BIKE RENTING

FEBRUARY 11, 2019

AKASH HEMANT SAXENA

Contents

Chapter 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. As it gets easy for an organisation to arrange the resource if the demand spikes.

1.2 Data

The task is to build a regression model which will predict the number of count with respect to other parameters.

Given below is the sample of the data set which is used to predict count of bike rented.

Following table shows all the features of the data set:

instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit
1	01-01-2011	1	0	1	0	6	0	2
2	02-01-2011	1	0	1	0	0	0	2
3	03-01-2011	1	0	1	0	1	1	1
4	04-01-2011	1	0	1	0	2	1	1
5	05-01-2011	1	0	1	0	3	1	1
6	06-01-2011	1	0	1	0	4	1	1
7	07-01-2011	1	0	1	0	5	1	2
8	08-01-2011	1	0	1	0	6	0	2

temp	atemp	hum	windspeed	casual	registered	cnt
0.344167	0.363625	0.805833	0.160446	331	654	985
0.363478	0.353739	0.696087	0.248539	131	670	801
0.196364	0.189405	0.437273	0.248309	120	1229	1349
0.2	0.212122	0.590435	0.160296	108	1454	1562
0.226957	0.22927	0.436957	0.1869	82	1518	1600
0.204348	0.233209	0.518261	0.089565	88	1518	1606
0.196522	0.208839	0.498696	0.168726	148	1362	1510
0.165	0.162254	0.535833	0.266804	68	891	959

As per the above table following are the variable using which the regression model has to predict count.
instant: Record index
dteday: Date
season: Season (1:springer, 2:summer, 3:fall, 4:winter)
yr: Year (0: 2011, 1:2012)
mnth: Month (1 to 12)
hr: Hour (0 to 23)
holiday: weather day is holiday or not (extracted fromHoliday Schedule)
weekday: Day of the week
workingday: If day is neither weekend nor holiday is 1, otherwise is 0.
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
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)
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)
hum: Normalized humidity. The values are divided to 100 (max)
windspeed: Normalized wind speed. The values are divided to 67 (max)
casual: count of casual users
registered: count of registered users
cnt: count of total rental bikes including both casual and registered

As per the above dataset, respect variable has redundant values:
1. dteday - this feature consists of the date, moreover we already has year and month in our data set.
2. casual & registered - addition of casual and registered is equal to cnt
3. Instant - We don't want the result to be bias based on the instant, ML algorithm will treat instant as numerical data.

Chapter 2
Methodology

2.1 Pre-processing

The output of the Machine Learning data is fully depended upon the data feed in the algorithm, ML algorithm does not generate the proper outcome on raw data. So it's important to transform the data so that the model accuracy of the model can be increased. In other words, we must apply some cleaning and processing for a better outcome with respect to the Machine learning algorithm. Some Machine learning algorithm required well-processed data like Decision Tree and random forest does not take null value. So it is important to process the data before feeding it in the respective machine learning algorithm.

Following are the pre-processing operation performed on the data to reduce the error rate and produce the optimal output.

2.1.1 Missing Value Analysis.

There are several options to handle missing value, but it mainly depends upon the nature of the data set, which missing analytics produce the optimum solution. Missing of data can occur due to nonresponse occur for example privacy concern. Moreover, if the value is missing completely at random, the data sample still represents the population but if the value is missing in some pattern, analysis produces bias output. There is two form of random missing values: MCAR and MAR. MCAR exists when the missing values are randomly distributed across all observations. This form can be confirmed by partitioning the data into two parts: one set containing the missing values, and the other containing the non-missing values. The second form is missing at random (MAR). In MAR, the missing values are not randomly distributed across observations but are distributed within one or more subsamples. Dropping the missing value is the good option if the data set has a low percentage of missing value and remaining data set can be used for further processing but it has its cons, dropping the missing value observation reduce the quantity of data. If our data set has a large number of missing value with respect to observation or feature, dropping is a good option. **Data set doesn't have missing value.**

Moreover, missing value analysis is done after outlier analysis due to the presence of outlier. Further description will be in Outlier analysis section.
Following code represent the missing value checking:

Python:

```
#missing value check
Org_data.isna().sum()
# there is no missing values in the data set
```

```
Output:
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
cnt           0
dtype: int64
```

R code:

```
> anyNA(new_dataset)
[1] FALSE
> # RESULT IS FALSE NO MISSING VALUE PRESENT IN THE RESPECTIVE DATA SET
```

2.1.2 Outlier Analysis

Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

We have used box and Whisker method to detect the outliers and performed the operation over it.

Following code is used to detect the outliers in the data set:

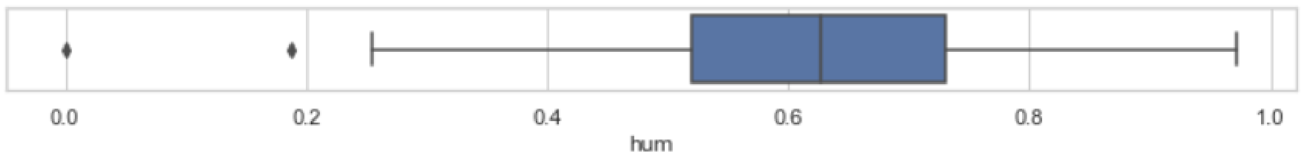
Python:

```
for i in Numerical_col:
    q75,q25 = np.percentile(Org_data.loc[:,i],[75,25])
    iqr=q75-q25
    l_fence,u_fence = round(q25-(iqr*1.5),4),round(q75+(iqr*1.5),4)
    total_outlier = len([num for num in Org_data[i] if num > u_fence or num < l_fence ])
    if total_outlier != 0:
        print ('OUTLIER PRESENT IN ',i)
        plt.figure(figsize=(12,0.8))
        sns.boxplot(Org_data[i])
        plt.show()
```

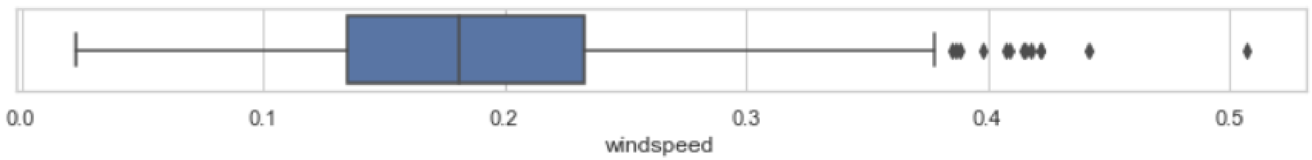
```
        print('25%:',q25,'  Median:',Org_data[i].median(),'   75%:',q75,'   IQR:',iqr)
        print('Minimum:',Org_data[i].min(),'    Lower Fence:',l_fence,'    Upper fence:',u_fence,'
Maximum:',Org_data[i].max())
        print('Number of Outliers:',total_outlier)

    else: print('++++++++----NO OUTLIER IN --->',i)
```

Output:

```
OUTLIER PRESENT IN  hum
```




```
25%: 0.52   Median: 0.626667    75%: 0.7302085    IQR: 0.21020850000000002
Minimum: 0.0      Lower Fence: 0.2047      Upper fence: 1.0455      Maximum: 0.9725
Number of Outliers: 2
+++==================================================================================+++
OUTLIER PRESENT IN  windspeed
```




```
25%: 0.13495   Median: 0.180975    75%: 0.2332145    IQR: 0.0982645
Minimum: 0.0223917     Lower Fence: -0.0124     Upper fence: 0.3806     Maximum: 0.507463
Number of Outliers: 13
```

Methodology of detecting the outliers:

As show in the above figure the points/terms located outside the outer fences of the box and Whisker graph are considered as the outliers. Outliers are the extreme low and extreme high value in the respective dataset.

Calculation for outlier detection:

1. 25% and 75% of the value is obtain from the data set.
   (25% of the data set is obtain by split the data from its median and finding the median of split data set.)
   | Median (25%) | Median (50%) | Median (75%) |
2. Finding the interquartile range (IQR = 75%(value) - 25%(Value))
3. Finding the lower fence and upper fence:
   Lower fence = 25%(value) – IQR*1.5
   Upper fence = 75%(value) – IQR*1.5

The value which are smaller the lower fence and bigger then upper fence are consider as outliers.

As there where outliers in our data set we have following option to deal with the outliers:

1. Delete the outliers and perform the further pre-processing
2. Replace the outliers with NA and impute the missing terms with imputation technique.

Procedure perform form finding the outliers:

1. Deleting the outliers:
   Following code is used to delete the outlier from the data set:
   Python:

```
Process_data = Org_data.copy()
for i in ['hum','windspeed']:
            print(i)
            q75, q25 = np.percentile(Process_data.loc[:,i], [75 ,25])
            iqr = q75 - q25
            min, max = q25 - (iqr*1.5),q75 + (iqr*1.5)
            print('Maximim:', max)
            print('Minimum:', min)
            Process_data = Process_data.drop(Process_data[Process_data.loc[:,i] <
         min].index)
            Process_data = Process_data.drop(Process_data[Process_data.loc[:,i] >
         max].index)
    print(i, '--> DONE')
```

   Output:

```
hum
Maximim: 1.0455212500000002
Minimum: 0.20468725
hum --> DONE
windspeed
Maximim: 0.380585
Minimum: -0.012431000000000025
windspeed --> DONE
```

   R code:

```
Process1 =  new_dataset
for(i in c('hum', 'windspeed'))
{
  val = Process1[,i][Process1[,i] %in% boxplot.stats(Process1[,i])$out]
  Process1 = Process1[which(!Process1[,i] %in% val),]
} # all the outliear got removed after applying this loop
```

2. Replacing the outliers with NA and imputation technique is used the replace the missing values.
   Following are the code the replace the outliers with NAN and KNN imputation is used to replace the missing terms.
   Python:

```
         Process_2 = Org_data.copy()
         for i in ['hum','windspeed']:
            print(i)
            q75_1, q25_1 = np.percentile(Process_2.loc[:,i], [75 ,25])
            iqr2 = q75_1 - q25_1
            minimum, maximum = q25_1 - (iqr2*1.5),q75_1 + (iqr2*1.5)
            print('Maximim:', maximum)
            print('Minimum:', minimum)
```

```
        Process_2[i] = np.where(Process_2[i] < minimum , np.nan, Process_2[i])
    Process_2[i] = np.where(Process_2[i] > maximum , np.nan, Process_2[i])
```

All the outliers are replacing with the NAN value.
Following code perform the KNN imputation:

```
from fancyimpute import KNN
Process_2 = pd.DataFrame(KNN(k = 3).complete(Process_2),
columns=Process_2.columns)
```

KNN imputation method checks all the nearest variable with respect to the missing value. KNN algorithm used for the matching point with the closest k neighbour in the multi-dimension. It can be used for data that are categorical, discrete and continuous which makes it more efficient with all kind of missing data.
KNN imputation uses the distance formula to replace the missing value.
Following are the formula used by KNN imputation to replace the missing value.
Euclidean distance:

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \, .$$

Reason for not performing the statistical imputation technique:
Statistical imputation technique involves mean, median and mode method to replace the missing value with the respective dataset and mean, median and mode are considered to be central tendency of a data set. On other hand outliers are the extreme low and extreme high values of a data set. The difference will be fare greater if statistical imputation will be performed as compare to the imputation method.
Moreover, only 2% of a data set has outliers. So rather than performing on falsely imputed data set it better to remove all the outliers and perform the operation on original data set.

2.1.3 Feature Selection

Feature selection is another pre-processing technique which decreases the load over machine learning algorithm checking the correlation between other feature and check which feature is highly correlated to another feature. If two feature carries the same data, this will create a bias output. Feature selection is a process where you can select those features which contribute most to the prediction output. Moreover, having a relevant feature in the data set can decrease the accuracy of the model.

Following correlation figure shows the correlation between the numerical variables. As suspected, temp and atemp are highly correlated with each other (0.99) as the definition of the feature was the temperature on particular instance and atemp was feeling temperature.

Chi- square analysis is not performed on this data set as dependent variable is continuous variable and chi-square analysis can only be performed on two categorical data set.

After performing the above operation ‘temp’ feature is dropped from the data set.

Correlation formula used to perform the respective analysis is as follows:

$$r = \frac{N\Sigma xy - (\Sigma x)(\Sigma y)}{\sqrt{[N\Sigma x^2 - (\Sigma x)^2][N\Sigma y^2 - (\Sigma y)^2]}}$$

Where:

N	=	number of pairs of scores
Σxy	=	sum of the products of paired scores
Σx	=	sum of x scores
Σy	=	sum of y scores
Σx^2	=	sum of squared x scores
Σy^2	=	sum of squared y scores

2.1.4 Feature Scaling

Feature scaling is a method to standardize the variable or the feature of the data set, It also knows as standardization of a data set. Some feature has magnitudes, unit or range and normalization should be performed when the scaling of the feature is irrelevant or have a high magnitude which will lead to the bases in the machine learning algorithm which will lean towards the higher magnitude dataset. Z-scoring/standardization is one technique in which the value of a feature is subtracted with the mean of the feature and their subtraction is divided by the standard deviation of the feature which will also set the feature in normalized bell shape form whereas another method involves the value of a feature is subtracted by the minimum value of the respective feature which is divided by the subtraction of the minimum value and maximum value of the respective feature.

Following are the formula to perform normalization and z-score of a variable:

Min-Max scaling:

Yi = [Xi - min(X)]/[max(X) - min(X)]

Z-score scaling:(For normally distributed data)

Z = (Xi – Mean(X))/ Standard Deviation(X)

As we only have atemp, hum, windspeed numerical variable which were already normalized so no further operation where perform in feature scaling.

2.2 Modelling

2.1.1 Model Selection

In our earlier stage of analysis, we came to know that our dependent variable in our case is cnt is numerical, so we are using a Regression analysis over our data set.
Following are the machine learning algorithm are performing on our data set to predict the future data.

2.1.2 Linear Regression

Linear regression is a linear model, a model that assumes a linear relationship between the independent variables (x) and the single dependent variable (y). More specifically, that y can be calculated from a linear combination of the input variables (x).
Following is the formula to calculate the Linear equation:

$$Y = b_0 + b_1X_1 + b_2X_2 \ldots\ldots\ldots\ldots\ldots\ b_NX_N$$

Where:
Y – Dependent variable
X_1......X_N – independent variable
b0 – intercept
b_1....... b_N – Coefficient
To find the optimal line Ordinary least squares is used. OLS draws the line which generating the least error while predicting the dependent variable.
Following are the code for generating the Linear Regression ML:
Python:

```
LRmodel = LinearRegression().fit(x_train,y_train)
LR_estimated = LRmodel.predict(x_test)
LR_result = reg_acc(y_test,LR_estimated)
# Linear Regression Result
print(LRmodel.get_params)
print('==================================================')
print('**Intercept: ',LRmodel.intercept_)
for col,coef in zip(x_train.columns,LRmodel.coef_):
    print (col, coef)
```

Output:

```
RMSE of data:  841.815220965525
R² :  0.8423552833944167
MAE: 610.0950826997182
RMSLE: 0.2376876631453713
MAPE : 17.9779
============================
Model Accuracy(%) : 82.0221
============================
<bound method BaseEstimator.get_params of LinearRegression(copy_X=True, fit_inter
cept=True, n_jobs=None,
```

```
         normalize=False)>
===============================================
**Intercept:  1465.364492179624
season 488.8434524249345
yr 1997.1782405909955
mnth -33.817800465942284
holiday -475.1422222035401
weekday 58.531990989184514
workingday 98.69463945941224
weathersit -483.99550451866145
atemp 5838.333351561126
hum -1340.5315112611815
windspeed -2368.713824760764
```

R code:

```
Linear1 = lm(cnt~., data = train) # Linear Regression model generation
Linear_predict1 = predict(Linear1, test[,-11]) # Prediction of the varaible for cross validation of an algorithm
lr_result = cal_result(test$cnt,Linear_predict1) # storing the generated variable for further corss varification with other ML models
summary(Linear1)
```

Output:

```
Call:
lm(formula = cnt ~ ., data = train)

Residuals:
    Min      1Q  Median      3Q     Max 
-3949.1  -339.1    59.1   487.1  2855.5 

Coefficients: (1 not defined because of singularities)
             Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1273.457    213.442   5.966 4.37e-09 ***
season2       779.206    202.896   3.840 0.000137 ***
season3       777.200    249.891   3.110 0.001968 ** 
season4      1487.820    213.466   6.970 9.20e-12 ***
yr1          2084.713     67.136  31.052  < 2e-16 ***
mnth2         180.059    171.198   1.052 0.293376    
mnth3         675.313    189.872   3.557 0.000408 ***
mnth4         683.912    281.067   2.433 0.015284 *  
mnth5         940.187    299.260   3.142 0.001771 ** 
mnth6         822.162    313.167   2.625 0.008900 ** 
mnth7         316.038    350.676   0.901 0.367867    
mnth8         767.074    338.565   2.266 0.023864 *  
mnth9        1205.478    302.116   3.990 7.51e-05 ***
mnth10        694.732    283.437   2.451 0.014555 *  
mnth11        -53.241    267.960  -0.199 0.842579    
mnth12          4.709    215.367   0.022 0.982562    
holiday1     -610.950    197.326  -3.096 0.002061 ** 
weekday1      313.278    127.656   2.454 0.014436 *  
weekday2      338.725    122.216   2.772 0.005770 ** 
weekday3      460.981    122.298   3.769 0.000182 ***
weekday4      378.418    126.682   2.987 0.002943 ** 
weekday5      511.965    123.450   4.147 3.90e-05 ***
weekday6      516.332    124.843   4.136 4.10e-05 ***
workingday1        NA         NA      NA       NA    
weathersit2  -486.544     89.923  -5.411 9.42e-08 ***
weathersit3 -1681.400    224.744  -7.481 2.96e-13 ***
atemp        3486.763    383.383   9.095  < 2e-16 ***
hum         -1157.130    250.947  -4.611 5.00e-06 ***
windspeed    -903.330    176.028  -5.132 4.00e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 780.5 on 545 degrees of freedom
Multiple R-squared:  0.8468,   Adjusted R-squared:  0.8392
F-statistic: 111.5 on 27 and 545 DF,  p-value: < 2.2e-16
```

All the variables are highly significant, Moreover, Adjusted R-squared is 0.8392 which means our model is explaining 83% of variance of our data

2.2.3. Decision Trees

Decision Tree algorithm creates a flow chat like tree structure where end nodes denote the outcome and the sub-nodes denote the decision flow of the tree.

Following are the code for generating the Decision Trees ML:

Python:

```
DTree_model = tree.DecisionTreeRegressor().fit(x_train,y_train)
DTree_estimated = DTree_model.predict(x_test)
Decision_result = reg_acc(y_test,DTree_estimated)
for col, per in zip(x_train,DTree_model.feature_importances_):
    print(col, round(per*100,2))
print('PARAMETER IN PERCENTAGE')
```

Output:

```
RMSE of data:  744.5747807753542
R² :  0.8766717648516062
MAE: 527.4791666666666
RMSLE: 0.24754600308429928
MAPE : 16.4318
============================
Model Accuracy(%) : 83.5682
============================
season 8.44
yr 29.98
mnth 3.68
holiday 0.14
weekday 1.06
workingday 0.84
weathersit 1.18
atemp 41.56
hum 9.0
windspeed 4.11
PARAMETER IN PERCENTAGE
```

The above parameter tells what percentage of variable are used to predict the dependent variable.

R code:

```
library(rpart)
Dtree1 = rpart(cnt ~., data = train, method = 'anova') # Decision Tree generation
Dtree_predict1 = predict(Dtree1, test[,-11])# Prediction of the varaible for cross validation of an algorithm
Dtree_result = cal_result(test$cnt,Dtree_predict1)# storing the generated variable for further corss varification with other ML models
summary(Dtree1)
```

```
Call:
rpart(formula = cnt ~ ., data = train, method = "anova")
  n= 573

           CP nsplit rel error    xerror       xstd
1  0.37720194      0 1.0000000 1.0020814 0.04393473
2  0.22105148      1 0.6227981 0.6760653 0.03417039
3  0.09164321      2 0.4017466 0.4432588 0.02907907
4  0.04495556      3 0.3101034 0.3416344 0.02226809
5  0.02750149      4 0.2651478 0.3121943 0.02322349
6  0.02229592      5 0.2376463 0.2846296 0.02274376
7  0.01651331      6 0.2153504 0.2481376 0.02044031
8  0.01210973      7 0.1988371 0.2401634 0.02059844
9  0.01055959      8 0.1867273 0.2385907 0.02057866
10 0.01000000      9 0.1761678 0.2351976 0.02067551

Variable importance
     atemp       mnth         yr     season        hum  windspeed weathersit
        27         23         19         19          7          4          1
```

Output:
The above parameter tell how much to particular variable is significant for the prediction in the respective model.

2.2.4. Random Forest

Similar to the decision tree, we also applied a random forest algorithm to check the prediction rate of a data set. Random forest works the same as decision tree but rather than creating a single tree, random forest creates multiple trees corresponding to the data set. Moreover, it fully depends on developer, the number of tree to be grown. More tree will be equal to the increase in accuracy of a model until the error rate stops decreasing. So we have generated multiple tree and checked the optimum number of tree required until error stops decreasing.

Following are the code for generating the Random Forest ML:

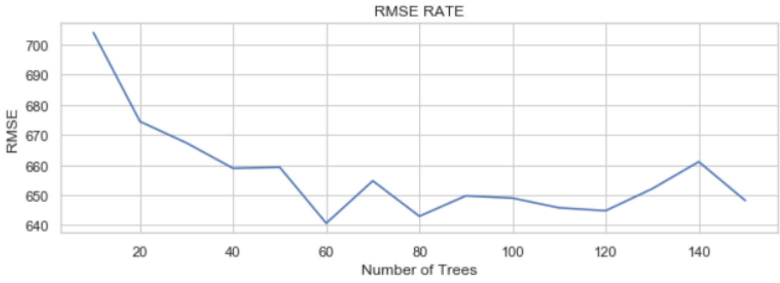
Python:

```
random_tree = [i*10 for i in range(1,16)]
rmse , r_sq , mae , rmsle , mape = [],[],[],[],[]
for tree_size in random_tree:
    print('Tree Size:', tree_size)
    random_model                                                        =
RandomForestRegressor(n_estimators=int(tree_size)).fit(x_train,y_train)
    estimation = random_model.predict(x_test)
    result = reg_acc(y_test,estimation)
    rmse.append(result[0])
    r_sq.append(result[1])
    mae.append(result[2])
```

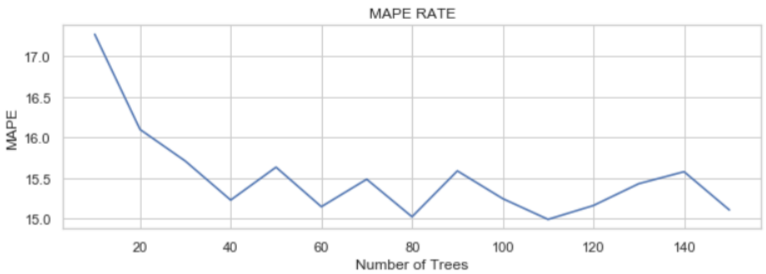
```
    rmsle.append(result[3])
    mape.append(result[4])
```

And following graph are generated to evaluate the respective number of trees:
RMSE, MAPE, MAE R-Square Error Rate:

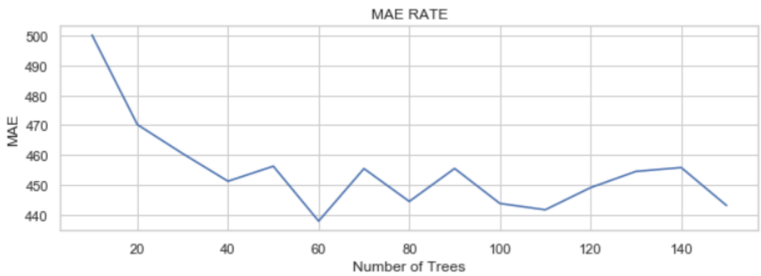
```
In [50]: plt.figure(figsize=(10,3))
         plt.title('RMSE RATE')
         plt.xlabel('Number of Trees')
         plt.ylabel('RMSE')
         sns.lineplot(x=random_tree,y=rmse)

Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x1e495898240>
```



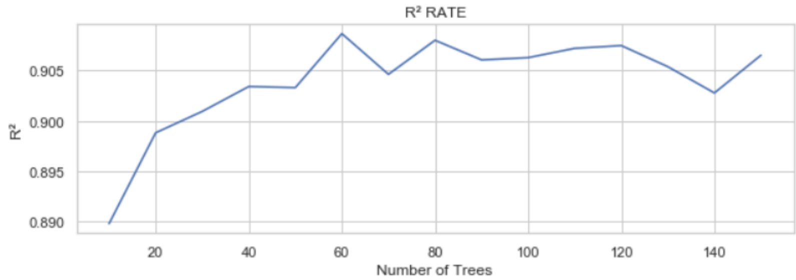

```
In [52]: plt.figure(figsize=(10,3))
         plt.title('MAPE RATE')
         plt.xlabel('Number of Trees')
         plt.ylabel('MAPE')
         sns.lineplot(x=random_tree,y=mape)

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x1e4957ed2e8>
```




```
In [56]: plt.figure(figsize=(10,3))
         plt.title('MAE RATE')
         plt.xlabel('Number of Trees')
         plt.ylabel('MAE')
         sns.lineplot(x=random_tree,y=mae)

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x1e4954c2358>
```



```
In [51]: plt.figure(figsize=(10,3))
         plt.title('R² RATE')
         plt.xlabel('Number of Trees')
         plt.ylabel('R²')
         sns.lineplot(x=random_tree,y=r_sq)

Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1e4958a1908>
```



With respect to the above graph we found that we got the optimum number are 40 – 60. So we can generate 40 to 60 trees and after that the error rate won't fall too much.

R code:

```
rf1 = randomForest(x = train[,-11], y = train$cnt, importance = TRUE, ntree = 500) # RANDOM FOREST generation
rf_predict1 = predict(rf1, test[,-11])# Prediction of the varaible for cross validation of an algorithm
rf_result = cal_result(test$cnt,rf_predict1)# storing the generated variable for further corss varification with other ML models
rf1
```

Output:

```
Call:
 randomForest(x = train[, -11], y = train$cnt, ntree = 500, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 3

          Mean of squared residuals: 483974.2
                    % Var explained: 87.2
```

2.2.5 K-Nearest Neighbours

Similar to the imputation technique, KNN uses distance formula to predict the dependent variable.

Following are the code for generating the K-Nearest Neighbours ML:

Python:

```
knn_model = KNeighborsRegressor(n_neighbors= 5).fit(x_train,y_train)
knn_estimated = knn_model.predict(x_test)
knn_result = reg_acc(y_test,knn_estimated)
```

Output:

```
RMSE of data:   864.9603117099266
R² :  0.8335674630773875
MAE: 669.7819444444444
```

```
RMSLE: 0.2594939488633473
MAPE : 20.9545
============================
Model Accuracy(%) : 79.0455
============================
```

R code:

```
knn1 = knnreg(train[,-11], train$cnt, k = 3) # K-NEAREST NEIGHBORS generation
knn_predict1 = predict(knn1, test[,-11])# Prediction of the varaible for cross validation of an algorithm
knn_result = cal_result(test$cnt,knn_predict1)# storing the generated variable for further corss varification with other ML models
summary(knn1)
```

Output:

```
        Length Class  Mode
learn   2      -none- list
k       1      -none- numeric
theDots 0      -none- list
```

3. Conclusion

3.1. Model Evaluation

As this is a regression model prediction, we are using MSE/RMSE and MAE to predict the accuracy and how well the value is predicted with respect to the machine learning algorithm. MAPE was also used as cnt value doesn't have value 0. Other parameter are also taken into consideration.

Python Model Evaluation:

Python Code:

```
result = {'linear Reg(OD)':LR_result,'Decision Tree(OD)':Decision_result,'Random Forest(OD)':rf_result,'KNN(OD)': knn_result}

result_dataset = pd.DataFrame(result)
result_dataset.index = ['RMSE','R²','MAE','RMSLE','MAPE','ACCURACY(%)']

result_dataset['linear Reg(IP)']     = LR2_result
result_dataset['Decision Tree(IP)'] = Decision_result2
result_dataset['Random Forest(IP)'] = rf_result2
result_dataset['KNN(IP)']           = knn_result2

# OD : Result after Outlier deletion
# IP : Result after Outlier imputation
result_dataset
```

Output:

]:

	linear Reg(OD)	Decision Tree(OD)	Random Forest(OD)	KNN(OD)	linear Reg(IP)	Decision Tree(IP)	Random Forest(IP)	KNN(IP)
RMSE	841.815221	744.574781	645.232343	864.960312	900.384887	961.933689	649.832601	957.438806
R²	0.842355	0.876672	0.907386	0.833567	0.809390	0.782439	0.900713	0.784468
MAE	610.095083	527.479167	440.895602	669.781944	679.208776	680.789116	489.630272	745.200000
RMSLE	0.237688	0.247546	0.221842	0.259494	0.245764	0.529241	0.263177	0.334724
MAPE	17.977900	16.431800	15.018400	20.954500	19.797500	21.902100	19.040700	28.318600
ACCURACY(%)	82.022100	83.568200	84.981600	79.045500	80.202500	78.097900	80.959300	71.681400

R Model Evaluation:

R code:

```
result = data.frame(knn_result,Dtree_result,rf_result,lr_result) # genreating the accuracy chart with respect to all the ML result
row.names(result) = c('rsq','mae','rmse','rmsle') # Naming the rows for clearing
```

Output:

	knn_result	Dtree_result	rf_result	lr_result
rsq	0.7682691	0.7412824	0.8755315	0.8374623
mae	632.6140046	683.2932687	453.2274137	539.4555701
rmse	905.1157539	956.3683277	663.3487513	758.0349224
rmsle	0.5132202	0.5188964	0.4739400	0.4892360

3.2 Model Selection

Random Forest machine learning algorithm produces the best outcome with respect to the data set and the prediction of 'cnt' was predicted quite accurately.

In both R and Python Mean absolute error is quite low compare to the other model. Moreover, MAPE which measure the accuracy based on percentage is quite high in respect to other machine learning algorithm.

Appendix A – Python Code

```
# # library import and data importing

# IMPORTING ALL THE LIBRARY
import os
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from fancyimpute import KNN
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import tree

##############################################################################
######################################
def dataloss(data1,data2):
    print('### DEFERENCE IN DATA SET')
    print('Feature deleted: ', data1.shape[1]-data2.shape[1])
    print('Observation deleted: ',data1.shape[0]-data2.shape[0])

##############################################################################
######################################
# Return MAE, MRSE, R², Adjusted R²
def reg_acc(y_true, y_pre):
    return_var = []
    from math import sqrt
    from sklearn.metrics import mean_squared_log_error
    rmse = sqrt(mean_squared_error(y_true,y_pre))
    return_var.append(rmse)
    print ("RMSE of data: ",rmse )
    r2 = r2_score(y_true,y_pre)
    return_var.append(r2)
    print ("R² : ",r2 )
    mae = mean_absolute_error(y_true,y_pre)
    return_var.append(mae)
    print ('MAE:',mae)
    rmsle =np.sqrt(mean_squared_log_error( y_true, y_pre ))
    return_var.append(rmsle)
    print('RMSLE:',rmsle)
    if 0 in y_true :
        print("MAPE can't be calculated")
        return_var.append(0)
    else :
        mape = round(np.mean(np.abs((y_true - y_pre)/y_true))*100,4)
        print ('MAPE :', mape)
        print('============================')
        print('Model Accuracy(%) :', 100-mape)
        print('============================')
        return_var.append(mape)
        return_var.append(100-mape)
    return return_var
```

```
##############################################################################
#######################################
#def ML(x,y):
#    from sklearn.model_selection import train_test_split
#    x_train,x_test,y_train,y_test = train_test_split(x,y, test_size =
0.2,random_state = 42)
#    model_name = ['Linear Regression','KNN','Decision Tree','Random Forest']
#    model = [LinearRegression(),KNeighborsRegressor(n_neighbors=5),
#
tree.DecisionTreeRegressor(),RandomForestRegressor(n_estimators=20)]
#    modeling = []
#    for i in range(len(model)):
#        mlmodel = model[i].fit(x_train,y_train)
#        y_predict = mlmodel.predict(x_test)
#        print(model_name[i])
#        reg_acc(y_test,y_predict)
#        modeling.append(mlmodel)
##############################################################################
#######################################
# Normalized the the data set with respect to the columns porvided
def normalize(data,columns):
    for i in columns:
        print(columns)
        minimum , maximum = data[i].min(), data[i].max()
        data[i] = (data[i] - minimum)/(maximum - minimum)
    return data
##############################################################################
#######################################
# Visualized the predicted value with respect to actual
#def

# changes the current working directory
os.chdir('D:/Data Science/EDWISOR/2_PORTFOLIO/project 2')
# saving the process data
Org_data = pd.read_csv('day.csv', header = 0)
# Creating the categorical feature  categorical
categorical_columns = ['season', 'yr', 'mnth', 'holiday', 'weekday',
'workingday','weathersit']
# Created the loop for converting the feature to categorical data type
for i in categorical_columns:
    Org_data[i] = pd.Categorical(Org_data[i])
Org_data = Org_data.drop(['instant',"dteday",'casual','registered'],axis=1)
# Reason for dropping the features
# 1. dteday --> this feature consist of the date, moreover we already has
year and month in our data set.
# 2. casual & registered --> addition of casual and registered is equal to
cnt
# 3. Instant --> We don't want the result to be bias based on the instant, ML
algorithm will treat instant as numerical data.
# Numerical_col hold all the feature which has numerical data type
Numerical_col = [ i for i in Org_data.columns if i not in categorical_columns
]

Org_data.head(10)

for i in categorical_columns:
    sns.countplot(Org_data[i], )
    plt.show()


# Following inference are made with respect to above graphs:
```

```
# 1. Season, yr, month, weekday - events are equaly distributed with respect
to season
# 2. weather - more bike where rent in 1 _(Clear, Few clouds, Partly cloudy,
Partly cloudy)_ instence with respect of 2 _(Mist + Cloudy, Mist + Broken
clouds, Mist + Few clouds, Mist)_ and 3 _(Light Snow, Light Rain +
Thunderstorm + Scattered clouds, Light Rain + Scatteredclouds)_
#
Org_data.head(10)

Org_data.boxplot(figsize=(9,5))
plt.show()
# INFERENCE:
# Outlier present in the respective feature (hum, windspeed,casual)
# describe function is used to check different parameter with respect to the
data set
Org_data.describe()
# INFERENCE:
# With the help of the describe function, we can infer that feature scaling
is needed as the
# casual & registered has bigger scale with repect to temperature and wind
speed by checking
# the minimum value and maximum value of the feature

sns.set(style="whitegrid")
plt.figure(figsize=(20,10))
sns.lineplot(data=Org_data[['hum','temp','windspeed']] , palette="tab10",
linewidth=2.5,)


# ## Exploratory Data Analysis

# ### Missing value analysis
#missing value check
Org_data.isna().sum()
# there is no missing values in the data set


# ### Outlier Analysis
# visualing 'casual','hum','windspeed' data with box and wister method for
further analysis
for i in Numerical_col:
    q75,q25 = np.percentile(Org_data.loc[:,i],[75,25])
    iqr=q75-q25
    l_fence,u_fence = round(q25-(iqr*1.5),4),round(q75+(iqr*1.5),4)
    total_outlier = len([num for num in Org_data[i] if num > u_fence or num <
l_fence ])
    if total_outlier != 0:

print('+++=============================================================
===============+++')
        print ('OUTLIER PRESENT IN ',i)
        plt.figure(figsize=(12,0.8))
        sns.boxplot(Org_data[i])
        plt.show()
        print('25%:',q25,'  Median:',Org_data[i].median(),'   75%:',q75,'
IQR:',iqr)
        print('Minimum:',Org_data[i].min(),'    Lower Fence:',l_fence,'
Upper fence:',u_fence,'    Maximum:',Org_data[i].max())
        print('Number of Outliers:',total_outlier)

    else: print('++++++----NO OUTLIER IN --->',i)
```

```
# We have two choice for outliers.
# 1. Delete the outliers
# 2. Impute the outliers
#
# we will try to use first method and will check how it will affect our model
# Pdata_Out1 stands for Outlier processed data by deleting the outliers
Process_data = Org_data.copy()
for i in ['hum','windspeed']:
    print(i)
    q75, q25 = np.percentile(Process_data.loc[:,i], [75 ,25])
    iqr = q75 - q25
    min, max = q25 - (iqr*1.5),q75 + (iqr*1.5)
    print('Maximim:', max)
    print('Minimum:', min)
    Process_data = Process_data.drop(Process_data[Process_data.loc[:,i] <
min].index)
    Process_data = Process_data.drop(Process_data[Process_data.loc[:,i] >
max].index)
    print(i, '--> DONE')

dataloss(Org_data,Process_data)

# Let's replace the Outliers with NAN and use KNN imputation for imputing the
missing value
# Reason for using missing value analysis:
# Generally outliers are extreme low or extremely high varaible so
statistical imputation won't be as accurate as KNN
# as KNN use distance formula to impute the missing variable

Process_2 = Org_data.copy()
for i in ['hum','windspeed']:
    print(i)
    q75_1, q25_1 = np.percentile(Process_2.loc[:,i], [75 ,25])
    iqr2 = q75_1 - q25_1
    minimum, maximum = q25_1 - (iqr2*1.5),q75_1 + (iqr2*1.5)
    print('Maximim:', maximum)
    print('Minimum:', minimum)
    Process_2[i] = np.where(Process_2[i] < minimum , np.nan, Process_2[i])
    Process_2[i] = np.where(Process_2[i] > maximum , np.nan, Process_2[i])

Process_2.isna().sum()

from fancyimpute import KNN
Process_2 = pd.DataFrame(KNN(k = 3).complete(Process_2),
columns=Process_2.columns)

Process_2.isna().sum()


# ### Feature Selection
plt.figure(figsize=(10,10))
corr=Process_data.corr()
sns.heatmap(corr, annot=True, mask=np.zeros_like(corr,dtype=np.bool),

cmap=sns.diverging_palette(220,10,as_cmap=True),square=True,vmin=-1,vmax=1)


# With respect the above heatmap we can infer:
```

```
# 1. As expected temp is positively correlated with atemp --> drop temp
feature
# 2. cnt increses as the temp increases, so people tend to rent more bike as
the temperature is high
# 3. humidity and the windspeed are negatively correlated with the cnt. if
the air humidity and windspeed is high people will less likely to rent the
bike.
#
Process_data = Process_data.drop(['temp'],axis = 1)

Process_2 = Process_2.drop(['temp'],axis = 1)


# ## Model Development

# __Following model will be taken into considaration:__
# 1. Linear Regression
# 2. Decision Tree
# 3. Random Forest
# 4. K-Nearest Neighbors
# Spliting the data set
# Outlier Deleted
independent_data = Process_data.drop(['cnt'],axis = 1)
dependent_data = Process_data['cnt']
x_train,x_test,y_train,y_test =
train_test_split(independent_data,dependent_data,
                                                  test_size = 0.2,
random_state = 0)

ind_data = Process_2.drop(['cnt'],axis = 1)
dep_data = Process_2['cnt']
x2_train,x2_test,y2_train,y2_test = train_test_split(ind_data,dep_data,
                                                  test_size = 0.2,
random_state = 0)


# ### _1. Linear Regression_
# Outlier Deleted
LRmodel = LinearRegression().fit(x_train,y_train)
LR_estimated = LRmodel.predict(x_test)
LR_result = reg_acc(y_test,LR_estimated)
# Linear Regression Result
print(LRmodel.get_params)
print('=================================================')
print('**Intercept: ',LRmodel.intercept_)
for col,coef in zip(x_train.columns,LRmodel.coef_):
    print (col, coef)

LR2model = LinearRegression().fit(x2_train,y2_train)
LR2_estimated = LR2model.predict(x2_test)
LR2_result = reg_acc(y2_test,LR2_estimated)
# Linear Regression Result
print(LR2model.get_params)
print('=================================================')
print('**Intercept: ',LR2model.intercept_)
for col2,coef2 in zip(x2_train.columns,LR2model.coef_):
    print (col2, coef2)


# ### _2. Decision Tree_
# Outlier Deleted
```

```
DTree_model = tree.DecisionTreeRegressor().fit(x_train,y_train)
DTree_estimated = DTree_model.predict(x_test)
Decision_result = reg_acc(y_test,DTree_estimated)
for col, per in zip(x_train,DTree_model.feature_importances_):
    print(col, round(per*100,2))
print('PARAMETER IN PERCENTAGE')

DTree_model2 = tree.DecisionTreeRegressor().fit(x2_train,y2_train)
DTree_estimated2 = DTree_model2.predict(x2_test)
Decision_result2 = reg_acc(y2_test,DTree_estimated2)
for col2, per2 in zip(x2_train,DTree_model2.feature_importances_):
    print(col2, round(per2*100,2))
print('PARAMETER IN PERCENTAGE')

# let's check the how tree grows
dotfile = open('pt.dot','w')
df = tree.export_graphviz(DTree_model, out_file= dotfile)


# ### _3. Random Forest_
# Outlier Deleted
random_tree = [i*10 for i in range(1,16)]
rmse , r_sq , mae , rmsle , mape = [],[],[],[],[]
for tree_size in random_tree:
    print('Tree Size:', tree_size)
    random_model =
RandomForestRegressor(n_estimators=int(tree_size)).fit(x_train,y_train)
    estimation = random_model.predict(x_test)
    result = reg_acc(y_test,estimation)
    rmse.append(result[0])
    r_sq.append(result[1])
    mae.append(result[2])
    rmsle.append(result[3])
    mape.append(result[4])

plt.figure(figsize=(10,3))
plt.title('RMSE RATE')
plt.xlabel('Number of Trees')
plt.ylabel('RMSE')
sns.lineplot(x=random_tree,y=rmse)

plt.figure(figsize=(10,3))
plt.title('R² RATE')
plt.xlabel('Number of Trees')
plt.ylabel('R²')
sns.lineplot(x=random_tree,y=r_sq)

plt.figure(figsize=(10,3))
plt.title('MAPE RATE')
plt.xlabel('Number of Trees')
plt.ylabel('MAPE')
sns.lineplot(x=random_tree,y=mape)

plt.figure(figsize=(10,3))
plt.title('MAE RATE')
plt.xlabel('Number of Trees')
plt.ylabel('MAE')
sns.lineplot(x=random_tree,y=mae)

# Outlier Deleted
RF_model = RandomForestRegressor(n_estimators= 60).fit(x_train,y_train)
```

```
RF_estimated = RF_model.predict(x_test)
rf_result = reg_acc(y_test,RF_estimated)
print('PARAMETER IN PERCENTAGE')
for col, per in zip(x_train,RF_model.feature_importances_):
    print(col, round(per*100,2))

RF_model2 = RandomForestRegressor(n_estimators= 60).fit(x2_train,y2_train)
RF_estimated2 = RF_model2.predict(x2_test)
rf_result2 = reg_acc(y2_test,RF_estimated2)
print('PARAMETER IN PERCENTAGE')
for col2, per2 in zip(x2_train,RF_model2.feature_importances_):
    print(col2, round(per2*100,2))


# ### _4. K-Nearest Neighbors_
# Outlier Deleted
knn_model = KNeighborsRegressor(n_neighbors= 5).fit(x_train,y_train)
knn_estimated = knn_model.predict(x_test)
knn_result = reg_acc(y_test,knn_estimated)

knn_model2 = KNeighborsRegressor(n_neighbors= 5).fit(x2_train,y2_train)
knn_estimated2 = knn_model2.predict(x2_test)
knn_result2 = reg_acc(y2_test,knn_estimated2)


# ### Accuaracy chart
result = {'linear Reg(OD)':LR_result,'Decision
Tree(OD)':Decision_result,'Random Forest(OD)':rf_result,'KNN(OD)':
knn_result}

result_dataset = pd.DataFrame(result)
result_dataset.index = ['RMSE','R²','MAE','RMSLE','MAPE','ACCURACY(%)']

result_dataset['linear Reg(IP)']    = LR2_result
result_dataset['Decision Tree(IP)'] = Decision_result2
result_dataset['Random Forest(IP)'] = rf_result2
result_dataset['KNN(IP)']           = knn_result2

# OD : Result after Outlier deletion
# IP : Result after Outlier imputation
result_dataset


# ## Final Inference:
# #### As per the above analysis and result generated shown in result
dataframe,
# #### Random Forest algorithm tend to provide the better result in repect to
this data set
# #### Moreover, after imputing the outliers with KNN imputation the accuracy
of all the algorithm decreases by 2%-5%
```

Appendix B – R Code

```
############# Project 2 - Bike Renting #############
# Assinging all the directories libraries
library(measures)
cal_result = function(test,predict){  # Calculate all the result of
regression model
  r1=measures::RSQ(test,predict)
  r2=measures::MAE(test,predict)
  r3=measures::RMSE(test,predict)
  r4=measures::RMSLE(test,predict)
  result= c(r1,r2,r3,r4)

  return(result)
}

setwd('D:/Data Science/EDWISOR/2_PORTFOLIO/project 2')
df <- read.csv(file = 'day.csv',header = TRUE)
# AFTER CHECK THE DATA SET WE ARE GOING TO REMOVE VARIABLES WHICH PROVIDING
THE SAME INFORMATION
# Reason for dropping the features
# 1. dteday --> this feature consist of the date, moreover we already has
year and month in our data set.
# 2. casual & registered --> addition of casual and registered is equal to
cnt
# 3. Instant --> We don't want the result to be bias based on the instant, ML
algorithm will treat instant as numerical data.
new_dataset<- subset(x = df,select = -c(instant,dteday,casual,registered))
new_dataset <- as.data.frame(new_dataset)
categorical_columns = c('season', 'yr', 'mnth', 'holiday',
'weekday','workingday', 'weathersit')
for(i in categorical_columns){new_dataset[,i] = as.factor(new_dataset[,i])} #
converting in categorical variable
# Let's check the data set and which inference can be taken out by
summerising the dataset
summary(new_dataset)
#season  yr          mnth     holiday weekday   workingday    weathersit
temp            atemp              hum            windspeed             cnt
#1:181   0:365   1      : 62   0:710   0:105   Min.   :0.000   1:463
Min.   :0.05913   Min.   :0.07907   Min.   :0.0000   Min.   :0.02239  Min.
:  22
#2:184   1:366   3      : 62   1: 21   1:105   1st Qu.:0.000   2:247      1st
Qu.:0.33708   1st Qu.:0.33784   1st Qu.:0.5200   1st Qu.:0.13495  1st
Qu.:3152
#3:188           5      : 62           2:104   Median :1.000   3: 21
Median :0.49833   Median :0.48673   Median :0.6267   Median :0.18097  Median
:4548
#4:178           7      : 62           3:104   Mean   :0.684
Mean   :0.49538   Mean   :0.47435   Mean   :0.6279   Mean   :0.19049  Mean
:4504
#                8      : 62           4:104   3rd Qu.:1.000              3rd
Qu.:0.65542   3rd Qu.:0.60860   3rd Qu.:0.7302   3rd Qu.:0.23321  3rd
Qu.:5956
#                10     : 62           5:104   Max.   :1.000
Max.   :0.86167   Max.   :0.84090   Max.   :0.9725   Max.   :0.50746  Max.
:8714
#                (Other):359           6:105

# BY STUDING THE ABOVE SUMMARY OF THE DATA SET WE CAN COME UP WITH THE
RESPECTIVE INFERENCE
# SEASON, YR, MNTH, WEEKDAY  - VARIABLE IS EQUALLY DISTRIBUTION
```

```
# HOLIDAY - BIKE RENTING IS MORE WHEN THE HOLIDAY VARAIBLE IS '0'
# WEATHERSIT - IT'S OBVIOUS THAT BIKE RENT IS HIGH IN CLEAR WEATHER

########################## EDA ##########################
############# MISSING VALUE CHECK #############
anyNA(new_dataset)
# RESULT IS FALSE NO MISSING VALUE PRESENT IN THE RESPECTIVE DATA SET

############# OUTLIER ANALYSIS #############
library(ggplot2)

numeric_index = sapply(new_dataset,is.numeric) #selecting only numeric
numeric_data = new_dataset[,numeric_index] # creating the dataset with
numerical varaible only
cnames = colnames(numeric_data) # fetching column names
for (i in 1:length(cnames)) # Ploting the boxplot
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "cnt"), data
= subset(new_dataset))+
           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="cnt")+
           ggtitle(paste("Box plot of cnt for",cnames[i])))
}

#gridExtra::grid.arrange(gn1,gn2,gn3, ncol = 3) # this boxplot doesn't have
any outliear
gridExtra::grid.arrange(gn4,gn5,gn6, ncol = 3)
# BY OBSERVING THE BOX PLOT
# gn4 : hum and gn5 : windspeed HAS THE OUTLIERS
# SO WE HAVE TWO CHOICE TO DEAL WITH THE OUTLIERS
# 1. DELETE THE OUTLIERS FROM THE DATA SET AND PERFORM THE FURTHER ANALYSIS
# 2. IMPUTE THE OUTLIER WITH IMPTATION TECHNIQUE OR ANY STATISTICAL TECHNIQUE

# Process1 will have the deletation technique
Process1 =  new_dataset
for(i in c('hum', 'windspeed'))
{
  val = Process1[,i][Process1[,i] %in% boxplot.stats(Process1[,i])$out]
  Process1 = Process1[which(!Process1[,i] %in% val),]
} # all the outliear got removed after applying this loop

# Process2 will have the imputation method
Process2 = new_dataset
for(i in c('hum', 'windspeed'))
{
  val = Process2[,i][Process2[,i] %in% boxplot.stats(Process2[,i])$out]
  Process2[,i][Process2[,i] %in% val] = NA
}

#Process2 = knnImputation(Process2,k=1)
#anyNA(Process2)
# In above code we have tried to impute the missing value with knnimputation
method but due to insufficicy in model, KNN imputation
# has generate the error.
    #Error in knnImputation(Process2, k = 1) :
      #Not sufficient complete cases for computing neighbors.
```

```
# Moreover, only 0.02 data was has the missing value. so dropping the
varaible and performing the further anaylsis will be the best option
# rather then impututtion it with some false data.
# further analysis and generating of machine learning algorithm is done on
Process1 data frame (by dropping the outliers)


############# FEATURE SELECTION #############
library(corrplot)
numerical_columns = c('temp','atemp','hum','windspeed','cnt')
corrplot(corr = cor(new_dataset[,numerical_columns]),method = 'number') #
ploting the correlation plot
# BY CHECKING THE CORRELATION CHART atemp AND temp IS HIGHLY CORRELATED WITH
EACHOTHER
# MOREOVER, BY CHECKING THE LAST PLOTS, temp, atemp HAS SOME AMOUNT OF
DEPENCENCE WITH RESPECT TO cnt
# AND HUM AND WINDSPEED HAS NEGATIVE DEPENDENCY TOWARDS cnt

# Deleting the temp feature
Process1 = subset(Process1, select = -temp)
#Process2 = subset(Process2, select = -temp)

############# FEATURE SCALING #############
# we have four numerical variable, let's check the histogram of the
respective feature
hist(Process1$atemp)
hist(Process1$hum)
hist(Process1$windspeed)
hist(Process1$cnt)
# INFERENCE:
# As THE DATA IS ALREADY NORMALIZED SO THERE IS NO NEED OF SCALING
########################## MODEL DEVELOPMENT ##########################

############# Spliting the dataset in test and train data ############
set.seed(123)
train_index = sample(1:nrow(Process1),0.8*nrow(Process1)) #Featching the 80%
index of the data set
train = Process1[train_index,]
test = Process1[-train_index,]

############################################################################
#
#Following model will be taken into considaration:
#1. Linear Regression
#2. Decision Tree
#3. Random Forest
#4. K-Nearest Neighbors


############# LINEAR REGRESSION #############
Linear1 = lm(cnt~., data = train) # Linear Regression model generation
Linear_predict1 = predict(Linear1, test[,-11]) # Prediction of the varaible
for cross validation of an algorithm
lr_result = cal_result(test$cnt,Linear_predict1) # storing the generated
variable for further corss varification with other ML models
summary(Linear1)
# Model is highly significance as the Adj R Square is above 80%
# Moreover all the variable are significant
############# DECISION TREE #############
library(rpart)
```

```
Dtree1 = rpart(cnt ~., data = train, method = 'anova') # Decision Tree
generation
Dtree_predict1 = predict(Dtree1, test[,-11])# Prediction of the varaible for
cross validation of an algorithm
Dtree_result = cal_result(test$cnt,Dtree_predict1)# storing the generated
variable for further corss varification with other ML models
summary(Dtree1)
############# RANDOM FOREST #############
library('randomForest')
rf1 = randomForest(x = train[,-11], y = train$cnt, importance = TRUE, ntree =
500) # RANDOM FOREST generation
rf_predict1 = predict(rf1, test[,-11])# Prediction of the varaible for cross
validation of an algorithm
rf_result = cal_result(test$cnt,rf_predict1)# storing the generated variable
for further corss varification with other ML models
rf1
# % Var explained: 87.2
############# K-NEAREST NEIGHBORS #############
library(caret)
knn1 = knnreg(train[,-11], train$cnt, k = 3) # K-NEAREST NEIGHBORS generation
knn_predict1 = predict(knn1, test[,-11])# Prediction of the varaible for
cross validation of an algorithm
knn_result = cal_result(test$cnt,knn_predict1)# storing the generated
variable for further corss varification with other ML models
summary(knn1)
########################## ACCUARACY CHECK ##########################
result = data.frame(knn_result,Dtree_result,rf_result,lr_result) # genreating
the accuracy chart with respect to all the ML result
row.names(result) = c('rsq','mae','rmse','rmsle') # Naming the rows for
clearing
```