Student Data Analysis

A Python program for academic data analysis of students

Student:

Abhilash Reddy Peram (10532966)

Sabitha Maram (10533048)

Srikanth Shilesh Pasam (10387794)

Teacher:

Paul Laird

Course:

Programming for Information Systems (B9IS123)

Table of Contents

[Table of Figures 3](#_Toc29254674)

[Introduction 4](#_Toc29254675)

[Requirements 5](#_Toc29254676)

[Functional Requirements 5](#_Toc29254677)

[Non-Functional Requirements 5](#_Toc29254678)

[Student Data 6](#_Toc29254679)

[Program Features 6](#_Toc29254680)

[Stage 1 – Login/Register 6](#_Toc29254681)

[Stage 2 – Loading files 6](#_Toc29254682)

[Stage 3 – Data Analysis 7](#_Toc29254683)

[Implementation 8](#_Toc29254684)

[Individual Contributions 16](#_Toc29254685)

[Srikanth Shilesh Pasam 16](#_Toc29254686)

[Reflection 16](#_Toc29254687)

[Sabitha Maram 17](#_Toc29254688)

[Abhilash Reddy Peram 18](#_Toc29254689)

[Appendix 1 19](#_Toc29254690)

[Appendix 2 20](#_Toc29254691)

# Table of Figures

[Figure 1 Login Section 7](#_Toc29253862)

[Figure 2 Creating new user 7](#_Toc29253863)

[Figure 3 Login of existing user 8](#_Toc29253864)

[Figure 4 Selecting folder location and opening file 9](#_Toc29253865)

[Figure 5 Calculating the class average 11](#_Toc29253866)

[Figure 6 Subject wise marks sorted display 11](#_Toc29253867)

[Figure 7 Searching for a student 12](#_Toc29253868)

[Figure 8 Plotting graph of student progress 13](#_Toc29253869)

[Figure 9 Plotting graph of class progress 14](#_Toc29253870)

[Figure 10 Adding or editing student details 14](#_Toc29253871)

[Figure 11 New student details entry 15](#_Toc29253872)

[Figure 12 Editing existing student details 16](#_Toc29253873)

[Figure 13 Successfully edited student details 16](#_Toc29253874)

[Figure 14 Fernet key used in encryption 17](#_Toc29253875)

[Figure 15 Encrypted and encoded user login details 17](#_Toc29253876)

[Figure 16 Saving generated graphs to local memory 17](#_Toc29253877)

# Introduction

The Student Data Analysis is a program which is designed to analyze a students or classrooms academic performance. It has been designed with the purpose of addressing the needs of Teach For India. Teach For India is a non-government non-profit organization and is part of Teach For All network. The organizations aim is to tackle the problems of education inequity in the country by providing excellent education to every child. As a part of this organization, qualified and motivated Fellows are placed in low-income private schools or under resourced government schools to drive change. During this time, the fellows collect large volumes of student and school data ranging from academic performance, culture and values of the class, social and economic levels of the parents, teacher resources, training modules, infrastructure of the school, school income and expenditure, external stakeholder investments and much more. All this data is used by the fellows to analyze and create action plans for the improvement of the quality of education received. Currently the organization has 900 fellows, 250 staff and 3000 alumni across 7 cities in India impacting 260 schools and 32000 students directly. As of this day, the organization only uses spreadsheets in order to record and analyze all the student data collected. This task gets increasingly difficult every year as the organization keeps expanding and reaches out to more and more schools and students. The efficiency with which all this data can be analyzed and used is severely limited to the scope and extent of what a spreadsheet can do.

This project aims to tackle a small portion of this issue as a part of this assignment. The program takes in the raw student data from the existing spreadsheets (.xlsx or .csv) runs them through the program to provide an analysis of the student or classrooms academic performance. The current program only focuses on academic analysis within the scope of this CA but plans are made to integrate other aspects like deployment in the Cloud, a backend database system, web interfacing along with a mobile app for GUI in the future.

# Requirements

## Functional Requirements

1. As the School Leader, I need to see a summary of the marks of each subject for each grade so that I can identify the areas of development for the respective grade. (Reporting feature)
2. As a School Leader, I need to see a summary of the marks scored by each student for a subject so that I can identify the areas of development for the respective subject teacher. (Reporting feature)
3. As a Class Teacher, I need to see a summary of the marks scored by each student for all subjects so that I can give feedback to students individually. (Search feature)
4. As a Class Teacher, I would like to be able to add or edit students data into the system. (Entry/update feature)
5. As a Subject Teacher, I want to see a summary of the marks scored by students in my subject so that I can identify which student requires extra classes scheduled.
6. As a Parent, I need to see a summary of the marks scored by my child so that I can keep a track of his/her academic progress. (Search feature)
7. As an external stakeholder, I need to see a summary of the growth of a classroom over 3 years in the form of graph so that I can decide if I should continue funding Teach For India. (Reporting feature)
8. As the City Director of Teach For India, I want access to the Student Data Analysis System to be provided only after proper authentication has taken place because it contains sensitive information of school and students. (Authentication feature)

## Non-Functional Requirements

1. As a fellow who frequently accesses the system, I want the program to load up quickly so that I can finish up my work faster.
2. As the School Relations Manager of Teach For India, I want the graphs to use the Teach For India’s blue color so that it matches the impact growth document shared with the stakeholders.

# Student Data

The organization uses spreadsheets to store their data. For the purpose of the CA, the organization agreed to let us use few of their historic data sets. They have been included in the CA submission under the ‘Source’ folder. From this data sets, eight CSV files have been created which will act as the raw data file for the program demonstration. These CSV files have the student roll, student name, gender, RC (reading comprehension), listening, writing and math columns. The grading system for each subject is different. RC levels range from -0.5 to 5.5, writing starts at 0 and goes all the way to 8, listening is from 0 to 5 and math is percentage based. There is much more data in the source files but for demonstration purposes we will be limiting the scope of the program to only these elements.

# Program Features

## Stage 1 – Login/Register

The program uses CLI as the interaction tool with the user. On executing the program, the user will be asked to either login or register.

* In both the login and register phases the user details are encoded and encrypted for security.
* The password input is taken using the GetPass library function which hides the password entry.
* The encryption is done using the cryptography library Fernet which uses symmetric encryption.
* The Fernet key is stored locally in the project folder under the file name 'Fernet\_key.txt’ for the sake of demonstration.
* The registered user details are encoded into byte code format. This is then encrypted using the Fernet Key created. All this data is stored locally under the file name ‘User\_details.txt’ for the sake of demonstration.

## Stage 2 – Loading files

Once the user logs in successfully they are granted access to the program.

* To access the files on the system the program uses Tkinter library functions.
* The Tkinter provides a dialogue box for the user to select the directory where the files are located.
* Once the user selects the directory, all the files in that directory are listed using the OS library function.
* The user then needs to select a file from that list which needs to be processed.
* After the file is selected, the program loads the CSV file into a DataFrame using the Pandas library functions.

## Stage 3 – Data Analysis

Once the files are loaded, the program can perform data analysis on it depending on what the user wants to do.

* The user has the option to:
  + - * + Calculate the class averages of each grade.
        + Calculate individual subject average for each grade.
        + Search for a student using their full name or part of their name and display their corresponding subject-wise marks.
        + Using the MatPlotLib library function to plot the graph which show the classroom growth for each subject over the span of three years.
        + Plot the graph to show an individual student growth for each subject over the span of three years.
  + The graphs which are generated post the analysis is saved locally in the 'Graphs' folder using a certain naming convention for user access.
  + The user can add new student details or edit existing ones.

# Implementation

The working code is included in Appendix 2 for reference. The implementation of the program is shown using screenshots.

A screenshot of a cell phone

Description automatically generated

Figure Login Section

A screenshot of a cell phone

Description automatically generated

Figure Creating new user

A screenshot of a cell phone

Description automatically generated

Figure Login of existing user

A screenshot of a cell phone

Description automatically generated

Figure Selecting folder location and opening file

A close up of text on a white background

Description automatically generated

Figure Calculating the class average

A close up of text on a white background

Description automatically generated

Figure Subject wise marks sorted display

A close up of text on a white background

Description automatically generated

Figure Searching for a student

A screenshot of a cell phone

Description automatically generated

Figure Plotting graph of student progress

A screenshot of a cell phone

Description automatically generated

Figure Plotting graph of class progress

A screenshot of a cell phone

Description automatically generated

Figure Adding or editing student details

A screenshot of a cell phone

Description automatically generated

Figure New student details entry

A screenshot of a cell phone

Description automatically generated

Figure Editing existing student details

A screenshot of a cell phone

Description automatically generated

Figure Successfully edited student details

A screenshot of a cell phone

Description automatically generated

Figure Fernet key used in encryption

A close up of text on a black background

Description automatically generated

Figure Encrypted and encoded user login details

A screenshot of a computer

Description automatically generated

Figure Saving generated graphs to local memory

# Individual Contributions

## Srikanth Shilesh Pasam

The three of us as a team contributed towards all the elements of the code. Each aspect has been discussed and brainstormed. This collaboration has helped us to learn from each other. Though inputs were taken from all team members, my major contribution for the CA is as follows:

1. Functional and non-functional requirements
2. User choice logic flow
3. LoadData() class
4. Avg() class
5. SubjectMarks() class
6. StudentMarks() class
7. Auth() class
8. Keys() class
9. Report writing
10. Power point presentation

## Reflection

I come from Electronics and Communication Engineering background with no prior knowledge of the subject. My professional experience has been majorly in Teach for India, a non-government organization within the education sector. So, I have no professional experience working with Python or any other form of programming language either. Because of this I had to put in extra efforts to learn everything form the scratch. This proved extremely difficult during the initial days of the course where I had to learn a new language for each subject in the module. Nonetheless I had to push myself as this was a challenge I willingly signed up for. A major setback I faced was during the first in-class test. I performed poorly and this was my wakeup call to roll up my sleeves. I took it upon myself to study and improve my command over Python as much as possible. The time I got during reading week proved very useful for this. I signed up for courses on Udemy and spent major of my time practicing python and trying to perfect the program I failed to do during the in-class test. Through a lot of hard work and perseverance I slowly but steadily started getting more comfortable and fluent with Python. I was able to complete the program and submit it nine days early on Moodle. This built a good sense of accomplishment within myself and motivated me to push further and harder.

Most of us in class had difficulty during the initial days with the subject due to various reasons and many a times we have let him down. But professor Paul has been extremely patient with us. He taught us from the scratch, helped us practice, shared resources for self-study and even followed up with our progress on Boot Camp. All this constant push and effort is what kept me motivated to try harder. Being a teacher myself I can personally relate to the amount of efforts that go into balancing time crunch in covering the syllabus and making sure the students are able to keep up with the pace. This only made me push myself and put in that extra hour every night. I am especially grateful to professor Paul for introducing me to GitHub. Coming from non tech background I had absolutely no awareness about it. But since then I have been very active on GitHub. I have even used it for all my other modules. In fact, I even based part of my CA in Network and System Administration module on GitHub using the Continuous Integration and Continuous Deployment concept.

This project idea has an association with me on a very personal level. I have been part of Teach For India for many years. My deep desire to impact change in the society by being an advocate for ed-equity is what made me choose this line of work. During my time in the organization I have noticed that one particular area of development here is the requirement of a proper data analysis system. Even in this current age of technology, the organizations vision was severely hampered by the lack of this system. This realization is also one of the major reasons for me to choose this particular course. I intend to do my final year thesis project around developing a system that can address these issues for Teach For India and donate the finished application. To this end I am confident that the module –Programming for Information Systems, the programming language – Python and the professor – Mr. Paul Laird has been a tremendous help in helping me get closer to achieving my goals.

## Sabitha Maram

It was really good experience working in group where we can share ideas and knowledge of different people and put them together. It was really a tough and challenging experience for me to learn python programming as I am not into programming in my previous Education.

For this Assignment we discussed and decided to create Student Data Analysis. The key features in this work are User Authentication, Calculating Class averages and student averages, Visualizing student and class average in the form of pictorial representation in the form of graphs and Student data entry. The area where i worked individually is Graph plots and tried to do as much I learned. Srikanth in my group helped me to go through this to the end.

## Abhilash Reddy Peram

This is very good learning experience with a new group of people who are our fellow members in class. Sitting all together and sharing knowledge on whole new subject with each other is very useful. We learned python it through Udemy classes and worked on the assignment whenever we got time while completing the other assignment challenges. And even data camp. Initially it’s a little bit slow to learn and catch up with each other eventually we got along with the assignment.

We have gone through the requirements and figured out what to be done with the data files we have. So, we chose to deal with student data files based on their academic performance. The actual data sets are gathered from the existing data base of a school by our group member Srikanth because has worked in teaching profession for a while in the past. We tried all the possible ways to meet the given requirement criteria in the module. After getting all the data files we all sorted out our work with each other. I have taken the authentication and data entry for the data files. Working on authentication is quite fun and interesting thing to learn like giving access to the users so that only they can have the access with the decryption side. And data entry is like updating the new student data if any student profile gets missed out. While I got struck with some problem in using keys and encryption my group members helped me with the problems and we worked together if any problem raises for any of the team members.

Finally, while coming to the module learning its very good experience to interact with the lecturer. This is the part I like the most as we came from the places where we have little bit interaction with the lecturer but, here MR. Paul keep on persist us to speak up and ask for the doubts and even ask us to share our ideas with him. We are very much great full for having MR. Paul as our module lecturer.

# Appendix 1

The GitHub link for our project is:

<https://github.com/sabithamaram/PIS_B9IS123_CA2>

We have not realized until it’s too late that all our individual commits were being made to the Master folder directly where instead we needed to fork the Master directory, commit to the forked directory and later on merge it to the Master. Hence, we all have the same GitHub link to share though each of our individual commits are visible in the ‘Insights’ section of the GitHub folder shared above.

# Appendix 2

# Importing required libraries  
  
import pandas as pd  
from tabulate import tabulate as tb  
import matplotlib.pyplot as plt  
import numpy as np  
import os  
import tkinter  
from tkinter import filedialog  
from datetime import datetime  
from getpass import getpass  
from cryptography.fernet import Fernet  
  
# Pandas to use DataFrames, reading and writing external files  
# Tabulate to display data in a neat tabular form  
# Matplotlib to plot graphs for student data analysis  
# OS to list the files in a directory  
# Tkinter to allow user to select a directory from where to load the external files  
# Datatime used as part of naming convention when saving generated graph files  
  
  
# Creating a class to read the external CSV data files  
class LoadData:  
   
 def \_\_init\_\_(self, choice):  
 self.choice = choice  
  
 # Creating a method to allow user to select the directory and location of file to open   
 def file\_select(self):  
 # Initializing the tkinter function  
 root = tkinter.Tk()  
   
 # Getting the directory location of the data files from the user. Initially providing a default location  
 dir\_name = filedialog.askdirectory(parent=root, initialdir="/Users/srikanthshileshpasam/OneDrive - Dublin Business School (DBS)/Python/CA2/Data/", title='Please select a directory')  
   
 root.quit() # Reference link - https://stackoverflow.com/questions/28590669/tkinter-tkfiledialog-doesnt-exist/28590707  
   
 print(f'\nThe directory path you selected is:\n{dir\_name}')  
   
 # Listing all the files in the directory selected by the user  
 files\_list = os.listdir(dir\_name) # Reference link - https://stackoverflow.com/questions/3207219/how-do-i-list-all-files-of-a-directory  
   
 print(f'\nThe available files are:\n {files\_list}')  
   
 # Choice 4 needs multiple files to be loaded. Hence checking for this first  
 if self.choice == 4:  
 grade\_choice = int(input('\nChoose a grade\n'))  
   
 if grade\_choice == 1:  
 # Loading the chosen grade data files from the directory path selected by the user  
 first\_data\_file = pd.read\_csv(dir\_name + '/' + '1\_2014.csv')  
 sec\_data\_file = pd.read\_csv(dir\_name + '/' + '1\_2015.csv')  
 third\_data\_file = pd.read\_csv(dir\_name + '/' + '1\_2016.csv')  
   
 return (first\_data\_file, sec\_data\_file, third\_data\_file, dir\_name, grade\_choice)  
   
 elif grade\_choice == 2:  
 first\_data\_file = pd.read\_csv(dir\_name + '/' + '2\_2014.csv')  
 sec\_data\_file = pd.read\_csv(dir\_name + '/' + '2\_2015.csv')  
 third\_data\_file = pd.read\_csv(dir\_name + '/' + '2\_2016.csv')  
   
 return (first\_data\_file, sec\_data\_file, third\_data\_file, dir\_name, grade\_choice)  
   
 else:  
 print('Grade data does not exist!')  
   
 # If any other choice then only a single file needs to be opened  
 else:  
 # Getting the name of the file to be opened from the user  
 file\_name = input('\nSelect a file from above:\n')  
   
 return (pd.read\_csv(dir\_name + '/' + file\_name + '.csv'), dir\_name, file\_name)   
   
  
# Creating a class to allow for manual entry/edit of student details  
class DataEntry:  
   
 def \_\_init\_\_(self, dir\_path, data\_file\_name, s\_data, s\_name, s\_gender, roll\_num, s\_rc, s\_lis, s\_wri, s\_math):  
 self.dir\_path = dir\_path  
 self.data\_file\_name = data\_file\_name  
 self.s\_data = s\_data  
 self.s\_name = s\_name  
 self.s\_gender = s\_gender  
 self.roll\_num = roll\_num  
 self.s\_rc = s\_rc  
 self.s\_lis = s\_lis  
 self.s\_wri = s\_wri  
 self.s\_math = s\_math  
   
 # Creating a method to collect new student entry from user and saving to file  
 def data\_entry(self):  
 # appending a new row to the existing DataFrame file. Using key-value pairs of dictionary to write the data to the appropriate columns  
 self.s\_data = self.s\_data.append({'Order':self.roll\_num, 'Name':self.s\_name, 'Gender':self.s\_gender, 'RC':self.s\_rc, 'Listening':self.s\_lis, 'Writing':self.s\_wri, 'Math':self.s\_math }, ignore\_index=True)  
   
 print(f'\nData entered successfully.\nSaving file..\n{self.s\_data}')  
   
 # Saving the data to the file in the directory loaction and file name given by the user previously when opening the file  
 self.s\_data.to\_csv(self.dir\_path + '/' + self.data\_file\_name + '.csv', index=False)  
   
 return self.s\_data  
   
   
 # Creating a method to collect and edit existing student data from user and saving to file   
 def data\_edit(self):  
   
 self.s\_data.loc[self.roll\_num,'Name'] = self.s\_name  
 self.s\_data.loc[self.roll\_num,'Gender'] = self.s\_gender  
 self.s\_data.loc[self.roll\_num,'RC'] = self.s\_rc  
 self.s\_data.loc[self.roll\_num,'Listening'] = self.s\_lis  
 self.s\_data.loc[self.roll\_num,'Writing'] = self.s\_wri  
 self.s\_data.loc[self.roll\_num,'Math'] = self.s\_math  
   
 # Saving the data to the file in the directory loaction and file name given by the user previously when opening the file  
 self.s\_data.to\_csv(self.dir\_path + '/' + self.data\_file\_name + '.csv', index=False)  
  
 return self.s\_data  
  
   
# Creating a class to calculate subject-wise averages  
class Avg:  
   
 def \_\_init\_\_(self, avg\_data):  
 self.avg\_data = avg\_data  
   
   
 # Creating separate methods for each subject because in real world the actual data format of each subject will be different and hence calculating avegaes for each of them will require different approaches   
 # Creating a method for collecting the RC data from the DataFrame  
 def rc(self):  
 rc\_data = []  
   
 # Creating a for loop to iterate over all values of 'RC' row and appending them to a list 'rc\_data'  
 for index, row in self.avg\_data.iterrows(): # Reference link - https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas  
 reading\_row\_rc = row['RC']  
 rc\_data.append(reading\_row\_rc)  
   
 # Claculating the RC average  
 return sum(rc\_data)/len(rc\_data)  
   
   
 # Creating a method for collecting the Listening data from the DataFrame  
 def listening(self):  
 lis\_data = []  
   
 # Creating a for loop to iterate over all values of 'Listening' row and appending them to a list 'lis\_data'  
 for index, row in self.avg\_data.iterrows():  
 reading\_row\_lis = row['Listening']  
 lis\_data.append(reading\_row\_lis)  
   
 # Claculating the Listening average  
 return sum(lis\_data)/len(lis\_data)  
   
   
 # Creating a method for collecting the Writing data from the DataFrame  
 def writing(self):  
 wri\_data = []  
   
 # Creating a for loop to iterate over all values of 'Writing' row and appending them to a list 'wri\_data'  
 for index, row in self.avg\_data.iterrows():  
 reading\_row\_wri = row['Writing']  
 wri\_data.append(reading\_row\_wri)  
   
 # Claculating the Writing average  
 return sum(wri\_data)/len(wri\_data)  
   
   
 # Creating a method for collecting the Math data from the DataFrame  
 def math(self):  
 math\_data = []  
   
 # Creating a for loop to iterate over all values of 'Math' row and appending them to a list 'math\_data'  
 for index, row in self.avg\_data.iterrows():  
 reading\_row\_math = row['Math']  
 math\_data.append(reading\_row\_math)  
   
 # Claculating the Math average  
 return (sum(math\_data)/len(math\_data)) \* 100  
   
   
   
# Creating a class to tabulate subject-wise marks of a class during a particular year  
class SubjectMarks:  
   
 def \_\_init\_\_(self, sub\_data):  
 self.sub\_data = sub\_data  
   
   
 # Creating a method to collect 'RC' data of all students from the DataFrame   
 def rc(self):  
 # Selecting the columns from the DataFrame  
 data\_df = self.sub\_data[['Order', 'Name', 'RC']]  
 data\_df.sort\_values(by=['RC'], inplace = True, ascending=False)  
   
 # Returning tabulated data with the column headings  
 return tb(data\_df, headers=["Roll Order", "Name", "RC"], tablefmt='grid', showindex='never') #Reference link - https://pypi.org/project/tabulate/  
  
   
 # Creating a method to collect 'Listening' data of all students from the DataFrame   
 def listening(self):  
 # Selecting the columns from the DataFrame  
 data\_df = self.sub\_data[['Order', 'Name', 'Listening']]  
 data\_df.sort\_values(by=['Listening'], inplace = True, ascending=False)  
   
 # Returning tabulated data with the column headings  
 return tb(data\_df, headers=["Roll Order", "Name", "Listening"], tablefmt='grid', showindex='never')  
   
  
 # Creating a method to collect 'Writing' data of all students from the DataFrame   
 def writing(self):  
 # Selecting the columns from the DataFrame  
 data\_df = self.sub\_data[['Order', 'Name', 'Writing']]  
 data\_df.sort\_values(by=['Writing'], inplace = True, ascending=False)  
   
 # Returning tabulated data with the column headings  
 return tb(data\_df, headers=["Roll Order", "Name", "Writing"], tablefmt='grid', showindex='never')  
   
  
 # Creating a method to collect 'Math' data of all students from the DataFrame   
 def math(self):  
 # Selecting the columns from the DataFrame  
 data\_df = self.sub\_data[['Order', 'Name', 'Math']]  
 data\_df.sort\_values(by=['Math'], inplace = True, ascending=False)  
   
 # Returning tabulated data with the column headings  
 return tb(data\_df, headers=["Roll Order", "Name", "Math"], tablefmt='grid', showindex='never')  
   
   
  
# Creating a class to tabulate student marks individually  
class StudentMarks:  
   
 def \_\_init\_\_(self, stu\_data, stu\_name):  
 self.stu\_data = stu\_data  
 self.stu\_name = stu\_name  
   
   
 # Creating a method to search for a student name in the DataFrame and tabulate his/her details   
 def marks(self, class\_call=None):  
 self.class\_call = class\_call  
   
 # Selecting the columns to be displayed  
 data\_df = pd.DataFrame(self.stu\_data, columns = ['Order', 'Name', 'RC', 'Listening', 'Writing', 'Math'])  
   
 # Searching for the entire or part of student name in the 'Name' column of the DataFrame using the name entered by the user  
 data\_df = data\_df[data\_df['Name'].str.contains(self.stu\_name)] # Reference link - https://davidhamann.de/2017/06/26/pandas-select-elements-by-string/  
  
 if data\_df.empty == True: # Reference link - https://pandas.pydata.org/pandas-docs/version/0.18/generated/pandas.DataFrame.empty.html  
 return 'No student found!'  
   
 # Checking to see if data is requested from another class or for tabulating and displaying the DataFrame  
 elif class\_call == None:  
 return tb(data\_df, headers=["Roll Order", "Name", "RC", "Listening", "Writing", "Math"], tablefmt='grid', showindex='never')  
   
 # If data requested from another class then DataFrame is sent directly without tabulating it  
 else:  
 return data\_df  
  
  
  
# Creating a class to display the data in the form of graphs  
class GraphPlot:  
   
 def \_\_init\_\_(self, graph\_data\_1, graph\_data\_2, graph\_data\_3, dir\_loc, graph\_grade):  
 self.graph\_data\_1 = graph\_data\_1  
 self.graph\_data\_2 = graph\_data\_2  
 self.graph\_data\_3 = graph\_data\_3  
 self.dir\_loc = dir\_loc  
 self.graph\_grade = graph\_grade  
   
  
 # Creating a method to collect the average RC values of a class over all the years  
 def class\_plot\_rc(self):  
 # Calling the 'Avg' class within this class and sending data to process  
 class\_call = Avg(self.graph\_data\_1)  
   
 # Calling a method of a different class within this class to calculate the average levels of sent data  
 rc\_1 = class\_call.rc()  
   
 class\_call = Avg(self.graph\_data\_2)  
 rc\_2 = class\_call.rc()  
   
 class\_call = Avg(self.graph\_data\_3)  
 rc\_3 = class\_call.rc()  
   
 # Sending the calculated average levels to a different method within this class to plot the graph  
 self.plot\_graph(rc\_1, rc\_2, rc\_3, 'RC', self.dir\_loc, self.graph\_grade)  
   
   
 # Creating a method to collect the average Listening values of a class over all the years   
 def class\_plot\_lis(self):  
 class\_call = Avg(self.graph\_data\_1)  
 lis\_1 = class\_call.listening()  
   
 class\_call = Avg(self.graph\_data\_2)  
 lis\_2 = class\_call.listening()  
   
 class\_call = Avg(self.graph\_data\_3)  
 lis\_3 = class\_call.listening()  
   
 self.plot\_graph(lis\_1, lis\_2, lis\_3, 'Listening', self.dir\_loc, self.graph\_grade)  
   
   
 # Creating a method to collect the average Writing values of a class over all the years   
 def class\_plot\_writing(self):  
 class\_call = Avg(self.graph\_data\_1)  
 wri\_1 = class\_call.writing()  
   
 class\_call = Avg(self.graph\_data\_2)  
 wri\_2 = class\_call.writing()  
   
 class\_call = Avg(self.graph\_data\_3)  
 wri\_3 = class\_call.writing()  
   
 self.plot\_graph(wri\_1, wri\_2, wri\_3, 'Writing', self.dir\_loc, self.graph\_grade)  
   
   
 # Creating a method to collect the average Math values of a class over all the years  
 def class\_plot\_math(self):  
 class\_call = Avg(self.graph\_data\_1)  
 math\_1 = class\_call.math()  
   
 class\_call = Avg(self.graph\_data\_2)  
 math\_2 = class\_call.math()  
   
 class\_call = Avg(self.graph\_data\_3)  
 math\_3 = class\_call.math()  
   
 self.plot\_graph(math\_1, math\_2, math\_3, 'Math', self.dir\_loc, self.graph\_grade)  
  
   
 # Creating a method to collect individual students levels over the years   
 def stu\_plot(self, student\_name):  
 self.student\_name = student\_name  
   
 # Collecting year\_1 marks of a student  
 year\_1 = StudentMarks(self.graph\_data\_1, student\_name)  
   
 # Collecting all subject marks for a student   
 marks\_1 = year\_1.marks(True)  
   
 # Checking to see if student data exists for user entry before proceeding further  
 # If data exists, the 'StudentMarks' class will return a DataFrame which when checked by 'isinstance' will result in true  
 # If data does not exist then the 'StudentMarks' class will return a string and the if condition will result in false thus not executing further  
 if isinstance(marks\_1, pd.DataFrame):  
   
 # Looping through the individual student data to collect subject data individually  
 for index, row in marks\_1.iterrows():  
 rc\_1 = row['RC']  
 lis\_1 = row['Listening']  
 wri\_1 = row['Writing']  
 math\_1 = row['Math']  
   
 year\_2 = StudentMarks(self.graph\_data\_2, student\_name)  
 marks\_2 = year\_2.marks(True)  
   
 for index, row in marks\_2.iterrows():  
 rc\_2 = row['RC']  
 lis\_2 = row['Listening']  
 wri\_2 = row['Writing']  
 math\_2 = row['Math']  
  
 year\_3 = StudentMarks(self.graph\_data\_3, student\_name)  
 marks\_3 = year\_3.marks(True)  
   
 for index, row in marks\_3.iterrows():  
 rc\_3 = row['RC']  
 lis\_3 = row['Listening']  
 wri\_3 = row['Writing']  
 math\_3 = row['Math']  
   
 # Sending the individual student subject-wise data to another method within the class to plot the graph  
 self.plot\_graph(rc\_1, rc\_2, rc\_3, 'RC', self.dir\_loc, self.graph\_grade, student\_name)  
 self.plot\_graph(lis\_1, lis\_2, lis\_3, 'Listening', self.dir\_loc, self.graph\_grade, student\_name)  
 self.plot\_graph(wri\_1, wri\_2, wri\_3, 'Writing', self.dir\_loc, self.graph\_grade, student\_name)  
 self.plot\_graph(math\_1, math\_2, math\_3, 'Math', self.dir\_loc, self.graph\_grade, student\_name)  
   
 else:  
 print('No student data found!')  
   
   
 # Creating a method to plot graphs  
 def plot\_graph(self, data\_1, data\_2, data\_3, sub, loc, grade\_num, stud\_name=None):  
 self.data\_1 = data\_1  
 self.data\_2 = data\_2  
 self.data\_3 = data\_3  
 self.sub = sub  
 self.loc = loc  
 self.grade\_num = grade\_num  
 self.stud\_name = stud\_name  
   
 x = [2014, 2015, 2016]  
 y = [data\_1, data\_2, data\_3]  
   
 # Creating bar graph  
 plt.bar(x, y)  
 plt.xlabel('Year')  
 plt.ylabel('Level')  
 plt.title(sub) # Reference link - https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/  
  
 # Setting the x-axis lable frequency  
 plt.xticks(np.arange(min(x), max(x)+1, 1.0)) # Reference link - https://stackoverflow.com/questions/12608788/changing-the-tick-frequency-on-x-or-y-axis-in-matplotlib  
   
 # Saving the generated graphs to the location selected by the user with a certain naming convention  
 if stud\_name == None:  
 plt.savefig(loc + '/Graphs/' + 'Grade\_' + str(grade\_num) + '\_' + sub + datetime.now().strftime('%Y-%m-%d %H:%M:%S') + '.png') # Reference link - https://stackoverflow.com/questions/415511/how-to-get-the-current-time-in-python  
   
 else:  
 plt.savefig(loc + '/Graphs/' + stud\_name + '\_Grade\_' + str(grade\_num) + '\_' + sub + datetime.now().strftime('%Y-%m-%d %H:%M:%S') + '.png')  
   
 return plt.show()  
   
  
  
# Creating a class to authenticate user access  
class Auth:  
   
 def \_\_init\_\_(self, user=None, pwd=None, fer\_key=None):  
 self.user = user  
 self.pwd = pwd  
 self.fer\_key = fer\_key  
   
   
 # Creating a method to validate existing user credentials  
 def credentials(self):  
 with open("User\_Details.txt", "rb") as file:  
 data = file.readlines()  
 file.close()  
   
 x = False  
 y = 0  
 while x == False and y < len(data):  
 if (self.fer\_key.decrypt(self.user) == self.fer\_key.decrypt(data[y])) and (self.fer\_key.decrypt(self.pwd) == self.fer\_key.decrypt(data[y+1])):  
 x = True  
 return True  
 else:  
 y += 2  
   
 if x == False:  
 return False  
   
   
 # Creating a method to allow for new user registrations   
 def new\_user(self):  
 file = open("User\_Details.txt", "ab")  
   
 file.write(self.user)  
 line = str(0) + "\n"  
 file.write(line.encode('utf-8'))  
   
 file.write(self.pwd)  
 line = str(0) + "\n"  
 file.write(line.encode('utf-8'))  
   
 file.close()  
   
 return 'User created successfully!'  
   
  
  
# Creating a class to generate and read encryption keys  
class Keys:  
   
 # Creating a method for generating new encryption key  
 def key\_gen():  
 key = Fernet.generate\_key()  
   
 file = open("Fernet\_Key.txt", "wb")  
 file.write(key)  
 file.close()  
   
 print('New key generated successfully!')  
   
   
 # Creating a method for accessing the encryption key in order to decrypt the login credentials  
 def read\_key():  
 with open('Fernet\_Key.txt', 'rb') as file:  
 data = file.read()  
 file.close()  
  
 return data  
  
   
  
  
# Keys.key\_gen()  
  
  
  
  
login = int(input('\nWelcome to School Data Analysis Tracker!\nAre you an existing user or a new user?\n1) Existing user\n2) New user\n'))  
  
if login == 1:  
 user\_id = input('\nUsername: ')  
 user\_id = user\_id.encode()  
 pwd = getpass('Password: ')  
 pwd = pwd.encode() # Reference link - https://stackoverflow.com/questions/9202224/getting-command-line-password-input-in-python  
   
 key\_code = Keys  
 code = key\_code.read\_key()  
   
 key = Fernet(code)  
   
 encrypted\_pwd = key.encrypt(pwd)  
 encrypted\_id = key.encrypt(user\_id)  
   
 user\_auth = Auth(encrypted\_id, encrypted\_pwd, key)  
   
 login\_status = user\_auth.credentials()  
  
   
  
 if login\_status == True:  
 print('\nLogin Successful\n')  
  
 # User choice for data processing  
 choice = int(input('\n\nPick a choice to process data in the way required:\n1) Claculate the class average\n2) Subject-wise marks\n3) Student-wise marks\n4) Plot graph of class growth\n5) Add/edit student data\n'))  
  
 load\_data = LoadData(choice)  
  
 # If user wants to see the growth in the form of graphs  
 if choice == 4:  
 comp\_choice = input('Pick the growth type:\n1) Student Growth\n2) Class Growth\n')  
  
 print('\nSelect the data directory from the pop up window to load the classroom data files from\n')  
  
 # Collecting all the CSV files as a DataFrame for a user chosen grade level  
 first\_file, sec\_file, third\_file, d\_loc, grade = load\_data.file\_select()  
  
 print(f'\nThe raw data of Grade {grade} is shown below:\nYEAR 1\n{first\_file}\nYEAR 2\n{sec\_file}\nYEAR 3\n{third\_file}')  
  
 # Sending all the DataFrame files to the GraphPlot class to plot graph  
 graph\_plot = GraphPlot(first\_file, sec\_file, third\_file, d\_loc, grade)  
  
 # For individual student growth over the years  
 if comp\_choice == '1':  
 student = input('\nEnter name of student\n')  
 stu\_growth = graph\_plot.stu\_plot(student.title())  
  
 # For overall class growth over the years  
 elif comp\_choice == '2':  
 class\_growth\_rc = graph\_plot.class\_plot\_rc()  
 class\_growth\_lis = graph\_plot.class\_plot\_lis()  
 class\_growth\_writing = graph\_plot.class\_plot\_writing()  
 class\_growth\_math = graph\_plot.class\_plot\_math()  
  
  
 elif choice == 1 or choice == 2 or choice == 3 or choice == 5:  
 print('\nSelect a directory from the pop up window to begin\n')  
  
 master\_data\_file, d\_loc, f\_name = load\_data.file\_select()  
  
 print(f'\nOpening file...\n{master\_data\_file}')  
  
 # If user wants overall class avarages subject-wise for a particular year  
 if choice == 1:  
 class\_average = Avg(master\_data\_file)  
  
 class\_average\_rc = class\_average.rc()  
 class\_average\_lis = class\_average.listening()  
 class\_average\_writing = class\_average.writing()  
 class\_average\_math = class\_average.math()  
  
 print('\n\n\nThe class average is as below:\n\nRC: %.2f' %class\_average\_rc)  
 print('\nListening: %.2f' %class\_average\_lis)  
 print('\nWriting: %.2f' %class\_average\_writing)  
 print('\nMath: %.2f' %class\_average\_math)  
  
 # If user wants subject-wise marks of a class for a particular year  
 elif choice == 2:  
 subject\_wise\_marks = SubjectMarks(master\_data\_file)  
  
 class\_rc = subject\_wise\_marks.rc()  
 class\_lis = subject\_wise\_marks.listening()  
 class\_writing = subject\_wise\_marks.writing()  
 class\_math = subject\_wise\_marks.math()  
  
 print(f'\n\n\nThe class RC marks from the highest to lowest are:\n{class\_rc}')  
 print(f'\n\n\nThe class Listening marks from the highest to lowest are:\n{class\_lis}')  
 print(f'\n\n\nThe class Writing marks from the highest to lowest are:\n{class\_writing}')  
 print(f'\n\n\nThe class Math marks from the highest to lowest are:\n{class\_math}')  
  
 # If user wants individual student marks  
 elif choice == 3:  
 student = input('\nEnter name of student\n')  
  
 student\_wise\_marks = StudentMarks(master\_data\_file, student.title())  
  
 student\_marks = student\_wise\_marks.marks()  
  
 print(f'\nMarks for {student} are as follows:\n{student\_marks}')  
  
 # If user wants to add/edit student data  
 elif choice == 5:   
 entry\_choice = int(input('\n1)New student entry\n2)Edit existing student data\n'))  
  
 if entry\_choice == 1:  
 more\_entries = True  
  
 # Using while loop to check if user wants to enter multiple student details  
 while more\_entries == True:  
 student = input('\nEnter name of student\n')  
 roll\_entry = int(input(f'\nEnter Roll number of {student.title()}:\n'))  
 gender = input('\nEnter gender:\n')  
 rc\_entry = float(input('\nEnter RC level:\n'))  
 lis\_entry = float(input('\nEnter Listening level:\n'))  
 wri\_entry = float(input('\nEnter Writing level:\n'))  
 math\_entry = float(input('\nEnter Math level:\n'))  
  
 enter\_data = DataEntry(d\_loc, f\_name, master\_data\_file, student.title(), gender.capitalize(), roll\_entry, rc\_entry, lis\_entry, wri\_entry, math\_entry)  
  
 appending\_data = enter\_data.data\_entry()  
   
 print(f'\nThe updated data file is shown below:\n{appending\_data}\n')  
  
 entries = input('\nWould you like to add another entry?\n1) Yes\n2) No\n')  
  
 if entries == '1' or entries.upper() == 'Y' or entries.upper() == 'YES':  
 more\_entries = True  
  
 else:  
 more\_entries = False  
 print('\nExiting...')  
  
 elif entry\_choice == 2:  
 student\_number = int(input('\nEnter the roll number of the student to begin editing\n'))  
 name\_entry = input(f'Enter new details at roll number {student\_number}\n\nEnter name of student:\n')  
   
 student\_number -= 1  
 name\_entry = name\_entry.title()  
   
 gender = input('\nEnter gender:\n')  
 rc\_entry = float(input('\nEnter RC level:\n'))  
 lis\_entry = float(input('\nEnter Listening level:\n'))  
 wri\_entry = float(input('\nEnter Writing level:\n'))  
 math\_entry = float(input('\nEnter Math level:\n'))  
   
 enter\_data = DataEntry(d\_loc, f\_name, master\_data\_file, name\_entry.title(), gender.capitalize(), student\_number, rc\_entry, lis\_entry, wri\_entry, math\_entry)  
  
 edit\_data = enter\_data.data\_edit()  
   
 print(f'\nUpdated file is shown below:\n{edit\_data}\n\nUpdate saved successfully!\n')  
   
 else:  
 print('Invalid choice!')  
   
 else:  
 print('\nLogin failed!')  
   
   
elif login == 2:  
 user\_id = input('\nCreate a user name: ')  
 user\_id = user\_id.encode()  
 pwd = getpass('Create your password: ')  
 pwd = pwd.encode() # Reference link - https://stackoverflow.com/questions/9202224/getting-command-line-password-input-in-python  
   
 key\_code = Keys  
 code = key\_code.read\_key()  
   
 key = Fernet(code)  
   
 encrypted\_pwd = key.encrypt(pwd)  
 encrypted\_id = key.encrypt(user\_id)  
   
 user\_auth = Auth(encrypted\_id, encrypted\_pwd)  
   
 login\_status = user\_auth.new\_user()  
   
 print(f'\n{login\_status}')  
  
else:  
 print('Invalid choice!')