

## Exploratory Data Analysis(EDA)

```
In [78]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
matplotlib inline

import warnings
warnings.filterwarnings('ignore')
pd.options.set_option('display.max_columns',None)

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix, roc_auc_score
from sklearn.metrics import roc_curve, auc
```

```
Out[79]: df
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076		34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.26	0.68	9.8	5
2	7.8	0.78	0.54	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.26	0.06	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
In [80]: df.tail()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.640	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.315	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

```
In [81]: df.shape
```

```
Out[81]: (1599, 12)
```

```
In [82]: df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.19637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467324	0.996747	3.311113	0.658149	5.96
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.01
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	3.100000	0.330000	3.00
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995000	3.210000	0.550000	5.00
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	6.00
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	7.00
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	11.00

```
In [83]: df.info()
```

```
class <pandas.core.frame.DataFrame>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  --
 0   fixed acidity        1599 non-null   float64
 1   volatile acidity     1599 non-null   float64
 2   citric acid          1599 non-null   float64
 3   residual sugar       1599 non-null   float64
 4   chlorides            1599 non-null   float64
 5   free sulfur dioxide  1599 non-null   float64
 6   total sulfur dioxide 1599 non-null   float64
 7   density              1599 non-null   float64
 8   pH                  1599 non-null   float64
 9   sulphates            1599 non-null   float64
10   alcohol              1599 non-null   float64
11   quality              1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```
In [84]: df.columns
```

```
Out[84]: Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
        'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
        'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
In [85]: df.nunique()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	quality
fixed acidity	96										
volatile acidity	143										
citric acid	80										
residual sugar	91										
chlorides	153										
free sulfur dioxide	60										
total sulfur dioxide	144										
density	416										
pH	89										
sulphates	96										
alcohol	65										
quality	6										
dtype:	int64										

```
In [86]: df["quality"].unique()
```

```
Out[86]: array([5, 6, 7, 4, 8, 3], dtype=int64)
```

```
In [87]: df.quality.value_counts()
```

```
Out[87]: 5    681
        6    638
        7    199
        4     53
        8     18
        3     10
        Name: quality, dtype: int64
```

```
In [88]: df.quality.value_counts(normalize = True)
```

```
Out[88]: 5    0.425891
        6    0.398999
        7    0.124453
        4    0.033146
        8    0.011327
        3    0.006254
        Name: quality, dtype: float64
```

```
In [89]: # Finding the missing values
df.isnull().sum()
```

```
Out[89]: fixed acidity      0
        volatile acidity  0
        citric acid      0
        residual sugar    0
        chlorides         0
        free sulfur dioxide 0
        total sulfur dioxide 0
        density           0
        pH               0
        sulphates         0
        alcohol           0
        quality           0
        dtype: int64
```

```
In [90]: [features for features in df.columns if df[features].isnull().sum()>0]
# There is no missing value in the entire dataset.
```

```
Out[90]: []
```

```
In [91]: # Relationship analysis
```

```
In [92]: sns.countplot(x = "quality", data = df)
```

# The wines with medium quality are more present within the given dataset.  
# The plot is slightly skewed.

```
Out[92]: <AxesSubplot:xlabel='quality', ylabel='count'>
```

```
In [93]: df1 = df.select_dtypes([np.int, np.float])
for i,col in enumerate(df1.columns):
    plt.figure(i)
    sns.boxplot(x = "quality", y = col, data= df1)
```

# We can understand that fixed acidity, residual sugar, density and pH show uniform distribution  
# with the quality. While citric acid and chlorides show ~ve relation and remaining  
# shows +ve relation with quality.

```
In [94]: # Box plots
```

```
In [95]: df1 = df.select_dtypes([np.int, np.float])
for i,col in enumerate(df1.columns):
    plt.figure(i)
    sns.boxplot(x = "quality", y = col, data= df1)
```

# While considering the box plot we can understand that many data points are there as outliers.  
# But since this is a classification problem we can keep the outliers as it is.

```
In [96]: sns.pairplot(df)
```

```
Out[96]: <seaborn.axisgrid.PairGrid at 0x2987406af0>
```

```
In [ ]: # Here from the above pairplot we are getting both histogram and scatter plot. The histogram on the diagonal of
# the distribution of a single variable. We can understand from the histogram that density and pH are normally
# distributed while the remaining are skewed.
# The scatter plot of density with fixed acidity shows a positive relation while the pH shows a negative relat.
# The scatter plot of each feature with quality is very different from the remaining scatter plots.
```

## Setting out correlation matrix

```
In [97]: correlation = df.corr()
```

```
In [98]: fig,ax=plt.subplots(figsize=(20,10))
sns.heatmap(correlation, ax ticks = correlation.columns, y ticks labels = correlation.columns, annot = True, x= correlation.columns, y= correlation.columns)
```

# Here the max +ve correlation value is 0.67 and it is between fixed acidity and citric acid content.  
# The min -ve correlation value is -0.68 which is between pH and fixed acidity.  
# There is a highest correlation value of 0.48 between alcohol content and quality.

```
Out[98]: <AxesSubplot>
```

## Data Preprocessing

```
In [99]: ## Treating the outliers.
# IQR method of detecting outliers
def iqr_outlier(data):
    Q1=np.percentile(data, 25, interpolation = 'midpoint')
    Q3 = np.percentile(data, 75, interpolation = 'midpoint')
    IQR=Q3-Q1
    lo=Q1-1.5*IQR
    up=Q3+1.5*IQR
    outlier=[]
    for x in data:
        if (x<lo or x>up):
            outlier.append(x)
    print('outlier in the dataset are:', outlier)
```

iqr\_outlier(df['alcohol'])

# From the pairplot we can understand the alcohol feature is skewed.  
# Hence IQR method is used here.

outlier in the dataset are: [14.0, 14.0, 14.0, 14.0, 14.9, 14.0, 13.6, 13.6, 13.6, 14.0, 14.0, 13.5666667, 13.6]

```
In [100]: # Using z-score method
def norm_outlier(data):
    outlier=[]
    mean=np.mean(data)
    std=np.std(data)
    for x in data:
        z=(x-mean)/std
        if z>3 or z<-3:
            outlier.append(x)
    print('outlier in the dataset are: ', outlier)
    final = [i*std+mean for i in outlier]
    print(final)
```

norm\_outlier(df['density'])

#The feature density is normally distributed.  
# Hence z-score method is used to find the outliers here.

outlier in the dataset are: [1.0032, 1.0026, 1.00315, 1.00315, 1.00315, 1.0026, 0.99064, 0.99064, 1.00289, 0.99007, 0.99007, 0.9902, 0.9909, 0.99084, 1.00369, 1.00369, 1.00242, 1.00242]

```
In [ ]: 
```

```
In [101]: 
```

```
In [102]: # Splitting of data into train and test.
```

```
In [103]: x = df.drop(["quality"], axis=1)
y = df["quality"]
```

```
In [103]: x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 42, train_size = 0.80)
```

```
In [104]: # Feature scaling.
# Scale the data between -1 to +1.
# For distance based models like Logistic regression feature scaling is required.
# For non distance based models like Decision tree and Random forest feature scaling is not required.
# It means that we have to keep different datasets for both.
# For distance based model we can use the feature scaled dataset(x_train_scaled) and for the non distance based model we can use the original dataset(x_train).
# For row dataset(x_train) itself.
```

```
In [105]: ss = StandardScaler()
x_train_scaled = ss.fit_transform(x_train)
x_test_scaled = ss.transform(x_test)
```

```
In [ ]: 
```

## Fitting of different classification models.

```
In [106]: #Create a function within many Machine Learning Models
# Here scaled values of x and y values are using for the models.
def models_scaled(x_train,y_train):
    #Using Logistic Regression Training Accuracy:
    log = LogisticRegression(random_state = 0)
    log.fit(X_train, Y_train)

    #Using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor algorithm
    knn = KNeighborsClassifier(n_neighbors = 5, metric = "minkowski", p = 2)
    knn.fit(X_train, Y_train)

    #Using SVC method of svm class to use Support Vector Machine Algorithm
    svc_lin = SVC(kernel = "linear", random_state = 0)
    svc_lin.fit(X_train, Y_train)

    #Using SVM method of svm class to use Kernel SVM Algorithm
    svc_rbf = SVC(kernel = "rbf", random_state = 0)
    svc_rbf.fit(X_train, Y_train)

    #Print model accuracy on the training data.
    print('[0]Logistic Regression Training Accuracy:', log.score(X_train, Y_train))
    print('[1]K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
    print('[2]Support Vector Machine (linear Classifier) Training Accuracy:', svc_lin.score(X_train, Y_train))
    print('[3]Support Vector Machine (RBF Classifier) Training Accuracy:', svc_rbf.score(X_train, Y_train))

    return log, knn, svc_lin, svc_rbf,
```

```
In [107]: model = models_scaled(x_train_scaled,y_train)
```

```
[0]Logistic Regression Training Accuracy: 0.619233776387803
[1]K Nearest Neighbor Training Accuracy: 0.737782408131553
[2]Support Vector Machine (linear Classifier) Training Accuracy: 0.5934323690383112
[3]Support Vector Machine (RBF Classifier) Training Accuracy: 0.679437060232838
```



```
# One dataset without scaling is used for training the models.
def models(X_train,Y_train):
    #Using GaussianNB method of naive_bayes class to use Naive Bayes Algorithm
    gauss = GaussianNB()
    gauss.fit(X_train, Y_train)

    #Using DecisionTreeClassifier of tree class to use Decision Tree Algorithm
    tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    tree.fit(X_train, Y_train)

    #Using RandomForestClassifier method of ensemble class to use Random Forest Classification algorithm
    forest = RandomForestClassifier(max_depth=70, max_features=2, min_samples_leaf=3,
                                   min_samples_split=8, n_estimators=300,random_state =22)
    forest.fit(X_train, Y_train)

    #print model accuracy on the training data.

    print('[4]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train, Y_train))
    print('[5]Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
    print('[6]Random Forest Classifier Training Accuracy:', forest.score(X_train, Y_train))

    return gauss, tree, forest

In [109]:
#Get and train all of the models
model = models(X_train,y_train)

[4]Gaussian Naive Bayes Training Accuracy: 0.5731039874902267
[5]Decision Tree Classifier Training Accuracy: 1.0
[6]Random Forest Classifier Training Accuracy: 0.8881939014855356

In [110]:
# From the above results we can conclude that Random Forest Classifier is giving a better training accuracy of
# Eventhough Decision Tree Classifier is giving 100% accuracy we can not rely on it.
# It may be due to over fitting as well.
# Let us evaluate the Random forest model by finding the accuracy score of test data
# Plotting confusion matrix, auc curve etc. Then finally let us
# do hyperparameter tuning as well.

In [111]:
# Instantiate and fit the RandomForestClassifier
forest = RandomForestClassifier(max_depth=70, max_features=2, min_samples_leaf=3,
                               min_samples_split=8, n_estimators=300,random_state =22)
forest.fit(X_train, y_train)
# Make predictions for the test set
y_pred_rf = forest.predict(x_test)
# View accuracy score
accuracy_score(y_test, y_pred_rf)

Out[111]:
0.63125

In [32]:
# Here we can see that the test accuracy is 63.125% and training accuracy is88.88%, which means that the model
# overfitting. We have to do hyperparameter tuning and cross validation to make it better.

In [112]:
# View confusion matrix for test data and predictions
confusion_matrix(y_test, y_pred_rf)

Out[112]:
array([[ 0,  0,  1,  0,  0,  0],
       [ 0,  0,  0,  3,  0,  0],
       [ 0,  0,  97,  32,  0,  0],
       [ 0,  0,  35,  92,  0,  0],
       [ 0,  0,  0,  29, 13,  0],
       [ 0,  0,  0,  2,  3,  0]], dtype=int64)

In [113]:
# View the classification report for test data and predictions
print(classification_report(y_test, y_pred_rf))

precision    recall  f1-score   support

      3         0.00         0.00         0.00         1
      4         0.00         0.00         0.00        10
      5         0.69         0.75         0.72        130
      6         0.59         0.70         0.63        132
      7         0.62         0.31         0.41         42
      8         0.00         0.00         0.00         5

 accuracy         0.63         0.63        320
 macro avg         0.32         0.29         0.29        320
weighted avg         0.60         0.63         0.61        320
```

## GridSearchCV

```
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [70,80, 90],
    'max_features': ['all', 'sqrt'],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300,1000],
    'criterion': ['gini', 'entropy'],
    'oob_score': [False],
    'random_state': [42],
}

# Create a based model
forest = RandomForestClassifier()
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = forest, param_grid = param_grid,
                           cv = 5, n_jobs = 3, verbose = True)

In [115]:
grid_search.fit(X_train, y_train)
grid_search.best_params_

Fitting 9 folds for each of 432 candidates, totalling 3888 fits

Out[115]:
{'bootstrap': True,
 'criterion': 'entropy',
 'max_depth': 70,
 'max_features': 2,
 'min_samples_leaf': 3,
 'min_samples_split': 8,
 'n_estimators': 1000,
 'oob_score': False,
 'random_state': 42}

In [116]:
best_grid = grid_search.best_estimator_

In [117]:
best_grid

Out[117]:
RandomForestClassifier(criterion='entropy', max_depth=70, max_features=2,
                        min_samples_leaf=3, min_samples_split=8,
                        n_estimators=1000, random_state=42)

In [118]:
forest_finetuned = forest.set_params(criterion='entropy', max_depth=70, max_features=2,
                                     min_samples_leaf=3, min_samples_split=8,
                                     n_estimators=1000)

In [119]:
model_final = forest_finetuned.fit(X_train, y_train)
y_pred = model_final.predict(x_test)
print(accuracy_score(y_test,y_pred))

0.640625

In [ ]:

In [121]:
# Cross validation
from sklearn.model_selection import cross_val_score
scores = cross_val_score(forest, X_train, y_train, cv=5)
print(scores)
print(scores.mean())
print("accuracy of training data is: ", scores.mean())

(0.65034965 0.66197183 0.64084507 0.62676056 0.68309859 0.66901408
 0.70422535 0.69014085 0.77464789)
0.6778948707077937
accuracy of training data is:  0.6778948750779737

In [ ]:
# Here we can observe that after fine tuning both training and testing accuracy have reduced.
# The model is no more overfitting.

In [74]:
difference = pd.DataFrame(y_test - y_pred)
difference.value_counts()

Out[74]:
quality
0         203
1          66
-1         45
-2          4
2           2
dtype: int64

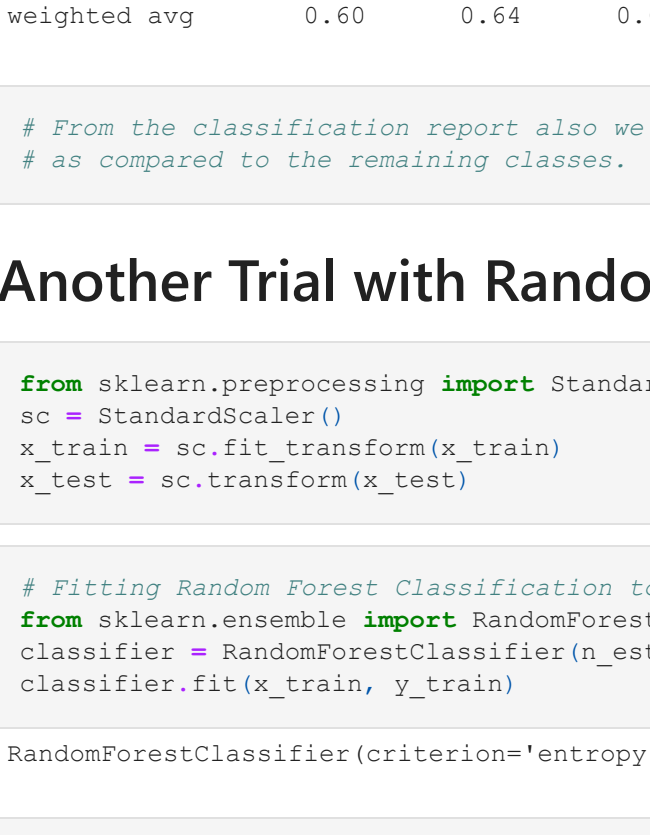
In [75]:
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test,y_pred)
conf_matrix

Out[75]:
array([[ 0,  0,  1,  0,  0,  0],
       [ 0,  0,  0,  3,  0,  0],
       [ 0,  0,  98,  32,  0,  0],
       [ 0,  0,  32,  94,  0,  0],
       [ 0,  0,  0,  31, 11,  0],
       [ 0,  0,  0,  2,  3,  0]], dtype=int64)

In [76]:
ax= plt.subplot()
sns.heatmap(conf_matrix,annot=True, ax= ax, fmt = ".2f")

ax.set_xlabel('Predicted labels');ax.set_ylabel('Actual labels');
ax.set_title('Actual vs. Predicted Confusion Matrix');
ax.xaxis.set_ticklabels([3,4,5,6,7,8]); ax.yaxis.set_ticklabels([3,4,5,6,7,8]);

plt.show()
```



```
In [ ]:
# From the above confusion matrix we can conclude that the TF for class 5 is 98%, for class 6 is 94%
# and for class 7 it is 11%. For class 5 and 6 the model is predicting with high accuracy.

In [122]:
print(classification_report(y_test, y_pred))

precision    recall  f1-score   support

      3         0.00         0.00         0.00         1
      4         0.00         0.00         0.00        10
      5         0.71         0.76         0.74        130
      6         0.59         0.71         0.65        132
      7         0.55         0.29         0.37         42
      8         0.00         0.00         0.00         5

 accuracy         0.31         0.29         0.29        320
 macro avg         0.31         0.29         0.29        320
weighted avg         0.60         0.64         0.61        320
```

```
In [ ]:
# From the classification report also we can understand that for class 5 and 6 th precision,recall and f1- score
# as compared to the remaining classes.
```

## Another Trial with RandomizedSearchCV

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

In [34]:

In [36]:
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 50)
classifier.fit(X_train, y_train)

Out[36]:
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=50)

In [39]:
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

rf = RandomForestClassifier(n_jobs=-1)
rf_p_dist={'max_depth':(3,5,10,None),
           'n_estimators':(10,100,200,300,400,500),
           'max_features':randint(1,12),
           'criterion':('gini','entropy'),
           'bootstrap':(True,False),
           'min_samples_leaf':randint(1,4),
           }

In [42]:
def hypertuning_rscv(test, p_dist, nbr_iter,x,y):
    rdmsearch = RandomizedSearchCV(est, param_distributions=p_dist,
                                   n_jobs=-1, n_iter=nbr_iter, cv=5)
    #CV = Cross-Validation ( here using Stratified KFold CV)
    rdmsearch.fit(x,y)
    ht_params = rdmsearch.best_params_
    ht_score = rdmsearch.best_score_
    return ht_params, ht_score

rf_parameters, rf_ht_score = hypertuning_rscv(test, rf_p_dist, 40, X, y)

classifier=RandomForestClassifier(n_jobs=-1, n_estimators=300,bootstrap= True,criterion='entropy',max_depth=3,
```

```
In [44]:
# Predicting the Test set results
y_pred = classifier.predict(x_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)

accuracy_score=accuracy_score(y_test,y_pred)
print(accuracy_score)
from sklearn.model_selection import cross_val_score
cross_val=cross_val_score(classifier,x,y,cv=10,scoring='accuracy').mean()
print(cross_val)

0.6525
0.5741155660377358

In [ ]:
```

## Conclusion

```
In [ ]:
# Eventhough tried both GridSearchCV and RandomizedSearchCV, with GridSearchCV better accuracy is getting for l
# training and testing. RandomizedSearchCV is taking less time for computation as compared with the GridSearchCV
# Since the results are better with GridSearchCV we can go with that.

# Also the randomforest was overfitting initially. But, doing better after the hyper tuning.

# Final Model Accuracy:

# For training: 67.789%
# For testing : 64.063%

In [ ]:
```