuous variables
Bike_corr= Bike_Data.loc[:,cnames]

# Generate correlation matrix
corr_matrix = Bike_corr.corr()
(print(corr_matrix))

# Set the width and hieght of the plot
f, ax = plt.subplots(figsize=(15,15))
```

```
#Plot using seaborn library
sns.heatmap(corr_matrix, mask=np.zeros_like(corr_matrix, dtype=np.bool), cmap=sns.diverging_palette(
220, 10, as_cmap=True),
         square=True, ax=ax,annot=True)

plt.title("Correlation Plot For Numeric or Continous Variables")

#from the below plot,we came to know that both temperature and atemp variables are carrying almost sam
e information
#hence there is no need to continue with both variables.
#so we need to drop any one of the variables
#here I am dropping atemp variables

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

# ANOVA test for categorical variables

for i in cat_cnames:
   mod = ols('count' + '~' + i, data = Bike_Data).fit()
   aov_table = sm.stats.anova_lm(mod, typ = 2)
   print(aov_table)

#based on the anova result, we are going to drop three variables namely,
                                    #HOLIDAY
                                    #WEEKDAY
                                    #WORKINGDAY
                  #because these variables having the p-value > 0.05

# Removing the variables which has p-value > 0.05 and correlated variable
Bike_Data = Bike_Data.drop(['atemp', 'holiday','weekday','workingday'], axis=1)

# After removing variables lets check dimension of the data
Bike_Data.shape

# After removing variables lets check column names of the data
Bike_Data.columns

#after removing the variables, we need update numerical and categorical variables

# numerical variable
cnames = ['temperature','humidity', 'windspeed', 'count']

# Categorical variables
catnames = ['season', 'year', 'month','weather']
```

Feature scaling

```
#based on the details of the attributes given, all the numerical variables are normalised

#lets visualise the numerical variables to see normality
for i in cnames:
    print(i)
    sm.qqplot(Bike_Data[i])
    plt.title("Normalized qq plot for " +i)
    plt.show()

for i in cnames:
    print(i)
    sns.distplot(Bike_Data[i],bins='auto',color='blue')
    plt.title("Distribution plot for "+i)
    plt.ylabel("Density")
    plt.show()

Bike_Data.describe()

#we confirmed the normalized data based on the qqplot,distribution plot and summary of the data
```

Model Development

```
# Load Required libraries for model development
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics

#In Regression problems, we can't pass directly categorical variables.
#so we need to convert all categorical variables into dummy variables

df = Bike_Data
Bike_Data = df

#  Converting categorical variables to dummy variables
Bike_Data = pd.get_dummies(Bike_Data,columns=catnames)


Bike_Data.shape

Bike_Data.columns

# Lets Divide the data into train and test set
```

```python
X= Bike_Data.drop(['count'],axis=1)
y= Bike_Data['count']

# Divide data into train and test sets
X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=.20)

# Function for Error metrics to calculate the performance of model
def MAPE(y_true,y_prediction):
    mape= np.mean(np.abs(y_true-y_prediction)/y_true)*100
    return mape

# Linear Regression model

# Import libraries
import statsmodels.api as sm

# Linear Regression model
LinearRegression_model= sm.OLS(y_train,X_train).fit()
print(LinearRegression_model.summary())

# Model prediction on  train data
LinearRegression_train= LinearRegression_model.predict(X_train)

# Model prediction on test data
LinearRegression_test= LinearRegression_model.predict(X_test)

# Model performance on train data
MAPE_train= MAPE(y_train,LinearRegression_train)

# Model performance on test data
MAPE_test= MAPE(y_test,LinearRegression_test)

# r2 value for train data
r2_train= r2_score(y_train,LinearRegression_train)

# r2 value for test data-
r2_test=r2_score(y_test,LinearRegression_test)

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

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

print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage 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))

Error_MetricsLT = {'Model Name': ['Linear Regression'],
                   'MAPE_Train':[MAPE_train],
                   'MAPE_Test':[MAPE_test],
                   'R-squared_Train':[r2_train],
                   'R-squared_Test':[r2_test],
                   'RMSE_train':[RMSE_train],
                   'RMSE_test':[RMSE_test]}

LinearRegression_Results = pd.DataFrame(Error_MetricsLT)
LinearRegression_Results
#Decision tree model
# Lets Build decision tree model on train and test data
from sklearn.tree import DecisionTreeRegressor

# Decision tree for regression
DecisionTree_model= DecisionTreeRegressor(max_depth=2).fit(X_train,y_train)

# Model prediction on train data
DecisionTree_train= DecisionTree_model.predict(X_train)

# Model prediction on test data
DecisionTree_test= DecisionTree_model.predict(X_test)

# Model performance on train data
MAPE_train= MAPE(y_train,DecisionTree_train)

# Model performance on test data
MAPE_test= MAPE(y_test,DecisionTree_test)

# r2 value for train data
r2_train= r2_score(y_train,DecisionTree_train)

# r2 value for test data
r2_test=r2_score(y_test,DecisionTree_test)

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

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

print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
```

```
print("Mean Absolute Precentage 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))
Error_MetricsDT = {'Model Name': ['Decision Tree'],
                   'MAPE_Train':[MAPE_train],
                   'MAPE_Test':[MAPE_test],
                   'R-squared_Train':[r2_train],
                   'R-squared_Test':[r2_test],
                   'RMSE_train':[RMSE_train],
                   'RMSE_test':[RMSE_test]}

DecisionTree_Results = pd.DataFrame(Error_MetricsDT)
DecisionTree_Results
# Random Search CV In Decision Tree
# Import libraries
from sklearn.model_selection import RandomizedSearchCV

RandomDecisionTree = DecisionTreeRegressor(random_state = 0)
depth = list(range(1,20,2))
random_search = {'max_depth': depth}

# Lets build a model using above parameters on train data
RandomDecisionTree_model= RandomizedSearchCV(RandomDecisionTree,param_distributions= rando
m_search,n_iter=5,cv=5)
RandomDecisionTree_model= RandomDecisionTree_model.fit(X_train,y_train)

# Lets look into best fit parameters
best_parameters = RandomDecisionTree_model.best_params_
print(best_parameters)
# Again rebuild decision tree model using randomsearch best fit parameter ie
# with maximum depth = 7
RDT_best_model = RandomDecisionTree_model.best_estimator_
print(RDT_best_model)
# Prediction on train data
RDT_train = RDT_best_model.predict(X_train)

# Prediction on test data
RDT_test = RDT_best_model.predict(X_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,r
square value
# MAPE for train data
MAPE_train= MAPE(y_train,RDT_train)

# MAPE for test data
```

```
MAPE_test= MAPE(y_test,RDT_test)

# Rsquare for train data
r2_train= r2_score(y_train,RDT_train)

# Rsquare for test data
r2_test=r2_score(y_test,RDT_test)

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

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


# Lets print the results
print("Best Parameter="+str(best_parameters))
print("Best Model="+str(RDT_best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage 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))

Error_MetricsRDT = {'Model Name': ['Random Search CV Decision Tree'],
                   'MAPE_Train':[MAPE_train],
                   'MAPE_Test':[MAPE_test],
                   'R-squared_Train':[r2_train],
                   'R-squared_Test':[r2_test],
                   'RMSE_train':[RMSE_train],
                   'RMSE_test':[RMSE_test]}

RandomDecisionTree_Results = pd.DataFrame(Error_MetricsRDT)
RandomDecisionTree_Results
# Grid Search CV in Decision Tree
# Import libraries
from sklearn.model_selection import GridSearchCV

GridDecisionTree= DecisionTreeRegressor(random_state=0)
depth= list(range(1,20,2))
grid_search= {'max_depth':depth}

# Lets build a model using above parameters on train data
GridDecisionTree_model= GridSearchCV(GridDecisionTree,param_grid=grid_search,cv=5)
GridDecisionTree_model= GridDecisionTree_model.fit(X_train,y_train)
# Lets look into best fit parameters from gridsearch cv DT
```

```
best_parameters = GridDecisionTree_model.best_params_
print(best_parameters)
# Again rebuild decision tree model using gridsearch best fit parameter ie
# with maximum depth = 7
GDT_best_model = GridDecisionTree_model.best_estimator_

# Prediction on train data  test data-
GDT_test = GDT_best_model.predict(X_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,r
square value
# MAPE for train data
MAPE_train= MAPE(y_train,GDT_train)

# MAPE for test data
MAPE_test= MAPE(y_test,GDT_test)

# Rsquare for train data
r2_train= r2_score(y_train,GDT_train)

# Rsquare for train data
r2_test=r2_score(y_test,GDT_test)

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

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


print("Best Parameter="+str(best_parameters))
print("Best Model="+str(GDT_best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage 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))

Error_MetricsGDT = {'Model Name': ['Grid Search CV Decision Tree'],
                    'MAPE_Train':[MAPE_train],
                    'MAPE_Test':[MAPE_test],
                    'R-squared_Train':[r2_train],
                    'R-squared_Test':[r2_test],
                    'RMSE_train':[RMSE_train],
                    'RMSE_test':[RMSE_test]}
```

```
GridDecisionTree_Results = pd.DataFrame(Error_MetricsGDT)
GridDecisionTree_Results
# Import libraris
from sklearn.ensemble import RandomForestRegressor

# Random Forest for regression
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)

# MAPE For train data
MAPE_train= MAPE(y_train,RF_train)

# MAPE For test data
MAPE_test= MAPE(y_test,RF_test)

# Rsquare  For train data
r2_train= r2_score(y_train,RF_train)

# Rsquare  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 Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage 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))
Error_MetricsRF = {'Model Name': ['Random Forest'],
                  'MAPE_Train':[MAPE_train],
                  'MAPE_Test':[MAPE_test],
                  'R-squared_Train':[r2_train],
                  'R-squared_Test':[r2_test],
                  'RMSE_train':[RMSE_train],
                  'RMSE_test':[RMSE_test]}

RandomForest_Results = pd.DataFrame(Error_MetricsRF)
```

```
RandomForest_Results
# Random Search CV in Random Forest
# Import libraries
from sklearn.model_selection import RandomizedSearchCV

RandomRandomForest = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,100,2))
depth = list(range(1,20,2))
random_search = {'n_estimators':n_estimator, 'max_depth': depth}

# Lets build a model using above parameters on train data
RandomRandomForest_model= RandomizedSearchCV(RandomRandomForest,param_distributions= ran
dom_search,n_iter=5,cv=5)
RandomRandomForest_model= RandomRandomForest_model.fit(X_train,y_train)
# Best parameters for model
best_parameters = RandomRandomForest_model.best_params_
print(best_parameters)
# Again rebuild random forest  model using gridsearch best fit parameter
RRF_best_model = RandomRandomForest_model.best_estimator_
# Prediction on train data
RRF_train = RRF_best_model.predict(X_train)

# Prediction on test data
RRF_test = RRF_best_model.predict(X_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,r
square value
# MAPE for train data
MAPE_train= MAPE(y_train,RRF_train)

# MAPE for test data
MAPE_test= MAPE(y_test,RRF_test)

# Rsquare for train data
r2_train= r2_score(y_train,RRF_train)

# Rsquare for test data
r2_test=r2_score(y_test,RRF_test)

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

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


print("Best Parameter="+str(best_parameters))
```

```
print("Best Model="+str(RRF_best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage 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))
Error_MetricsRSRF = {'Model Name': ['Random Search CV Random Forest'],
                    'MAPE_Train':[MAPE_train],
                    'MAPE_Test':[MAPE_test],
                    'R-squared_Train':[r2_train],
                    'R-squared_Test':[r2_test],
                    'RMSE_train':[RMSE_train],
                    'RMSE_test':[RMSE_test]}

RandomSearchRandomForest_Results = pd.DataFrame(Error_MetricsRSRF)
RandomSearchRandomForest_Results
# Grid search CV in Random Forest

# Import libraries
from sklearn.model_selection import GridSearchCV

GridRandomForest= RandomForestRegressor(random_state=0)
n_estimator = list(range(1,20,2))
depth= list(range(1,20,2))
grid_search= {'n_estimators':n_estimator, 'max_depth': depth}
# Lets build a model using above parameters on train data using random forest grid search cv
GridRandomForest_model= GridSearchCV(GridRandomForest,param_grid=grid_search,cv=5)
GridRandomForest_model= GridRandomForest_model.fit(X_train,y_train)
# Best fit parameters for model
best_parameters_GRF = GridRandomForest_model.best_params_
print(best_parameters_GRF)
# Again rebuild random forest model using gridsearch best fit parameter
GRF_best_model = GridRandomForest_model.best_estimator_
# Prediction on train data
GRF_train = GRF_best_model.predict(X_train)

# Prediction on test data
GRF_test = GRF_best_model.predict(X_test)

# Lets check Model performance on both test and train using error metrics of regression like mape,r square value
# MAPE for train data
MAPE_train= MAPE(y_train,GRF_train)

# MAPE for test data
MAPE_test= MAPE(y_test,GRF_test)
```

```
# Rsquare for train data
r2_train= r2_score(y_train,GRF_train)

# Rsquare for test data
r2_test=r2_score(y_test,GRF_test)

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

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

print("Best Parameter="+str(best_parameters))
print("Best Model="+str(GRF_best_model))
print("Mean Absolute Precentage Error for train data="+str(MAPE_train))
print("Mean Absolute Precentage 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))
Error_MetricsGSRF = {'Model Name': ['Grid search CV Random Forest'],
                    'MAPE_Train':[MAPE_train],
                    'MAPE_Test':[MAPE_test],
                    'R-squared_Train':[r2_train],
                    'R-squared_Test':[r2_test],
                    'RMSE_train':[RMSE_train],
                    'RMSE_test':[RMSE_test]}

GridSearchRandomForest_Results = pd.DataFrame(Error_MetricsGSRF)
GridSearchRandomForest_Results
Final_Results = pd.concat([LinearRegression_Results,
                           DecisionTree_Results,
                           RandomDecisionTree_Results,
                           GridDecisionTree_Results,
                           RandomForest_Results,
                           RandomSearchRandomForest_Results,
                           GridSearchRandomForest_Results,], ignore_index=True, sort =False)
```

Final_Results

 # From results Random Forest model have optimum values and this algorithm is good for our data

```
Final_Results
```

	Model Name	MAPE_Train	MAPE_Test	R-squared_Train	R-squared_Test	RMSE_train	RMSE_test
0	Linear Regression	45.480011	17.528674	0.835720	0.847770	776.486700	782.728203
1	Decision Tree	81.433513	29.346978	0.669888	0.725704	1100.706423	1050.678261
2	Random Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
3	Grid Search CV Decision Tree	8.834388	19.961435	0.929061	0.799082	510.252314	899.227765
4	Random Forest	16.726797	17.056622	0.981439	0.871775	261.001514	718.368952
5	Random Search CV Random Forest	17.740802	17.042779	0.981037	0.874703	263.810977	710.117599
6	Grid search CV Random Forest	21.536863	17.406332	0.964378	0.872677	361.574910	715.836573

6. REFERENCES

1) Edwisor Learning
2) www.analyticsvidhya.com
3) www.geeksforgeeks.org
4) towardsdatascience.com
5) rbloggers.com
6) Kaggle.com
7) github.com