DATS 6103- Intro to Datamining
Individual Final Report
Group-1
Mahtab Barkhordarian

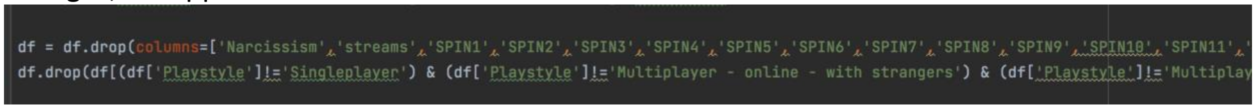
1.Introduction:
These days people suffer from anxiety a lot and they even don't know what makes it happened. As we know these days people are so into online gaming. People do online gaming regardless of their education level, age, gender and others. In this dataset the impact of online gaming on Generalized Anxiety is analyzed. The information in this dataset is including the age , gender, education and 50 other variables which may have the impact on Generalized Anxiety Disorder(GAD_T).

By models, the GAD_T will be predicted and we can see which variables have the most impact on it. The project is demonstrated by using three machine learning algorithms: Random Forest Classifier, Decision Tree and Support Vector Machine respectively and develop a GUI based application to display the end-to-end modelling.

2.Personal Contribution:

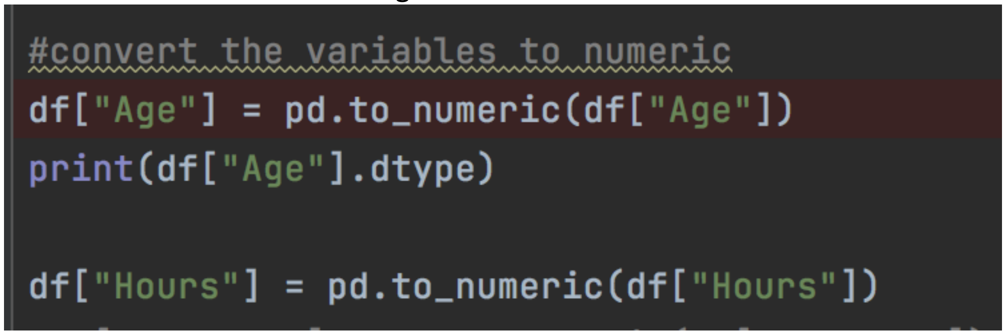
Code:
Pre-processing:

1- At the beginning, I dropped 'accept', because the whole column was either accept or NA. then dropped 'highestleague' because the whole column is NA. dropped listed columns here since after plotting the heatmap and realized that if two variables have the high co relation we can choose either of them. For that I put them one by one at the end to model to see if they change the accuracy percentage and if they didn't make any changes, I dropped them here.



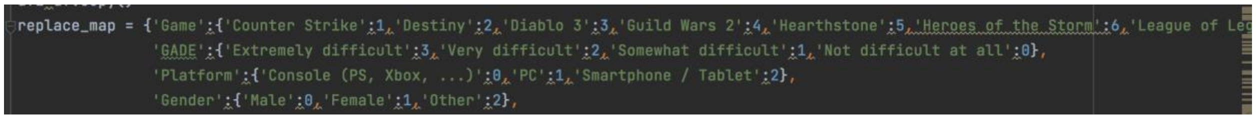
```
df = df.drop(columns=['Narcissism','streams','SPIN1','SPIN2','SPIN3','SPIN4','SPIN5','SPIN6','SPIN7','SPIN8','SPIN9','SPIN10','SPIN11','
df.drop(df[(df['Playstyle']!='Singleplayer') & (df['Playstyle']!='Multiplayer - online - with strangers') & (df['Playstyle']!='Multiplay
```

2- Convert some variables like 'Age' and 'Hours' to numeric



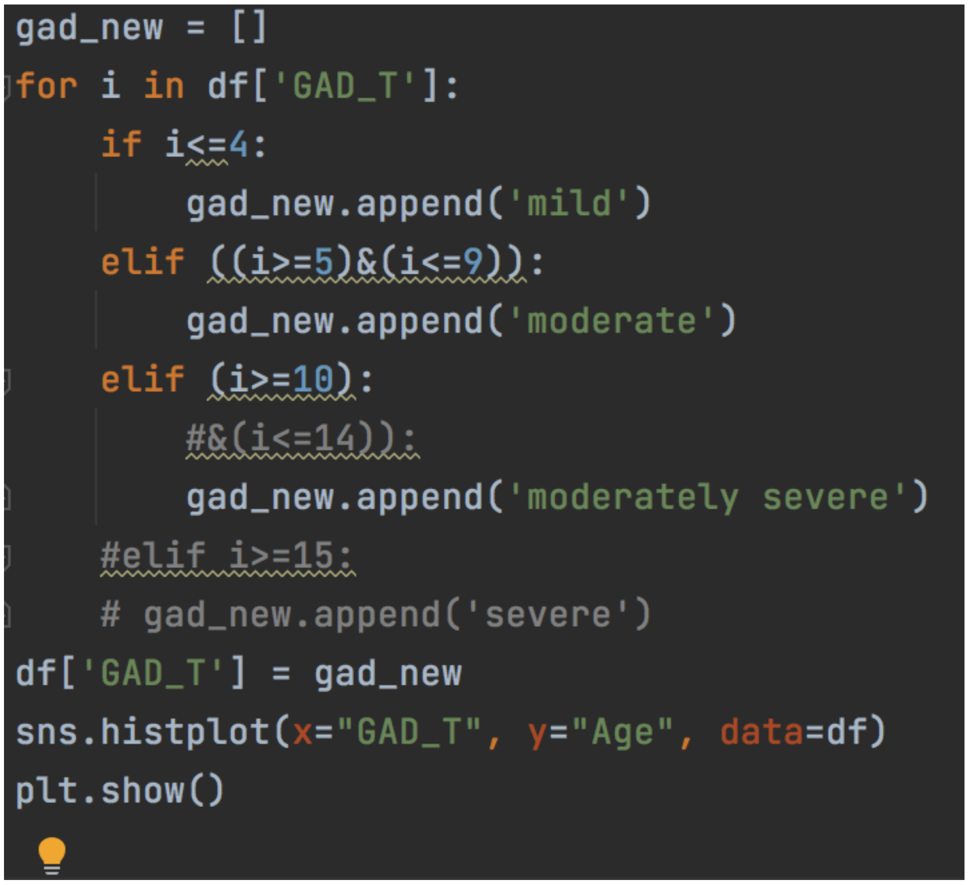
```
#convert the variables to numeric
df["Age"] = pd.to_numeric(df["Age"])
print(df["Age"].dtype)

df["Hours"] = pd.to_numeric(df["Hours"])
```

3- Using dictionary for most of the variables which were string to convert them to categorical since the decision tree didn't accept the string.



```
replace_map = {'Game':{'Counter Strike':1,'Destiny':2,'Diablo 3':3,'Guild Wars 2':4,'Hearthstone':5,'Heroes of the Storm':6,'League of Leg
               'GADE':{'Extremely difficult':3,'Very difficult':2,'Somewhat difficult':1,'Not difficult at all':0},
               'Platform':{'Console (PS, Xbox, ...)':0,'PC':1,'Smartphone / Tablet':2},
               'Gender':{'Male':0,'Female':1,'Other':2},
```

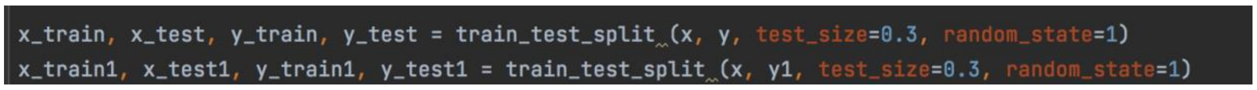
4-Make classification in our target variable(convert it to categorical) .



```
gad_new = []
for i in df['GAD_T']:
    if i<=4:
        gad_new.append('mild')
    elif ((i>=5)&(i<=9)):
        gad_new.append('moderate')
    elif (i>=10):
        #&(i<=14)):
        gad_new.append('moderately severe')
    #elif i>=15:
    # gad_new.append('severe')
df['GAD_T'] = gad_new
sns.histplot(x="GAD_T", y="Age", data=df)
plt.show()
```

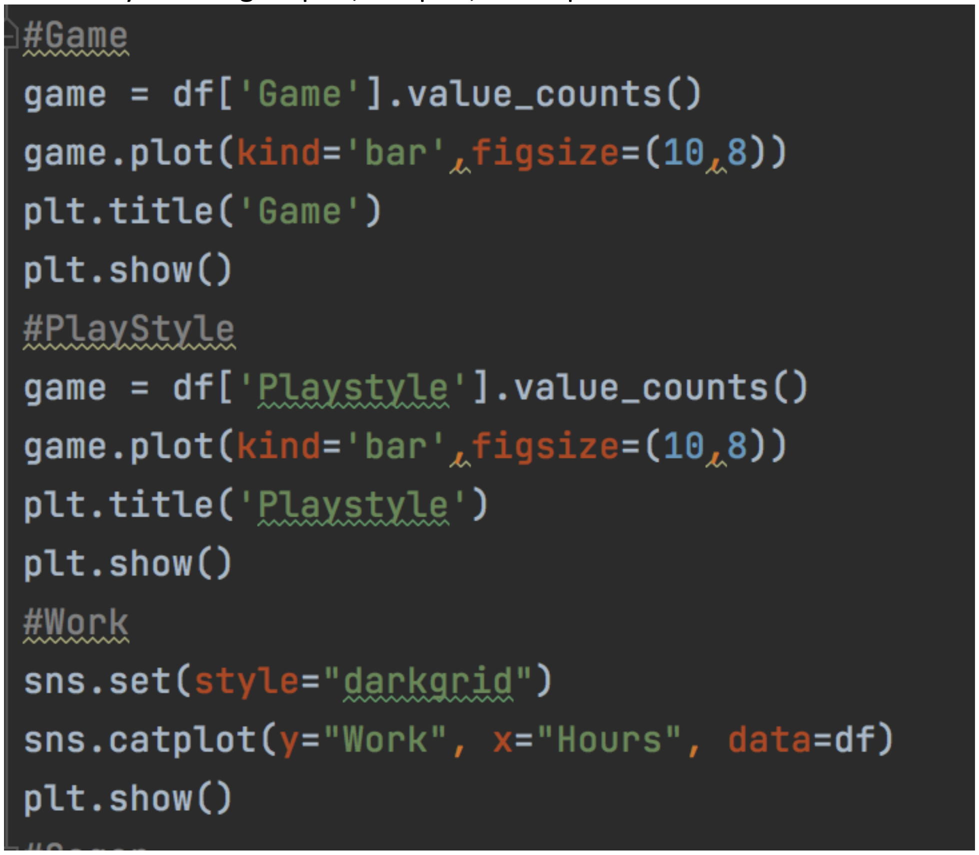
Here we put our two parts in our target variable together for more observations.(the items between 10 and 14 merged the items more than 15)

5- Split the data set into train and test dataset.

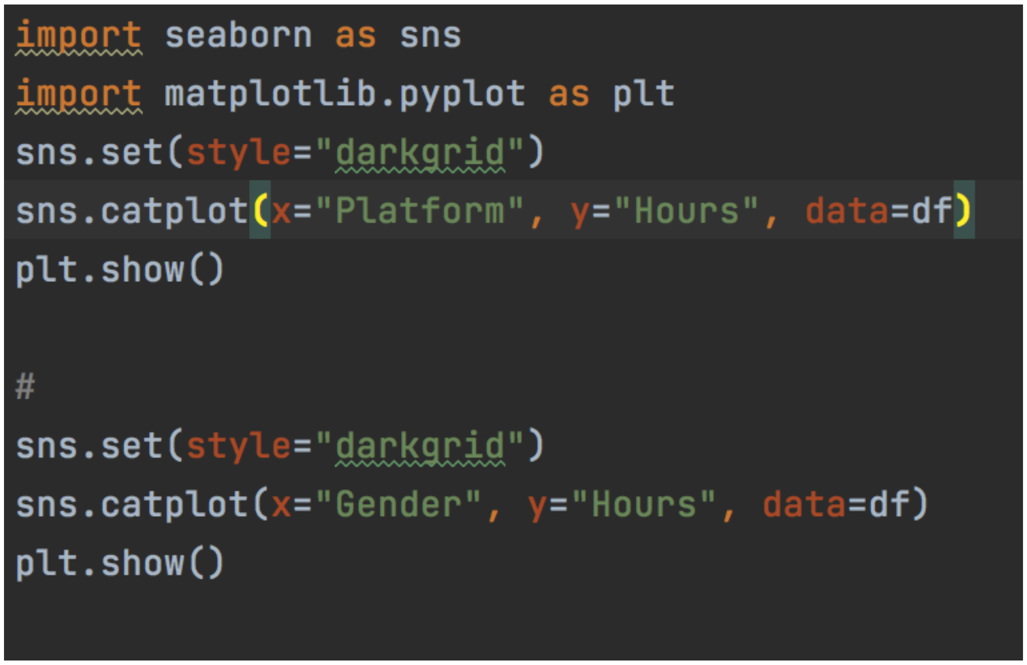


```
x_train, x_test, y_train, y_test = train_test_split (x, y, test_size=0.3, random_state=1)
x_train1, x_test1, y_train1, y_test1 = train_test_split (x, y1, test_size=0.3, random_state=1)
```

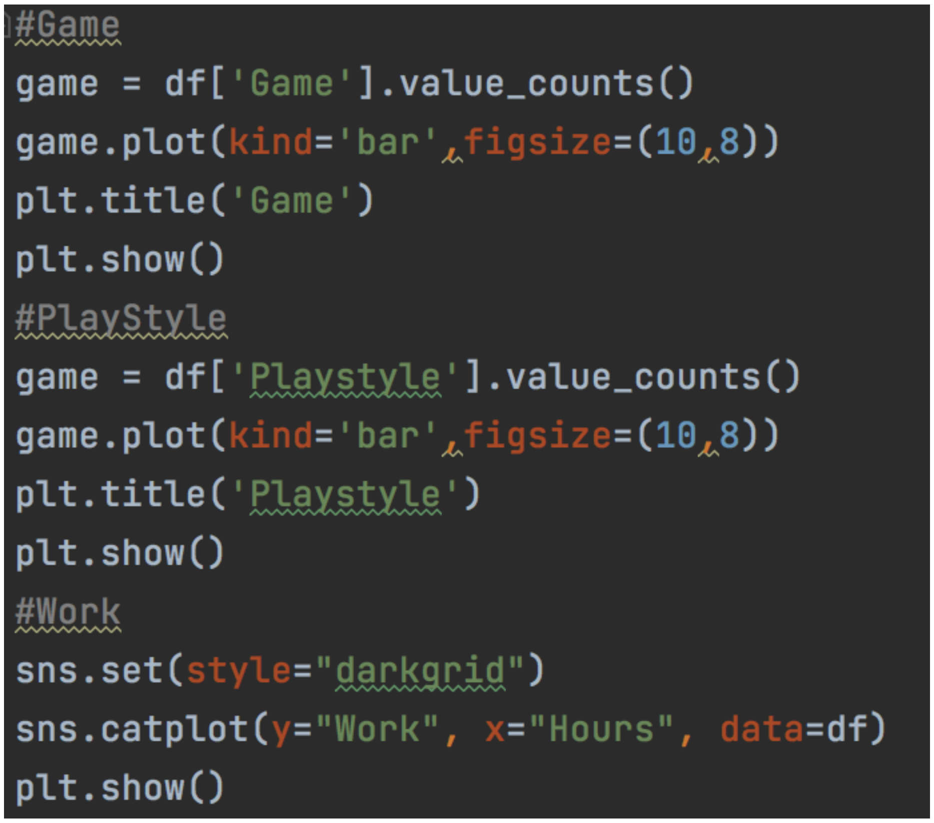
EDA Analyze :Using catplot, bar plot, count plot for data visualization.



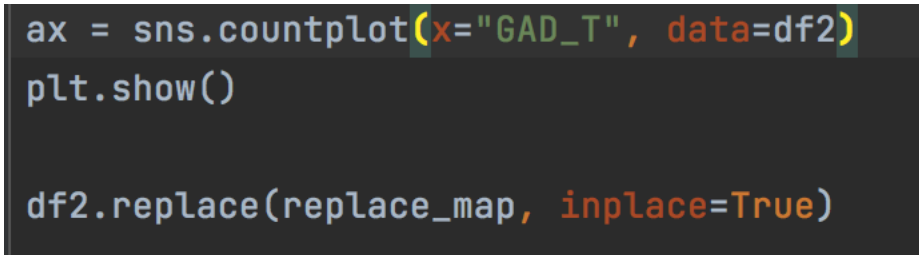
```
#Game
game = df['Game'].value_counts()
game.plot(kind='bar',figsize=(10,8))
plt.title('Game')
plt.show()
#PlayStyle
game = df['Playstyle'].value_counts()
game.plot(kind='bar',figsize=(10,8))
plt.title('Playstyle')
plt.show()
#Work
sns.set(style="darkgrid")
sns.catplot(y="Work", x="Hours", data=df)
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="darkgrid")
sns.catplot(x="Platform", y="Hours", data=df)
plt.show()


#
sns.set(style="darkgrid")
sns.catplot(x="Gender", y="Hours", data=df)
plt.show()
```



```
#Game
game = df['Game'].value_counts()
game.plot(kind='bar',figsize=(10,8))
plt.title('Game')
plt.show()
#PlayStyle
game = df['Playstyle'].value_counts()
game.plot(kind='bar',figsize=(10,8))
plt.title('Playstyle')
plt.show()
#Work
sns.set(style="darkgrid")
sns.catplot(y="Work", x="Hours", data=df)
plt.show()
```



```
ax = sns.countplot(x="GAD_T", data=df2)
plt.show()

df2.replace(replace_map, inplace=True)
```

Model Building:

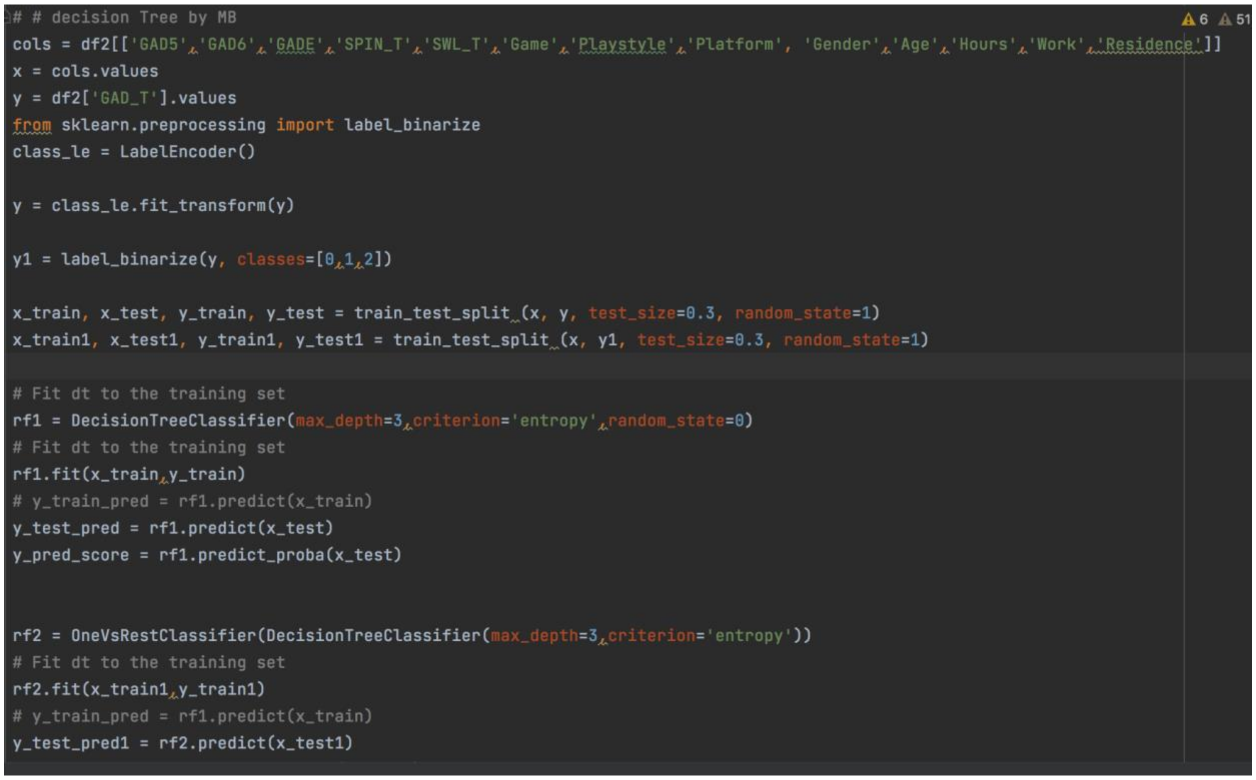
Built the (Entropy) Decision Tree Model.

Decision Tree Classifier :

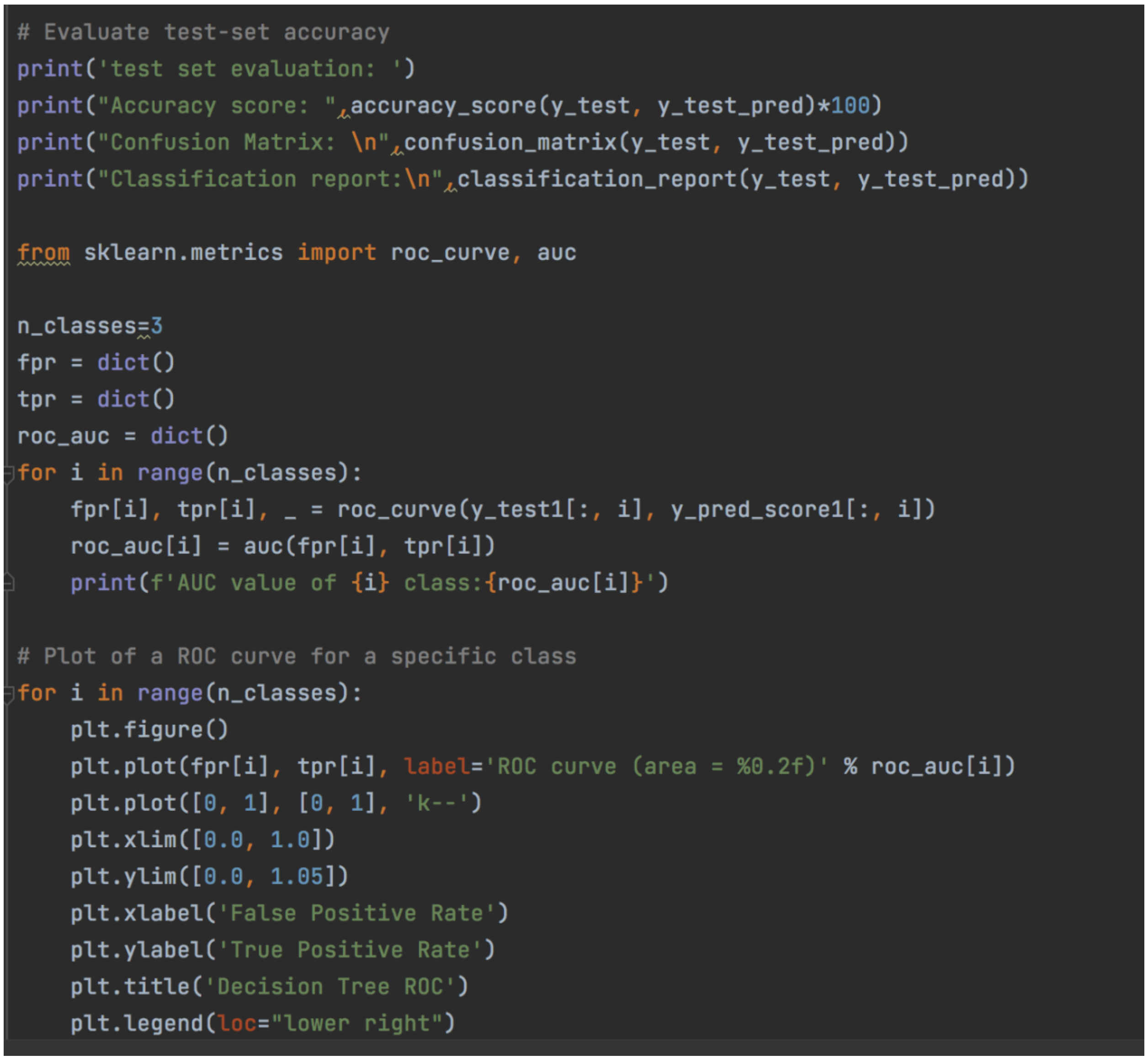
Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

Entropy: It is defined as a measure of impurity present in the data. The entropy is almost zero when the sample attains homogeneity but is one when it is equally divided. Entropy with the lowest value makes a model better in terms of prediction as it segregates the classes better.

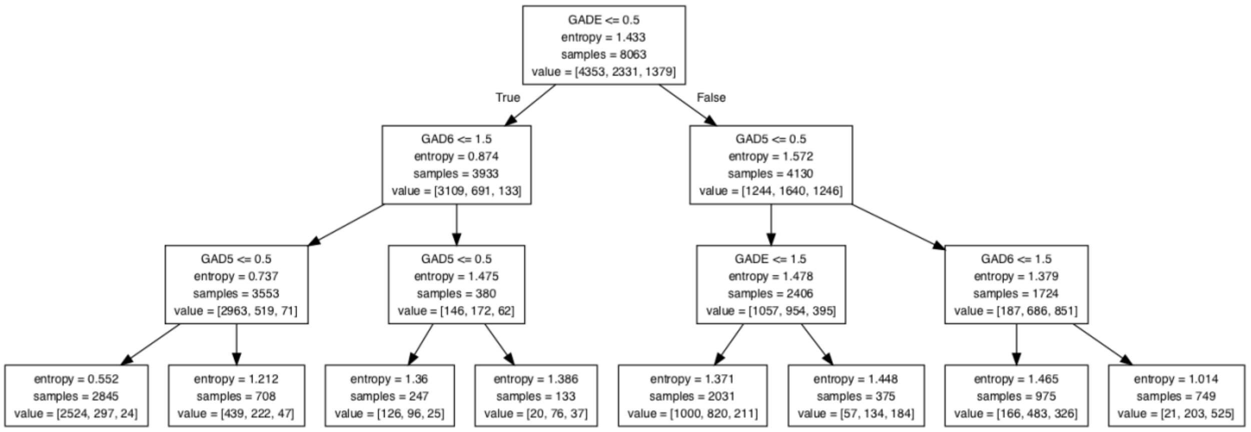
2. The results for Decision Tree model which includes confusion matrix, classification report, accuracy, ROC curve and Area under the Curve and the display of the trees were calculated.



```
# # decision Tree by MB
cols = df2[['GAD5','GAD6','GADE','SPIN_T','SWL_T','Game','Playstyle','Platform', 'Gender','Age','Hours','Work','Residence']]
x = cols.values
y = df2['GAD_T'].values
from sklearn.preprocessing import label_binarize
class_le = LabelEncoder()

y = class_le.fit_transform(y)

y1 = label_binarize(y, classes=[0,1,2])

x_train, x_test, y_train, y_test = train_test_split (x, y, test_size=0.3, random_state=1)
x_train1, x_test1, y_train1, y_test1 = train_test_split (x, y1, test_size=0.3, random_state=1)

# Fit dt to the training set
rf1 = DecisionTreeClassifier(max_depth=3,criterion='entropy',random_state=0)
# Fit dt to the training set
rf1.fit(x_train,y_train)
# y_train_pred = rf1.predict(x_train)
y_test_pred = rf1.predict(x_test)
y_pred_score = rf1.predict_proba(x_test)


rf2 = OneVsRestClassifier(DecisionTreeClassifier(max_depth=3,criterion='entropy'))
# Fit dt to the training set
rf2.fit(x_train1,y_train1)
# y_train_pred = rf1.predict(x_train)
y_test_pred1 = rf2.predict(x_test1)
```

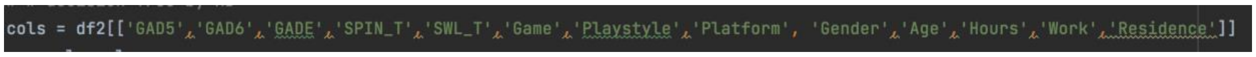


```python
# Evaluate test-set accuracy
print('test set evaluation: ')
print("Accuracy score: ",accuracy_score(y_test, y_test_pred)*100)
print("Confusion Matrix: \n",confusion_matrix(y_test, y_test_pred))
print("Classification report:\n",classification_report(y_test, y_test_pred))

from sklearn.metrics import roc_curve, auc

n_classes=3
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test1[:, i], y_pred_score1[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
    print(f'AUC value of {i} class:{roc_auc[i]}')

# Plot of a ROC curve for a specific class
for i in range(n_classes):
    plt.figure()
    plt.plot(fpr[i], tpr[i], label='ROC curve (area = %0.2f)' % roc_auc[i])
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Decision Tree ROC')
    plt.legend(loc="lower right")
```



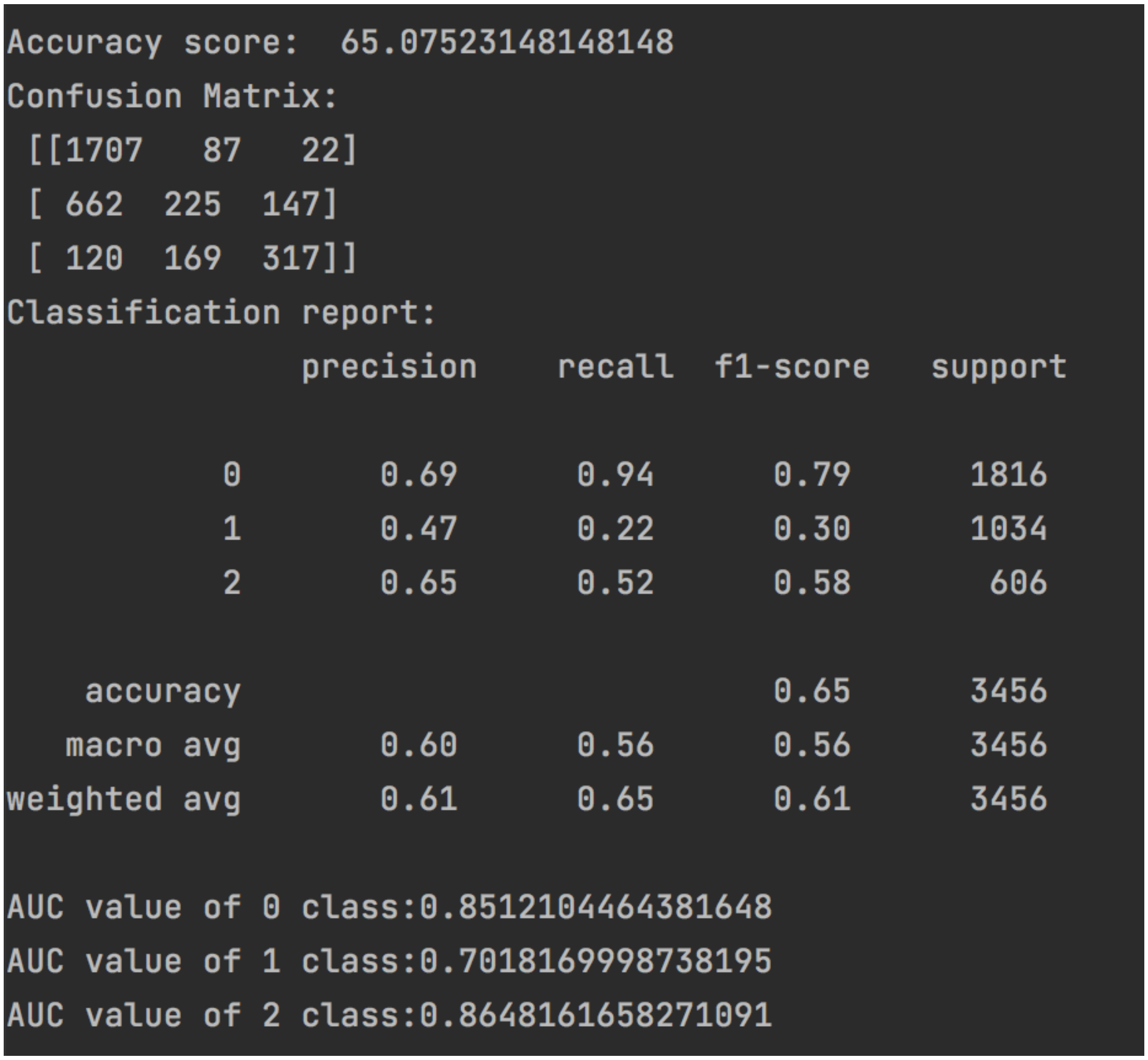
This tree graph shows the split of branches for the entropy model of the decision tree model. The maximum depth of the above model is 3.

Feature Importance Graph (Decision Tree): The feature importance graph in decision tree displays the most important features which needs to be considered while building a model.



```
cols = df2[['GAD5','GAD6','GADE','SPIN_T','SWL_T','Game','Playstyle','Platform', 'Gender','Age','Hours','Work','Residence']]
```

The result of this model is :

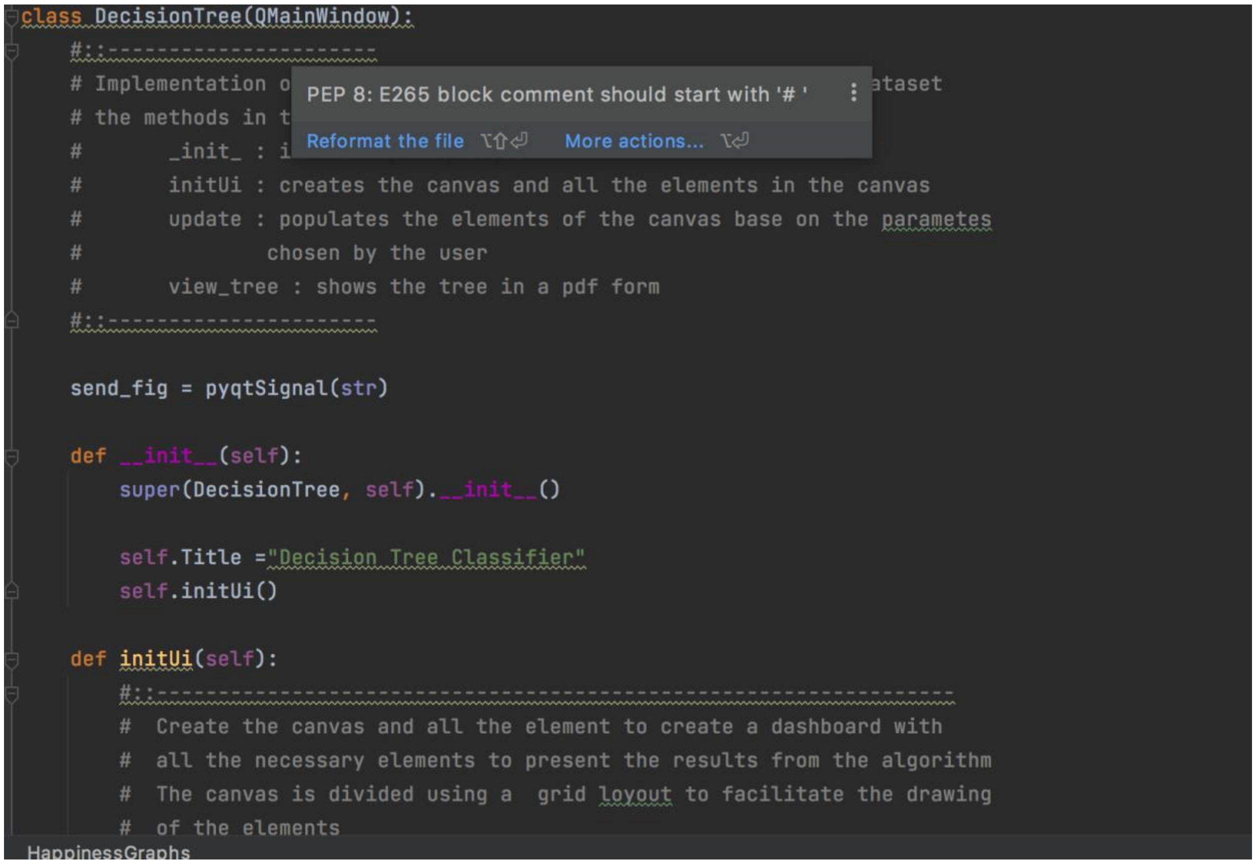


```
Accuracy score:  65.07523148148148
Confusion Matrix:
 [[1707   87   22]
 [ 662  225  147]
 [ 120  169  317]]
Classification report:
               precision    recall  f1-score   support

           0       0.69      0.94      0.79      1816
           1       0.47      0.22      0.30      1034
           2       0.65      0.52      0.58       606

    accuracy                           0.65      3456
   macro avg       0.60      0.56      0.56      3456
weighted avg       0.61      0.65      0.61      3456

AUC value of 0 class:0.8512104464381648
AUC value of 1 class:0.7018169998738195
AUC value of 2 class:0.8648161658271091
```

The Accuracy is 65 percent which is acceptable but is not great . we had another two models which we can see the accuracy over there as well.

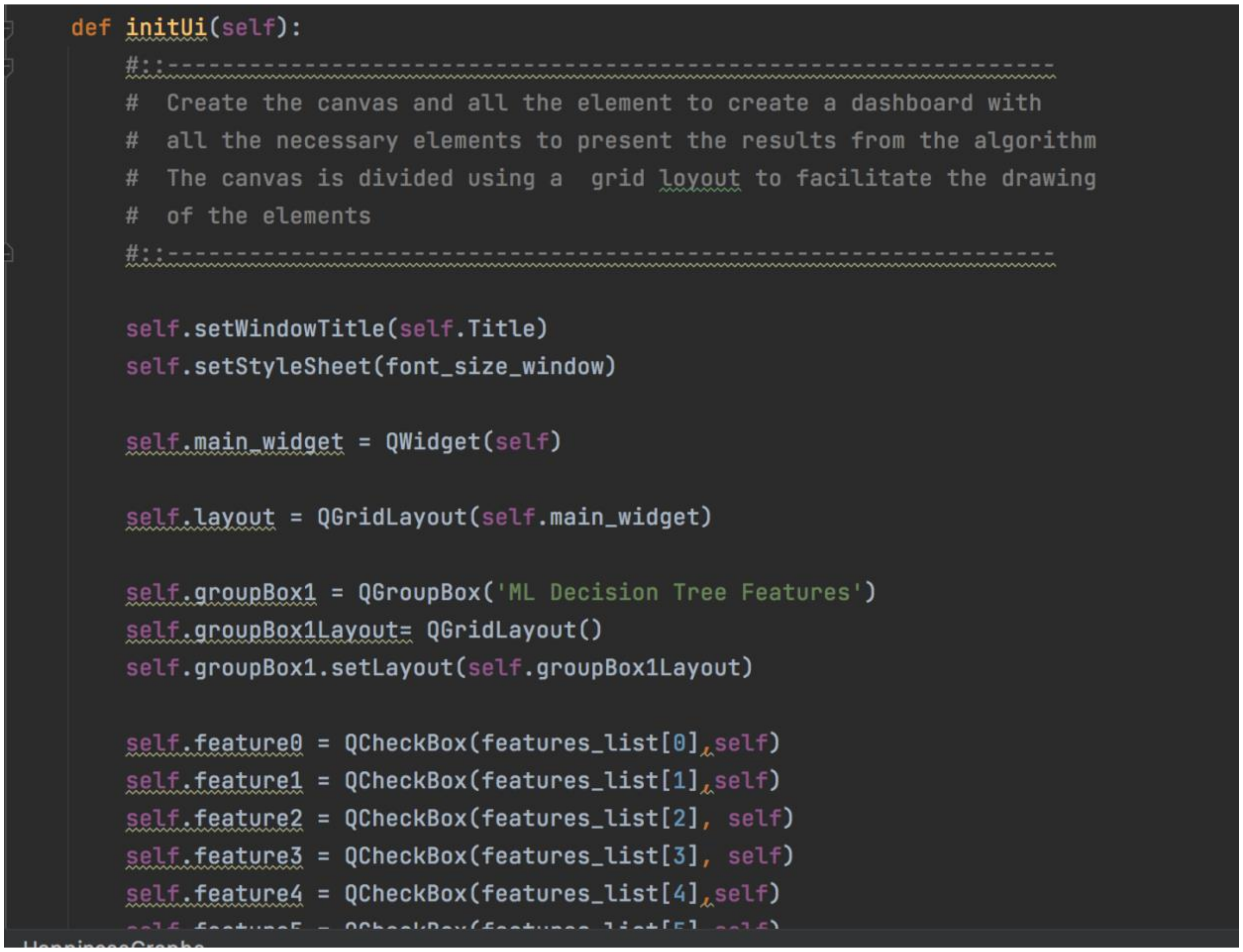
PYQT5 part of the code:

The code for main decision Tree class:

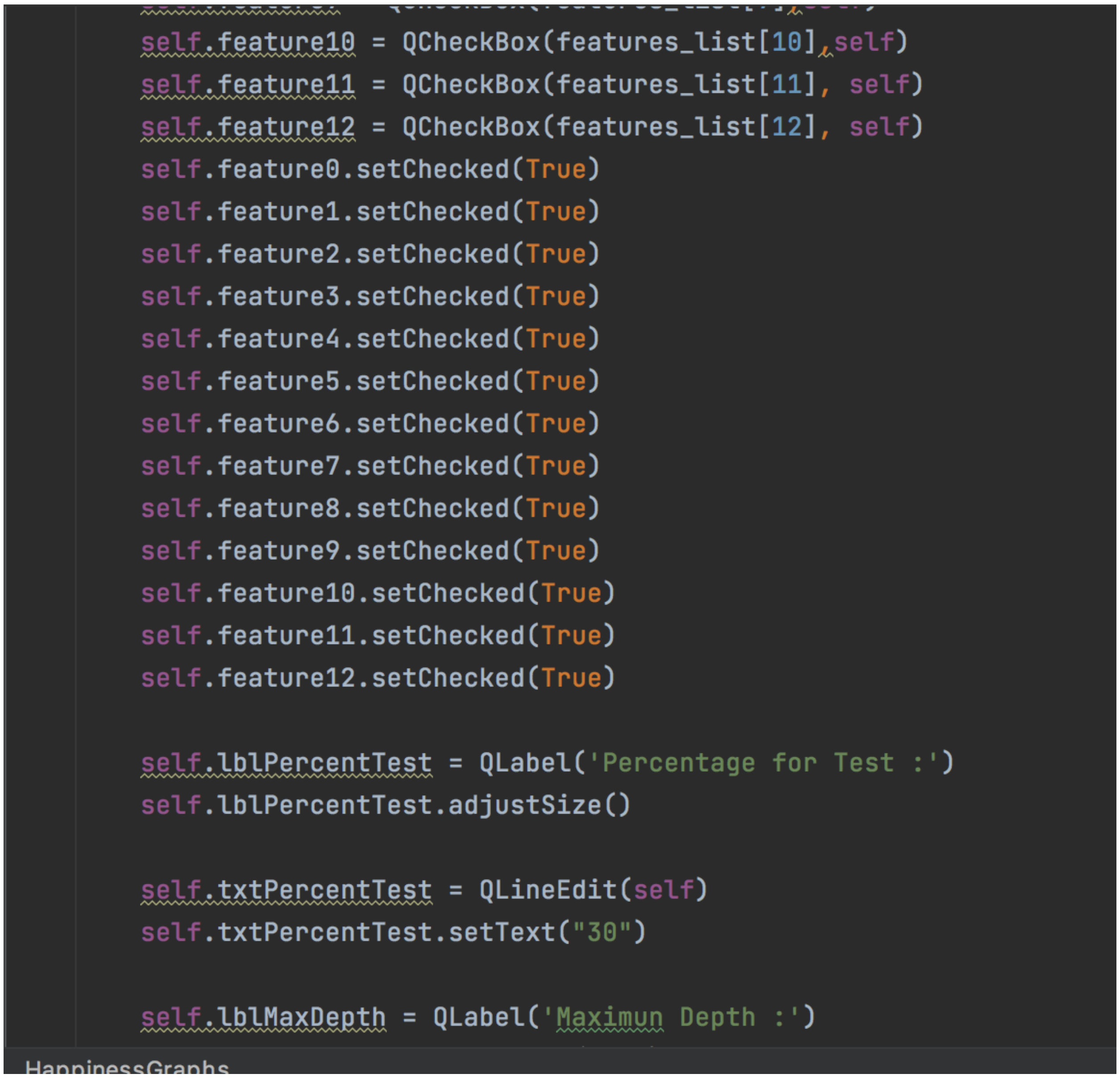


```
class DecisionTree(QMainWindow):
    #::----------------------
    # Implementation o                                               ataset
    # the methods in t
    #       _init_ : i
    #       initUi : creates the canvas and all the elements in the canvas
    #       update : populates the elements of the canvas base on the parametes
    #               chosen by the user
    #       view_tree : shows the tree in a pdf form
    #::----------------------

    send_fig = pyqtSignal(str)

    def __init__(self):
        super(DecisionTree, self).__init__()

        self.Title ="Decision Tree Classifier"
        self.initUi()

    def initUi(self):
        #::--------------------------------------------------------------
        #  Create the canvas and all the element to create a dashboard with
        #  all the necessary elements to present the results from the algorithm
        #  The canvas is divided using a  grid loyout to facilitate the drawing
        #  of the elements
```

PEP 8: E265 block comment should start with '# '

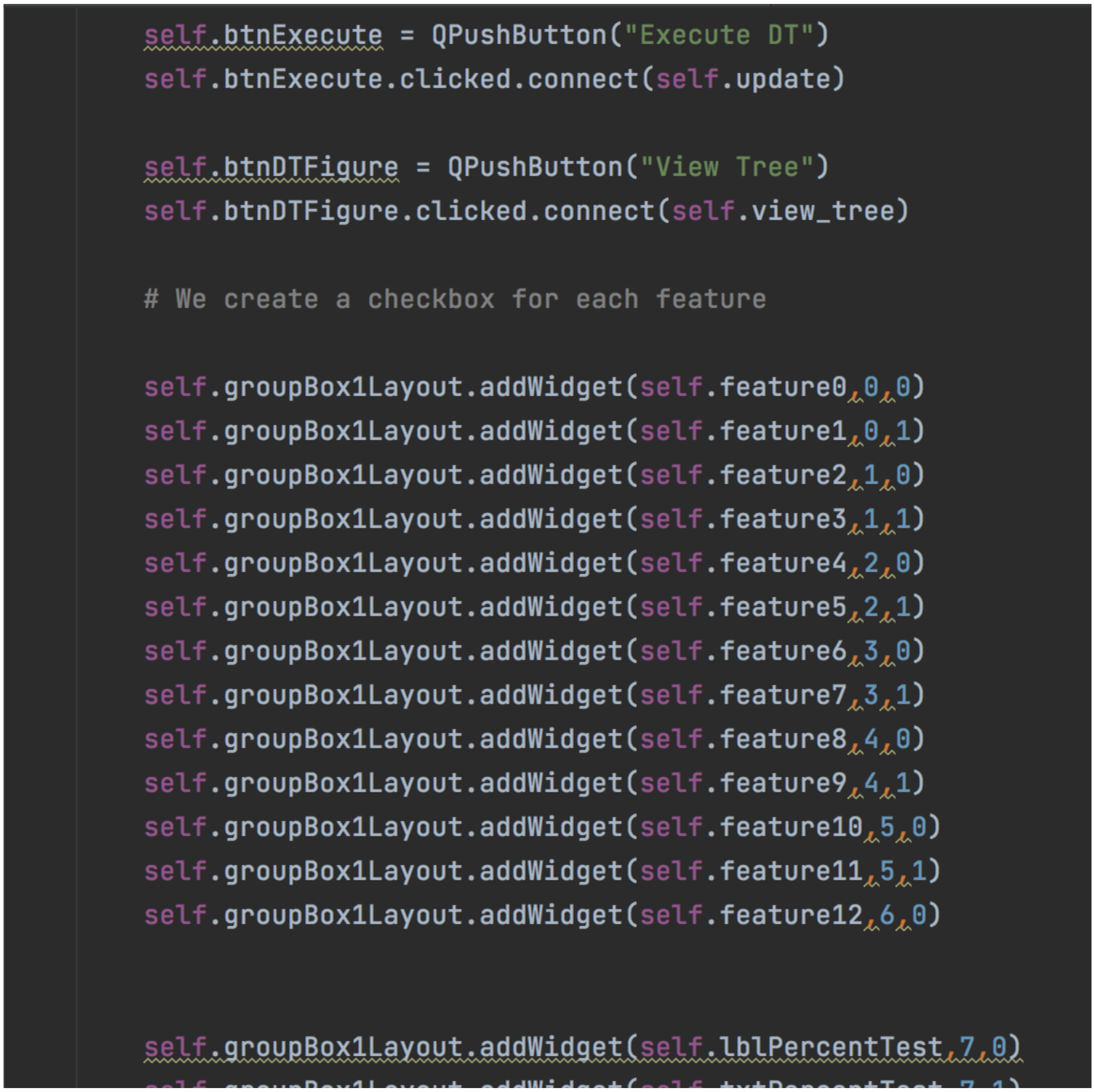


```
def initUi(self):
    #::--------------------------------------------------------------------
    #  Create the canvas and all the element to create a dashboard with
    #  all the necessary elements to present the results from the algorithm
    #  The canvas is divided using a  grid loyout to facilitate the drawing
    #  of the elements
    #::--------------------------------------------------------------------

    self.setWindowTitle(self.Title)
    self.setStyleSheet(font_size_window)

    self.main_widget = QWidget(self)

    self.layout = QGridLayout(self.main_widget)

    self.groupBox1 = QGroupBox('ML Decision Tree Features')
    self.groupBox1Layout= QGridLayout()
    self.groupBox1.setLayout(self.groupBox1Layout)

    self.feature0 = QCheckBox(features_list[0],self)
    self.feature1 = QCheckBox(features_list[1],self)
    self.feature2 = QCheckBox(features_list[2], self)
    self.feature3 = QCheckBox(features_list[3], self)
    self.feature4 = QCheckBox(features_list[4],self)
    self.feature5 = QCheckBox(features_list[5],self)
```

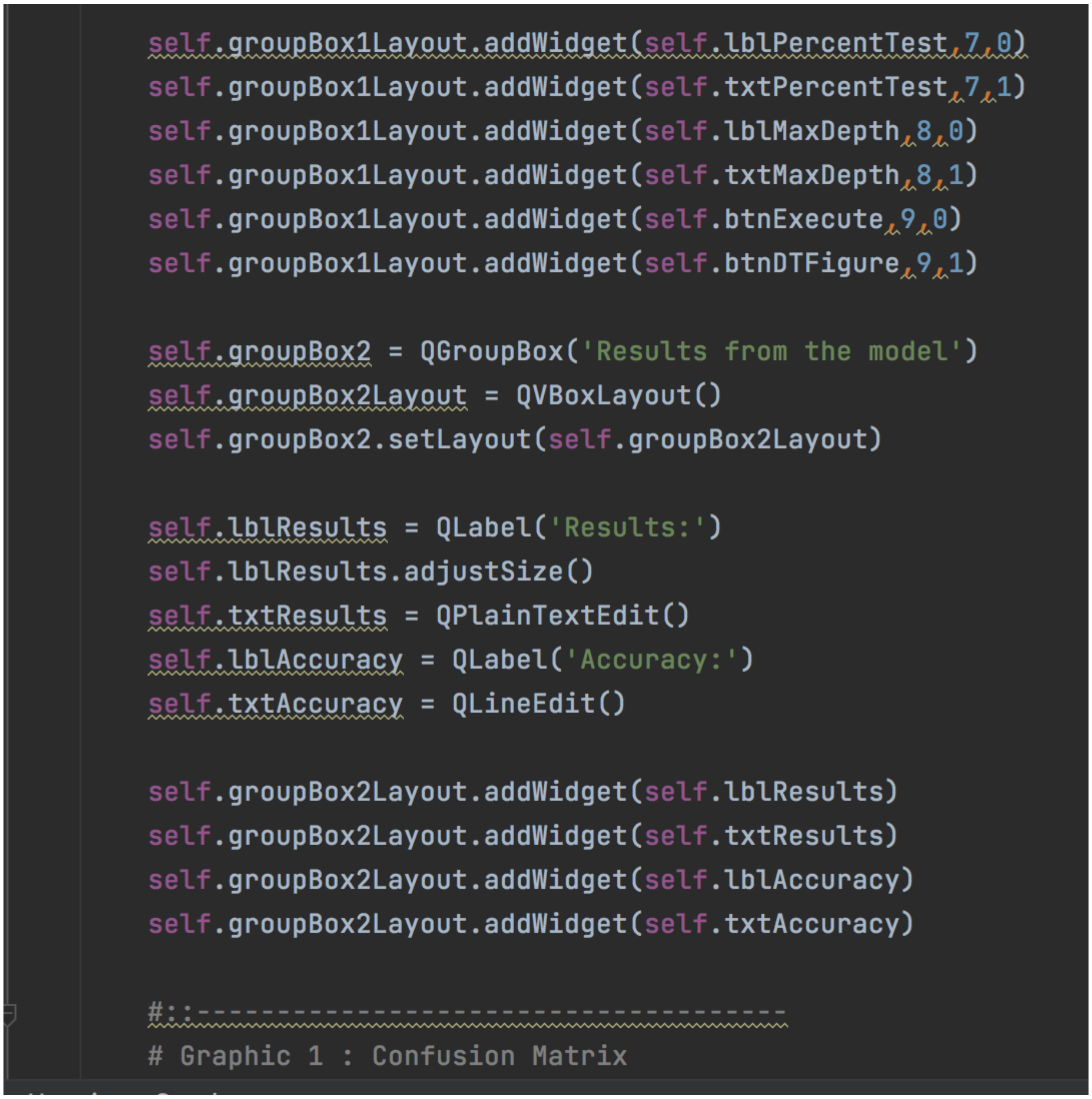


```
        self.feature10 = QCheckBox(features_list[10],self)
        self.feature11 = QCheckBox(features_list[11], self)
        self.feature12 = QCheckBox(features_list[12], self)
        self.feature0.setChecked(True)
        self.feature1.setChecked(True)
        self.feature2.setChecked(True)
        self.feature3.setChecked(True)
        self.feature4.setChecked(True)
        self.feature5.setChecked(True)
        self.feature6.setChecked(True)
        self.feature7.setChecked(True)
        self.feature8.setChecked(True)
        self.feature9.setChecked(True)
        self.feature10.setChecked(True)
        self.feature11.setChecked(True)
        self.feature12.setChecked(True)

        self.lblPercentTest = QLabel('Percentage for Test :')
        self.lblPercentTest.adjustSize()

        self.txtPercentTest = QLineEdit(self)
        self.txtPercentTest.setText("30")

        self.lblMaxDepth = QLabel('Maximun Depth :')
```

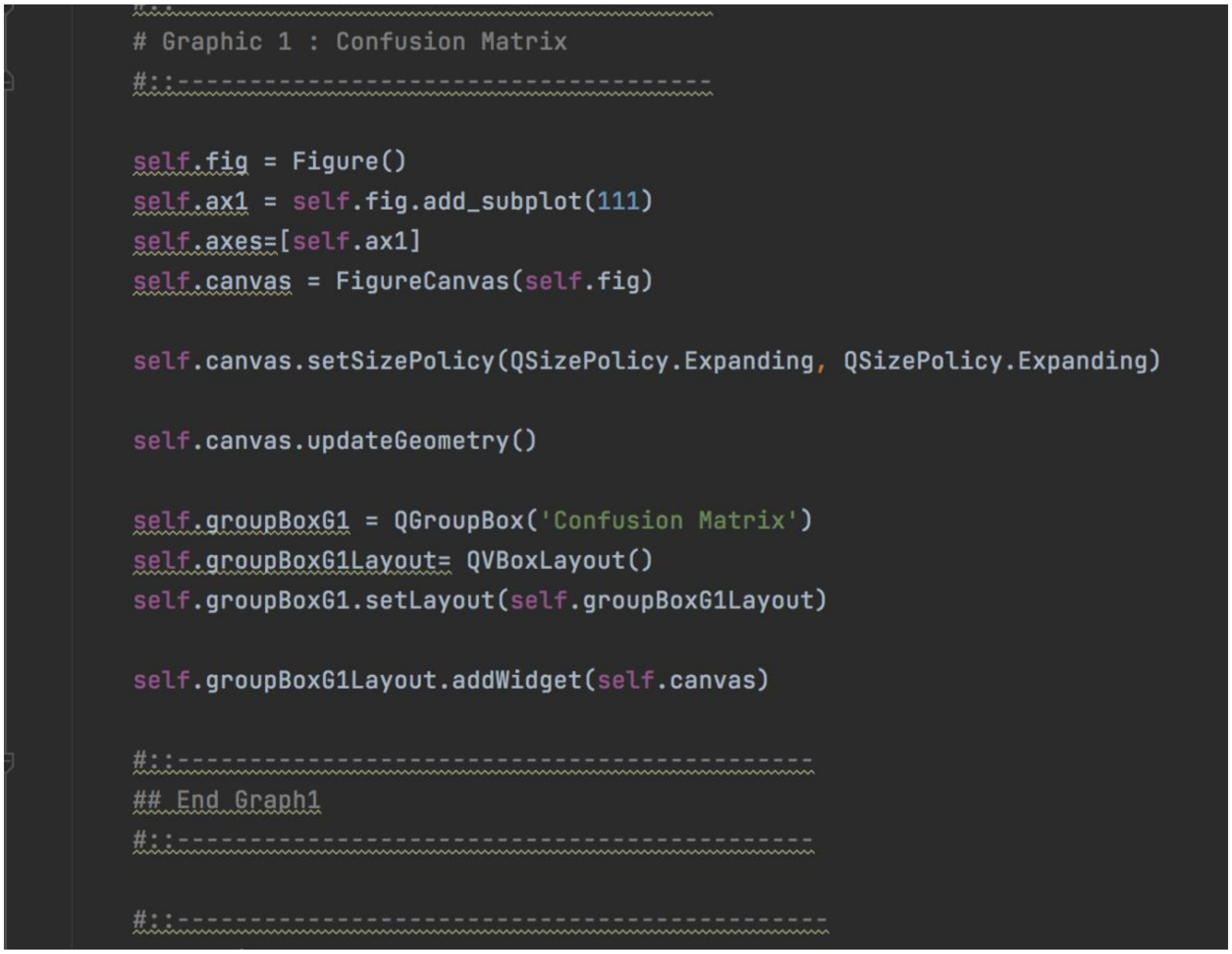
HappinessGraphs



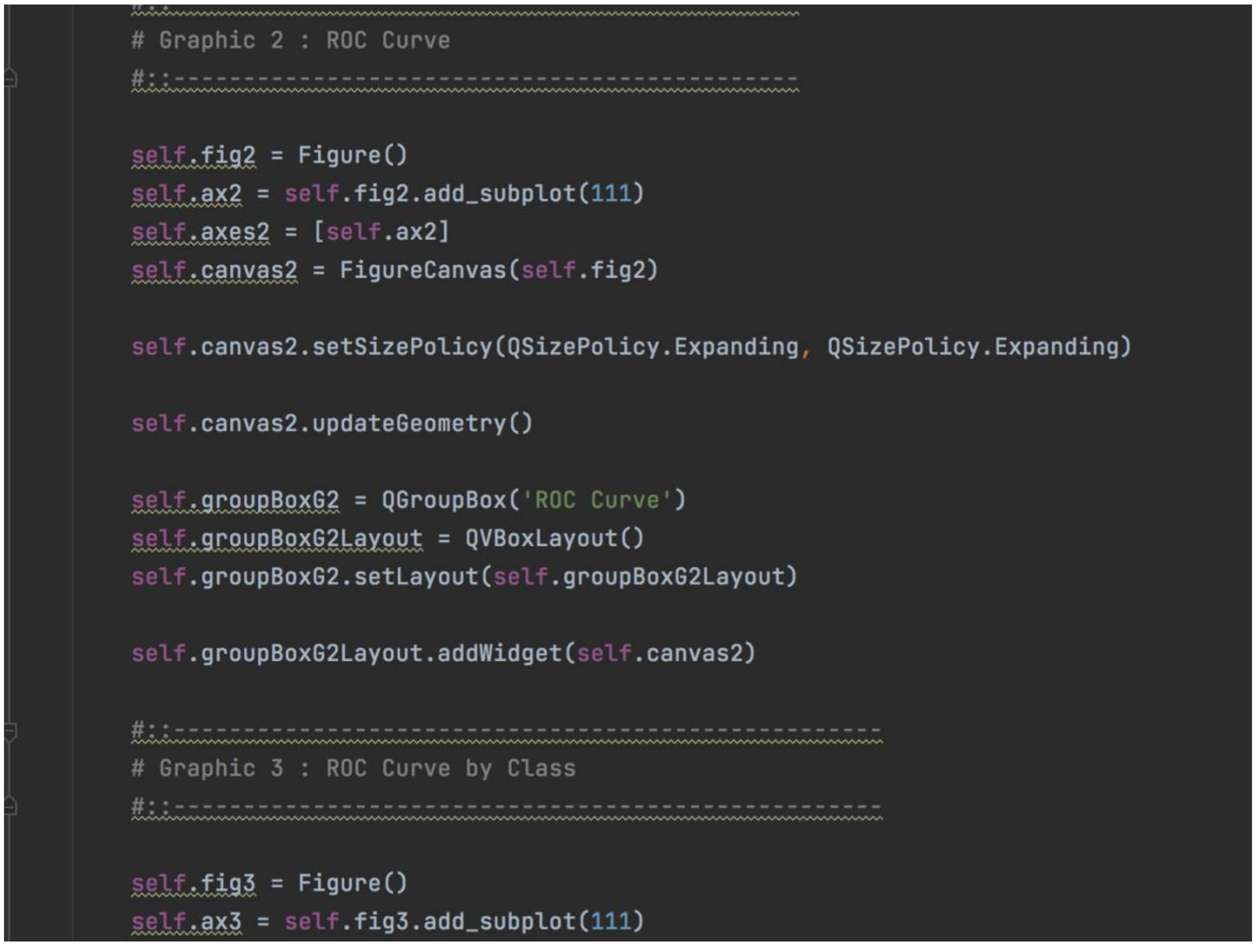
```python
self.btnExecute = QPushButton("Execute DT")
self.btnExecute.clicked.connect(self.update)

self.btnDTFigure = QPushButton("View Tree")
self.btnDTFigure.clicked.connect(self.view_tree)

# We create a checkbox for each feature

self.groupBox1Layout.addWidget(self.feature0,0,0)
self.groupBox1Layout.addWidget(self.feature1,0,1)
self.groupBox1Layout.addWidget(self.feature2,1,0)
self.groupBox1Layout.addWidget(self.feature3,1,1)
self.groupBox1Layout.addWidget(self.feature4,2,0)
self.groupBox1Layout.addWidget(self.feature5,2,1)
self.groupBox1Layout.addWidget(self.feature6,3,0)
self.groupBox1Layout.addWidget(self.feature7,3,1)
self.groupBox1Layout.addWidget(self.feature8,4,0)
self.groupBox1Layout.addWidget(self.feature9,4,1)
self.groupBox1Layout.addWidget(self.feature10,5,0)
self.groupBox1Layout.addWidget(self.feature11,5,1)
self.groupBox1Layout.addWidget(self.feature12,6,0)


self.groupBox1Layout.addWidget(self.lblPercentTest,7,0)
self.groupBox1Layout.addWidget(self.txtPercentTest,7,1)
```



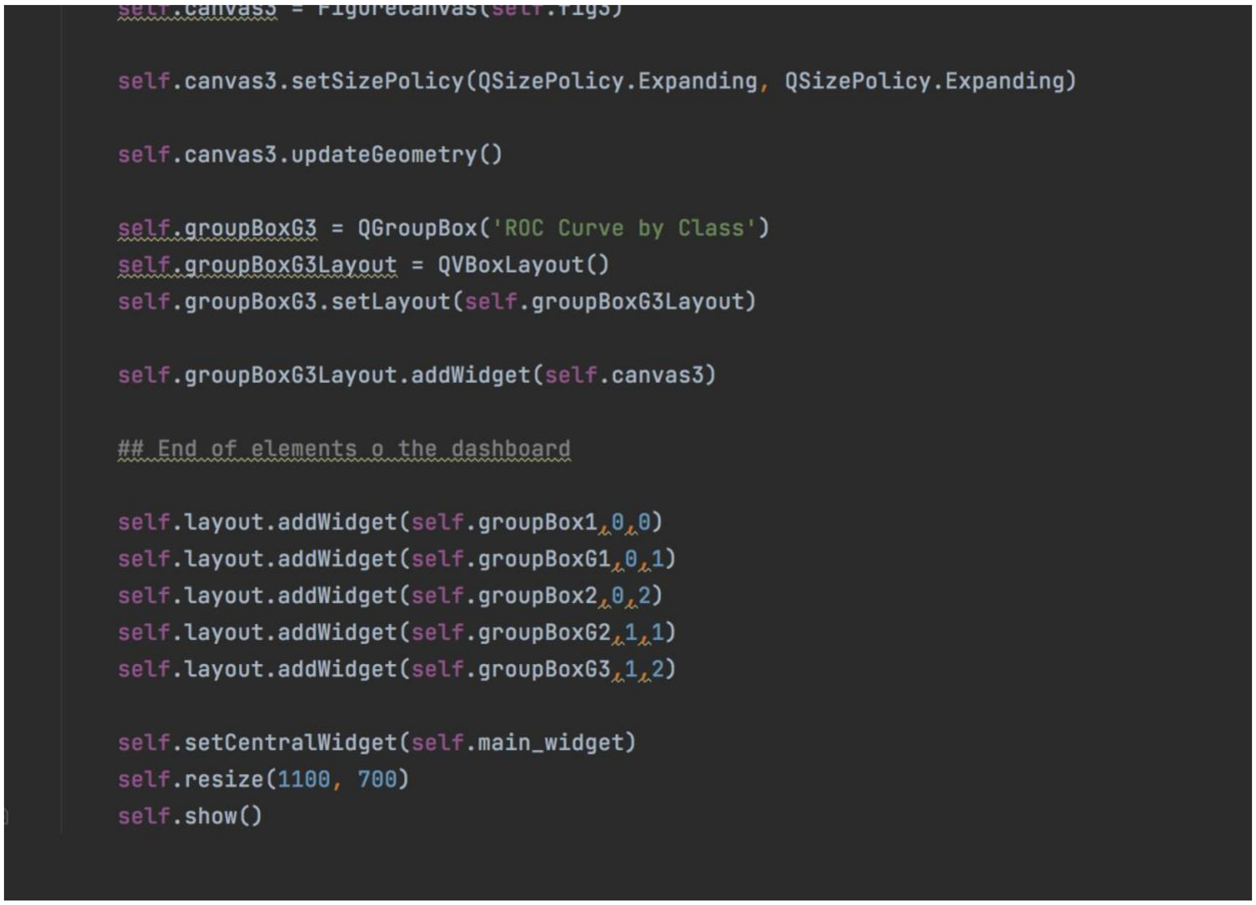
```
        self.groupBox1Layout.addWidget(self.lblPercentTest,7,0)
        self.groupBox1Layout.addWidget(self.txtPercentTest,7,1)
        self.groupBox1Layout.addWidget(self.lblMaxDepth,8,0)
        self.groupBox1Layout.addWidget(self.txtMaxDepth,8,1)
        self.groupBox1Layout.addWidget(self.btnExecute,9,0)
        self.groupBox1Layout.addWidget(self.btnDTFigure,9,1)

        self.groupBox2 = QGroupBox('Results from the model')
        self.groupBox2Layout = QVBoxLayout()
        self.groupBox2.setLayout(self.groupBox2Layout)

        self.lblResults = QLabel('Results:')
        self.lblResults.adjustSize()
        self.txtResults = QPlainTextEdit()
        self.lblAccuracy = QLabel('Accuracy:')
        self.txtAccuracy = QLineEdit()

        self.groupBox2Layout.addWidget(self.lblResults)
        self.groupBox2Layout.addWidget(self.txtResults)
        self.groupBox2Layout.addWidget(self.lblAccuracy)
        self.groupBox2Layout.addWidget(self.txtAccuracy)

        #::----------------------------------------
        # Graphic 1 : Confusion Matrix
```



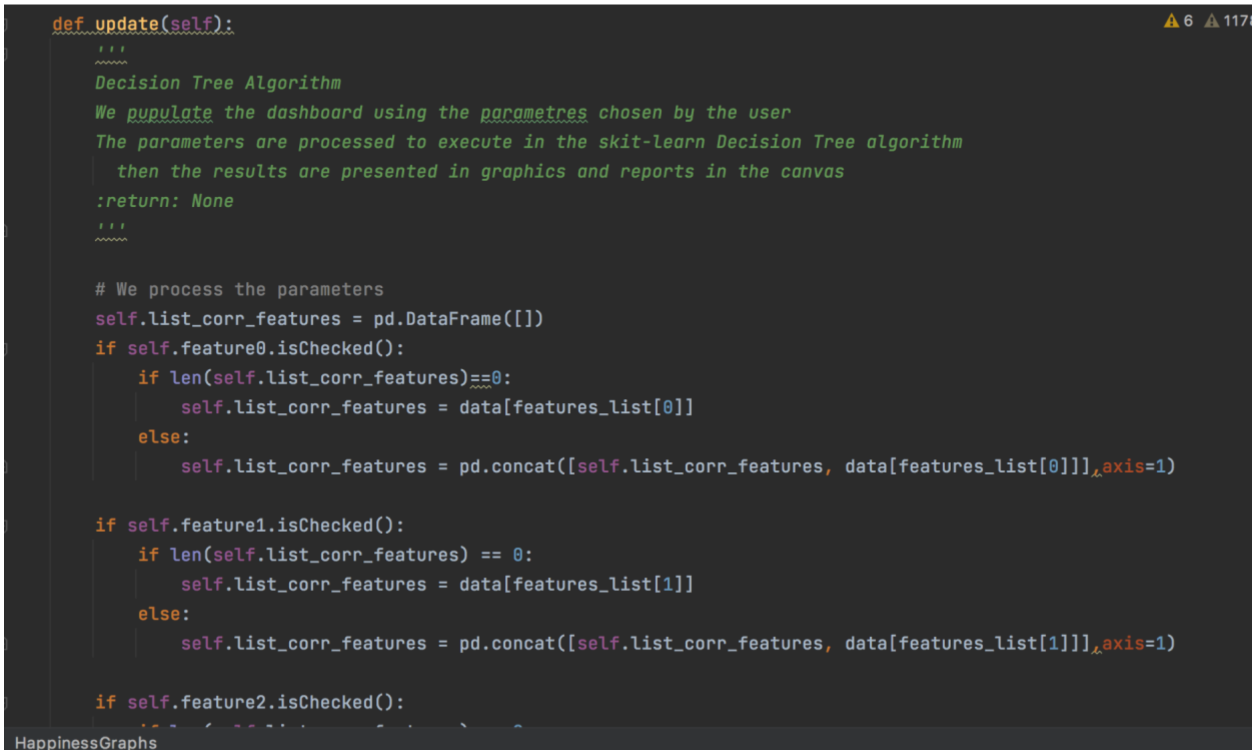
```
#::---------------------------------------
# Graphic 1 : Confusion Matrix
#::---------------------------------------

self.fig = Figure()
self.ax1 = self.fig.add_subplot(111)
self.axes=[self.ax1]
self.canvas = FigureCanvas(self.fig)

self.canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas.updateGeometry()

self.groupBoxG1 = QGroupBox('Confusion Matrix')
self.groupBoxG1Layout= QVBoxLayout()
self.groupBoxG1.setLayout(self.groupBoxG1Layout)

self.groupBoxG1Layout.addWidget(self.canvas)

#::----------------------------------------------
## End Graph1
#::----------------------------------------------

#::----------------------------------------------
```



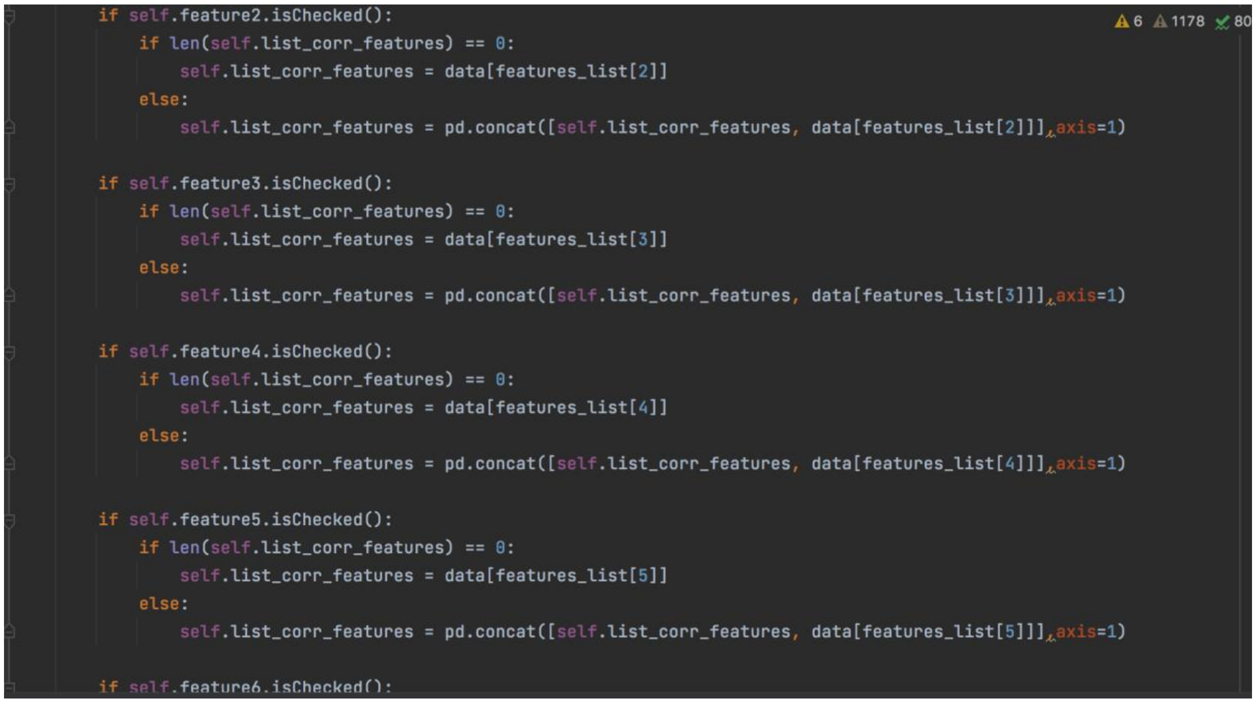
```
# Graphic 2 : ROC Curve
#::---------------------------------------------

self.fig2 = Figure()
self.ax2 = self.fig2.add_subplot(111)
self.axes2 = [self.ax2]
self.canvas2 = FigureCanvas(self.fig2)

self.canvas2.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas2.updateGeometry()

self.groupBoxG2 = QGroupBox('ROC Curve')
self.groupBoxG2Layout = QVBoxLayout()
self.groupBoxG2.setLayout(self.groupBoxG2Layout)

self.groupBoxG2Layout.addWidget(self.canvas2)

#::---------------------------------------------------
# Graphic 3 : ROC Curve by Class
#::---------------------------------------------------

self.fig3 = Figure()
self.ax3 = self.fig3.add_subplot(111)
```



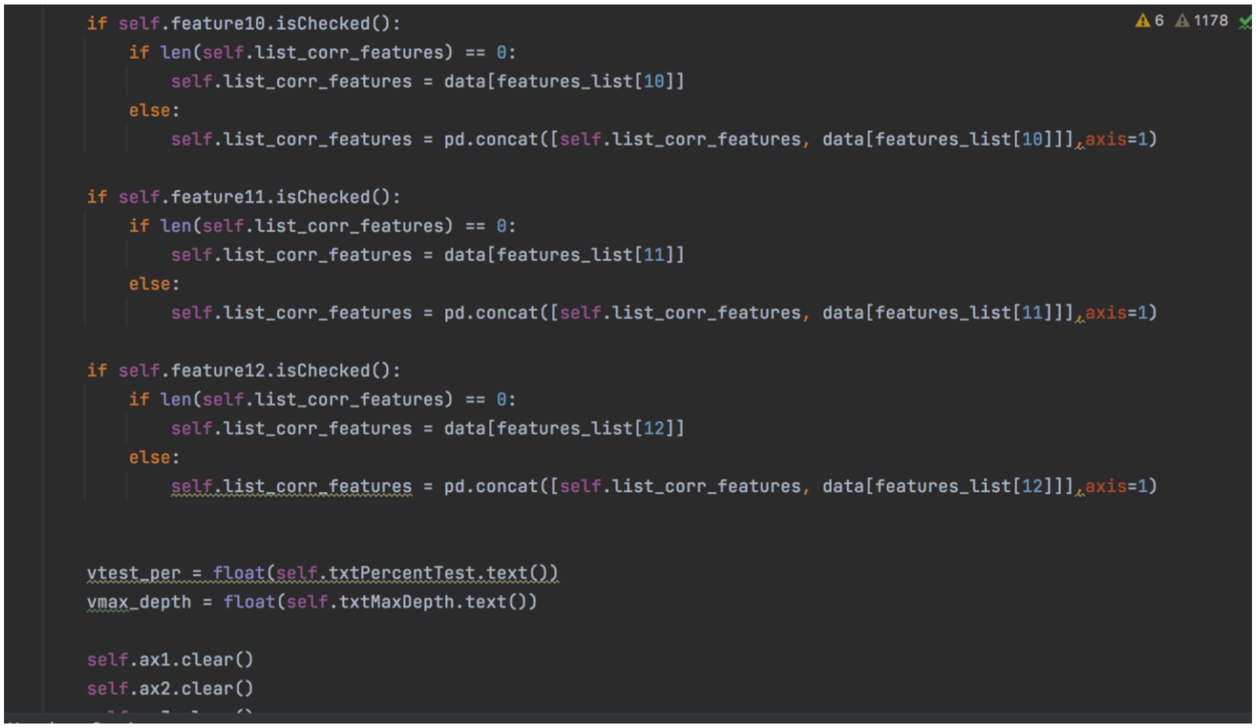
```python
        self.canvas3 = FigureCanvas(self.fig3)

        self.canvas3.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.canvas3.updateGeometry()

        self.groupBoxG3 = QGroupBox('ROC Curve by Class')
        self.groupBoxG3Layout = QVBoxLayout()
        self.groupBoxG3.setLayout(self.groupBoxG3Layout)

        self.groupBoxG3Layout.addWidget(self.canvas3)

        ## End of elements o the dashboard

        self.layout.addWidget(self.groupBox1,0,0)
        self.layout.addWidget(self.groupBoxG1,0,1)
        self.layout.addWidget(self.groupBox2,0,2)
        self.layout.addWidget(self.groupBoxG2,1,1)
        self.layout.addWidget(self.groupBoxG3,1,2)

        self.setCentralWidget(self.main_widget)
        self.resize(1100, 700)
        self.show()
```



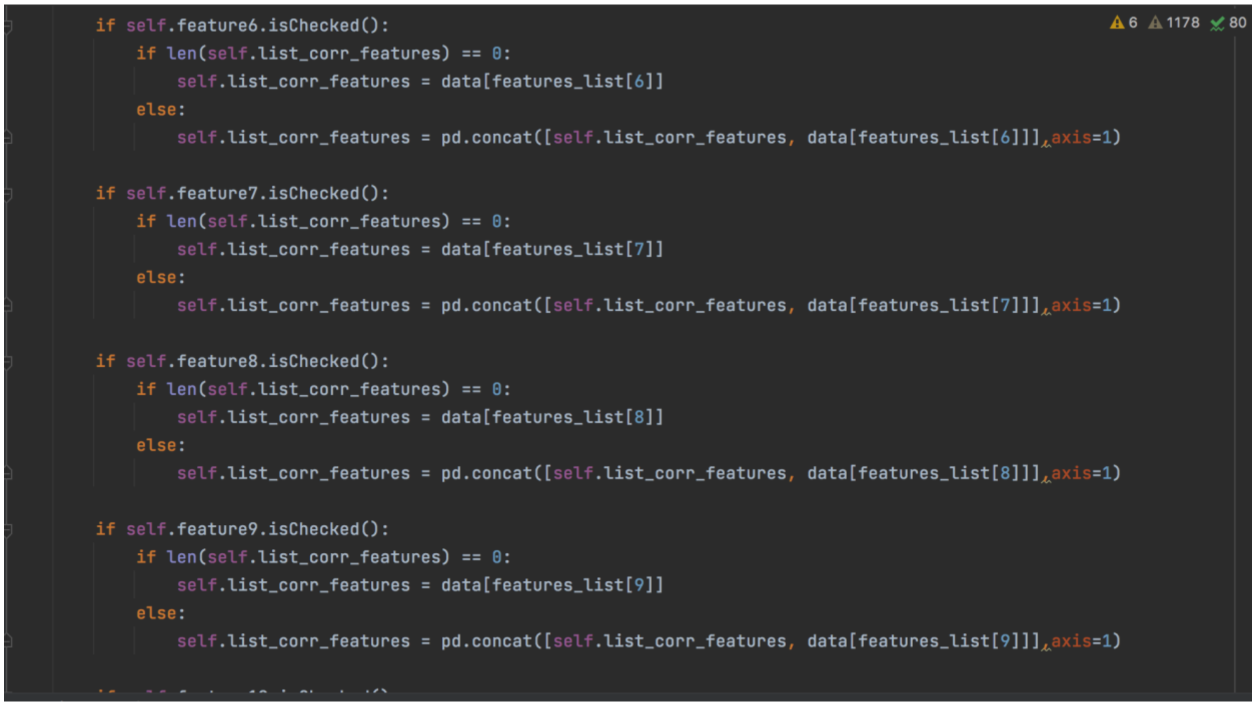
```python
    def update(self):
        '''
        Decision Tree Algorithm
        We pupulate the dashboard using the parametres chosen by the user
        The parameters are processed to execute in the skit-learn Decision Tree algorithm
          then the results are presented in graphics and reports in the canvas
        :return: None
        '''

        # We process the parameters
        self.list_corr_features = pd.DataFrame([])
        if self.feature0.isChecked():
            if len(self.list_corr_features)==0:
                self.list_corr_features = data[features_list[0]]
            else:
                self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[0]]],axis=1)

        if self.feature1.isChecked():
            if len(self.list_corr_features) == 0:
                self.list_corr_features = data[features_list[1]]
            else:
                self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[1]]],axis=1)

        if self.feature2.isChecked():
```



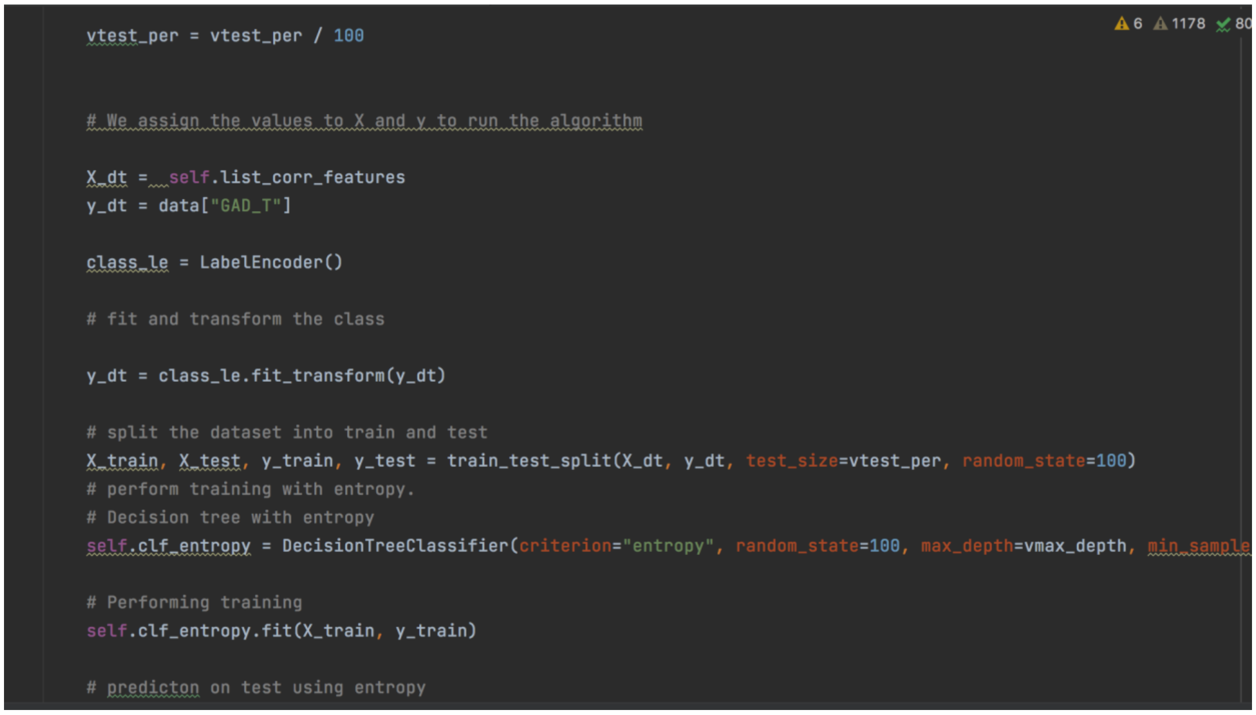
```
if self.feature2.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[2]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[2]]],axis=1)

if self.feature3.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[3]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[3]]],axis=1)

if self.feature4.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[4]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[4]]],axis=1)

if self.feature5.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[5]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[5]]],axis=1)

if self.feature6.isChecked():
```



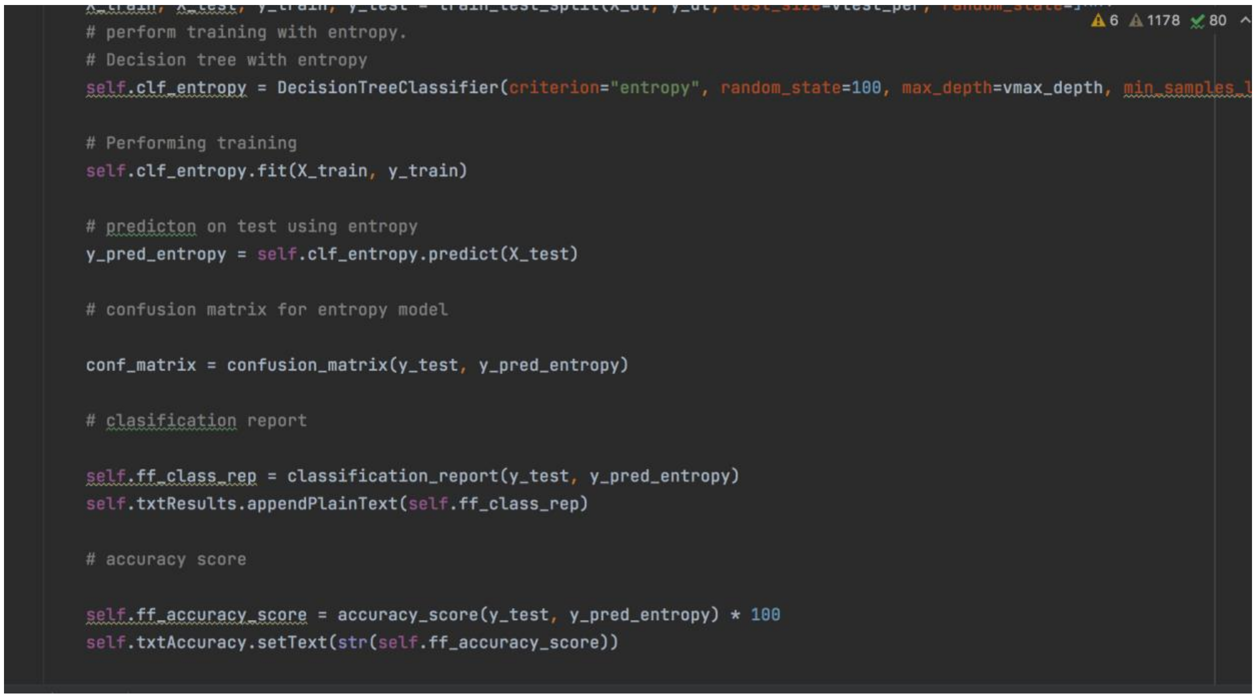
```
if self.feature10.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[10]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[10]]],axis=1)

if self.feature11.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[11]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[11]]],axis=1)

if self.feature12.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[12]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[12]]],axis=1)


vtest_per = float(self.txtPercentTest.text())
vmax_depth = float(self.txtMaxDepth.text())

self.ax1.clear()
self.ax2.clear()
```



```
if self.feature6.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[6]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[6]]],axis=1)

if self.feature7.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[7]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[7]]],axis=1)

if self.feature8.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[8]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[8]]],axis=1)

if self.feature9.isChecked():
    if len(self.list_corr_features) == 0:
        self.list_corr_features = data[features_list[9]]
    else:
        self.list_corr_features = pd.concat([self.list_corr_features, data[features_list[9]]],axis=1)
```



```
vtest_per = vtest_per / 100


# We assign the values to X and y to run the algorithm

X_dt =  self.list_corr_features
y_dt = data["GAD_T"]

class_le = LabelEncoder()

# fit and transform the class

y_dt = class_le.fit_transform(y_dt)

# split the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X_dt, y_dt, test_size=vtest_per, random_state=100)
# perform training with entropy.
# Decision tree with entropy
self.clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=100, max_depth=vmax_depth, min_sample

# Performing training
self.clf_entropy.fit(X_train, y_train)

# predicton on test using entropy
```



```
X_train, X_test, y_train, y_test = train_test_split(X_dt, y_dt, test_size=vtest_per, random_state=
# perform training with entropy.
# Decision tree with entropy
self.clf_entropy = DecisionTreeClassifier(criterion="entropy", random_state=100, max_depth=vmax_depth, min_samples_l

# Performing training
self.clf_entropy.fit(X_train, y_train)

# predicton on test using entropy
y_pred_entropy = self.clf_entropy.predict(X_test)

# confusion matrix for entropy model

conf_matrix = confusion_matrix(y_test, y_pred_entropy)

# clasification report

self.ff_class_rep = classification_report(y_test, y_pred_entropy)
self.txtResults.appendPlainText(self.ff_class_rep)

# accuracy score

self.ff_accuracy_score = accuracy_score(y_test, y_pred_entropy) * 100
self.txtAccuracy.setText(str(self.ff_accuracy_score))
```

4.RESULTS:

FIRST MODEL:

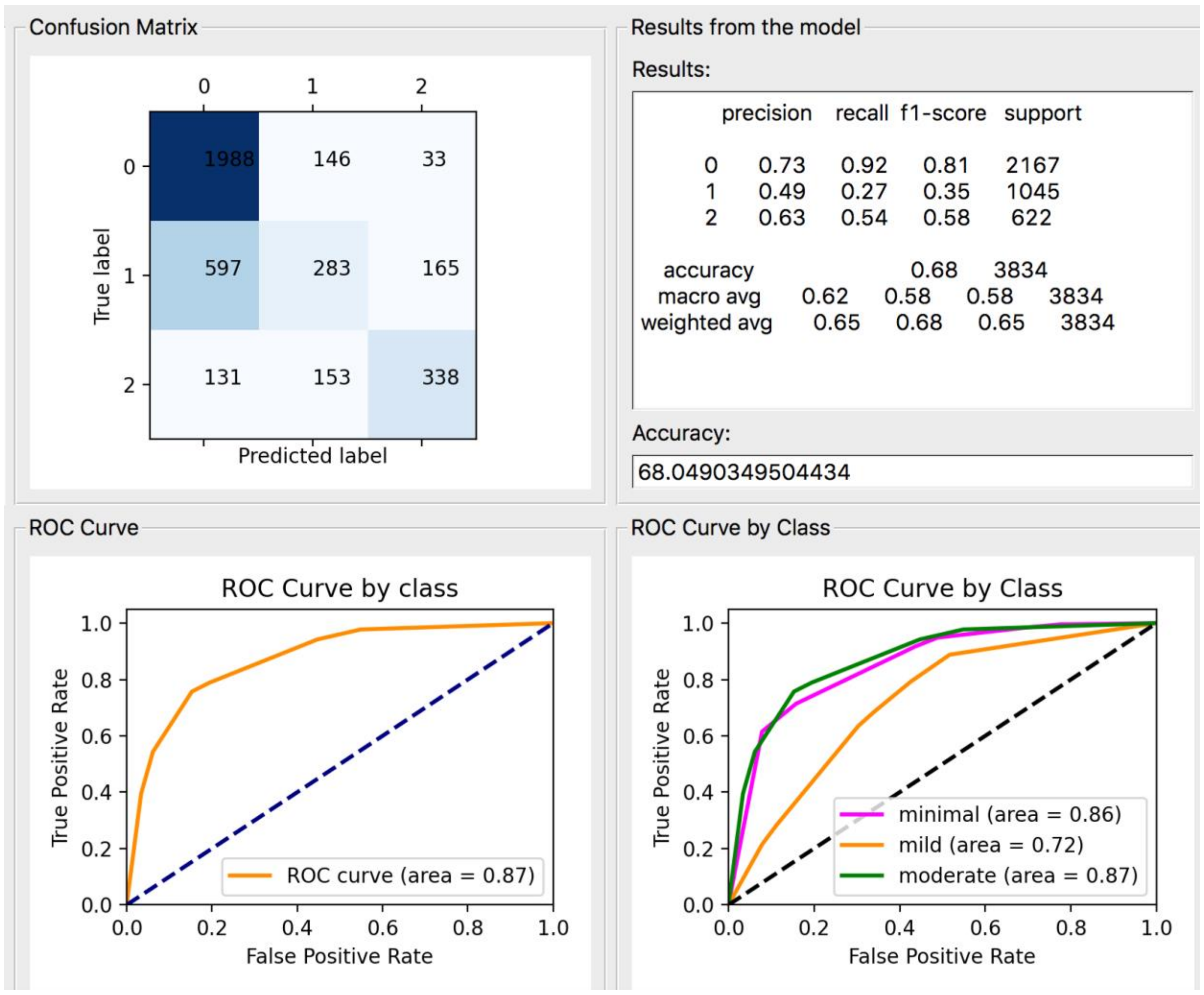
First I chose all the variables and add them to my model. Then just remove the variables which either had the high co-relations together or did not increase the accuracy. We calculate the accuracy, classification report and area under the curve.

We find the confusion matrix and plot it as a heatmap, plot the ROC curve.

We obtain following results:

The following result is my decision tree result in GUI:

Decision Tree Classifier: The image displays the decision tree dashboard and the user can manually change the percentage for test and also the maximum depth, The features can be selected as per the user's choice and execute the decision tree. The plot roc button displays the ROC graph.



```
             precision    recall  f1-score   support

           0      0.73      0.92      0.81      2167
           1      0.49      0.27      0.35      1045
           2      0.63      0.54      0.58       622

    accuracy                          0.68      3834
   macro avg      0.62      0.58      0.58      3834
weighted avg      0.65      0.68      0.65      3834
```

```
68.0490349504434
```

Confusion matrix: we use the confusion matrix to describe the performance of the classification model. We have three classes on confusion matrix: 0 is minimal- 1 is mild and 2 is moderate. This Function finds the distribution of all the predicted responses and shows how they compare to their true classes.

Accuracy: The Decision Tree model has the Accuracy of 68% and f1 score for our first class is 81 percent which means that it can predict the 81% of the first class but the second and third one is low which are 35 and 58.

ROC Curve by class:

The ROC curve is produced by calculating and plotting the true positive rate against the false positive for a single classifier at a variety of thresholds. ROC is a probability curve and AUC represent the degree or measure of separability. The orange colored represents the roc curve and blue represents the auc. The roc_auc value of decision tree models is 0.87.

5.SUMMARY and CONCLUSION:

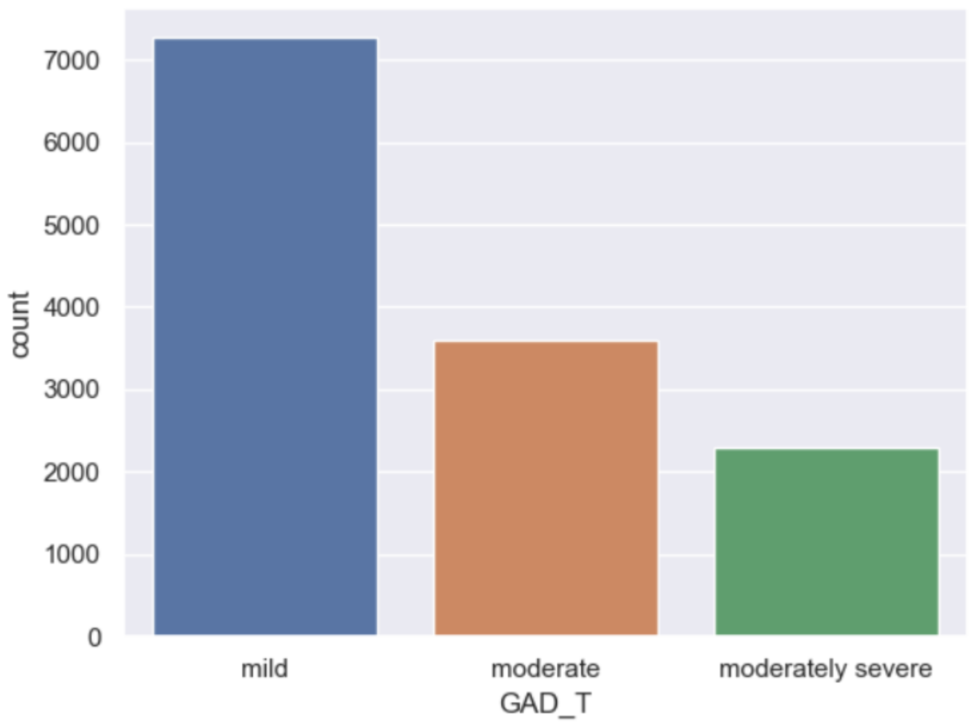
SUMMARY:
FIRST MODEL(decision tree):

1. Accuracy : 68%
2. F1 score for 0 : 0.81 ,for 1: 0.35 , for 2: 0.58
3. Precision for 0 :0.73, for 1:0.49 , for 2 :0.63
4. Recall for 0: 0.92, for 1:0.27, for 2:0.54
5. Area under the ROC curve for the decision tree is : 0.87 .
6. The features we finalized with are these 13 columns which have the most impact on GAD_T:

```
7. 'GAD5','GAD6','GADE','SPIN_T','SWL_T','Game','Playstyle','Platform',
   'Gender','Age','Hours','Work','Residence'
```

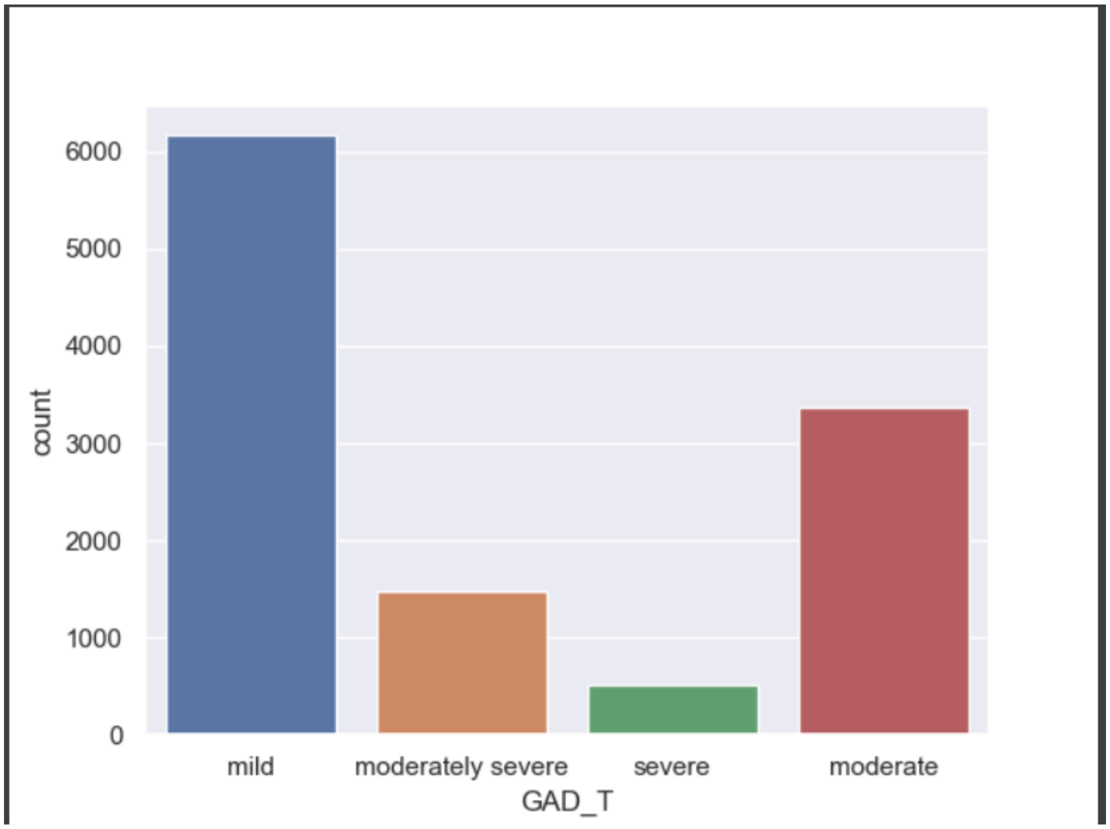
These are the most important features in our model, however the decision Tree model wasn't the best one.

We can observe that f1-scores, precision and recall rate is high for 0's in this model. The reason is because there are more number of observations which have 0's in target variable than those which have 1's and 2's as we have observed the histogram of target variable in EDA analysis.



Here is the Countplot that I mentioned for the GAD_T column.

It was 4 columns in the past but we decided the put the last 2 columns together since the observation on them was so low and the data was not balanced.(Because you asked the question in the class to see columns, I add the previous plot here for you)



CONCLUSION:

SVM model is the best model of the three different classifiers because it has the highest accuracy, f1-score and precision. It has the best AUC of the three classifiers as well. Also we have built the model with 13 features which have highest importance of features, so it is more efficient of the three classifiers in the sense of having more accuracy with these 13 dimensions used.

6.PERCENT OF CODE:

I haven’t copied any portion of code from anywhere, but I searched for the syntaxes and I used the codes which provided by Professor Amir Jafari in the github.

7.REFERENCES:

• https://numpy.org/doc/
• https://matplotlib.org/stable/users/index.html
• https://matplotlibguide.readthedocs.io/en/latest/ • https://sklearn.org/user_guide.html
• https://www.w3schools.com/