

**Introduction to Machine learning**

**Project Phase 2**

**Team 8**

|  |  |
| --- | --- |
| Name | ID |
| AbdulRaouf Monir Kamal Mahmoud | 19P4442 |
| Abdallah Amin Fahmy | 20p5880 |
| Michael Joseph Adeeb | 1901075 |

**Submission Date: 30/12/2023**

**Submitted to: Prof. Dr. Mahmoud Khalil**

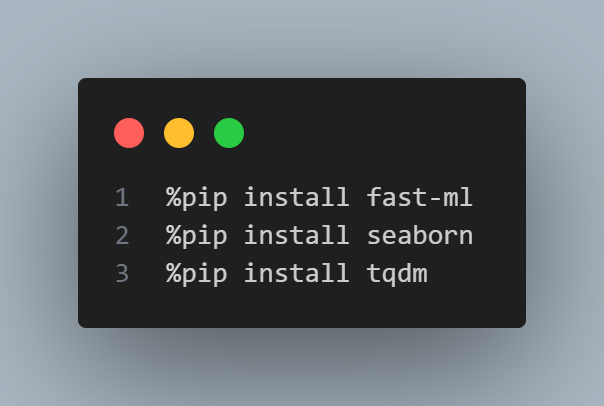
**Eng. Omar Elessawy**

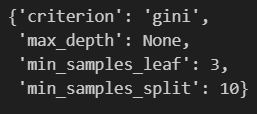
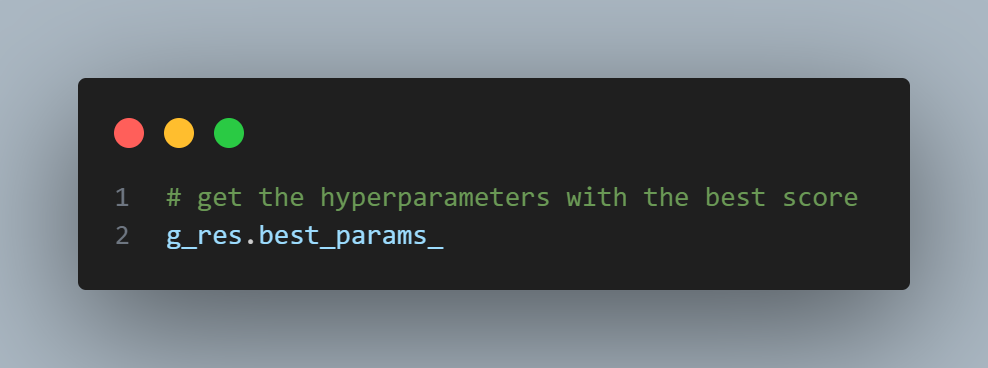
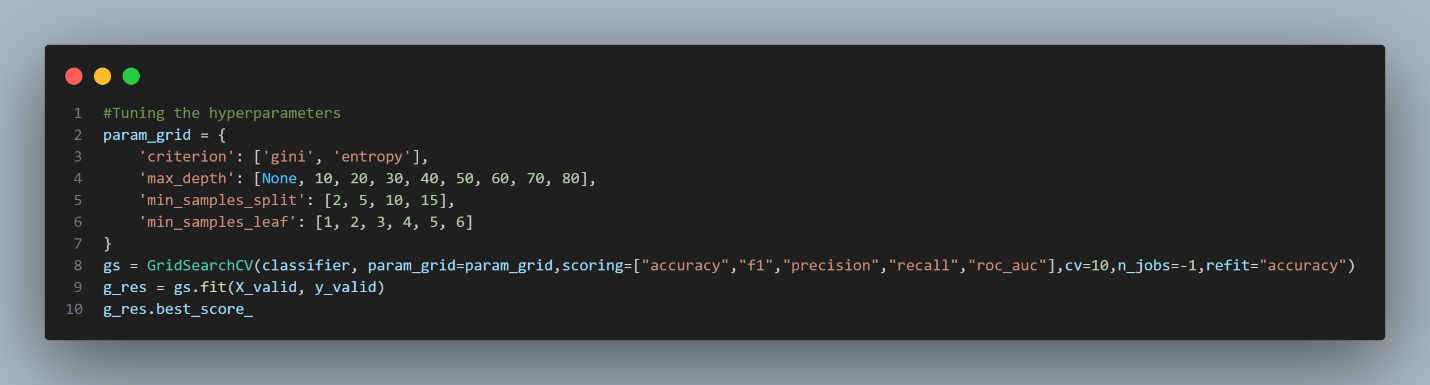
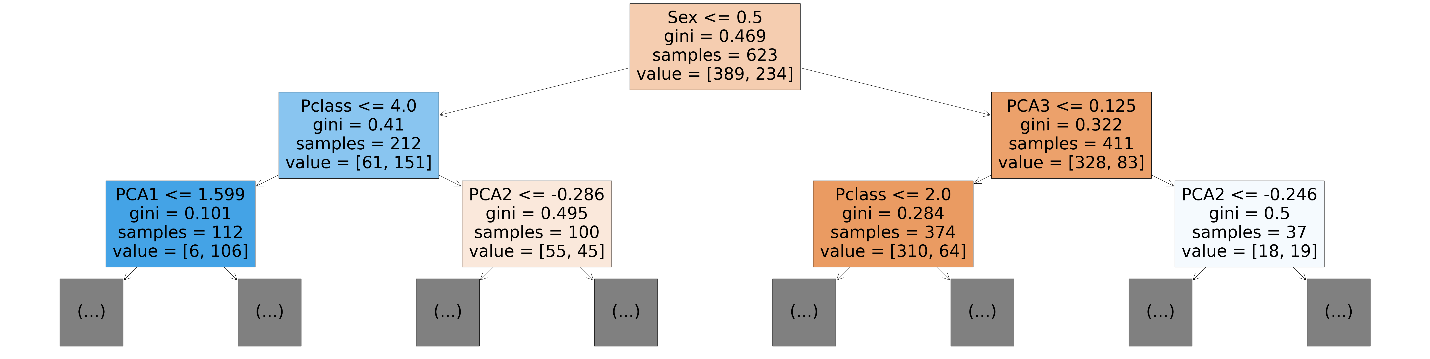
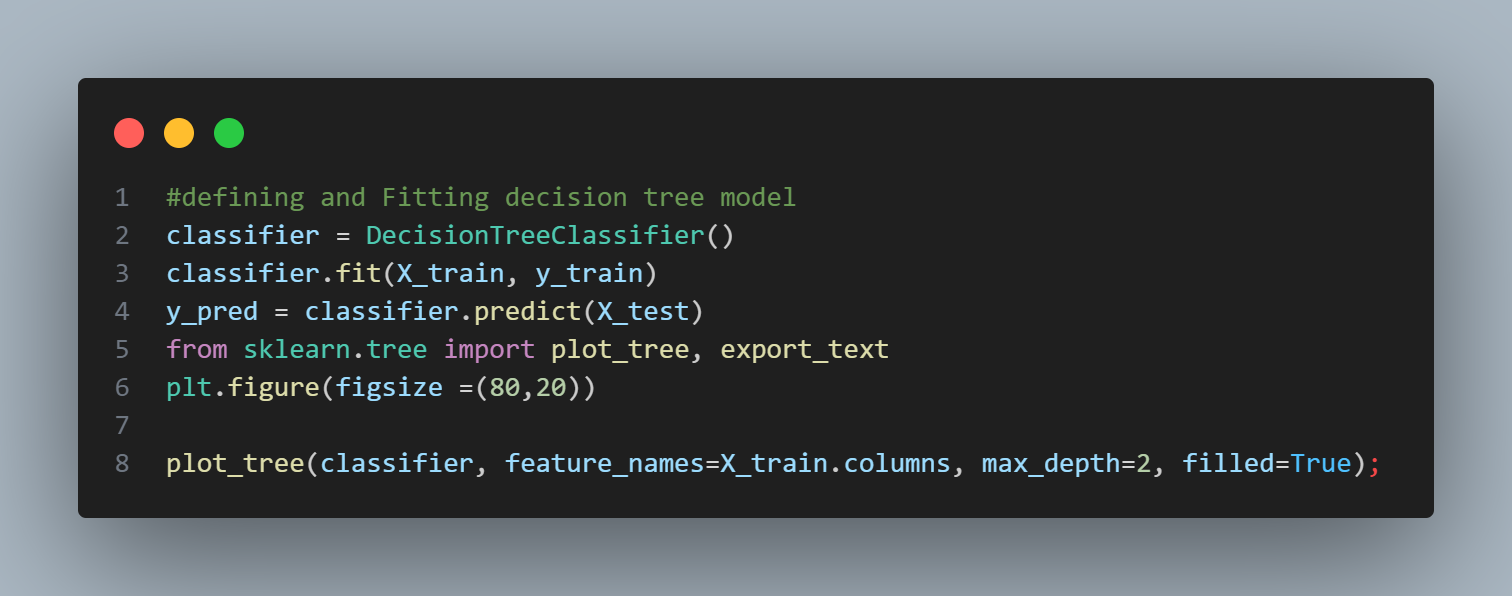
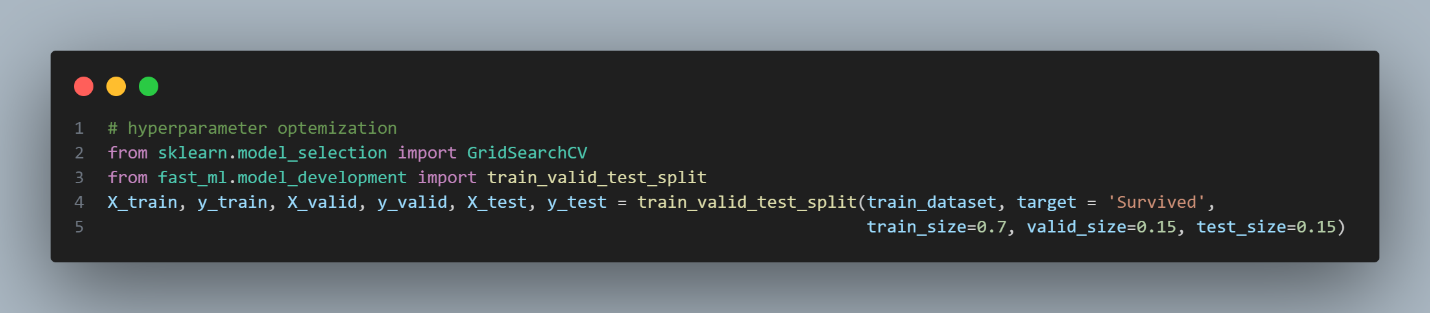
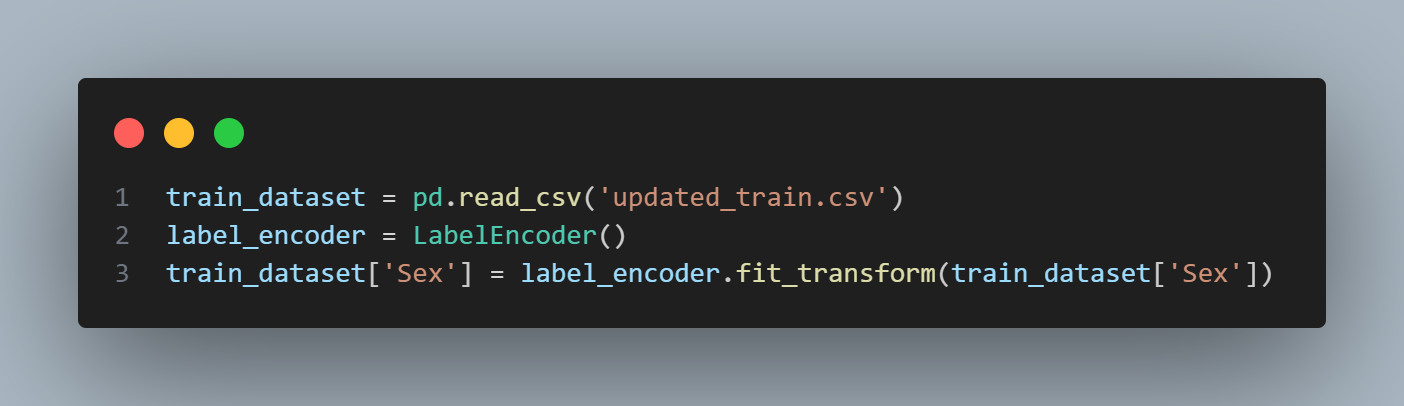
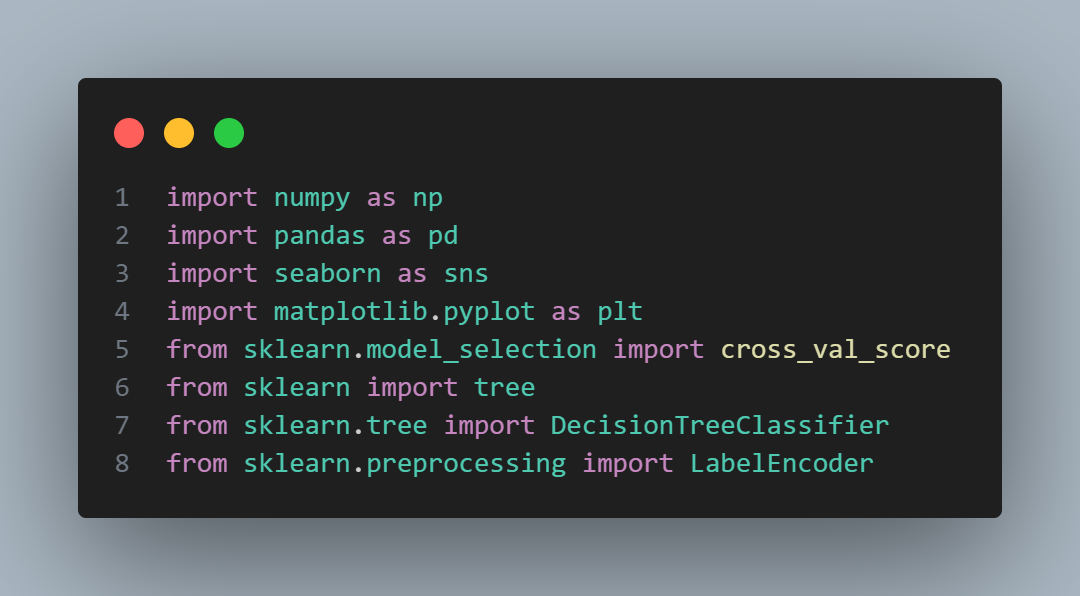
**Decision Trees**

**Steps on how to run project**

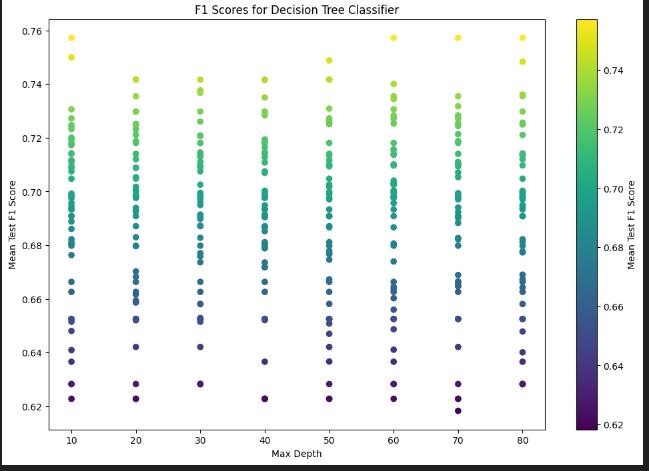
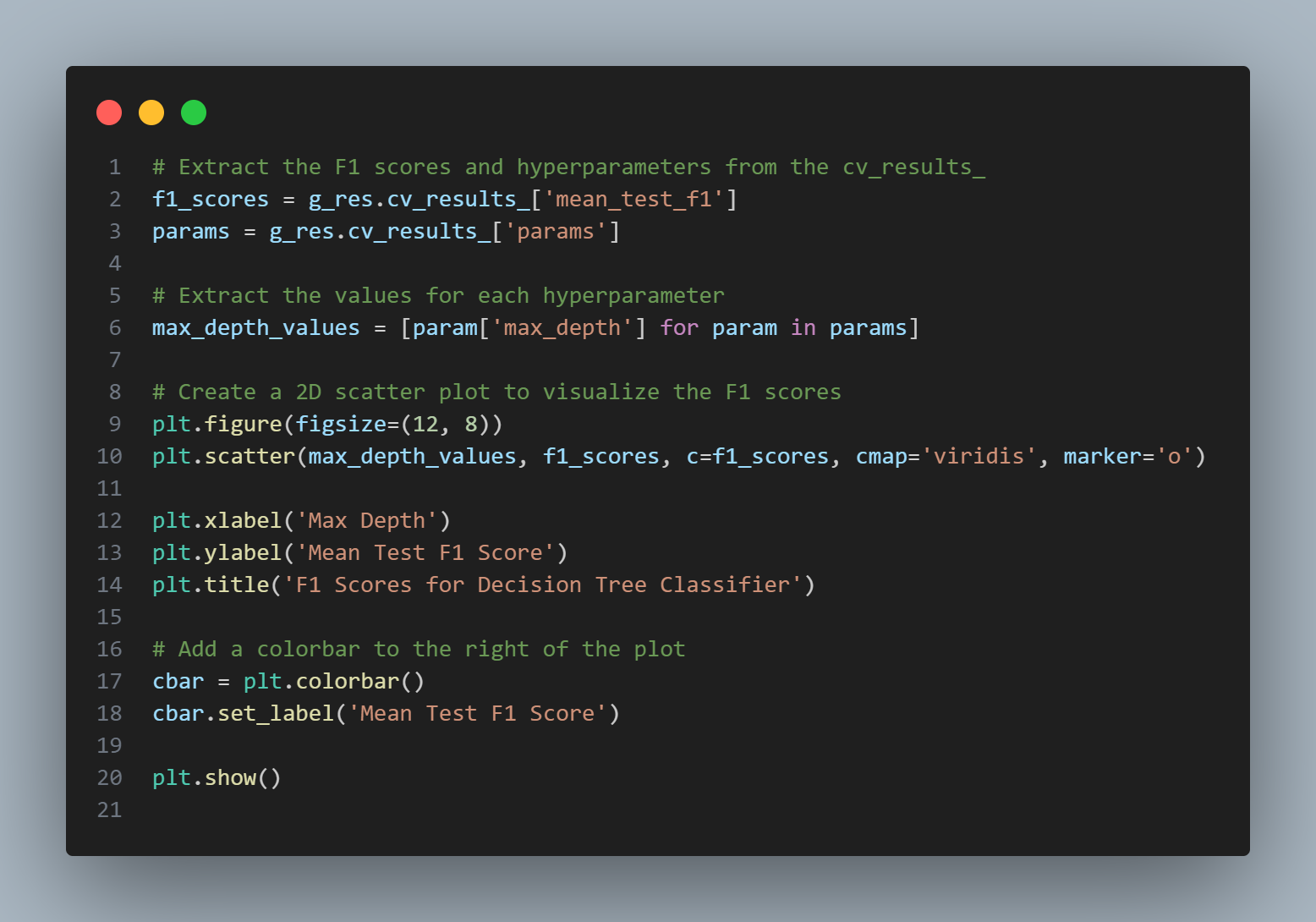
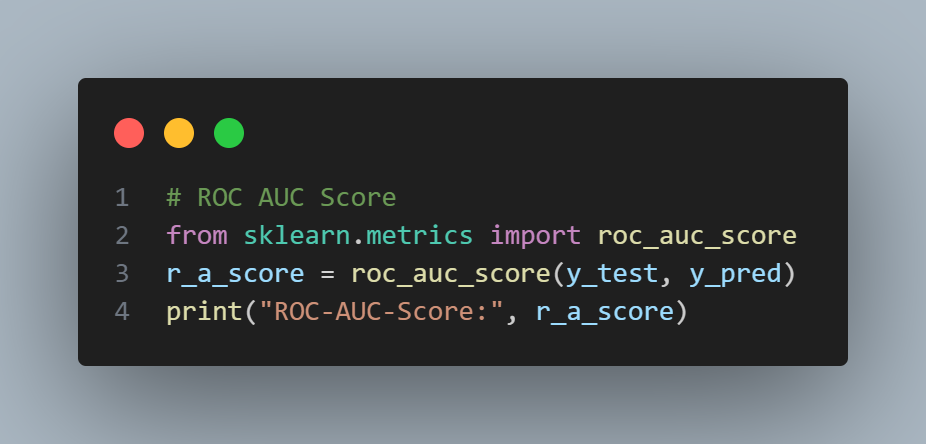
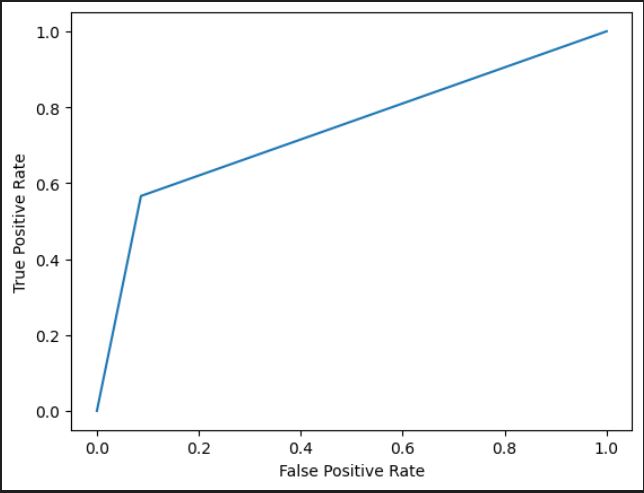
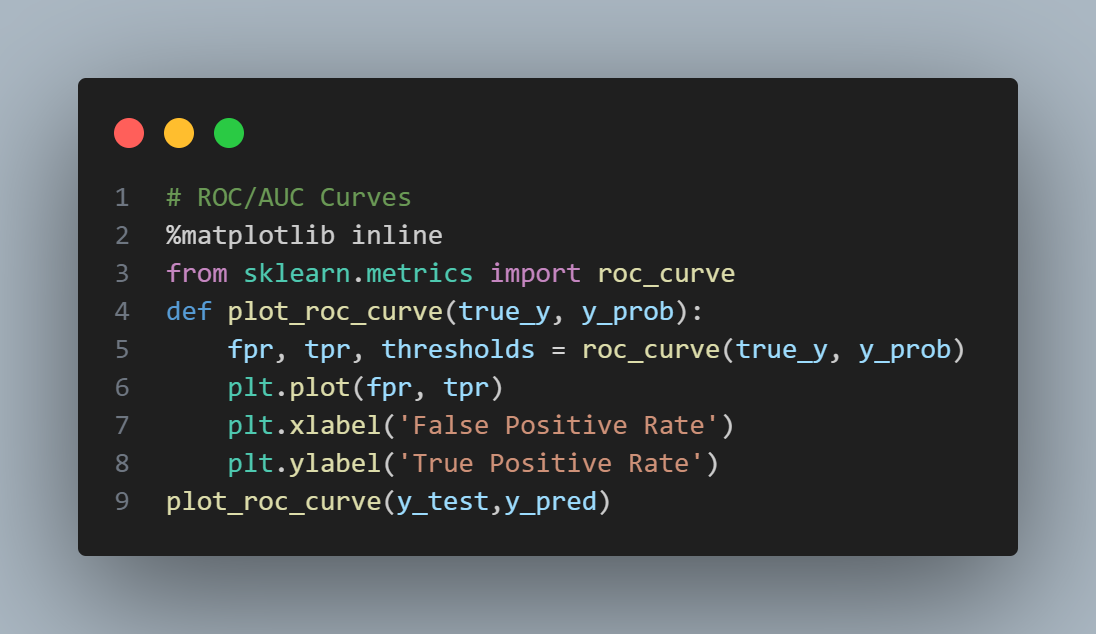
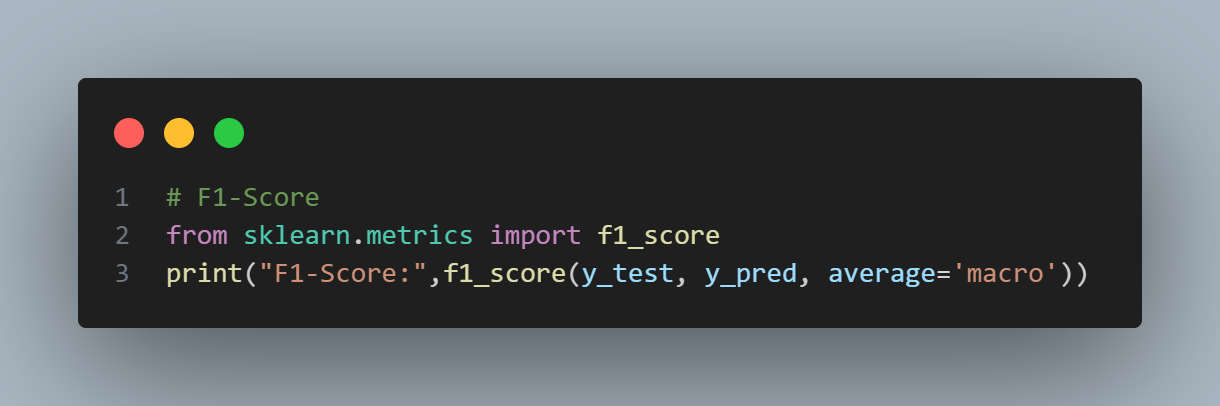
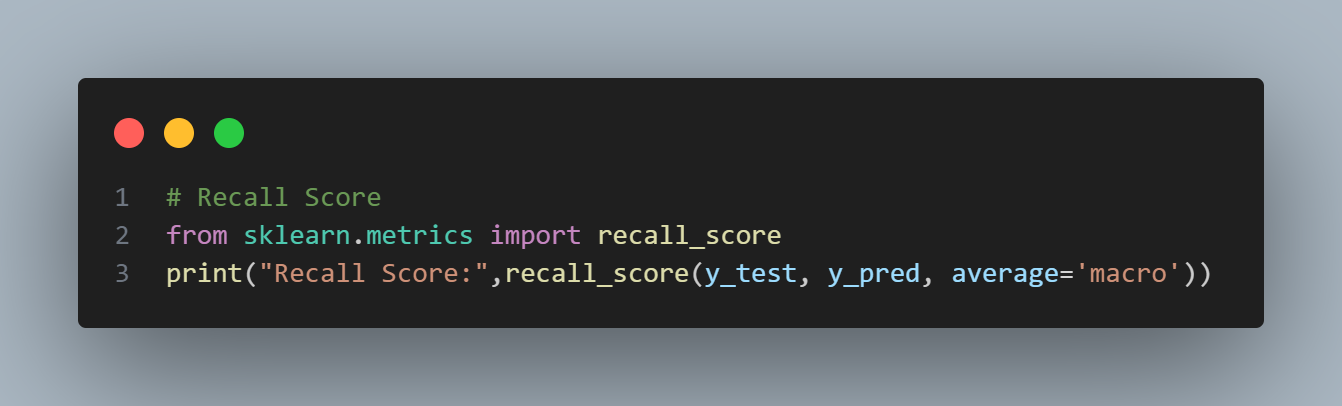
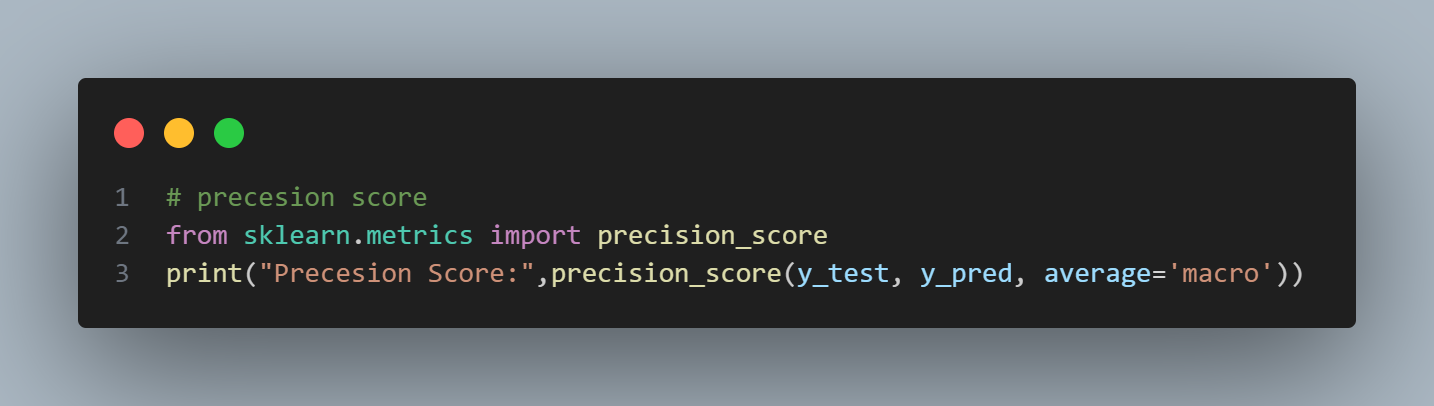
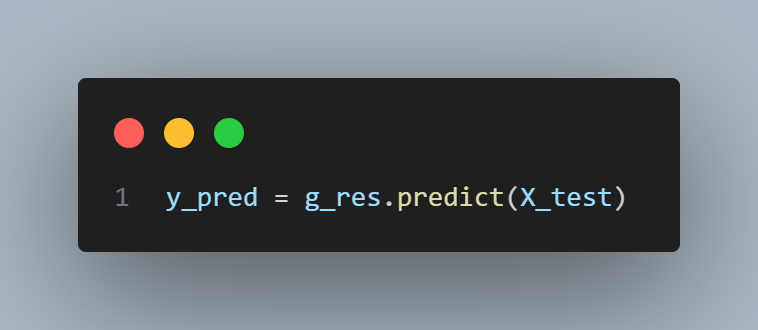
1. Open Decision tree.ipynb and then run the first cell to import all the required libraries
2. Project is ready to run!

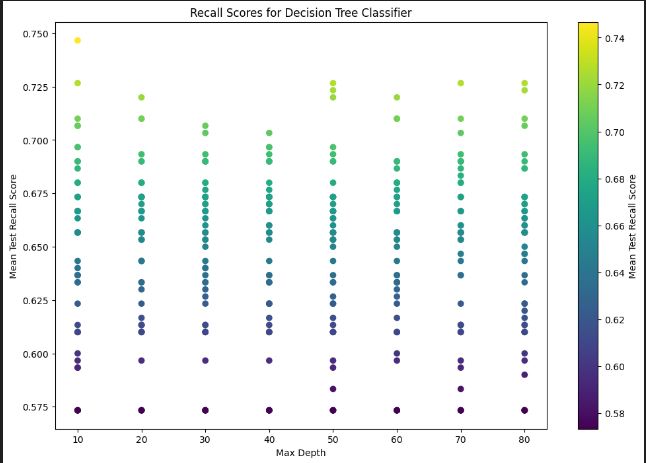
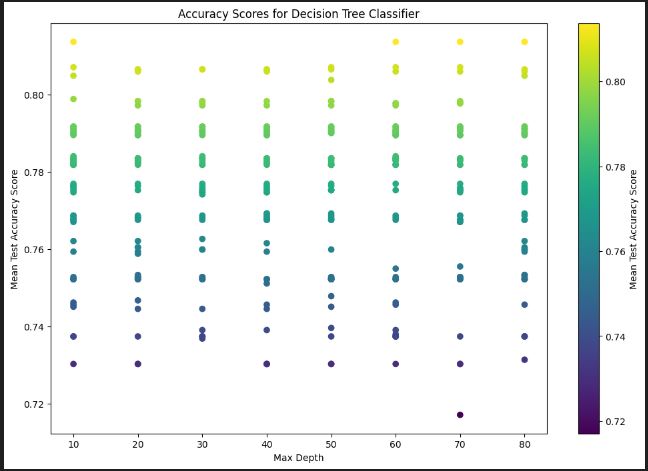
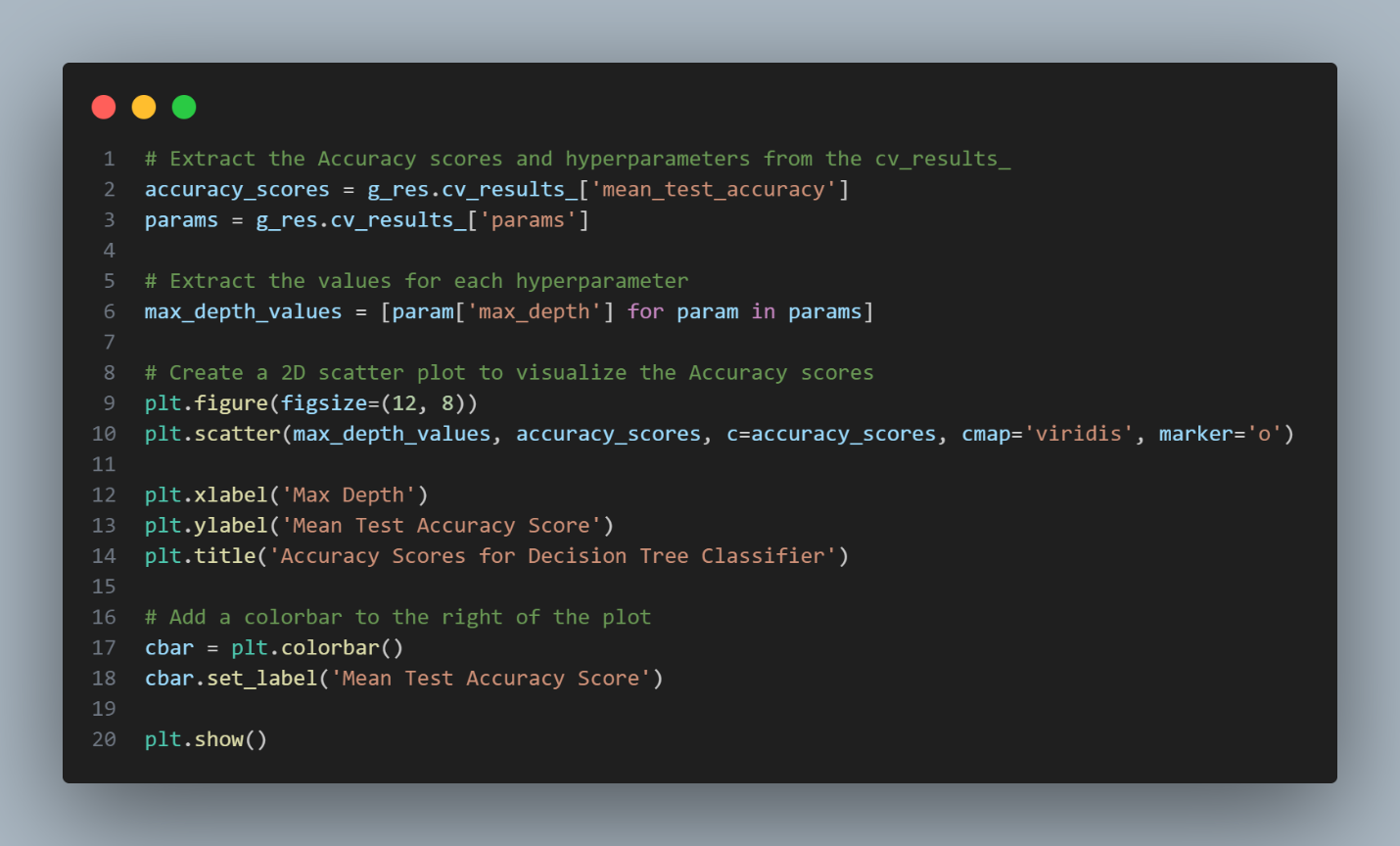
**Screenshots of the Code including the output**

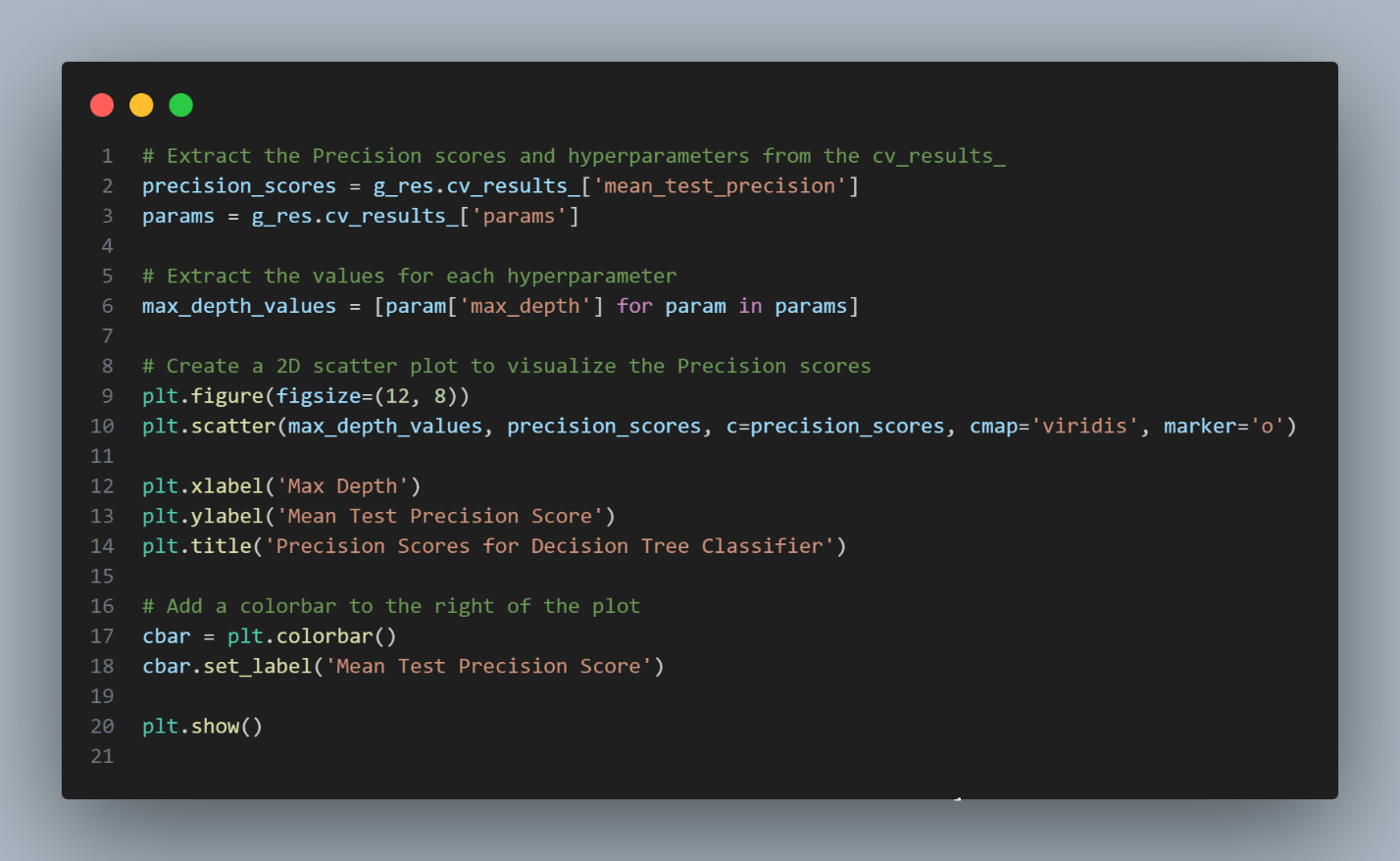


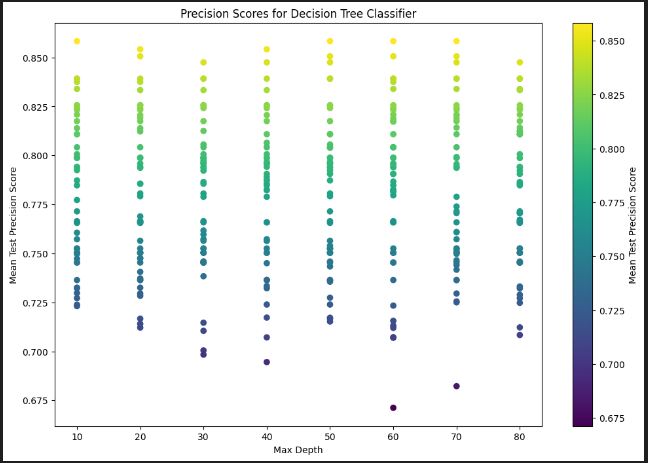


These are the optimal hyperparameters for the Decision tree classifier









**Multilayer Perceptron**

**Steps on how to run project**

1. Open Multilayer-Perceptron.ipynb and then run the first cell to import all the required libraries
2. Project is ready to run!

**Code Screenshots and Details**

%pip install fast-ml

%pip install scikit-learn

%pip install numpy

%pip install pandas

%pip install seaborn

First, we install all the required libraries for initializing and training our multilayer perceptron model.

import pandas as pd

from sklearn.preprocessing import LabelEncoder

train\_dataset = pd.read\_csv('../updated\_train.csv')

label\_encoder = LabelEncoder()

train\_dataset['Sex'] = label\_encoder.fit\_transform(train\_dataset['Sex'])

X  = train\_dataset.drop("Survived",*axis*=1)

y = train\_dataset["Survived"]

Then we import our preprocessed dataset and encode the “Sex” Column as Multilayer perceptron only accepts numerical features. Afterwards, we split the data into input features (X) and the output (y)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,*train\_size*=0.85,*test\_size*=0.15)

Then we split the input and output features into training and testing sets, but this time we won’t use the fast-ml library, as there is no need to create a validation set, which we will see as progress on, will be generated by the multilayer perceptron model via the validation\_factor parameter.

def generate\_hidden\_layer\_sizes(*num\_sizes*, *min\_nodes*, *max\_nodes*):

    return [(np.random.randint(*min\_nodes*, *max\_nodes*),) for \_ in range(*num\_sizes*)]

This is a method we will use to generate a list of different hidden\_layer\_sizes ( the number of hidden nodes per layer ) that will be supplied to GridSearchCV to determine the optimal hyperparameter as per the supplied parameter grid.

import numpy as np

num\_hidden\_sizes = 10

min\_nodes\_per\_layer = 50

max\_nodes\_per\_layer = 250

hidden\_layer\_sizes = generate\_hidden\_layer\_sizes(num\_hidden\_sizes, min\_nodes\_per\_layer, max\_nodes\_per\_layer)

Now, we will call the generate\_hidden\_layer\_sizes method to generate different hidden layer sizes as described above

from sklearn.neural\_network import MLPClassifier

from sklearn.model\_selection import GridSearchCV

param\_grid = {

    "activation": ["identity","logistic","tanh","relu",],

    "solver": ["lbfgs","sgd","adam",],

    "learning\_rate": ["constant","invscaling","adaptive",],

    "early\_stopping": [True, False,],

    "validation\_fraction": [0.15,],

    "hidden\_layer\_sizes": hidden\_layer\_sizes

}

mlp = GridSearchCV(

    MLPClassifier(),

*param\_grid*=param\_grid,

*scoring*=["accuracy","f1","precision","recall","roc\_auc"],

*n\_jobs*=-1,

*cv*=2,

*refit*="f1",

*verbose*=True

      )

mlp.fit(X\_train,y\_train)

mlp.best\_score\_

Finally, we will call the GridSearchCV object to supply it with a multilayer perceptron object, and parameter grid that act our hyperparameters with almost each and every possible value of each hyperparameter, and only two folds for simplicity, and the refit is going to be based primarily on F1-score

Then we will print our best achieved score, which is approximately 75%



y\_pred = mlp.predict(X\_test)

In this line of code, we just create a variable that will be used for calculating precision, recall, f1-score, and roc/auc measures

A screen shot of a computer code

Description automatically generated

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve

def plot\_roc\_curve(*true\_y*, *y\_prob*):

    fpr, tpr, thresholds = roc\_curve(*true\_y*, *y\_prob*)

    plt.plot(fpr, tpr)

    plt.xlabel('False Positive Rate')

    plt.ylabel('True Positive Rate')

Above is a function to plot roc/auc curve

from sklearn.metrics import roc\_auc\_score

plot\_roc\_curve(y\_test, y\_pred)

plt.show()

print(f'Multilayer perceptron AUC score: {roc\_auc\_score(y\_test, y\_pred)}')

A blue line graph with numbers

Description automatically generated



import matplotlib.pyplot as plt

iterations = [i for i in range(1,mlp.best\_estimator\_.n\_iter\_+1)]

loss\_curve = mlp.best\_estimator\_.loss\_curve\_

plt.plot(iterations,loss\_curve)

plt.xlabel("Iteration Number")

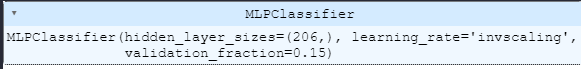
plt.ylabel("Loss Function")

plt.title("Loss Curve")

This is a function to visualize the loss curve of our best estimator(model)

A graph with a line

Description automatically generated



These are the properties of the best estimator as per the parameters in parameter grid, and are the **optimal hyperparameters**

Now we will visualize the fit against accuracy metric (precision, recall, f1-score, and roc/auc)

fits = [fit for fit in range(1,73)]

accuracy = mlp.cv\_results\_["mean\_test\_precision"][::10]

plt.plot(fits,accuracy)

plt.xlabel("Fit")

plt.ylabel("Precision")

plt.title("Precision")

A graph with blue lines

Description automatically generated

The code is similar for all other accuracy change metrics, note that many of the fits have been removed for better visualization of the accuracy change

A graph with blue lines

Description automatically generated

A graph with blue lines

Description automatically generated A graph with blue lines

Description automatically generated