

Decision Tree & Random Forest

September 29, 2021

```
[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 mean
      print("Before BMI mean : ",round(df['BMI'].mean(),1))
      df['BMI'] = df['BMI'].replace(0, df['BMI'].mean())
      print("After BMI mean : ",round(df['BMI'].mean(),1))
```

```
Before BMI mean : 32.0
```

```
After BMI mean : 32.5
```

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

Before SkinThickness mean : 20.5

After SkinThickness mean : 26.6

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

Before BloodPressure mean : 69.1

After BloodPressure mean : 72.3

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

```
[9]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	72.254807	26.606479	79.799479
std	3.369578	31.972618	12.115932	9.631241	115.244002
min	0.000000	0.000000	24.000000	7.000000	0.000000
25%	1.000000	99.000000	64.000000	20.536458	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	32.450805	0.471876	33.240885	0.348958
std	6.875374	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

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

```
[10]:
```

	Pregnancies	Glucose	BloodPressure	SkinThickness \
Pregnancies	1.000000	0.129459	0.208984	0.013376
Glucose	0.129459	1.000000	0.218579	0.145378
BloodPressure	0.208984	0.218579	1.000000	0.134155
SkinThickness	0.013376	0.145378	0.134155	1.000000
Insulin	-0.073535	0.331357	-0.038147	0.286469
BMI	0.021546	0.218814	0.281231	0.535703
DiabetesPedigreeFunction	-0.033523	0.137337	0.000371	0.154961
Age	0.544341	0.263514	0.326740	0.026423
Outcome	0.221898	0.466581	0.162986	0.175026

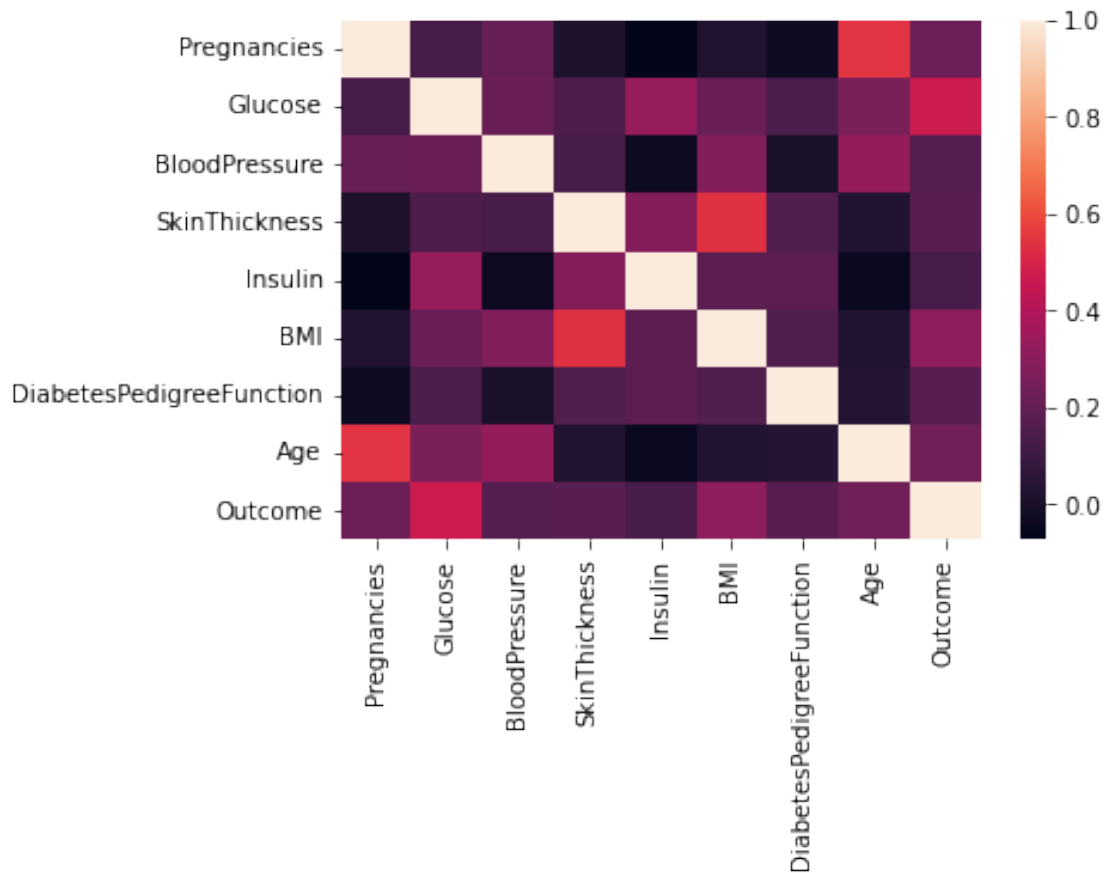
	Insulin	BMI	DiabetesPedigreeFunction \
Pregnancies	-0.073535	0.021546	-0.033523

Glucose	0.331357	0.218814	0.137337
BloodPressure	-0.038147	0.281231	0.000371
SkinThickness	0.286469	0.535703	0.154961
Insulin	1.000000	0.185365	0.185071
BMI	0.185365	1.000000	0.153508
DiabetesPedigreeFunction	0.185071	0.153508	1.000000
Age	-0.042163	0.025748	0.033561
Outcome	0.130548	0.312254	0.173844

	Age	Outcome
Pregnancies	0.544341	0.221898
Glucose	0.263514	0.466581
BloodPressure	0.326740	0.162986
SkinThickness	0.026423	0.175026
Insulin	-0.042163	0.130548
BMI	0.025748	0.312254
DiabetesPedigreeFunction	0.033561	0.173844
Age	1.000000	0.238356
Outcome	0.238356	1.000000

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

```
[11]: <AxesSubplot:>
```



1 Data split

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

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

```
[13]: Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin  BMI  \
0             6      148           72.0      35.000000         0  33.6
1             1       85           66.0      29.000000         0  26.6
2             8      183           64.0      20.536458         0  23.3
3             1       89           66.0      23.000000        94  28.1
4             0      137           40.0      35.000000       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

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

```
[14]: 0    1
      1    0
      2    1
      3    0
      4    1
      Name: Outcome, dtype: int64
```

```
[15]: # 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)
```

```
[16]: 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

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

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

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

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

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

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

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

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

Accuracy: 0.6623376623376623

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

```
[[73 26]
 [26 29]]
```

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

	precision	recall	f1-score	support
0	0.74	0.74	0.74	99
1	0.53	0.53	0.53	55
accuracy			0.66	154
macro avg	0.63	0.63	0.63	154
weighted avg	0.66	0.66	0.66	154

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

```
[24]: # 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)
```

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

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

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

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

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

Accuracy: 0.6818181818181818

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

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

```
[29]: 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

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

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

```
[31]: 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 0 0 0 1 0 1 0 0 0 1 0 1 1 1 0
 1 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 1 1 0 0
 1 1 1 1 0 1 1 1 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 0 0 0 1
 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 1 0 0
 0 1 0 1 1 1]
```

```
[32]: 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]
```

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

Accuracy: 0.6883116883116883


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

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

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

	precision	recall	f1-score	support
0	0.77	0.74	0.75	99
1	0.56	0.60	0.58	55
accuracy			0.69	154
macro avg	0.66	0.67	0.67	154
weighted avg	0.69	0.69	0.69	154

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

```
[36]: # 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)
```

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

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

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

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

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

```
Accuracy: 0.7012987012987013
```

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

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

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

	precision	recall	f1-score	support
0	0.75	0.80	0.77	99
1	0.59	0.53	0.56	55
accuracy			0.70	154
macro avg	0.67	0.66	0.67	154
weighted avg	0.70	0.70	0.70	154

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

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

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

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

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

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

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

Accuracy: 0.6753246753246753

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

```
[[71 28]
 [22 33]]
```

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

	precision	recall	f1-score	support
0	0.76	0.72	0.74	99

	1	0.54	0.60	0.57	55
accuracy				0.68	154
macro avg		0.65	0.66	0.65	154
weighted avg		0.68	0.68	0.68	154

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

```
[48]: # 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)
```

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

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

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

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

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

Accuracy: 0.7597402597402597

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

```
[[84 15]
 [22 33]]
```

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

	precision	recall	f1-score	support
0	0.79	0.85	0.82	99
1	0.69	0.60	0.64	55
accuracy			0.76	154
macro avg	0.74	0.72	0.73	154

weighted avg	0.75	0.76	0.76	154
--------------	------	------	------	-----

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

```
[54]: # 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)
```

```
[55]: print(y_pred)

[0 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 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]
```

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

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

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

Accuracy: 0.7922077922077922

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

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

```
[59]: 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

```
[60]: 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()

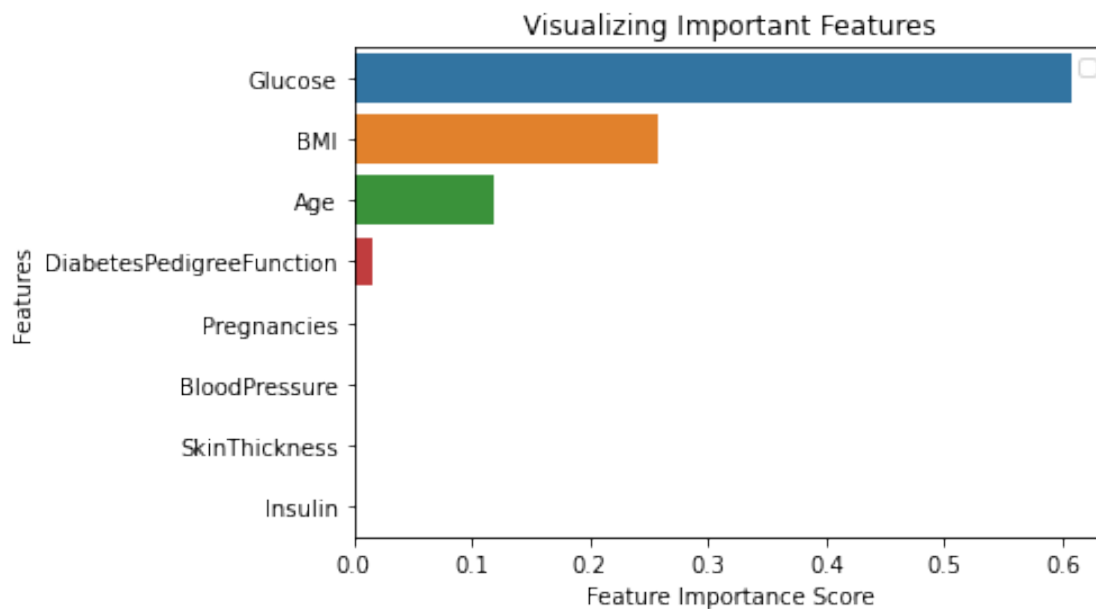
```

```

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.



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

```

[61]: # 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)
```

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

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

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

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

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

Accuracy: 0.7142857142857143

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

```
[[69 30]
 [14 41]]
```

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

	precision	recall	f1-score	support
0	0.83	0.70	0.76	99
1	0.58	0.75	0.65	55
accuracy			0.71	154
macro avg	0.70	0.72	0.70	154
weighted avg	0.74	0.71	0.72	154

3 Accuracy visulization of Decision Tree

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

```
[67]:      Arguments  Accuracy
0      dt_gini_best  0.662338
```

```

1      dt_gini_best_8  0.681818
2      dt_entropy_best 0.688312
3      dt_entropy_best_8 0.701299
4      dt_entropy_random 0.675325
5      dt_entropy_random_8 0.759740
6      dt_entropy_best_3 0.792208
7      dt_entropy_random_3 0.714286

```

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

4 Random Forest

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

4.0.1 n_estimators = 1000, criterion='entropy'

```
[70]: # 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)
```

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

```

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

```

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

```

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

```

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

Accuracy: 0.7987012987012987

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

```

[[87 12]
 [19 36]]

```

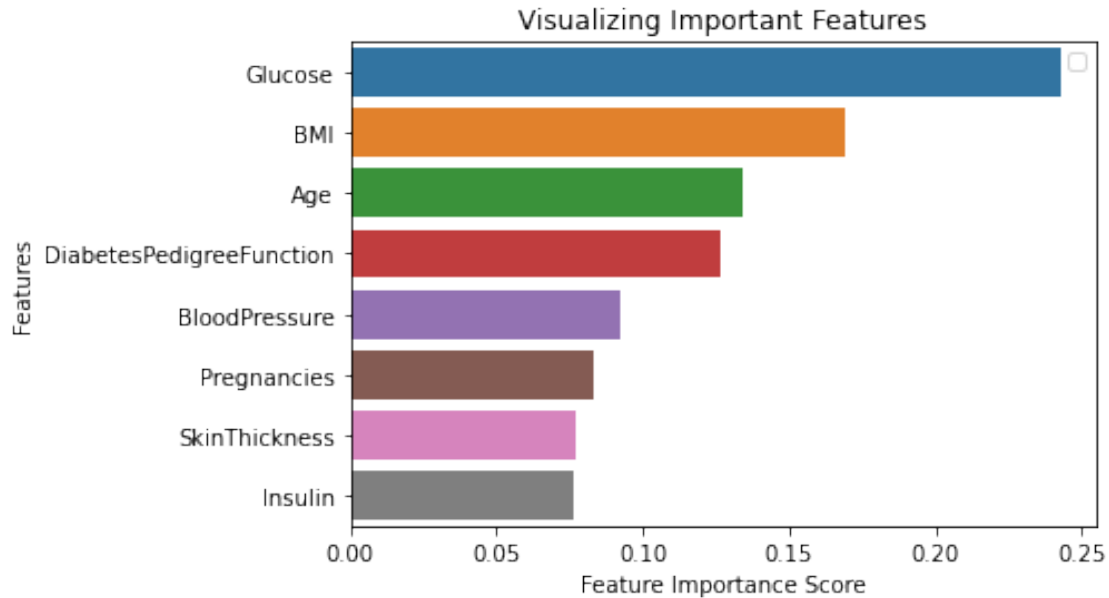
```
[75]: print(metrics.classification_report(y_test, y_pred))
```

	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

```
[76]: 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	0.242477
BMI	0.169410
Age	0.133850
DiabetesPedigreeFunction	0.126468
BloodPressure	0.091976
Pregnancies	0.083135
SkinThickness	0.076731
Insulin	0.075954
dtype:	float64



4.0.2 n_estimators = 100, criterion='entropy'

```
[77]: # 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)
```

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

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

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

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

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

Accuracy: 0.8051948051948052

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

```
[[89 10]
 [20 35]]
```

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

	precision	recall	f1-score	support
0	0.82	0.90	0.86	99
1	0.78	0.64	0.70	55
accuracy			0.81	154
macro avg	0.80	0.77	0.78	154
weighted avg	0.80	0.81	0.80	154

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

```
[83]: # 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)
```

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

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

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

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

```
[86]: 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

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

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

```
[88]: 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

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

```
[89]: # 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)
```

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

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

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

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

```
[92]: 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
```

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

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

```
[94]: 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

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

```
[95]: # 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)
```

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

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

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

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

```
[98]: 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

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

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

```
[100]: 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'

```
[101]: # 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)
```

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

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

```
[103]: 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]
```

```
[104]: 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

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

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

```
[106]: 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

4.0.7 n_estimators = 1000

```
[107]: # 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)
```

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

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

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

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

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

Accuracy: 0.8051948051948052

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

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

```
[112]: 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

```
[113]: # 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)
```

```
[114]: 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 1 0 1 0 1 0 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0
1 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]
```

```
[115]: 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]
```

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

Accuracy: 0.7857142857142857

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

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

```
[118]: 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

4.0.9 n_estimators = 1000, random_state = 42

```
[119]: # 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)
```

```
[120]: print(y_pred)

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

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

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

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

Accuracy: 0.7987012987012987

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

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

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

	precision	recall	f1-score	support
0	0.81	0.89	0.85	99
1	0.76	0.64	0.69	55
accuracy			0.80	154
macro avg	0.79	0.76	0.77	154
weighted avg	0.80	0.80	0.79	154

4.0.10 n_estimators = 100, random_state = 42

```
[125]: # 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)
```

```
[126]: 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 1 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 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

```
[127]: 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 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]
```

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

Accuracy: 0.7922077922077922

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

```
[[85 14]
 [18 37]]
```

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

	precision	recall	f1-score	support
0	0.83	0.86	0.84	99
1	0.73	0.67	0.70	55
accuracy			0.79	154
macro avg	0.78	0.77	0.77	154
weighted avg	0.79	0.79	0.79	154

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

```
[131]: # 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)
```

```
[132]: print(y_pred)

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

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

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

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

Accuracy: 0.7792207792207793

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

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

```
[136]: 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

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

```
[137]: # 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)
```

```
[138]: 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 1 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 1 0 1 0 0 0 1 0 0 0 1 1 0 0 1
 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 0 0 0
 0 0 0 1 1 0]
```

```
[139]: 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 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]
```

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

Accuracy: 0.7922077922077922

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

```
[[85 14]
 [18 37]]
```

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

	precision	recall	f1-score	support
0	0.83	0.86	0.84	99
1	0.73	0.67	0.70	55
accuracy			0.79	154
macro avg	0.78	0.77	0.77	154
weighted avg	0.79	0.79	0.79	154

5 Accuracy visulization of Random Forest

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

```
[143]:
```

	Arguments	Accuracy
0	rf_entropy_1000	0.798701
1	rf_entropy_100	0.805195
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	0.798701
9	rf_gini_100_42	0.792208
10	rf_gini_1000_42_8	0.779221
11	rf_gini_100_42_8	0.792208

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

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