Decision Tree & Random Forest V0

November 21, 2021

```python
[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

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

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

```
[3]:    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  Outcome
     0                     0.627   50        1
     1                     0.351   31        0
     2                     0.672   32        1
     3                     0.167   21        0
     4                     2.288   33        1
```

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
```

```
[6]: df.corr()
```

```
[6]:                           Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.129459       0.141282      -0.081672
Glucose                      0.129459  1.000000       0.152590       0.057328
BloodPressure                0.141282  0.152590       1.000000       0.207371
SkinThickness               -0.081672  0.057328       0.207371       1.000000
Insulin                     -0.073535  0.331357       0.088933       0.436783
BMI                          0.017683  0.221071       0.281805       0.392573
DiabetesPedigreeFunction    -0.033523  0.137337       0.041265       0.183928
Age                          0.544341  0.263514       0.239528      -0.113970
Outcome                      0.221898  0.466581       0.065068       0.074752
```

```
                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.073535  0.017683                 -0.033523
Glucose                   0.331357  0.221071                  0.137337
BloodPressure             0.088933  0.281805                  0.041265
SkinThickness             0.436783  0.392573                  0.183928
Insulin                   1.000000  0.197859                  0.185071
BMI                       0.197859  1.000000                  0.140647
DiabetesPedigreeFunction  0.185071  0.140647                  1.000000
Age                      -0.042163  0.036242                  0.033561
Outcome                   0.130548  0.292695                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.263514  0.466581
BloodPressure             0.239528  0.065068
SkinThickness            -0.113970  0.074752
Insulin                  -0.042163  0.130548
BMI                       0.036242  0.292695
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[7]: `sns.heatmap(df.corr())`

[7]: <AxesSubplot:>
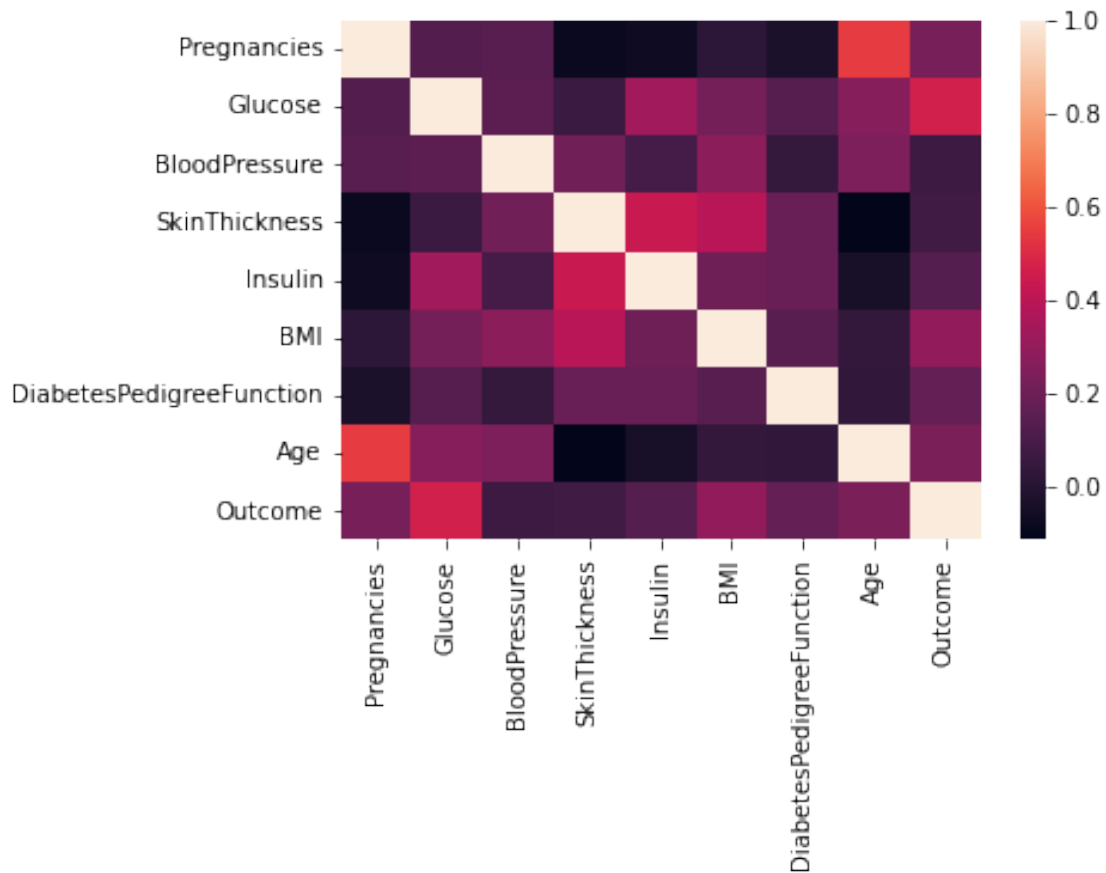
1 Data split

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

```python
[9]: X.head()
```

```
[9]:    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
```

[10]: Y.head()

[10]: 0   1
      1   0
      2   1
      3   0
      4   1
      Name: Outcome, dtype: int64

[11]:
```python
# 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)
```

[12]:
```python
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 Decision Tree

[13]: accuracy = {}

2.0.1 criterion="gini", splitter="best"

[14]:
```python
# Define and build model
clf = DecisionTreeClassifier(criterion="gini", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[15]: print(y_pred)

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

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

```
[0 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]
```

[17]:
```python
accuracy["dt_gini_best"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6688311688311688
```

[18]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[73 26]
 [25 30]]
```

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

```
              precision    recall  f1-score   support

           0       0.74      0.74      0.74        99
           1       0.54      0.55      0.54        55

    accuracy                           0.67       154
   macro avg       0.64      0.64      0.64       154
weighted avg       0.67      0.67      0.67       154
```

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

[20]:
```python
# 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)
```

[21]:
```python
print(y_pred)
```

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

[22]:
```python
print(np.array(y_test))
```

```
[0 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]
```

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

Accuracy: 0.7142857142857143

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

```
[[82 17]
 [27 28]]
```

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

```
              precision    recall  f1-score   support

           0       0.75      0.83      0.79        99
           1       0.62      0.51      0.56        55

    accuracy                           0.71       154
   macro avg       0.69      0.67      0.67       154
weighted avg       0.71      0.71      0.71       154
```

2.0.3 criterion="entropy", splitter="best"

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

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

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

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

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

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

Accuracy: 0.6883116883116883

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

      [[73 26]
       [22 33]]
```

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

                    precision    recall  f1-score   support

                 0       0.77      0.74      0.75        99
                 1       0.56      0.60      0.58        55

          accuracy                           0.69       154
         macro avg       0.66      0.67      0.67       154
      weighted avg       0.69      0.69      0.69       154
```

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

```
[32]: # 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)
```

```
[33]: print(y_pred)

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

```
[34]: 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 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 0 1 1 0 0 0 0 1 0 0 1 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]
```

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

      Accuracy: 0.7077922077922078
```

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

      [[78 21]
       [24 31]]
```

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

```
              precision    recall  f1-score   support

           0       0.76      0.79      0.78        99
           1       0.60      0.56      0.58        55

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

2.0.5 criterion="entropy", splitter="random"

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

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

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

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

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

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

```
Accuracy: 0.6883116883116883
```

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

```
[[77 22]
 [26 29]]
```

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

```
              precision    recall  f1-score   support

           0       0.75      0.78      0.76        99
```

	precision	recall	f1-score	support
1	0.57	0.53	0.55	55
accuracy			0.69	154
macro avg	0.66	0.65	0.65	154
weighted avg	0.68	0.69	0.69	154

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

```
[44]: # 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)
```

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

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

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

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

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

```
Accuracy: 0.7532467532467533
```

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

```
[[82 17]
 [21 34]]
```

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

	precision	recall	f1-score	support
0	0.80	0.83	0.81	99
1	0.67	0.62	0.64	55
accuracy			0.75	154
macro avg	0.73	0.72	0.73	154

```
weighted avg       0.75      0.75      0.75       154
```

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

```
[50]: # 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)
```

```
[51]: 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 1 0 1 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 0 0]
```

```
[52]: 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 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 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 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 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

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

```
Accuracy: 0.7987012987012987
```

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

```
[[88 11]
 [20 35]]
```

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

```
              precision    recall  f1-score   support

           0       0.81      0.89      0.85        99
           1       0.76      0.64      0.69        55

    accuracy                           0.80       154
   macro avg       0.79      0.76      0.77       154
weighted avg       0.80      0.80      0.79       154
```

```
[56]: 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()
```

```
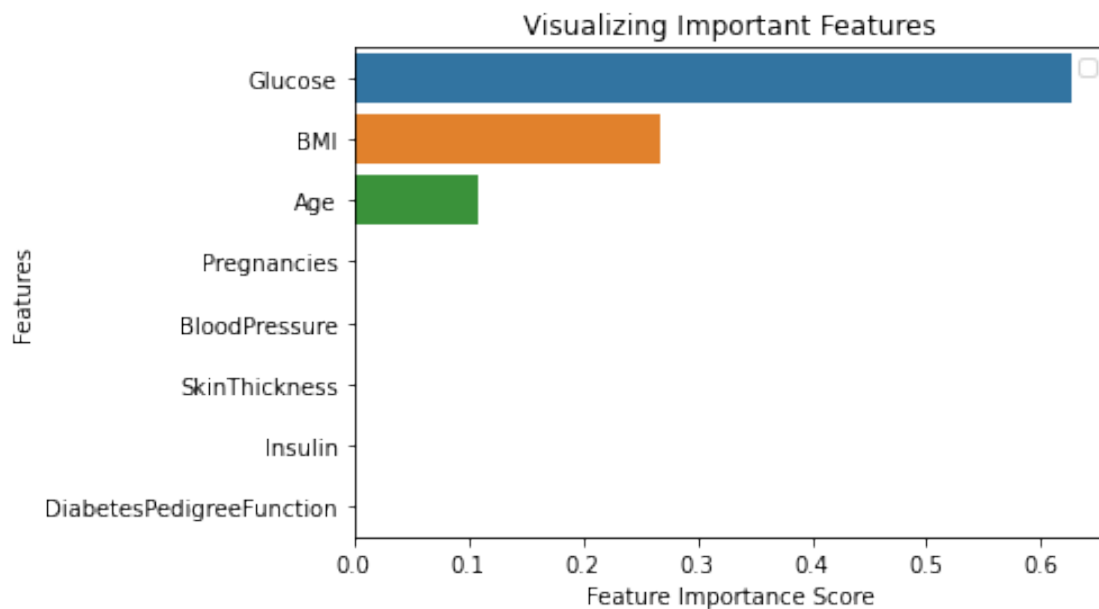Glucose                     0.625810
BMI                         0.266582
Age                         0.107608
Pregnancies                 0.000000
BloodPressure               0.000000
SkinThickness               0.000000
Insulin                     0.000000
DiabetesPedigreeFunction    0.000000
dtype: float64
```

No handles with labels found to put in legend.

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

```
[57]:  # 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)
```

[58]: `print(y_pred)`

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

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

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

[60]:
```
accuracy["dt_entropy_random_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7077922077922078
```

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

```
[[69 30]
 [15 40]]
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.70      0.75        99
           1       0.57      0.73      0.64        55

    accuracy                           0.71       154
   macro avg       0.70      0.71      0.70       154
weighted avg       0.73      0.71      0.71       154
```

3   Accuracy visulization of Decision Tree

[63]:
```
accuracy_df_dt = pd.DataFrame(list(zip(accuracy.keys(), accuracy.values())),
 →columns =['Arguments', 'Accuracy'])
accuracy_df_dt
```

[63]:
```
        Arguments  Accuracy
0    dt_gini_best  0.668831
```

```
1         dt_gini_best_8  0.714286
2         dt_entropy_best  0.688312
3      dt_entropy_best_8  0.707792
4      dt_entropy_random  0.688312
5    dt_entropy_random_8  0.753247
6      dt_entropy_best_3  0.798701
7    dt_entropy_random_3  0.707792
```

[64]:
```python
fig = px.bar(accuracy_df_dt, x='Arguments', y='Accuracy')
fig.show()
```

4 Random Forest

[65]:
```python
accuracy_rf = {}
```

4.0.1 n_estimators = 1000, criterion='entropy'

[66]:
```python
# 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)
```

[67]:
```python
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 1 0 0 1 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0 1 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 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 0 0]
```

[68]:
```python
print(np.array(y_test))
```

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

[69]:
```python
accuracy_rf["rf_entropy_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8246753246753247
```

[70]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[88 11]
 [16 39]]
```

14

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

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        99
           1       0.78      0.71      0.74        55

    accuracy                           0.82       154
   macro avg       0.81      0.80      0.80       154
weighted avg       0.82      0.82      0.82       154
```

```
[72]: 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                     0.239327
BMI                         0.168033
Age                         0.132346
DiabetesPedigreeFunction    0.128437
BloodPressure               0.095808
Pregnancies                 0.086516
Insulin                     0.076237
SkinThickness               0.073297
dtype: float64
```

Visualizing Important Features

Glucose

BMI

Age

DiabetesPedigreeFunction

BloodPressure

Pregnancies

Insulin

SkinThickness

Feature Importance Score
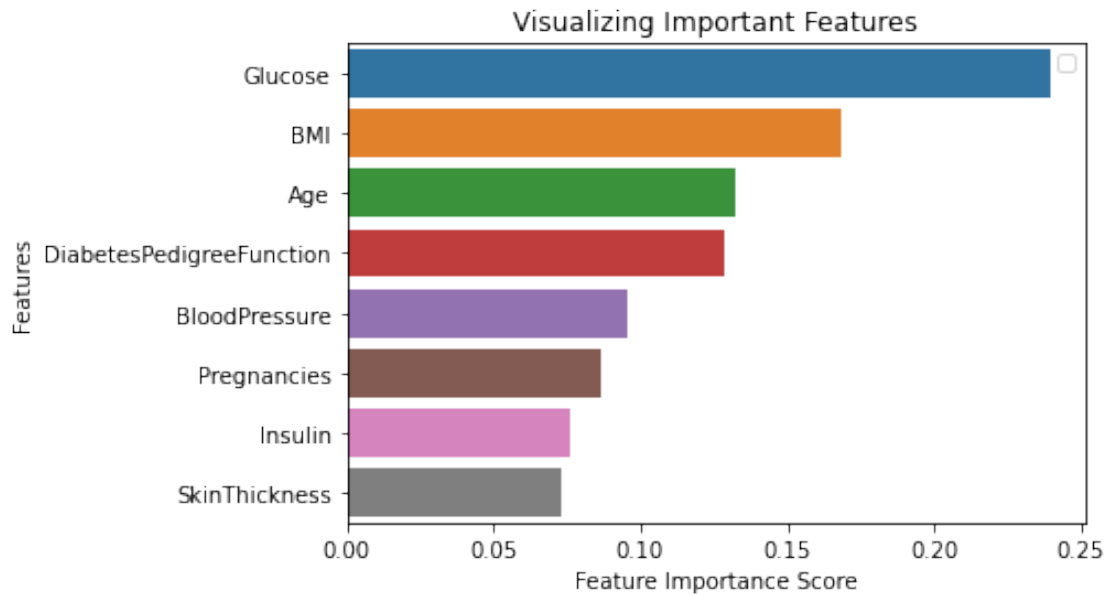
4.0.2 n_estimators = 100, criterion='entropy'

```python
[73]: # 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)
```

```python
[74]: print(y_pred)
```

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

```python
[75]: print(np.array(y_test))
```

```
[0 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 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 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]
```

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

Accuracy: 0.7857142857142857

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

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

```
[78]: 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

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

```
[79]: # 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)
```

```
[80]: 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 1 0 0 0 1 0 1 0 1 0 0 1 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 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 1 0 0
 0 0 0 1 0 0]
```

```
[81]: 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 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 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 0 1 1 0 0 0 0 1 0 0 1 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]
```

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

Accuracy: 0.8051948051948052

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

      [[87 12]
       [18 37]]

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

                    precision    recall  f1-score   support

                 0       0.83      0.88      0.85        99
                 1       0.76      0.67      0.71        55

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

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

```
[85]: # 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)
```

```
[86]: print(y_pred)

      [1 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 1 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 1 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 0 0]
```

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

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

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

      Accuracy: 0.7987012987012987
```

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

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

[90]: `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
```

4.0.5 $n\_estimators = 1000$, $random\_state = 42$, $max\_depth = 8$, criterion='entropy'

[91]:
```python
# 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)
```

[92]: `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 1 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 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 0 0]
```

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

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

[94]:
```python
accuracy_rf["rf_entropy_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8051948051948052
```

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

```
[[88 11]
 [19 36]]
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85        99
           1       0.77      0.65      0.71        55

    accuracy                           0.81       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.80      0.81      0.80       154
```

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

```
[97]: # 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)
```

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

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

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

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

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

```
Accuracy: 0.7987012987012987
```

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

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

```
[102]: 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

4.0.7 n_estimators = 1000

```
[103]: # 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)
```

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

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

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

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

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

```
Accuracy: 0.8181818181818182
```

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

```
[[88 11]
 [17 38]]
```

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

	precision	recall	f1-score	support
0	0.84	0.89	0.86	99

```
          1        0.78       0.69       0.73        55

   accuracy                              0.82       154
  macro avg         0.81       0.79       0.80       154
weighted avg        0.82       0.82       0.82       154
```

4.0.8 n_estimators = 100

```
[109]: # 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)
```

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

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

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

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

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

```
Accuracy: 0.7922077922077922
```

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

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

```
[114]: 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
```

	precision	recall	f1-score	support
macro avg	0.78	0.76	0.77	154
weighted avg	0.79	0.79	0.79	154

4.0.9 n__estimators = 1000, random__state = 42

```python
# 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)
```
[115]:

[116]:
```python
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 1 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0
 1 0 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 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 1 0 0
 0 0 0 1 0 0]
```

[117]:
```python
print(np.array(y_test))
```

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

[118]:
```python
accuracy_rf["rf_gini_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8116883116883117
```

[119]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[88 11]
 [18 37]]
```

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

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        99
           1       0.77      0.67      0.72        55

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

4.0.10 n_estimators = 100, random_state = 42

```
[121]: # 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)
```

```
[122]: 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 1 0 0 1 1 0 0 0 0 1 0 1 0 1 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 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1
 0 1 0 0 0 0 1 0 1 0 1 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 0 0]
```

```
[123]: 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 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 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 1 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

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

```
Accuracy: 0.8116883116883117
```

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

```
[[88 11]
 [18 37]]
```

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

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        99
           1       0.77      0.67      0.72        55

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

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

```
[127]: # 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)
```

```
[128]: 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 1 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 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 0 0]
```

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

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

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

```
Accuracy: 0.8051948051948052
```

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

```
[[88 11]
 [19 36]]
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85        99
           1       0.77      0.65      0.71        55

    accuracy                           0.81       154
   macro avg       0.79      0.77      0.78       154
weighted avg       0.80      0.81      0.80       154
```

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

```python
[133]: # 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)
```

```python
[134]: 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 1 0 0 1 1 0 0 0 0 1 0 1 0 1 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 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 1
 0 1 0 0 0 0 1 0 1 0 1 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 0 0]
```

```python
[135]: print(np.array(y_test))
```

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

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

```
Accuracy: 0.8116883116883117
```

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

```
[[88 11]
 [18 37]]
```

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

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        99
           1       0.77      0.67      0.72        55

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

5 Accuracy visulization of Random Forest

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

```
[139]:              Arguments  Accuracy
       0         rf_entropy_1000  0.824675
       1          rf_entropy_100  0.785714
       2       rf_entropy_1000_42  0.805195
       3        rf_entropy_100_42  0.798701
       4     rf_entropy_1000_42_8  0.805195
       5      rf_entropy_100_42_8  0.798701
       6            rf_gini_1000  0.818182
       7             rf_gini_100  0.792208
       8          rf_gini_1000_42  0.811688
       9           rf_gini_100_42  0.811688
       10       rf_gini_1000_42_8  0.805195
       11        rf_gini_100_42_8  0.811688
```

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

```
[141]: 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()
```

```
82.47
```