GUI Project
CSV File Application

Hejaab F Naqvi

Signed: Hejaab FN .....Date: 02/8/2024

Table of Contents:

Introduction

This report presents a Python-developed data analysis application. The program offers a graphical user interface (GUI) for examining CSV files that contain data about office workers, such as department, age, and pay. The project's main objective is to provide an intuitive and user-friendly interface for data visualization and analysis. The project uses a number of important tools, such as Matlab for plotting, pandas for data handling, and PyQt for the GUI.

Code:

```
import sys  #import the sys module
import pandas as pd  #import the pandas library
import matplotlib.pyplot as plt  #import the matlab plot module
from PyQt5.QtWidgets import QApplication, QMainWindow, QVBoxLayout,
QPushButton, QRadioButton, QFileDialog, QTableWidget, QTableWidgetItem,
QMessageBox, QWidget, QLabel, QComboBox, QSpinBox  #import all the widgets
needed
from PyQt5.QtGui import QPixmap  #import QPixmap
from PyQt5.QtCore import Qt  #import Qt

class DataAnalysisApp(QMainWindow):  #define the data analysis apps class
    def __init__(self):  #function called when code is ran
        super().__init__()  #making sure the class is set up correctly
        self.title = 'Data Analysis Application'  #add a title to the app
        self.initUI()  #call the initUI function within this function so
```

```
after the window is created

    def initUI(self):  #function to create the main functions of the code
        self.setWindowTitle(self.title)  #set the title we made earlier
        self.setGeometry(100, 100, 900, 700)  # set te winfow dimensions
        self.central_widget = QWidget()  # create the main widget which is
gonna contain the other widgets
        self.setCentralWidget(self.central_widget)  #set it as te central
widegt of the main window
        self.layout = QVBoxLayout(self.central_widget)  #creates a main
vertical column for all the widgets to be plced in
        self.load_button = QPushButton('Load CSV', self)  #create a button
called load cvs
        self.load_button.clicked.connect(self.loadCSV)  #when this button
is pressed run the loadCVS function
        self.layout.addWidget(self.load_button)  #add this button into the
main verticsl layout first thing after title
        self.plot_button = QPushButton('Generate Plot', self)  #create
another button called Generate plot
        self.plot_button.clicked.connect(self.generatePlot)  #run the
generateplot function when thius button is pressed
        self.layout.addWidget(self.plot_button)  #add this to the main
vertical layout next
        self.radio_line = QRadioButton('Line Plot', self)  #create a radio
button called Line plot
        self.radio_bar = QRadioButton('Bar Plot', self)  #another radio
button called Bar plot
        self.layout.addWidget(self.radio_line)  # add both the radio
buttons to the vertical column
        self.layout.addWidget(self.radio_bar)
        self.column_combo = QComboBox(self)  #create a combo box for the
data lists to be selected
        self.layout.addWidget(self.column_combo)  #add this combo box to
the main vertical layout
        self.top_n_spinbox = QSpinBox(self)  # make a spin box widget this
will help pick how many catagories to include in the plot
        self.top_n_spinbox.setRange(1, 100)  #set a range 100 just
depending on the cvs given
        self.top_n_spinbox.setValue(10)  #set a devault value of 10if less
than 10 availible will just put all the catagories
        self.layout.addWidget(self.top_n_spinbox)  #now add this to the
vertical main layout
        self.table_widget = QTableWidget()  #create a table widget so we
can display the dataset
        self.layout.addWidget(self.table_widget)  #add the table widget to
the vertucal layout
```

```
        self.plot_label = QLabel(self)  #create a lable widget so we can
display the plot
        self.layout.addWidget(self.plot_label)  #add the lable widget to
the main vertical layout too
        self.show()  #display the window onto the users screen

    def loadCSV(self):  #function is called when the user pushes the load
csv button
        options = QFileDialog.Options()  # File dialog options
        fileName, _ = QFileDialog.getOpenFileName(self, "Open CSV", "",
"CSV Files (*.csv);;All Files (*)", options=options)  # Open a file dialog
to select a CSV file
        if fileName:  #when the user selects a file
            try:  #attempts this code
                self.data = pd.read_csv(fileName)  #read the file path
into a pandas datafram
                self.displayData()  #shows this data in the table widget
                self.populateColumns()  #fills the columns of the combo
box with the column names in the file
            except Exception as e:  #if theres an error in loading the
file
                self.showMessageBox('Error', f'Could not load the
file:{str(e)}')  # diplays an error message

    def displayData(self):  # function to display the contents of the file
        self.table_widget.setRowCount(self.data.shape[0])  #make sure the
number of rows in the tablewidget match the rows in the data
        self.table_widget.setColumnCount(self.data.shape[1])  # match
number of columns in tbale to those in the data
        self.table_widget.setHorizontalHeaderLabels(self.data.columns)
#make the column names the same as the data

        for i in range(self.data.shape[0]):  #go through each row in the
index
            for j in range(self.data.shape[1]):  #go theough each column
                self.table_widget.setItem(i, j,
QTableWidgetItem(str(self.data.iat[i, j])))  # copy the rows and columns
of data into the table widget

    def populateColumns(self):  #function for filling the combo box
        self.column_combo.clear()  #clear any data from the combo box
        self.column_combo.addItems(self.data.columns)  #add the data from
the file into the combo box

    def generatePlot(self):  #call this function when the generate plot
```

```
button is pressed it creates the plots
        if hasattr(self, 'data'):  #make sure theres data loaded in
            if not (self.radio_line.isChecked() or
self.radio_bar.isChecked()):  #check if the user has bpicked the type of
plot they want
                self.showMessageBox('Error', 'Please select either Line
Plot or Bar Plot before generating.')  #if they havent ask the user to in
an error box
                return  # leave the generate plot function so user can
pick the plot type then try again
            column = self.column_combo.currentText()  # get the column
that was selected
            top_n = self.top_n_spinbox.value()  #sets the value the user
put in the spin box so how many items will be plotted
            if column:  #if a column was selected
                plt.figure()  # Create a new figure for plot
                ax = plt.subplot(111)  #create a subplot for the plot
                if self.radio_line.isChecked():  #if the user selected the
line plot
                    if self.data[column].dtype in ['int64', 'float64']:  #
check if the column selected has numerical data
                        self.data[column].head(top_n).plot(ax=ax,
kind='line')  #Plot the line plot
                        ax.set_title(f'Line Plot of {column}') #give te
plot a title
                        ax.set_xlabel('Index')  #label the x axis
                        ax.set_ylabel(column)  #and the y axis
                    else:  #if the sata isnt numerical
                        self.showMessageBox('Error', 'Line plot is only
applicable for numeric data.')  #show an error so the user can pick a
diffrent type of plot
                        return  # exit the function so uder can pick a
diffrent plot type
                elif self.radio_bar.isChecked():  #if the user picks the
bar plot option
                    if self.data[column].dtype == 'object':  # check if
the data selected is catagorical
                        value_counts =
self.data[column].value_counts().nlargest(top_n)  # get the number of
catagories
                        value_counts.plot(ax=ax, kind='bar')  # Generate
the bar plot
                        ax.set_xticklabels(value_counts.index,
rotation=90)  # Rotate the x-axis labels for better readability
                        ax.set_title(f'Bar Plot of {column}')  #give the
bar plot a title and set a lable for x and y axis
                        ax.set_xlabel(column)
```

```
                        ax.set_ylabel('Count')
                    else:  #if the sata selected isnt catagorical then
                        self.showMessageBox('Error', 'Bar plot is only
applicable for categorical data.')  #show an error and exit from the
function
                        return

                plt.tight_layout()  # Adjust plot layout so that it fits
into the figure area
                plt.savefig('plot.png')  # Save the plot as an image file
                self.displayPlot('plot.png')  # display the plot
            else:
                self.showMessageBox('Error', 'No column selected to
plot.')  # Show error if no column was selected
        else:
            self.showMessageBox('Error', 'No data loaded to plot.')  #
Show error if there was no data loaded

    def displayPlot(self, plot_path): #function to make the plot an image
and save it
        pixmap = QPixmap(plot_path)  #using the filepath load the image
from the file
        self.plot_label.setPixmap(pixmap)  #display the loaded image
        self.plot_label.setAlignment(Qt.AlignCenter)  #center the image
        self.plot_label.setScaledContents(True)  #sizing the image so it
fits right

    def showMessageBox(self, title, message):  #function for the message
boxes
        msg = QMessageBox()  #create a message box
        msg.setIcon(QMessageBox.Information)  #add the message icon
        msg.setText(message)  #set the message argument as the text
        msg.setWindowTitle(title)  #set the title agrument as the title of
the message box
        msg.exec_()  #when the user presses the ok button the window will
close

if __name__ == '__main__':  #when the script is being directly run
    app = QApplication(sys.argv)  #start running the application
    ex = DataAnalysisApp()  #initailizes and displayes the main window
    sys.exit(app.exec_())  # starts the main loop and makes sure the
window exits properly

#end
```

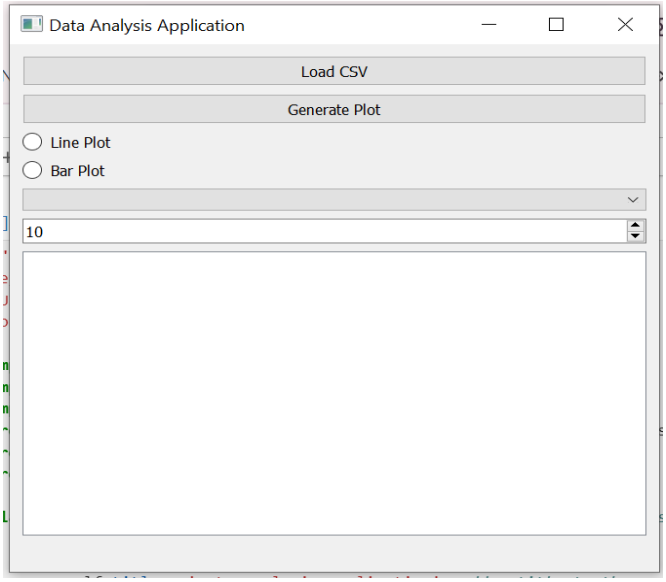
Technical Overview:

When the user launches the Data Analysis Application, the main window pops up asking the user to load in their CSV file. After selecting a file, the data is displayed in a table and the columns names are loaded into a combo box for the user to pick from. A spin box allows the user to pick the number of top categories they want displayed in their plot, which is useful for large datasets. They then choose the desired plot type, and the application generates and displays the plot based on their selection.

Technologies used

PyQt5:

PyQt5 is the latest version of GUI widgets toolkit and is a blend of the Python programming language and the Qt library. It is used to create the GUI of this project. In this code PyQt provides a wide range of widgets which are used to build the application's interface.

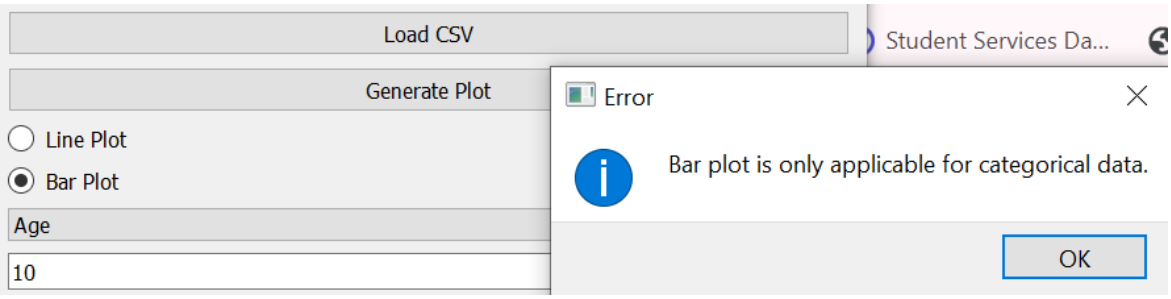


[Figure 1]

The main window of the application consists of a vertical layout which includes the buttons for loading CSV files and generating the plots, a combo box for selecting the data columns, and a table widget for displaying the data.
Figure 1 shows the layout that appears when the code is run, we can see all the widgets created using PyQt5.

The application also uses PyQt5 to include error handling in the code to ensure smooth user interaction. Appropriate error messages are displayed if issues arise. For example, figure 2 shows what happens if the user chooses the wrong

plot type for the category selected.



We can see the user tried to use a bar plot to generate numerical data, so the application tells the user to change the plot type.

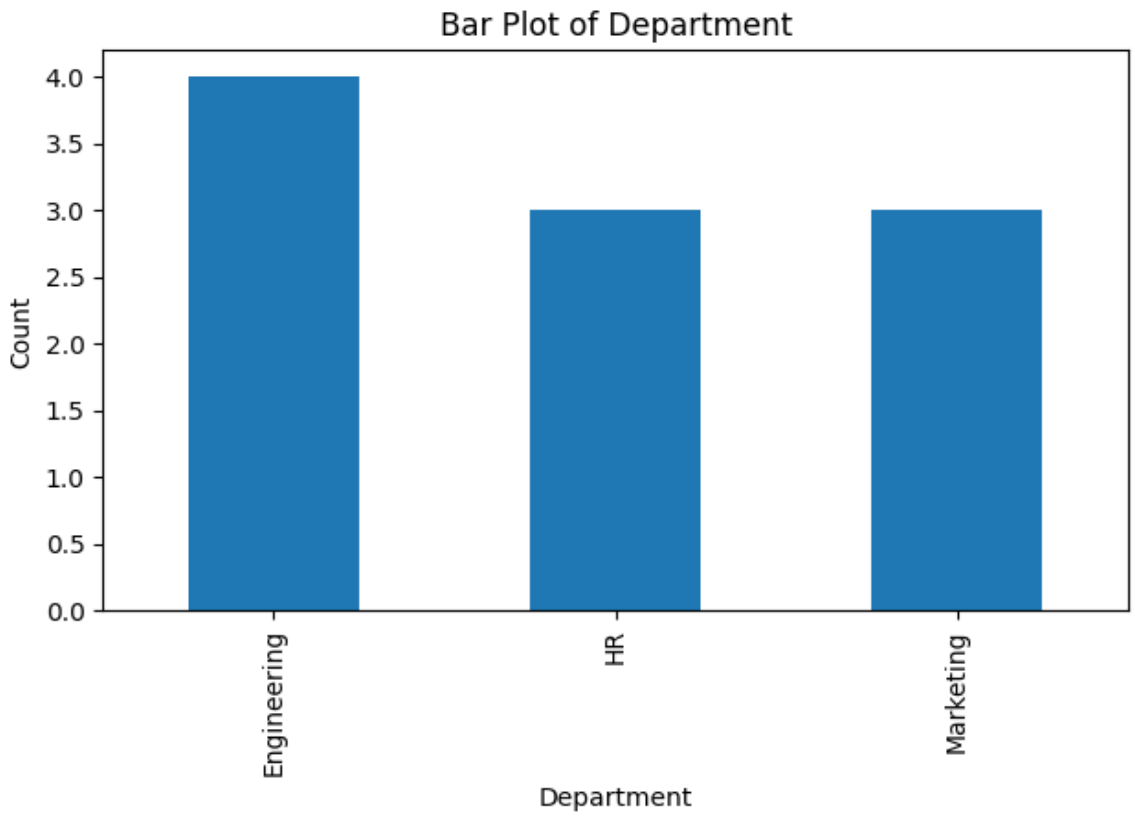
Pandas:

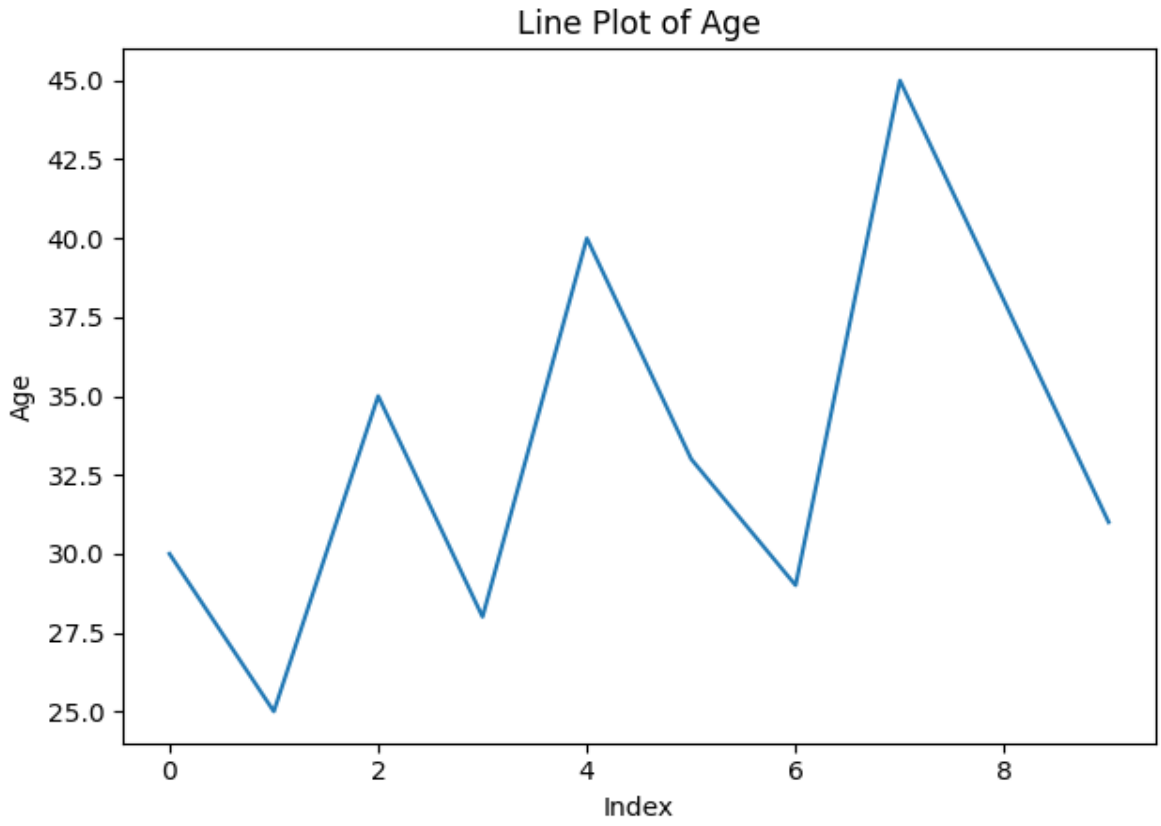
Pandas is a powerful data manipulation and analysis library for Python. It is used to read and process the CSV file in this project. The load cvs function shown below is used to read the file's data into Pandas. When a CSV file is loaded by the user, pandas reads it into a dataframe, which allows for easy manipulation and analysis of the data. The dataframe is then displayed in the application's table widget, and the column names are filled into the combo box, for the user to choose from.

```
def loadCSV(self):  #function is called when the user pushes the load csv button
    options = QFileDialog.Options()  # File dialog options
    fileName, _ = QFileDialog.getOpenFileName(self, "Open CSV", "", "CSV Files (*.csv);;All Files (*)", options=options)
    if fileName:  #when the user selects a file
        try:  #attempts this code
            self.data = pd.read_csv(fileName)  #read the file path into a pandas datafram
            self.displayData()  #shows this data in the table widget
            self.populateColumns()  #fills the columns of the combo box with the column names in the file
        except Exception as e:  #if theres an error in loading the file
            self.showMessageBox('Error', f'Could not load the file: {str(e)}')  # diplays an error message
```

Matplotlib:

Matplotlib is a plotting library for Python, and it is used for data visualization. In this code Matplotlip is used to generate the plot depending on the user's selection of either a line or bar plot. The line plots are used for numerical data, while the bar plots are used for categorical data. The generated plot is then saved as an image and displayed within the application. Using our office workers CSV file the user can pick the category of either categorical or numerical data to generate the plot they want, below we can see an example of a bar plot and a line plot generated by the application.


Bar Plot of Department
Count
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Engineering
HR
Marketing
Department


Line Plot of Age
Age
45.0
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
0
2
4
6
8
Index

Dataset:

The dataset used in this project consists of data on office employees, including their age, salary, and department. For this purpose of this project, a CSV file was created with sample data to demonstrate the application's functions. The CSV file contains columns with the employee's age, pay, department, and other relevant information. This data is used to show how this application can load, display, and visualize different aspects of the data.

Conclusion

This project demonstrates how to combine several Python libraries to create a useful data analysis application. It demonstrates the integration of PyQt, pandas, and matplotlib to create a functional and easy to use tool for data analysis.

Users may analyze and comprehend complex datasets more easily with the help of this program since it offers a simple interface for loading, presenting, and visualizing their CSV file. Building this application helped me gain a deeper understanding of how different Python libraries can work together to solve real-world problems. This project has enhanced my practical skills in data analysis and software development.

References:

https://drive.google.com/file/d/1GtMytMBSUeGpP_J46N3zg0xo7Ek_26P9/view?usp=drive_link (CVS file used)