Decision Tree & Random Forest V2

November 21, 2021

Replace BMI, BP, ST with median

[1]:
```
import numpy as np # Import numpy for data preprocessing
import pandas as pd # Import pandas for data frame read
import matplotlib.pyplot as plt # Import matplotlib for data visualisation
import seaborn as sns # Import seaborn for data visualisation
import plotly.express as px # Import plotly for data visualisation
from sklearn.model_selection import train_test_split # Import train_test_split␣
↪for data split
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree␣
↪Classifier
from sklearn.ensemble import RandomForestClassifier # Import Random Forest␣
↪Classifier
from sklearn.model_selection import train_test_split # Import train_test_split␣
↪function
from sklearn import metrics #Import scikit-learn metrics module for accuracy␣
↪calculation
from sklearn import tree # Import export_graphviz for visualizing Decision Trees
```

0.1 Data read

[2]:
```
df = pd.read_csv("data/diabetes.csv") # Data read
```

[3]:
```
df.head() # print data
```

[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[4]:
```
df.isna().sum() # check for null value
```

[4]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

[5]:
```
df.describe()
```

[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	\
count	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	
std	3.369578	31.972618	19.355807	15.952218	115.244002	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	

	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000
mean	31.992578	0.471876	33.240885	0.348958
std	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.078000	21.000000	0.000000
25%	27.300000	0.243750	24.000000	0.000000
50%	32.000000	0.372500	29.000000	0.000000
75%	36.600000	0.626250	41.000000	1.000000
max	67.100000	2.420000	81.000000	1.000000

1 Data split

[6]:
```
X = df.iloc[:,0:-1] # All features
Y = df.iloc[:,-1] # Target
```

[7]:
```
X.head()
```

[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	

```
4             0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age
0                     0.627   50
1                     0.351   31
2                     0.672   32
3                     0.167   21
4                     2.288   33
```

[8]:
```
Y.head()
```

[8]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

[9]:
```
# Data split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=1)
# x_dev, x_test, y_dev, y_test = train_test_split(x_test, y_test, test_size= 0.
 ↪5)
```

[10]:
```
print("Original data size : ", X.shape, Y.shape)
print("Train data size : ", x_train.shape, y_train.shape)
# print("Dev data size : ", x_dev.shape, y_dev.shape)
print("Test data size : ", x_test.shape, y_test.shape)
```

```
Original data size :  (768, 8) (768,)
Train data size :  (614, 8) (614,)
Test data size :  (154, 8) (154,)
```

2 Preprocessing

[11]:
```
# replace zero bmi value with it's median
print("Before BMI median : ",round(x_train.loc[:, 'BMI'].median(),1))
x_test.loc[:, 'BMI'] = x_test.loc[:, 'BMI'].replace(0, x_train.loc[:, 'BMI'].
 ↪median())
x_train.loc[:, 'BMI'] = x_train.loc[:, 'BMI'].replace(0, x_train.loc[:, 'BMI'].
 ↪median())
print("After BMI median : ",round(x_train.loc[:, 'BMI'].median(),1))
```

```
Before BMI median :  32.0
After BMI median :  32.0
```

```
/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

[12]:
```
# replace zero SkinThickness value with it's median
print("Before SkinThickness median : ",round(x_train.loc[:, 'SkinThickness'].
 ↪median(),1))
x_test.loc[:, 'SkinThickness'] = x_test.loc[:, 'SkinThickness'].replace(0,␣
 ↪x_train.loc[:, 'SkinThickness'].median())
x_train.loc[:, 'SkinThickness'] = x_train.loc[:, 'SkinThickness'].replace(0,␣
 ↪x_train.loc[:, 'SkinThickness'].median())
print("After SkinThickness median : ",round(x_train.loc[:, 'SkinThickness'].
 ↪median(),1))
```

```
Before SkinThickness median :  22.0
After SkinThickness median :  22.0

/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

[13]:
```
# replace zero BloodPressure value with it's median
print("Before BloodPressure median : ",round(x_train.loc[:, 'BloodPressure'].
 ↪median(),1))
```

```
x_test.loc[:, 'BloodPressure'] = x_test.loc[:, 'BloodPressure'].replace(0,␣
 ↪x_train.loc[:, 'BloodPressure'].median())
x_train.loc[:, 'BloodPressure'] = x_train.loc[:, 'BloodPressure'].replace(0,␣
 ↪x_train.loc[:, 'BloodPressure'].median())
print("After BloodPressure median : ",round(x_train.loc[:, 'BloodPressure'].
 ↪median(),1))
```

```
Before BloodPressure median :  72.0
After BloodPressure median :  72.0
```

```
/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

3 Decision Tree

[14]:
```
accuracy = {}
```

3.0.1 criterion="gini", splitter="best"

[15]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="gini", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[16]:
```
print(y_pred)
```

```
[0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 1 0 0 0 1 1 1 1 1 0
 0 0 1 1 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1
 0 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0
 0 0 0 1 1 0]
```

[17]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[18]:
```
accuracy["dt_gini_best"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6493506493506493
```

[19]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[72 27]
 [27 28]]
```

[20]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.73      0.73        99
           1       0.51      0.51      0.51        55

    accuracy                           0.65       154
   macro avg       0.62      0.62      0.62       154
weighted avg       0.65      0.65      0.65       154
```

3.0.2 criterion="gini", splitter="best", max_depth=8

[21]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=8)
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[22]:
```
print(y_pred)
```

```
[0 0 0 1 0 0 1 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 1 1 1 0 0 0
 0 0 1 1 0 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0
 0 0 0 1 1 0]
```

[23]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[24]:
```
accuracy["dt_gini_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7077922077922078
```

[25]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[80 19]
 [26 29]]
```

[26]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.81      0.78        99
           1       0.60      0.53      0.56        55

    accuracy                           0.71       154
   macro avg       0.68      0.67      0.67       154
weighted avg       0.70      0.71      0.70       154
```

3.0.3 criterion="entropy", splitter="best"

[27]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[28]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0
 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0
 0 1 1 1 0 0 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0 1 0 0 0 0 1 0 0 1
 0 0 1 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 0 1 0 0
 0 1 0 1 0 1]
```

[29]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[30]:
```
accuracy["dt_entropy_best"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6558441558441559
```

[31]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[73 26]
 [27 28]]
```

[32]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.73      0.74      0.73        99
           1       0.52      0.51      0.51        55

    accuracy                           0.66       154
   macro avg       0.62      0.62      0.62       154
weighted avg       0.65      0.66      0.66       154
```

3.0.4 criterion="entropy", splitter="best", max_depth=8

[33]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best", max_depth=8)
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[34]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0
 0 1 0 1 1 1]
```

[35]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[36]:
```
accuracy["dt_entropy_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7077922077922078
```

[37]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[75 24]
 [21 34]]
```

[38]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.76      0.77        99
           1       0.59      0.62      0.60        55

    accuracy                           0.71       154
   macro avg       0.68      0.69      0.69       154
weighted avg       0.71      0.71      0.71       154
```

3.0.5 criterion="entropy", splitter="random"

[39]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="random")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[40]:
```
print(y_pred)
```

```
[0 0 0 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0
 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0
 1 1 1 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 1 0 0 1 0 0 0 0 1 1 0 0 0
 0 1 0 1 0 0 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 1 0 1 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[41]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[42]:
```
accuracy["dt_entropy_random"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7077922077922078
```

[43]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[72 27]
 [18 37]]
```

[44]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.73      0.76        99
```

```
           1       0.58      0.67      0.62        55

    accuracy                           0.71       154
   macro avg       0.69      0.70      0.69       154
weighted avg       0.72      0.71      0.71       154
```

3.0.6 criterion="entropy", splitter="random", max_depth=8

[45]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="random",␣
 ↪max_depth=8)
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[46]:
```
print(y_pred)
```

```
[1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0
 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0
 0 0 0 1 0 0]
```

[47]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[48]:
```
accuracy["dt_entropy_random_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7727272727272727
```

[49]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[88 11]
 [24 31]]
```

[50]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.89      0.83        99
           1       0.74      0.56      0.64        55

    accuracy                           0.77       154
   macro avg       0.76      0.73      0.74       154
```

```
weighted avg       0.77      0.77      0.76       154
```

3.0.7 criterion="entropy", splitter="best", max_depth=3

[51]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best", max_depth=3)
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[52]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[53]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[54]:
```
accuracy["dt_entropy_best_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[55]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [20 35]]
```

[56]:
```
print(metrics.classification_report(y_test, y_pred))
```

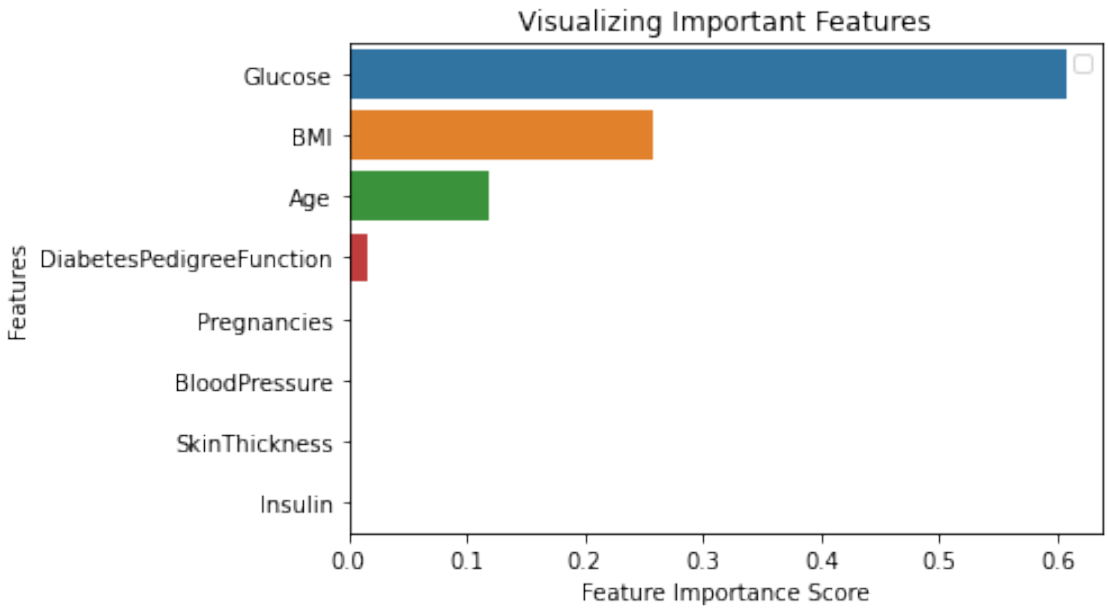
```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.64      0.69        55

    accuracy                           0.79       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.79      0.79      0.79       154
```

[57]:
```
feature_imp = pd.Series(clf.feature_importances_,index=X.columns).
↪sort_values(ascending=False)
```

```
print(feature_imp)
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
Glucose                     0.606802
BMI                         0.258369
Age                         0.118413
DiabetesPedigreeFunction    0.016416
Pregnancies                 0.000000
BloodPressure               0.000000
SkinThickness               0.000000
Insulin                     0.000000
dtype: float64
```

```
No handles with labels found to put in legend.
```



3.0.8 criterion="entropy", splitter="random", max_depth=3

[58]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="random",␣
 ↪max_depth=3)
```

```
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[59]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 1 0 0]
```

[60]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[61]:
```
accuracy["dt_entropy_random_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7077922077922078
```

[62]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[98  1]
 [44 11]]
```

[63]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.99      0.81        99
           1       0.92      0.20      0.33        55

    accuracy                           0.71       154
   macro avg       0.80      0.59      0.57       154
weighted avg       0.77      0.71      0.64       154
```

4 Accuracy visulization of Decision Tree

[64]:
```
accuracy_df_dt = pd.DataFrame(list(zip(accuracy.keys(), accuracy.values())),␣
↪columns =['Arguments', 'Accuracy'])
accuracy_df_dt
```

[64]:

	Arguments	Accuracy
0	dt_gini_best	0.649351

```
1       dt_gini_best_8  0.707792
2      dt_entropy_best  0.655844
3    dt_entropy_best_8  0.707792
4    dt_entropy_random  0.707792
5  dt_entropy_random_8  0.772727
6    dt_entropy_best_3  0.792208
7  dt_entropy_random_3  0.707792
```

[65]:
```
fig = px.bar(accuracy_df_dt, x='Arguments', y='Accuracy')
fig.show()
```

5 Random Forest

[66]:
```
accuracy_rf = {}
```

5.0.1 n_estimators = 1000, criterion='entropy'

[67]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, criterion='entropy')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[68]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0
 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0
 0 0 0 1 1 0]
```

[69]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[70]:
```
accuracy_rf["rf_entropy_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8051948051948052
```

[71]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [17 38]]
```

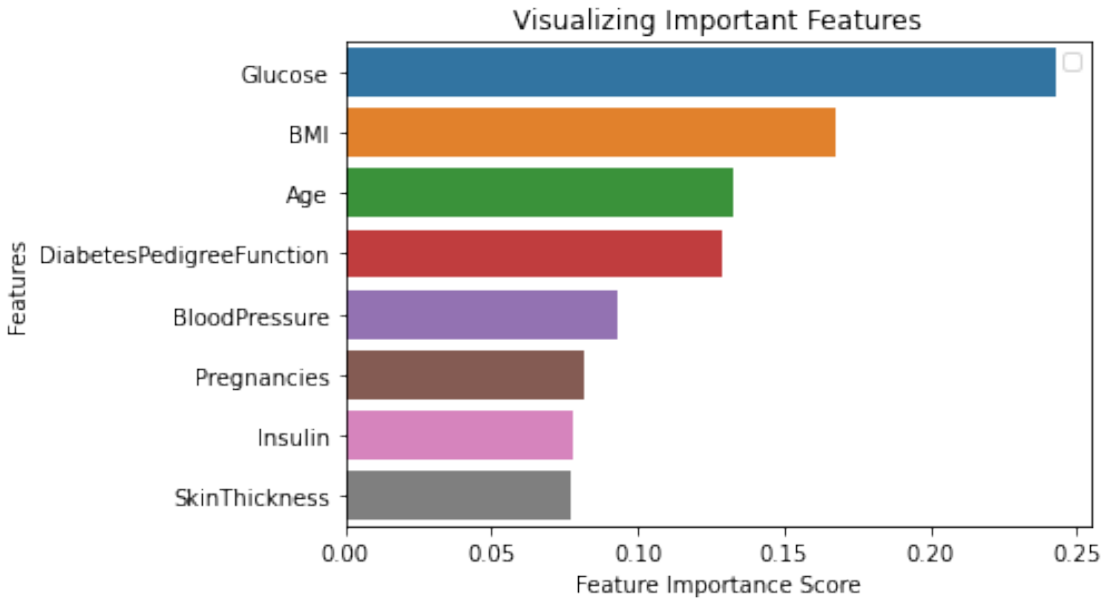
[72]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85        99
           1       0.75      0.69      0.72        55

    accuracy                           0.81       154
   macro avg       0.79      0.78      0.78       154
weighted avg       0.80      0.81      0.80       154
```

[73]:
```
feature_imp = pd.Series(rf.feature_importances_,index=X.columns).
 ↪sort_values(ascending=False)
print(feature_imp)
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```

```
No handles with labels found to put in legend.
```

```
Glucose                     0.242482
BMI                         0.167555
Age                         0.132380
DiabetesPedigreeFunction    0.128408
BloodPressure               0.092683
Pregnancies                 0.081925
Insulin                     0.077621
SkinThickness               0.076947
dtype: float64
```



5.0.2 n_estimators = 100, criterion='entropy'

[74]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, criterion='entropy')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[75]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 1 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[76]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[77]:
```
accuracy_rf["rf_entropy_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[78]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [21 34]]
```

[79]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.62      0.67        55

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.78      0.79      0.78       154
```

5.0.3 n_estimators = 1000, random_state = 42, criterion='entropy'

[80]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42,␣
 ↪criterion='entropy')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[81]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1 1 0
 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0
 0 0 0 1 1 0]
```

[82]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[83]:
```
accuracy_rf["rf_entropy_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7987012987012987
```

[84]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [18 37]]
```

[85]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85        99
           1       0.74      0.67      0.70        55

    accuracy                           0.80       154
   macro avg       0.78      0.77      0.78       154
weighted avg       0.80      0.80      0.80       154
```

5.0.4 n_estimators = 100, random_state = 42, criterion='entropy'

[86]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
↪8, criterion='entropy')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[87]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[88]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[89]:
```
accuracy_rf["rf_entropy_100_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[90]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [20 35]]
```

[91]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84        99
           1       0.73      0.64      0.68        55

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.78      0.79      0.78       154
```

5.0.5 n_estimators = 1000, random_state = 42, max_depth = 8, criterion='entropy'

[92]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth =␣
↪8, criterion='entropy')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[93]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[94]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[95]:
```
accuracy_rf["rf_entropy_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7792207792207793
```

[96]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [22 33]]
```

[97]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.88      0.84        99
           1       0.73      0.60      0.66        55

    accuracy                           0.78       154
   macro avg       0.77      0.74      0.75       154
weighted avg       0.78      0.78      0.77       154
```

5.0.6 n_estimators = 100, random_state = 42, max_depth = 8, criterion='entropy'

[98]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
 ↪8, criterion='entropy')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[99]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[100]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[101]:
```
accuracy_rf["rf_entropy_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[102]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [20 35]]
```

[103]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84        99
           1       0.73      0.64      0.68        55

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.78      0.79      0.78       154
```

5.0.7 n_estimators = 1000

[104]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000)
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[105]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[106]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[107]:
```
accuracy_rf["rf_gini_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7987012987012987
```

[108]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [19 36]]
```

[109]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        99
```

```
           1       0.75      0.65      0.70        55

    accuracy                           0.80       154
   macro avg       0.79      0.77      0.77       154
weighted avg       0.80      0.80      0.80       154
```

5.0.8 n_estimators = 100

[110]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100)
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[111]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 1 1 1 1 0
 1 1 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[112]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[113]:
```
accuracy_rf["rf_gini_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7662337662337663
```

[114]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[84 15]
 [21 34]]
```

[115]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.85      0.82        99
           1       0.69      0.62      0.65        55

    accuracy                           0.77       154
```

```
   macro avg       0.75      0.73      0.74       154
weighted avg       0.76      0.77      0.76       154
```

5.0.9 n_estimators = 1000, random_state = 42

[116]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[117]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0
 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[118]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[119]:
```
accuracy_rf["rf_gini_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[120]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [20 35]]
```

[121]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.64      0.69        55

    accuracy                           0.79       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.79      0.79      0.79       154
```

5.0.10 n_estimators = 100, random_state = 42

[122]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
↪8)
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[123]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 1 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[124]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[125]:
```
accuracy_rf["rf_gini_100_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[126]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [19 36]]
```

[127]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        99
           1       0.73      0.65      0.69        55

    accuracy                           0.79       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.79      0.79      0.79       154
```

5.0.11 n_estimators = 1000, random_state = 42, max_depth = 8

[128]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth =␣
↪8)
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[129]:
```
print(y_pred)
```

```
[1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[130]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[131]:
```
accuracy_rf["rf_gini_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[132]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [21 34]]
```

[133]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.62      0.67        55

    accuracy                           0.79       154
   macro avg       0.77      0.75      0.76       154
weighted avg       0.78      0.79      0.78       154
```

5.0.12 n_estimators = 100, random_state = 42, max_depth = 8

[134]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
↪8)
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

[135]:
```
print(y_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 0 1 0
 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 1 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[136]:
```
print(np.array(y_test))
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0
 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 0 0 0
 1 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 0 1
 0 0 0 1 0 0 1 0 1 0 1 1 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[137]:
```
accuracy_rf["rf_gini_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[138]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [19 36]]
```

[139]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        99
           1       0.73      0.65      0.69        55

    accuracy                           0.79       154
   macro avg       0.78      0.76      0.77       154
weighted avg       0.79      0.79      0.79       154
```

6 Accuracy visulization of Random Forest

[140]:
```
accuracy_df_rf = pd.DataFrame(list(zip(accuracy_rf.keys(), accuracy_rf.
↪values())), columns =['Arguments', 'Accuracy'])
accuracy_df_rf
```

[140]:

	Arguments	Accuracy
0	rf_entropy_1000	0.805195
1	rf_entropy_100	0.785714
2	rf_entropy_1000_42	0.798701
3	rf_entropy_100_42	0.785714
4	rf_entropy_1000_42_8	0.779221
5	rf_entropy_100_42_8	0.785714
6	rf_gini_1000	0.798701
7	rf_gini_100	0.766234
8	rf_gini_1000_42	0.792208
9	rf_gini_100_42	0.792208
10	rf_gini_1000_42_8	0.785714
11	rf_gini_100_42_8	0.792208

[141]:
```
fig = px.bar(accuracy_df_rf, x='Arguments', y='Accuracy')
fig.show()
```

[142]:
```
accuracy_df = pd.concat([accuracy_df_dt, accuracy_df_rf])
accuracy_df['Accuracy'] = round(accuracy_df['Accuracy'] * 100, 2)
fig = px.bar(accuracy_df, x='Arguments', y='Accuracy')
print(accuracy_df['Accuracy'].max())
fig.show()
```

```
80.52
```