Project Report

On

Cab Fare Prediction

BY: Esha Rudra

Chapter 1- Introduction
1.1 Problem Statement
1.2 Data

Chapter 2- Methodology
• Pre-Processing
• Modelling
• Model Selection

Chapter 3- Pre-Processing
3.1 Data exploration and Cleaning (Missing Values and Outliers)
3.2 Creating some new variables from the given variables
3.3 Selection of variables
3.4 Some more data exploration
- Dependent and Independent Variables
- Uniqueness of Variables
- Dividing the variables categories

3.5 Feature Scaling

Chapter 4- Modelling
4.1 Linear Regression
4.2 Decision Tree
4.3 Random Forest
4.4 Gradient Boosting
4.5 Hyper Parameters Tunings for optimizing the results

Chapter 5- Conclusion
5.1 Model Evaluation
5.2 Model Selection
5.3 Some Visualization facts
References

Chapter 1

Introduction

Now a day’s cab rental services are expanding with the multiplier rate. The ease of using the services and flexibility gives their customer a great experience with competitive prices like OLA, UBER etc.

1.1 Problem Statement

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot project and now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

1.2 Data

Understanding of data is the very first and most crucial step in the process of finding solution to any business problem. Here in our case our company has provided a data set with following features, we need to go through each and every variable of it to understand and for better functioning. We need to understand the business insights in order to predict fares.

Size of Dataset Provided: - 16067 rows, 7 Columns (including dependent variable i.e Fare Amount)

Missing Values: **Yes**

Outliers Present: **Yes**

Below mentioned is a list of all the variable names with their meanings:

Variables	Description
fare_amount	**Fare amount charged for cab ride**
pickup_datetime	Cab pickup date and time
pickup_longitude	Pickup location longitude
pickup_latitude	Pickup location latitude
dropoff_longitude	Drop location longitude
dropoff_latitude	Drop location latitude
passenger_count	Number of passengers travelling in the cab

Chapter 2 Methodology

➢ Pre-Processing

When we are required to build a predictive model, we need to understand and manipulate the data before we start modelling which includes multiple pre-processing steps such as exploring the data, cleaning the data as well as visualizing the data through graphs and plots, all these steps are combined under one head which is **Exploratory Data Analysis**, which includes following steps:

- Data exploration and Cleaning
- Missing values analysis
- Outlier Analysis
- Feature Selection
- Features Scaling
- Skewness and Log transformation
- Visualization

➢ Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement(regression problem) and dataset, we will try some models on our pre-processed data and post comparing the output results we will select the best suitable model for our problem. As per our data set following models need to be tested:

- Linear regression
- Decision Tree
- Random forest
- Gradient Boosting

❖ We have also used hyper parameter tunings to check the parameters on which our model runs best. Following are two techniques of hyper parameter tuning we have used:

  - **Random Search CV**

- **Grid Search CV**

➢ Model Selection

The final step of our methodology will be the selection of the model based on the different output and results shown by different models. We have multiple parameters which we will study further in our report to test whether the model is suitable for our problem statement or not.

Chapter 3

Pre-Processing

3.1 Data exploration and Cleaning (Missing Values and Outliers)

The very first step which comes with any data science project is data exploration and cleaning which includes following points as per this project:

a. Separate the combined variables(Pickup_date_time)

b. As we know we have some negative values in fare amount so we have to remove those values, since fare can't be negative.

c. Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passenger count more than 6 and less than 1.

d. There are some outlier figures in the fare (like 54,343, 4343) so we need to remove those.

e. Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges.

3.2 Creating some new variables from the given variables.

Here in our data set our variable name pickup_datetime which contains date and time for pickup. So we tried to extract some important variables from pickup_datetime:

• Year

• Month

• Date

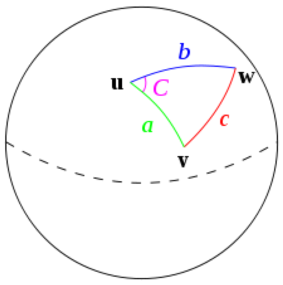
• Day of Week

• Hour

• Minute

Also, we tried to find out the distance using the haversine formula which says:

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles.



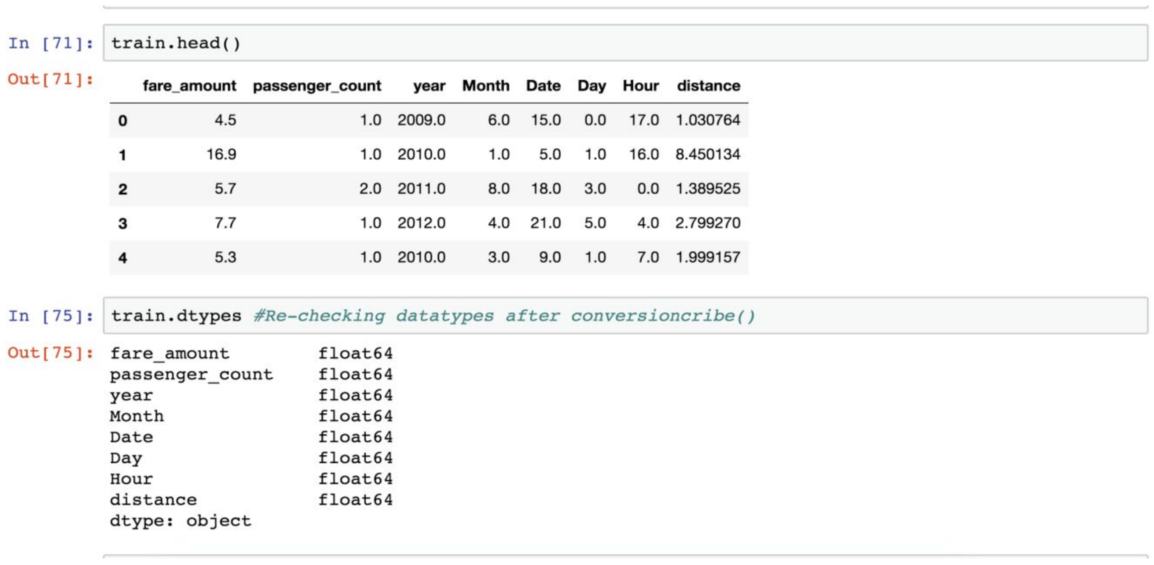
So our new extracted variables are:

- fare_amount
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count
- year
- Month
- Date
- Day of Week
- Hour
- Minute
- Distance

3.3 Selection of variables Now as we know that all above variables are of now use so we will drop the redundant variables since we already extracted relevant information from these variables:

- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- Minute

Now only following variables we will use for further steps:



```
In [71]: train.head()
```

Out[71]:

	fare_amount	passenger_count	year	Month	Date	Day	Hour	distance
0	4.5	1.0	2009.0	6.0	15.0	0.0	17.0	1.030764
1	16.9	1.0	2010.0	1.0	5.0	1.0	16.0	8.450134
2	5.7	2.0	2011.0	8.0	18.0	3.0	0.0	1.389525
3	7.7	1.0	2012.0	4.0	21.0	5.0	4.0	2.799270
4	5.3	1.0	2010.0	3.0	9.0	1.0	7.0	1.999157

```
In [75]: train.dtypes #Re-checking datatypes after conversioncribe()
```

```
Out[75]: fare_amount        float64
         passenger_count    float64
         year               float64
         Month              float64
         Date               float64
         Day                float64
         Hour               float64
         distance           float64
         dtype: object
```

3.4 Some more data exploration

In this report we are trying to predict the fare prices of a cab rental company. So here we have a data set of 16067 observations with 8 variables including one dependent variable (i.e fare amount).

3.4.1 Below are the names of Independent variables:

passenger_count,

year

Month

Date

Day of Week

Hour

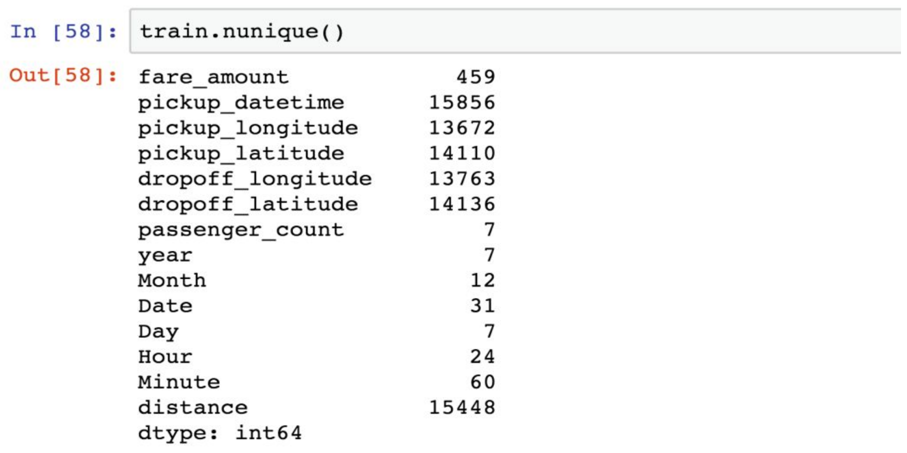
distance

Our Dependent variable is:

fare_amount

3.4.2 Uniqueness in Variable

We need to look at the unique values in the variables which will help us to decide whether the variable is categorical or numeric. So, by using python script 'nunique' we are going to find out the unique values in each variable. We have also added the table below:



```
In [58]: train.nunique()

Out[58]: fare_amount            459
         pickup_datetime      15856
         pickup_longitude     13672
         pickup_latitude      14110
         dropoff_longitude    13763
         dropoff_latitude     14136
         passenger_count          7
         year                     7
         Month                   12
         Date                    31
         Day                      7
         Hour                    24
         Minute                  60
         distance             15448
         dtype: int64
```

3.4.3 Dividing the variables into two categories basis their data types:

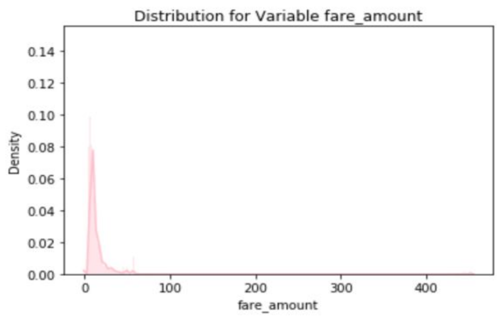
Continuous variables - 'fare_amount', 'distance'.

Categorical Variables - 'year', 'Month', 'Date', 'Day of Week', 'Hour', 'passenger_count'

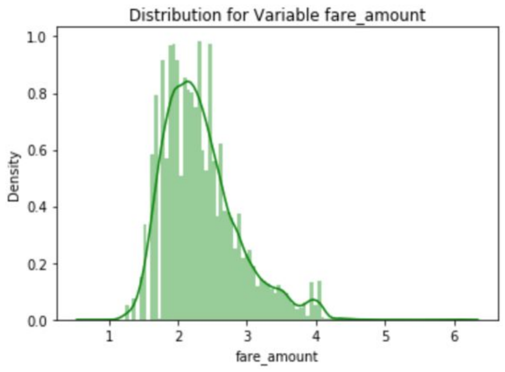
3.5 Feature Scaling

Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we found that our target variable absenteeism in hours is having one sided skewed so by using **log transform technique** we tried to reduce the skewness of the same. Below mentioned graphs shows the probability distribution plot to check distribution before log transformation:

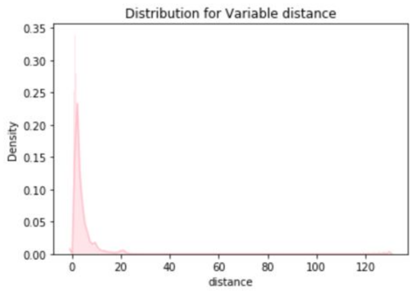
Before Log transformation



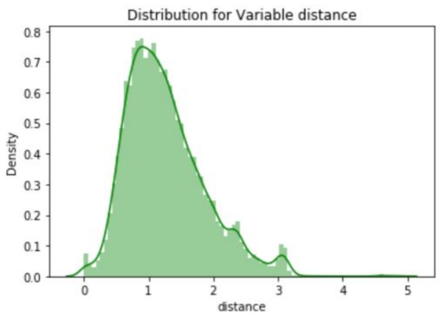
After Log transformation



Before Log transformation



After Log transformation



As our continuous variables appears to be normally distributed so we don’t need to use feature scaling techniques like normalization and standardization for the same.

Chapter 4 Modelling

After a thorough pre-processing, finally our data is ready. Now we will use some regression models on our processed data to predict the target variable. Following are the models which we have built –

1)Linear Regression

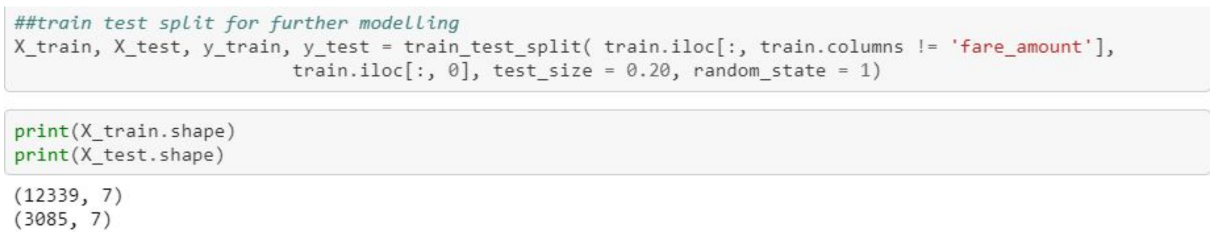
2)Decision Tree

3)Random Forest

4)Gradient Boosting

Before running any model, we will split our data into two parts which is train and test data. Here in our case we have taken 80% of the data as our train data. Below is image of the split of train test.

Now we will split our train data in two parts so that we can check our prediction accuracy.



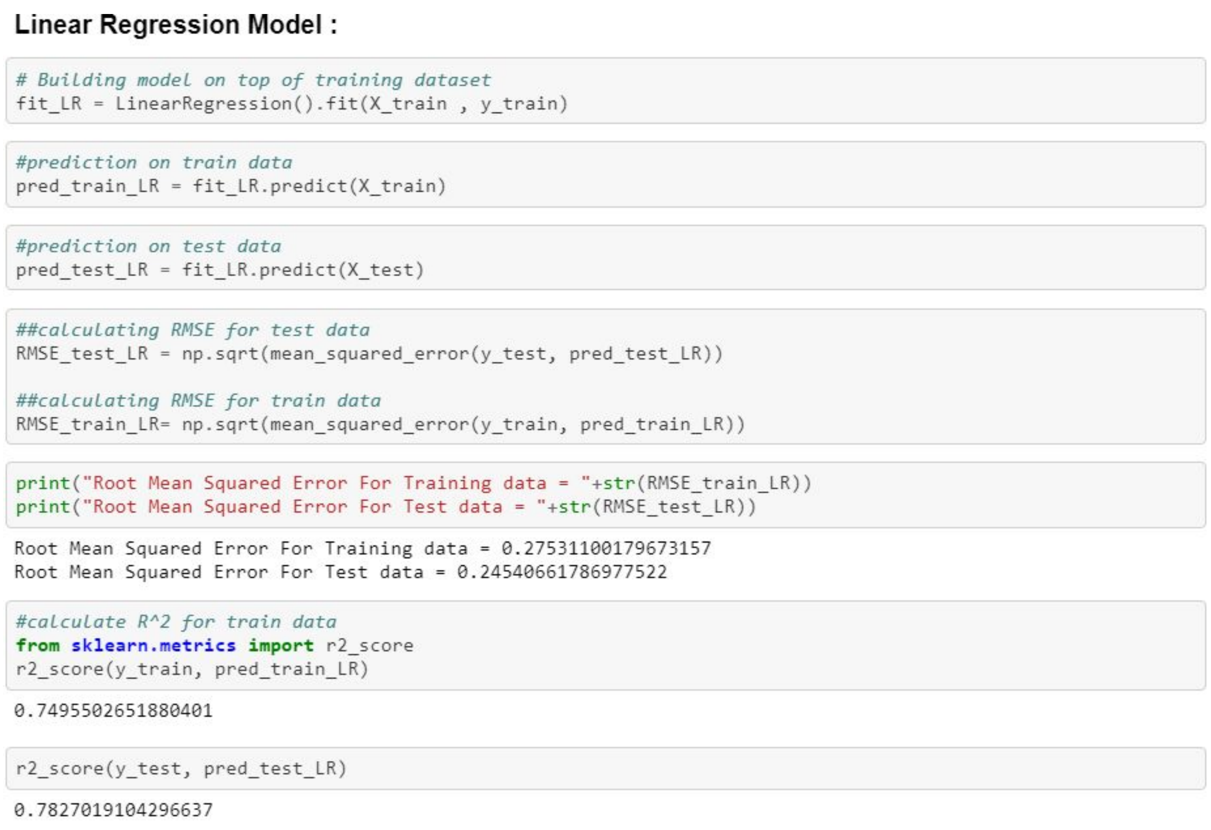
```
##train test split for further modelling
X_train, X_test, y_train, y_test = train_test_split( train.iloc[:, train.columns != 'fare_amount'],
                         train.iloc[:, 0], test_size = 0.20, random_state = 1)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
(12339, 7)
(3085, 7)
```

4.1 Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable). Below is a screenshot of the model we built and its output:



Linear Regression Model :

```
# Building model on top of training dataset
fit_LR = LinearRegression().fit(X_train , y_train)
```

```
#prediction on train data
pred_train_LR = fit_LR.predict(X_train)
```

```
#prediction on test data
pred_test_LR = fit_LR.predict(X_test)
```

```
##calculating RMSE for test data
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

##calculating RMSE for train data
RMSE_train_LR= np.sqrt(mean_squared_error(y_train, pred_train_LR))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_LR))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))
```

```
Root Mean Squared Error For Training data = 0.27531100179673157
Root Mean Squared Error For Test data = 0.24540661786977522
```

```
#calculate R^2 for train data
from sklearn.metrics import r2_score
r2_score(y_train, pred_train_LR)
```

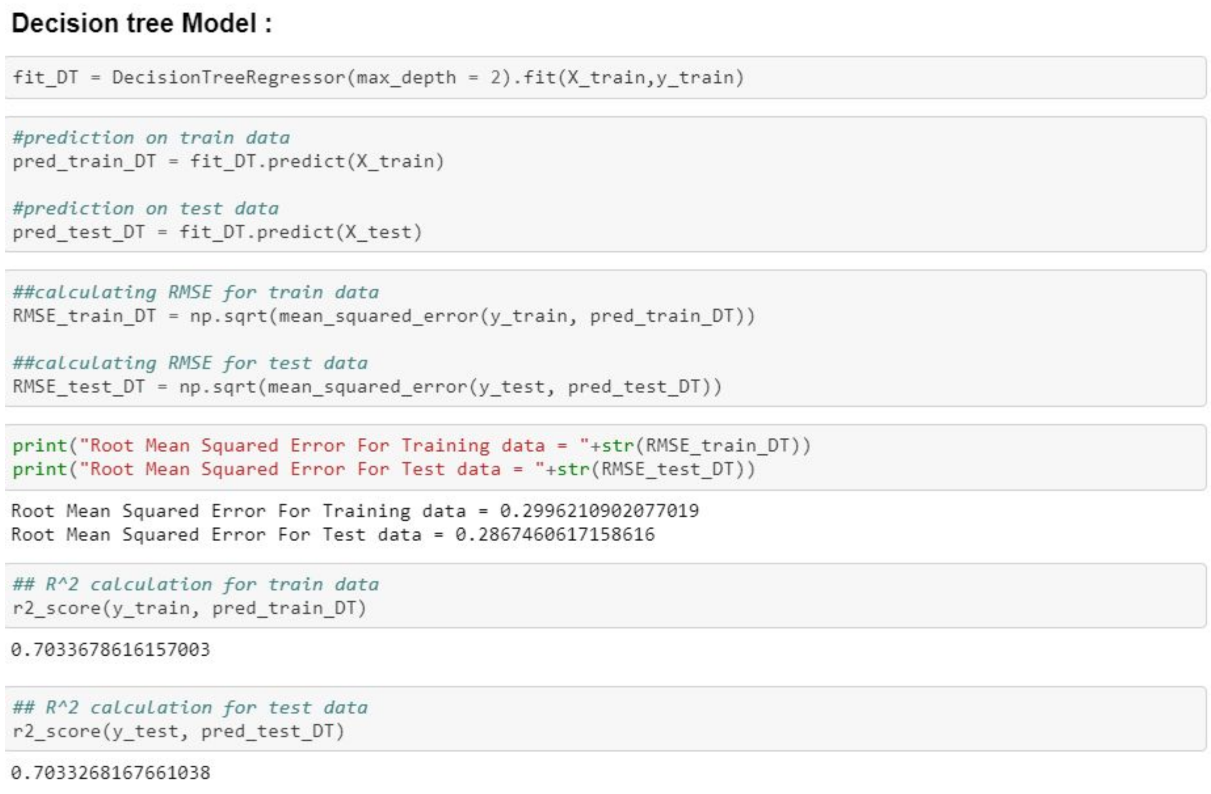
```
0.7495502651880401
```

```
r2_score(y_test, pred_test_LR)
```

```
0.7827019104296637
```

4.2 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 consisting of one parent node and many child nodes. Below is the screenshot of the query we executed and the result shown, we will compare the results of each model in a combined table later on.



Decision tree Model :

```
fit_DT = DecisionTreeRegressor(max_depth = 2).fit(X_train,y_train)
```

```
#prediction on train data
pred_train_DT = fit_DT.predict(X_train)

#prediction on test data
pred_test_DT = fit_DT.predict(X_test)
```

```
##calculating RMSE for train data
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))

##calculating RMSE for test data
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))
```

```
Root Mean Squared Error For Training data = 0.2996210902077019
Root Mean Squared Error For Test data = 0.2867460617158616
```

```
## R^2 calculation for train data
r2_score(y_train, pred_train_DT)
```

```
0.7033678616157003
```

```
## R^2 calculation for test data
r2_score(y_test, pred_test_DT)
```

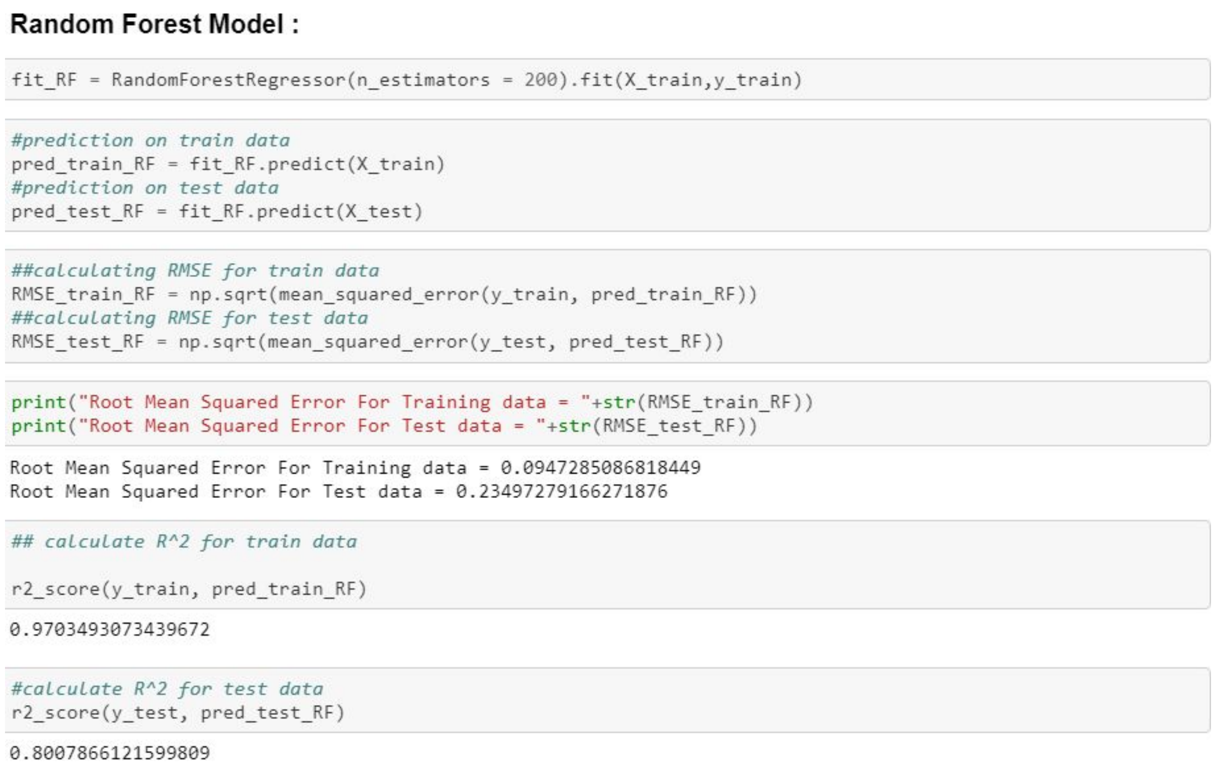
```
0.7033268167661038
```

4.3 Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, 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.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

Below is a screenshot of the model we build and its output:



Random Forest Model :

```
fit_RF = RandomForestRegressor(n_estimators = 200).fit(X_train,y_train)
```

```
#prediction on train data
pred_train_RF = fit_RF.predict(X_train)
#prediction on test data
pred_test_RF = fit_RF.predict(X_test)
```

```
##calculating RMSE for train data
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
##calculating RMSE for test data
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_RF))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_RF))
```

```
Root Mean Squared Error For Training data = 0.0947285086818449
Root Mean Squared Error For Test data = 0.23497279166271876
```

```
## calculate R^2 for train data

r2_score(y_train, pred_train_RF)
```

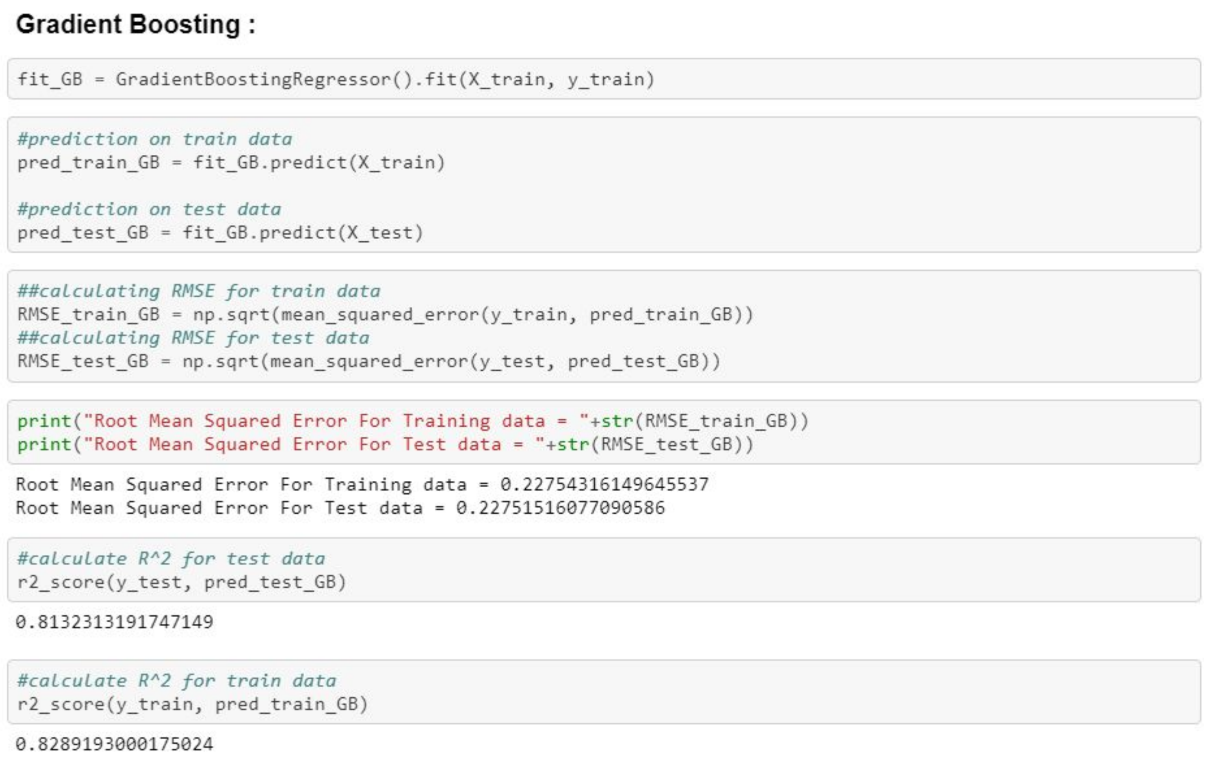
```
0.9703493073439672
```

```
#calculate R^2 for test data
r2_score(y_test, pred_test_RF)
```

```
0.8007866121599809
```

4.4 Gradient Boosting

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. Below is a screenshot of the model we build and its output:


Gradient Boosting :

```
fit_GB = GradientBoostingRegressor().fit(X_train, y_train)
```

```
#prediction on train data
pred_train_GB = fit_GB.predict(X_train)

#prediction on test data
pred_test_GB = fit_GB.predict(X_test)
```

```
##calculating RMSE for train data
RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
##calculating RMSE for test data
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_GB))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_GB))
```

```
Root Mean Squared Error For Training data = 0.22754316149645537
Root Mean Squared Error For Test data = 0.22751516077090586
```

```
#calculate R^2 for test data
r2_score(y_test, pred_test_GB)
```

```
0.8132313191747149
```

```
#calculate R^2 for train data
r2_score(y_train, pred_train_GB)
```

```
0.8289193000175024
```

4.5 Hyper Parameters

Tunings for optimizing the results Model hyperparameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter because this is set by the data scientist. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation dataset - for a problem.

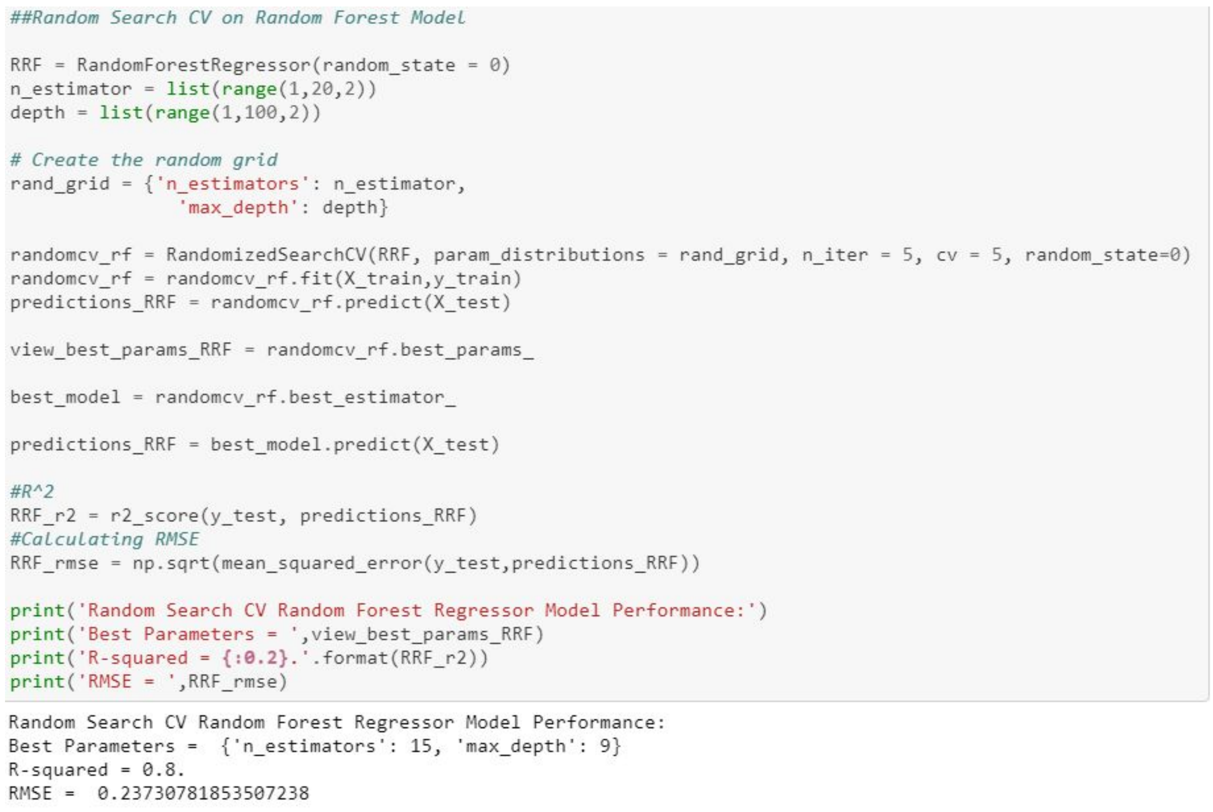
Here we have used two hyper parameters tuning techniques

➢ **Random Search CV**

➢ **Grid Search CV**

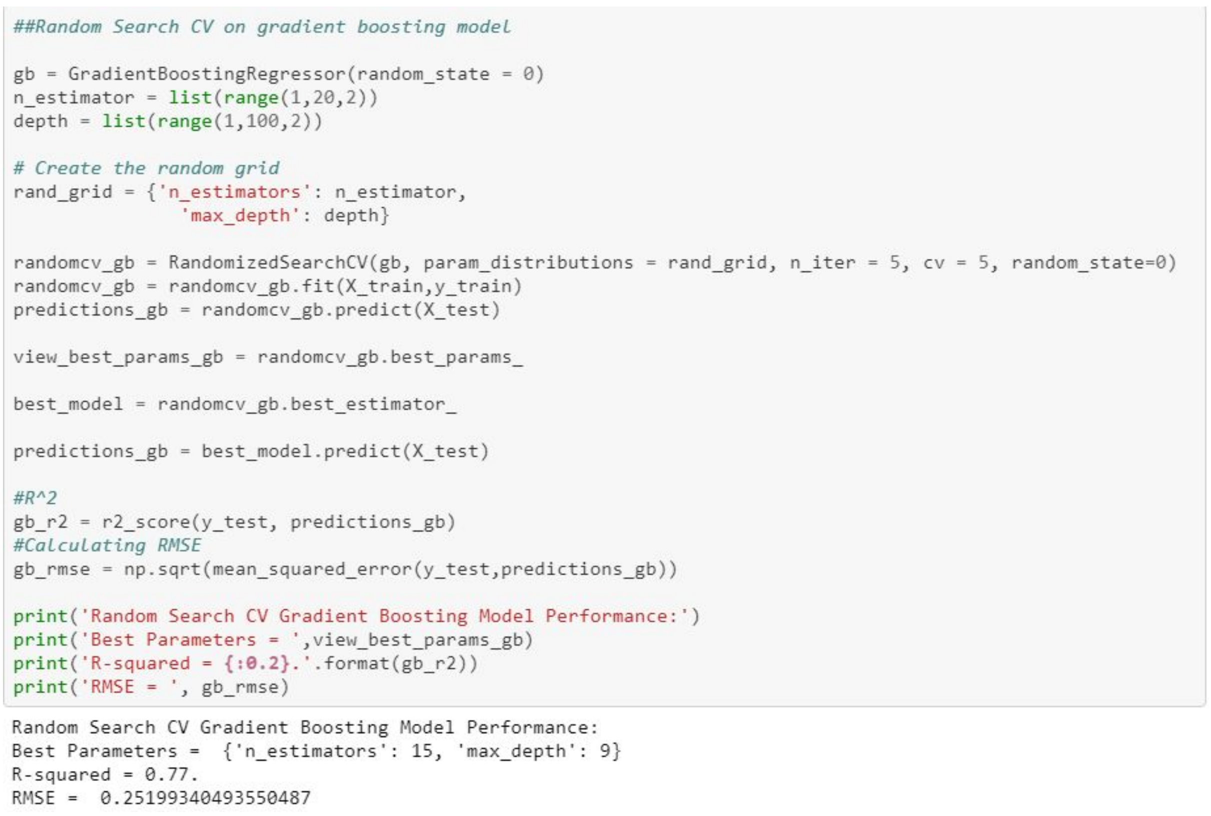
1. Random Search CV: This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.

2. Grid Search CV: This algorithm set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient.



```python
##Random Search CV on Random Forest Model

RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
               'max_depth': depth}

randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_rf = randomcv_rf.fit(X_train,y_train)
predictions_RRF = randomcv_rf.predict(X_test)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)

#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating RMSE
RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('RMSE = ',RRF_rmse)
```

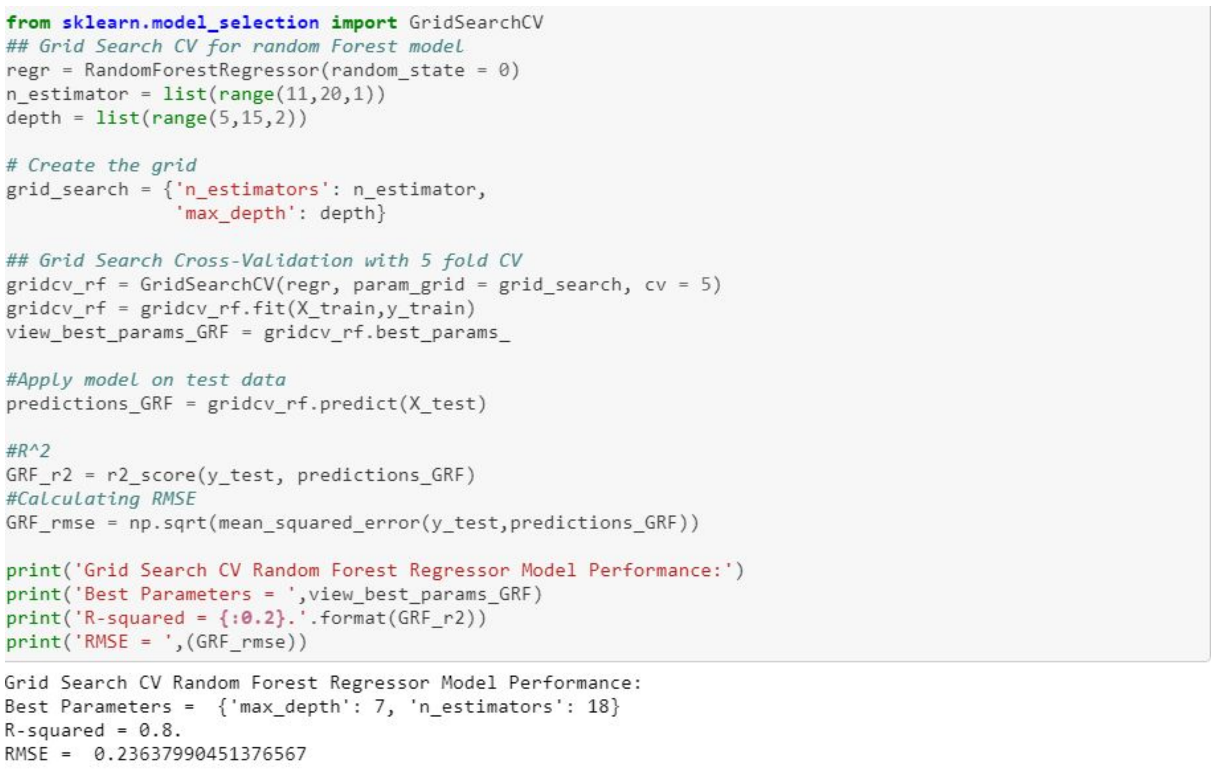
```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.8.
RMSE =  0.23730781853507238
```



```
##Random Search CV on gradient boosting model

gb = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
               'max_depth': depth}

randomcv_gb = RandomizedSearchCV(gb, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_gb = randomcv_gb.fit(X_train,y_train)
predictions_gb = randomcv_gb.predict(X_test)

view_best_params_gb = randomcv_gb.best_params_

best_model = randomcv_gb.best_estimator_

predictions_gb = best_model.predict(X_test)

#R^2
gb_r2 = r2_score(y_test, predictions_gb)
#Calculating RMSE
gb_rmse = np.sqrt(mean_squared_error(y_test,predictions_gb))

print('Random Search CV Gradient Boosting Model Performance:')
print('Best Parameters = ',view_best_params_gb)
print('R-squared = {:0.2}.'.format(gb_r2))
print('RMSE = ', gb_rmse)
```

```
Random Search CV Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.77.
RMSE =  0.25199340493550487
```

Check results after using Grid Search CV on Random forest and gradient boosting model:



```
from sklearn.model_selection import GridSearchCV
## Grid Search CV for random Forest model
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
               'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_rf = GridSearchCV(regr, param_grid = grid_search, cv = 5)
gridcv_rf = gridcv_rf.fit(X_train,y_train)
view_best_params_GRF = gridcv_rf.best_params_

#Apply model on test data
predictions_GRF = gridcv_rf.predict(X_test)

#R^2
GRF_r2 = r2_score(y_test, predictions_GRF)
#Calculating RMSE
GRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_GRF))

print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(GRF_r2))
print('RMSE = ',(GRF_rmse))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 18}
R-squared = 0.8.
RMSE =  0.23637990451376567
```

Chapter 5 Conclusion

5.1 Model Evaluation

The main concept of looking at what is called residuals or difference between our predictions and actual outcomes.

In general, most data scientists use two methods to evaluate the performance of the model:

I. 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.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

II. R Squared(R^2): is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficient of multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained

III. We have shown both train and test data results, the main reason behind showing both the results is to check whether our data is overfitted or not.

Below table shows the model results before applying hyper tuning:

Model Name	RMSE		R Squared	
Linear Regression	0.27	0.25	0.74	0.77
Decision Tree	0.30	0.28	0.70	0.70
Random Forest model	0.09	0.23	0.96	0.79
Gradient Boosting	0.22	0.22	0.82	0.81

Below table shows results post using hyper parameter tuning techniques:

Model Name	Parameter	RMSE (Test)	R Squared (Test)
Random Search CV	Random Forest	0.24	0.79
	Gradient Boosting	0.25	0.77
Grid Search CV	Random Forest	0.23	0.80
	Gradient Boosting	0.24	0.79

Above table shows the results after tuning the parameters of our two best suited models i.e. **Random Forest** and **Gradient Boosting**. For tuning the parameters, we have used Random Search CV and Grid Search CV under which we have given the range of n_estimators, depth and CV folds.

5.2 Model Selection

On the basis RMSE and R Squared results a good model should have **least** RMSE and **max** R Squared value. So, from above tables we can see:

• From the observation of all RMSE Value and R-Squared Value we have concluded that,

• Both the models- Gradient Boosting Default and Random Forest perform comparatively well after comparing their RMSE and R-Squared value.

• After this, I chose Random Forest CV and Grid Search CV to apply cross validation technique and see changes brought about by that.

• After applying tunings, Random forest model shows best results compared to gradient boosting.

• So finally, we can say that **Random forest model is the best method** to make prediction for this project with highest explained variance of the target variables and lowest error chances with parameter tuning technique Grid Search CV.

Finally, I used this method to predict the target variable for the test data file shared in the problem statement. Results that I found are attached with my submissions.

References

1. https://edwisor.com/career-data-scientist

2. https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysispython/

3. https://www.udemy.com/python-for-data-science-and-machine-learning-bootcamp/

4. https://www.linkedin.com/learning/nlp-with-python-for-machine-learning-essential-training/evaluate-gradient-boosting-model-performance

5. https://stackoverflow.com/

6. https://www.guru99.com/r-scatter-plot-ggplot2.html