Introduction to Datamining
DATS 6103

Individual Final Report
Team-1
December 06, 2021
Sagar Tripathi

# 1.Introduction:

Many people are afraid of video games, even if they desperately try to relax. I am one of them. The game wasn't always stressful, but as life became more complex as we got older and the tendency to complete and control the game improved, the sense of security we gained from the game turned into horror. Fear can affect a person's life in many ways. Public health plays an important role in promoting people's well-being, maintaining safety and protection from infectious diseases and environmental risks, and ensuring people's safety and quality of treatment. The Internet is transforming the economy, education, government, medical care, and even the way we communicate with loved ones on a daily basis, making it one of the major drivers of social change. People started using the incredible internet and spent a lot of time playing online games on it for the development of the internet. People's mental and mental health is compromised by online games, causing a variety of mental illnesses. This dataset contains information collected from gamers' global surveys. In the survey, psychologists usually asked questions to ask people with anxiety, social phobia, and little or no life satisfaction. The questionnaire consists of a series of questions asked during a psychological test. Use different features of the model when analyzing the impact of online games on generalized anxiety disorder, including GAD, games, playstyles, platforms to play, age, gender, working hours, place of work, and place of residence. .. This project is demonstrated by developing a GUI-based application that represents end-to-end modeling using three machine learning algorithms: a random forest classifier, a decision tree, and a support vector machine. Dataset cleanup, exploratory data analysis, preprocessing, modeling, model comparison, GUI creation, Power Point presentation preparation, group report creation, and GUI demo generation were part of the collaboration.

# 2.Personal Contribution:

Code:

Pre-processing:

1. Mapping Data
2. Dropped few columns as it was not relevant to the prediction.
3. Finding missing values
4. Done the heatmap for GAD Disorder

Eda analysis:

1. Built the box plot (whisker plot) for hours, age, and GAD_T to find out the outliers in the dataset.
2. Built histogram, Scatter plot, Heatmaps.

Model Building:

1. Built the PYQT5 framework and assembled all the codes.
2. Building of confusion matrix in all three models.

# PYQT5.

PyQt is a Python binding for Qt that provides C ++ libraries and development tools that provide platform-independent graphical user interface (GUI) abstractions, networks, threads, regular expressions, SQL databases, SVG, OpenGL, and XML. It is a set. PyQt5 is based on Qtv5 and includes classes covering graphical user interfaces, XML processing, network communication, regular expressions, threads, SQL databases, multimedia, web browsing, and other technologies available in Qt.

PyQt5 implements these Qt classes in a number of Python modules, all of which are contained in a top-level Python package called PyQt5. PyQt5 is compatible with Windows, Unix, Linux, macOS, iOS, and Android. This can be a compelling feature if you are looking for a library or framework for developing multi-platform applications with a native look and feel on any platform. The PyQt5 license must be compatible with the Qt license. If you are using the GPL version, your code must also use a GPL-compliant license. If you want to build a commercial application using PyQt5, you need a commercial license for installation. There are several options to choose from when installing PyQt on your system or development environment. The first option is to build from source. This can be a bit tricky and should be avoided as much as possible. If you really need to build from source code, you can refer to the recommendations in the library documentation in these cases.

Another option is to use a binary wheel. Wheels are a very common way to manage the installation of Python packages. However, it should be considered that the PyQt5 wheel is only available in Python 3.5 and above. There is a wheel: Linux (64 bit) Mac OS Windows (32-bit and 64-bit) All of these wheels come with a copy of the corresponding Qt library and do not need to be installed separately. The third option is to use a package manager on Linux distributions and macOS. On Windows, you can use binary .exe files. The fourth and final option is to install PyQt on your system using the anaconda distribution. The following sections describe some options for properly installing PyQt 5 from different sources on different platforms.

## **Presentation**:

Finding the dataset, structuring the application, and verifying the results of all the models in the application, gathering knowledge about generalized anxiety disorder.

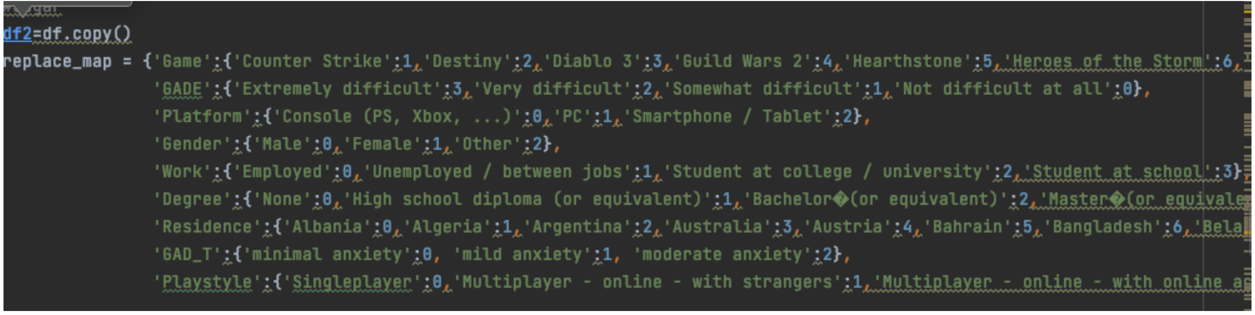
Generalized anxiety disorder (or GAD) is characterized by excessively exaggerated fear and anxiety about everyday events for no apparent reason. People with symptoms of generalized anxiety disorder are always prone to anticipate disasters and cannot stop worrying about their health, money, family, work, or school. Everyone is sometimes afraid, and for good reason. However, for people with generalized anxiety disorder, concerns are often unrealistic or not proportional to the situation. Everyday life is always worried, afraid, and anxious. After all, fear can even dominate one's thoughts and can even find it difficult to complete routine tasks in the workplace, school, social environment, and relationships. However, there are treatments available to help relieve anxiety so that it does not dominate your life.

## Group Final Report:

I worked on a few aspects of data pre-processing, EDA analysis, and finalizing the classes and building confusion matrix.

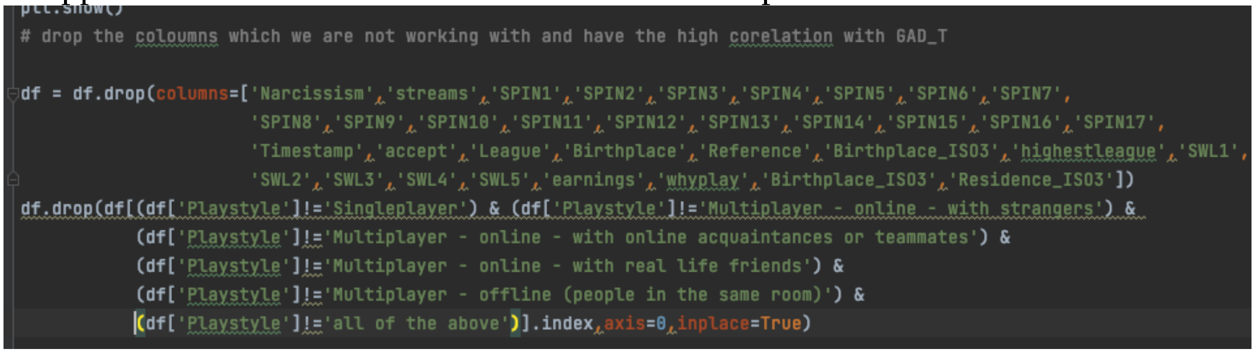
# 3.Personal Contribution in detail:

1. Mapping Data



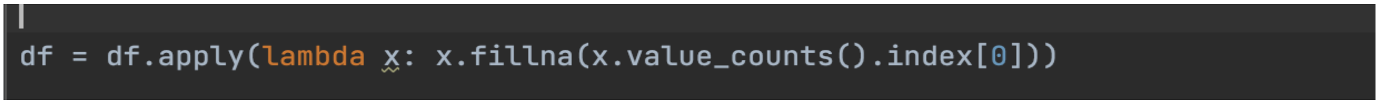
```
df2=df.copy()
replace_map = {'Game':{'Counter Strike':1,'Destiny':2,'Diablo 3':3,'Guild Wars 2':4,'Hearthstone':5,'Heroes of the Storm':6,
               'GADE':{'Extremely difficult':3,'Very difficult':2,'Somewhat difficult':1,'Not difficult at all':0},
               'Platform':{'Console (PS, Xbox, ...)':0,'PC':1,'Smartphone / Tablet':2},
               'Gender':{'Male':0,'Female':1,'Other':2},
               'Work':{'Employed':0,'Unemployed / between jobs':1,'Student at college / university':2,'Student at school':3},
               'Degree':{'None':0,'High school diploma (or equivalent)':1,'Bachelor�(or equivalent)':2,'Master�(or equivale
               'Residence':{'Albania':0,'Algeria':1,'Argentina':2,'Australia':3,'Austria':4,'Bahrain':5,'Bangladesh':6,'Bela
               'GAD_T':{'minimal anxiety':0, 'mild anxiety':1, 'moderate anxiety':2},
               'Playstyle':{'Singleplayer':0,'Multiplayer - online - with strangers':1,'Multiplayer - online - with online a
```

2. Dropped few columns as it was not relevant to the prediction.



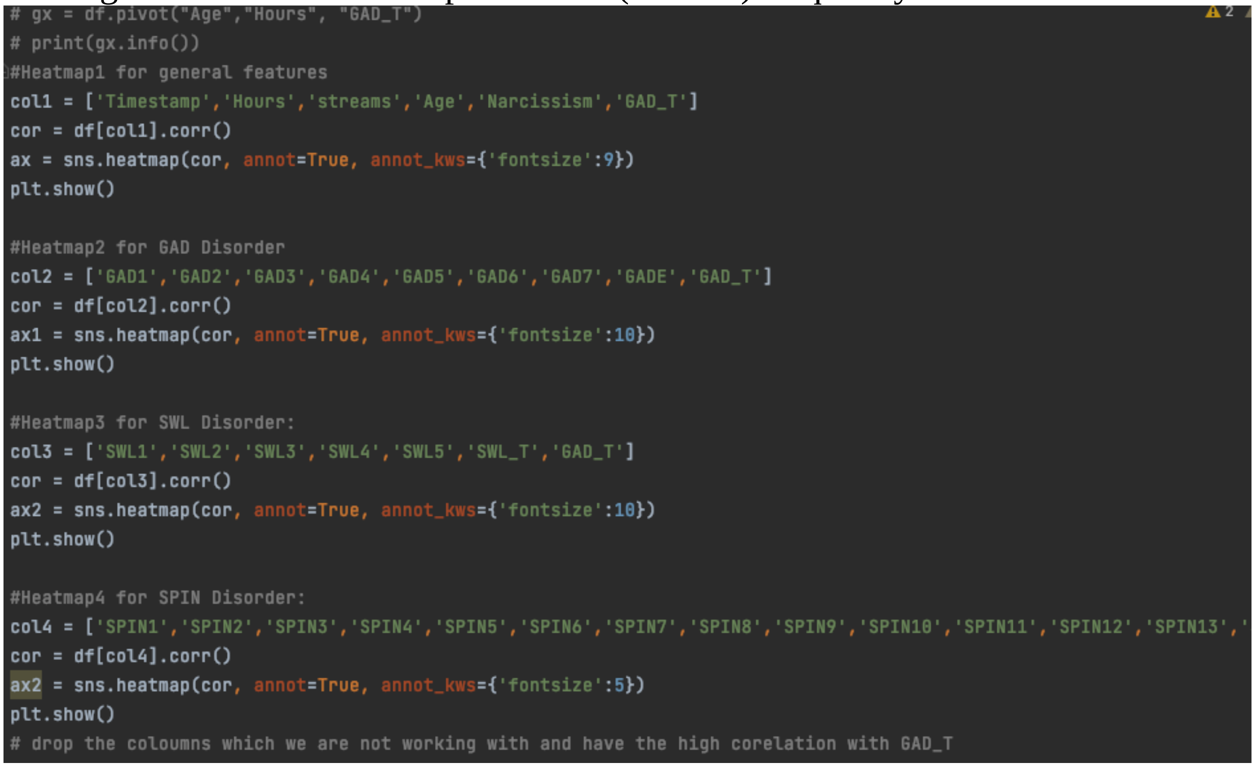
```
plt.show()
# drop the coloumns which we are not working with and have the high corelation with GAD_T

df = df.drop(columns=['Narcissism','streams','SPIN1','SPIN2','SPIN3','SPIN4','SPIN5','SPIN6','SPIN7',
                      'SPIN8','SPIN9','SPIN10','SPIN11','SPIN12','SPIN13','SPIN14','SPIN15','SPIN16','SPIN17',
                      'Timestamp','accept','League','Birthplace','Reference','Birthplace_ISO3','highestleague','SWL1',
                      'SWL2','SWL3','SWL4','SWL5','earnings','whyplay','Birthplace_ISO3','Residence_ISO3'])
df.drop(df[(df['Playstyle']!='Singleplayer') & (df['Playstyle']!='Multiplayer - online - with strangers') &
           (df['Playstyle']!='Multiplayer - online - with online acquaintances or teammates') &
           (df['Playstyle']!='Multiplayer - online - with real life friends') &
           (df['Playstyle']!='Multiplayer - offline (people in the same room)') &
           (df['Playstyle']!='all of the above')].index,axis=0,inplace=True)
```

3. Because most of the variables with missing values are categorical data, I filled in the blank values using the most often occurring phrase in that column.

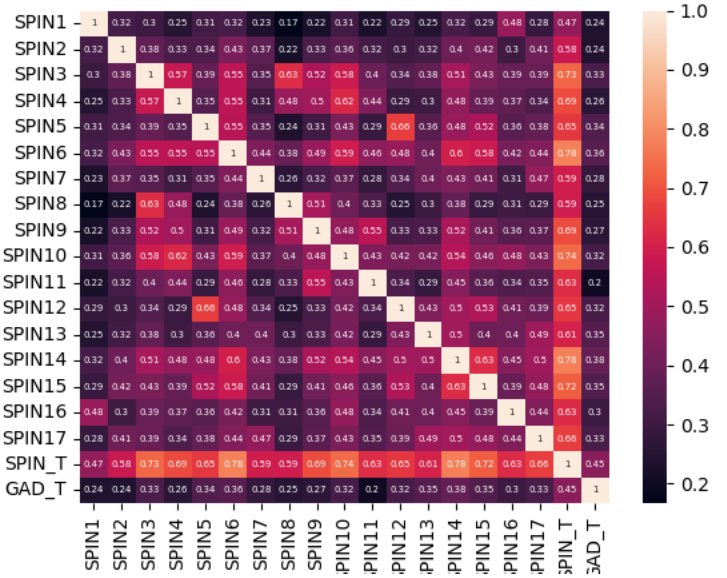


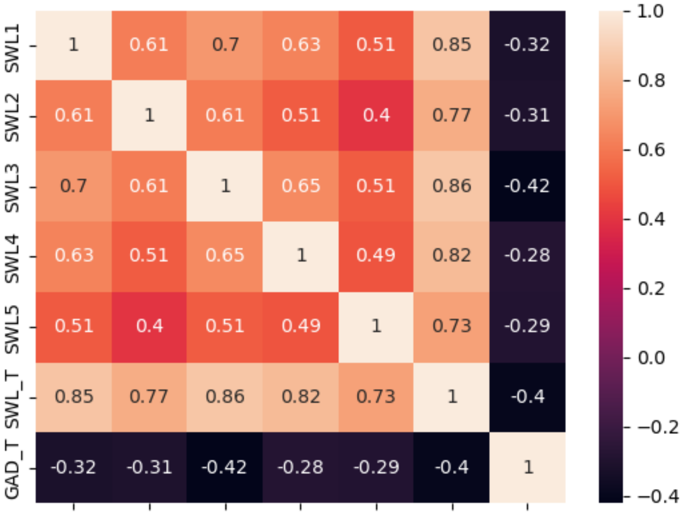
```
df = df.apply(lambda x: x.fillna(x.value_counts().index[0]))
```

4. Done the heatmap : A Heat plot is a graph that shows some aspects of the data  as gradients. A simple example is a bivariate histogram. Color gradients are used to represent the (relative) frequency in the X and Y bins.



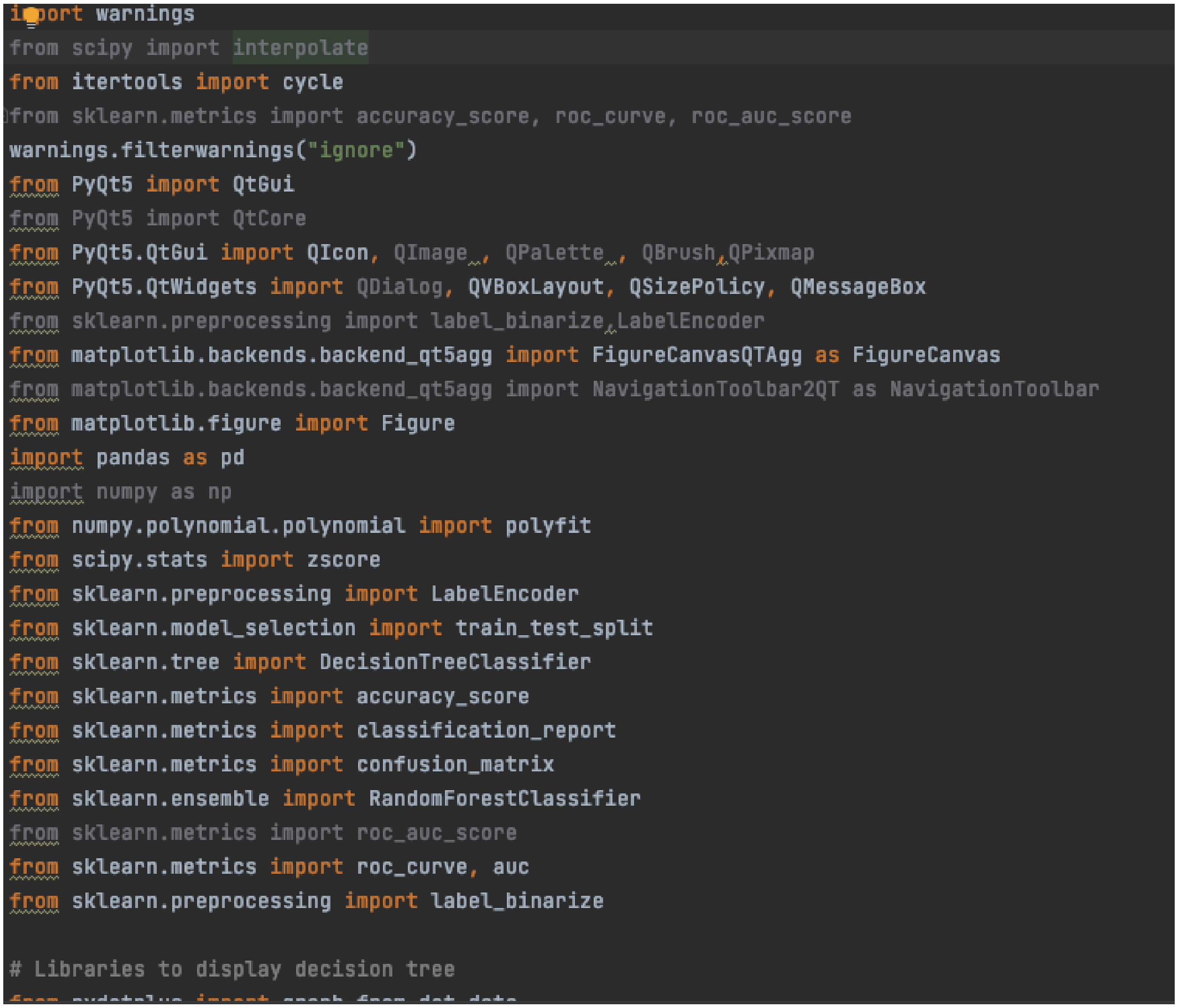
```
# gx = df.pivot("Age","Hours", "GAD_T")
# print(gx.info())
#Heatmap1 for general features
col1 = ['Timestamp','Hours','streams','Age','Narcissism','GAD_T']
cor = df[col1].corr()
ax = sns.heatmap(cor, annot=True, annot_kws={'fontsize':9})
plt.show()

#Heatmap2 for GAD Disorder
col2 = ['GAD1','GAD2','GAD3','GAD4','GAD5','GAD6','GAD7','GADE','GAD_T']
cor = df[col2].corr()
ax1 = sns.heatmap(cor, annot=True, annot_kws={'fontsize':10})
plt.show()

#Heatmap3 for SWL Disorder:
col3 = ['SWL1','SWL2','SWL3','SWL4','SWL5','SWL_T','GAD_T']
cor = df[col3].corr()
ax2 = sns.heatmap(cor, annot=True, annot_kws={'fontsize':10})
plt.show()

#Heatmap4 for SPIN Disorder:
col4 = ['SPIN1','SPIN2','SPIN3','SPIN4','SPIN5','SPIN6','SPIN7','SPIN8','SPIN9','SPIN10','SPIN11','SPIN12','SPIN13','
cor = df[col4].corr()
ax2 = sns.heatmap(cor, annot=True, annot_kws={'fontsize':5})
plt.show()
# drop the coloumns which we are not working with and have the high corelation with GAD_T
```


SPIN1 1 0.32 0.3 0.25 0.31 0.32 0.23 0.17 0.22 0.31 0.22 0.29 0.25 0.32 0.29 0.48 0.28 0.47 0.24
SPIN2 0.32 1 0.38 0.33 0.34 0.43 0.37 0.22 0.33 0.36 0.32 0.3 0.32 0.4 0.42 0.3 0.41 0.58 0.24
SPIN3 0.3 0.38 1 0.57 0.39 0.55 0.35 0.63 0.52 0.58 0.4 0.34 0.38 0.51 0.43 0.39 0.39 0.73 0.33
SPIN4 0.25 0.33 0.57 1 0.35 0.55 0.31 0.48 0.5 0.62 0.44 0.29 0.3 0.48 0.39 0.37 0.34 0.69 0.26
SPIN5 0.31 0.34 0.39 0.35 1 0.55 0.35 0.24 0.31 0.43 0.29 0.66 0.36 0.48 0.52 0.36 0.38 0.65 0.34
SPIN6 0.32 0.43 0.55 0.55 0.55 1 0.44 0.38 0.49 0.59 0.46 0.48 0.4 0.6 0.58 0.42 0.44 0.78 0.36
SPIN7 0.23 0.37 0.35 0.31 0.35 0.44 1 0.26 0.32 0.37 0.28 0.34 0.4 0.43 0.41 0.31 0.47 0.59 0.28
SPIN8 0.17 0.22 0.63 0.48 0.24 0.38 0.26 1 0.51 0.4 0.33 0.25 0.3 0.38 0.29 0.31 0.29 0.59 0.25
SPIN9 0.22 0.33 0.52 0.5 0.31 0.49 0.32 0.51 1 0.48 0.55 0.33 0.33 0.52 0.41 0.36 0.37 0.69 0.27
SPIN10 0.31 0.36 0.58 0.62 0.43 0.59 0.37 0.4 0.48 1 0.43 0.42 0.42 0.54 0.46 0.48 0.43 0.74 0.32
SPIN11 0.22 0.32 0.4 0.44 0.29 0.46 0.28 0.33 0.55 0.43 1 0.34 0.29 0.45 0.36 0.34 0.35 0.63 0.2
SPIN12 0.29 0.3 0.34 0.29 0.66 0.48 0.34 0.25 0.33 0.42 0.34 1 0.43 0.5 0.53 0.41 0.39 0.65 0.32
SPIN13 0.25 0.32 0.38 0.3 0.36 0.4 0.4 0.3 0.33 0.42 0.29 0.43 1 0.5 0.4 0.4 0.49 0.61 0.35
SPIN14 0.32 0.4 0.51 0.48 0.48 0.6 0.43 0.38 0.52 0.54 0.45 0.5 0.5 1 0.63 0.45 0.5 0.78 0.38
SPIN15 0.29 0.42 0.43 0.39 0.52 0.58 0.41 0.29 0.41 0.46 0.36 0.53 0.4 0.63 1 0.39 0.48 0.72 0.35
SPIN16 0.48 0.3 0.39 0.37 0.36 0.42 0.31 0.31 0.36 0.48 0.34 0.41 0.4 0.45 0.39 1 0.44 0.63 0.3
SPIN17 0.28 0.41 0.39 0.34 0.38 0.44 0.47 0.29 0.37 0.43 0.35 0.39 0.49 0.5 0.48 0.44 1 0.66 0.33
SPIN_T 0.47 0.58 0.73 0.69 0.65 0.78 0.59 0.59 0.69 0.74 0.63 0.65 0.61 0.78 0.72 0.63 0.66 1 0.45
GAD_T 0.24 0.24 0.33 0.26 0.34 0.36 0.28 0.25 0.27 0.32 0.2 0.32 0.35 0.38 0.35 0.3 0.33 0.45 1
SPIN1
SPIN2
SPIN3
SPIN4
SPIN5
SPIN6
SPIN7
SPIN8
SPIN9
PIN10
PIN11
PIN12
PIN13
PIN14
PIN15
PIN16
PIN17
SPIN_T
GAD_T
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

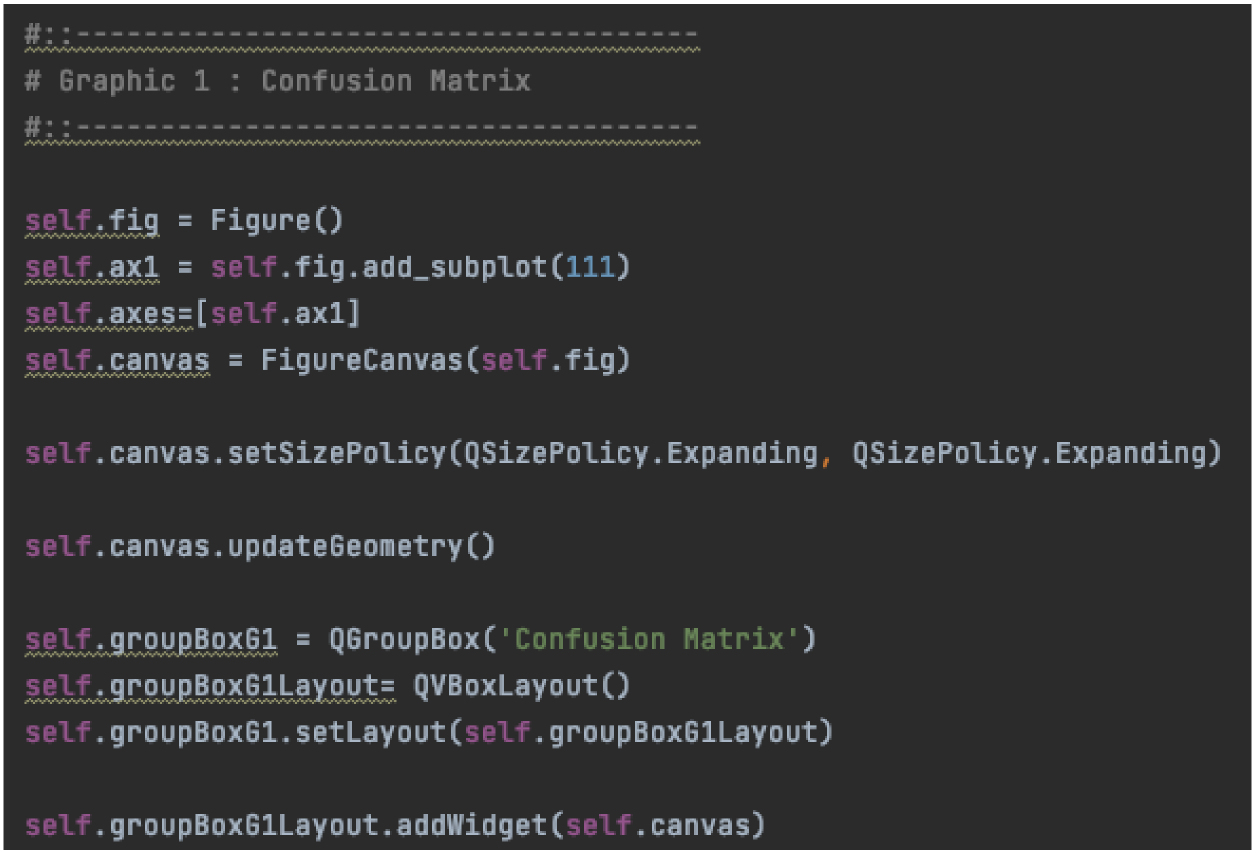

SWL1 1 0.61 0.7 0.63 0.51 0.85 -0.32
SWL2 0.61 1 0.61 0.51 0.4 0.77 -0.31
SWL3 0.7 0.61 1 0.65 0.51 0.86 -0.42
SWL4 0.63 0.51 0.65 1 0.49 0.82 -0.28
SWL5 0.51 0.4 0.51 0.49 1 0.73 -0.29
SWL_T 0.85 0.77 0.86 0.82 0.73 1 -0.4
GAD_T -0.32 -0.31 -0.42 -0.28 -0.29 -0.4 1
SWL1
SWL2
SWL3
SWL4
SWL5
SWL_T
GAD_T
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

# Model Building (PYQT5 code):

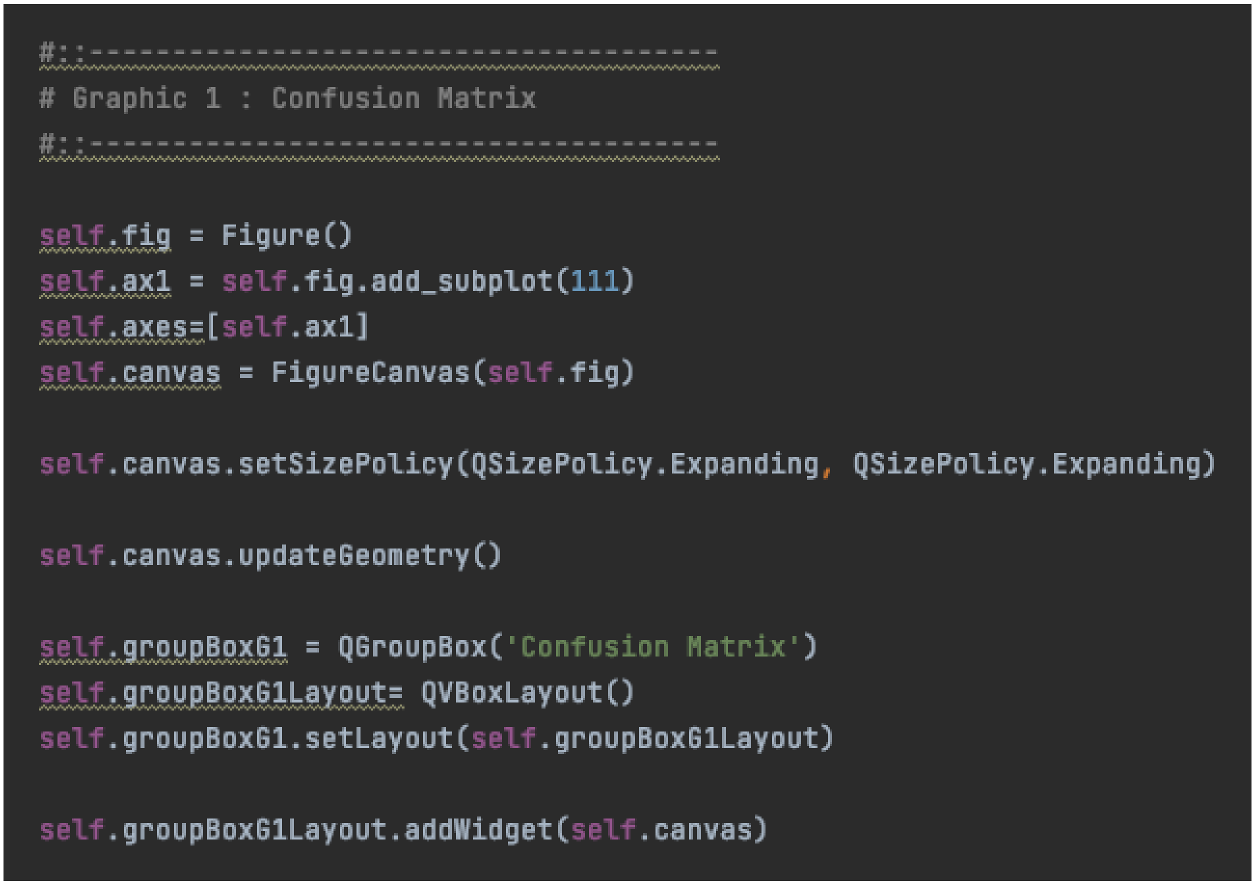
- Structuring the model
- Data-processing
- Data cleaning
- Data mapping
- EDA Analysis
- Roc_Auc Curve



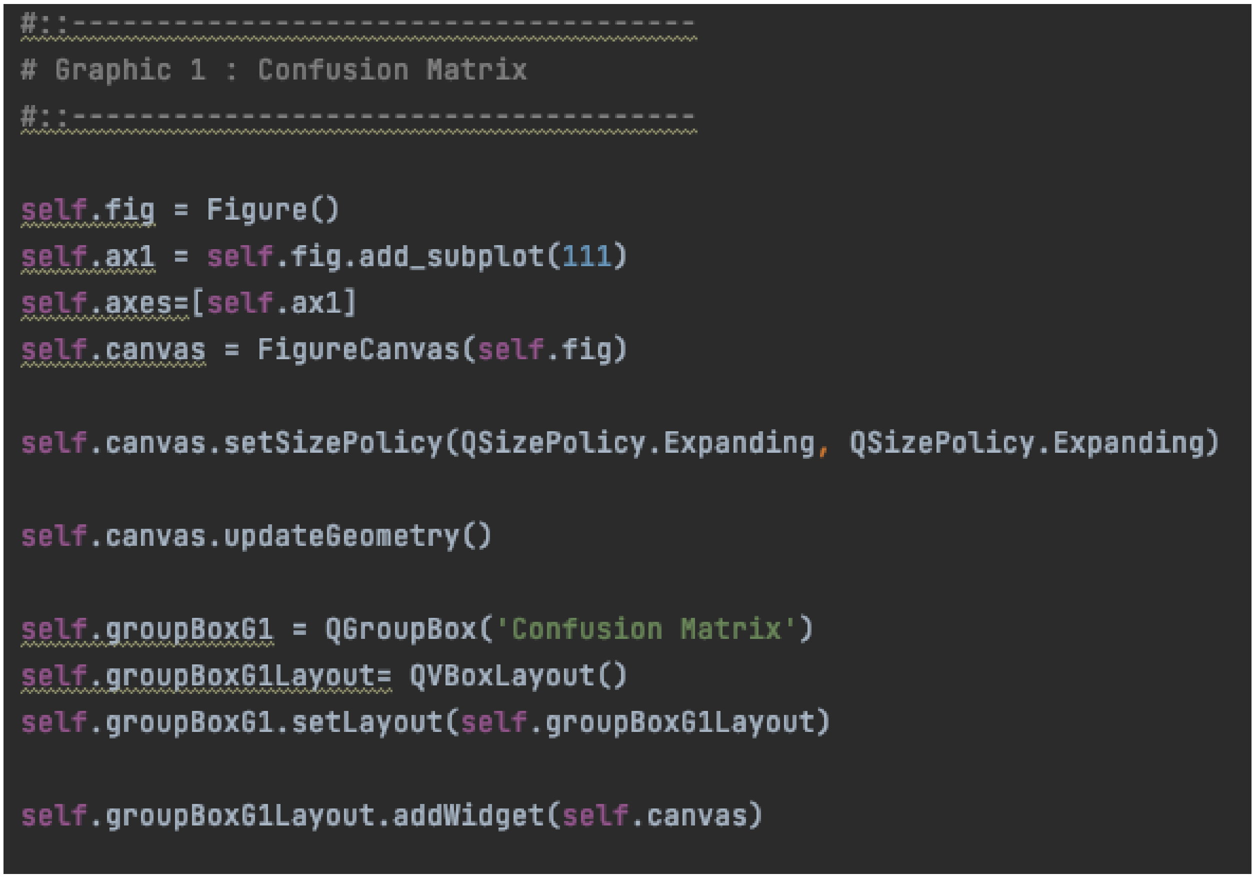
```
import warnings
from scipy import interpolate
from itertools import cycle
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
warnings.filterwarnings("ignore")
from PyQt5 import QtGui
from PyQt5 import QtCore
from PyQt5.QtGui import QIcon, QImage , QPalette , QBrush,QPixmap
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QSizePolicy, QMessageBox
from sklearn.preprocessing import label_binarize,LabelEncoder
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar
from matplotlib.figure import Figure
import pandas as pd
import numpy as np
from numpy.polynomial.polynomial import polyfit
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Libraries to display decision tree
from pydotplus import graph_from_dot_data
```



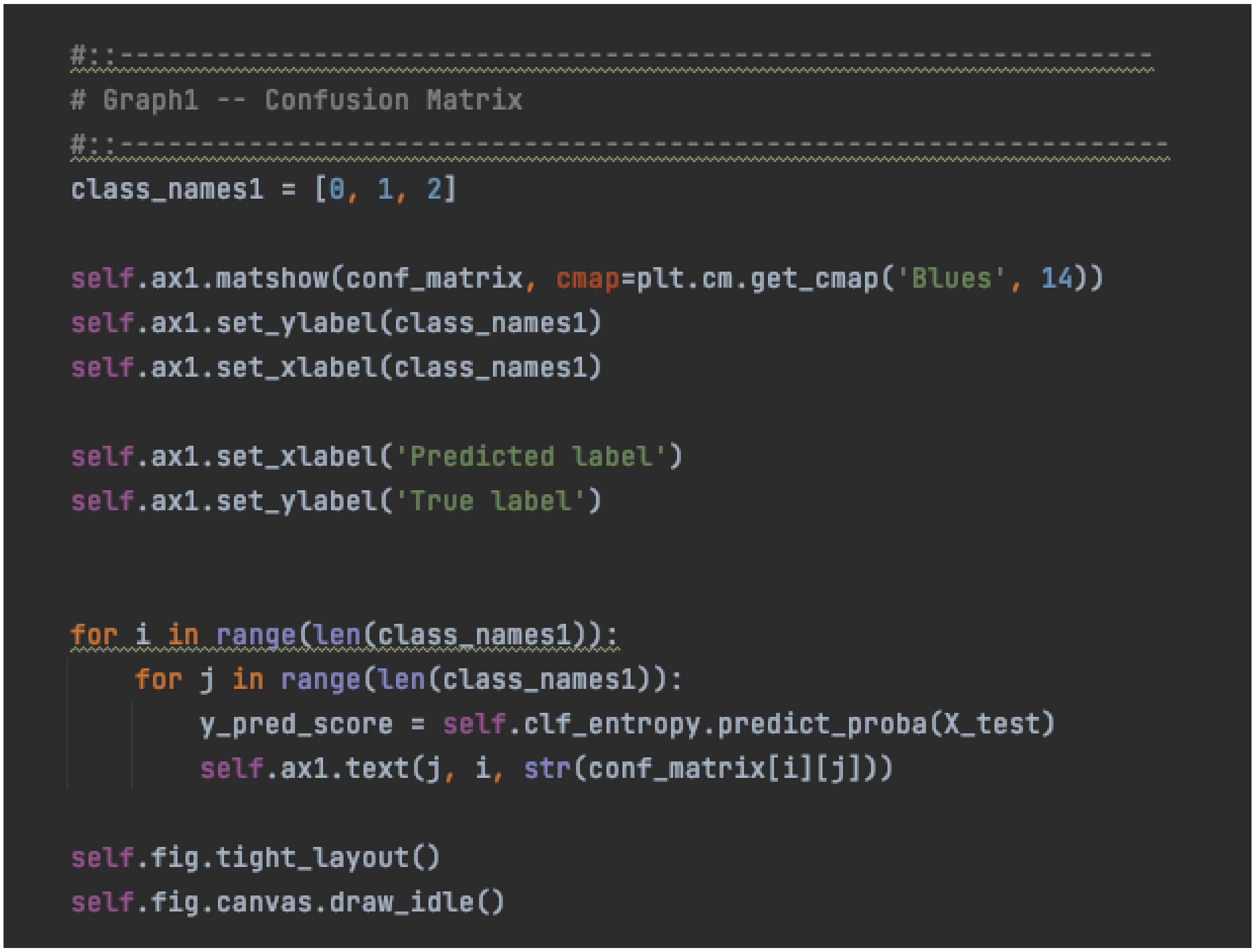
```
#::-------------------------------------
# Graphic 1 : Confusion Matrix
#::-------------------------------------

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)
```



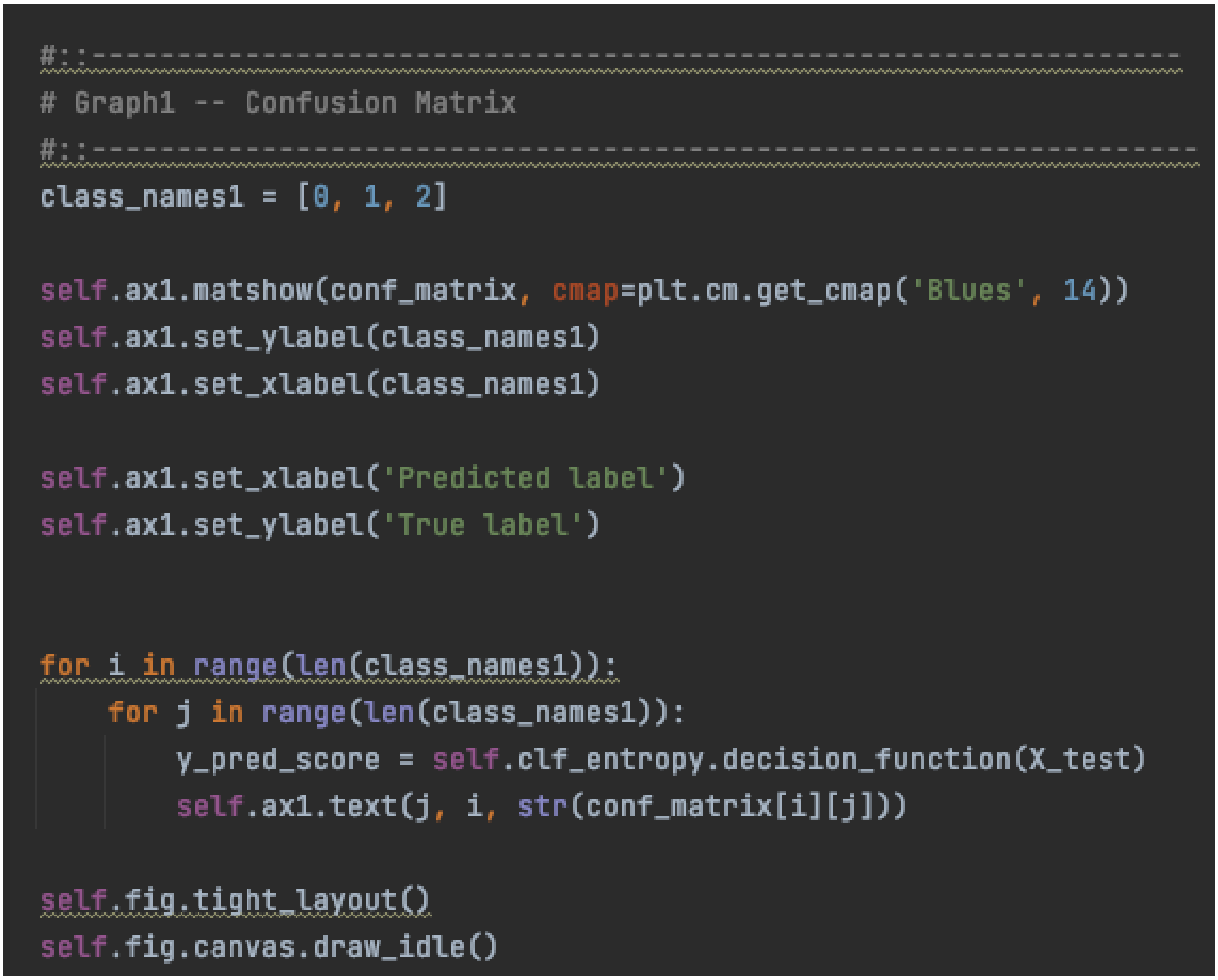
```
#::------------------------------------
# Graphic 1 : Confusion Matrix
#::------------------------------------

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)
```



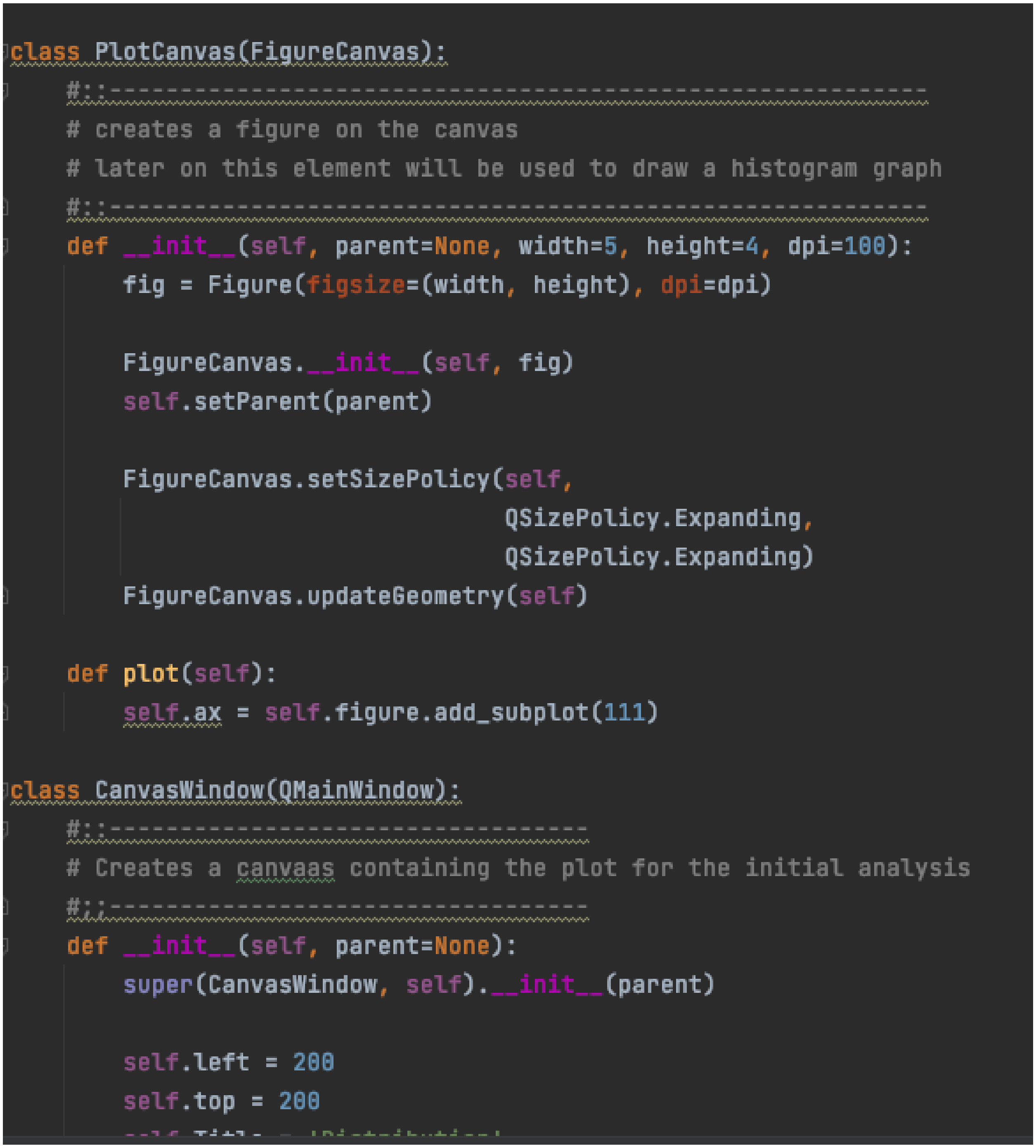
```python
#::-------------------------------------
# Graphic 1 : Confusion Matrix
#::-------------------------------------

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)
```



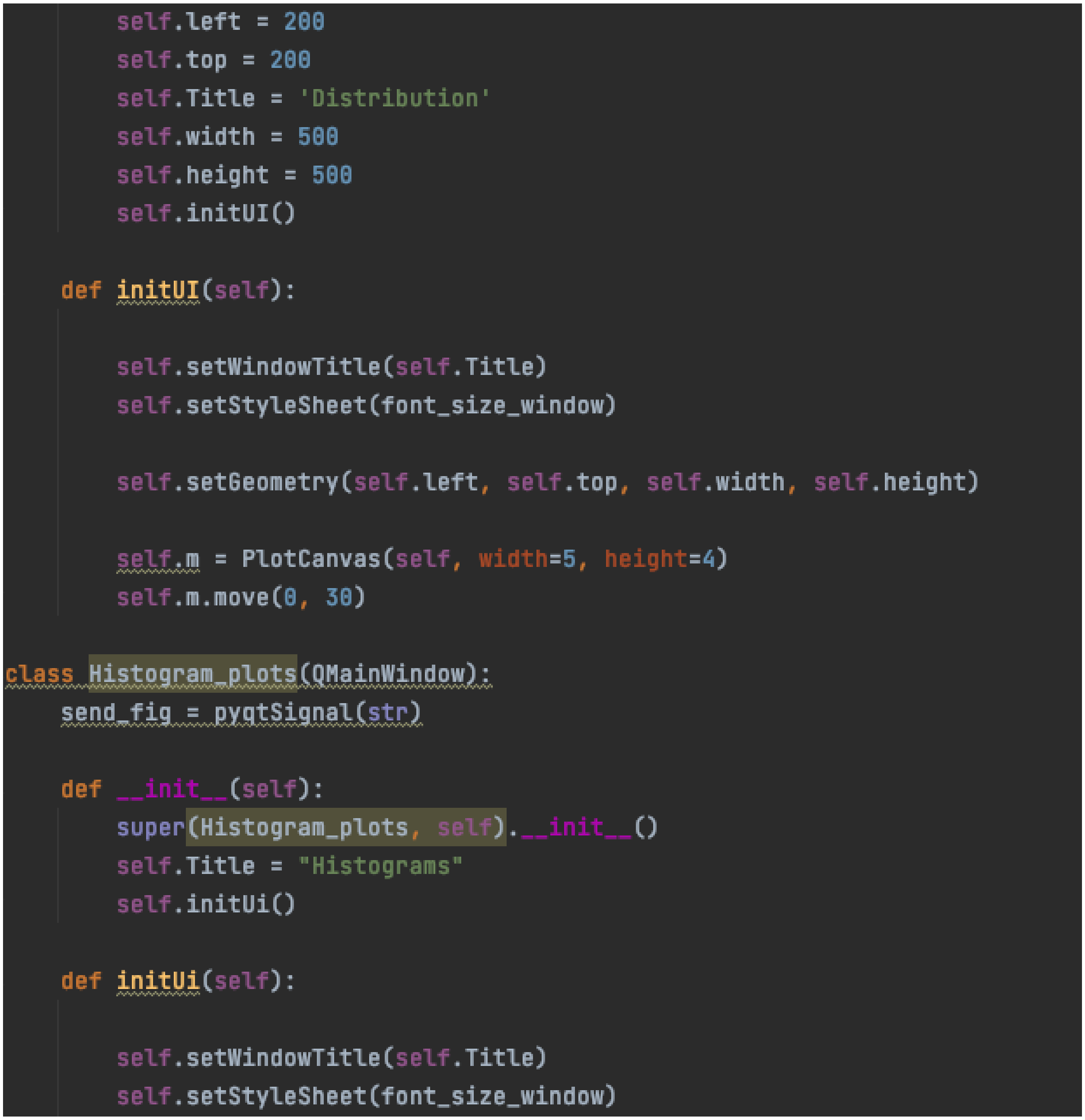
```
#::----------------------------------------------------------------
# Graph1 -- Confusion Matrix
#::-----------------------------------------------------------------
class_names1 = [0, 1, 2]

self.ax1.matshow(conf_matrix, cmap=plt.cm.get_cmap('Blues', 14))
self.ax1.set_ylabel(class_names1)
self.ax1.set_xlabel(class_names1)

self.ax1.set_xlabel('Predicted label')
self.ax1.set_ylabel('True label')


for i in range(len(class_names1)):
    for j in range(len(class_names1)):
        y_pred_score = self.clf_entropy.predict_proba(X_test)
        self.ax1.text(j, i, str(conf_matrix[i][j]))

self.fig.tight_layout()
self.fig.canvas.draw_idle()
```



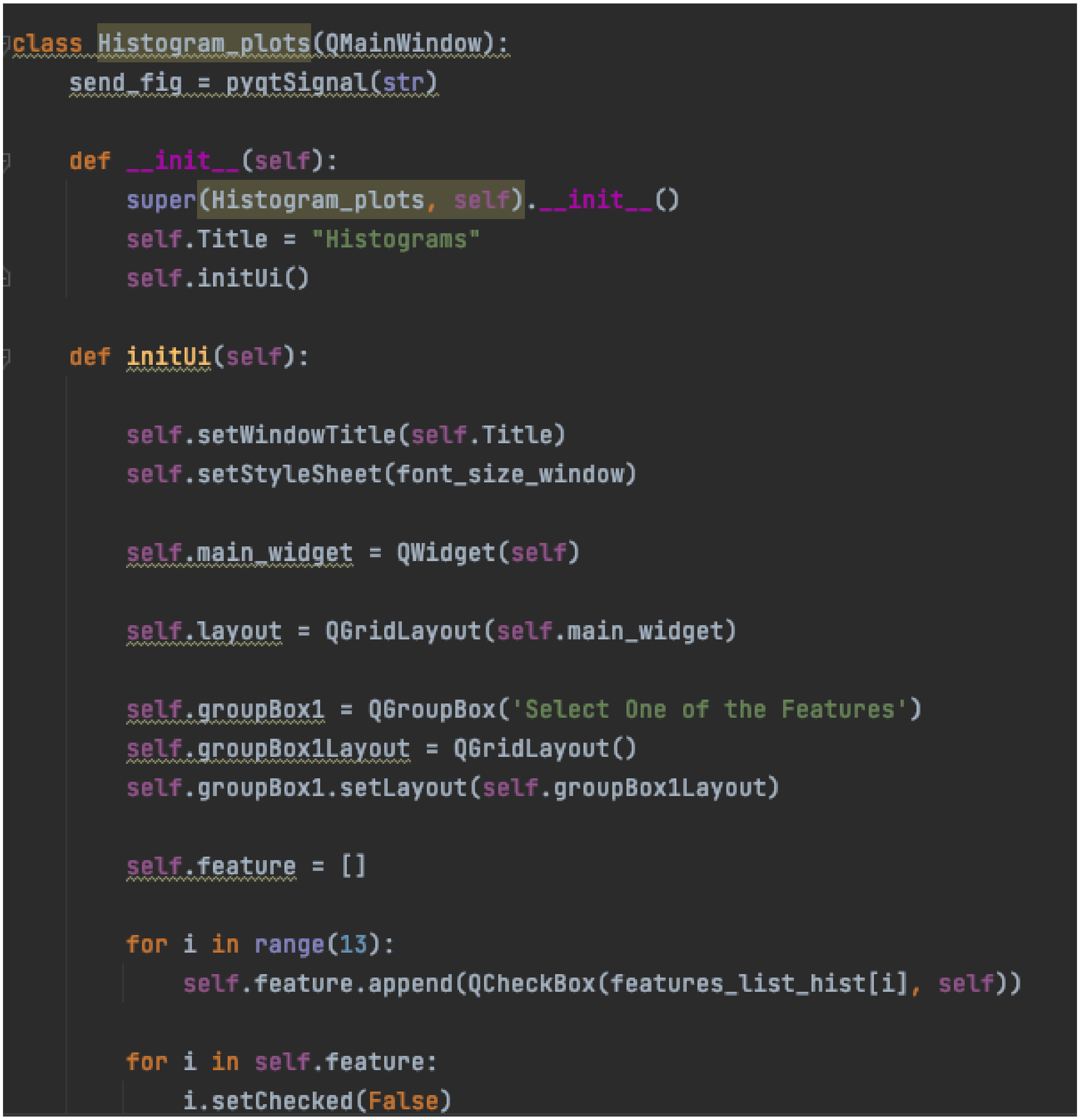
```
#::----------------------------------------------------------------
# Graph1 -- Confusion Matrix
#::-----------------------------------------------------------------
class_names1 = [0, 1, 2]

self.ax1.matshow(conf_matrix, cmap=plt.cm.get_cmap('Blues', 14))
self.ax1.set_ylabel(class_names1)
self.ax1.set_xlabel(class_names1)

self.ax1.set_xlabel('Predicted label')
self.ax1.set_ylabel('True label')


for i in range(len(class_names1)):
    for j in range(len(class_names1)):
        y_pred_score = self.clf_entropy.decision_function(X_test)
        self.ax1.text(j, i, str(conf_matrix[i][j]))

self.fig.tight_layout()
self.fig.canvas.draw_idle()
```



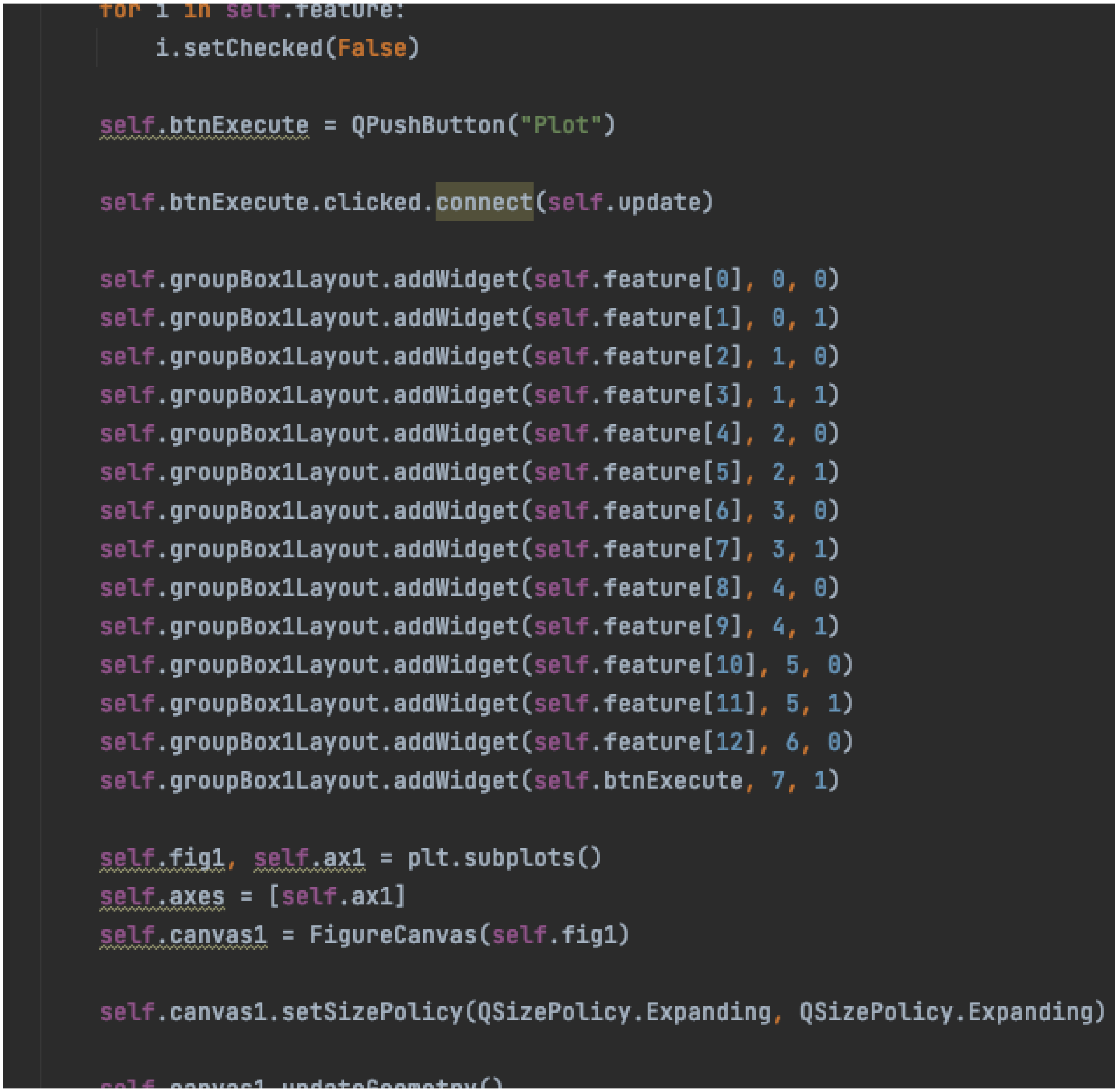
```
class PlotCanvas(FigureCanvas):
    #::----------------------------------------------------------
    # creates a figure on the canvas
    # later on this element will be used to draw a histogram graph
    #::----------------------------------------------------------
    def __init__(self, parent=None, width=5, height=4, dpi=100):
        fig = Figure(figsize=(width, height), dpi=dpi)

        FigureCanvas.__init__(self, fig)
        self.setParent(parent)

        FigureCanvas.setSizePolicy(self,
                                   QSizePolicy.Expanding,
                                   QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)

    def plot(self):
        self.ax = self.figure.add_subplot(111)

class CanvasWindow(QMainWindow):
    #::----------------------------------
    # Creates a canvaas containing the plot for the initial analysis
    #;;----------------------------------
    def __init__(self, parent=None):
        super(CanvasWindow, self).__init__(parent)

        self.left = 200
        self.top = 200
        self.Title = 'Distribution'
```



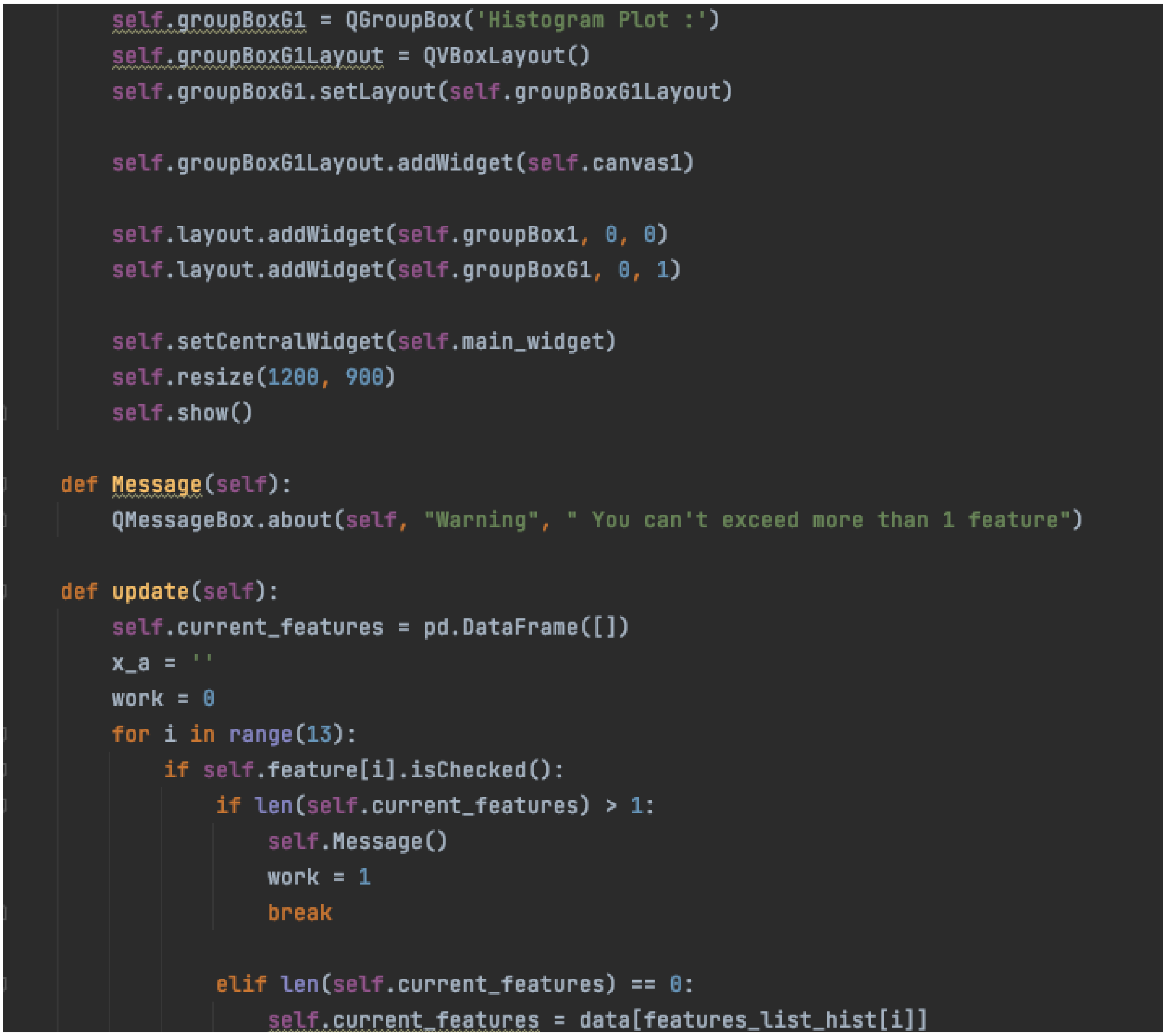
```
        self.left = 200
        self.top = 200
        self.Title = 'Distribution'
        self.width = 500
        self.height = 500
        self.initUI()

    def initUI(self):

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

        self.setGeometry(self.left, self.top, self.width, self.height)

        self.m = PlotCanvas(self, width=5, height=4)
        self.m.move(0, 30)

class Histogram_plots(QMainWindow):
    send_fig = pyqtSignal(str)

    def __init__(self):
        super(Histogram_plots, self).__init__()
        self.Title = "Histograms"
        self.initUi()

    def initUi(self):

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



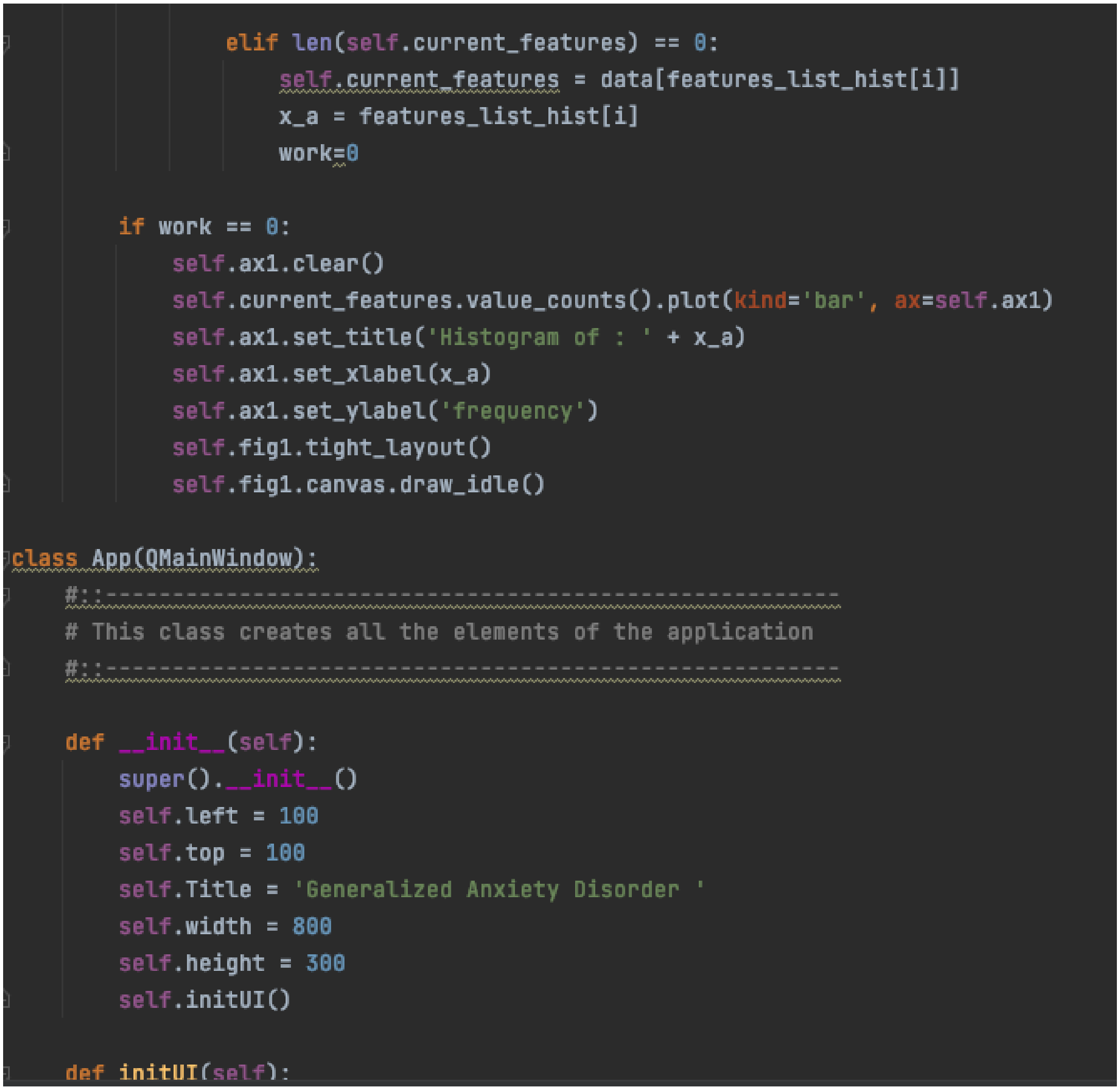
```python
class Histogram_plots(QMainWindow):
    send_fig = pyqtSignal(str)

    def __init__(self):
        super(Histogram_plots, self).__init__()
        self.Title = "Histograms"
        self.initUi()

    def initUi(self):

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

        self.main_widget = QWidget(self)

        self.layout = QGridLayout(self.main_widget)

        self.groupBox1 = QGroupBox('Select One of the Features')
        self.groupBox1Layout = QGridLayout()
        self.groupBox1.setLayout(self.groupBox1Layout)

        self.feature = []

        for i in range(13):
            self.feature.append(QCheckBox(features_list_hist[i], self))

        for i in self.feature:
            i.setChecked(False)
```



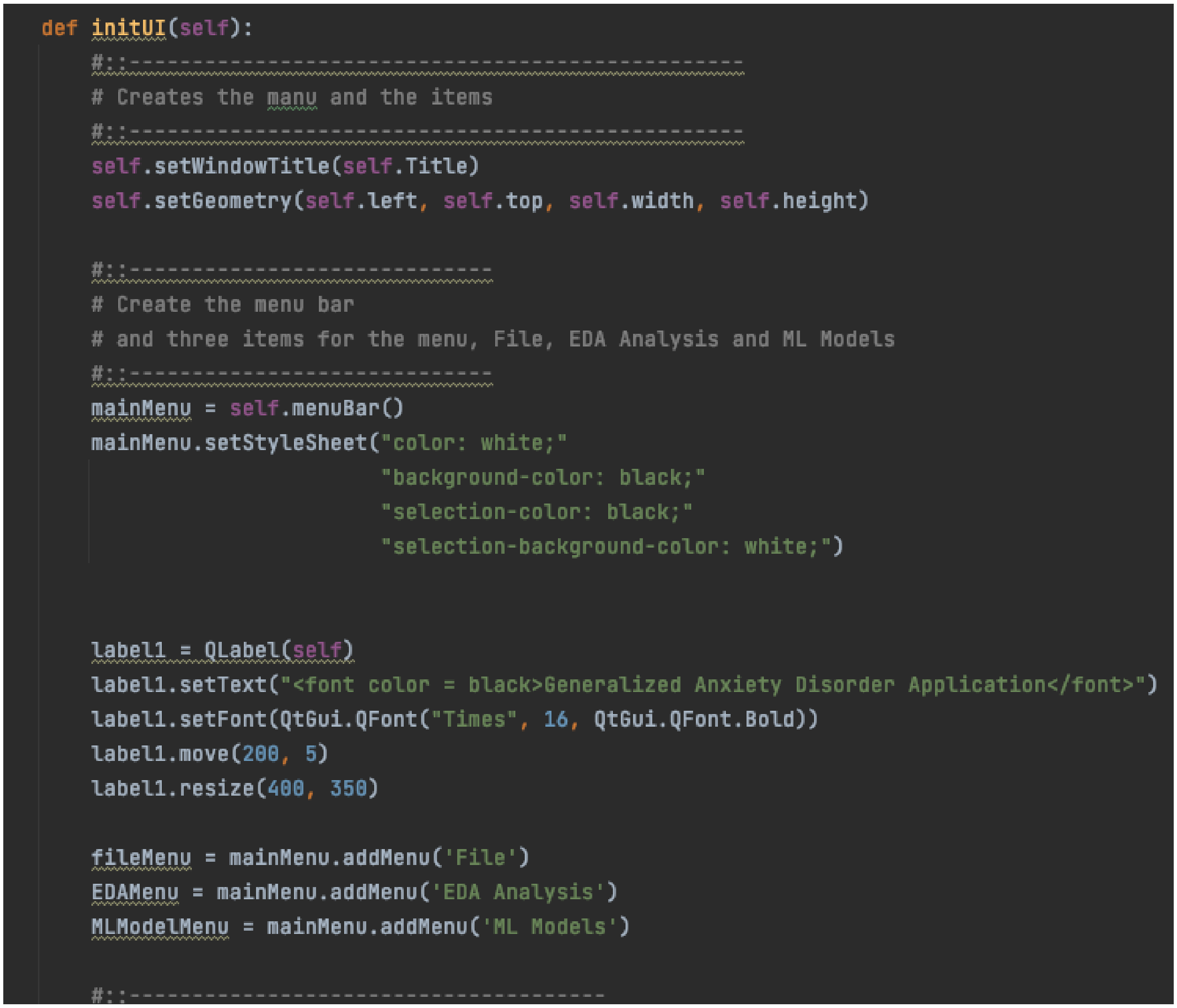
```
for i in self.feature:
    i.setChecked(False)

self.btnExecute = QPushButton("Plot")

self.btnExecute.clicked.connect(self.update)

self.groupBox1Layout.addWidget(self.feature[0], 0, 0)
self.groupBox1Layout.addWidget(self.feature[1], 0, 1)
self.groupBox1Layout.addWidget(self.feature[2], 1, 0)
self.groupBox1Layout.addWidget(self.feature[3], 1, 1)
self.groupBox1Layout.addWidget(self.feature[4], 2, 0)
self.groupBox1Layout.addWidget(self.feature[5], 2, 1)
self.groupBox1Layout.addWidget(self.feature[6], 3, 0)
self.groupBox1Layout.addWidget(self.feature[7], 3, 1)
self.groupBox1Layout.addWidget(self.feature[8], 4, 0)
self.groupBox1Layout.addWidget(self.feature[9], 4, 1)
self.groupBox1Layout.addWidget(self.feature[10], 5, 0)
self.groupBox1Layout.addWidget(self.feature[11], 5, 1)
self.groupBox1Layout.addWidget(self.feature[12], 6, 0)
self.groupBox1Layout.addWidget(self.btnExecute, 7, 1)

self.fig1, self.ax1 = plt.subplots()
self.axes = [self.ax1]
self.canvas1 = FigureCanvas(self.fig1)

self.canvas1.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

self.canvas1.updateGeometry()
```



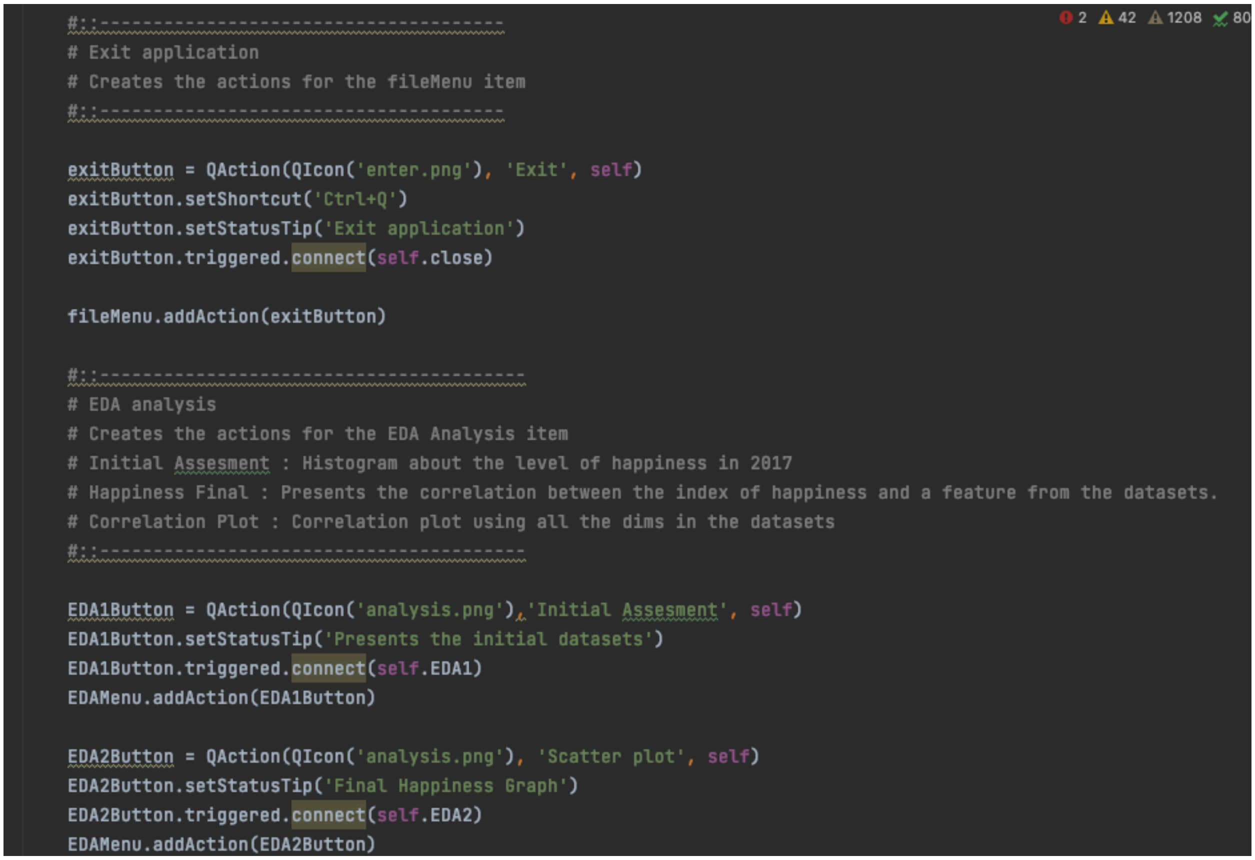
```
        self.groupBoxG1 = QGroupBox('Histogram Plot :')
        self.groupBoxG1Layout = QVBoxLayout()
        self.groupBoxG1.setLayout(self.groupBoxG1Layout)

        self.groupBoxG1Layout.addWidget(self.canvas1)

        self.layout.addWidget(self.groupBox1, 0, 0)
        self.layout.addWidget(self.groupBoxG1, 0, 1)

        self.setCentralWidget(self.main_widget)
        self.resize(1200, 900)
        self.show()

    def Message(self):
        QMessageBox.about(self, "Warning", " You can't exceed more than 1 feature")

    def update(self):
        self.current_features = pd.DataFrame([])
        x_a = ''
        work = 0
        for i in range(13):
            if self.feature[i].isChecked():
                if len(self.current_features) > 1:
                    self.Message()
                    work = 1
                    break

                elif len(self.current_features) == 0:
                    self.current_features = data[features_list_hist[i]]
```



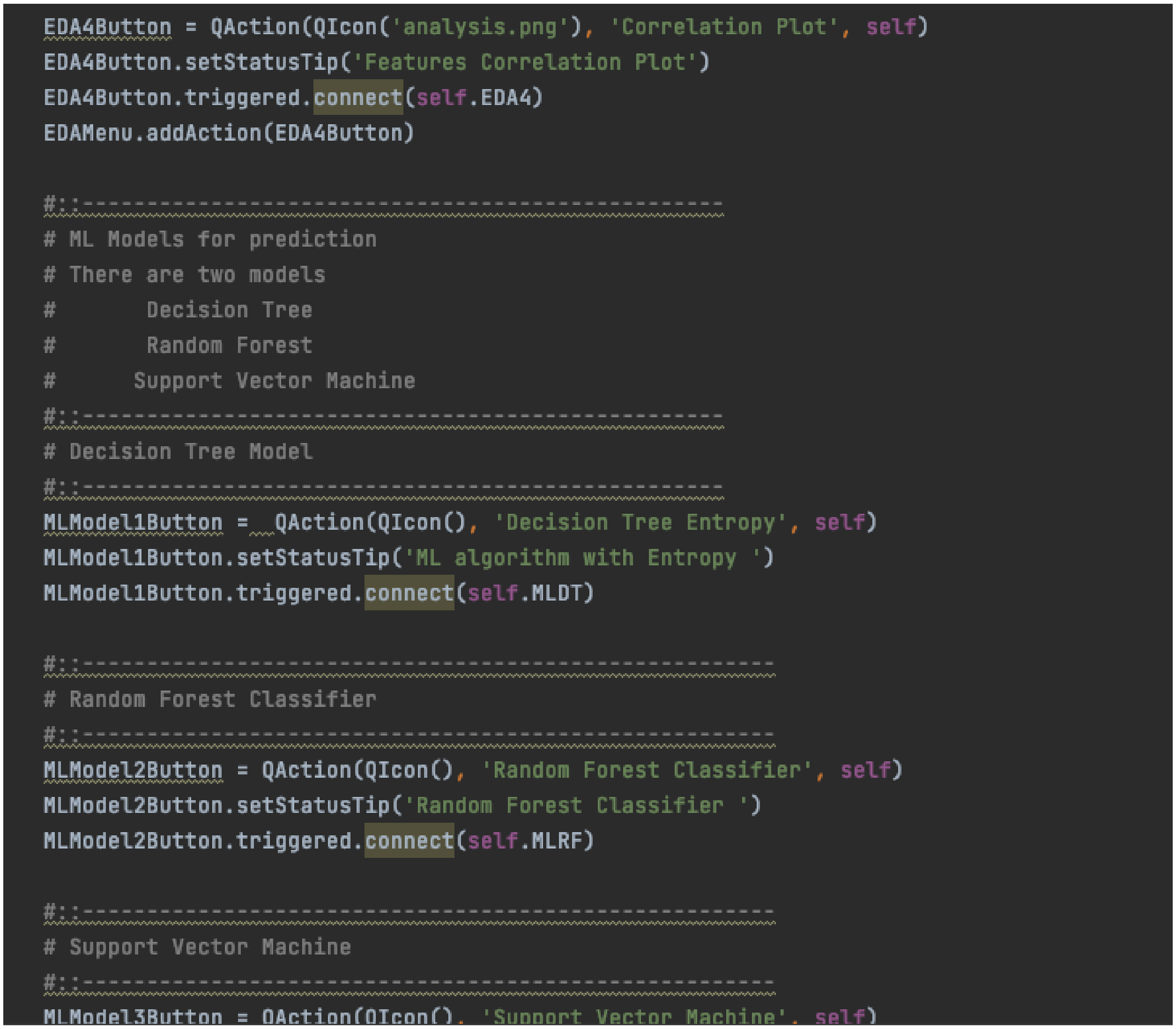
```
                elif len(self.current_features) == 0:
                    self.current_features = data[features_list_hist[i]]
                    x_a = features_list_hist[i]
                    work=0

        if work == 0:
            self.ax1.clear()
            self.current_features.value_counts().plot(kind='bar', ax=self.ax1)
            self.ax1.set_title('Histogram of : ' + x_a)
            self.ax1.set_xlabel(x_a)
            self.ax1.set_ylabel('frequency')
            self.fig1.tight_layout()
            self.fig1.canvas.draw_idle()

class App(QMainWindow):
    #::--------------------------------------------------------
    # This class creates all the elements of the application
    #::--------------------------------------------------------

    def __init__(self):
        super().__init__()
        self.left = 100
        self.top = 100
        self.Title = 'Generalized Anxiety Disorder '
        self.width = 800
        self.height = 300
        self.initUI()

    def initUI(self):
```



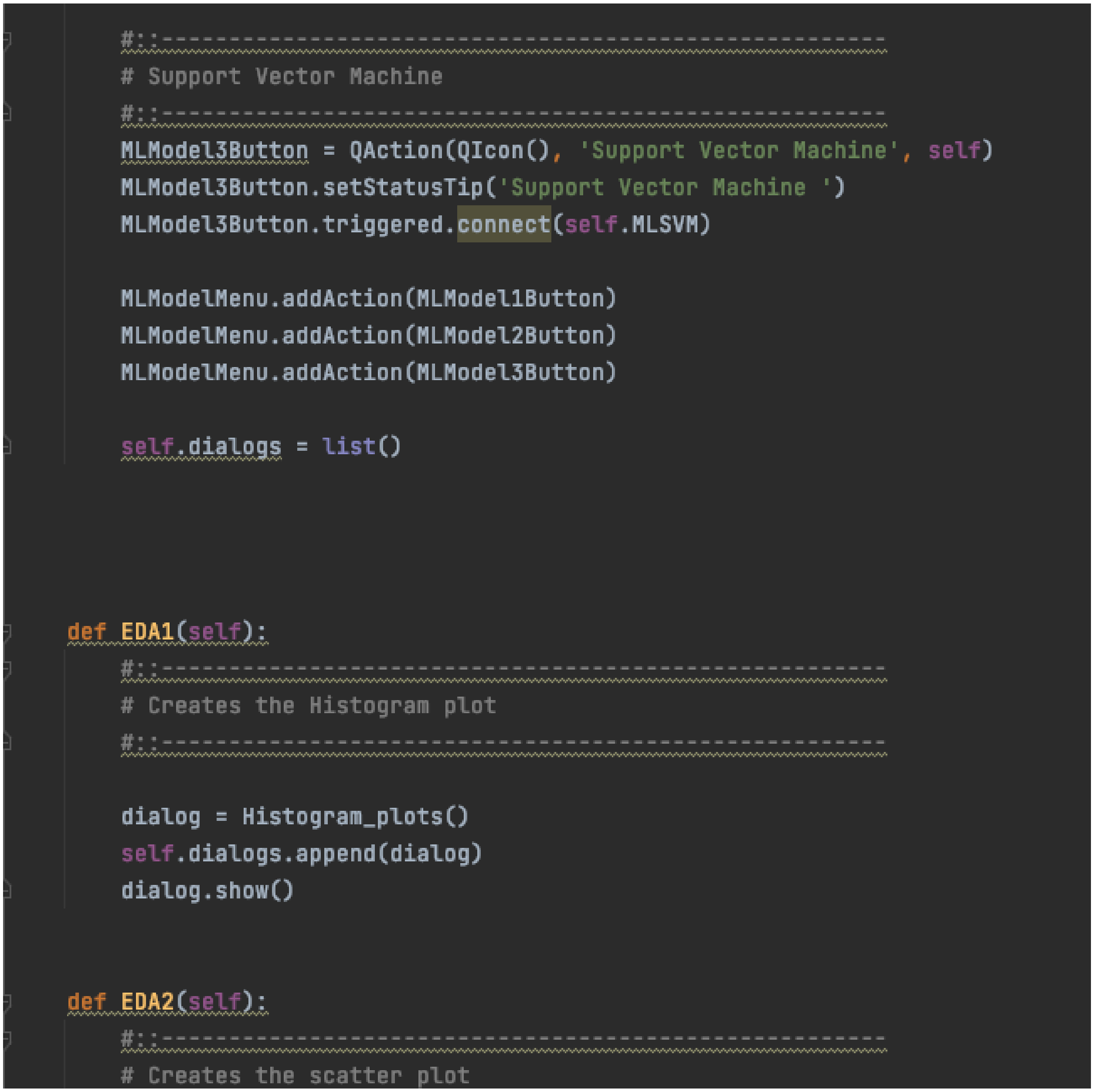
```
def initUI(self):
    #::-------------------------------------------------
    # Creates the manu and the items
    #::-------------------------------------------------
    self.setWindowTitle(self.Title)
    self.setGeometry(self.left, self.top, self.width, self.height)

    #::-----------------------------
    # Create the menu bar
    # and three items for the menu, File, EDA Analysis and ML Models
    #::-----------------------------
    mainMenu = self.menuBar()
    mainMenu.setStyleSheet("color: white;"
                           "background-color: black;"
                           "selection-color: black;"
                           "selection-background-color: white;")


    label1 = QLabel(self)
    label1.setText("<font color = black>Generalized Anxiety Disorder Application</font>")
    label1.setFont(QtGui.QFont("Times", 16, QtGui.QFont.Bold))
    label1.move(200, 5)
    label1.resize(400, 350)

    fileMenu = mainMenu.addMenu('File')
    EDAMenu = mainMenu.addMenu('EDA Analysis')
    MLModelMenu = mainMenu.addMenu('ML Models')

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



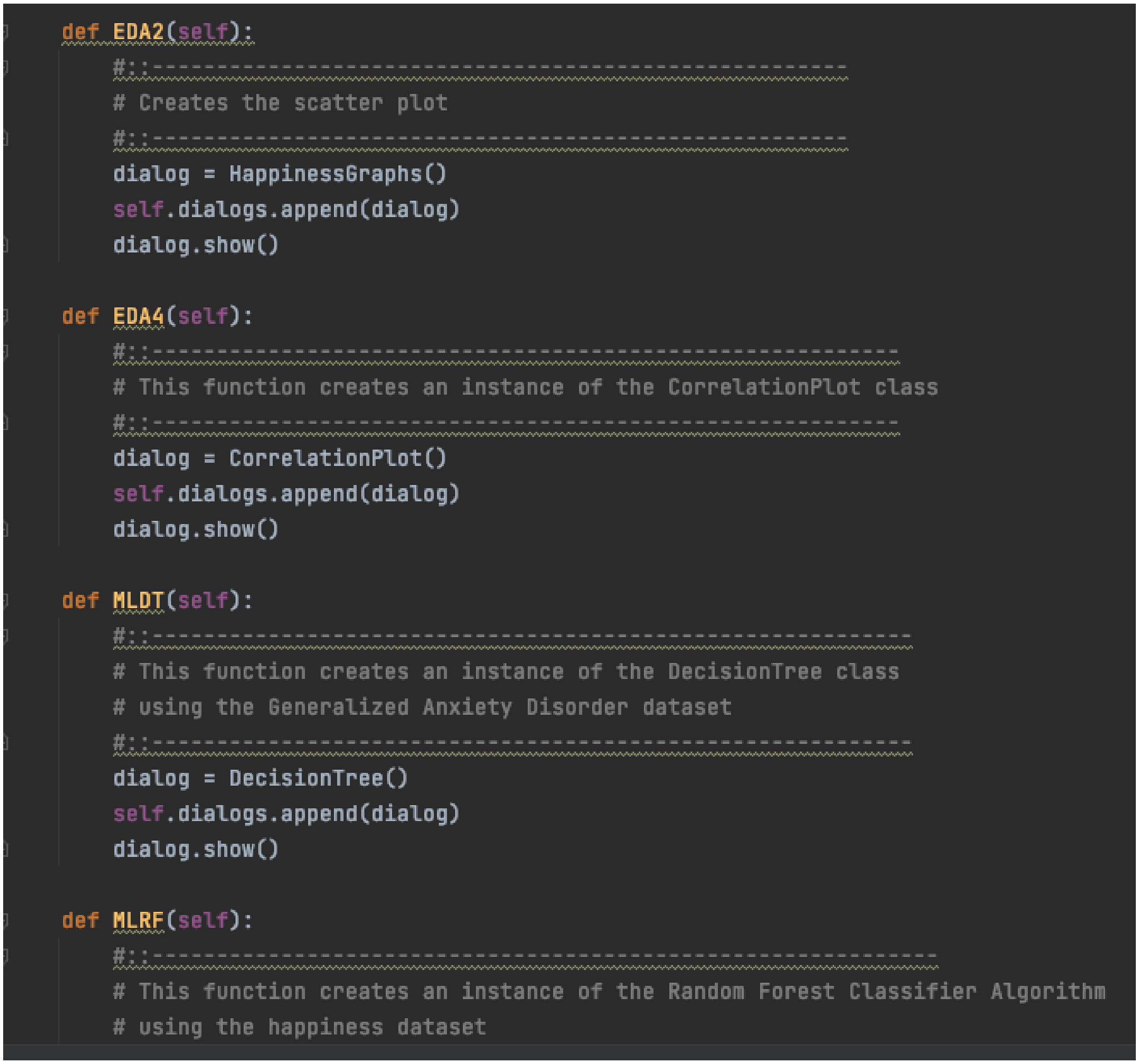
```
#::--------------------------------------
# Exit application
# Creates the actions for the fileMenu item
#::--------------------------------------

exitButton = QAction(QIcon('enter.png'), 'Exit', self)
exitButton.setShortcut('Ctrl+Q')
exitButton.setStatusTip('Exit application')
exitButton.triggered.connect(self.close)

fileMenu.addAction(exitButton)

#::----------------------------------------
# EDA analysis
# Creates the actions for the EDA Analysis item
# Initial Assesment : Histogram about the level of happiness in 2017
# Happiness Final : Presents the correlation between the index of happiness and a feature from the datasets.
# Correlation Plot : Correlation plot using all the dims in the datasets
#::----------------------------------------

EDA1Button = QAction(QIcon('analysis.png'),'Initial Assesment', self)
EDA1Button.setStatusTip('Presents the initial datasets')
EDA1Button.triggered.connect(self.EDA1)
EDAMenu.addAction(EDA1Button)

EDA2Button = QAction(QIcon('analysis.png'), 'Scatter plot', self)
EDA2Button.setStatusTip('Final Happiness Graph')
EDA2Button.triggered.connect(self.EDA2)
EDAMenu.addAction(EDA2Button)
```



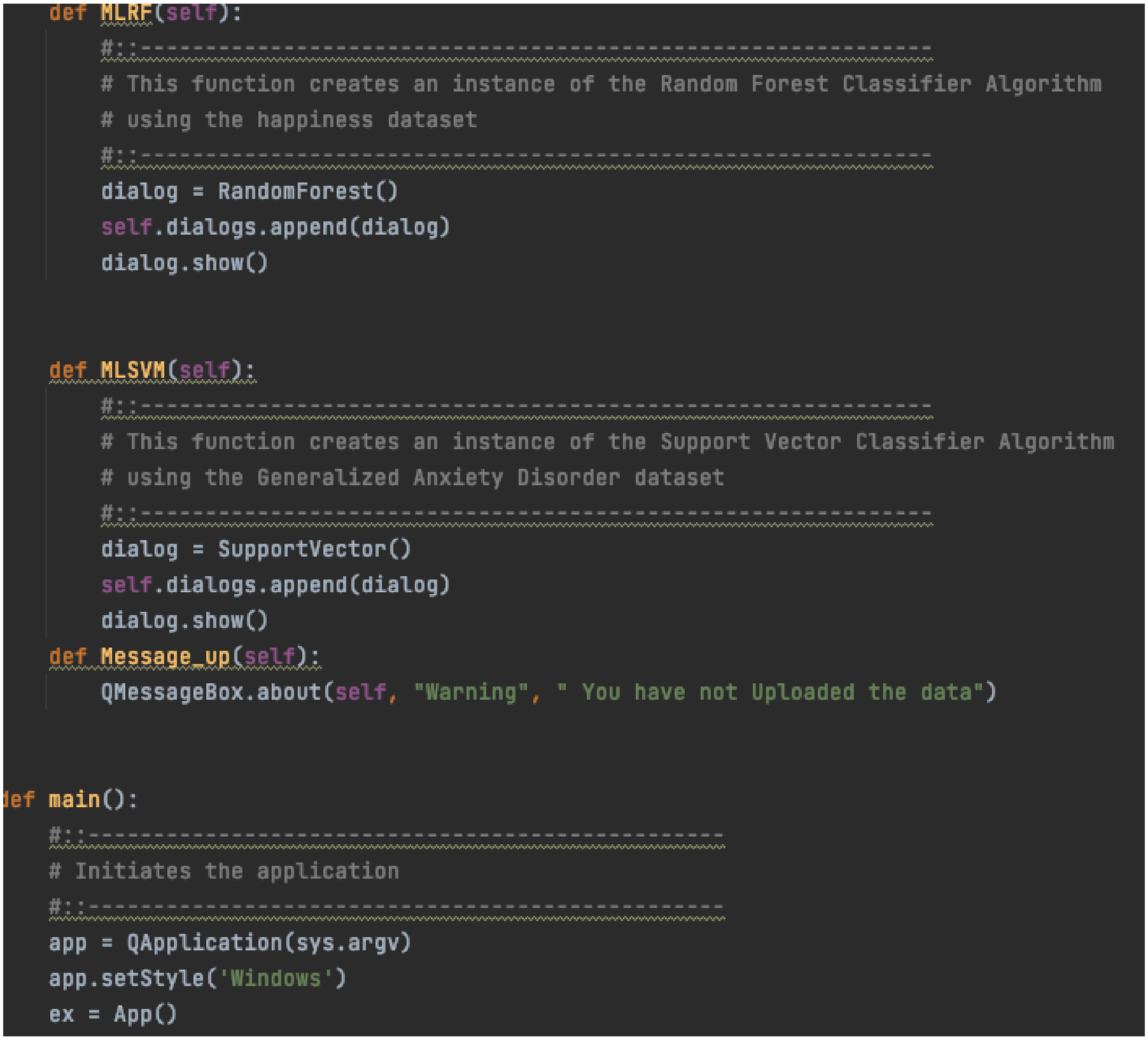
```
EDA4Button = QAction(QIcon('analysis.png'), 'Correlation Plot', self)
EDA4Button.setStatusTip('Features Correlation Plot')
EDA4Button.triggered.connect(self.EDA4)
EDAMenu.addAction(EDA4Button)

#::--------------------------------------------------
# ML Models for prediction
# There are two models
#       Decision Tree
#       Random Forest
#      Support Vector Machine
#::--------------------------------------------------
# Decision Tree Model
#::--------------------------------------------------
MLModel1Button =  QAction(QIcon(), 'Decision Tree Entropy', self)
MLModel1Button.setStatusTip('ML algorithm with Entropy ')
MLModel1Button.triggered.connect(self.MLDT)

#::------------------------------------------------------
# Random Forest Classifier
#::------------------------------------------------------
MLModel2Button = QAction(QIcon(), 'Random Forest Classifier', self)
MLModel2Button.setStatusTip('Random Forest Classifier ')
MLModel2Button.triggered.connect(self.MLRF)

#::------------------------------------------------------
# Support Vector Machine
#::------------------------------------------------------
MLModel3Button = QAction(QIcon(), 'Support Vector Machine', self)
```



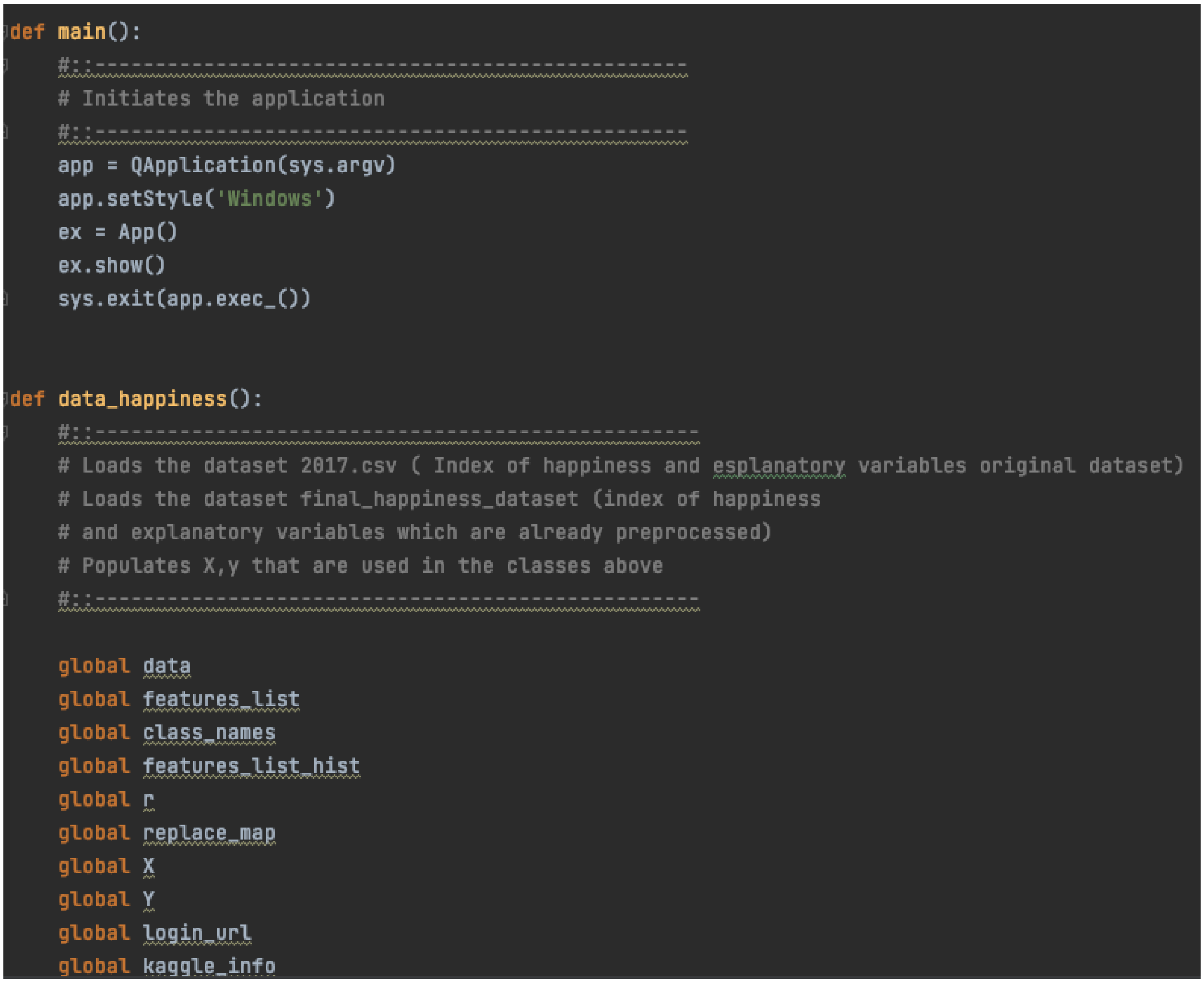
```
        #::------------------------------------------------------
        # Support Vector Machine
        #::------------------------------------------------------
        MLModel3Button = QAction(QIcon(), 'Support Vector Machine', self)
        MLModel3Button.setStatusTip('Support Vector Machine ')
        MLModel3Button.triggered.connect(self.MLSVM)

        MLModelMenu.addAction(MLModel1Button)
        MLModelMenu.addAction(MLModel2Button)
        MLModelMenu.addAction(MLModel3Button)

        self.dialogs = list()




    def EDA1(self):
        #::------------------------------------------------------
        # Creates the Histogram plot
        #::------------------------------------------------------

        dialog = Histogram_plots()
        self.dialogs.append(dialog)
        dialog.show()


    def EDA2(self):
        #::------------------------------------------------------
        # Creates the scatter plot
```



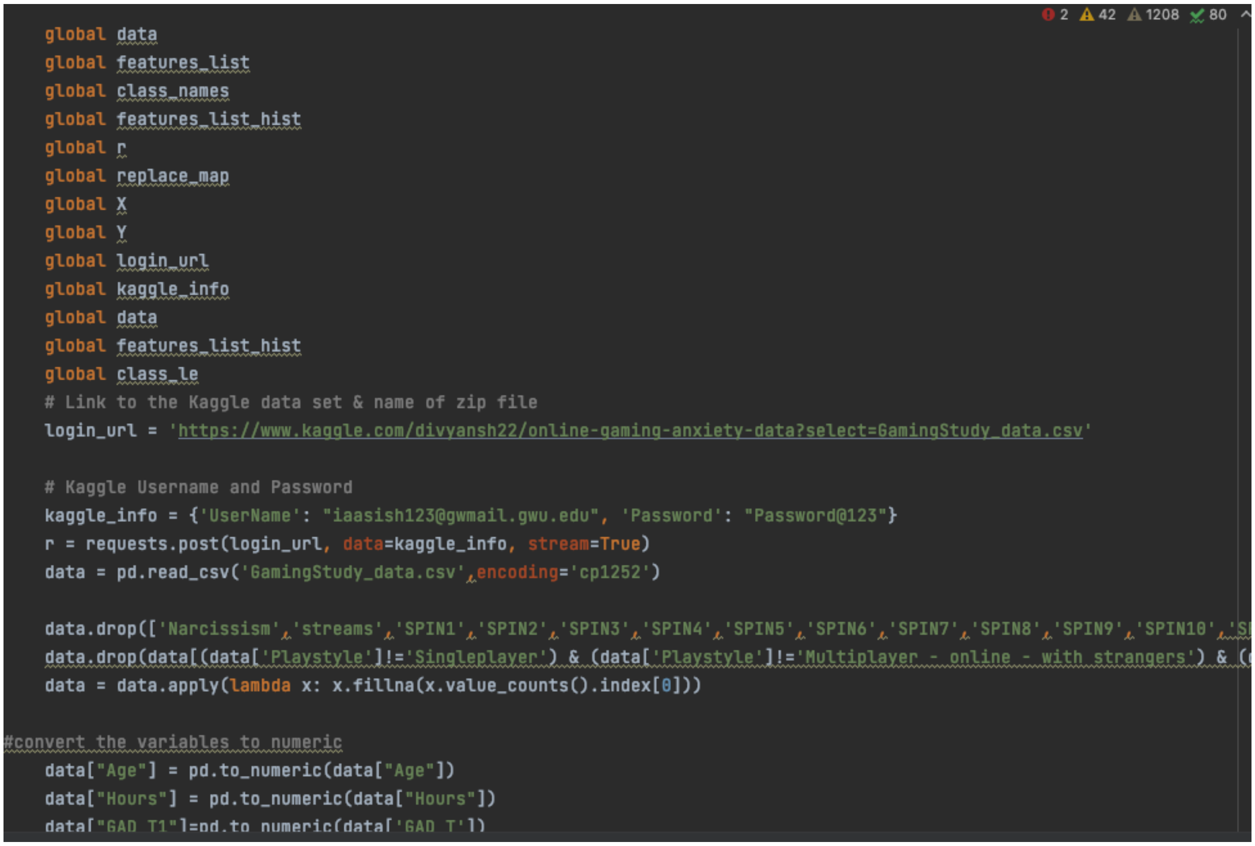
```
    def EDA2(self):
        #::------------------------------------------------------
        # Creates the scatter plot
        #::------------------------------------------------------
        dialog = HappinessGraphs()
        self.dialogs.append(dialog)
        dialog.show()

    def EDA4(self):
        #::----------------------------------------------------------
        # This function creates an instance of the CorrelationPlot class
        #::----------------------------------------------------------
        dialog = CorrelationPlot()
        self.dialogs.append(dialog)
        dialog.show()

    def MLDT(self):
        #::-----------------------------------------------------------
        # This function creates an instance of the DecisionTree class
        # using the Generalized Anxiety Disorder dataset
        #::-----------------------------------------------------------
        dialog = DecisionTree()
        self.dialogs.append(dialog)
        dialog.show()

    def MLRF(self):
        #::-------------------------------------------------------------
        # This function creates an instance of the Random Forest Classifier Algorithm
        # using the happiness dataset
```



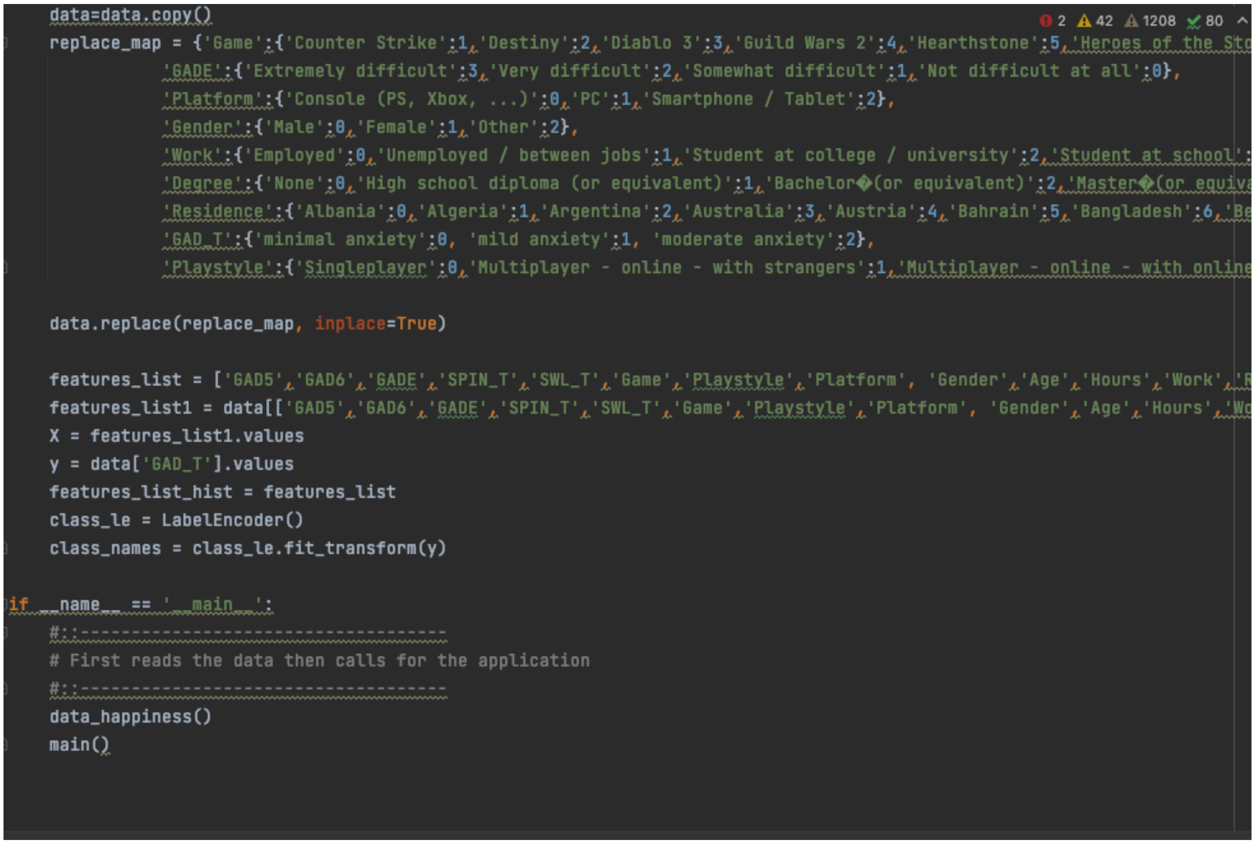
```
    def MLRF(self):
        #::-------------------------------------------------------------
        # This function creates an instance of the Random Forest Classifier Algorithm
        # using the happiness dataset
        #::-------------------------------------------------------------
        dialog = RandomForest()
        self.dialogs.append(dialog)
        dialog.show()


    def MLSVM(self):
        #::-------------------------------------------------------------
        # This function creates an instance of the Support Vector Classifier Algorithm
        # using the Generalized Anxiety Disorder dataset
        #::-------------------------------------------------------------
        dialog = SupportVector()
        self.dialogs.append(dialog)
        dialog.show()
    def Message_up(self):
        QMessageBox.about(self, "Warning", " You have not Uploaded the data")


def main():
    #::-------------------------------------------------
    # Initiates the application
    #::-------------------------------------------------
    app = QApplication(sys.argv)
    app.setStyle('Windows')
    ex = App()
```



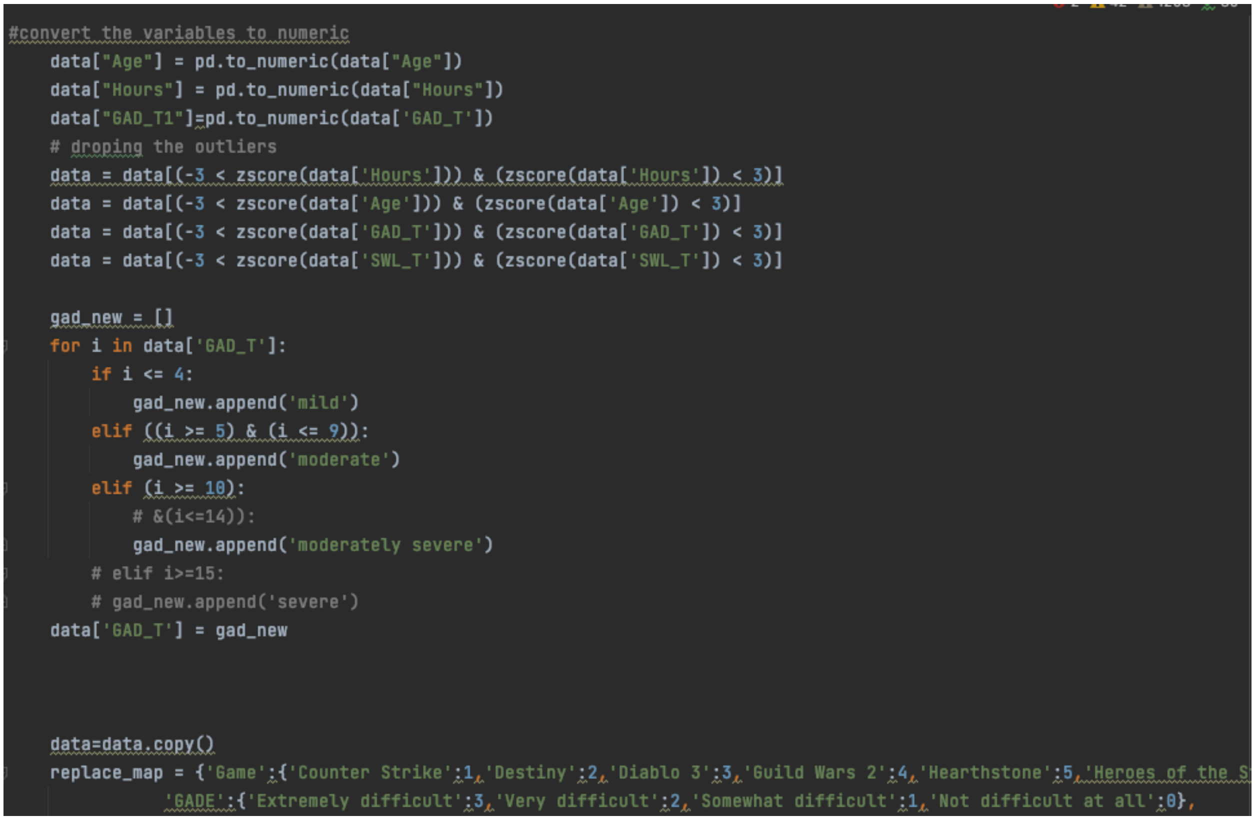
```
def main():
    #::-------------------------------------------------
    # Initiates the application
    #::-------------------------------------------------
    app = QApplication(sys.argv)
    app.setStyle('Windows')
    ex = App()
    ex.show()
    sys.exit(app.exec_())


def data_happiness():
    #::--------------------------------------------------
    # Loads the dataset 2017.csv ( Index of happiness and esplanatory variables original dataset)
    # Loads the dataset final_happiness_dataset (index of happiness
    # and explanatory variables which are already preprocessed)
    # Populates X,y that are used in the classes above
    #::--------------------------------------------------

    global data
    global features_list
    global class_names
    global features_list_hist
    global r
    global replace_map
    global X
    global Y
    global login_url
    global kaggle_info
```



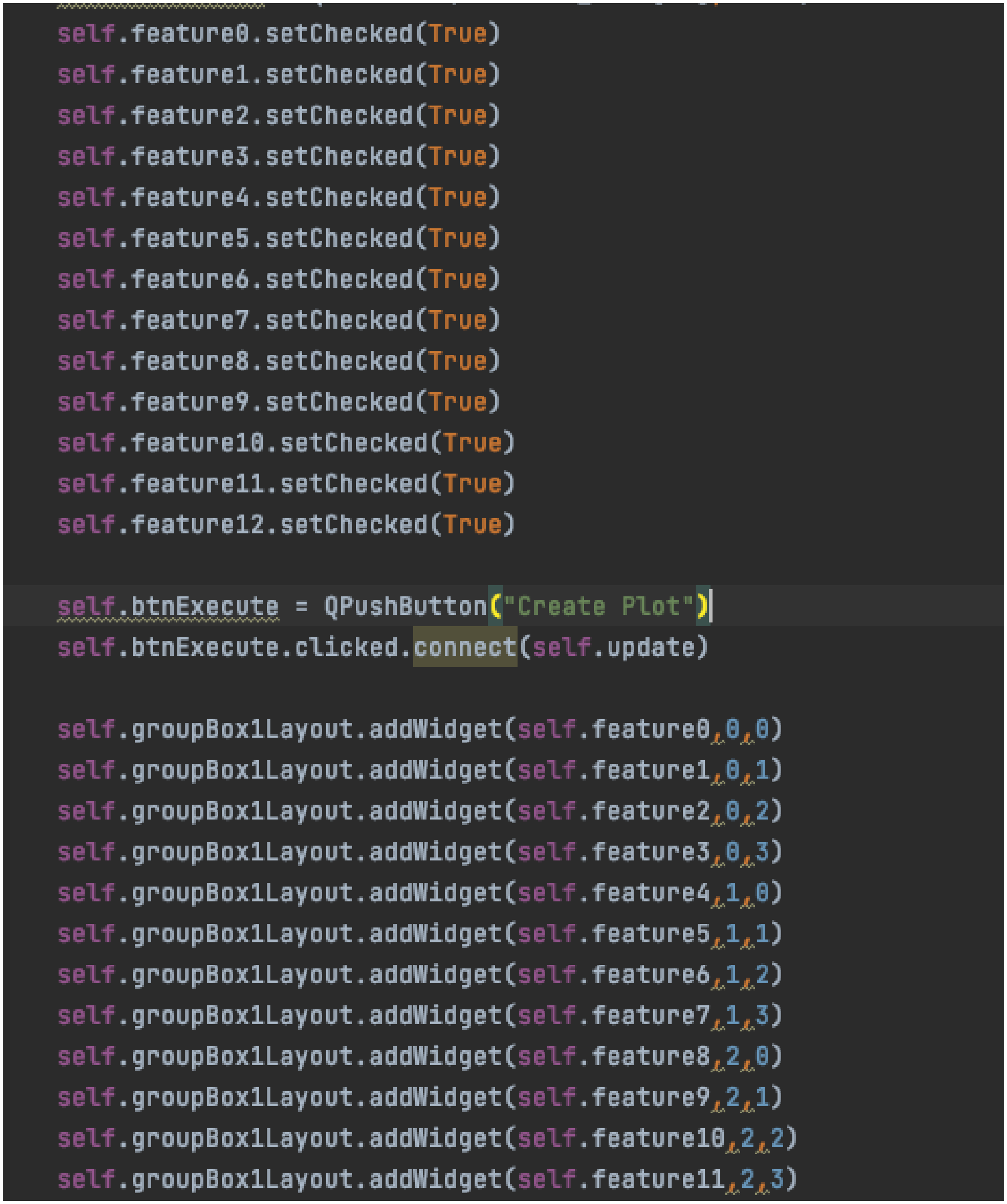
```
    global data
    global features_list
    global class_names
    global features_list_hist
    global r
    global replace_map
    global X
    global Y
    global login_url
    global kaggle_info
    global data
    global features_list_hist
    global class_le
    # Link to the Kaggle data set & name of zip file
    login_url = 'https://www.kaggle.com/divyansh22/online-gaming-anxiety-data?select=GamingStudy_data.csv'

    # Kaggle Username and Password
    kaggle_info = {'UserName': "iaasish123@gwmail.gwu.edu", 'Password': "Password@123"}
    r = requests.post(login_url, data=kaggle_info, stream=True)
    data = pd.read_csv('GamingStudy_data.csv',encoding='cp1252')

    data.drop(['Narcissism','streams','SPIN1','SPIN2','SPIN3','SPIN4','SPIN5','SPIN6','SPIN7','SPIN8','SPIN9','SPIN10','S
    data.drop(data[(data['Playstyle']!='Singleplayer') & (data['Playstyle']!='Multiplayer - online - with strangers') & (
    data = data.apply(lambda x: x.fillna(x.value_counts().index[0]))

#convert the variables to numeric
    data["Age"] = pd.to_numeric(data["Age"])
    data["Hours"] = pd.to_numeric(data["Hours"])
    data["GAD_T1"]=pd.to_numeric(data['GAD_T'])
```



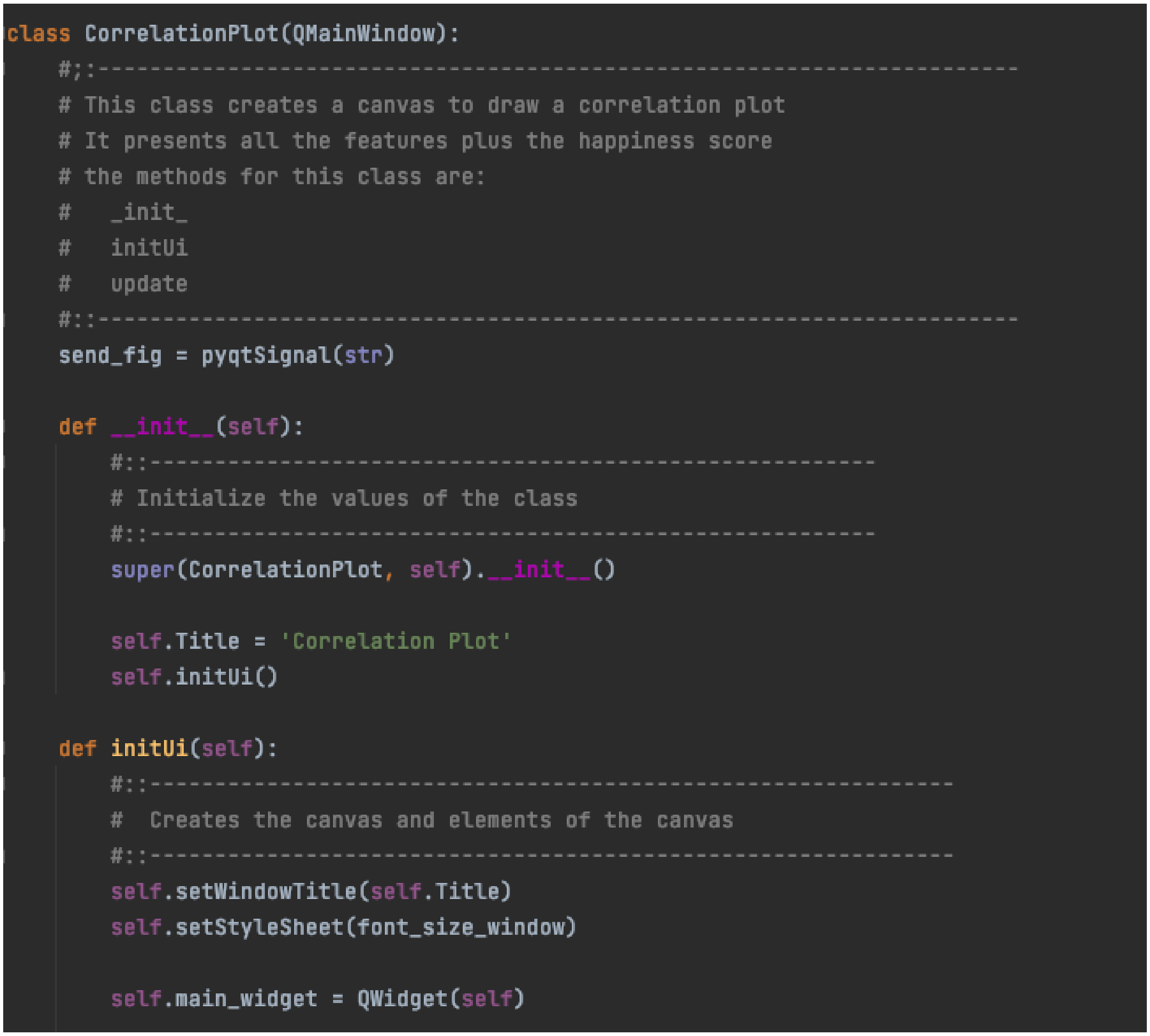
```python
    data=data.copy()
    replace_map = {'Game':{'Counter Strike':1,'Destiny':2,'Diablo 3':3,'Guild Wars 2':4,'Hearthstone':5,'Heroes of the Sto
                'GADE':{'Extremely difficult':3,'Very difficult':2,'Somewhat difficult':1,'Not difficult at all':0},
                'Platform':{'Console (PS, Xbox, ...)':0,'PC':1,'Smartphone / Tablet':2},
                'Gender':{'Male':0,'Female':1,'Other':2},
                'Work':{'Employed':0,'Unemployed / between jobs':1,'Student at college / university':2,'Student at school':
                'Degree':{'None':0,'High school diploma (or equivalent)':1,'Bachelor�(or equivalent)':2,'Master�(or equiva
                'Residence':{'Albania':0,'Algeria':1,'Argentina':2,'Australia':3,'Austria':4,'Bahrain':5,'Bangladesh':6,'Be
                'GAD_T':{'minimal anxiety':0, 'mild anxiety':1, 'moderate anxiety':2},
                'Playstyle':{'Singleplayer':0,'Multiplayer - online - with strangers':1,'Multiplayer - online - with online

    data.replace(replace_map, inplace=True)

    features_list = ['GAD5','GAD6','GADE','SPIN_T','SWL_T','Game','Playstyle','Platform', 'Gender','Age','Hours','Work','R
    features_list1 = data[['GAD5','GAD6','GADE','SPIN_T','SWL_T','Game','Playstyle','Platform', 'Gender','Age','Hours','Wo
    X = features_list1.values
    y = data['GAD_T'].values
    features_list_hist = features_list
    class_le = LabelEncoder()
    class_names = class_le.fit_transform(y)

if __name__ == '__main__':
    #::-----------------------------------
    # First reads the data then calls for the application
    #::-----------------------------------
    data_happiness()
    main()
```



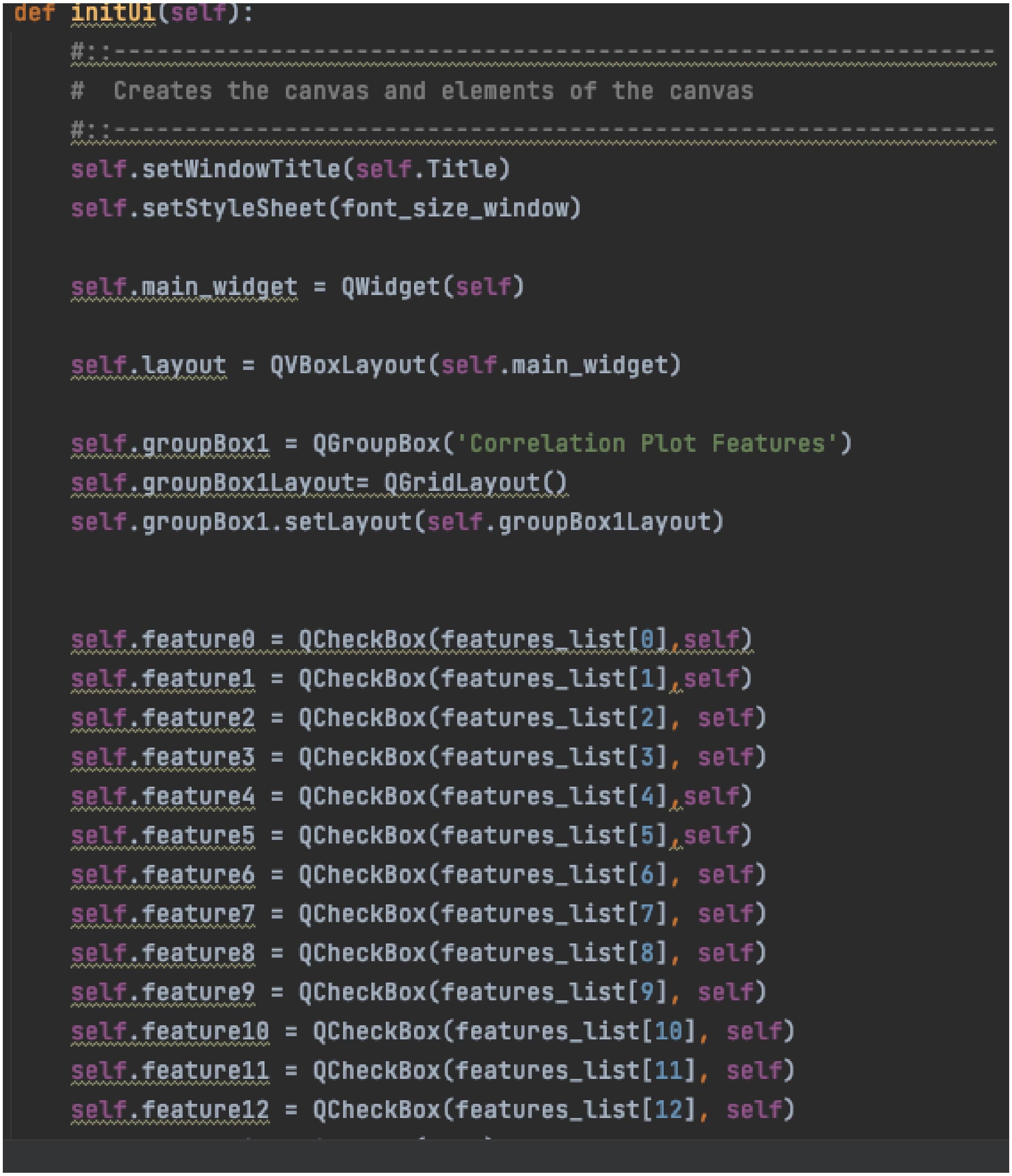
```
#convert the variables to numeric
    data["Age"] = pd.to_numeric(data["Age"])
    data["Hours"] = pd.to_numeric(data["Hours"])
    data["GAD_T1"]=pd.to_numeric(data['GAD_T'])
    # droping the outliers
    data = data[(-3 < zscore(data['Hours'])) & (zscore(data['Hours']) < 3)]
    data = data[(-3 < zscore(data['Age'])) & (zscore(data['Age']) < 3)]
    data = data[(-3 < zscore(data['GAD_T'])) & (zscore(data['GAD_T']) < 3)]
    data = data[(-3 < zscore(data['SWL_T'])) & (zscore(data['SWL_T']) < 3)]

    gad_new = []
    for i in data['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')
    data['GAD_T'] = gad_new



    data=data.copy()
    replace_map = {'Game':{'Counter Strike':1,'Destiny':2,'Diablo 3':3,'Guild Wars 2':4,'Hearthstone':5,'Heroes of the S
               'GADE':{'Extremely difficult':3,'Very difficult':2,'Somewhat difficult':1,'Not difficult at all':0},
```



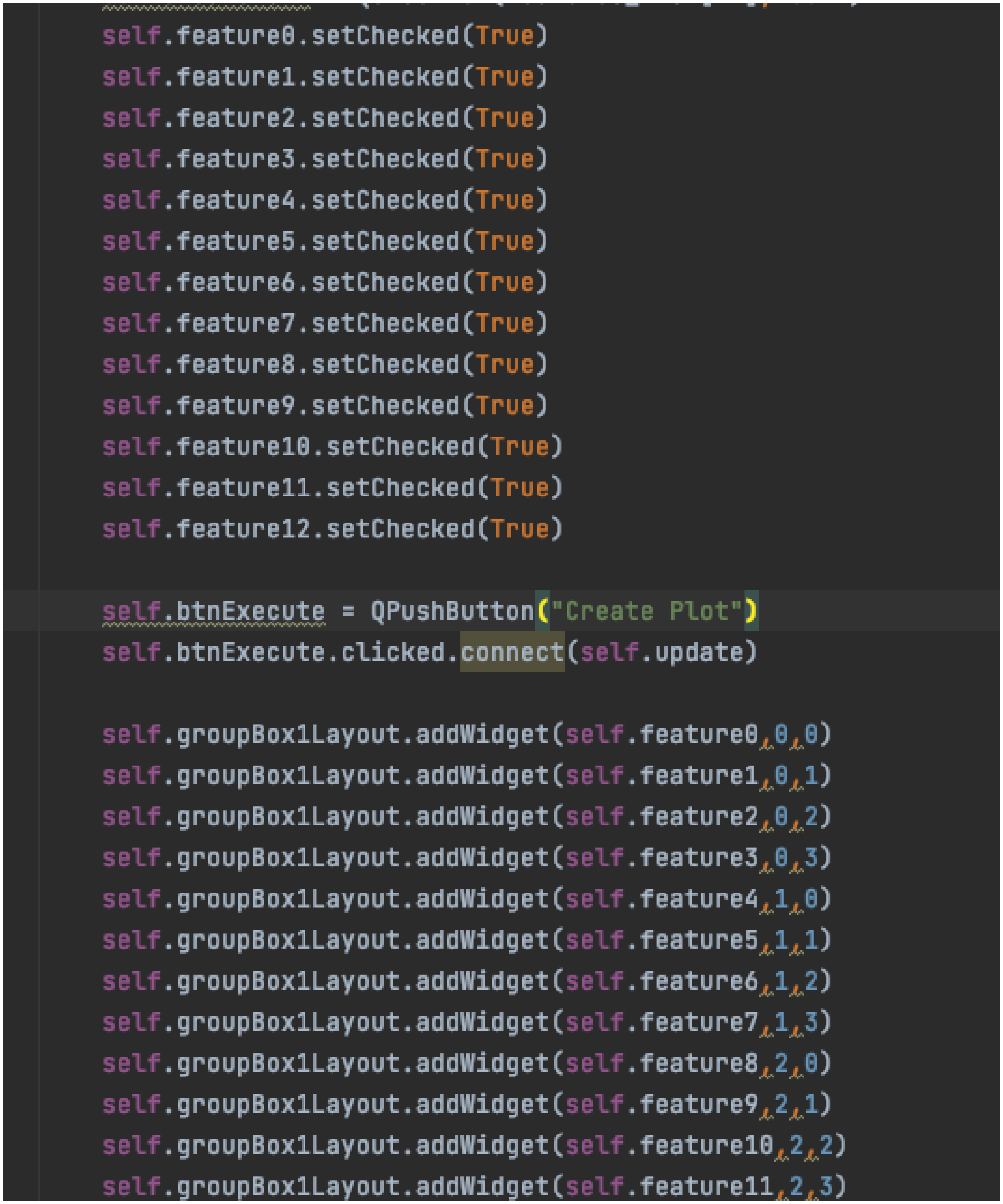
```
self.feature0.setChecked(True)
self.feature1.setChecked(True)
self.feature2.setChecked(True)
self.feature3.setChecked(True)
self.feature4.setChecked(True)
self.feature5.setChecked(True)
self.feature6.setChecked(True)
self.feature7.setChecked(True)
self.feature8.setChecked(True)
self.feature9.setChecked(True)
self.feature10.setChecked(True)
self.feature11.setChecked(True)
self.feature12.setChecked(True)

self.btnExecute = QPushButton("Create Plot")
self.btnExecute.clicked.connect(self.update)

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



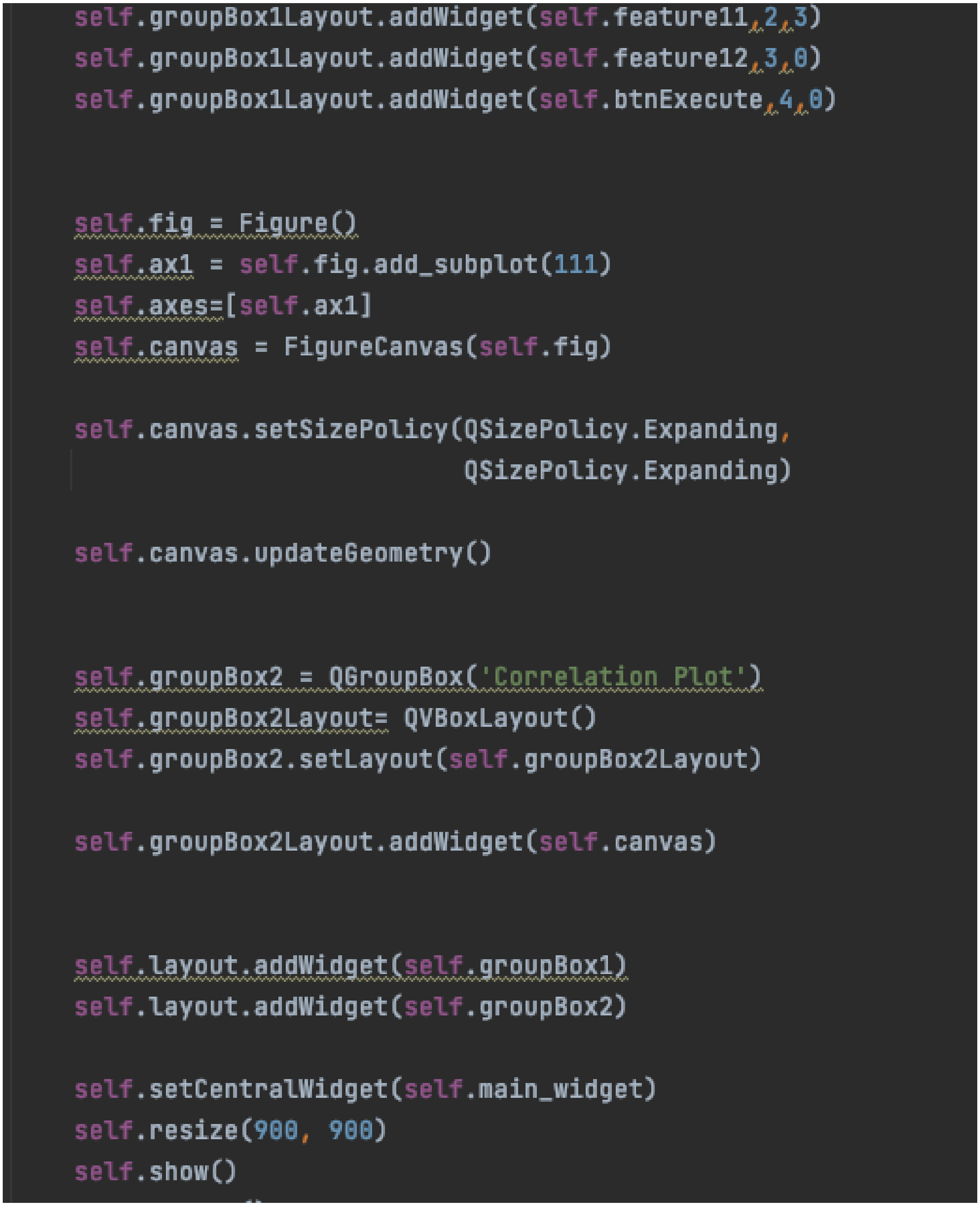
```
class CorrelationPlot(QMainWindow):
    #;:-----------------------------------------------------------------------
    # This class creates a canvas to draw a correlation plot
    # It presents all the features plus the happiness score
    # the methods for this class are:
    #   _init_
    #   initUi
    #   update
    #::-----------------------------------------------------------------------
    send_fig = pyqtSignal(str)

    def __init__(self):
        #::--------------------------------------------------------
        # Initialize the values of the class
        #::--------------------------------------------------------
        super(CorrelationPlot, self).__init__()

        self.Title = 'Correlation Plot'
        self.initUi()

    def initUi(self):
        #::--------------------------------------------------------------
        #  Creates the canvas and elements of the canvas
        #::--------------------------------------------------------------
        self.setWindowTitle(self.Title)
        self.setStyleSheet(font_size_window)

        self.main_widget = QWidget(self)
```



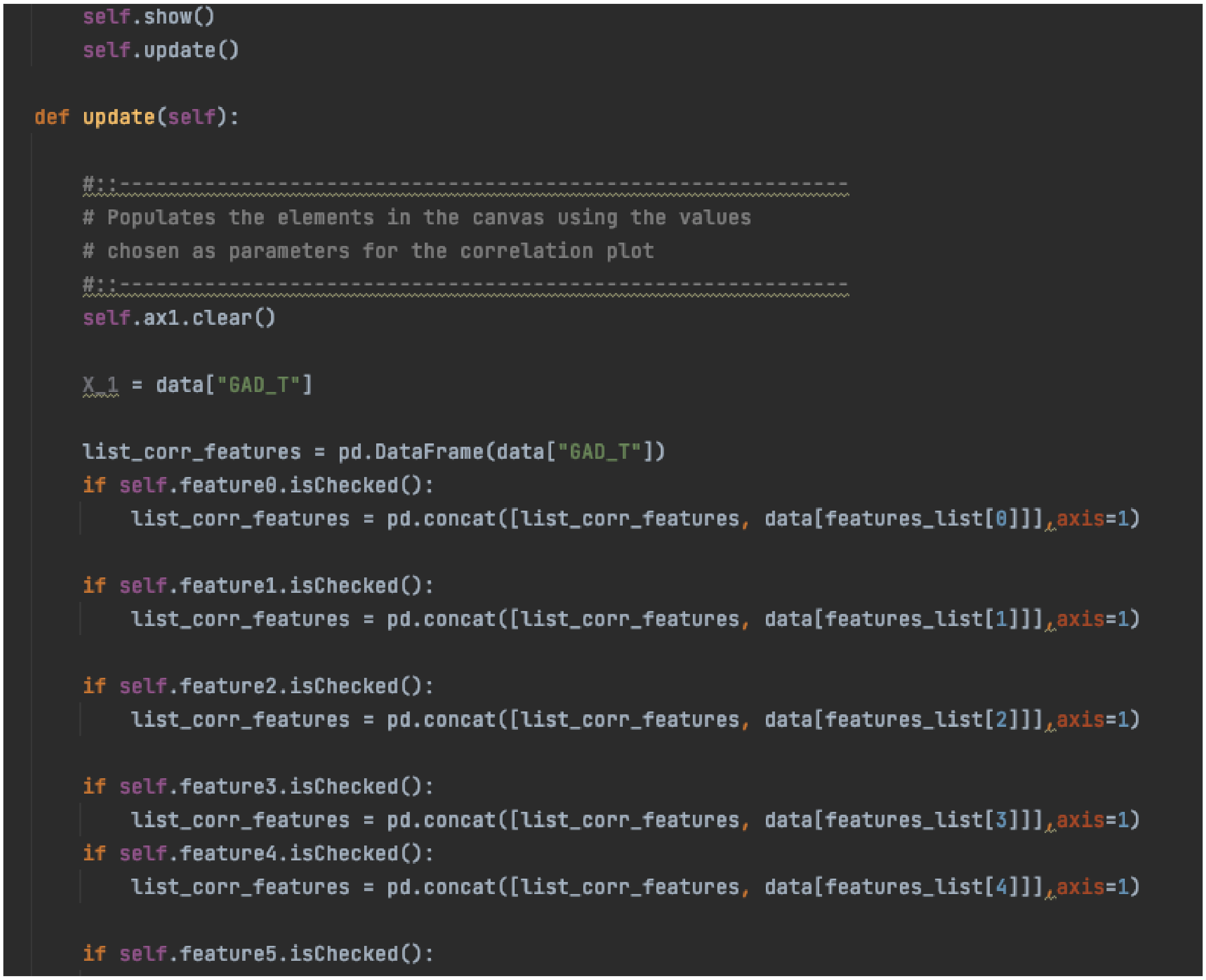
```
def initUi(self):
    #::--------------------------------------------------------------
    #  Creates the canvas and elements of the canvas
    #::--------------------------------------------------------------
    self.setWindowTitle(self.Title)
    self.setStyleSheet(font_size_window)

    self.main_widget = QWidget(self)

    self.layout = QVBoxLayout(self.main_widget)

    self.groupBox1 = QGroupBox('Correlation Plot 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.feature6 = QCheckBox(features_list[6], self)
    self.feature7 = QCheckBox(features_list[7], self)
    self.feature8 = QCheckBox(features_list[8], self)
    self.feature9 = QCheckBox(features_list[9], self)
    self.feature10 = QCheckBox(features_list[10], self)
    self.feature11 = QCheckBox(features_list[11], self)
    self.feature12 = QCheckBox(features_list[12], self)
```



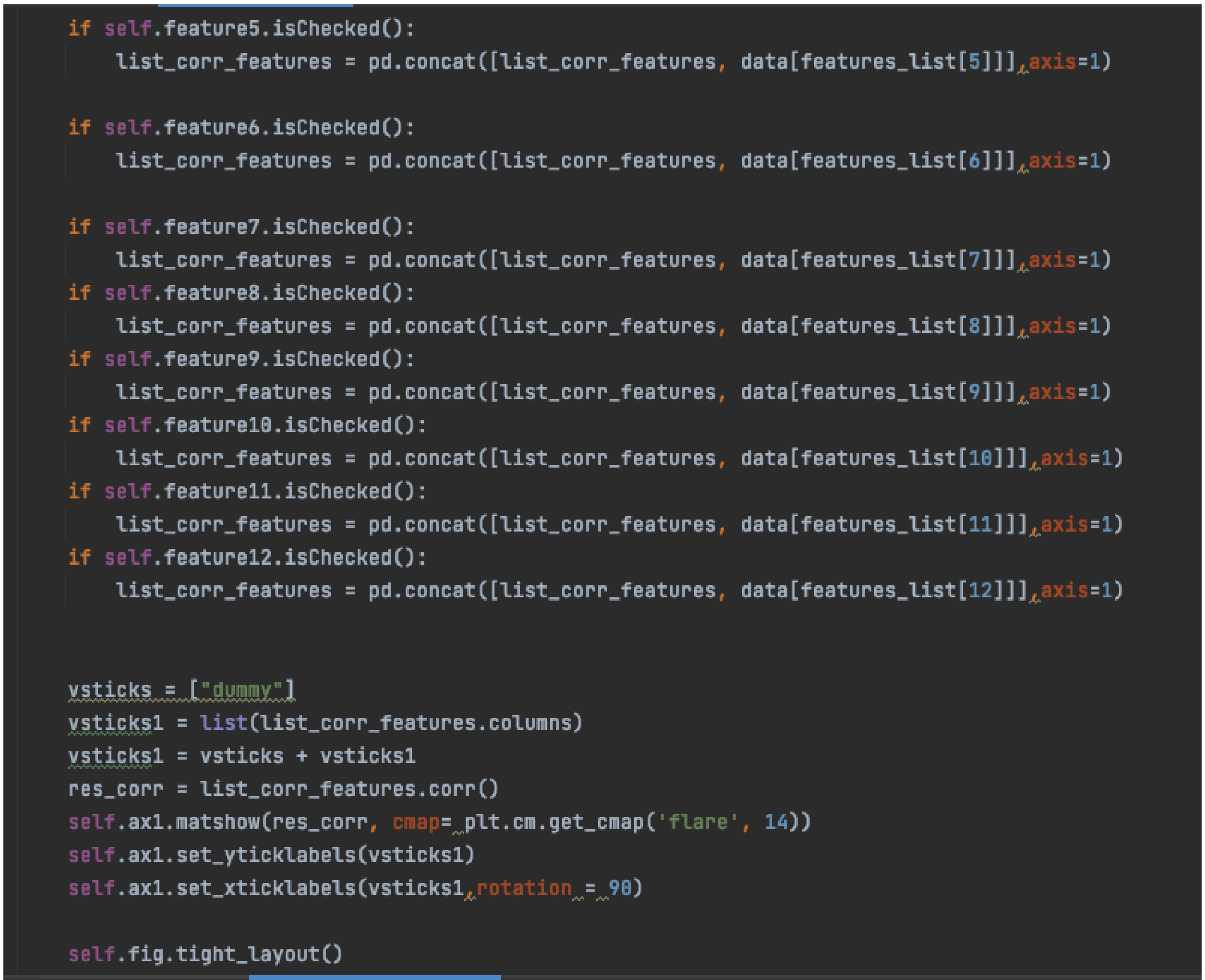
```
        self.feature0.setChecked(True)
        self.feature1.setChecked(True)
        self.feature2.setChecked(True)
        self.feature3.setChecked(True)
        self.feature4.setChecked(True)
        self.feature5.setChecked(True)
        self.feature6.setChecked(True)
        self.feature7.setChecked(True)
        self.feature8.setChecked(True)
        self.feature9.setChecked(True)
        self.feature10.setChecked(True)
        self.feature11.setChecked(True)
        self.feature12.setChecked(True)

        self.btnExecute = QPushButton("Create Plot")
        self.btnExecute.clicked.connect(self.update)

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



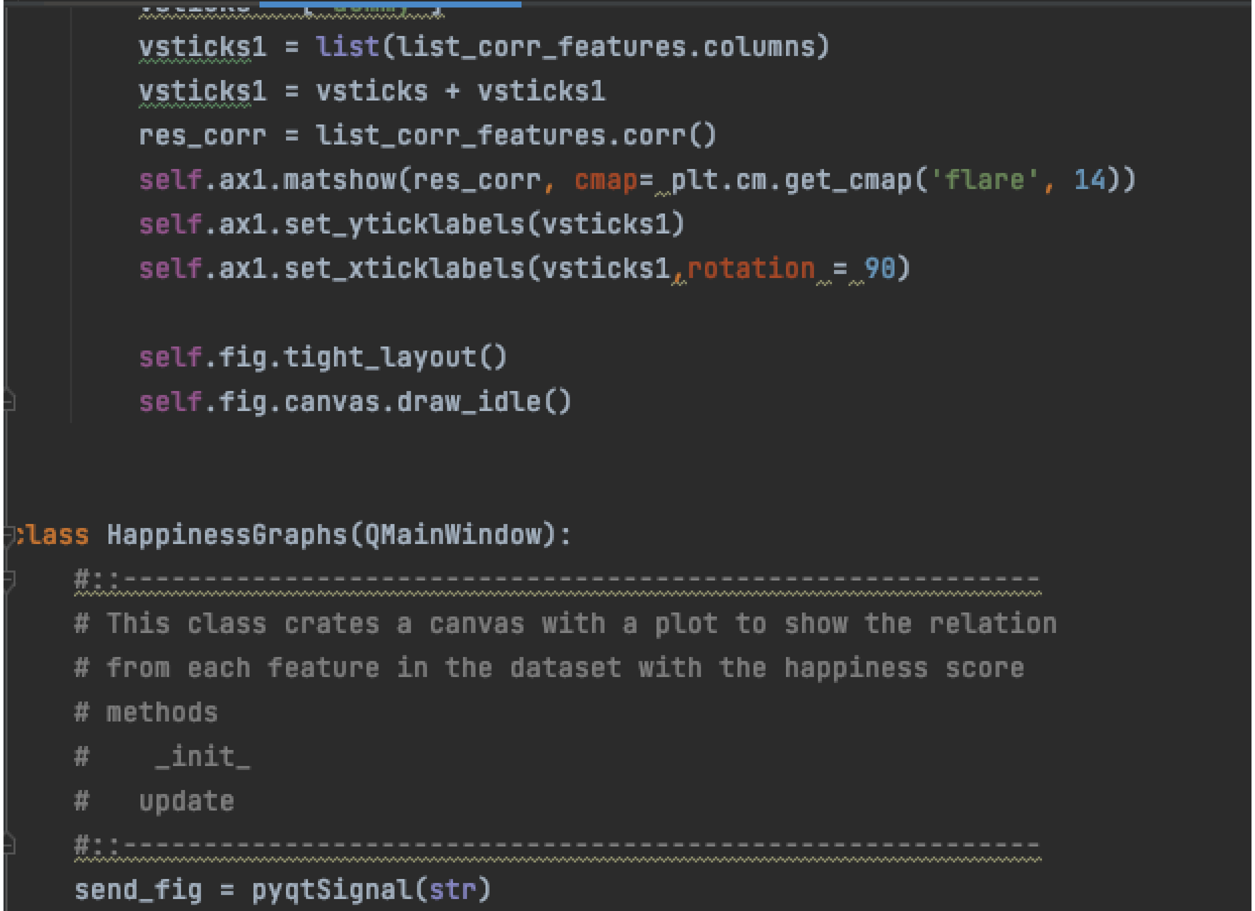
```
self.groupBox1Layout.addWidget(self.feature11,2,3)
self.groupBox1Layout.addWidget(self.feature12,3,0)
self.groupBox1Layout.addWidget(self.btnExecute,4,0)


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.groupBox2 = QGroupBox('Correlation Plot')
self.groupBox2Layout= QVBoxLayout()
self.groupBox2.setLayout(self.groupBox2Layout)

self.groupBox2Layout.addWidget(self.canvas)


self.layout.addWidget(self.groupBox1)
self.layout.addWidget(self.groupBox2)

self.setCentralWidget(self.main_widget)
self.resize(900, 900)
self.show()
```



```
    self.show()
    self.update()

def update(self):

    #::--------------------------------------------------------------
    # Populates the elements in the canvas using the values
    # chosen as parameters for the correlation plot
    #::--------------------------------------------------------------
    self.ax1.clear()

    X_1 = data["GAD_T"]

    list_corr_features = pd.DataFrame(data["GAD_T"])
    if self.feature0.isChecked():
        list_corr_features = pd.concat([list_corr_features, data[features_list[0]]],axis=1)

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

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

    if self.feature3.isChecked():
        list_corr_features = pd.concat([list_corr_features, data[features_list[3]]],axis=1)
    if self.feature4.isChecked():
        list_corr_features = pd.concat([list_corr_features, data[features_list[4]]],axis=1)

    if self.feature5.isChecked():
```

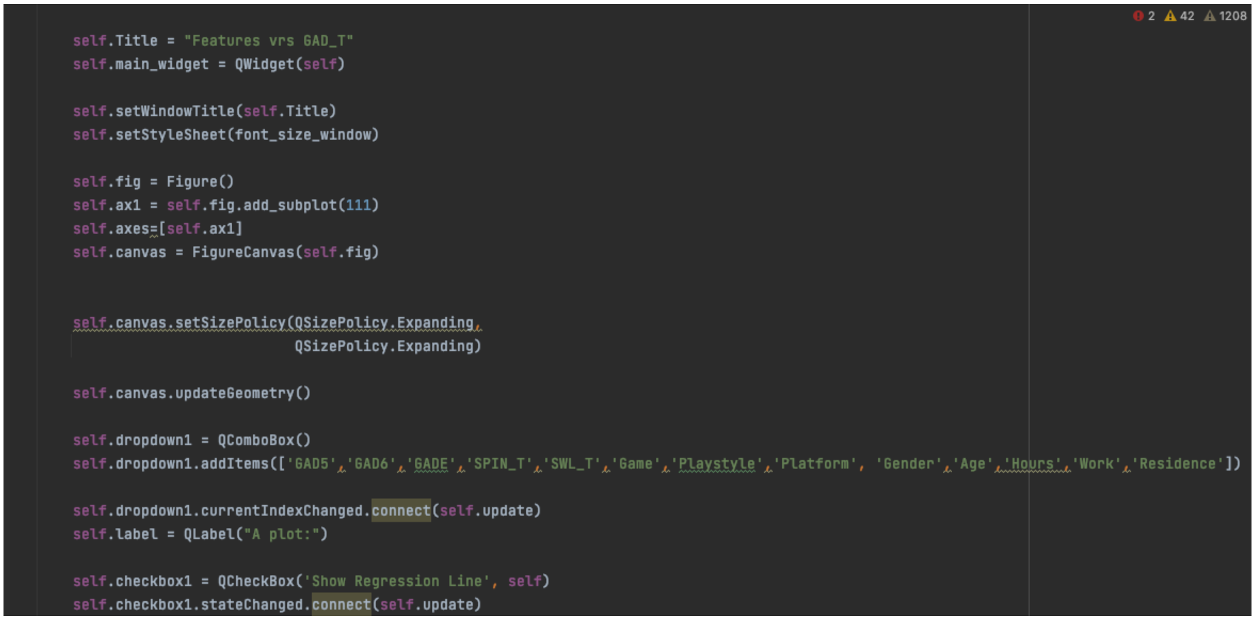


```
        if self.feature5.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[5]]],axis=1)

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

        if self.feature7.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[7]]],axis=1)
        if self.feature8.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[8]]],axis=1)
        if self.feature9.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[9]]],axis=1)
        if self.feature10.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[10]]],axis=1)
        if self.feature11.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[11]]],axis=1)
        if self.feature12.isChecked():
            list_corr_features = pd.concat([list_corr_features, data[features_list[12]]],axis=1)


        vsticks = ["dummy"]
        vsticks1 = list(list_corr_features.columns)
        vsticks1 = vsticks + vsticks1
        res_corr = list_corr_features.corr()
        self.ax1.matshow(res_corr, cmap= plt.cm.get_cmap('flare', 14))
        self.ax1.set_yticklabels(vsticks1)
        self.ax1.set_xticklabels(vsticks1,rotation = 90)

        self.fig.tight_layout()
```

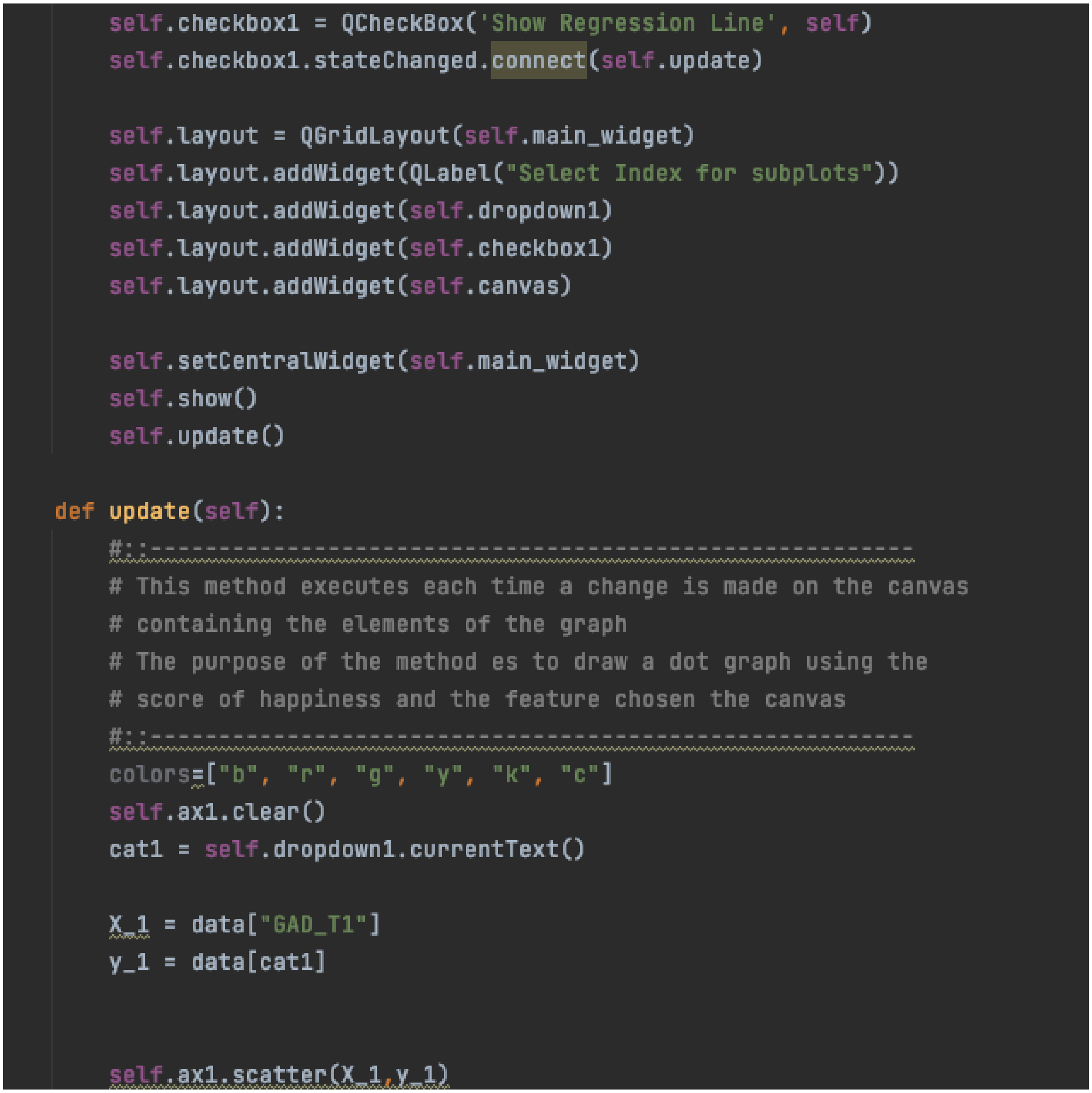


```
        vsticks1 = list(list_corr_features.columns)
        vsticks1 = vsticks + vsticks1
        res_corr = list_corr_features.corr()
        self.ax1.matshow(res_corr, cmap= plt.cm.get_cmap('flare', 14))
        self.ax1.set_yticklabels(vsticks1)
        self.ax1.set_xticklabels(vsticks1,rotation = 90)

        self.fig.tight_layout()
```

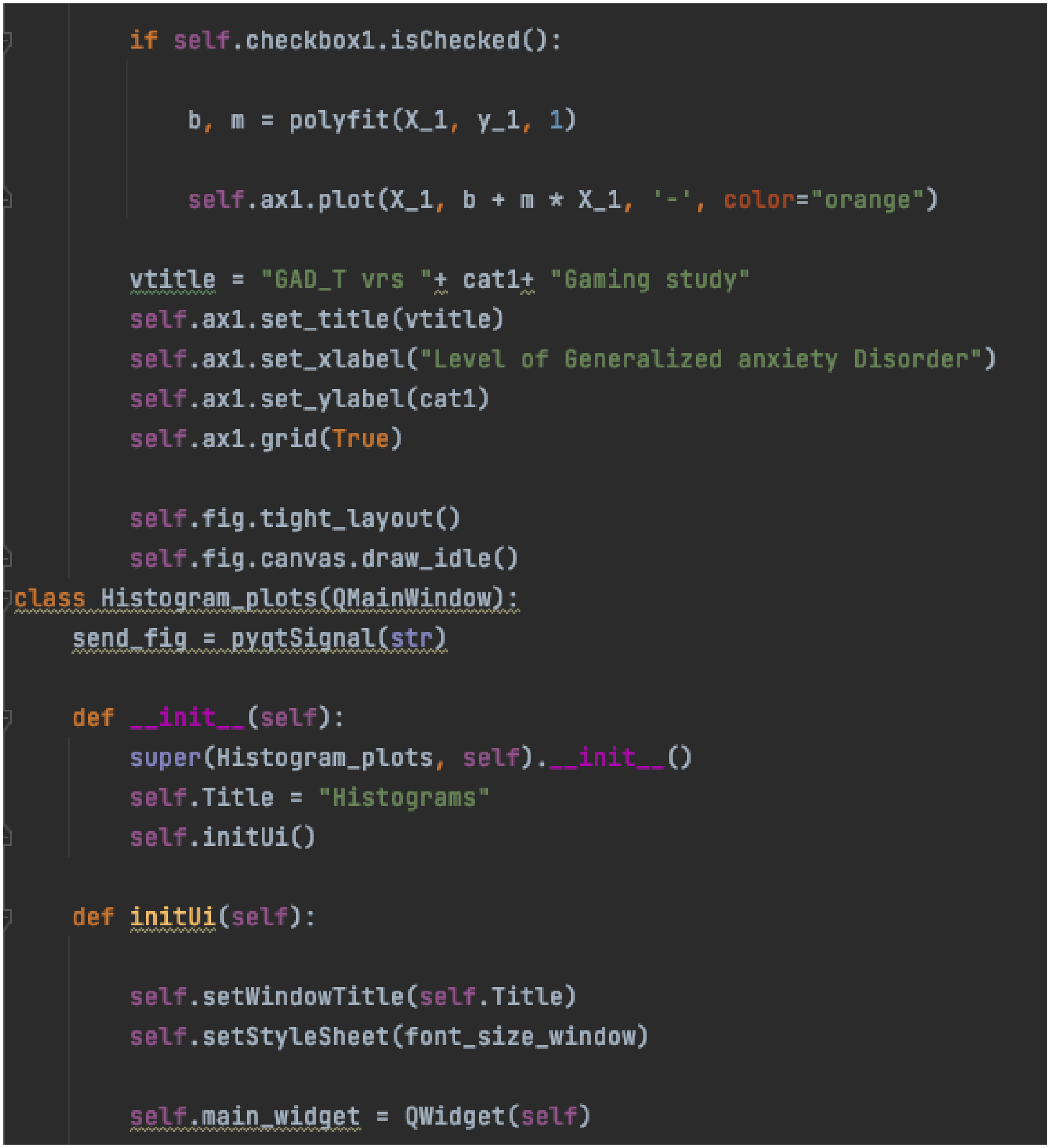
```
        self.fig.canvas.draw_idle()


class HappinessGraphs(QMainWindow):
    #::--------------------------------------------------------
    # This class crates a canvas with a plot to show the relation
    # from each feature in the dataset with the happiness score
    # methods
    #    _init_
    #   update
    #::--------------------------------------------------------
    send_fig = pyqtSignal(str)
```



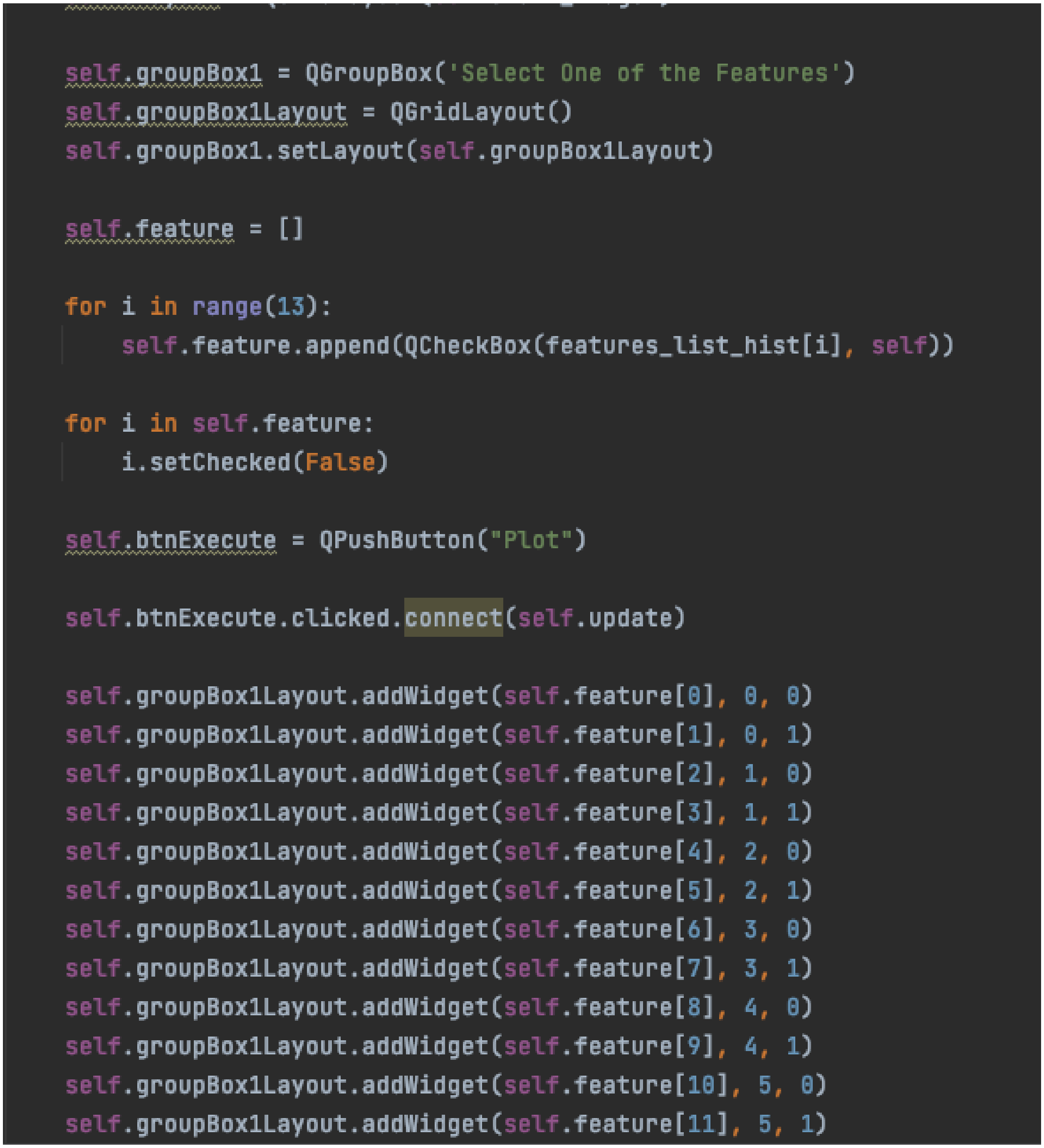
```
self.Title = "Features vrs GAD_T"
self.main_widget = QWidget(self)

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

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.dropdown1 = QComboBox()
self.dropdown1.addItems(['GAD5','GAD6','GADE','SPIN_T','SWL_T','Game','Playstyle','Platform', 'Gender','Age','Hours','Work','Residence'])

self.dropdown1.currentIndexChanged.connect(self.update)
self.label = QLabel("A plot:")

self.checkbox1 = QCheckBox('Show Regression Line', self)
self.checkbox1.stateChanged.connect(self.update)
```



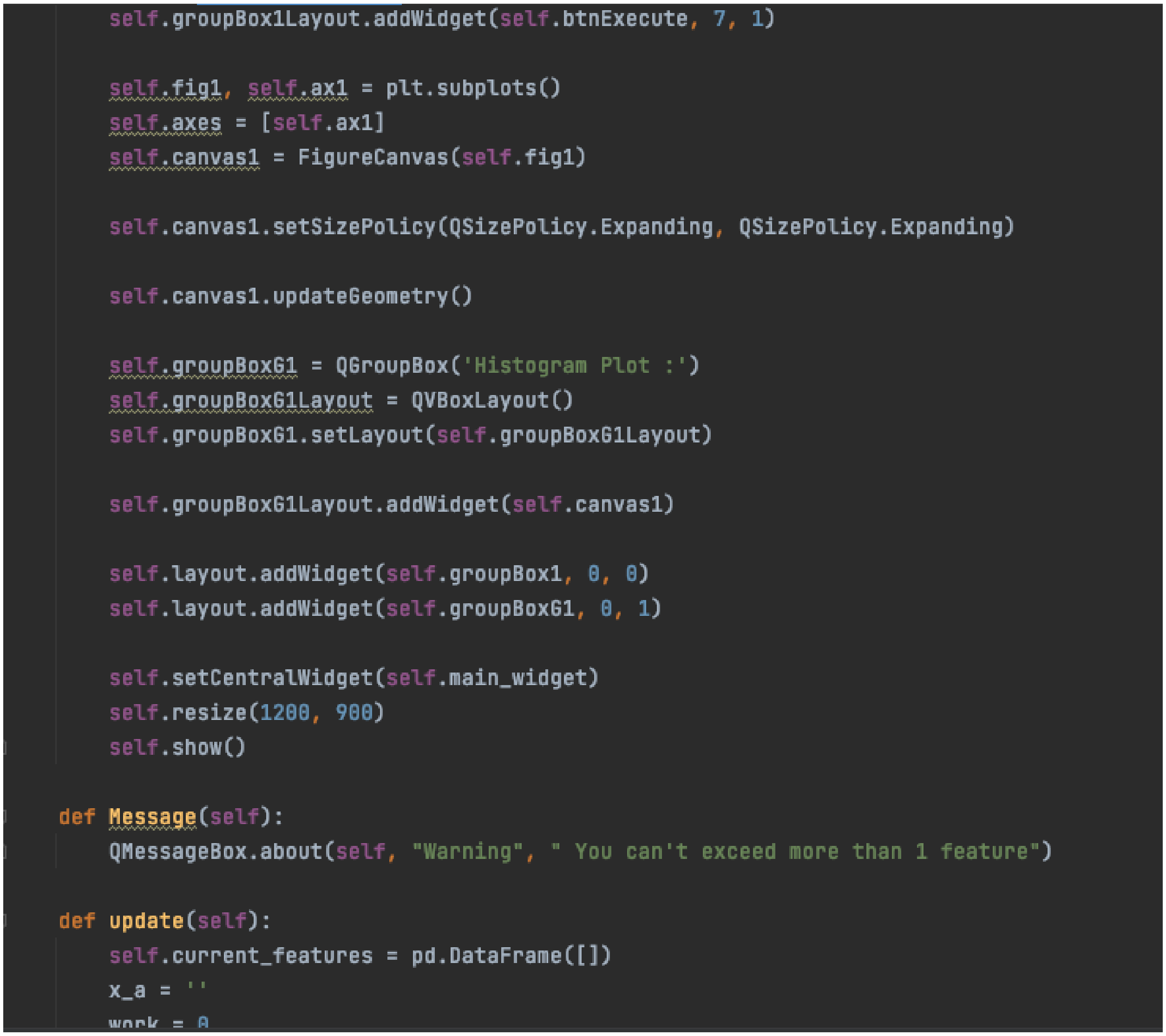
```
        self.checkbox1 = QCheckBox('Show Regression Line', self)
        self.checkbox1.stateChanged.connect(self.update)

        self.layout = QGridLayout(self.main_widget)
        self.layout.addWidget(QLabel("Select Index for subplots"))
        self.layout.addWidget(self.dropdown1)
        self.layout.addWidget(self.checkbox1)
        self.layout.addWidget(self.canvas)

        self.setCentralWidget(self.main_widget)
        self.show()
        self.update()

    def update(self):
        #::------------------------------------------------------------
        # This method executes each time a change is made on the canvas
        # containing the elements of the graph
        # The purpose of the method es to draw a dot graph using the
        # score of happiness and the feature chosen the canvas
        #::------------------------------------------------------------
        colors=["b", "r", "g", "y", "k", "c"]
        self.ax1.clear()
        cat1 = self.dropdown1.currentText()

        X_1 = data["GAD_T1"]
        y_1 = data[cat1]


        self.ax1.scatter(X_1,y_1)
```



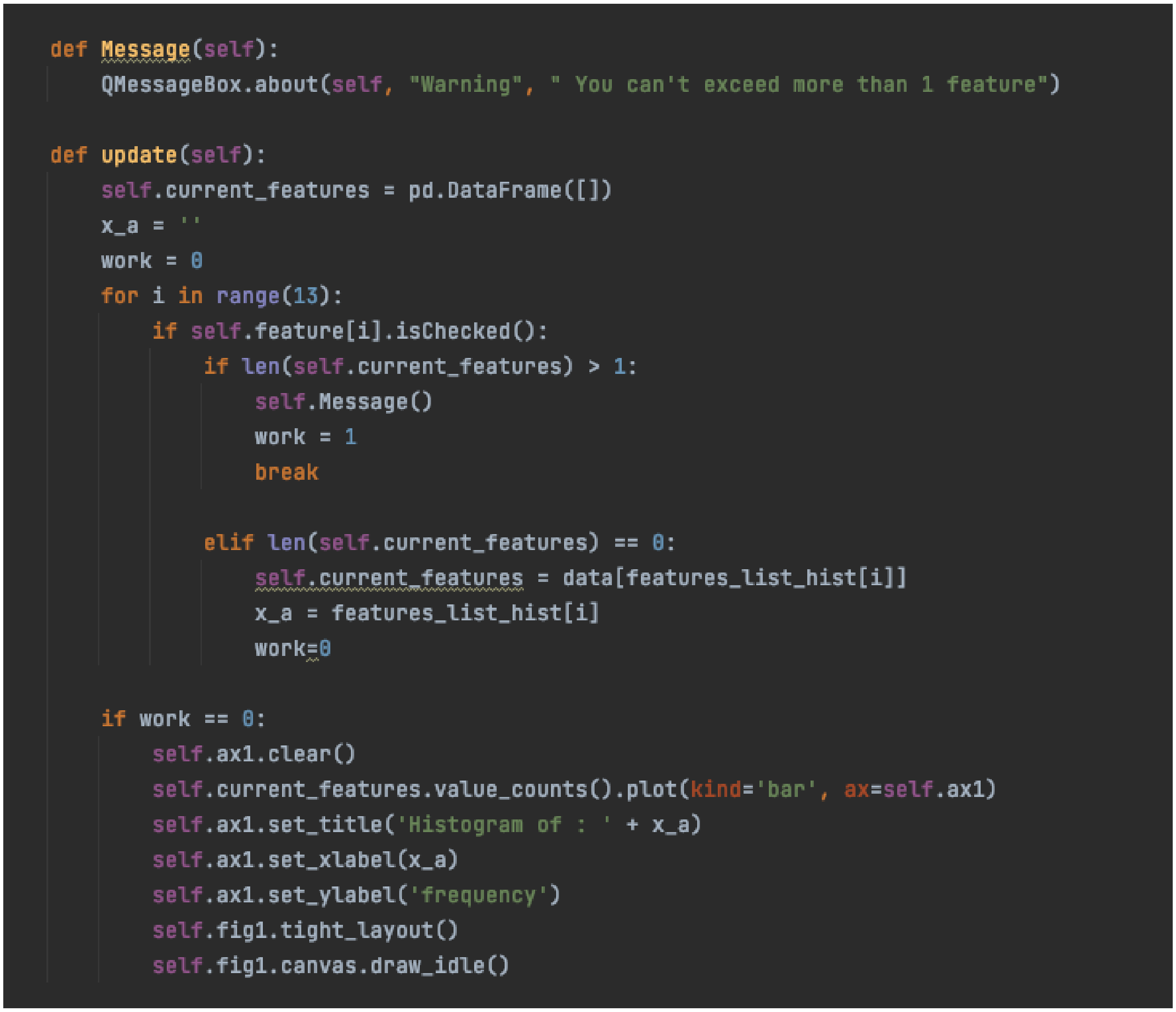
```
        if self.checkbox1.isChecked():

            b, m = polyfit(X_1, y_1, 1)

            self.ax1.plot(X_1, b + m * X_1, '-', color="orange")

        vtitle = "GAD_T vrs "+ cat1+ "Gaming study"
        self.ax1.set_title(vtitle)
        self.ax1.set_xlabel("Level of Generalized anxiety Disorder")
        self.ax1.set_ylabel(cat1)
        self.ax1.grid(True)

        self.fig.tight_layout()
        self.fig.canvas.draw_idle()
class Histogram_plots(QMainWindow):
    send_fig = pyqtSignal(str)

    def __init__(self):
        super(Histogram_plots, self).__init__()
        self.Title = "Histograms"
        self.initUi()

    def initUi(self):

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

        self.main_widget = QWidget(self)
```



```
self.groupBox1 = QGroupBox('Select One of the Features')
self.groupBox1Layout = QGridLayout()
self.groupBox1.setLayout(self.groupBox1Layout)

self.feature = []

for i in range(13):
    self.feature.append(QCheckBox(features_list_hist[i], self))

for i in self.feature:
    i.setChecked(False)

self.btnExecute = QPushButton("Plot")

self.btnExecute.clicked.connect(self.update)

self.groupBox1Layout.addWidget(self.feature[0], 0, 0)
self.groupBox1Layout.addWidget(self.feature[1], 0, 1)
self.groupBox1Layout.addWidget(self.feature[2], 1, 0)
self.groupBox1Layout.addWidget(self.feature[3], 1, 1)
self.groupBox1Layout.addWidget(self.feature[4], 2, 0)
self.groupBox1Layout.addWidget(self.feature[5], 2, 1)
self.groupBox1Layout.addWidget(self.feature[6], 3, 0)
self.groupBox1Layout.addWidget(self.feature[7], 3, 1)
self.groupBox1Layout.addWidget(self.feature[8], 4, 0)
self.groupBox1Layout.addWidget(self.feature[9], 4, 1)
self.groupBox1Layout.addWidget(self.feature[10], 5, 0)
self.groupBox1Layout.addWidget(self.feature[11], 5, 1)
```



```
        self.groupBox1Layout.addWidget(self.btnExecute, 7, 1)

        self.fig1, self.ax1 = plt.subplots()
        self.axes = [self.ax1]
        self.canvas1 = FigureCanvas(self.fig1)

        self.canvas1.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)

        self.canvas1.updateGeometry()

        self.groupBoxG1 = QGroupBox('Histogram Plot :')
        self.groupBoxG1Layout = QVBoxLayout()
        self.groupBoxG1.setLayout(self.groupBoxG1Layout)

        self.groupBoxG1Layout.addWidget(self.canvas1)

        self.layout.addWidget(self.groupBox1, 0, 0)
        self.layout.addWidget(self.groupBoxG1, 0, 1)

        self.setCentralWidget(self.main_widget)
        self.resize(1200, 900)
        self.show()

    def Message(self):
        QMessageBox.about(self, "Warning", " You can't exceed more than 1 feature")

    def update(self):
        self.current_features = pd.DataFrame([])
        x_a = ''
        work = 0
```



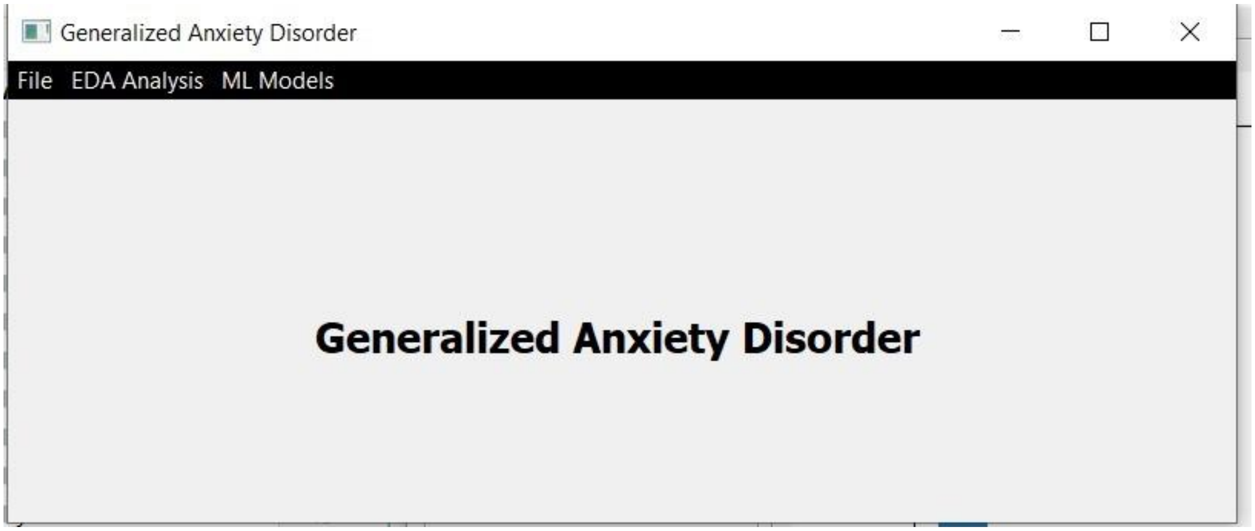
```
def Message(self):
    QMessageBox.about(self, "Warning", " You can't exceed more than 1 feature")

def update(self):
    self.current_features = pd.DataFrame([])
    x_a = ''
    work = 0
    for i in range(13):
        if self.feature[i].isChecked():
            if len(self.current_features) > 1:
                self.Message()
                work = 1
                break

            elif len(self.current_features) == 0:
                self.current_features = data[features_list_hist[i]]
                x_a = features_list_hist[i]
                work=0

    if work == 0:
        self.ax1.clear()
        self.current_features.value_counts().plot(kind='bar', ax=self.ax1)
        self.ax1.set_title('Histogram of : ' + x_a)
        self.ax1.set_xlabel(x_a)
        self.ax1.set_ylabel('frequency')
        self.fig1.tight_layout()
        self.fig1.canvas.draw_idle()
```

# 5.Results

Decision Tree Classifier:
Accuracy of model = 68.04%(Test size=30%)
Precision of 0’s=0.73 Precision of 1’s=0.49 Precision of 2’s = 0.63

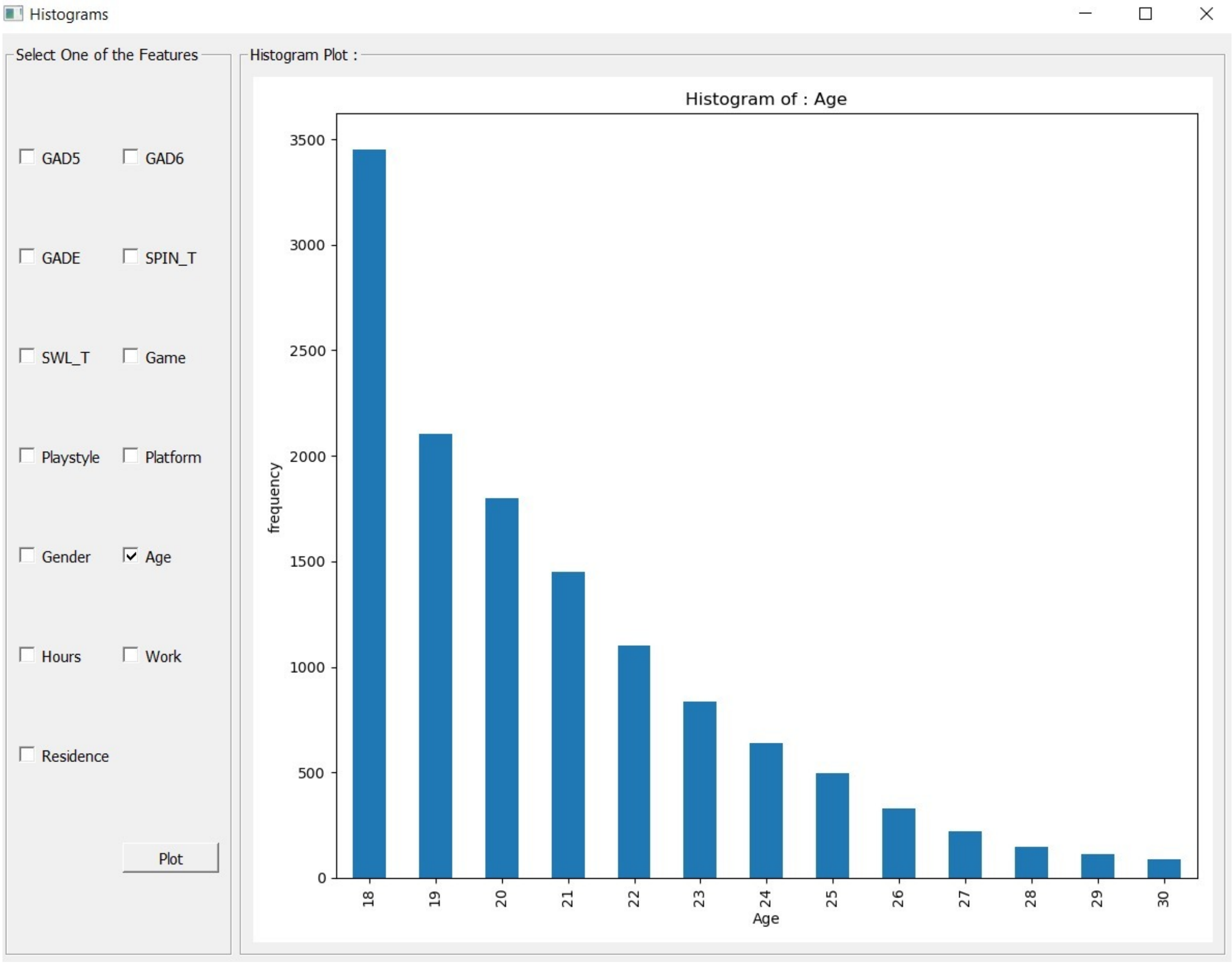
Random Forest Classifier:
Accuracy of model = 72.11% (Test size=30%)
Precision of 0’s=0.84 Precision of 1’s=0.50 Precision of 2’s= 0.65

Support Vector Classifier:
Accuracy of model = 73.47% (Test size= 30%)
Precision of 0’s=0.82 Precision of 1’s=0.54 Precision of 2’s = 0.72



Generalized Anxiety Disorder

**Histogram:** The graphic depicts the pictorial representation of the histogram as well as the numerous features available on the left grid, and the histogram is presented on the canvas after pressing the plot button.



Select One of the Features

☐ GAD5 ☐ GAD6

☐ GADE ☐ SPIN_T

☐ SWL_T ☐ Game

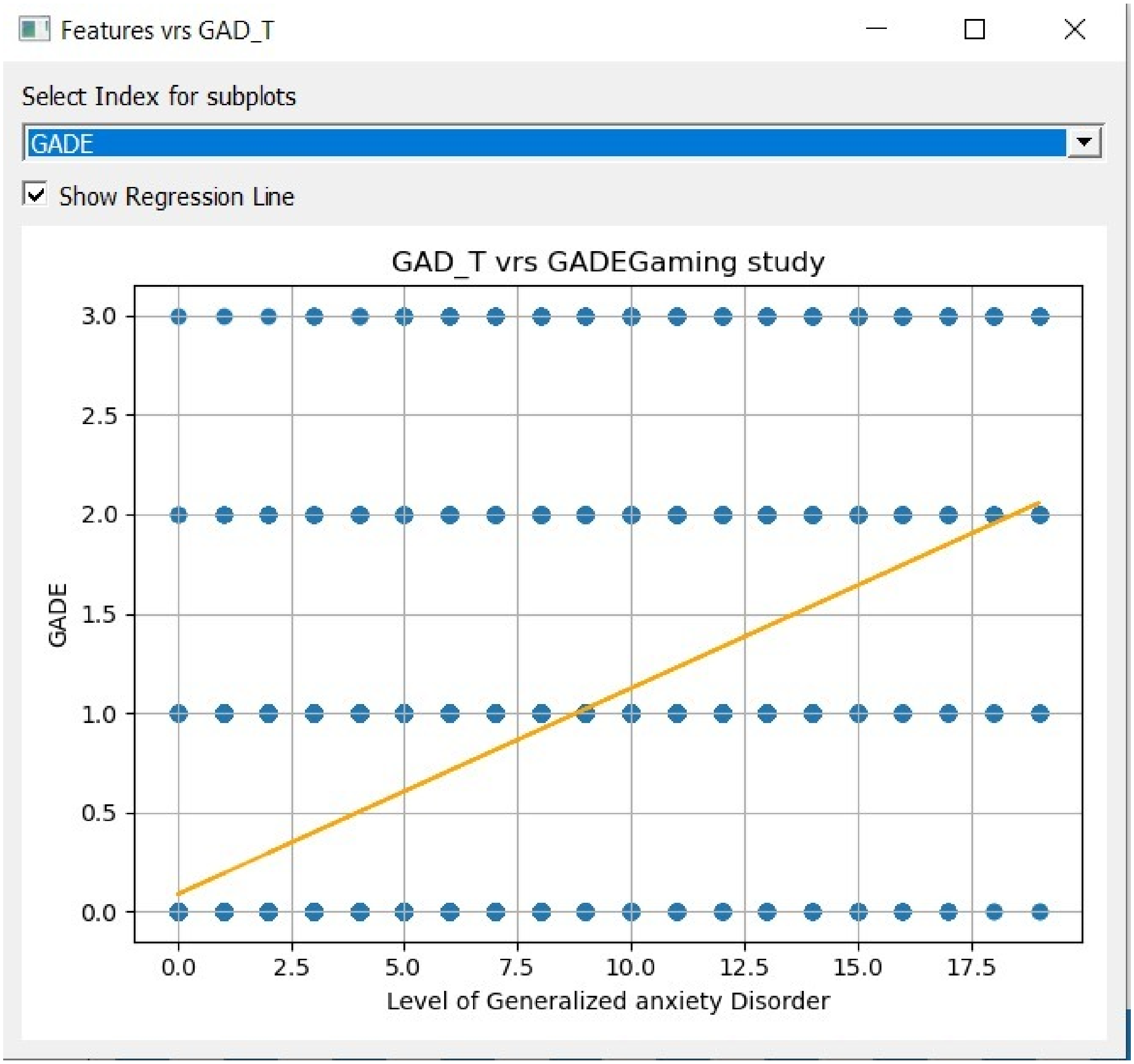
☐ Playstyle ☐ Platform

☐ Gender ☑ Age

☐ Hours ☐ Work

☐ Residence

**Scatter Plot:** Scatter plots are primarily used to observe and plot the relationships between two numerical variables. In addition to displaying the values of individual data points, the points in the scatter plot also represent the pattern when the data is displayed. Scatter plots are the graphs that present the relationship between two variables in a dataset. It represents data points on a two-dimensional plane or on a Cartesian system. The independent variable or attribute is plotted on the X-axis, while the dependent variable is plotted on the Y-axis. These plots are often called scatter graphs or scatter diagrams .



Features vrs GAD_T

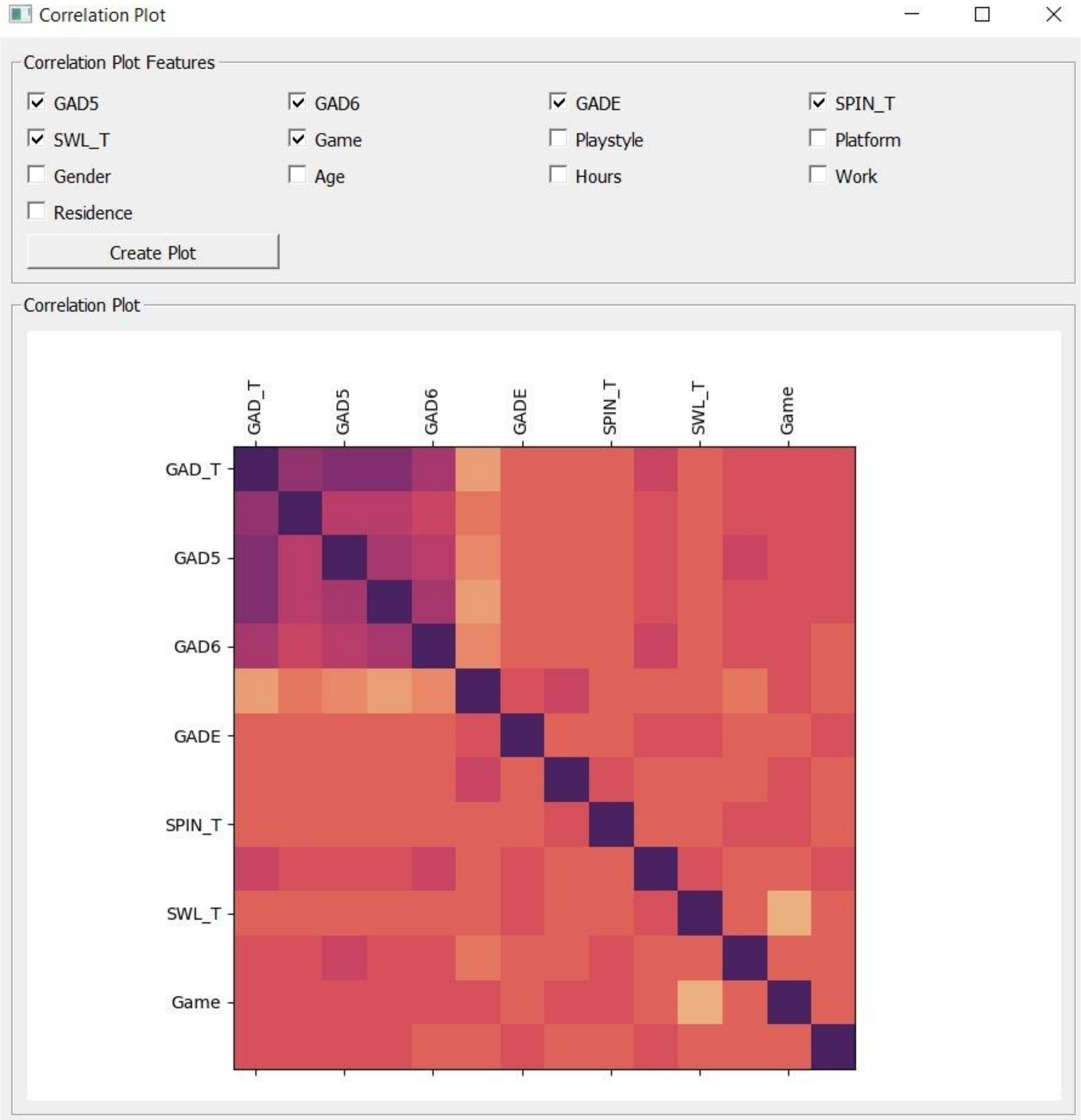
Select Index for subplots

GADE

☑ Show Regression Line

GAD_T vrs GADEGaming study
GADE
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Level of Generalized anxiety Disorder

**Heat Map:** A Heat plot is a graph that shows some aspects of the data as gradients. A simple example is a bivariate histogram. Color gradients are used to represent the (relative) frequency in the X and Y bins.



Correlation Plot

Correlation Plot Features

☑ GAD5 ☑ GAD6 ☑ GADE ☑ SPIN_T

☑ SWL_T ☑ Game □ Playstyle □ Platform

□ Gender □ Age □ Hours □ Work

□ Residence

Create Plot

Correlation Plot

# 5. Summary and Conclusions

- Comparing the results of models, almost all the three models have accuracy value in between 68% -74%.
- Support Vector Machine tops the list by having the highest accuracy of 73.47%
- The model in future enhancement needs to be tuned to predict the effect of online gaming on General Anxiety Disorder (GAD).

# 6.Percent of the code

0% of the code is used from the internet. A syntax search was done to apply the function, I have never copied it from the internet. The entire code was developed by individual coding developed by the professor. Amir Jafari.

# 7.Rerferences

- 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.javatpoint.com/