Decision Tree & Random Forest V4

November 21, 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]:
```python
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]:
```python
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
```

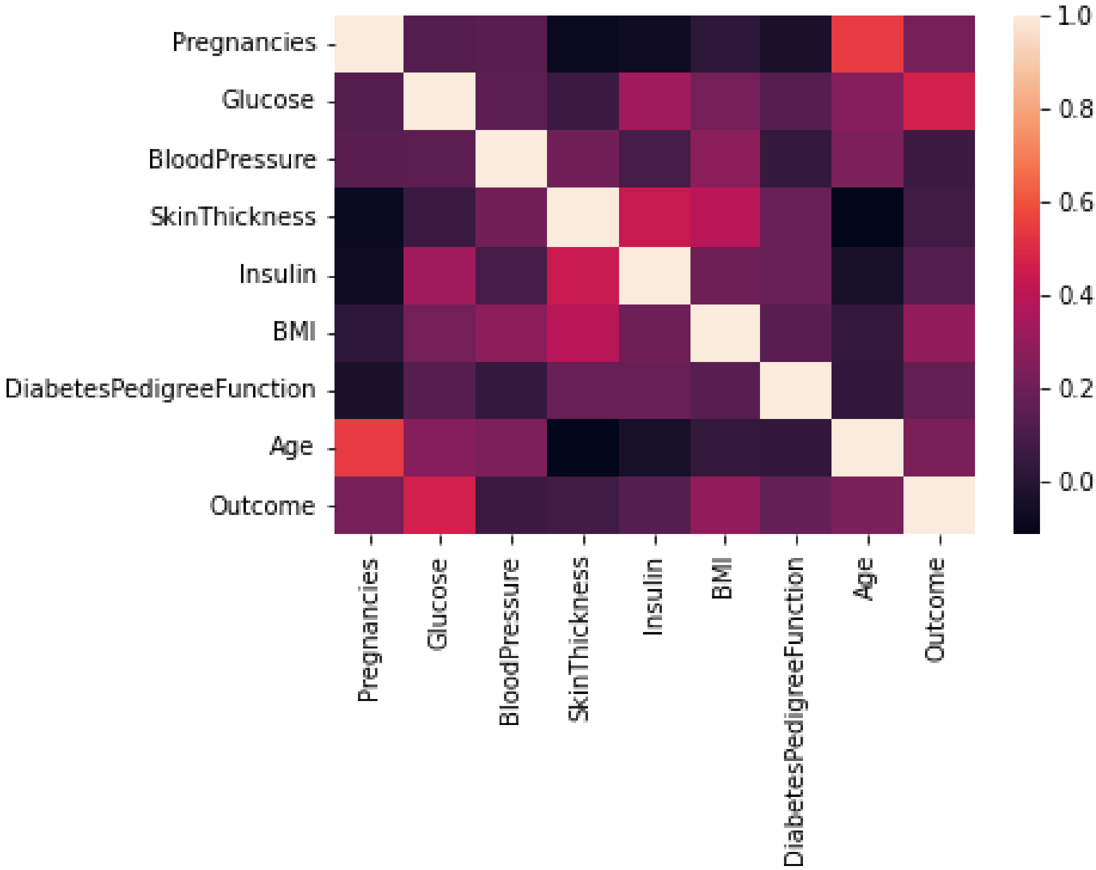
[6]:
```python
df.corr()
```

[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

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

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

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

2 Preprocessing

[13]:
```
# replace zero bmi value with it's median
print("Before BMI median : ",round(x_train.loc[:, 'BMI'].median(),1))
x_test.loc[:, 'BMI'] = x_test.loc[:, 'BMI'].replace(0, x_train.loc[:, 'BMI'].
↪median())
x_train.loc[:, 'BMI'] = x_train.loc[:, 'BMI'].replace(0, x_train.loc[:, 'BMI'].
↪median())
print("After BMI median : ",round(x_train.loc[:, 'BMI'].median(),1))
```

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

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

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

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

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

[14]:

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

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

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

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

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

[15]:

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

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

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

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

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

[16]:
```python
# replace zero Glucose value with it's median
print("Before Glucose median : ",round(x_train.loc[:, 'Glucose'].median(),1))
x_test.loc[:, 'Glucose'] = x_test.loc[:, 'Glucose'].replace(0, x_train.loc[:,␣
 ↪'Glucose'].median())
x_train.loc[:, 'Glucose'] = x_train.loc[:, 'Glucose'].replace(0, x_train.loc[:,␣
 ↪'Glucose'].median())
print("After Glucose median : ",round(x_train.loc[:, 'Glucose'].median(),1))
```

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

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

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

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

[17]:
```
# replace zero Insulin value with it's median
print("Before Insulin median : ",round(x_train.loc[:, 'Insulin'].median(),1))
x_test.loc[:, 'Insulin'] = x_test.loc[:, 'Insulin'].replace(0, x_train.loc[:,␣
↪'Insulin'].median())
x_train.loc[:, 'Insulin'] = x_train.loc[:, 'Insulin'].replace(0, x_train.loc[:,␣
↪'Insulin'].median())
print("After Insulin median : ",round(x_train.loc[:, 'Insulin'].median(),1))
```

```
Before Insulin median :  16.5
After Insulin median :  17.2
```

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

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

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

3 Decision Tree

[18]:
```
accuracy = {}
```

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

[19]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="gini", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[20]:
```
print(y_pred)
```

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

[21]:
```
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]
```

[22]:
```python
accuracy["dt_gini_best"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6753246753246753
```

[23]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[72 27]
 [23 32]]
```

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

```
              precision    recall  f1-score   support

           0       0.76      0.73      0.74        99
           1       0.54      0.58      0.56        55

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

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

[25]:
```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)
```

[26]:
```python
print(y_pred)
```

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

[27]:
```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]
```

[28]:
```
accuracy["dt_gini_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6948051948051948
```

[29]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[79 20]
 [27 28]]
```

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

```
              precision    recall  f1-score   support

           0       0.75      0.80      0.77        99
           1       0.58      0.51      0.54        55

    accuracy                           0.69       154
   macro avg       0.66      0.65      0.66       154
weighted avg       0.69      0.69      0.69       154
```

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

[31]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best")
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[32]:
```
print(y_pred)
```

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

[33]:
```
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]
```

[34]:
```
accuracy["dt_entropy_best"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6688311688311688
```

[35]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

[36]:
```
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
```

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

[37]:
```
# 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)
```

[38]:
```
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 0 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 0 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]
```

[39]:
```
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]
```

[40]:
```
accuracy["dt_entropy_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6818181818181818
```

[41]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[78 21]
 [28 27]]
```

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

```
              precision    recall  f1-score   support

           0       0.74      0.79      0.76        99
           1       0.56      0.49      0.52        55

    accuracy                           0.68       154
   macro avg       0.65      0.64      0.64       154
weighted avg       0.67      0.68      0.68       154
```

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

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

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

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

```
[45]: 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]
```

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

```
Accuracy: 0.7272727272727273
```

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

```
[[74 25]
 [17 38]]
```

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

```
              precision    recall  f1-score   support

           0       0.81      0.75      0.78        99
```

```
           1       0.60      0.69      0.64        55

    accuracy                           0.73       154
   macro avg       0.71      0.72      0.71       154
weighted avg       0.74      0.73      0.73       154
```

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

[49]:
```
# 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)
```

[50]:
```
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 1 0 1 1 1 0 0 0 1 1 1 0
 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 1 0 0 0
 1 0 1 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 1 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[51]:
```
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]
```

[52]:
```
accuracy["dt_entropy_random_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6948051948051948
```

[53]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[75 24]
 [23 32]]
```

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

```
              precision    recall  f1-score   support

           0       0.77      0.76      0.76        99
           1       0.57      0.58      0.58        55

    accuracy                           0.69       154
   macro avg       0.67      0.67      0.67       154
```

```
weighted avg       0.70      0.69      0.70       154
```

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

[55]:
```python
# 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)
```

[56]:
```python
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]
```

[57]:
```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]
```

[58]:
```python
accuracy["dt_entropy_best_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[59]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

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

[60]:
```python
print(metrics.classification_report(y_test, y_pred))
```

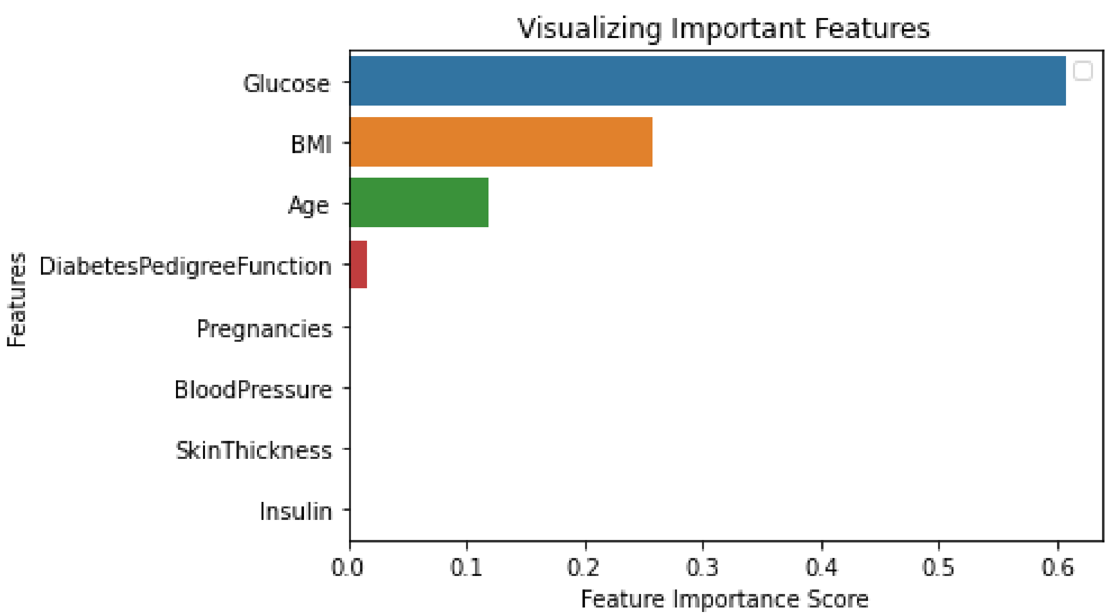
```
              precision    recall  f1-score   support

           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
```

[61]:
```python
feature_imp = pd.Series(clf.feature_importances_,index=X.columns).
 ↪sort_values(ascending=False)
```

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

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

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



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

[62]:
```
# 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)
```

[63]:
```
print(y_pred)
```

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

[64]:
```
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]
```

[65]:
```
accuracy["dt_entropy_random_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7142857142857143
```

[66]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[62 37]
 [ 7 48]]
```

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

```
              precision    recall  f1-score   support

           0       0.90      0.63      0.74        99
           1       0.56      0.87      0.69        55

    accuracy                           0.71       154
   macro avg       0.73      0.75      0.71       154
weighted avg       0.78      0.71      0.72       154
```

4 Accuracy visulization of Decision Tree

[68]:
```
accuracy_df_dt = pd.DataFrame(list(zip(accuracy.keys(), accuracy.values())),␣
↪columns =['Arguments', 'Accuracy'])
accuracy_df_dt
```

[68]:

	Arguments	Accuracy
0	dt_gini_best	0.675325

```
1        dt_gini_best_8  0.694805
2       dt_entropy_best  0.668831
3     dt_entropy_best_8  0.681818
4     dt_entropy_random  0.727273
5   dt_entropy_random_8  0.694805
6     dt_entropy_best_3  0.792208
7   dt_entropy_random_3  0.714286
```

[69]:
```python
fig = px.bar(accuracy_df_dt, x='Arguments', y='Accuracy')
fig.show()
```

5 Random Forest

[70]:
```python
accuracy_rf = {}
```

5.0.1 n_estimators = 1000, criterion='entropy'

[71]:
```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)
```

[72]:
```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 0 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1 0 0
 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 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]
```

[73]:
```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]
```

[74]:
```python
accuracy_rf["rf_entropy_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7987012987012987
```

[75]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

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

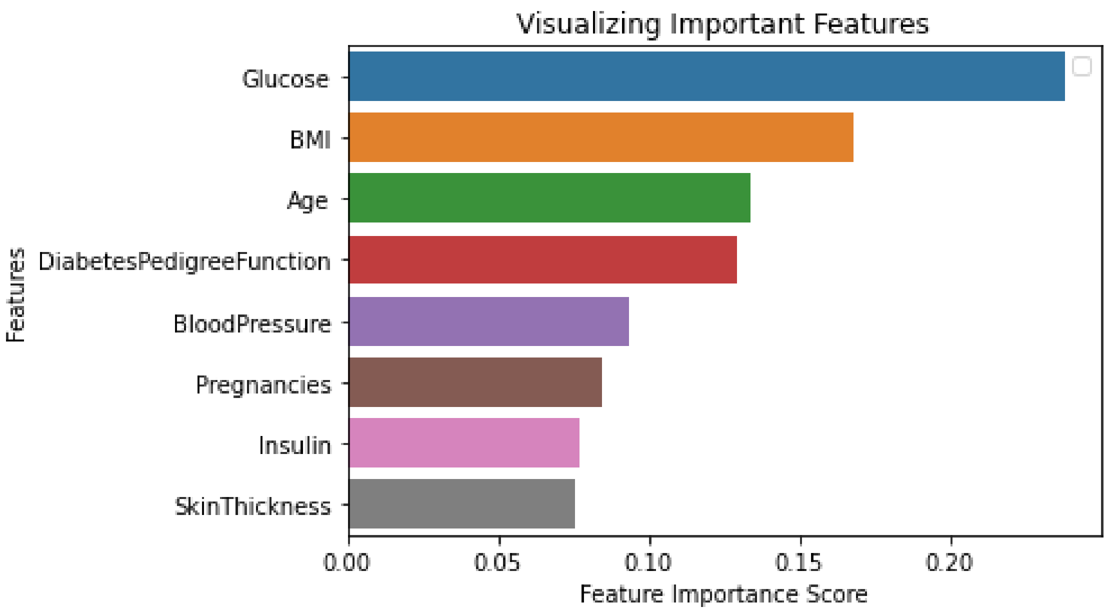
[76]:
```python
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85        99
           1       0.74      0.67      0.70        55

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

[77]:
```python
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                     0.237800
BMI                         0.168316
Age                         0.133889
DiabetesPedigreeFunction    0.129277
BloodPressure               0.093506
Pregnancies                 0.084315
Insulin                     0.077204
SkinThickness               0.075694
dtype: float64
```



5.0.2 n_estimators = 100, criterion='entropy'

[78]:
```
# 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)
```

[79]:
```
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 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 0 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 0 0 1 0 0 0 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]
```

[80]:
```
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]
```

[81]:
```
accuracy_rf["rf_entropy_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[82]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.64      0.69        55

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

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

[84]:
```python
# 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)
```

[85]:
```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 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 0
 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 1 0 0
 0 0 0 1 1 0]
```

[86]:
```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]
```

[87]:
```python
accuracy_rf["rf_entropy_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[88]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        99
           1       0.73      0.65      0.69        55

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

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

[90]:
```
# 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)
```

[91]:
```
print(y_pred)
```

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

[92]:
```
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]
```

[93]:
```
accuracy_rf["rf_entropy_100_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[94]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        99
           1       0.73      0.65      0.69        55

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

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

[96]:
```
# 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)
```

[97]:
```
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 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 0
 0 1 0 0 0 0 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]
```

[98]:
```
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 0 1 0 1 0 0 1 1 1 0 0
 1 0 0 1 0 0]
```

[99]:
```
accuracy_rf["rf_entropy_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7727272727272727
```

[100]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [22 33]]
```

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

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.83        99
           1       0.72      0.60      0.65        55

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

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

[102]:
```python
# 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)
```

[103]:
```python
print(y_pred)
```

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

[104]:
```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]
```

[105]:
```python
accuracy_rf["rf_entropy_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[106]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        99
           1       0.73      0.65      0.69        55

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

5.0.7 n_estimators = 1000

[108]:
```
# 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)
```

[109]:
```
print(y_pred)
```

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

[110]:
```
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]
```

[111]:
```
accuracy_rf["rf_gini_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[112]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84        99
```

```
           1       0.73      0.64      0.68        55

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

5.0.8 n_estimators = 100

[114]:
```python
# 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)
```

[115]:
```python
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 0 0 0 0 1 0 1 0 1 0 1 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 0 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]
```

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

[117]:
```python
accuracy_rf["rf_gini_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

[118]:
```python
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.64      0.69        55

    accuracy                           0.79       154
```

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

5.0.9 n_estimators = 1000, random_state = 42

[120]:
```
# 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)
```

[121]:
```
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 1 1 1 1 1 0 0
 1 1 1 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

[122]:
```
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]
```

[123]:
```
accuracy_rf["rf_gini_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[124]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84        99
           1       0.73      0.64      0.68        55

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

5.0.10 n_estimators = 100, random_state = 42

[126]:
```
# 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)
```

[127]:
```
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 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 1 0]
```

[128]:
```
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]
```

[129]:
```
accuracy_rf["rf_gini_100_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7792207792207793
```

[130]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [21 34]]
```

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

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.83        99
           1       0.72      0.62      0.67        55

    accuracy                           0.78       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.78      0.78      0.77       154
```

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

[132]:
```
# 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)
```

[133]:
```
print(y_pred)
```

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

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

[135]:
```
accuracy_rf["rf_gini_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7857142857142857
```

[136]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

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

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

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84        99
           1       0.74      0.62      0.67        55

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

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

[138]:
```
# 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)
```

[139]:
```
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 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 1 0]
```

[140]:
```
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]
```

[141]:
```
accuracy_rf["rf_gini_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7792207792207793
```

[142]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[86 13]
 [21 34]]
```

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

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.83        99
           1       0.72      0.62      0.67        55

    accuracy                           0.78       154
   macro avg       0.76      0.74      0.75       154
weighted avg       0.78      0.78      0.77       154
```

6 Accuracy visulization of Random Forest

[144]:
```
accuracy_df_rf = pd.DataFrame(list(zip(accuracy_rf.keys(), accuracy_rf.
 ↪values())), columns =['Arguments', 'Accuracy'])
accuracy_df_rf
```

[144]:

	Arguments	Accuracy
0	rf_entropy_1000	0.798701
1	rf_entropy_100	0.792208
2	rf_entropy_1000_42	0.792208
3	rf_entropy_100_42	0.792208
4	rf_entropy_1000_42_8	0.772727
5	rf_entropy_100_42_8	0.792208
6	rf_gini_1000	0.785714
7	rf_gini_100	0.792208
8	rf_gini_1000_42	0.785714
9	rf_gini_100_42	0.779221
10	rf_gini_1000_42_8	0.785714
11	rf_gini_100_42_8	0.779221

[145]:
```
fig = px.bar(accuracy_df_rf, x='Arguments', y='Accuracy')
fig.show()
```

[146]:
```
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()
```

```
79.87
```