Heart Attack Analysis & Prediction

SEPTEMBER 2, 2021

MUHAMMAD AMIN GHIAS

Table of Contents

ABSTRACT:

Health is the most important aspect of life, people need to take extra precautions to remain healthy. In this era heart attack has become common, this project will analyze the data of heart attack and predict which features cause heart attack

INTRODUCTION

As heart attack has become common nowadays it is important for people to understand what causes heart attack. This report will analyze the data, which features and parameters are common in order for it to occur, Further more using the dataset this project will predict people who are likely to have heart attack.

MAIN OBJECTIVE:

The main objective of this project is to help in predicting which patients people are more likely to suffer from heart attack and find the best model with least error.

DATA ACQUISITION AND WRANGLING

DATA SOURCE

For this project we will use online data the source is from kaggle

CSV file heart.cv from url:" https://www.kaggle.com/rashikrahmanpritom/heart-attack-analysis-prediction-dataset"

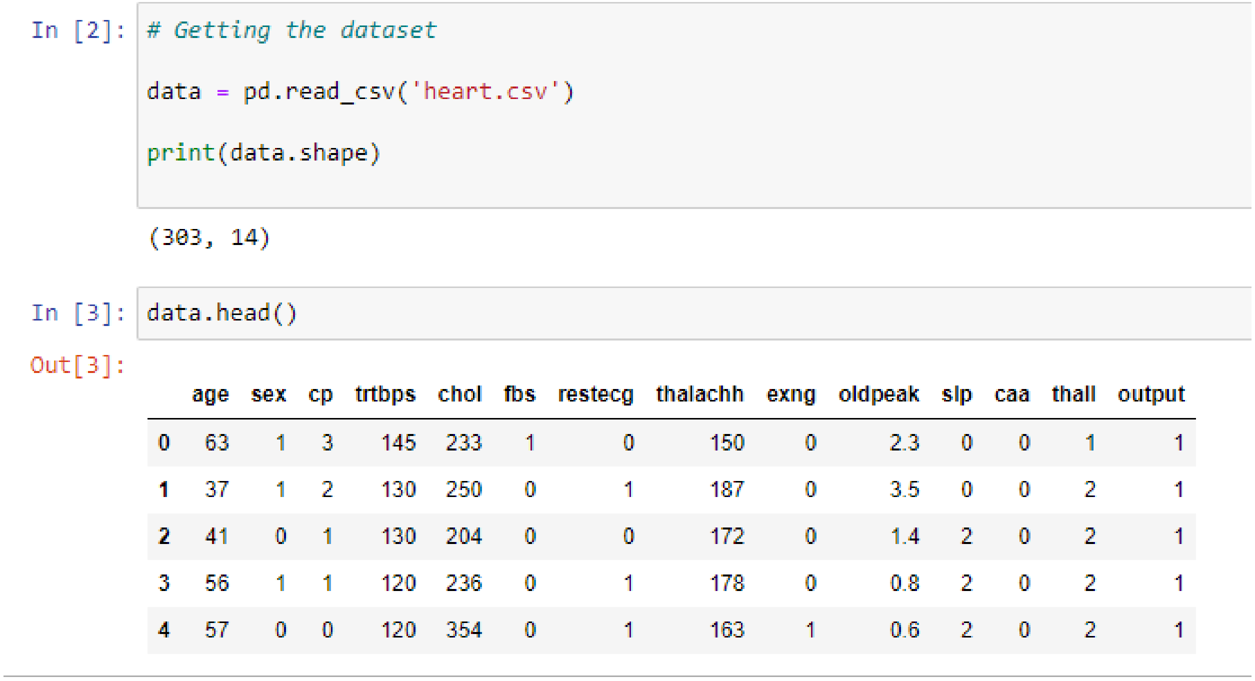
The Dataset is for heart attack classification, following is the dataframe which consists of 14 columns and 303 rows

DATA DESCRIPTION

The description of columns are:

- age - age in years
- sex - sex (1 = male; 0 = female)
- cp - chest pain type (1 = typical angina; 2 = atypical angina; 3 = non-anginal pain; 0 = asymptomatic)
- trestbps - resting blood pressure (in mm Hg on admission to the hospital)
- chol - serum cholestoral in mg/dl
- fbs - fasting blood sugar > 120 mg/dl (1 = true; 0 = false)
- restecg - resting electrocardiographic results (1 = normal; 2 = having ST-T wave abnormality; 0 = hypertrophy)
- thalach - maximum heart rate achieved
- exang - exercise induced angina (1 = yes; 0 = no)
- oldpeak - ST depression induced by exercise relative to rest

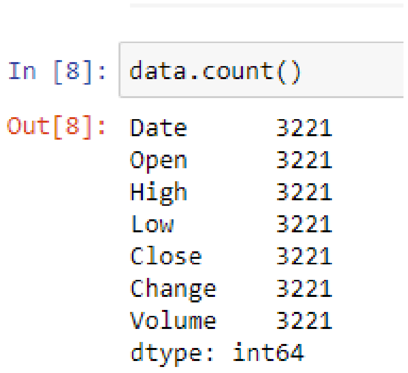
- slope - the slope of the peak exercise ST segment (2 = upsloping; 1 = flat; 0 = downsloping)
- ca - number of major vessels (0-3) colored by flourosopy
- thal - 2 = normal; 1 = fixed defect; 3 = reversable defect
- output - 0= less chance of heart attack 1= more chance of heart attack, the predicted attribute - diagnosis of heart disease (angiographic disease status) (Value 0 = < diameter narrowing; Value 1 = > 50% diameter narrowing)



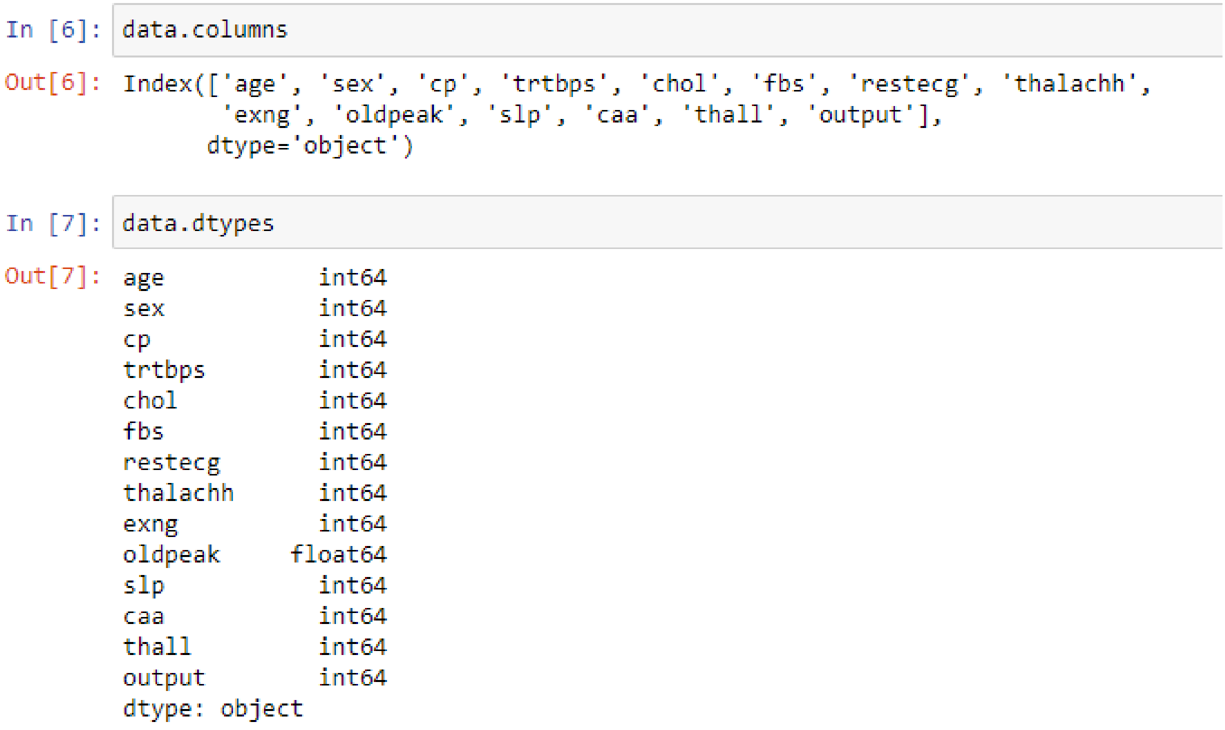
```
In [2]: # Getting the dataset

        data = pd.read_csv('heart.csv')

        print(data.shape)

        (303, 14)
```

```
In [3]: data.head()
Out[3]:
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

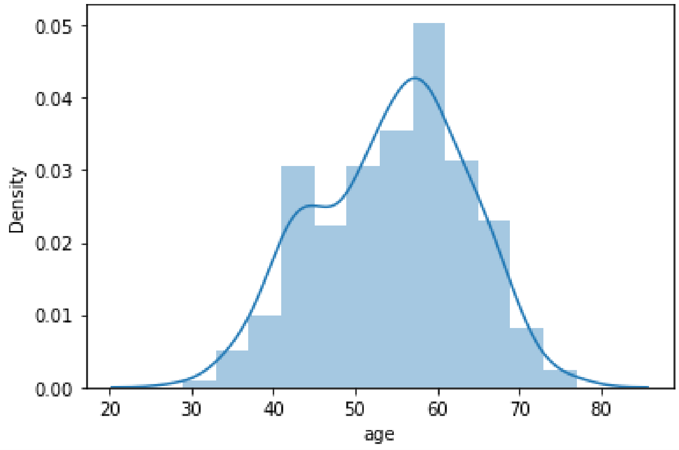


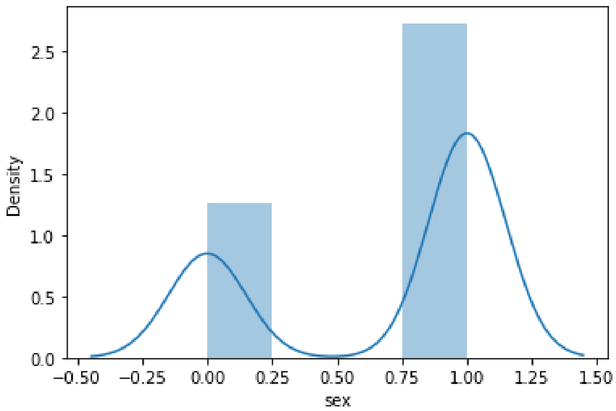
```
In [8]: data.count()

Out[8]: Date       3221
        Open       3221
        High       3221
        Low        3221
        Close      3221
        Change     3221
        Volume     3221
        dtype: int64
```

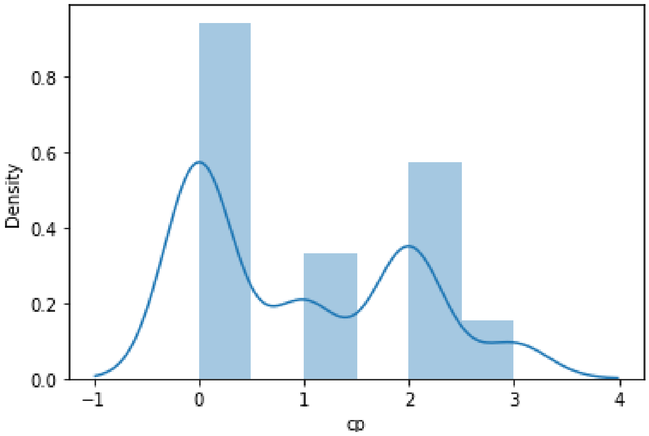


```
In [6]: data.columns

Out[6]: Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
               'exng', 'oldpeak', 'slp', 'caa', 'thall', 'output'],
              dtype='object')
```

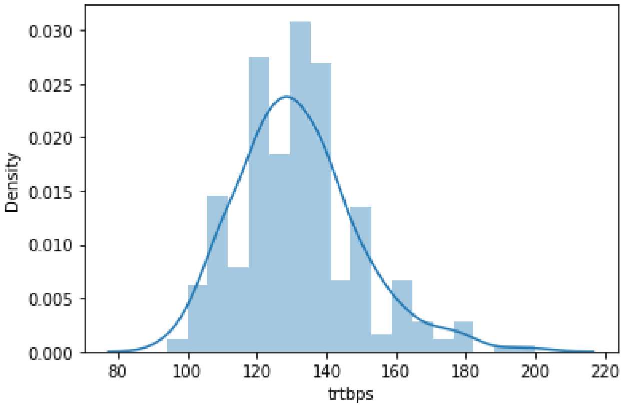
```
In [7]: data.dtypes

Out[7]: age           int64
        sex           int64
        cp            int64
        trtbps        int64
        chol          int64
        fbs           int64
        restecg       int64
        thalachh      int64
        exng          int64
        oldpeak     float64
        slp           int64
        caa           int64
        thall         int64
        output        int64
        dtype: object
```



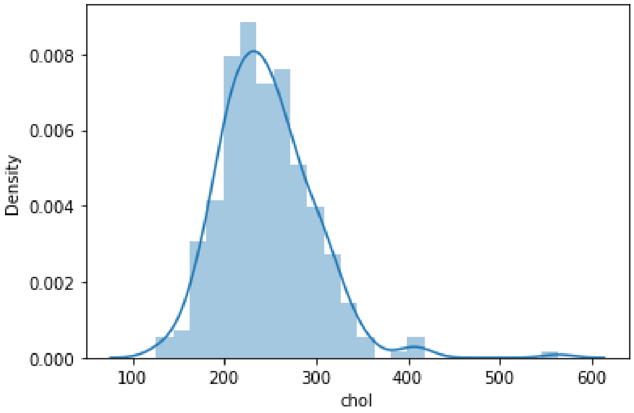




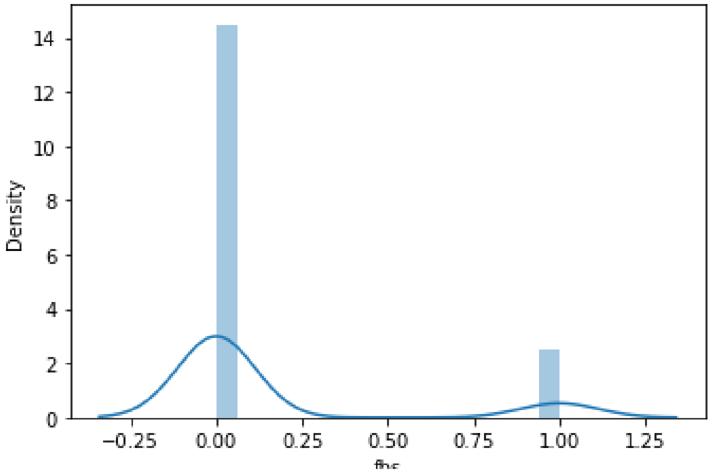
0.0
0.2
0.4
0.6
0.8
Density
−1
0
1
2
3
4
cp

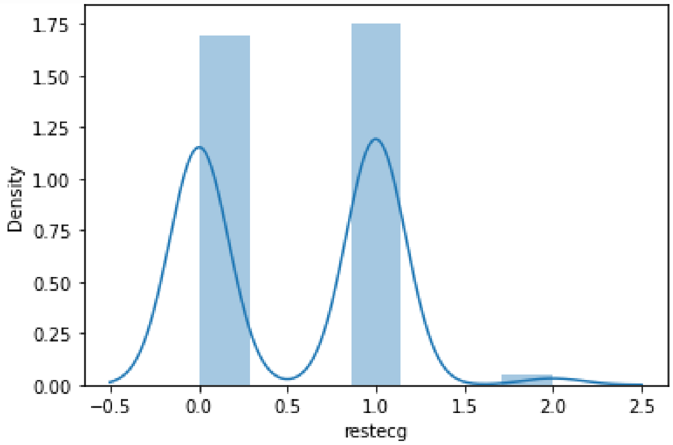


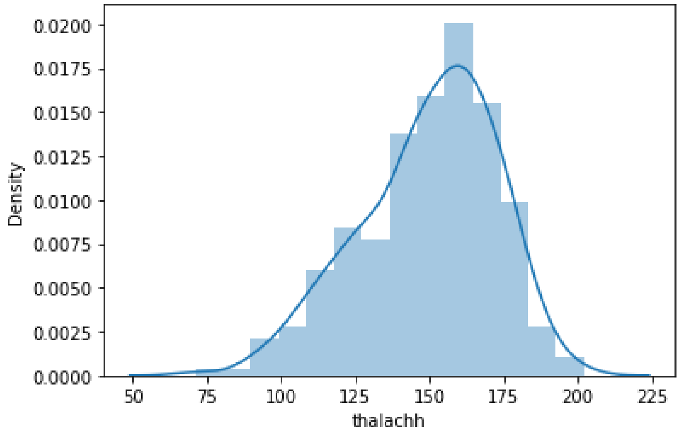
0.000
0.005
0.010
0.015
0.020
0.025
0.030
Density
80
100
120
140
160
180
200
220
trtbps

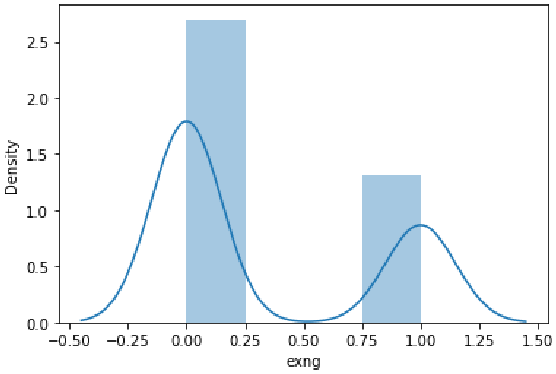


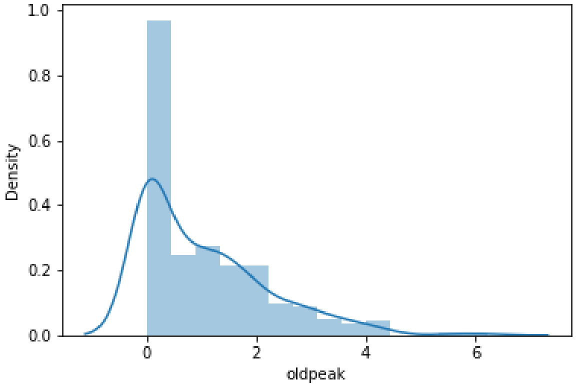
0.000
0.002
0.004
0.006
0.008
Density
100
200
300
400
500
600
chol

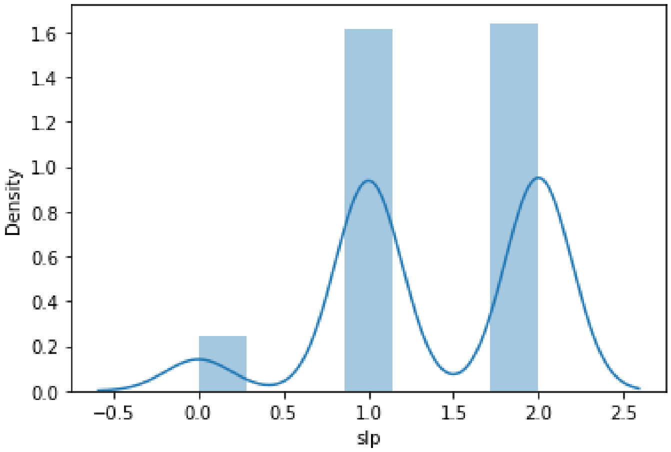

Density
14
12
10
8
6
4
2
0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
fbs

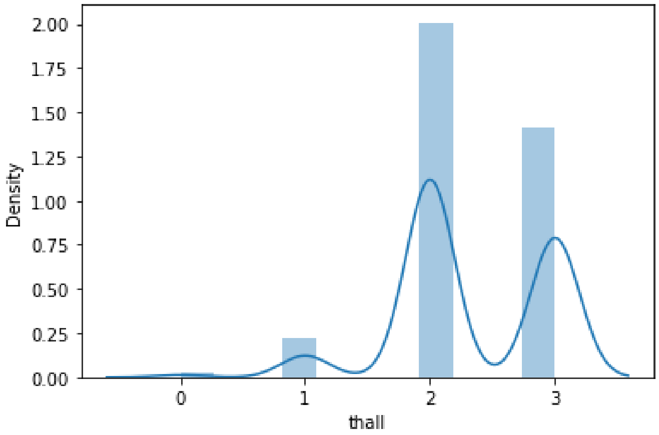

Density
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
restecg

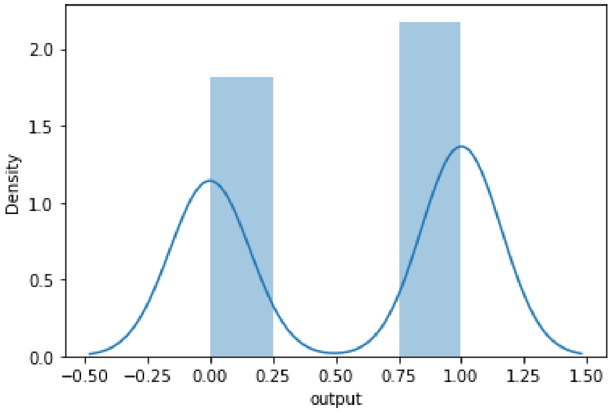

Density
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
50
75
100
125
150
175
200
225
thalachh


Density
2.5
2.0
1.5
1.0
0.5
0.0
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50
exng


Density
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
oldpeak

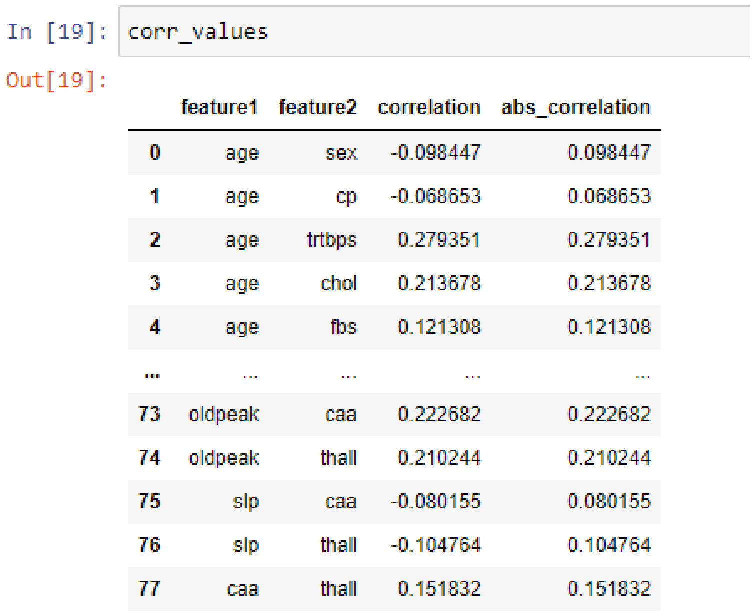

Density
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
slp





DATA CLEANING & ANALYSIS

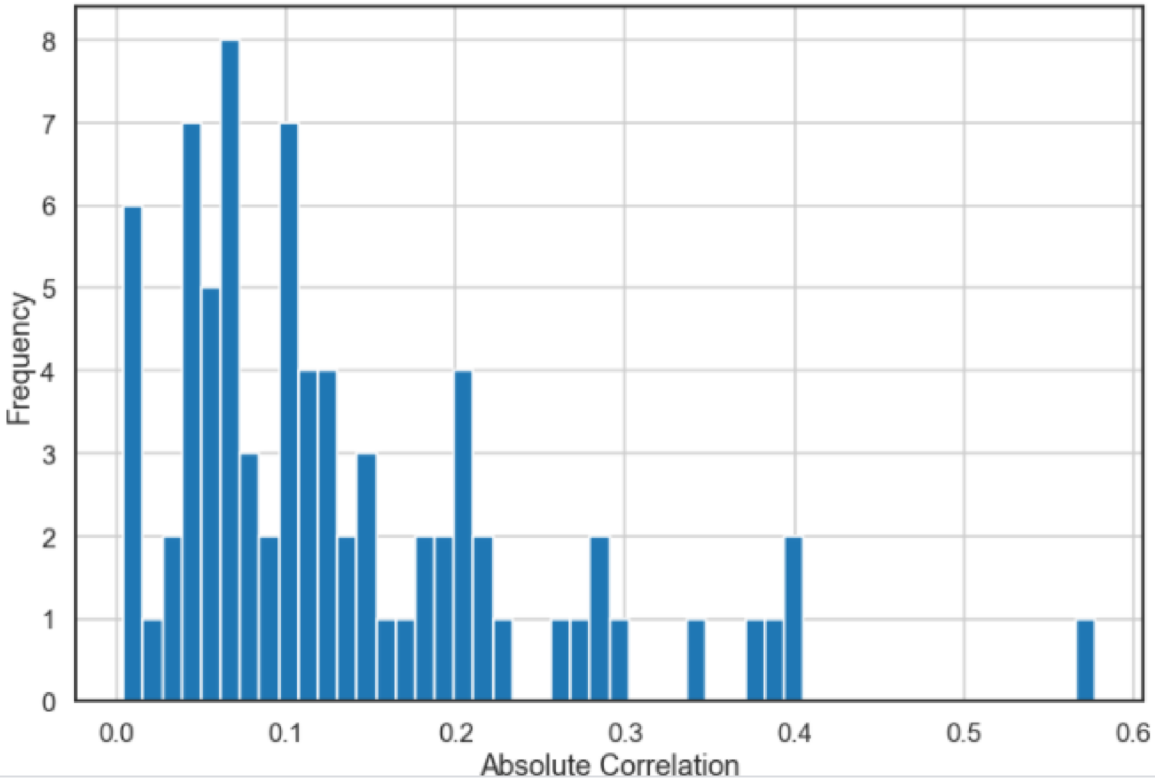
Finding correlation between different features

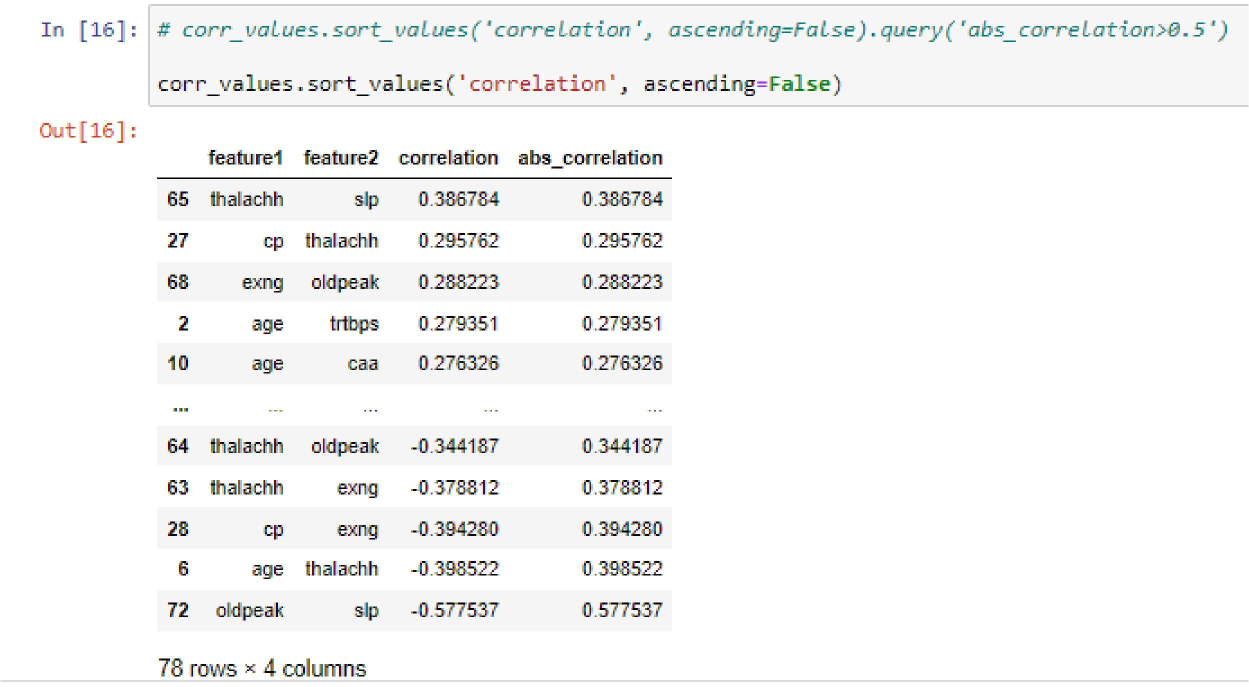


```
In [19]: corr_values
```

Out[19]:

	feature1	feature2	correlation	abs_correlation
0	age	sex	-0.098447	0.098447
1	age	cp	-0.068653	0.068653
2	age	trtbps	0.279351	0.279351
3	age	chol	0.213678	0.213678
4	age	fbs	0.121308	0.121308
...	...	...	...	...
73	oldpeak	caa	0.222682	0.222682
74	oldpeak	thall	0.210244	0.210244
75	slp	caa	-0.080155	0.080155
76	slp	thall	-0.104764	0.104764
77	caa	thall	0.151832	0.151832





```
In [16]: # corr_values.sort_values('correlation', ascending=False).query('abs_correlation>0.5')

         corr_values.sort_values('correlation', ascending=False)
```

Out[16]:

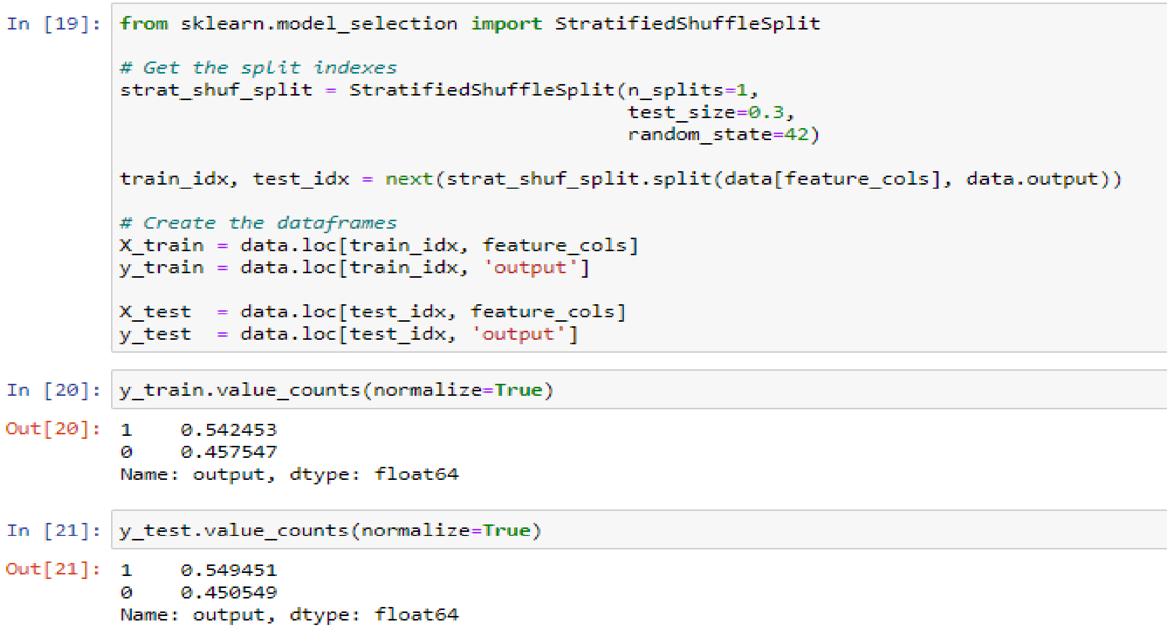
	feature1	feature2	correlation	abs_correlation
65	thalachh	slp	0.386784	0.386784
27	cp	thalachh	0.295762	0.295762
68	exng	oldpeak	0.288223	0.288223
2	age	trtbps	0.279351	0.279351
10	age	caa	0.276326	0.276326
...	...	...	...	...
64	thalachh	oldpeak	-0.344187	0.344187
63	thalachh	exng	-0.378812	0.378812
28	cp	exng	-0.394280	0.394280
6	age	thalachh	-0.398522	0.398522
72	oldpeak	slp	-0.577537	0.577537

78 rows × 4 columns

Splitting data into test and split data

The X and Y data set has been made from data frame

The target variable is the output (chances of heart attack) which we will be predicting in our Analysis

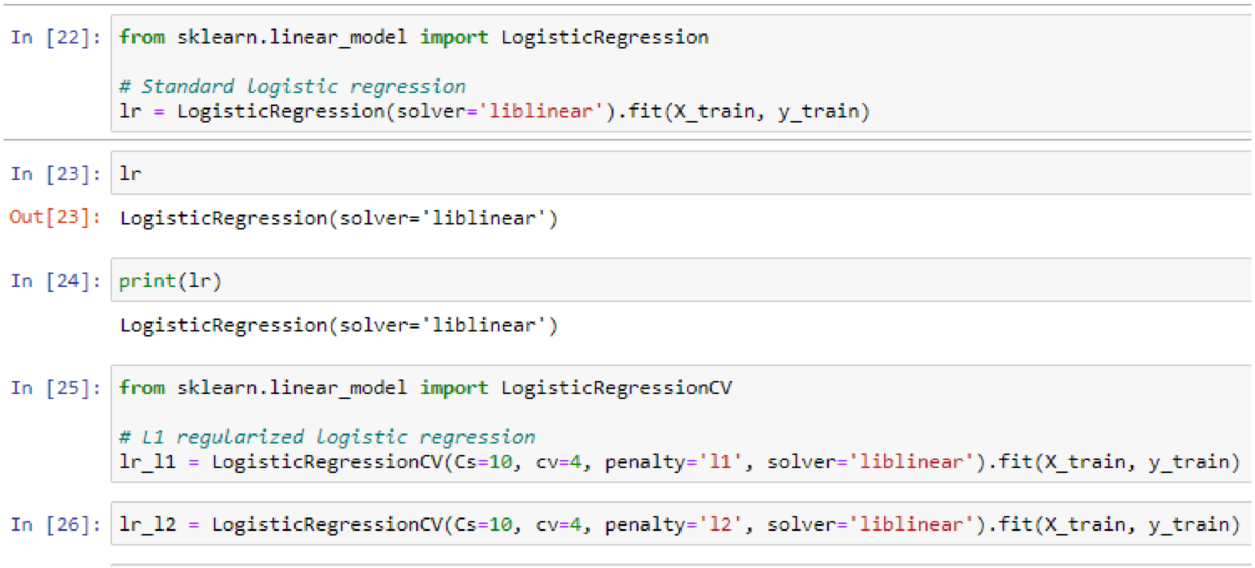


```
In [19]: from sklearn.model_selection import StratifiedShuffleSplit

         # Get the split indexes
         strat_shuf_split = StratifiedShuffleSplit(n_splits=1,
                                                   test_size=0.3,
                                                   random_state=42)

         train_idx, test_idx = next(strat_shuf_split.split(data[feature_cols], data.output))

         # Create the dataframes
         X_train = data.loc[train_idx, feature_cols]
         y_train = data.loc[train_idx, 'output']

         X_test  = data.loc[test_idx, feature_cols]
         y_test  = data.loc[test_idx, 'output']

In [20]: y_train.value_counts(normalize=True)

Out[20]: 1    0.542453
         0    0.457547
         Name: output, dtype: float64

In [21]: y_test.value_counts(normalize=True)

Out[21]: 1    0.549451
         0    0.450549
         Name: output, dtype: float64
```

DATA ANALYSIS & USE OF DIFFERENT MODELS:

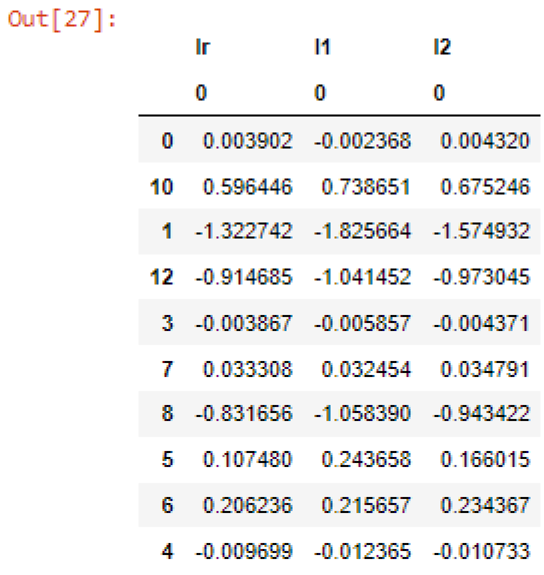
LOGISTIC REGRESSION

Initially we use logistic Regression and its different models are used to find output



```
In [22]: from sklearn.linear_model import LogisticRegression

         # Standard logistic regression
         lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)

In [23]: lr

Out[23]: LogisticRegression(solver='liblinear')

In [24]: print(lr)

         LogisticRegression(solver='liblinear')

In [25]: from sklearn.linear_model import LogisticRegressionCV

         # L1 regularized logistic regression
         lr_l1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear').fit(X_train, y_train)

In [26]: lr_l2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='liblinear').fit(X_train, y_train)
```

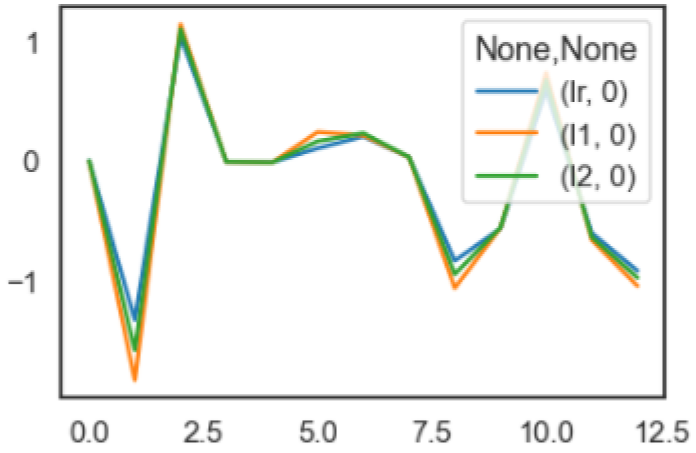
The coeffiects of the models are found



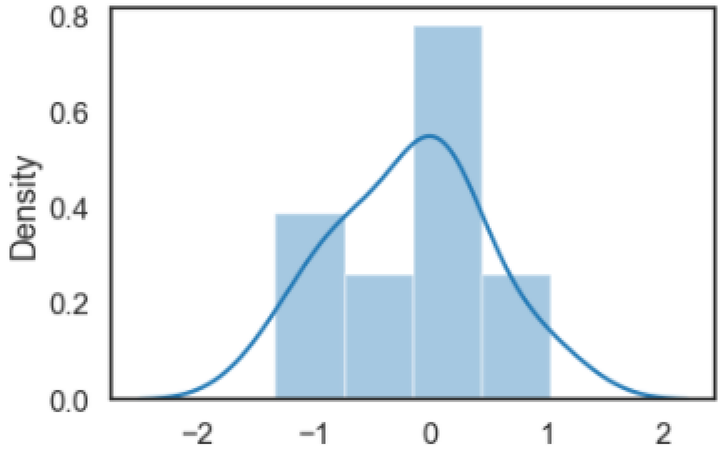
Out[27]:

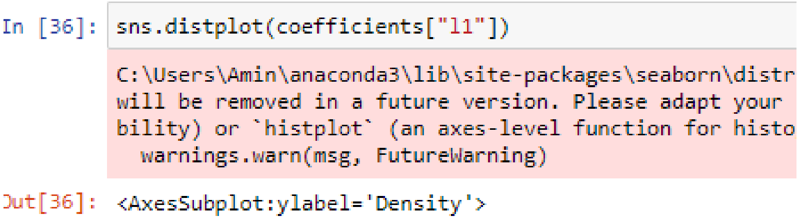
	lr	l1	l2
	0	0	0
0	0.003902	-0.002368	0.004320
10	0.596446	0.738651	0.675246
1	-1.322742	-1.825664	-1.574932
12	-0.914685	-1.041452	-0.973045
3	-0.003867	-0.005857	-0.004371
7	0.033308	0.032454	0.034791
8	-0.831656	-1.058390	-0.943422
5	0.107480	0.243658	0.166015
6	0.206236	0.215657	0.234367
4	-0.009699	-0.012365	-0.010733

```
In [34]: coefficients.plot()

Out[34]: <AxesSubplot:>
```

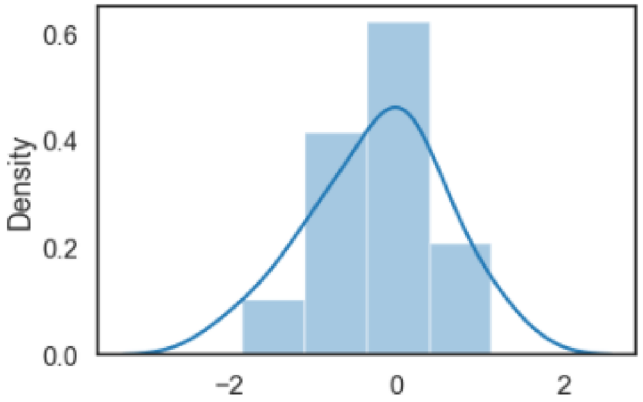


```
In [35]: sns.distplot(coefficients["lr"])

C:\Users\Amin\anaconda3\lib\site-packages\seaborn\distri
will be removed in a future version. Please adapt your c
bility) or `histplot` (an axes-level function for histog
  warnings.warn(msg, FutureWarning)

Out[35]: <AxesSubplot:ylabel='Density'>
```

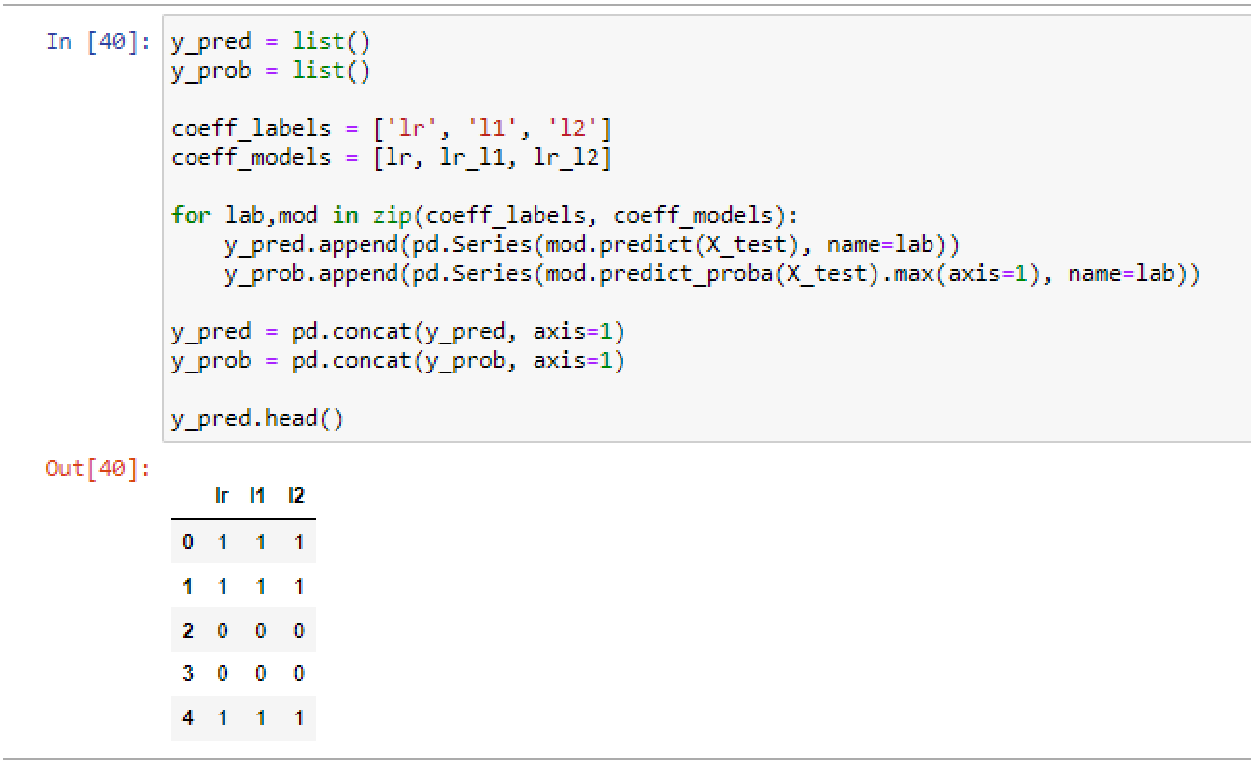




```
In [36]: sns.distplot(coefficients["l1"])

         C:\Users\Amin\anaconda3\lib\site-packages\seaborn\distr
         will be removed in a future version. Please adapt your
         bility) or `histplot` (an axes-level function for histo
           warnings.warn(msg, FutureWarning)

Out[36]: <AxesSubplot:ylabel='Density'>
```

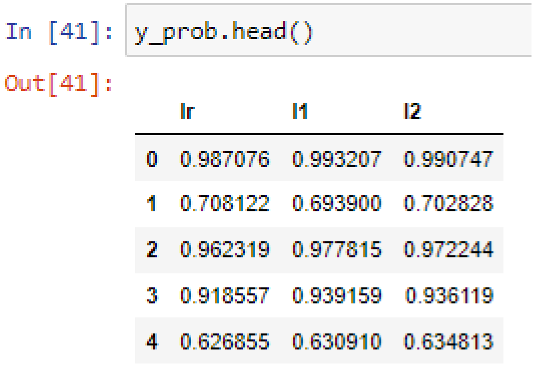


The predictions are:



```
In [40]: y_pred = list()
         y_prob = list()

         coeff_labels = ['lr', 'l1', 'l2']
         coeff_models = [lr, lr_l1, lr_l2]

         for lab,mod in zip(coeff_labels, coeff_models):
             y_pred.append(pd.Series(mod.predict(X_test), name=lab))
             y_prob.append(pd.Series(mod.predict_proba(X_test).max(axis=1), name=lab))

         y_pred = pd.concat(y_pred, axis=1)
         y_prob = pd.concat(y_prob, axis=1)

         y_pred.head()
```

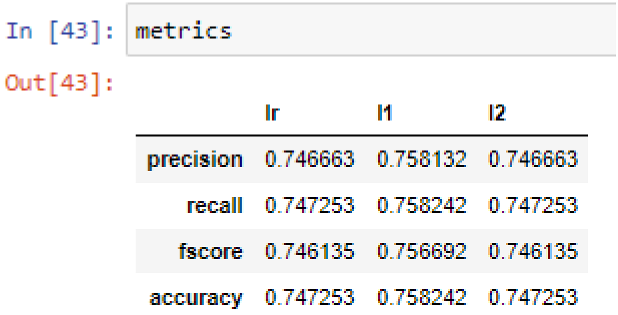
```
Out[40]:
            lr  l1  l2
         0   1   1   1
         1   1   1   1
         2   0   0   0
         3   0   0   0
         4   1   1   1
```



```
In [41]: y_prob.head()
Out[41]:
```

	lr	l1	l2
0	0.987076	0.993207	0.990747
1	0.708122	0.693900	0.702828
2	0.962319	0.977815	0.972244
3	0.918557	0.939159	0.936119
4	0.626855	0.630910	0.634813

Finding the errors of the models:

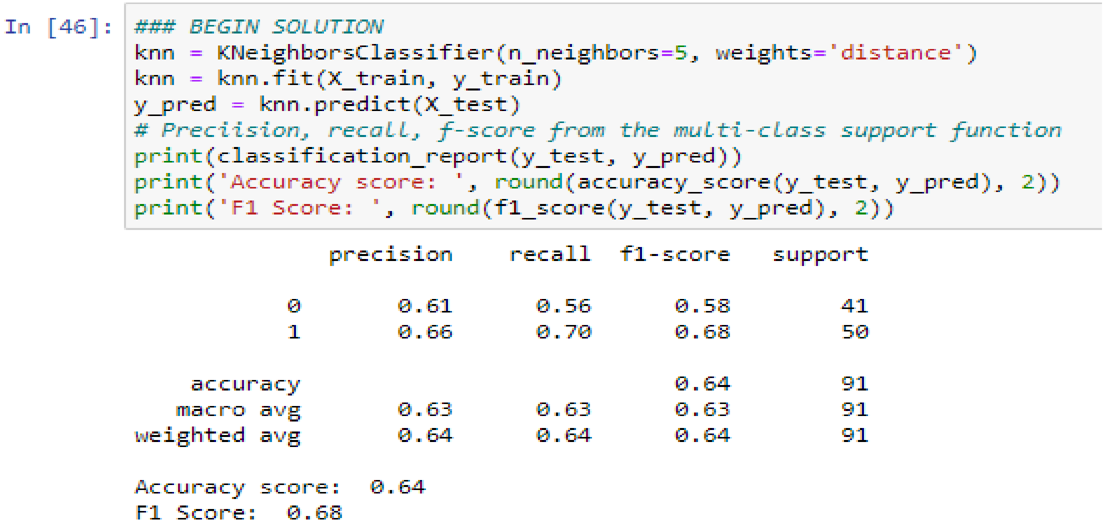


```
In [43]: metrics
Out[43]:
```

	lr	l1	l2
precision	0.746663	0.758132	0.746663
recall	0.747253	0.758242	0.747253
fscore	0.746135	0.756692	0.746135
accuracy	0.747253	0.758242	0.747253

KNN MODEL:

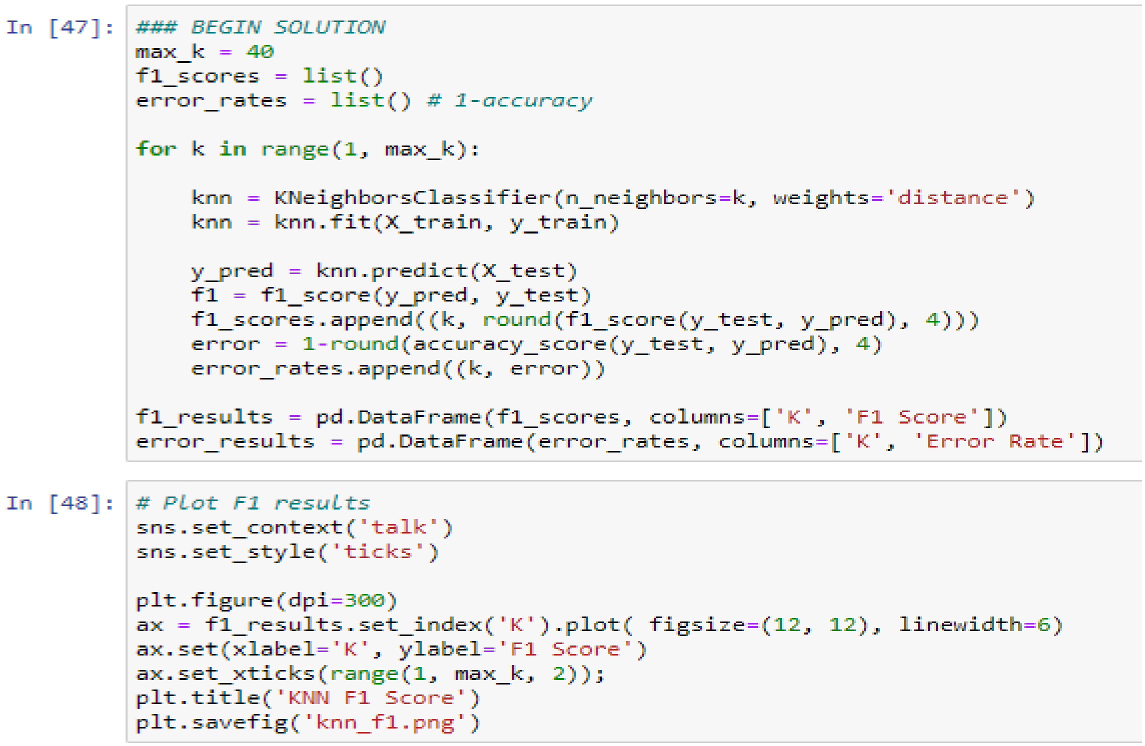
Initially for



```
In [46]: ### BEGIN SOLUTION
         knn = KNeighborsClassifier(n_neighbors=5, weights='distance')
         knn = knn.fit(X_train, y_train)
         y_pred = knn.predict(X_test)
         # Precision, recall, f-score from the multi-class support function
         print(classification_report(y_test, y_pred))
         print('Accuracy score: ', round(accuracy_score(y_test, y_pred), 2))
         print('F1 Score: ', round(f1_score(y_test, y_pred), 2))
```

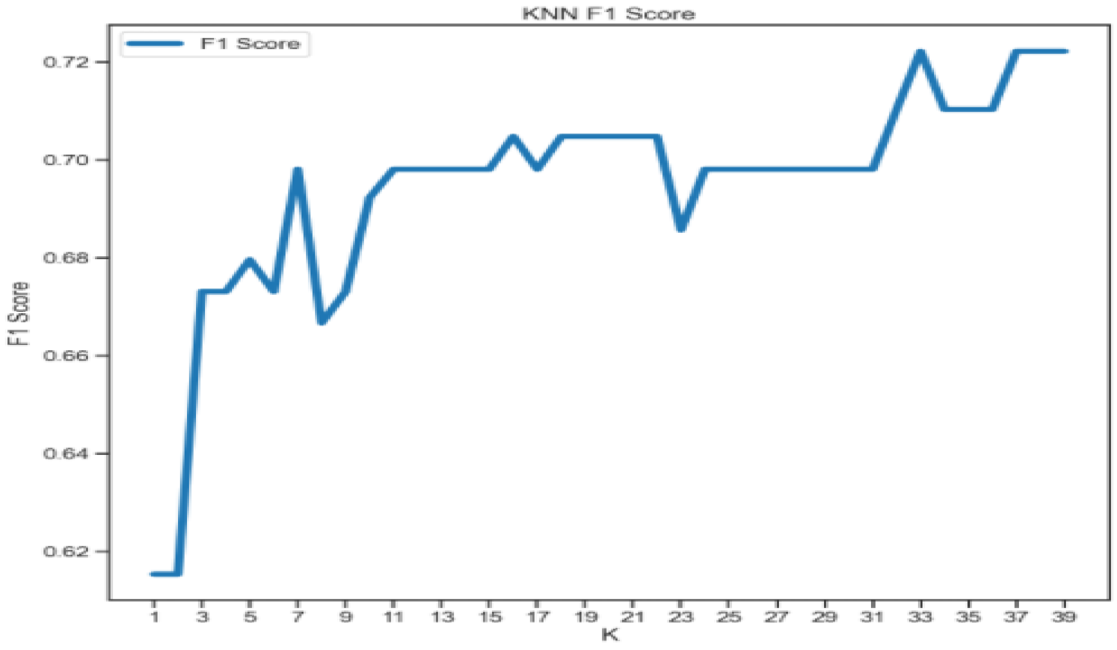
```
              precision    recall  f1-score   support

           0       0.61      0.56      0.58        41
           1       0.66      0.70      0.68        50

    accuracy                           0.64        91
   macro avg       0.63      0.63      0.63        91
weighted avg       0.64      0.64      0.64        91

Accuracy score:  0.64
F1 Score:  0.68
```

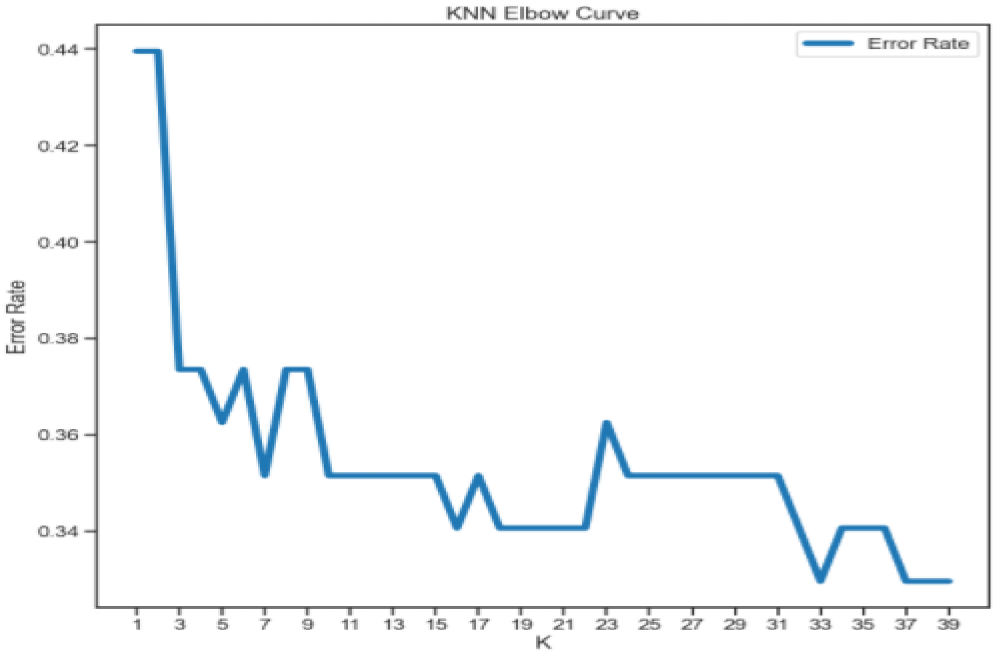
Finding mode for best n



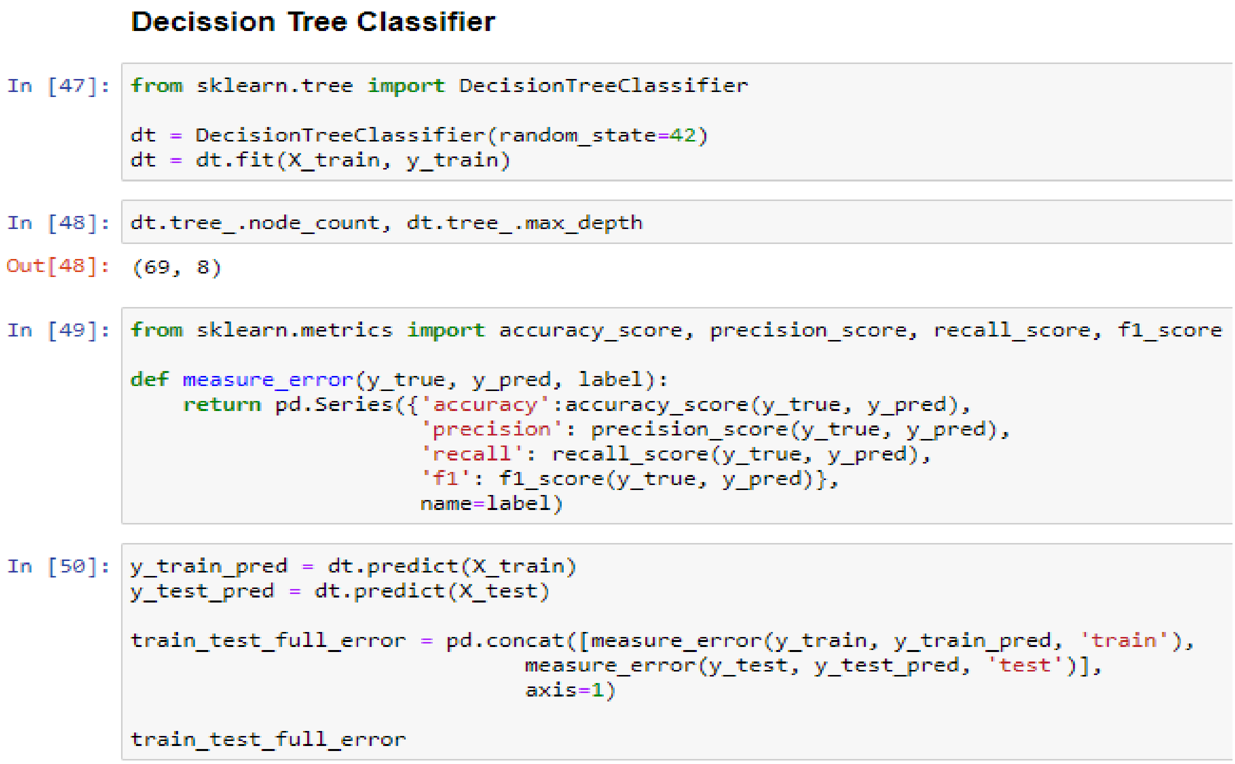
```
In [47]: ### BEGIN SOLUTION
         max_k = 40
         f1_scores = list()
         error_rates = list() # 1-accuracy

         for k in range(1, max_k):

             knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
             knn = knn.fit(X_train, y_train)

             y_pred = knn.predict(X_test)
             f1 = f1_score(y_pred, y_test)
             f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
             error = 1-round(accuracy_score(y_test, y_pred), 4)
             error_rates.append((k, error))

         f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
         error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
```

```
In [48]: # Plot F1 results
         sns.set_context('talk')
         sns.set_style('ticks')

         plt.figure(dpi=300)
         ax = f1_results.set_index('K').plot( figsize=(12, 12), linewidth=6)
         ax.set(xlabel='K', ylabel='F1 Score')
         ax.set_xticks(range(1, max_k, 2));
         plt.title('KNN F1 Score')
         plt.savefig('knn_f1.png')
```





Decision Tree Classifier:



Decission Tree Classifier

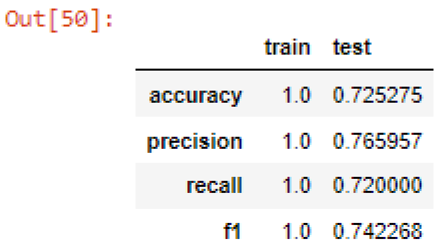
In [47]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train, y_train)
```

In [48]:
```
dt.tree_.node_count, dt.tree_.max_depth
```

Out[48]:
```
(69, 8)
```

In [49]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

def measure_error(y_true, y_pred, label):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=label)
```

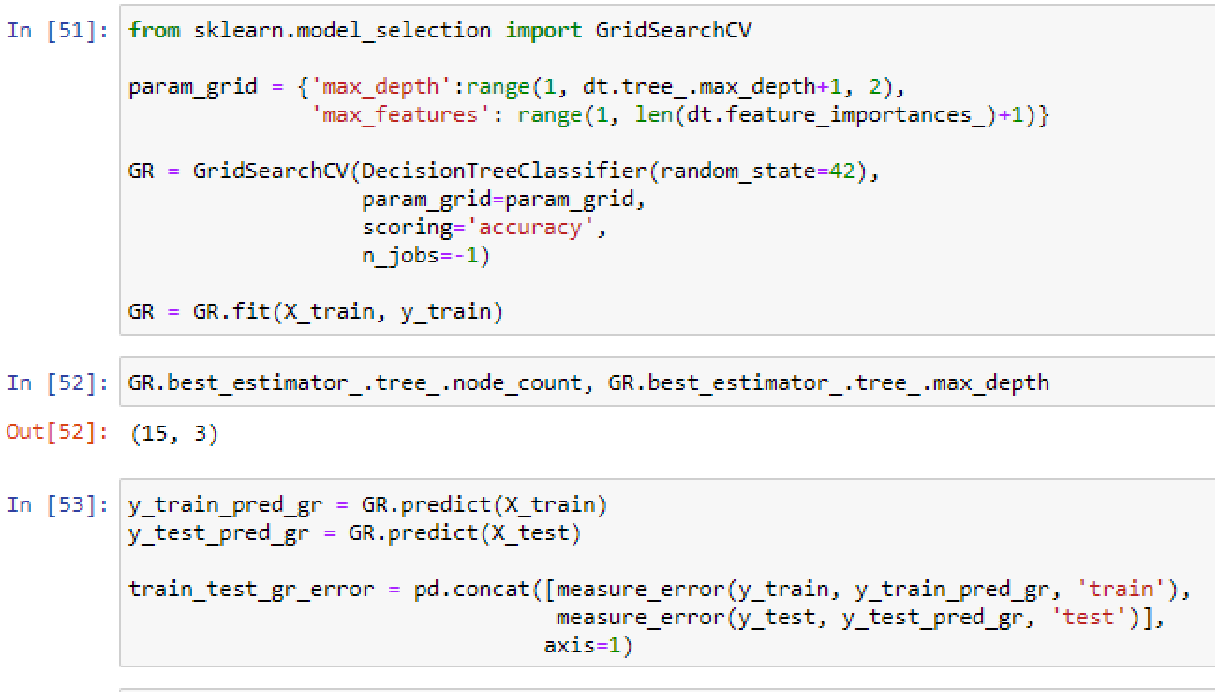
In [50]:
```
y_train_pred = dt.predict(X_train)
y_test_pred = dt.predict(X_test)

train_test_full_error = pd.concat([measure_error(y_train, y_train_pred, 'train'),
                              measure_error(y_test, y_test_pred, 'test')],
                              axis=1)

train_test_full_error
```



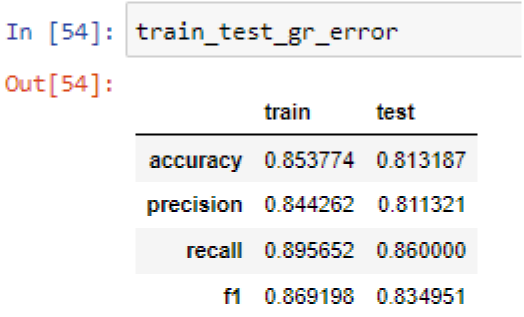
Out[50]:

	train	test
accuracy	1.0	0.725275
precision	1.0	0.765957
recall	1.0	0.720000
f1	1.0	0.742268

Finding the best fitter



```
In [51]: from sklearn.model_selection import GridSearchCV

         param_grid = {'max_depth':range(1, dt.tree_.max_depth+1, 2),
                       'max_features': range(1, len(dt.feature_importances_)+1)}

         GR = GridSearchCV(DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           scoring='accuracy',
                           n_jobs=-1)

         GR = GR.fit(X_train, y_train)

In [52]: GR.best_estimator_.tree_.node_count, GR.best_estimator_.tree_.max_depth

Out[52]: (15, 3)

In [53]: y_train_pred_gr = GR.predict(X_train)
         y_test_pred_gr = GR.predict(X_test)

         train_test_gr_error = pd.concat([measure_error(y_train, y_train_pred_gr, 'train'),
                                          measure_error(y_test, y_test_pred_gr, 'test')],
                                         axis=1)
```

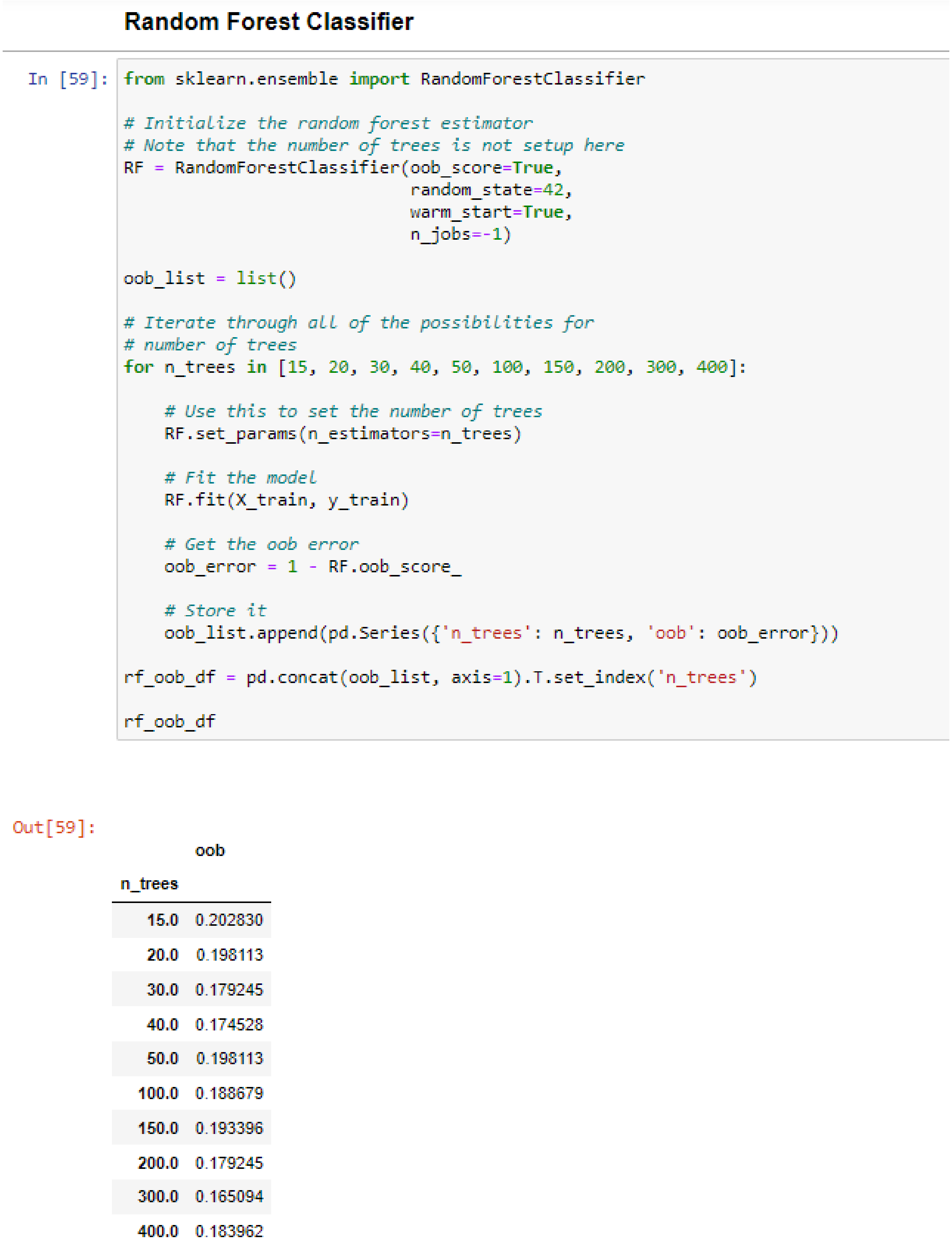


```
In [54]: train_test_gr_error
```

Out[54]:

	train	test
accuracy	0.853774	0.813187
precision	0.844262	0.811321
recall	0.895652	0.860000
f1	0.869198	0.834951

RANDOM FOREST CLASSIFIER



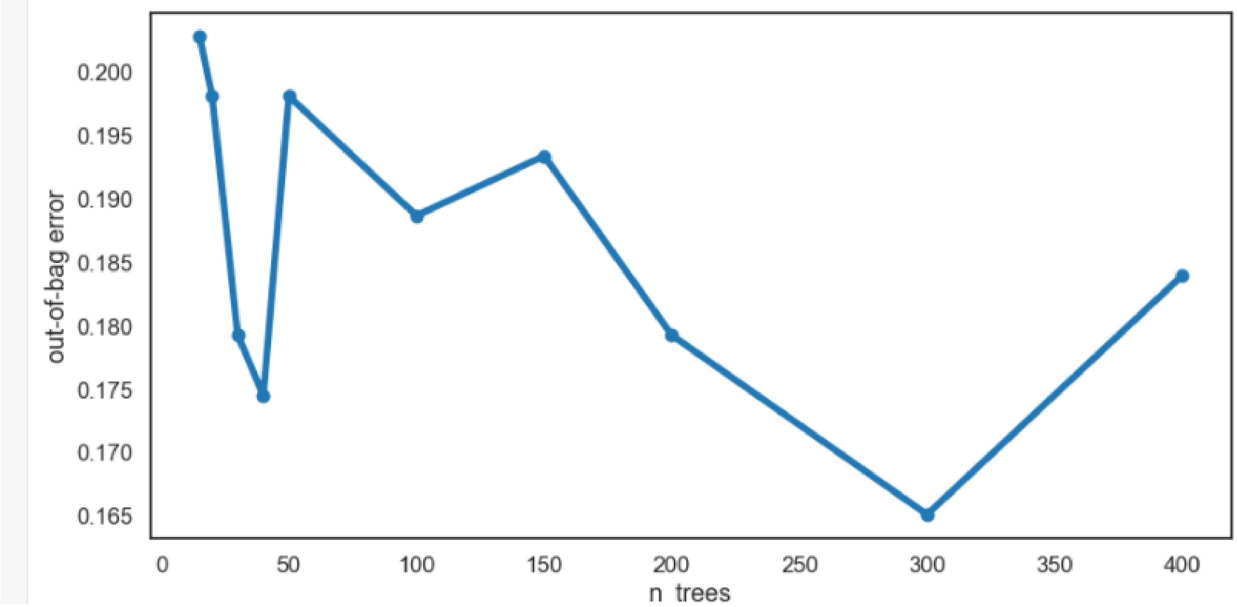
Random Forest Classifier

In [59]:

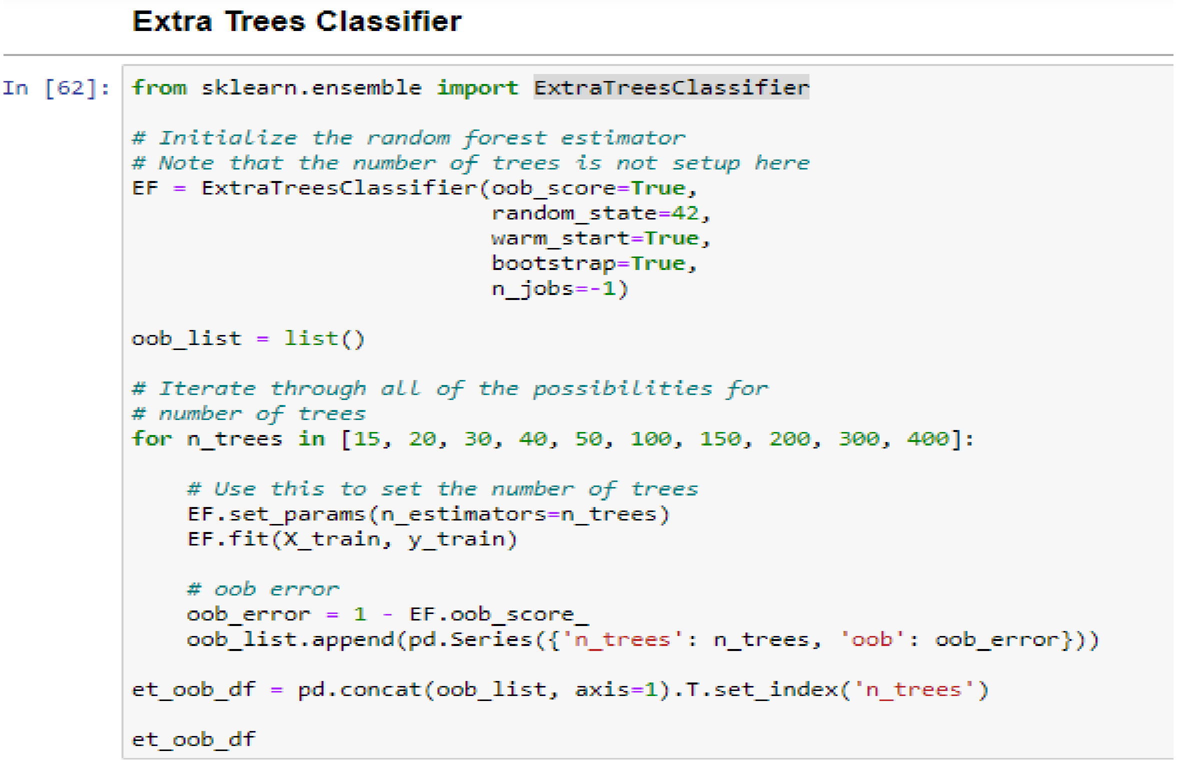
```
from sklearn.ensemble import RandomForestClassifier

# Initialize the random forest estimator
# Note that the number of trees is not setup here
RF = RandomForestClassifier(oob_score=True,
                            random_state=42,
                            warm_start=True,
                            n_jobs=-1)

oob_list = list()

# Iterate through all of the possibilities for
# number of trees
for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

    # Use this to set the number of trees
    RF.set_params(n_estimators=n_trees)

    # Fit the model
    RF.fit(X_train, y_train)

    # Get the oob error
    oob_error = 1 - RF.oob_score_

    # Store it
    oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

rf_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')

rf_oob_df
```

Out[59]:

	oob
n_trees	
15.0	0.202830
20.0	0.198113
30.0	0.179245
40.0	0.174528
50.0	0.198113
100.0	0.188679
150.0	0.193396
200.0	0.179245
300.0	0.165094
400.0	0.183962

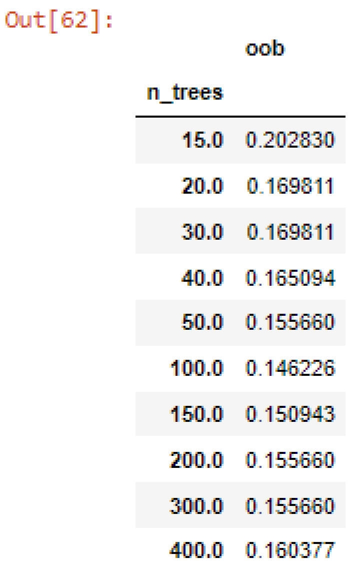


EXTRA TREE CLASSIFIER



Extra Trees Classifier

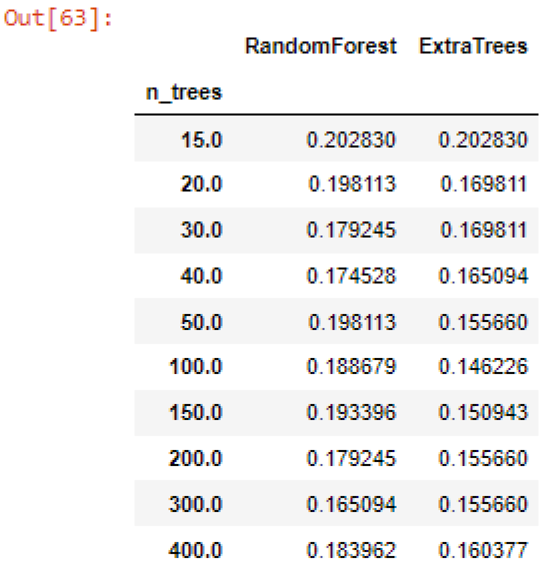
```
In [62]: from sklearn.ensemble import ExtraTreesClassifier

         # Initialize the random forest estimator
         # Note that the number of trees is not setup here
         EF = ExtraTreesClassifier(oob_score=True,
                                   random_state=42,
                                   warm_start=True,
                                   bootstrap=True,
                                   n_jobs=-1)

         oob_list = list()

         # Iterate through all of the possibilities for
         # number of trees
         for n_trees in [15, 20, 30, 40, 50, 100, 150, 200, 300, 400]:

             # Use this to set the number of trees
             EF.set_params(n_estimators=n_trees)
             EF.fit(X_train, y_train)

             # oob error
             oob_error = 1 - EF.oob_score_
             oob_list.append(pd.Series({'n_trees': n_trees, 'oob': oob_error}))

         et_oob_df = pd.concat(oob_list, axis=1).T.set_index('n_trees')

         et_oob_df
```



Out[62]:

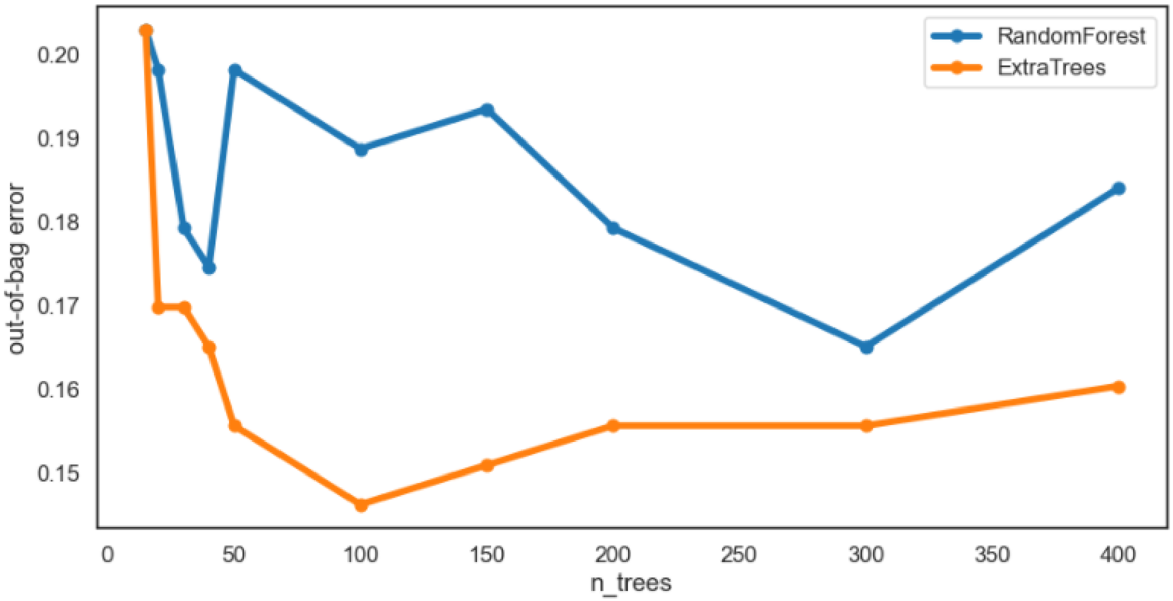
n_trees	oob
15.0	0.202830
20.0	0.169811
30.0	0.169811
40.0	0.165094
50.0	0.155660
100.0	0.146226
150.0	0.150943
200.0	0.155660
300.0	0.155660
400.0	0.160377

OUT OF BAG ERRORS

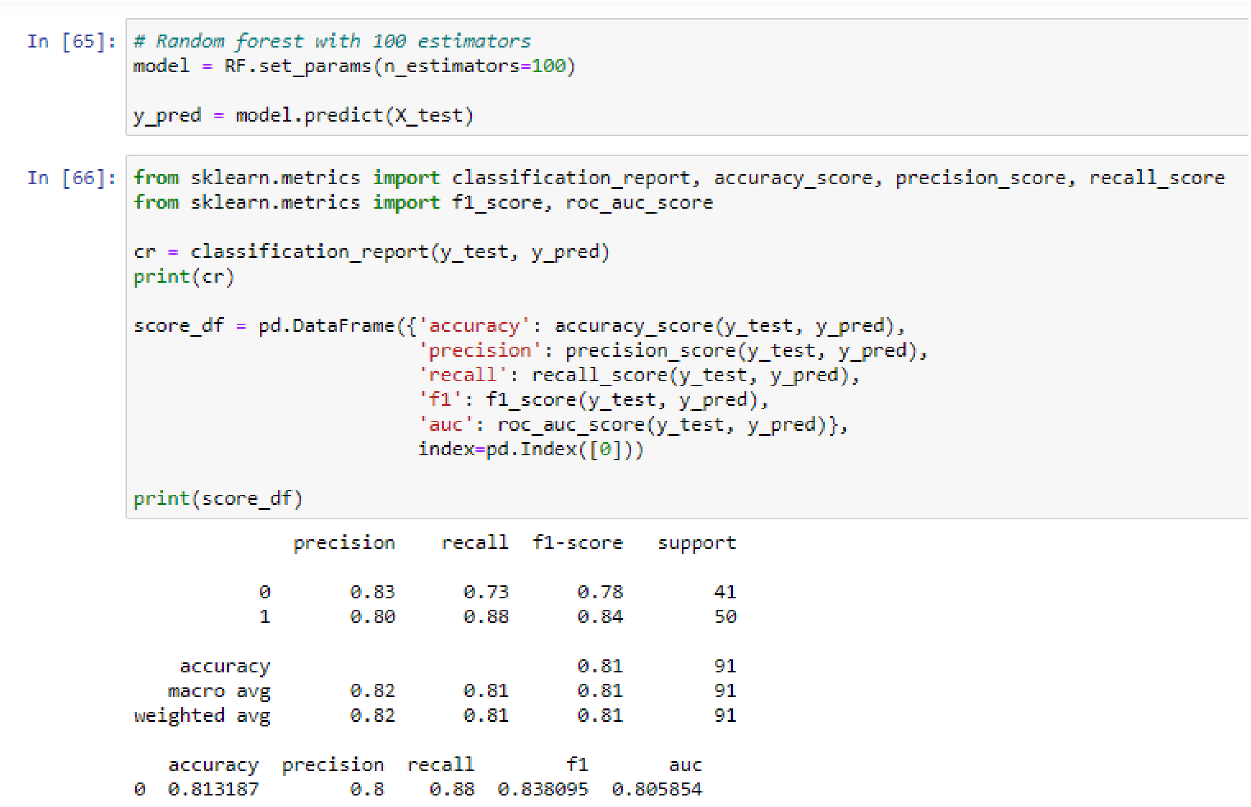


Out[63]:

n_trees	RandomForest	ExtraTrees
15.0	0.202830	0.202830
20.0	0.198113	0.169811
30.0	0.179245	0.169811
40.0	0.174528	0.165094
50.0	0.198113	0.155660
100.0	0.188679	0.146226
150.0	0.193396	0.150943
200.0	0.179245	0.155660
300.0	0.165094	0.155660
400.0	0.183962	0.160377



No calculation random forrest for n=100



In [65]:

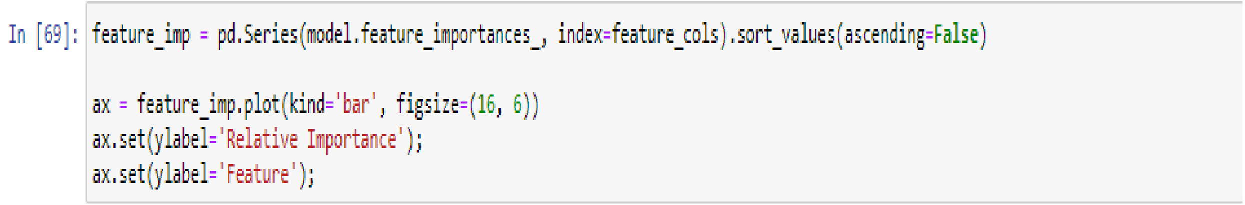
```
# Random forest with 100 estimators
model = RF.set_params(n_estimators=100)

y_pred = model.predict(X_test)
```

In [66]:

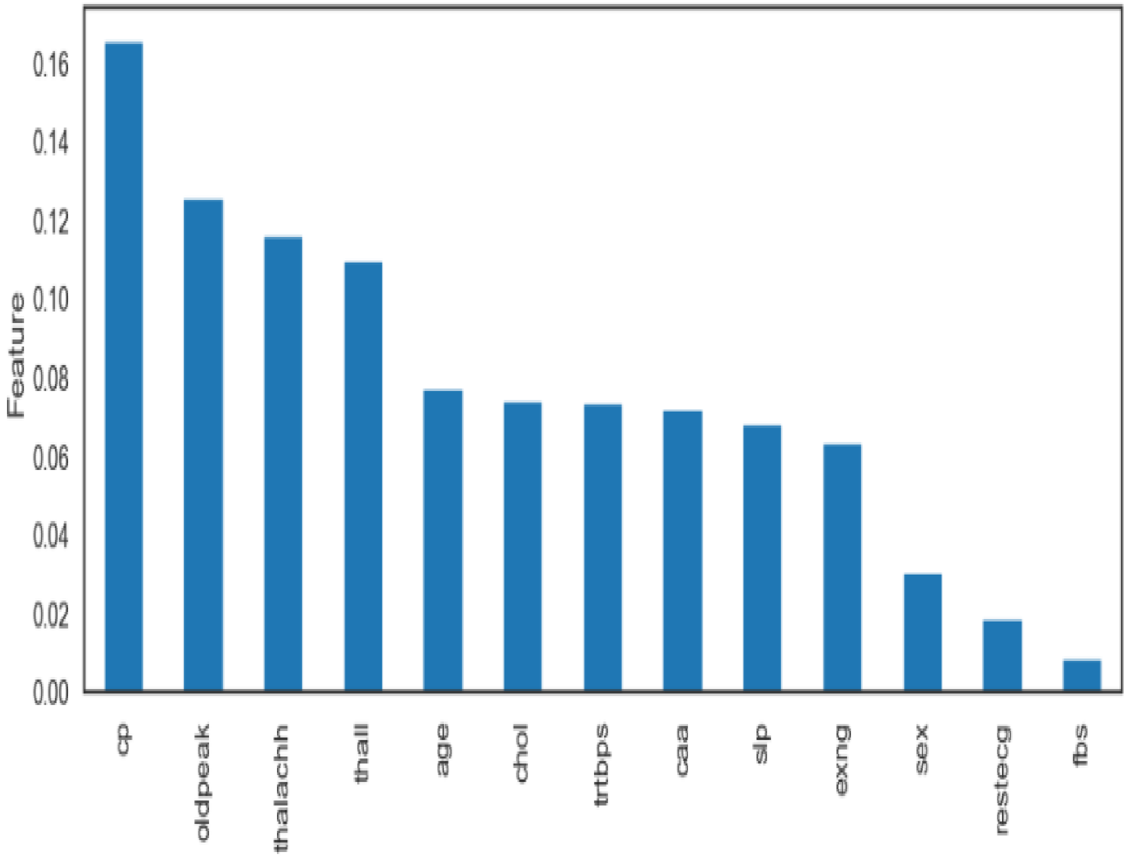
```
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score
from sklearn.metrics import f1_score, roc_auc_score

cr = classification_report(y_test, y_pred)
print(cr)

score_df = pd.DataFrame({'accuracy': accuracy_score(y_test, y_pred),
                         'precision': precision_score(y_test, y_pred),
                         'recall': recall_score(y_test, y_pred),
                         'f1': f1_score(y_test, y_pred),
                         'auc': roc_auc_score(y_test, y_pred)},
                         index=pd.Index([0]))

print(score_df)
```

```
              precision    recall  f1-score   support

           0       0.83      0.73      0.78        41
           1       0.80      0.88      0.84        50

    accuracy                           0.81        91
   macro avg       0.82      0.81      0.81        91
weighted avg       0.82      0.81      0.81        91

   accuracy  precision  recall        f1       auc
0  0.813187        0.8    0.88  0.838095  0.805854
```

Important Feature of the data set



```
In [69]: feature_imp = pd.Series(model.feature_importances_, index=feature_cols).sort_values(ascending=False)

         ax = feature_imp.plot(kind='bar', figsize=(16, 6))
         ax.set(ylabel='Relative Importance');
         ax.set(ylabel='Feature');
```



RESULTS:

DISCUSSION:

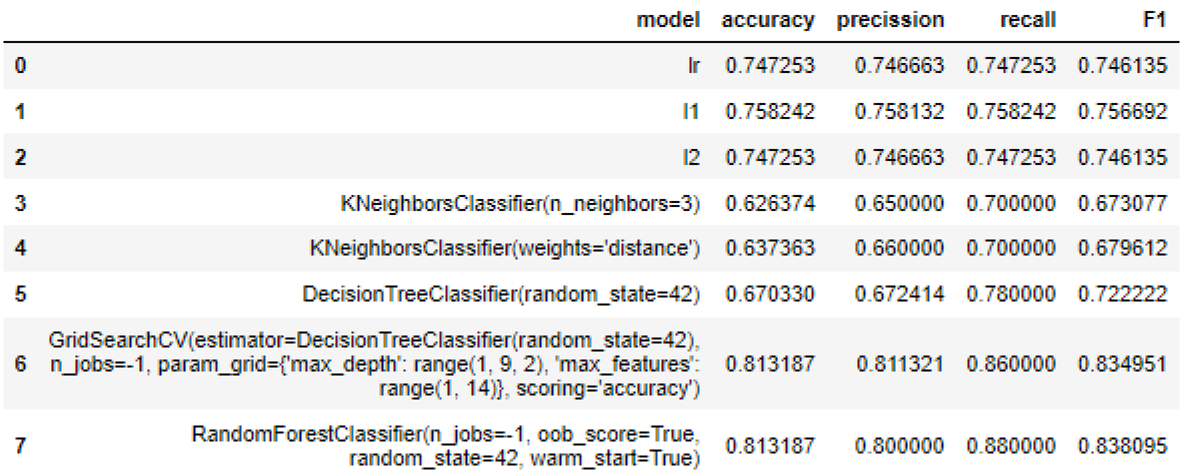
In this project the errors from different models was calculated:

- Logistic Regression
- KNN
- Decission Tree
- Random Forrest

All the models were studied to find which is best for our data set

CONCLUSION

After all the analysis we find the following table from which we can conclude that Random forest model is best for our data



	model	accuracy	precission	recall	F1
0	lr	0.747253	0.746663	0.747253	0.746135
1	l1	0.758242	0.758132	0.758242	0.756692
2	l2	0.747253	0.746663	0.747253	0.746135
3	KNeighborsClassifier(n_neighbors=3)	0.626374	0.650000	0.700000	0.673077
4	KNeighborsClassifier(weights='distance')	0.637363	0.660000	0.700000	0.679612
5	DecisionTreeClassifier(random_state=42)	0.670330	0.672414	0.780000	0.722222
6	GridSearchCV(estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1, param_grid={'max_depth': range(1, 9, 2), 'max_features': range(1, 14)}, scoring='accuracy')	0.813187	0.811321	0.860000	0.834951
7	RandomForestClassifier(n_jobs=-1, oob_score=True, random_state=42, warm_start=True)	0.813187	0.800000	0.880000	0.838095

The feature which had the most effect on the output prediction is CP Chest Pain as we found out in our analysis.

SUGGESTIONS

The dataset can include more features but it should have more data (rows) for better training of models and larger train set can be used