PREDICATION OF BIKE RENTAL COUNT

- MOULIESWARAN R

- 15 June 2019

CONTENTS

Chapter 1

Introduction

1.1 Problem Statement

The Bike Rental Data contains the daily count of rental bikes between the year 2011 and 2012 with corresponding weather and seasonal information. We would like to predict the daily count of rental count in order to automate the system.

1.2 Data

Our task is to build Regression model which will give the daily count of rental bikes based on weather and season. Given below is a sample of the data set that we are using to predict the count:

Table 1.1: Bike Rental Sample Data (Columns: 1-8)

instant	dteday	Season	yr	mnth	holiday	weekday
1	1/1/2011	1	0	1	0	6
2	1/2/2011	1	0	1	0	0
3	1/3/2011	1	0	1	0	1
4	1/4/2011	1	0	1	0	2
5	1/5/2011	1	0	1	0	3

Table 1.2: Bike Rental Sample Data (Columns: 9-14)

weathersit	temp	atemp	Hum	windspeed	casual	registered	cnt
2	0.344167	0.363625	0.805833	0.160446	331	654	985
2	0.363478	0.353739	0.696087	0.248539	131	670	801
1	0.196364	0.189405	0.437273	0.248309	120	1229	1349
1	0.2	0.212122	0.590435	0.160296	108	1454	1562
1	0.226957	0.22927	0.436957	0.1869	82	1518	1600

The following variables which are used to predict the count of bike rental

Table 1.3: Bike Rental Predictors

s.no	Variables
1	Dteday
2	Season
3	Yr
4	Mnth
5	Holiday
6	Weekday
7	workingday
8	weathersit
9	Temp
10	Atemp
11	Hum
12	windspeed
13	Casual
14	registered

Chapter 2

Methodology

2.1 Data Analysis

Any predictive modelling requires that we look at the data before we start modelling. However, in data mining terms *looking at data* refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as **Exploratory Data Analysis**. To start this process we will first try and look at all the probability distributions of the variables.

2.1.1 Univariate Analysis

In Figure 2.1 and 2.2 we have plotted the probability density functions numeric variables present in the data including target variable cnt.

i. Target variable cnt is normally distributed

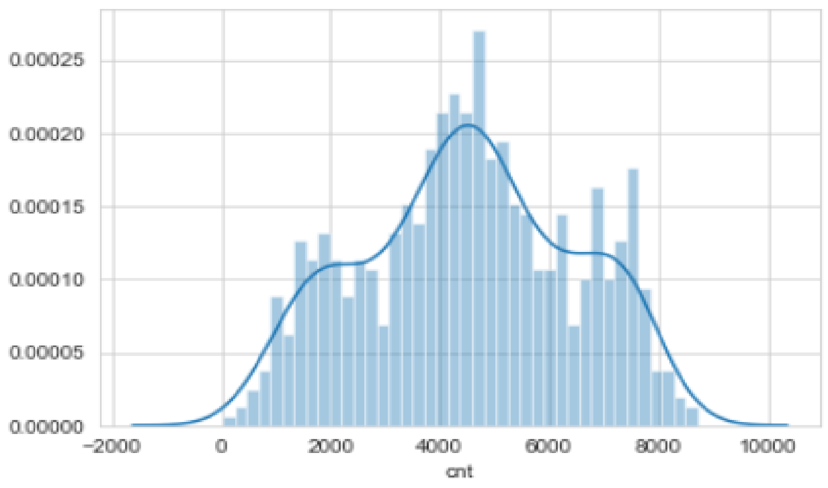
ii. Independent variables like ‘temp’,’atemp’, and ‘regestered’ data is distributed normally.

iii. Independent variable ‘casual’ data is slightly skewed to the right so, there is chances of getting outliers.

iv. Other Independent variable ‘hum’ data is slightly skewed to the left, here data is already in normalize form so outliers are discarded.

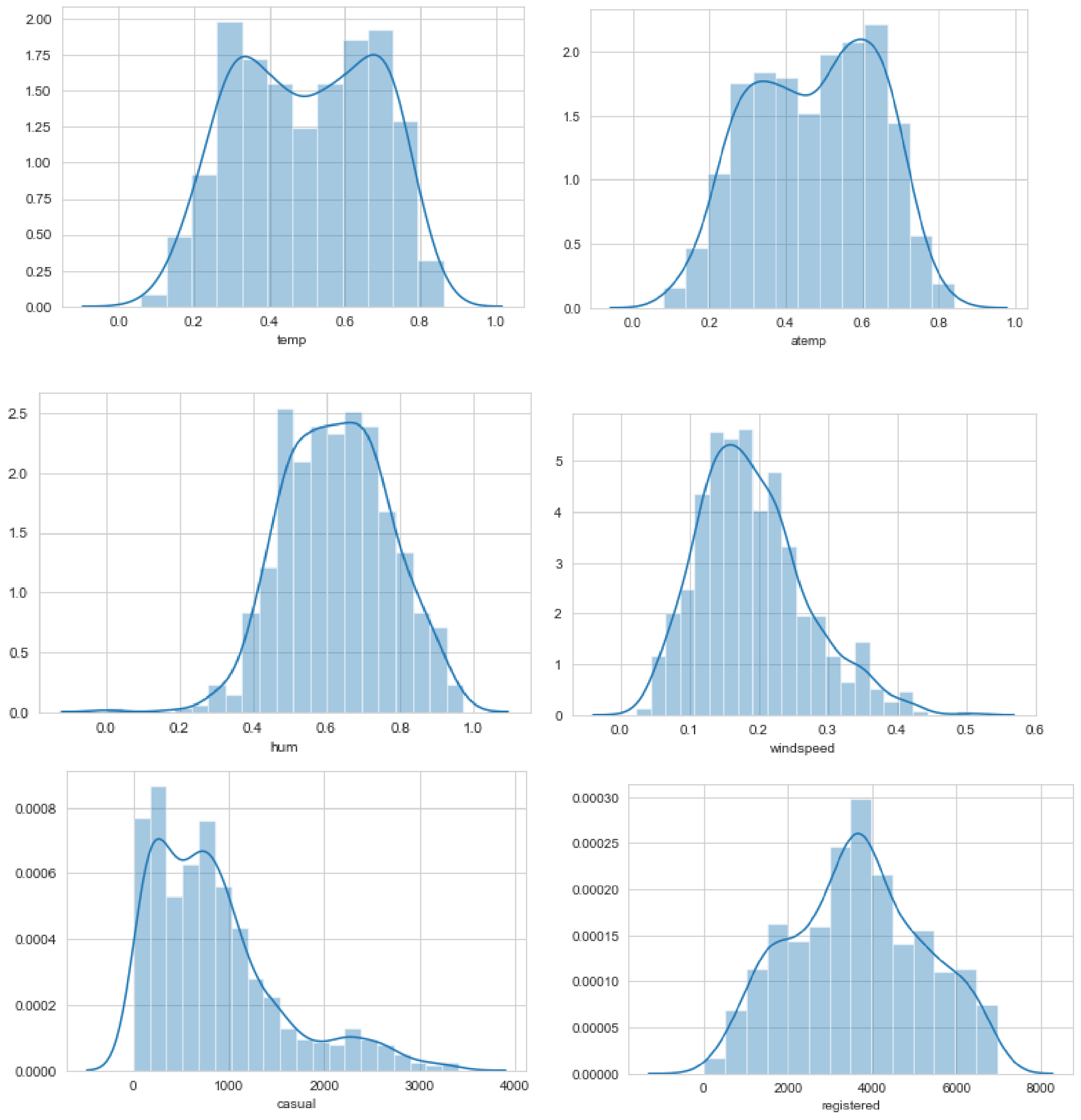
Figure 2.1 Distribution of target variable (CNT) (python code in Appendix B)

```
Skewness: -0.047353
Kurtosis: -0.811922
```



Kurtosis is very less so the curve is less tailed and no outlier is present in target variable

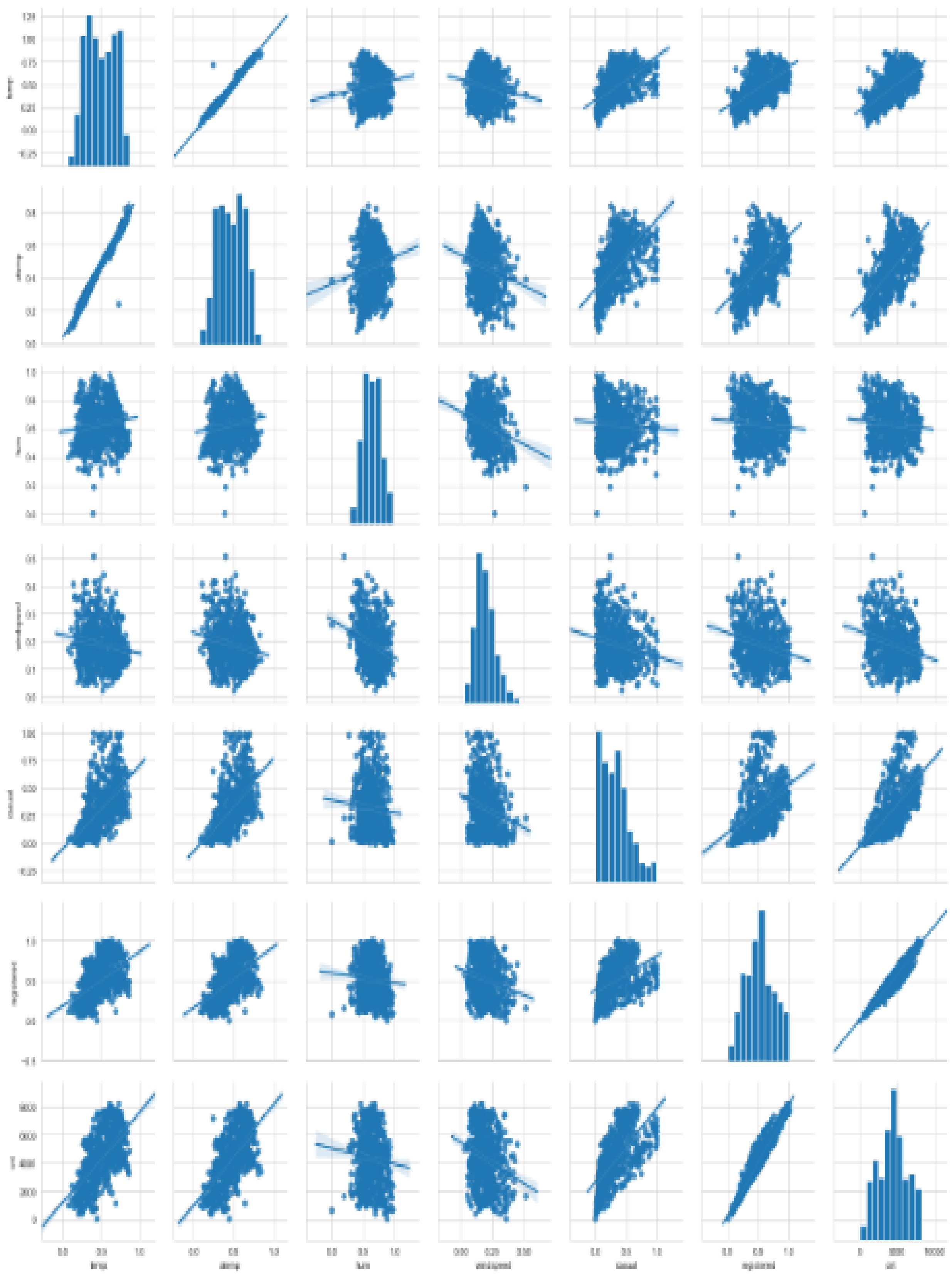
Figure 2.2 showing distribution of dependent variables (python code in Appendix B)



2.1.2 Bivariate Analysis

Ggpair function built upon ggplot2, GGally provides templates for combining plots into a matrix through the ggpairs function. Such a matrix of plots can be useful for quickly exploring the relationships between multiple columns of data in a data frame. The lower and upper arguments to the ggpairs function specifies the type of plot or data in each position of the lower or upper diagonal of the matrix, respectively.   For continuous X and Y data, one can specify the smooth option to include a regression line.

Below figures shows relationship between independent variables and also with numeric target variable using ggpair. Below ggpair graph is showing clearly that relationship between independent variables ‘temp’ and ‘atemp’ are very strong. The relationship between ‘hum’ , ‘windspeed’ with target variable ‘cnt’ is less.



temp
atemp
hum
windspeed
casual
registered
cnt

2.2 Pre Processing

2.2.1 Missing Value Analysis

Missing values in data is a common phenomenon in real world problems. Knowing how to handle missing values effectively is a required step to reduce bias and to produce powerful models.

Below table illustrate no missing value present in the data.

Fig 2.1 missing values

s.no	Variables	missing values
1	dteday	0
2	season	0
3	yr	0
4	mnth	0
5	holiday	0
6	weekday	0
7	workingday	0
8	weathersit	0
9	temp	0
10	atemp	0
11	hum	0
12	windspeed	0
13	casual	0
14	registered	0

2.2.2 Outlier Analysis

The Other steps of Preprocessing Technique is Outliers analysis, an outlier is an observation point that is distant from other observations. Outliers in data can distort predictions and affect the accuracy, if you don’t detect and handle them appropriately especially in regression models.

As we are observed in fig 2.2 the data is skewed so, there is chance of outlier in independent variable ‘casual’, one of the best method to detect outliers is Boxplot.

Fig 2.4 shows the presence of Outliers in variable ‘casual’

Fig 2.5 shows the boxplot of ‘casual’ after removing outliers

Box plot :- Boxplot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles

Figure 2.4 ‘casual’ Boxplot before removing outliers

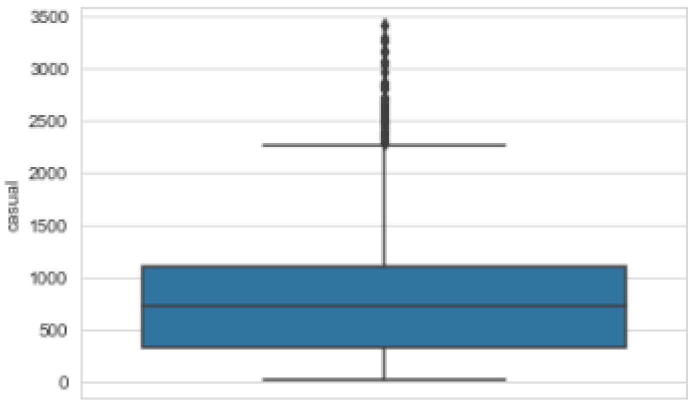
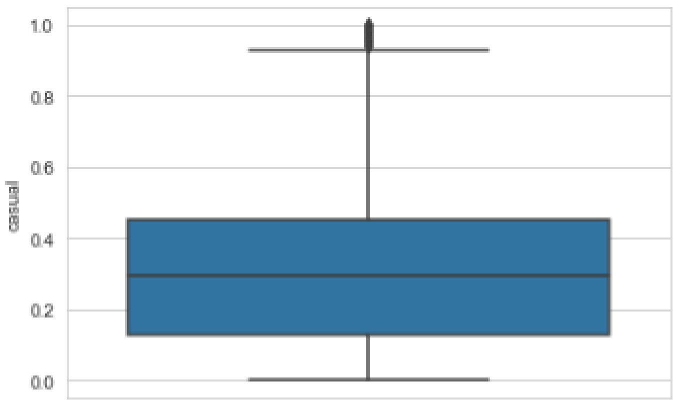


Figure 2.5 ‘casual’ Boxplot after removing outliers



Since there is significant difference between Pearson coefficient correlation between before and after outlier detection for ‘casual’ and ‘cnt’ and losing nearly 40 observation so, we are not going to treat the outliers.

2.2.3 Feature Selection

Machine learning works on a simple rule – if you put garbage in, you will only get garbage to come out. By garbage here, I mean noise in data.

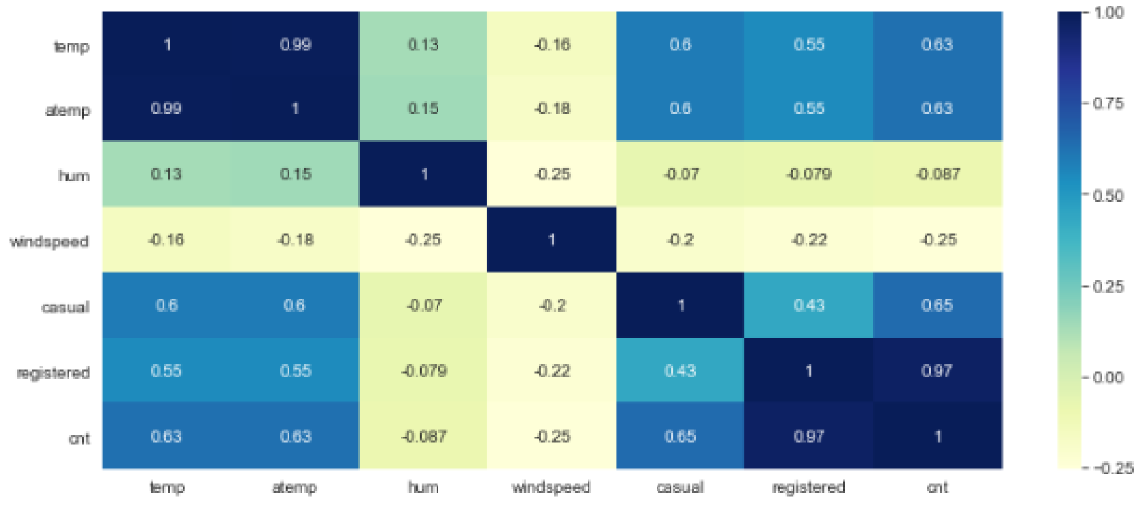
This becomes even more important when the number of features are very large. You need not use every feature at your disposal for creating an algorithm. You can assist your algorithm by feeding in only those features that are really important. I have myself witnessed feature subsets giving better results than complete set of feature for the same algorithm or – “Sometimes, less is better!”

We should consider the selection of feature for model based on below criteria

i. The relationship between two independent variable should be less and
ii. The relationship between Independent and Target variables should be high.

Below fig 2.6 illustrates that relationship between all numeric variables using Corrgram plot.

Figure 2.6 Correlation plot of numeric variables



Color dark blue indicates there is strong positive relationship and if darkness is decreasing indicates relation between variables are decreasing.

Color white indicates there is strong negative relationship and if darkness is decreasing indicates relationship between variables are decreasing.

Corrgram : it help us visualize the data in correlation matrices. correlograms are implimented through the **corrgram(x, order = , panel=, lower.panel=, upper.panel=, text.panel=, diag.panel=)**

2.2.3.1 Dimensionality Reduction for numeric variables

Above Fig 2.6 is showing

There is strong relationship between independent variables ‘temp’ and ‘atemp’ so considering any one feature enough to predict the better.

And it is also showing there is almost no relationship between independent variable ‘hum’ and dependent variable ‘cnt’. So, ‘hum’ variable is not so important to predict.

Subsetting two independent features ‘atemp’ and ‘hum’ from actual dataset.

2.2.4 Feature Scaling

The word “normalization” is used informally in statistics, and so the term normalized data can have multiple meanings. In most cases, when you normalize data you eliminate the units of measurement for data, enabling you to more easily compare data from different places. Some of the more common ways to normalize data include:

Transforming data using a z-score or t-score. This is usually called standardization. In the vast majority of cases, if a statistics textbook is talking about normalizing data, then this is the definition of “normalization” they are probably using.

Rescaling data to have values between 0 and 1. This is usually called feature scaling. One possible formula to achieve this is.

$$\boxed{x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}}$$

In rental dataset numeric variables like ‘temp’ , ‘atemp’ ,’hum’ and ‘ windspeed’ are in normalization form so , we have to Normalize two variables ‘casual’ and ‘registered’

After normalize ‘casual’ and ‘registered’ variables look like in table below where all values between 0 and 1.

Table Normalization of ‘casual’ and ‘registered

casual	registered
0.037852113	0.09384926
0.034624413	0.17455963
0.025234742	0.21628646
0.042840376	0.21628646
0.019366197	0.12575801

Chapter 3

Modelling

3.1 Model Selection

In out earlier stage of analysis, we have come to understand that few variables like ‘temp’,’casual, ‘registered ‘ are going to play key role in model development , for model development dependent variable may fall under below categories

i. Nominal
ii. Ordinal
iii. Interval
iv. Ratio

In our case dependent variable is interval so, the predictive analysis that we can perform is Regression Analysis

3.1.1 Evaluating Regression Model

The main concept of looking at what is called **residuals** or difference between our predictions f(x[I,]) and actual outcomes y[i].

We are using two methods to evaluating performance of model

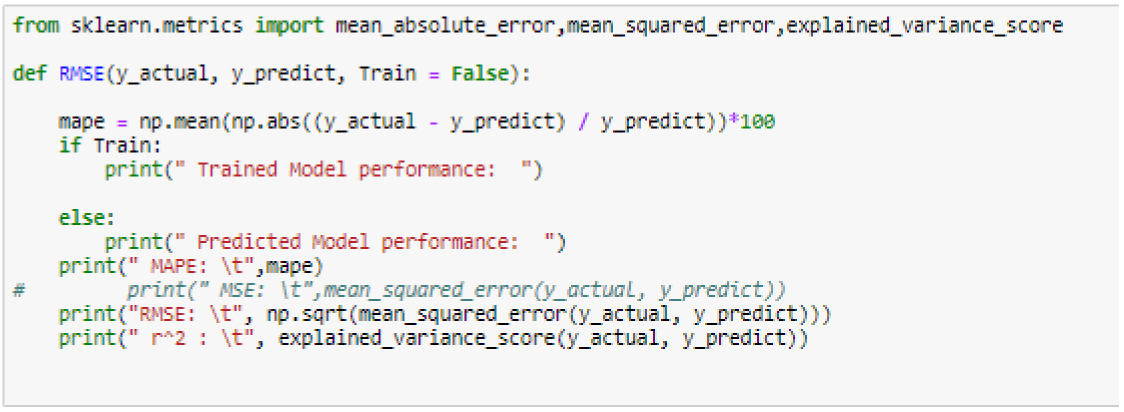
i. **MAPE** : (Mean Absolute Percent Error) measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.

$$\left(\frac{1}{n}\sum\frac{|Actual - Forecast|}{|Actual|}\right)*100$$

ii. **RMSE :** (Root Mean Square Error) is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}}$$

Figure 3.1.1 Evaluation using MAPE and RMSE in python



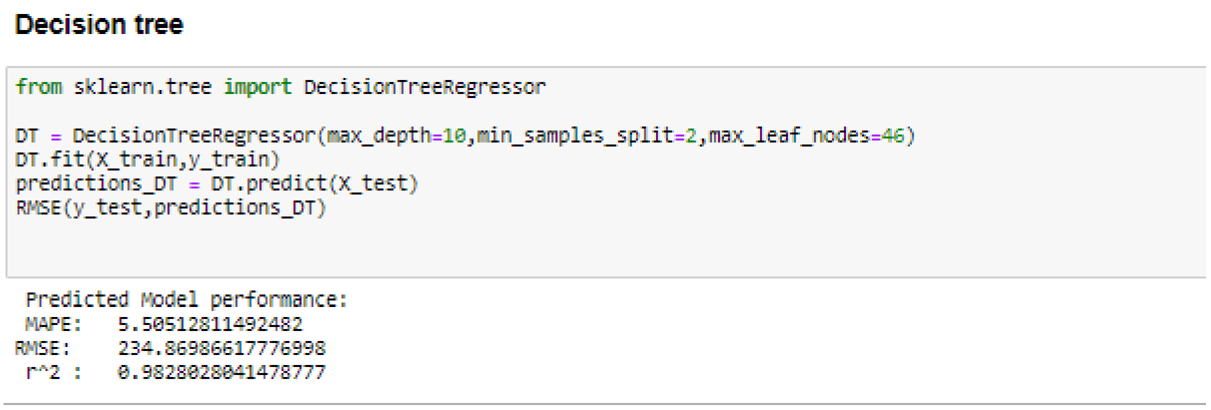
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,explained_variance_score

def RMSE(y_actual, y_predict, Train = False):

    mape = np.mean(np.abs((y_actual - y_predict) / y_predict))*100
    if Train:
        print(" Trained Model performance:  ")

    else:
        print(" Predicted Model performance:  ")
    print(" MAPE: \t",mape)
#       print(" MSE: \t",mean_squared_error(y_actual, y_predict))
    print("RMSE: \t", np.sqrt(mean_squared_error(y_actual, y_predict)))
    print(" r^2 : \t", explained_variance_score(y_actual, y_predict))
```

We will start our model building from Decision Tree.

3.2 Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Figure 3.2.1 Decision Tree Algorithm



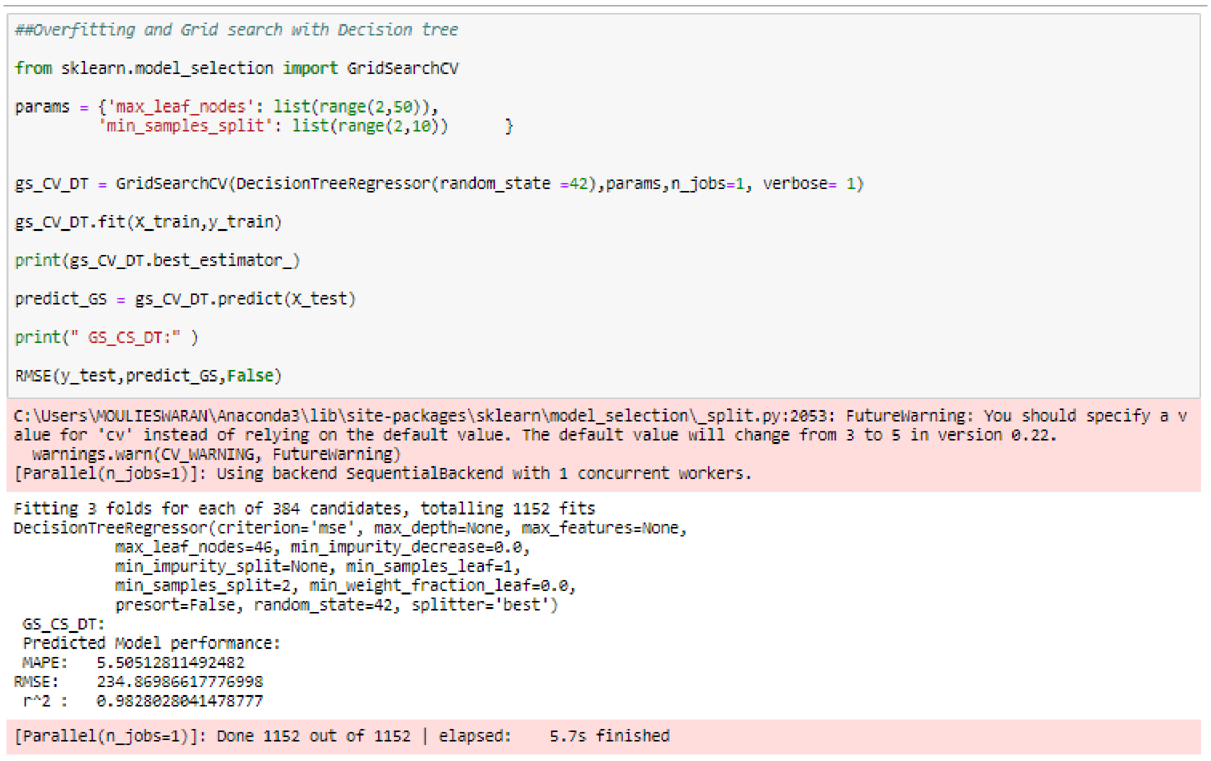
Decision tree

```
from sklearn.tree import DecisionTreeRegressor

DT = DecisionTreeRegressor(max_depth=10,min_samples_split=2,max_leaf_nodes=46)
DT.fit(X_train,y_train)
predictions_DT = DT.predict(X_test)
RMSE(y_test,predictions_DT)
```

```
 Predicted Model performance:
 MAPE:   5.50512811492482
RMSE:    234.86986617776998
 r^2 :   0.9828028041478777
```

Using GridSearchCV method, to improve the performance of Decision Tree.

Figure 3.2.2 Decision Tree Algorithm using GridSearchCV



```
##Overfitting and Grid search with Decision tree

from sklearn.model_selection import GridSearchCV

params = {'max_leaf_nodes': list(range(2,50)),
         'min_samples_split': list(range(2,10))      }


gs_CV_DT = GridSearchCV(DecisionTreeRegressor(random_state =42),params,n_jobs=1, verbose= 1)

gs_CV_DT.fit(X_train,y_train)

print(gs_CV_DT.best_estimator_)

predict_GS = gs_CV_DT.predict(X_test)

print(" GS_CS_DT:" )

RMSE(y_test,predict_GS,False)
```

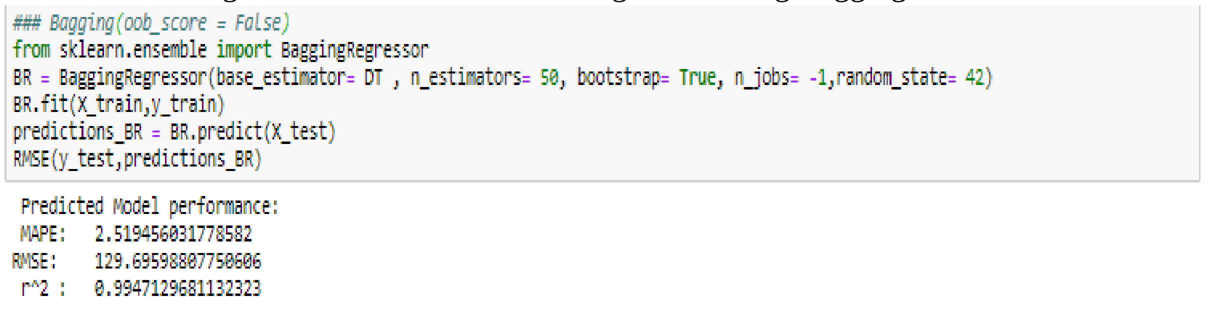
```
C:\Users\MOULIESWARAN\Anaconda3\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarning: You should specify a v
alue for 'cv' instead of relying on the default value. The default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
Fitting 3 folds for each of 384 candidates, totalling 1152 fits
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
           max_leaf_nodes=46, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           presort=False, random_state=42, splitter='best')
 GS_CS_DT:
 Predicted Model performance:
 MAPE:   5.50512811492482
RMSE:    234.86986617776998
 r^2 :   0.9828028041478777
```

```
[Parallel(n_jobs=1)]: Done 1152 out of 1152 | elapsed:    5.7s finished
```

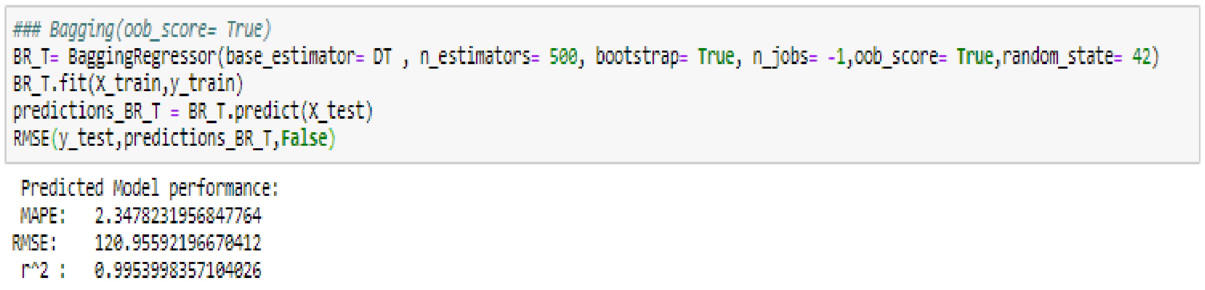
Using BaggingRegressor method to improve the performance of Decision Tree Model

Figure 3.2.3 Decision Tree Algorithm using Bagging method



```
### Bagging(oob_score = False)
from sklearn.ensemble import BaggingRegressor
BR = BaggingRegressor(base_estimator= DT , n_estimators= 50, bootstrap= True, n_jobs= -1,random_state= 42)
BR.fit(X_train,y_train)
predictions_BR = BR.predict(X_test)
RMSE(y_test,predictions_BR)
```

```
 Predicted Model performance:
 MAPE:   2.519456031778582
RMSE:    129.69598807750606
 r^2 :   0.9947129681132323
```



```
### Bagging(oob_score= True)
BR_T= BaggingRegressor(base_estimator= DT , n_estimators= 500, bootstrap= True, n_jobs= -1,oob_score= True,random_state= 42)
BR_T.fit(X_train,y_train)
predictions_BR_T = BR_T.predict(X_test)
RMSE(y_test,predictions_BR_T,False)
```

```
 Predicted Model performance:
 MAPE:   2.3478231956847764
RMSE:    120.95592196670412
 r^2 :   0.9953998357104026
```

In Figure 3.2.3 Model Accuracy is 100- 2.34 = 97.66 which is nearly 98% it is quite good but RMSE is 120 which is very high so it's clearly stating that our Decision Tree Model is Overfitted and it working well for training data but won't predict good for new set of data.

3.3 Random Forest

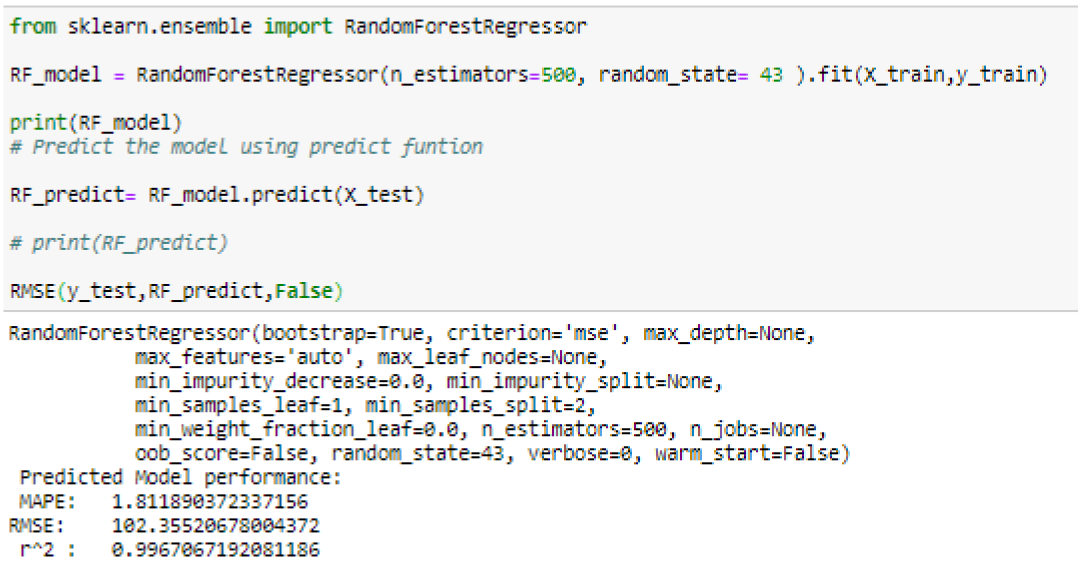
Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Random forest functions in below way

i. Draws a bootstrap sample from training data.
ii. For each sample grow a decision tree and at each node of the tree
    a. Ramdomly draws a subset of mtry variable and p total of features that are available
    b. Picks the best variable and best split from the subset of mtry variable
    c. Continues until the tree is fully grown.

As we saw in section 3.2 Decision tree is overfitting and its accuracy MAPE and RMSE is also poor in order to improve the performance of the model developing model using Random Forest.

Figure 3.3.1 Random Forest Implementation

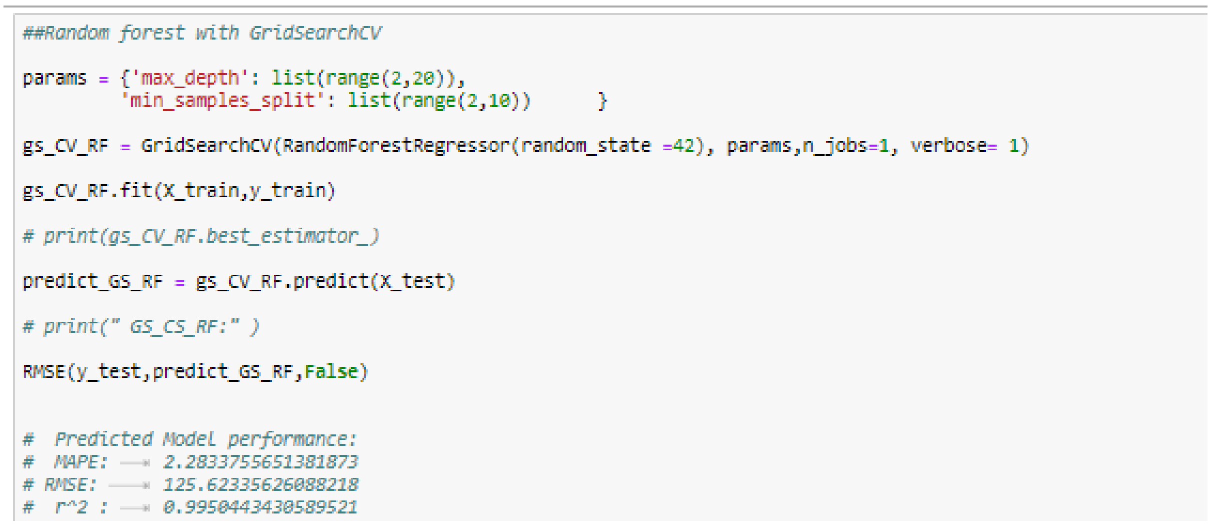


```
from sklearn.ensemble import RandomForestRegressor

RF_model = RandomForestRegressor(n_estimators=500, random_state= 43 ).fit(X_train,y_train)

print(RF_model)
# Predict the model using predict funtion

RF_predict= RF_model.predict(X_test)

# print(RF_predict)

RMSE(y_test,RF_predict,False)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=None,
           oob_score=False, random_state=43, verbose=0, warm_start=False)
 Predicted Model performance:
 MAPE:   1.811890372337156
RMSE:    102.35520678004372
 r^2 :   0.9967067192081186
```

Trying to improve the performance for Random Forest using GridSearchCV but the performance of Random forest is better without using GridSearchCV.

Figure 3.3.2 Random Forest with GridSearchCV Implementation

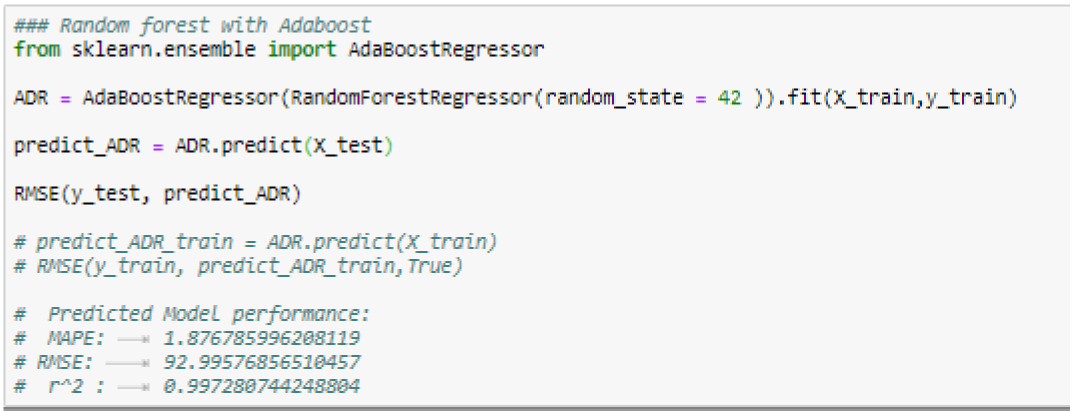


```
##Random forest with GridSearchCV

params = {'max_depth': list(range(2,20)),
         'min_samples_split': list(range(2,10))      }

gs_CV_RF = GridSearchCV(RandomForestRegressor(random_state =42), params,n_jobs=1, verbose= 1)

gs_CV_RF.fit(X_train,y_train)

# print(gs_CV_RF.best_estimator_)

predict_GS_RF = gs_CV_RF.predict(X_test)

# print(" GS_CS_RF:" )

RMSE(y_test,predict_GS_RF,False)


#  Predicted Model performance:
#  MAPE:    2.2833755651381873
# RMSE:     125.62335626088218
#  r^2 :    0.9950443430589521
```

Random Forest with AdaBoost

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Yoav Freund and Robert Schapire, who won the 2003 Gödel Prize for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

Figure 3.3.3 Random Forest with AdaBoost Implementation



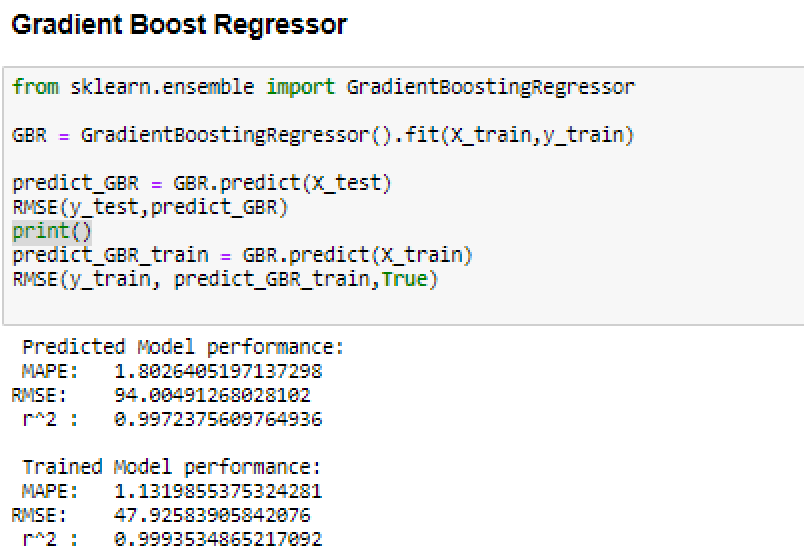
```
### Random forest with Adaboost
from sklearn.ensemble import AdaBoostRegressor

ADR = AdaBoostRegressor(RandomForestRegressor(random_state = 42 )).fit(X_train,y_train)

predict_ADR = ADR.predict(X_test)

RMSE(y_test, predict_ADR)

# predict_ADR_train = ADR.predict(X_train)
# RMSE(y_train, predict_ADR_train,True)

#  Predicted Model performance:
#  MAPE:  → 1.876785996208119
# RMSE:   → 92.99576856510457
#  r^2 :  → 0.997280744248804
```

The RMSE and MAPE for Random Forest is 1.81 and 102.35 respectively. But, the RMSE and MAPE for Random Forest with AdaBoost is 1.87 and 92.995 respectively. The Error is significantly reduced in Random Forest with AdaBoost.

3.4 Gradient Boosting Regressor

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function.

Figure 3.3.4 Random Forest with Gradient Boost Implementation



Gradient Boost Regressor

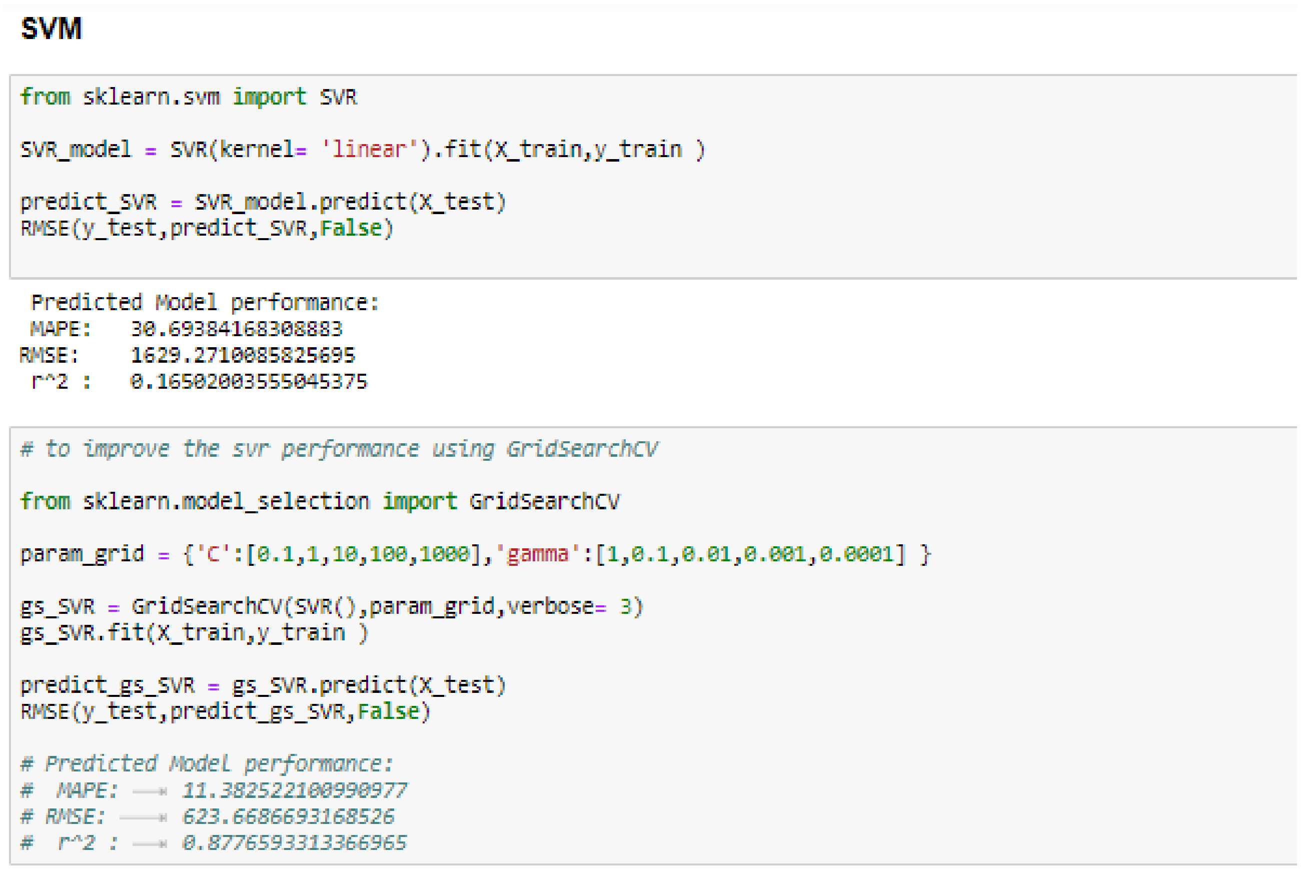
```
from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor().fit(X_train,y_train)

predict_GBR = GBR.predict(X_test)
RMSE(y_test,predict_GBR)
print()
predict_GBR_train = GBR.predict(X_train)
RMSE(y_train, predict_GBR_train,True)
```

```
 Predicted Model performance:
 MAPE:   1.8026405197137298
RMSE:    94.00491268028102
 r^2 :   0.9972375609764936

 Trained Model performance:
 MAPE:   1.1319855375324281
RMSE:    47.92583905842076
 r^2 :   0.9993534865217092
```

The accuracy of Gradient Boost is increased comparatively than Random forest with AdaBoost. for The RMSE is less than AdaBoost. But still the root mean square error is 94. It is high and not suitable for model implementation.

3.5 Support Vector Machine - Regression (SVR)

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.



SVM

```
from sklearn.svm import SVR

SVR_model = SVR(kernel= 'linear').fit(X_train,y_train )

predict_SVR = SVR_model.predict(X_test)
RMSE(y_test,predict_SVR,False)
```

```
 Predicted Model performance:
 MAPE:    30.69384168308883
RMSE:     1629.2710085825695
 r^2 :    0.16502003555045375
```

```
# to improve the svr performance using GridSearchCV

from sklearn.model_selection import GridSearchCV

param_grid = {'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001] }

gs_SVR = GridSearchCV(SVR(),param_grid,verbose= 3)
gs_SVR.fit(X_train,y_train )

predict_gs_SVR = gs_SVR.predict(X_test)
RMSE(y_test,predict_gs_SVR,False)

# Predicted Model performance:
#  MAPE:    11.382522100990977
# RMSE:     623.6686693168526
#  r^2 :    0.8776593313366965
```

Eventhough trying to increase the performance of SVR with GridSearchCV, still the RMSE value is quite high. Hence, SVR is not suitable for this model.

3.6 KNN

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

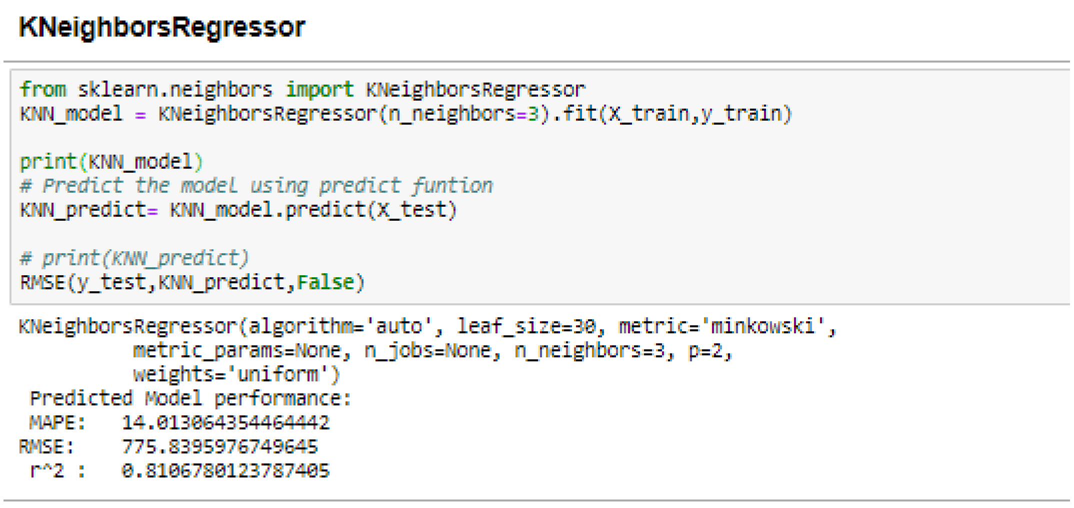
In k-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data.



KNeighborsRegressor

```python
from sklearn.neighbors import KNeighborsRegressor
KNN_model = KNeighborsRegressor(n_neighbors=3).fit(X_train,y_train)

print(KNN_model)
# Predict the model using predict funtion
KNN_predict= KNN_model.predict(X_test)

# print(KNN_predict)
RMSE(y_test,KNN_predict,False)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=None, n_neighbors=3, p=2,
          weights='uniform')
 Predicted Model performance:
 MAPE:   14.013064354464442
RMSE:    775.8395976749645
 r^2 :   0.8106780123787405
```

The RMSE for KNeighborsRegressor is 775. Hence, the KNeighborsRegressor is not best suited.

3.7 Linear Regression

Multiple linear regression attempts to model the relationship between two or more explanatory variables and a response variable by fitting a linear equation to observed data. Every value of the independent variable *x* is associated with a value of the dependent variable *y*. The population regression line for *p* explanatory variables $x_1, x_2, \ldots, x_p$ is defined to be $\mu_y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p$. This line describes how the mean response μ_y changes with the explanatory variables. The observed values for *y* vary about their means μ_y and are assumed to have the same standard deviation σ. The fitted values $b_0, b_1, \ldots, b_p$ estimate the parameters $\beta_0, \beta_1, \ldots, \beta_p$ of the population regression line.

Since the observed values for *y* vary about their means μ_y, the multiple regression model includes a term for this variation. In words, the model is expressed as DATA = FIT + RESIDUAL, where the "FIT" term represents the expression $\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_p x_p$. The "RESIDUAL" term represents the deviations of the observed values *y* from their means μ_y, which are normally distributed with mean 0 and variance σ. The notation for the model deviations is ε.

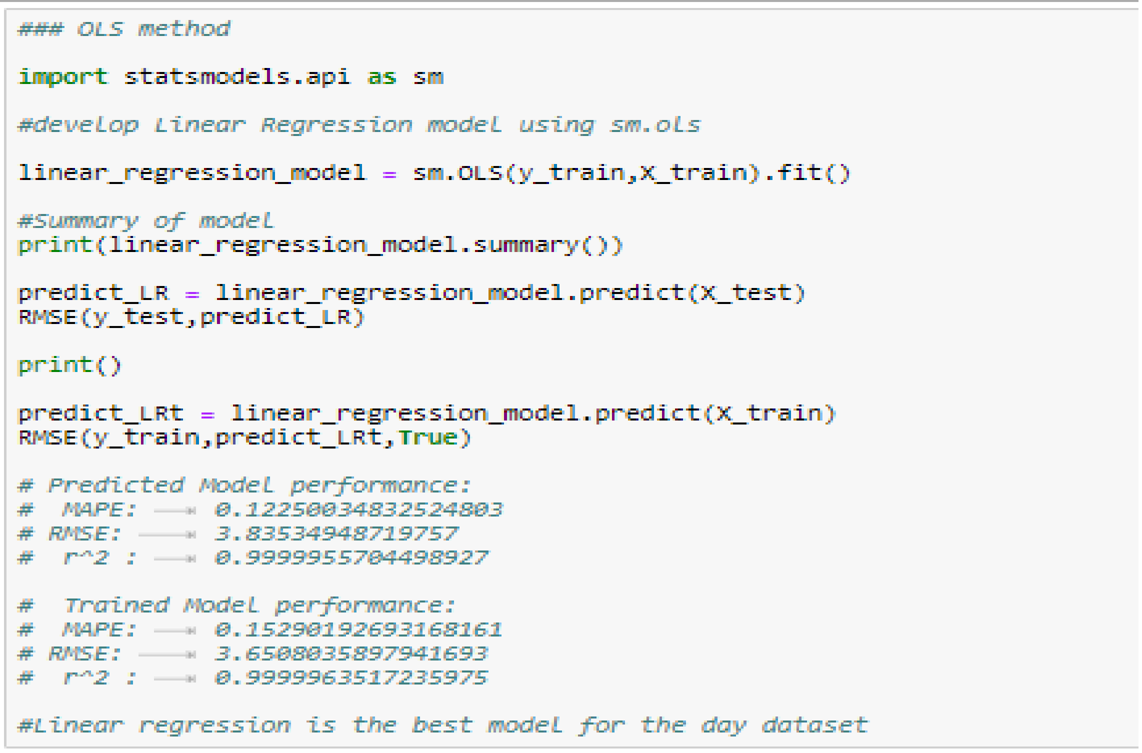
Formally, the model for multiple linear regression, given *n* observations, is $y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots \beta_p x_{ip} + \varepsilon_i$ for $i = 1,2, \ldots n$.

In the least-squares model, the best-fitting line for the observed data is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line (if a point lies on the fitted line exactly, then its vertical deviation is 0). Because the deviations are first squared, then summed, there are no cancellations between positive and negative values. The least-squares estimates $b_0, b_1, \ldots b_p$ are usually computed by statistical software.

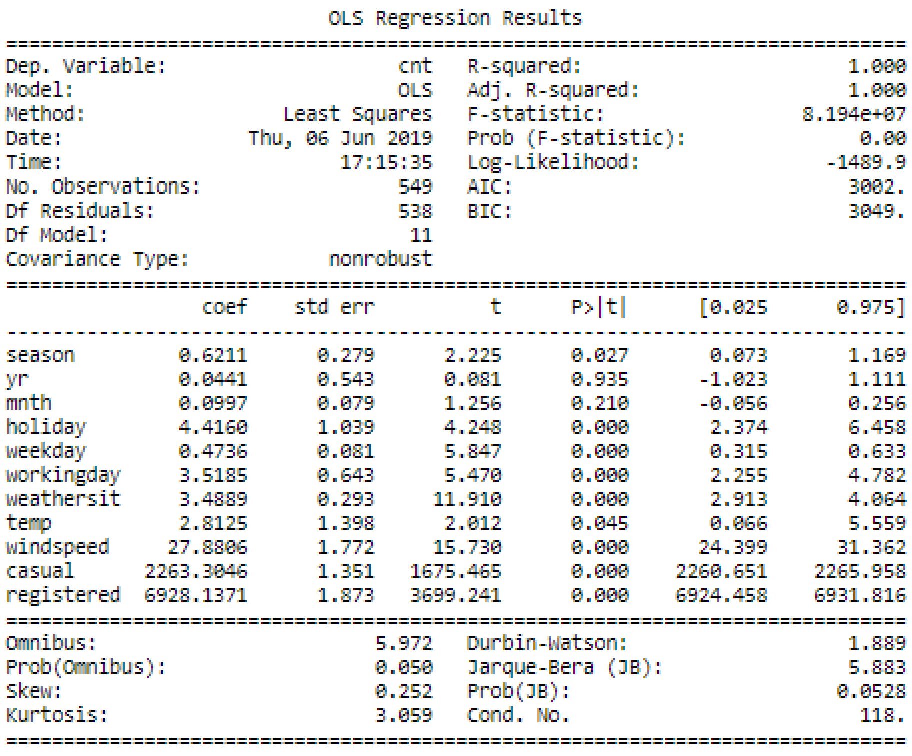
The values fit by the equation $b_0 + b_1 x_{i1} + \ldots + b_p x_{ip}$ are denoted $\hat{y}_i$, and the residuals e_i are equal to $y_i - \hat{y}_i$, the difference between the observed and fitted values. The sum of the residuals is equal to zero.

The variance σ^2 may be estimated by $s^2 = \dfrac{\sum e_i^2}{n - p - 1}$, also known as the mean-squared error (or MSE).
The estimate of the standard error *s* is the square root of the MSE.

Figure 3.6.1 Multiple Linear Regression Model



```
### OLS method

import statsmodels.api as sm

#develop Linear Regression model using sm.ols

linear_regression_model = sm.OLS(y_train,X_train).fit()

#Summary of model
print(linear_regression_model.summary())

predict_LR = linear_regression_model.predict(X_test)
RMSE(y_test,predict_LR)

print()

predict_LRt = linear_regression_model.predict(X_train)
RMSE(y_train,predict_LRt,True)

# Predicted Model performance:
#  MAPE:  →  0.1225003483252480 3
# RMSE:   →  3.835349487197 57
#  r^2 :  →  0.9999955704498927

#  Trained Model performance:
#  MAPE:  →  0.15290192693168161
# RMSE:   →  3.6508035897941693
#  r^2 :  →  0.9999963517235975

#Linear regression is the best model for the day dataset
```



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    cnt   R-squared:                       1.000
Model:                            OLS   Adj. R-squared:                  1.000
Method:                 Least Squares   F-statistic:                 8.194e+07
Date:                Thu, 06 Jun 2019   Prob (F-statistic):               0.00
Time:                        17:15:35   Log-Likelihood:                -1489.9
No. Observations:                 549   AIC:                             3002.
Df Residuals:                     538   BIC:                             3049.
Df Model:                          11
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
season         0.6211      0.279      2.225      0.027       0.073       1.169
yr             0.0441      0.543      0.081      0.935      -1.023       1.111
mnth           0.0997      0.079      1.256      0.210      -0.056       0.256
holiday        4.4160      1.039      4.248      0.000       2.374       6.458
weekday        0.4736      0.081      5.847      0.000       0.315       0.633
workingday     3.5185      0.643      5.470      0.000       2.255       4.782
weathersit     3.4889      0.293     11.910      0.000       2.913       4.064
temp           2.8125      1.398      2.012      0.045       0.066       5.559
windspeed     27.8806      1.772     15.730      0.000      24.399      31.362
casual      2263.3046      1.351   1675.465      0.000    2260.651    2265.958
registered  6928.1371      1.873   3699.241      0.000    6924.458    6931.816
==============================================================================
Omnibus:                        5.972   Durbin-Watson:                   1.889
Prob(Omnibus):                  0.050   Jarque-Bera (JB):                5.883
Skew:                           0.252   Prob(JB):                       0.0528
Kurtosis:                       3.059   Cond. No.                         118.
==============================================================================
```

Here :

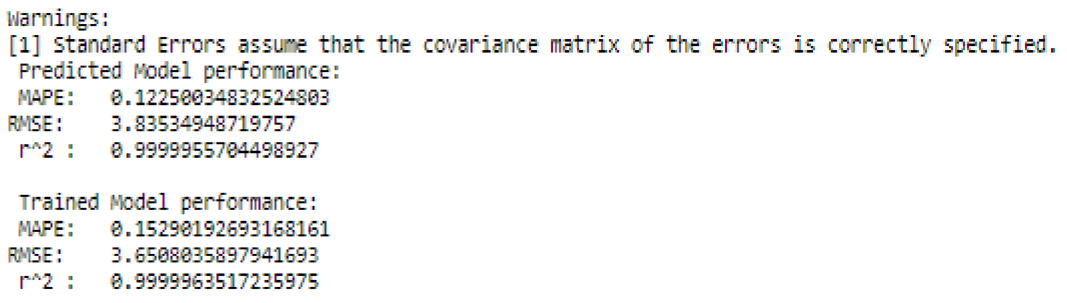
Multiple R-squared: 1, Adjusted R-squared: 1

Here residual Standard error is quite less so the distance between predicted values f(x[I,]) and actaual values f(x) are very less so this model is predicted almost accurate values.

And Multiple R-Square value is 1 so, we can explain about 100 % of the data using our multiple linear regression model. This is very impressive.

3.7.1 Evaluation of Linear regression Model

Figure 3.7.1 Evaluation of Regression Model



```
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 Predicted Model performance:
 MAPE:   0.12250034832524803
RMSE:    3.83534948719757
 r^2 :   0.9999955704498927

 Trained Model performance:
 MAPE:   0.15290192693168161
RMSE:    3.6508035897941693
 r^2 :   0.9999963517235975
```

From above figure, it is clearly showing that Model Accuracy is 99.88 % and RMSE is nearly equal to 3.83.

The predicted model accuracy is 99.88% and the trained model accuracy is 99.85 %. There is negligible difference in accuracy.

Model Selection

As we predicted counts for Bike Rental using three Models Decision Tree, Random Forest and Linear Regression as MAPE is high and RMSE is less for the Linear regression Model so conclusion is as below:

Conclusion: - For the Bike Rental Data, Linear Regression Model is the best model to predict the count.

Appendix A

Figure Relationship between Weekdays and cnt

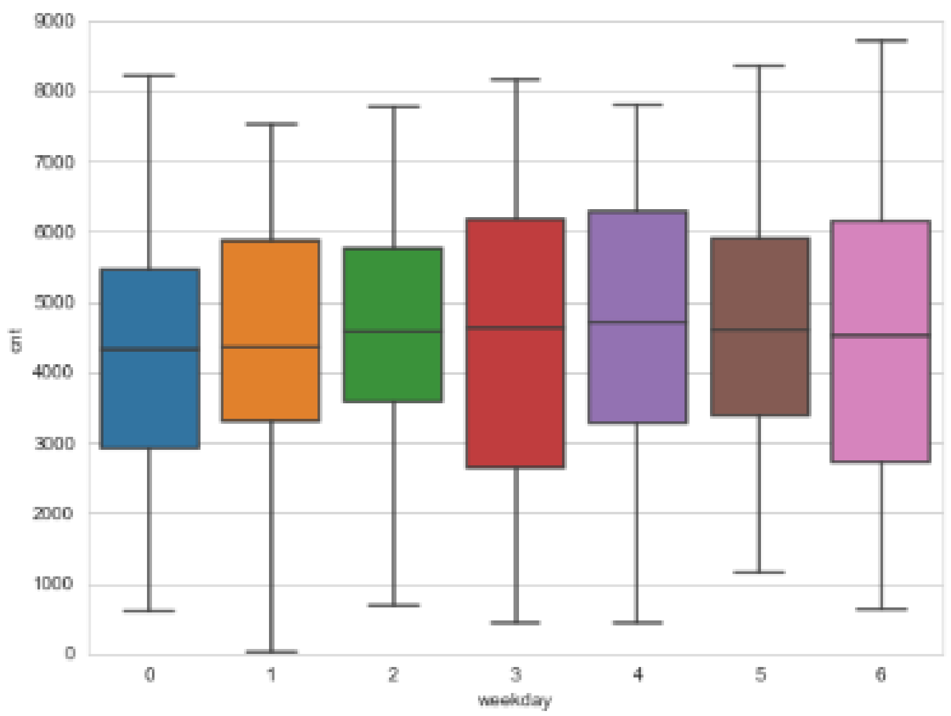


Figure Relationship between Holiday and cnt

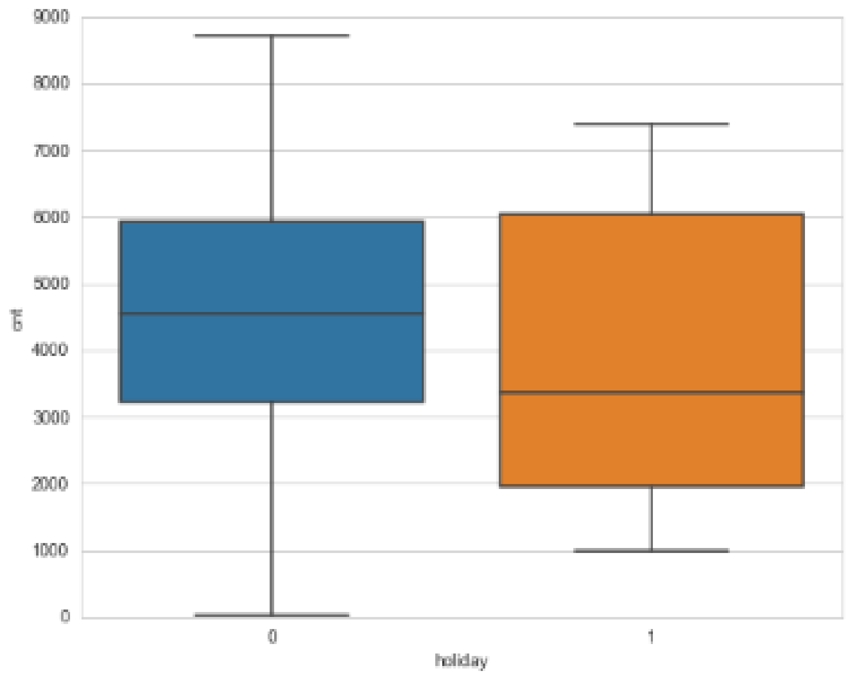
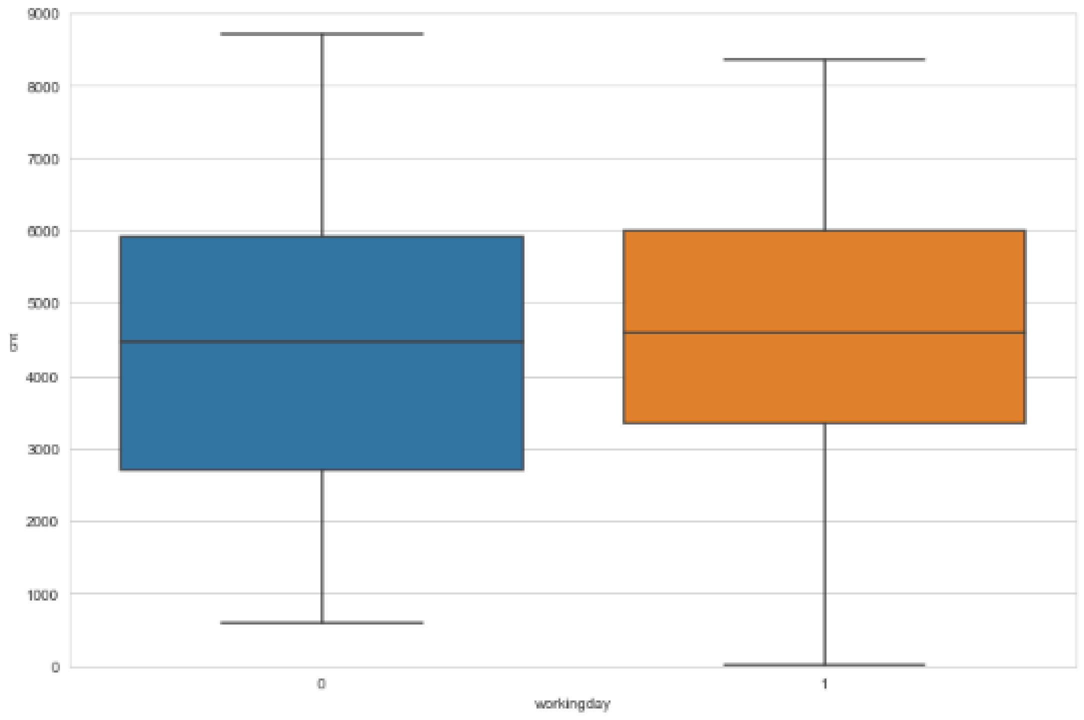


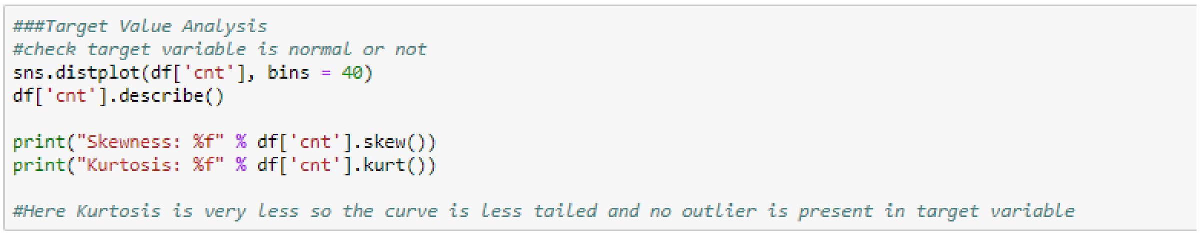
Figure Relationship between Workingday and cnt



9000
8000
7000
6000
5000
4000
3000
2000
1000
0
cnt
0
1
workingday

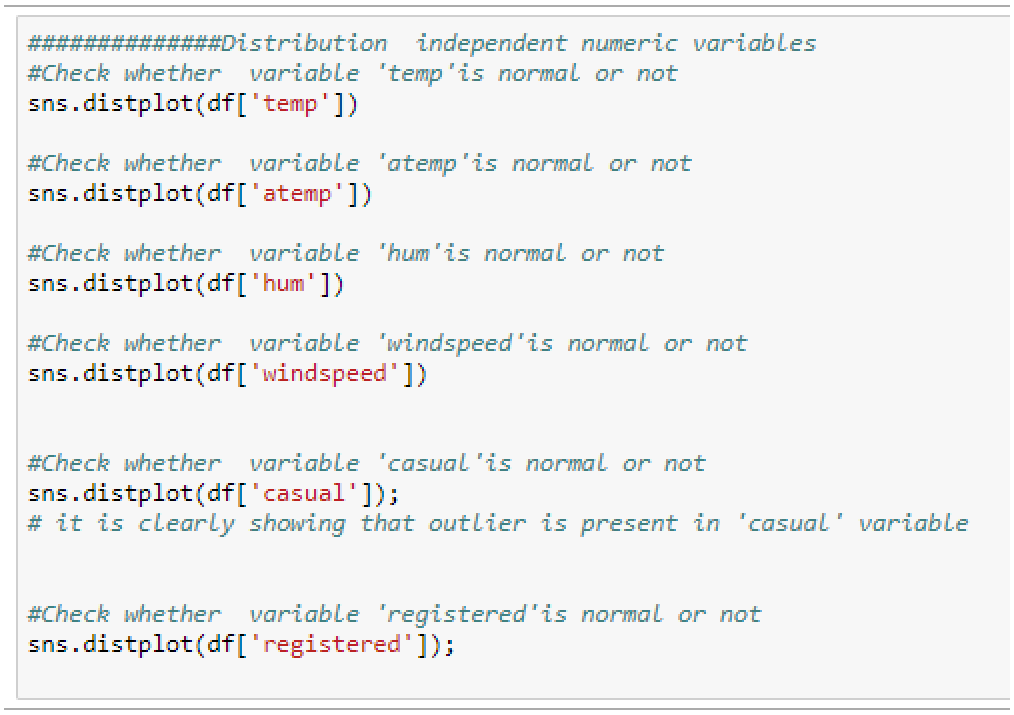
Appendix B - Python Code

Fig 2.1 Python Code



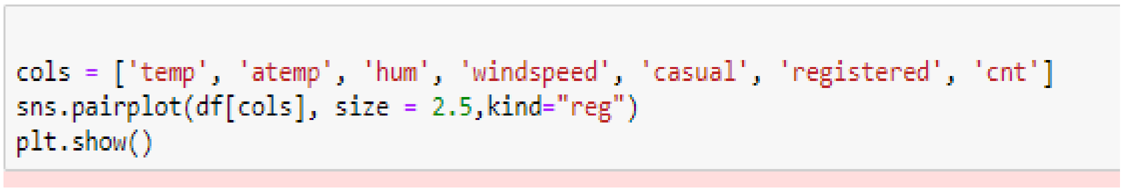
```
###Target Value Analysis
#check target variable is normal or not
sns.distplot(df['cnt'], bins = 40)
df['cnt'].describe()

print("Skewness: %f" % df['cnt'].skew())
print("Kurtosis: %f" % df['cnt'].kurt())

#Here Kurtosis is very less so the curve is less tailed and no outlier is present in target variable
```

Fig 2.2 Python Code



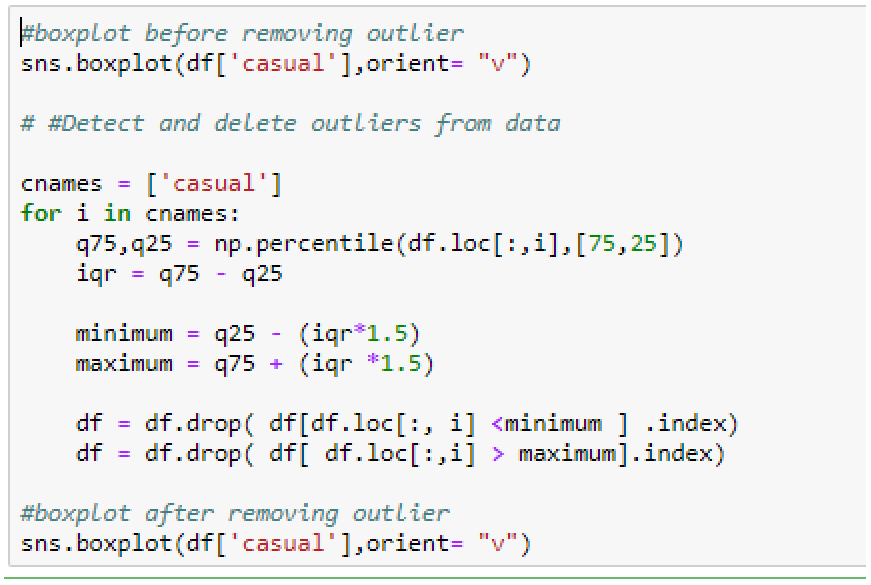
```
###############Distribution  independent numeric variables
#Check whether  variable 'temp'is normal or not
sns.distplot(df['temp'])

#Check whether  variable 'atemp'is normal or not
sns.distplot(df['atemp'])

#Check whether  variable 'hum'is normal or not
sns.distplot(df['hum'])

#Check whether  variable 'windspeed'is normal or not
sns.distplot(df['windspeed'])


#Check whether  variable 'casual'is normal or not
sns.distplot(df['casual']);
# it is clearly showing that outlier is present in 'casual' variable


#Check whether  variable 'registered'is normal or not
sns.distplot(df['registered']);
```

Fig 2.3 Python Code



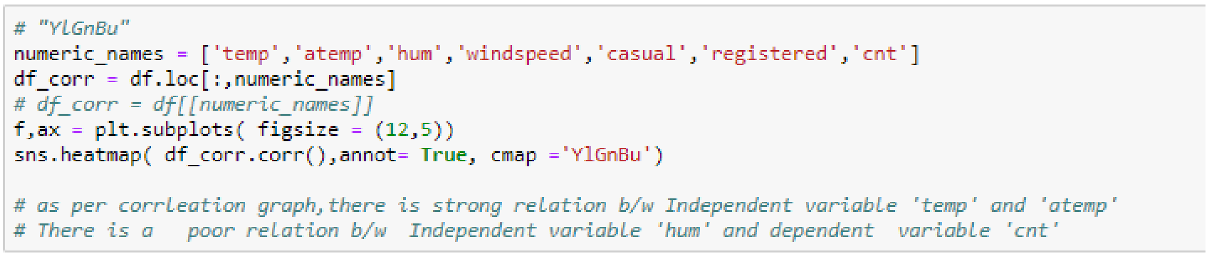
```
cols = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']
sns.pairplot(df[cols], size = 2.5,kind="reg")
plt.show()
```

Fig 2.4 and 2.5 Python Code



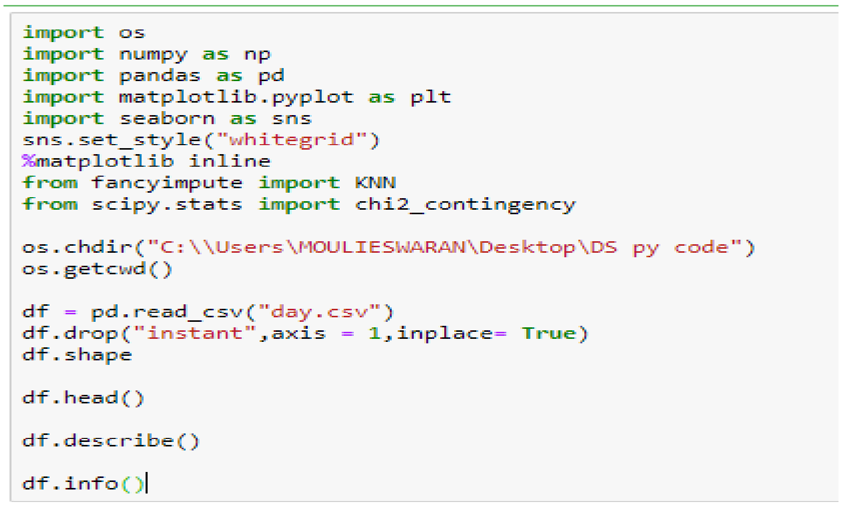
```
#boxplot before removing outlier
sns.boxplot(df['casual'],orient= "v")

# #Detect and delete outliers from data

cnames = ['casual']
for i in cnames:
    q75,q25 = np.percentile(df.loc[:,i],[75,25])
    iqr = q75 - q25

    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr *1.5)

    df = df.drop( df[df.loc[:, i] <minimum ] .index)
    df = df.drop( df[ df.loc[:,i] > maximum].index)

#boxplot after removing outlier
sns.boxplot(df['casual'],orient= "v")
```

Fig 2.6 Python Code



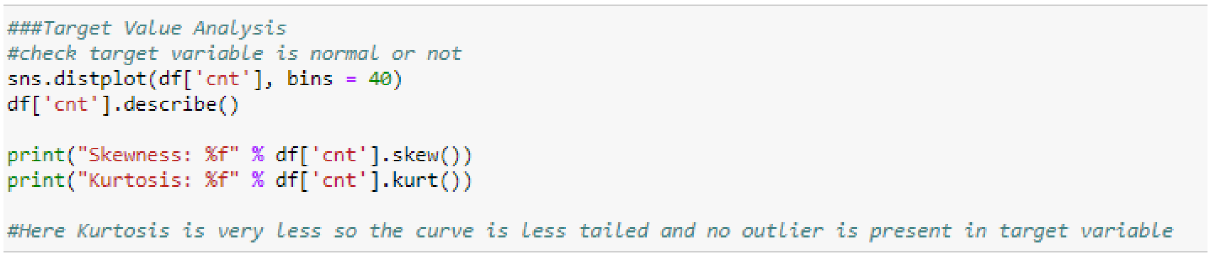
```
# "YlGnBu"
numeric_names = ['temp','atemp','hum','windspeed','casual','registered','cnt']
df_corr = df.loc[:,numeric_names]
# df_corr = df[[numeric_names]]
f,ax = plt.subplots( figsize = (12,5))
sns.heatmap( df_corr.corr(),annot= True, cmap ='YlGnBu')

# as per corrleation graph,there is strong relation b/w Independent variable 'temp' and 'atemp'
# There is a   poor relation b/w  Independent variable 'hum' and dependent  variable 'cnt'
```

Complete Python File

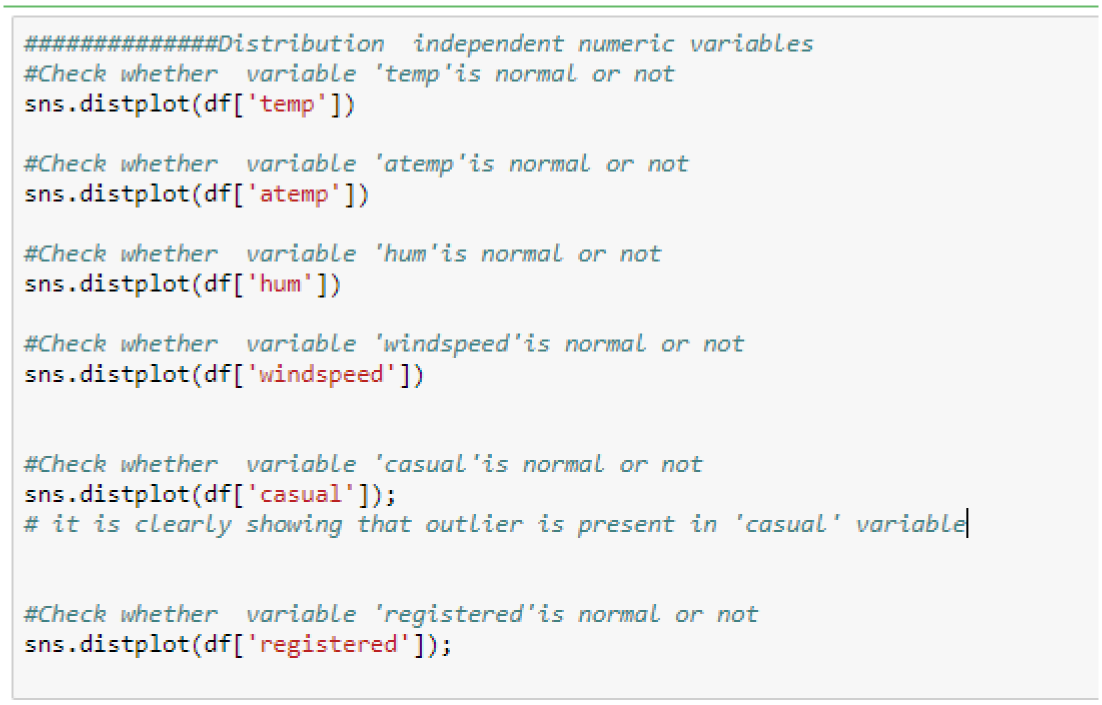


```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")
%matplotlib inline
from fancyimpute import KNN
from scipy.stats import chi2_contingency

os.chdir("C:\\Users\MOULIESWARAN\Desktop\DS py code")
os.getcwd()

df = pd.read_csv("day.csv")
df.drop("instant",axis = 1,inplace= True)
df.shape

df.head()

df.describe()

df.info()
```

Exploratory Data Analysis



```python
###Target Value Analysis
#check target variable is normal or not
sns.distplot(df['cnt'], bins = 40)
df['cnt'].describe()

print("Skewness: %f" % df['cnt'].skew())
print("Kurtosis: %f" % df['cnt'].kurt())

#Here Kurtosis is very less so the curve is less tailed and no outlier is present in target variable
```



```python
###############Distribution  independent numeric variables
#Check whether  variable 'temp'is normal or not
sns.distplot(df['temp'])

#Check whether  variable 'atemp'is normal or not
sns.distplot(df['atemp'])

#Check whether  variable 'hum'is normal or not
sns.distplot(df['hum'])

#Check whether  variable 'windspeed'is normal or not
sns.distplot(df['windspeed'])


#Check whether  variable 'casual'is normal or not
sns.distplot(df['casual']);
# it is clearly showing that outlier is present in 'casual' variable


#Check whether  variable 'registered'is normal or not
sns.distplot(df['registered']);
```

```
################ Bivariate  Relationship
#relation between Numerical Variable 'temp' and target variable 'cnt'
sns.jointplot(x = 'temp' , y='cnt', data= df)

#relation between Numerical Variable 'atemp' and target variable 'cnt'
sns.jointplot(x = 'atemp' , y='cnt', data= df)

#relation between Numerical Variable 'hue' and target variable 'cnt'
sns.jointplot(x ='hum', y='cnt', data= df,color ="g")

#relation between Numerical Variable 'windspeed' and target variable 'cnt'
sns.jointplot(x = 'windspeed' , y='cnt', data= df,color ="g")

#box plot 'weekday' with 'CNT'
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x= 'weekday', y="cnt", data=df)
fig.axis(ymin=0, ymax=9000);

#all the weekdays median lying between 4000- 5000

#box plot 'holiday' with 'CNT'
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x= 'holiday', y="cnt", data= df)
fig.axis(ymin=0, ymax=9000);

#median  high on  holidays when compare to weekdays

#box plot 'workingday' with 'CNT'
f, ax = plt.subplots(figsize = (12,8))
fig = sns.boxplot( x ='workingday' , y ='cnt',data = df)
fig.axis( ymin = 0, ymax = 9000)
#median around 4500
```

Missing Value Analysis

```
#Create dataframe with missing percentage
missing_val = pd.DataFrame(df.isnull().sum())
missing_val
#no value is missing
```

```
#boxplot before removing outlier
sns.boxplot(df['casual'],orient= "v")

# #Detect and delete outliers from data

cnames = ['casual']
for i in cnames:
    q75,q25 = np.percentile(df.loc[:,i],[75,25])
    iqr = q75 - q25

    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr *1.5)

    df = df.drop( df[df.loc[:, i] <minimum ] .index)
    df = df.drop( df[ df.loc[:,i] > maximum].index)

#boxplot after removing outlier
sns.boxplot(df['casual'],orient= "v")
```

Feature Selection

```
# "YlGnBu"
numeric_names = ['temp','atemp','hum','windspeed','casual','registered','cnt']
df_corr = df.loc[:,numeric_names]
# df_corr = df[[numeric_names]]
f,ax = plt.subplots( figsize = (12,5))
sns.heatmap( df_corr.corr(),annot= True, cmap ='YlGnBu')

# as per corrleation graph,there is strong relation b/w Independent variable 'temp' and 'atemp'
# There is a   poor relation b/w  Independent variable 'hum' and dependent  variable 'cnt'
```

```
cols = ['temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered', 'cnt']
sns.pairplot(df[cols], size = 2.5,kind="reg")
plt.show()
```

Feature Scaling

```
cnames = ['casual','registered']

for i in cnames:
    print(i)
    df[i] =( df[i] - min(df[i]) ) / (max(df[i]) - min( df[i]))
```

```
casual
registered
```

Model Development

```
#diividing  Test and train data  using skilearn   train_test_split

df_feature_selection = df.drop(['atemp','hum'],axis = 1)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train,y_test = train_test_split(df_feature_selection.iloc[:,1:-1],df_feature_selection['cnt'], test_size=0.2)
```

Evaluation Metrics

```
from sklearn.metrics import mean_absolute_error,mean_squared_error,explained_variance_score

def RMSE(y_actual, y_predict, Train = False):

    mape = np.mean(np.abs((y_actual - y_predict) / y_predict))*100
    if Train:
        print(" Trained Model performance:  ")

    else:
        print(" Predicted Model performance:  ")
    print(" MAPE: \t",mape)
#         print(" MSE: \t",mean_squared_error(y_actual, y_predict))
    print("RMSE: \t", np.sqrt(mean_squared_error(y_actual, y_predict)))
    print(" r^2 : \t", explained_variance_score(y_actual, y_predict))
```

Decision tree

```
from sklearn.tree import DecisionTreeRegressor

DT = DecisionTreeRegressor(max_depth=10,min_samples_split=2,max_leaf_nodes=46)
DT.fit(X_train,y_train)
predictions_DT = DT.predict(X_test)
RMSE(y_test,predictions_DT)
```

```
##Overfitting and Grid search with Decision tree

from sklearn.model_selection import GridSearchCV

params = {'max_leaf_nodes': list(range(2,50)),
          'min_samples_split': list(range(2,10))      }


gs_CV_DT = GridSearchCV(DecisionTreeRegressor(random_state =42),params,n_jobs=1, verbose= 1)

gs_CV_DT.fit(X_train,y_train)

print(gs_CV_DT.best_estimator_)

predict_GS = gs_CV_DT.predict(X_test)

print(" GS_CS_DT:" )

RMSE(y_test,predict_GS,False)
```

```
### Bagging(oob_score = False)
from sklearn.ensemble import BaggingRegressor
BR = BaggingRegressor(base_estimator= DT , n_estimators= 50, bootstrap= True, n_jobs= -1,random_state= 42)
BR.fit(X_train,y_train)
predictions_BR = BR.predict(X_test)
RMSE(y_test,predictions_BR)
```

```
### Bagging(oob_score= True)
BR_T= BaggingRegressor(base_estimator= DT , n_estimators= 500, bootstrap= True, n_jobs= -1,oob_score= True,random_state= 42)
BR_T.fit(X_train,y_train)
predictions_BR_T = BR_T.predict(X_test)
RMSE(y_test,predictions_BR_T,False)
```

```
# Create dot file to visualise tree  #http://webgraphviz.com/
from sklearn.tree import export_graphviz
dotfile = open("pt1.dot", 'w')
df = export_graphviz(DT, out_file=dotfile)
```

Random Forest

```python
from sklearn.ensemble import RandomForestRegressor

RF_model = RandomForestRegressor(n_estimators=500, random_state= 43 ).fit(X_train,y_train)

print(RF_model)
# Predict the model using predict funtion

RF_predict= RF_model.predict(X_test)

# print(RF_predict)

RMSE(y_test,RF_predict,False)
```

```python
##Random forest with GridSearchCV

params = {'max_depth': list(range(2,10)),
         'min_samples_split': list(range(2,20))      }

gs_CV_RF = GridSearchCV(RandomForestRegressor(random_state =42), params,n_jobs=1, verbose= 1)

gs_CV_RF.fit(X_train,y_train)

# print(gs_CV_RF.best_estimator_)

predict_GS_RF = gs_CV_RF.predict(X_test)

# print(" GS_CS_RF:" )

RMSE(y_test,predict_GS_RF,False)

# Predicted Model performance:
#  MAPE:	2.2474260956323886
# RMSE:	122.59138072490313
#  r^2 :	0.9952880921987182
```

```python
### Random forest with Adaboost
from sklearn.ensemble import AdaBoostRegressor

ADR = AdaBoostRegressor(RandomForestRegressor(random_state = 42 )).fit(X_train,y_train)

predict_ADR = ADR.predict(X_test)

RMSE(y_test, predict_ADR)

# predict_ADR_train = ADR.predict(X_train)
# RMSE(y_train, predict_ADR_train,True)

#  Predicted Model performance:
#  MAPE:	1.876785996208119
# RMSE:	92.99576856510457
#  r^2 :	0.997280744248804
```

Gradient Boost Regressor

```python
from sklearn.ensemble import GradientBoostingRegressor

GBR = GradientBoostingRegressor().fit(X_train,y_train)

predict_GBR = GBR.predict(X_test)
RMSE(y_test,predict_GBR)
print()
predict_GBR_train = GBR.predict(X_train)
RMSE(y_train, predict_GBR_train,True)
```

Extra Tree Regressor

```python
from sklearn.ensemble import ExtraTreesRegressor

ETR = ExtraTreesRegressor(max_leaf_nodes= 49, min_samples_split= 2, random_state= 42).fit(X_train,y_train)
ETR_predict = ETR.predict(X_test)

RMSE(y_test, ETR_predict)
```

SVM

```python
from sklearn.svm import SVR

SVR_model = SVR(kernel= 'linear').fit(X_train,y_train )

predict_SVR = SVR_model.predict(X_test)
RMSE(y_test,predict_SVR,False)
```

```python
# to improve the svr performance using GridSearchCV

from sklearn.model_selection import GridSearchCV

param_grid = {'C':[0.1,1,10,100,1000],'gamma':[1,0.1,0.01,0.001,0.0001] }

gs_SVR = GridSearchCV(SVR(),param_grid,verbose= 3)
gs_SVR.fit(X_train,y_train )

predict_gs_SVR = gs_SVR.predict(X_test)
RMSE(y_test,predict_gs_SVR,False)

# Predicted Model performance:
#  MAPE: →  11.382522100990977
# RMSE: →  623.6686693168526
#  r^2 : →  0.8776593313366965
```

KNeighborsRegressor

```python
from sklearn.neighbors import KNeighborsRegressor
KNN_model = KNeighborsRegressor(n_neighbors=3).fit(X_train,y_train)

print(KNN_model)
# Predict the model using predict funtion
KNN_predict= KNN_model.predict(X_test)

# print(KNN_predict)
RMSE(y_test,KNN_predict,False)
```

Linear Regression¶

```python
### Linear model
from sklearn.linear_model import LinearRegression

LR = LinearRegression().fit(X_train,y_train )

predict_LR_1 = LR.predict(X_test)
predict_LR_2 = LR.predict(X_train)

RMSE(y_test,predict_LR_1,False)
print()
RMSE(y_train,predict_LR_2,True)
```

```python
### OLS method

import statsmodels.api as sm

#develop Linear Regression model using sm.ols

linear_regression_model = sm.OLS(y_train,X_train).fit()

#Summary of model
print(linear_regression_model.summary())

predict_LR = linear_regression_model.predict(X_test)
RMSE(y_test,predict_LR)

print()

predict_LRt = linear_regression_model.predict(X_train)
RMSE(y_train,predict_LRt,True)

# Predicted Model performance:
#  MAPE:     0.12250034832524803
# RMSE:      3.83534948719757
#  r^2 :     0.9999955704498927

#  Trained Model performance:
#  MAPE:     0.15290192693168161
# RMSE:      3.6508035897941693
#  r^2 :     0.9999963517235975

#Linear regression is the best model for the day dataset
```

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An Introduction to Statistical Learning*. Vol. 6. Springer.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer Science & Business Media.