Decision Tree & Random Forest V5

November 19, 2021

Replace All zero features with mean
compute_class_weight

[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

from sklearn.utils.class_weight import compute_class_weight
```

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

```
Before Glucose mean :  120.9
After Glucose mean :  121.7
```

[10]:
```
# replace zero Insulin value with it's mean
print("Before Insulin mean : ",round(df['Insulin'].mean(),1))
df['Insulin'] = df['Insulin'].replace(0, df['Insulin'].mean())
print("After Insulin mean : ",round(df['Insulin'].mean(),1))
```

```
Before Insulin mean :  79.8
After Insulin mean :  118.7
```

[11]:
```
df.describe()
```

[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin \
count	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	121.681605	72.254807	26.606479	118.660163
std	3.369578	30.436016	12.115932	9.631241	93.080358
min	0.000000	44.000000	24.000000	7.000000	14.000000
25%	1.000000	99.750000	64.000000	20.536458	79.799479
50%	3.000000	117.000000	72.000000	23.000000	79.799479
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

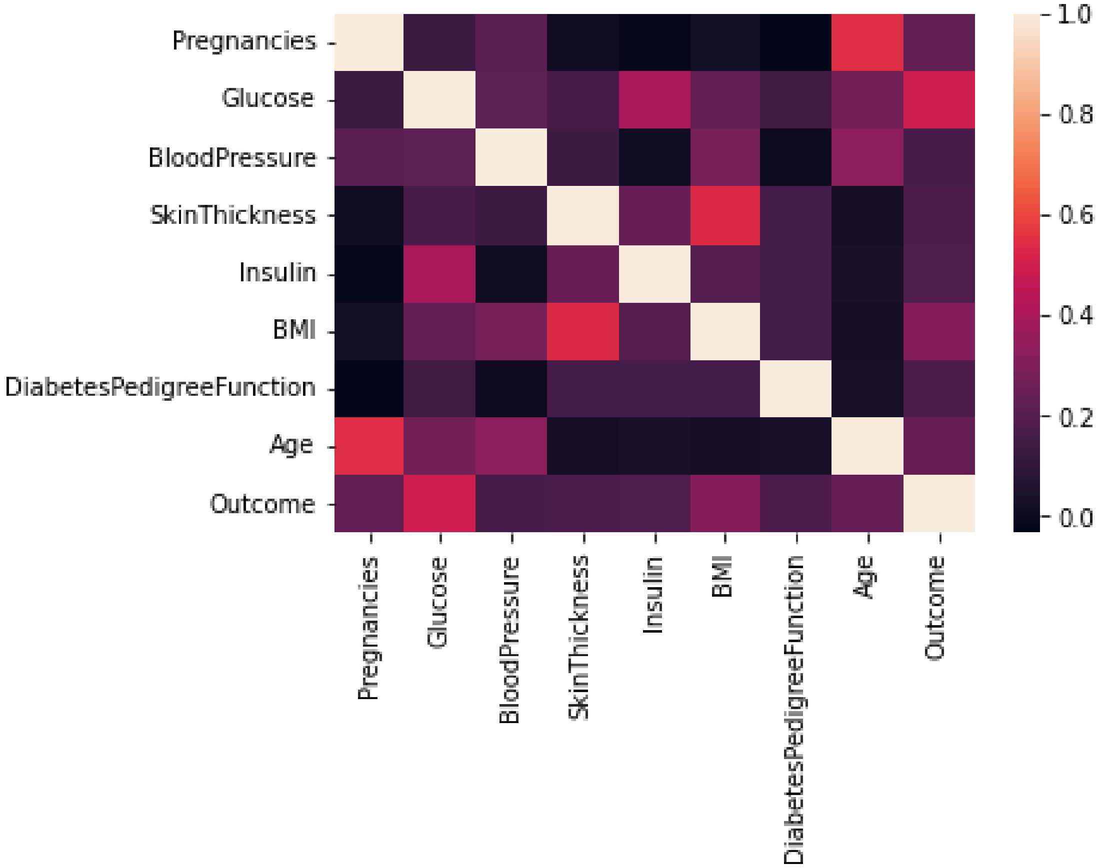
```
25%     27.500000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

[12]:
```
df.corr()
```

[12]:
```
                          Pregnancies   Glucose  BloodPressure  SkinThickness  \
Pregnancies                  1.000000  0.127964       0.208984       0.013376
Glucose                      0.127964  1.000000       0.219666       0.160766
BloodPressure                0.208984  0.219666       1.000000       0.134155
SkinThickness                0.013376  0.160766       0.134155       1.000000
Insulin                     -0.018082  0.396597       0.010926       0.240361
BMI                          0.021546  0.231478       0.281231       0.535703
DiabetesPedigreeFunction    -0.033523  0.137106       0.000371       0.154961
Age                          0.544341  0.266600       0.326740       0.026423
Outcome                      0.221898  0.492908       0.162986       0.175026

                           Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies              -0.018082  0.021546                 -0.033523
Glucose                   0.396597  0.231478                  0.137106
BloodPressure             0.010926  0.281231                  0.000371
SkinThickness             0.240361  0.535703                  0.154961
Insulin                   1.000000  0.189856                  0.157806
BMI                       0.189856  1.000000                  0.153508
DiabetesPedigreeFunction  0.157806  0.153508                  1.000000
Age                       0.038652  0.025748                  0.033561
Outcome                   0.179185  0.312254                  0.173844

                               Age   Outcome
Pregnancies               0.544341  0.221898
Glucose                   0.266600  0.492908
BloodPressure             0.326740  0.162986
SkinThickness             0.026423  0.175026
Insulin                   0.038652  0.179185
BMI                       0.025748  0.312254
DiabetesPedigreeFunction  0.033561  0.173844
Age                       1.000000  0.238356
Outcome                   0.238356  1.000000
```

[13]:
```
sns.heatmap(df.corr())
```

[13]:
```
<AxesSubplot:>
```



1 Data split

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

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

[15]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness     Insulin   BMI  \
0            6    148.0           72.0      35.000000   79.799479  33.6
1            1     85.0           66.0      29.000000   79.799479  26.6
2            8    183.0           64.0      20.536458   79.799479  23.3
3            1     89.0           66.0      23.000000   94.000000  28.1
4            0    137.0           40.0      35.000000  168.000000  43.1

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

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

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

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

[17]:
```
# Data split
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=1)
# x_dev, x_test, y_dev, y_test = train_test_split(x_test, y_test, test_size= 0.
 ↪5)
```

[18]:
```
class_weights = compute_class_weight('balanced', np.unique(y_train),y_train)
```

```
/Users/kamal/opt/anaconda3/lib/python3.8/site-
packages/sklearn/utils/validation.py:70: FutureWarning: Pass classes=[0 1],
y=663    1
712    1
161    0
509    0
305    0
      ..
645    0
715    1
72     1
235    1
37     1
Name: Outcome, Length: 614, dtype: int64 as keyword args. From version 1.0
(renaming of 0.25) passing these as positional arguments will result in an error
  warnings.warn(f"Pass {args_msg} as keyword args. From version "
```

[19]:
```
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

[20]:
```
accuracy = {}
```

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

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

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

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

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

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

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

```
Accuracy: 0.6688311688311688
```

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

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

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

```
              precision    recall  f1-score   support

           0       0.74      0.74      0.74        99
           1       0.54      0.55      0.54        55

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

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

[27]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=8,␣
↪class_weight='balanced')
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

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

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

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

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

[30]:
```
accuracy["dt_gini_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7337662337662337
```

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

```
[[72 27]
 [14 41]]
```

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

```
              precision    recall  f1-score   support

           0       0.84      0.73      0.78        99
           1       0.60      0.75      0.67        55

    accuracy                           0.73       154
   macro avg       0.72      0.74      0.72       154
weighted avg       0.75      0.73      0.74       154
```

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

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

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

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

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

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

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

```
Accuracy: 0.6883116883116883
```

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

```
[[75 24]
 [24 31]]
```

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

```
              precision    recall  f1-score   support

           0       0.76      0.76      0.76        99
           1       0.56      0.56      0.56        55

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

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

[39]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best", max_depth=8,␣
↪class_weight='balanced')
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

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

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

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

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

[42]:
```
accuracy["dt_entropy_best_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6818181818181818
```

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

```
[[68 31]
 [18 37]]
```

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

```
              precision    recall  f1-score   support

           0       0.79      0.69      0.74        99
           1       0.54      0.67      0.60        55

    accuracy                           0.68       154
   macro avg       0.67      0.68      0.67       154
weighted avg       0.70      0.68      0.69       154
```

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

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

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

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

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

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

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

```
Accuracy: 0.6948051948051948
```

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

```
[[74 25]
 [22 33]]
```

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

```
              precision    recall  f1-score   support

           0       0.77      0.75      0.76        99
           1       0.57      0.60      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
```

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

[51]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="random",␣
 ↪max_depth=8, class_weight='balanced')
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

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

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

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

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

[54]:
```
accuracy["dt_entropy_random_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6688311688311688
```

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

```
[[63 36]
 [15 40]]
```

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

```
              precision    recall  f1-score   support

           0       0.81      0.64      0.71        99
           1       0.53      0.73      0.61        55

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

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

[57]:
```python
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="best", max_depth=3,␣
↪class_weight='balanced')
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[58]:
```python
print(y_pred)
```

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

[59]:
```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]
```

[60]:
```python
accuracy["dt_entropy_best_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6753246753246753
```

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

```
[[54 45]
 [ 5 50]]
```

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

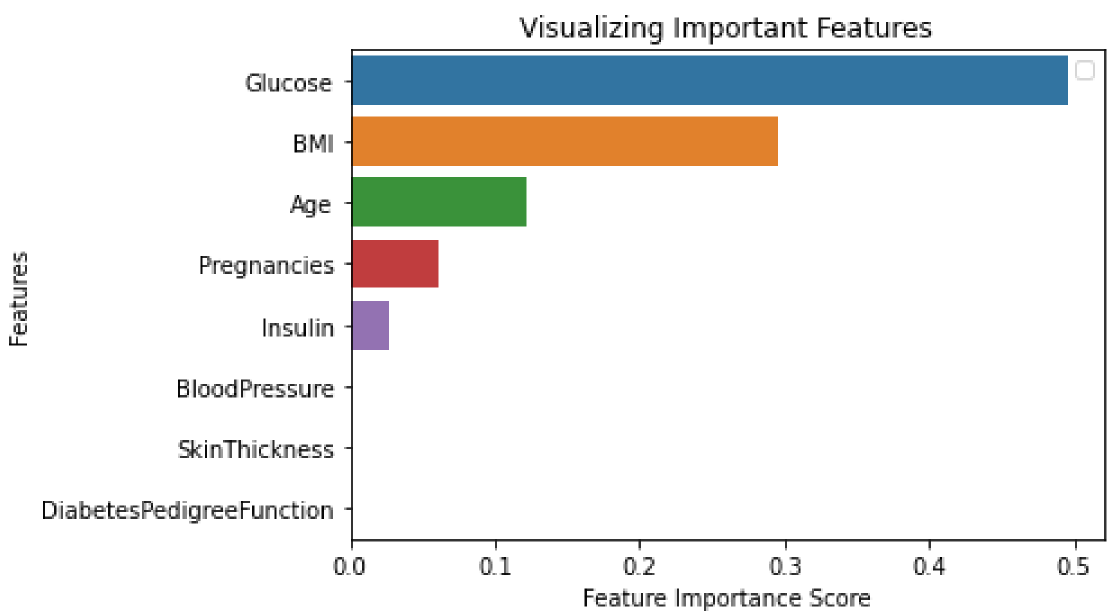
```
              precision    recall  f1-score   support

           0       0.92      0.55      0.68        99
           1       0.53      0.91      0.67        55

    accuracy                           0.68       154
   macro avg       0.72      0.73      0.68       154
weighted avg       0.78      0.68      0.68       154
```

[63]:
```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                     0.495224
BMI                         0.296275
Age                         0.121487
Pregnancies                 0.060543
Insulin                     0.026471
BloodPressure               0.000000
SkinThickness               0.000000
DiabetesPedigreeFunction    0.000000
dtype: float64
```

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



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

[64]:
```
# Define and build model
clf = DecisionTreeClassifier(criterion="entropy", splitter="random",␣
↪max_depth=3, class_weight='balanced')
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[65]:
```python
print(y_pred)
```

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

[66]:
```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]
```

[67]:
```python
accuracy["dt_entropy_random_3"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7597402597402597
```

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

```
[[71 28]
 [ 9 46]]
```

[69]:
```python
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.72      0.79        99
           1       0.62      0.84      0.71        55

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

3 Accuracy visulization of Decision Tree

[70]:
```python
accuracy_df_dt = pd.DataFrame(list(zip(accuracy.keys(), accuracy.values())),␣
 ↪columns =['Arguments', 'Accuracy'])
accuracy_df_dt
```

[70]:

	Arguments	Accuracy
0	dt_gini_best	0.668831
1	dt_gini_best_8	0.733766
2	dt_entropy_best	0.688312
3	dt_entropy_best_8	0.681818

```
4     dt_entropy_random  0.694805
5   dt_entropy_random_8  0.668831
6     dt_entropy_best_3  0.675325
7   dt_entropy_random_3  0.759740
```

[71]:
```
fig = px.bar(accuracy_df_dt, x='Arguments', y='Accuracy')
fig.show()
```

4 Random Forest

[72]:
```
accuracy_rf = {}
```

4.0.1 n_estimators = 1000, criterion='entropy'

[73]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, criterion='entropy',␣
 ↪class_weight='balanced')
# 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)
```

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

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

[75]:
```
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]
```

[76]:
```
accuracy_rf["rf_entropy_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

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

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

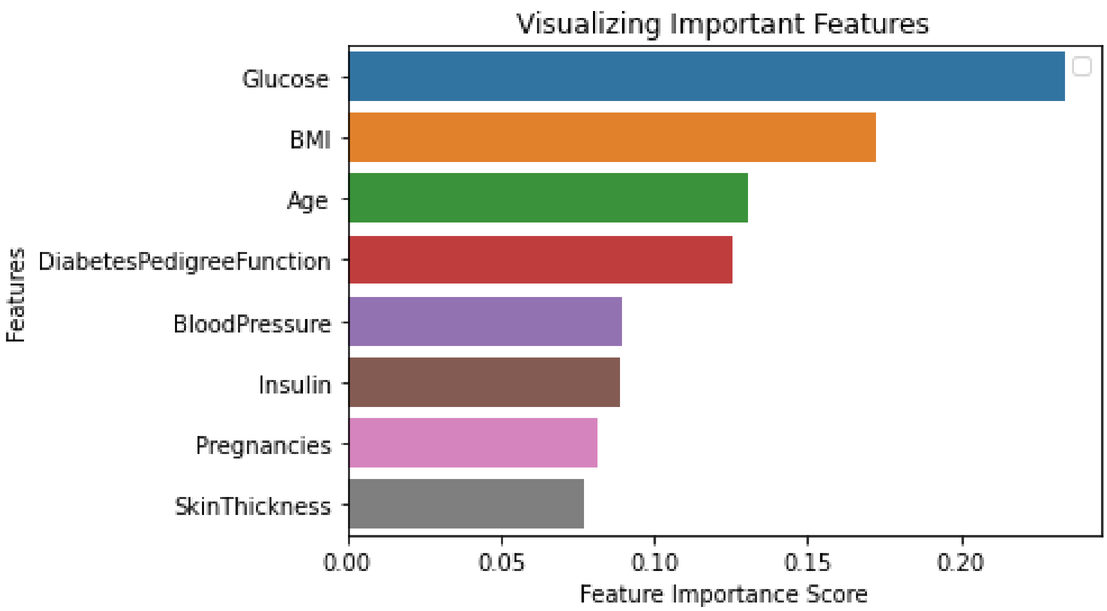
[78]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              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
```

[79]:
```
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                     0.233633
BMI                         0.172378
Age                         0.130843
DiabetesPedigreeFunction    0.125787
BloodPressure               0.089786
Insulin                     0.088766
Pregnancies                 0.081497
SkinThickness               0.077311
dtype: float64
```



4.0.2 n_estimators = 100, criterion='entropy'

[80]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, criterion='entropy',␣
 ↪class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[83]:
```
accuracy_rf["rf_entropy_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7792207792207793
```

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

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

[85]:
```
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
```

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

[86]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42,␣
 ↪criterion='entropy', class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[89]:
```
accuracy_rf["rf_entropy_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

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

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

[91]:
```
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
```

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

[92]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =
 ↪8, criterion='entropy', class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[95]:
```
accuracy_rf["rf_entropy_100_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8051948051948052
```

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

```
[[79 20]
 [10 45]]
```

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

```
              precision    recall  f1-score   support

           0       0.89      0.80      0.84        99
           1       0.69      0.82      0.75        55

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

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

[98]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth =␣
↪8, criterion='entropy', class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[101]:
```
accuracy_rf["rf_entropy_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8116883116883117
```

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

```
[[81 18]
 [11 44]]
```

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

```
              precision    recall  f1-score   support

           0       0.88      0.82      0.85        99
           1       0.71      0.80      0.75        55

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

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

[104]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
 ↪8, criterion='entropy', class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[107]:
```
accuracy_rf["rf_entropy_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8051948051948052
```

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

```
[[79 20]
 [10 45]]
```

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

```
              precision    recall  f1-score   support

           0       0.89      0.80      0.84        99
           1       0.69      0.82      0.75        55

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

4.0.7 n_estimators = 1000

[110]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[113]:
```
accuracy_rf["rf_gini_1000"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

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

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

[115]:
```
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
```

4.0.8 n_estimators = 100

[116]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[119]:
```
accuracy_rf["rf_gini_100"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

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

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

[121]:
```
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
```

4.0.9 n_estimators = 1000, random_state = 42

[122]:
```python
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42,␣
 ↪class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

[124]:
```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]
```

[125]:
```python
accuracy_rf["rf_gini_1000_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7922077922077922
```

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

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

[127]:
```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
```

4.0.10 n_estimators = 100, random_state = 42

[128]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
↪8, class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[131]:
```
accuracy_rf["rf_gini_100_42"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8181818181818182
```

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

```
[[83 16]
 [12 43]]
```

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

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86        99
           1       0.73      0.78      0.75        55

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

```
weighted avg       0.82      0.82      0.82       154
```

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

[134]:
```
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42, max_depth =
 ↪8, class_weight='balanced')
# Train the model on training data
rf.fit(x_train,y_train)
# Use the forest's predict method on the test data
y_pred = rf.predict(x_test)
```

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

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

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

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

[137]:
```
accuracy_rf["rf_gini_1000_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8051948051948052
```

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

```
[[81 18]
 [12 43]]
```

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

```
              precision    recall  f1-score   support

           0       0.87      0.82      0.84        99
           1       0.70      0.78      0.74        55

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

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

[140]:
```
# Instantiate model with 100 decision trees
rf = RandomForestClassifier(n_estimators = 100, random_state = 42, max_depth =␣
 ↪8, class_weight='balanced')
# 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)
```

[141]:
```
print(y_pred)
```

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

[142]:
```
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]
```

[143]:
```
accuracy_rf["rf_gini_100_42_8"] = metrics.accuracy_score(y_test, y_pred);
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8181818181818182
```

[144]:
```
print(metrics.confusion_matrix(y_test, y_pred))
```

```
[[83 16]
 [12 43]]
```

[145]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.86        99
           1       0.73      0.78      0.75        55

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

5 Accuracy visulization of Random Forest

[146]:
```
accuracy_df_rf = pd.DataFrame(list(zip(accuracy_rf.keys(), accuracy_rf.
 ↪values())), columns =['Arguments', 'Accuracy'])
accuracy_df_rf
```

[146]:

	Arguments	Accuracy
0	rf_entropy_1000	0.792208
1	rf_entropy_100	0.779221
2	rf_entropy_1000_42	0.792208
3	rf_entropy_100_42	0.805195
4	rf_entropy_1000_42_8	0.811688
5	rf_entropy_100_42_8	0.805195
6	rf_gini_1000	0.792208
7	rf_gini_100	0.792208
8	rf_gini_1000_42	0.792208
9	rf_gini_100_42	0.818182
10	rf_gini_1000_42_8	0.805195
11	rf_gini_100_42_8	0.818182

[147]:
```
fig = px.bar(accuracy_df_rf, x='Arguments', y='Accuracy')
fig.show()
```

[148]:
```
accuracy_df = pd.concat([accuracy_df_dt, accuracy_df_rf])
accuracy_df['Accuracy'] = round(accuracy_df['Accuracy'] * 100, 2)
fig = px.bar(accuracy_df, x='Arguments', y='Accuracy')
print(accuracy_df['Accuracy'].max())
fig.show()
```

```
81.82
```

[ ]:
```

```