Decision Tree & Random Forest V4

November 19, 2021

Replace All zero features 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  \
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
```

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

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

```
[7]: # replace zero skinthickness value with it's median
     print("Before SkinThickness median : ",round(df['SkinThickness'].median(),1))
     df['SkinThickness'] = df['SkinThickness'].replace(0, df['SkinThickness'].
      ↪median())
     print("After SkinThickness median : ",round(df['SkinThickness'].median(),1))
```

```
Before SkinThickness median :  23.0
After SkinThickness median :  23.0
```

[8]:

```
# replace zero bloodpressure value with it's median
print("Before BloodPressure median : ",round(df['BloodPressure'].median(),1))
df['BloodPressure'] = df['BloodPressure'].replace(0, df['BloodPressure'].
 ↪median())
print("After BloodPressure median : ",round(df['BloodPressure'].median(),1))
```

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

[9]:

```
# replace zero Glucose value with it's median
print("Before Glucose median : ",round(df['Glucose'].median(),1))
df['Glucose'] = df['Glucose'].replace(0, df['Glucose'].median())
print("After Glucose median : ",round(df['Glucose'].median(),1))
```

```
Before Glucose median :  117.0
After Glucose median :  117.0
```

[10]:

```
# replace zero Insulin value with it's median
print("Before Insulin median : ",round(df['Insulin'].median(),1))
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].median())
print("After Insulin median : ",round(df['Insulin'].median(),1))
```

```
Before Insulin median :  30.5
After Insulin median :  31.2
```

[11]:

```
df.describe()
```

[11]:

```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  121.656250      72.386719      27.334635   94.652344
std       3.369578   30.438286      12.096642       9.229014  105.547598
min       0.000000   44.000000      24.000000       7.000000   14.000000
25%       1.000000   99.750000      64.000000      23.000000   30.500000
50%       3.000000  117.000000      72.000000      23.000000   31.250000
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    32.450911                  0.471876   33.240885    0.348958
std      6.875366                  0.331329   11.760232    0.476951
min     18.200000                  0.078000   21.000000    0.000000
25%     27.500000                  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
```

[12]:

```
df.corr()
```

[12]:

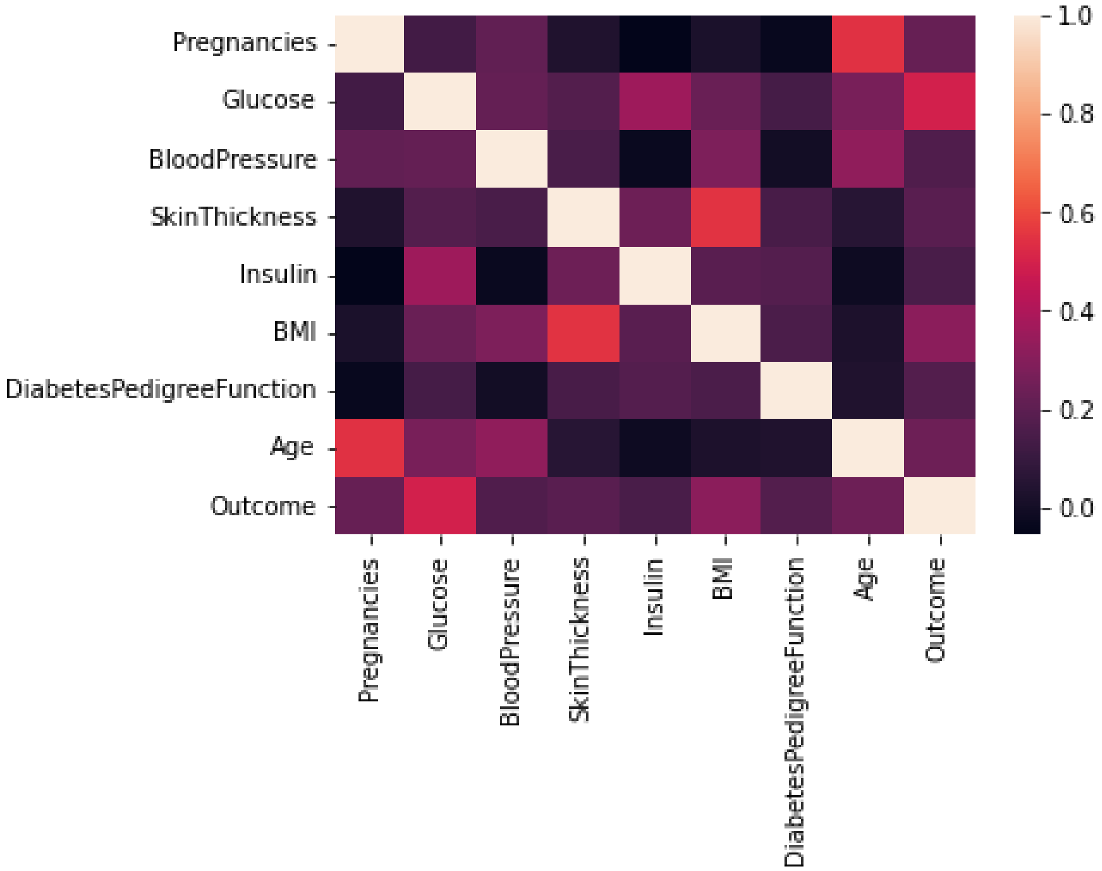
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.128213       0.208615       0.032568
Glucose                      0.128213  1.000000       0.218937       0.172143
BloodPressure                0.208615  0.218937       1.000000       0.147809
SkinThickness                0.032568  0.172143       0.147809       1.000000
Insulin                     -0.055697  0.357573      -0.028721       0.238188
BMI                          0.021546  0.231400       0.281132       0.546951
DiabetesPedigreeFunction    -0.033523  0.137327      -0.002378       0.142977
Age                          0.544341  0.266909       0.324915       0.054514
Outcome                      0.221898  0.492782       0.165723       0.189065

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.055697  0.021546                 -0.033523
Glucose                   0.357573  0.231400                  0.137327
BloodPressure            -0.028721  0.281132                 -0.002378
SkinThickness             0.238188  0.546951                  0.142977
Insulin                   1.000000  0.189022                  0.178029
BMI                       0.189022  1.000000                  0.153506
DiabetesPedigreeFunction  0.178029  0.153506                  1.000000
Age                      -0.015413  0.025744                  0.033561
Outcome                   0.148457  0.312249                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266909  0.492782
BloodPressure             0.324915  0.165723
SkinThickness             0.054514  0.189065
Insulin                  -0.015413  0.148457
BMI                       0.025744  0.312249
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[13]:

```
sns.heatmap(df.corr())
```

[13]:

```
<AxesSubplot:>
```



1 Data split

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

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

```
[15]:    Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
      0            6      148             72             35     30.5  33.6
      1            1       85             66             29     30.5  26.6
      2            8      183             64             23     30.5  23.3
      3            1       89             66             23     94.0  28.1
      4            0      137             40             35    168.0  43.1

         DiabetesPedigreeFunction  Age
      0                     0.627   50
      1                     0.351   31
      2                     0.672   32
```

```
3                       0.167   21
4                       2.288   33
```

[16]:
```
Y.head()
```

[16]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

[17]:
```
# 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)
```

[18]:
```
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

[19]:
```
accuracy = {}
```

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

[20]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="gini", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

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

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

[22]:
```
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]
```

[23]:

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

```
Accuracy: 0.6818181818181818
```

[24]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[73 26]
 [23 32]]
```

[25]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.74      0.75        99
           1       0.55      0.58      0.57        55

    accuracy                           0.68       154
   macro avg       0.66      0.66      0.66       154
weighted avg       0.69      0.68      0.68       154
```

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

[26]:

```
# 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)
```

[27]:

```
print(y_pred)
```

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

[28]:

```
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]
```

[29]:
```python
accuracy["dt_gini_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7142857142857143
```

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

```
[[79 20]
 [24 31]]
```

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

```
              precision    recall  f1-score   support

           0       0.77      0.80      0.78        99
           1       0.61      0.56      0.58        55

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

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

[32]:
```python
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[33]:
```python
print(y_pred)
```

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

[34]:
```python
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]
```

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

```
Accuracy: 0.6558441558441559
```

[36]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

[37]:

```
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
```

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

[38]:

```
# 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)
```

[39]:

```
print(y_pred)
```

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

[40]:

```
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]
```

[41]:

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

```
Accuracy: 0.6818181818181818
```

[42]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[77 22]
 [27 28]]
```

[43]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.74      0.78      0.76        99
           1       0.56      0.51      0.53        55

    accuracy                           0.68       154
   macro avg       0.65      0.64      0.65       154
weighted avg       0.68      0.68      0.68       154
```

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

[44]:

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

[45]:

```
print(y_pred)
```

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

[46]:

```
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 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]
```

[47]:

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

```
Accuracy: 0.7077922077922078
```

[48]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

[49]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.83      0.78        99
           1       0.61      0.49      0.55        55

    accuracy                           0.71       154
   macro avg       0.68      0.66      0.67       154
weighted avg       0.70      0.71      0.70       154
```

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

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

[51]:
```
print(y_pred)
```

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

[53]:
```
accuracy["dt_entropy_random_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6818181818181818
```

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

```
[[79 20]
 [29 26]]
```

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

```
              precision    recall  f1-score   support

           0       0.73      0.80      0.76        99
           1       0.57      0.47      0.51        55

    accuracy                           0.68       154
   macro avg       0.65      0.64      0.64       154
```

```
weighted avg       0.67      0.68      0.67       154
```

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

[56]:

```
# 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)
```

[57]:

```
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]
```

[58]:

```
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]
```

[59]:

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

```
Accuracy: 0.7922077922077922
```

[60]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

[61]:

```
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
```

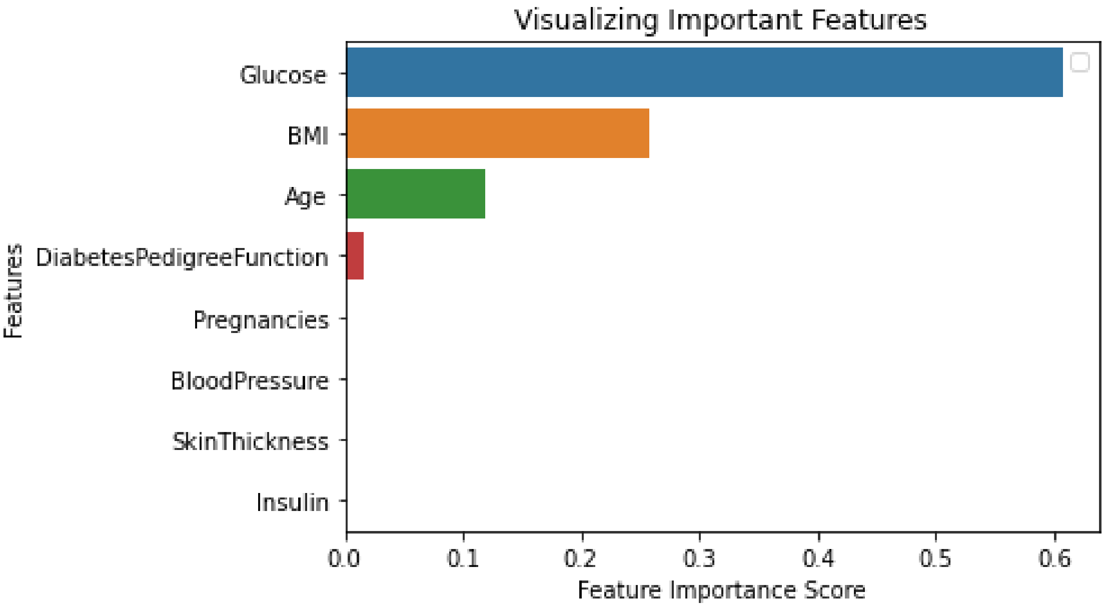
[62]:

```
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()
```

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

```
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
```



2.0.8 criterion=“entropy”, splitter=“random”, max_depth=3

[63]:

```
# 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)
```

[64]:
```
print(y_pred)
```

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

[65]:
```
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]
```

[66]:
```
accuracy["dt_entropy_random_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7402597402597403
```

[67]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[97  2]
 [38 17]]
```

[68]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.72      0.98      0.83        99
           1       0.89      0.31      0.46        55

    accuracy                           0.74       154
   macro avg       0.81      0.64      0.64       154
weighted avg       0.78      0.74      0.70       154
```

3 Accuracy visulization of Decision Tree

[69]:
```
accuracy_df_dt = pd.DataFrame(list(zip(accuracy.keys(), accuracy.values())),␣
 ↪columns =['Arguments', 'Accuracy'])
accuracy_df_dt
```

[69]:
```
             Arguments  Accuracy
0         dt_gini_best  0.681818
```

```
1       dt_gini_best_8  0.714286
2      dt_entropy_best  0.655844
3    dt_entropy_best_8  0.681818
4    dt_entropy_random  0.707792
5  dt_entropy_random_8  0.681818
6    dt_entropy_best_3  0.792208
7  dt_entropy_random_3  0.740260
```

[70]:
```
fig = px.bar(accuracy_df_dt, x='Arguments', y='Accuracy')
fig.show()
```

4 Random Forest

[71]:
```
accuracy_rf = {}
```

4.0.1 n_estimators = 1000, criterion='entropy'

[72]:
```
# 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)
```

[73]:
```
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 1 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 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]
```

[74]:
```
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]
```

[75]:
```
accuracy_rf["rf_entropy_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7792207792207793
```

[76]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[85 14]
 [20 35]]
```

[77]:

```
print(metrics.classification_report(y_test, y_pred))
```

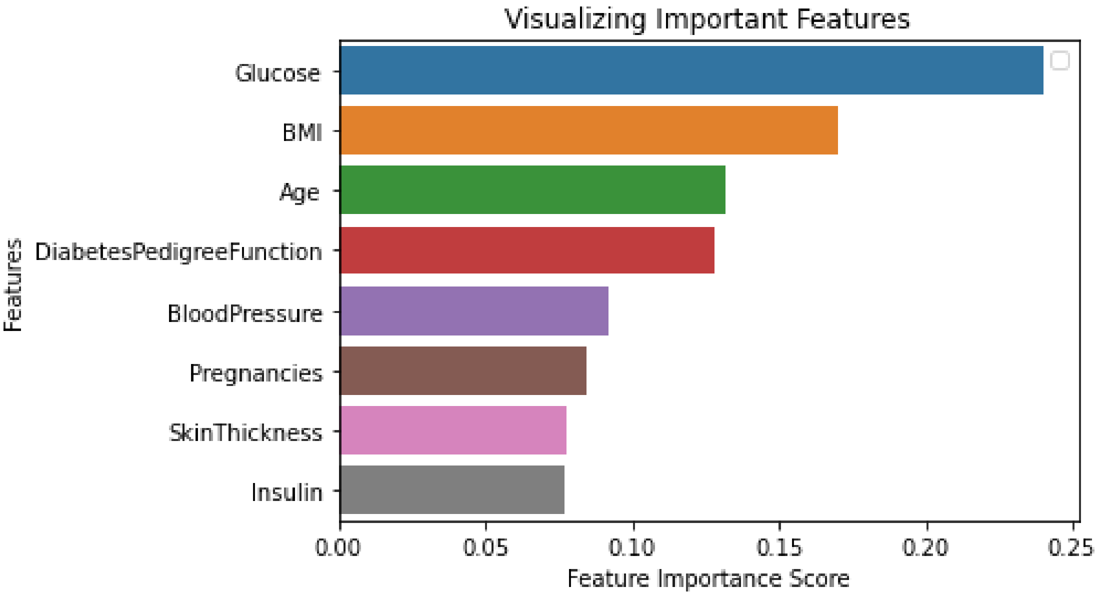
```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83        99
           1       0.71      0.64      0.67        55

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

[78]:

```
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                     0.239902
BMI                         0.170029
Age                         0.131580
DiabetesPedigreeFunction    0.128233
BloodPressure               0.091674
Pregnancies                 0.084247
SkinThickness               0.077441
Insulin                     0.076894
dtype: float64
```



4.0.2 n_estimators = 100, criterion='entropy'

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

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

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

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

```
Accuracy: 0.7987012987012987
```

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

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

[84]:
```
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
```

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

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

[86]:
```
print(y_pred)
```

```
[0 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 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 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]
```

[87]:
```
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]
```

[88]:
```
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
```

[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
```

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

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

[92]:
```
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 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 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 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]
```

[93]:
```
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]
```

[94]:
```
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
```

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

```
[[85 14]
 [19 36]]
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        99
           1       0.72      0.65      0.69        55

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

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

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

[98]:
```
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 0 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 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 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]
```

[99]:
```
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]
```

[100]:
```
accuracy_rf["rf_entropy_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

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

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

[102]:

```
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
```

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

[103]:

```
# 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)
```

[104]:

```
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 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 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 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]
```

[105]:

```
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]
```

[106]:

```
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
```

[107]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[85 14]
 [19 36]]
```

[108]:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        99
           1       0.72      0.65      0.69        55

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

4.0.7 n_estimators = 1000

[109]:

```
# 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)
```

[110]:

```
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 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 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 1 0]
```

[111]:

```
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]
```

[112]:

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

```
Accuracy: 0.8051948051948052
```

[113]:

```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[87 12]
 [18 37]]
```

[114]:

```
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
```

4.0.8 n_estimators = 100

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

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

[117]:
```
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]
```

[118]:
```
accuracy_rf["rf_gini_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

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

```
[[85 14]
 [19 36]]
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        99
           1       0.72      0.65      0.69        55

    accuracy                           0.79       154
```

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

4.0.9 n_estimators = 1000, random_state = 42

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

[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 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 1 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]
```

[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 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]
```

[124]:
```
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
```

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

```
[[87 12]
 [17 38]]
```

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

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86        99
           1       0.76      0.69      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

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

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

```
[129]: 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]
```

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

```
Accuracy: 0.7857142857142857
```

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

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

```
[132]: 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
```

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

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

[134]:
```
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 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]
```

[135]:
```
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]
```

[136]:
```
accuracy_rf["rf_gini_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7987012987012987
```

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

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

[138]:
```
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
```

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

[139]:
```
# 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)
```

[140]:
```
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 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 1 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 1 0 0 0 1 0 0 0 0 0 1 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]
```

[141]:
```
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]
```

[142]:
```
accuracy_rf["rf_gini_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[143]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

[144]:
```
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 Accuracy visulization of Random Forest

[145]:

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

[145]:

```
              Arguments  Accuracy
0       rf_entropy_1000  0.779221
1        rf_entropy_100  0.798701
2    rf_entropy_1000_42  0.798701
3     rf_entropy_100_42  0.785714
4   rf_entropy_1000_42_8  0.785714
5    rf_entropy_100_42_8  0.785714
6          rf_gini_1000  0.805195
7           rf_gini_100  0.785714
8       rf_gini_1000_42  0.811688
9        rf_gini_100_42  0.785714
10    rf_gini_1000_42_8  0.798701
11     rf_gini_100_42_8  0.785714
```

[146]:

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

[147]:

```
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()
```

```
81.17
```