COMPUTER SCIENCE   
BOARD PROJECT 2020-2021

**BUCKSTRACKER**

Submitted by

Ashwath R  
12-B   
120209

Along with   
A K Venkatakrishnan

12-B 120234

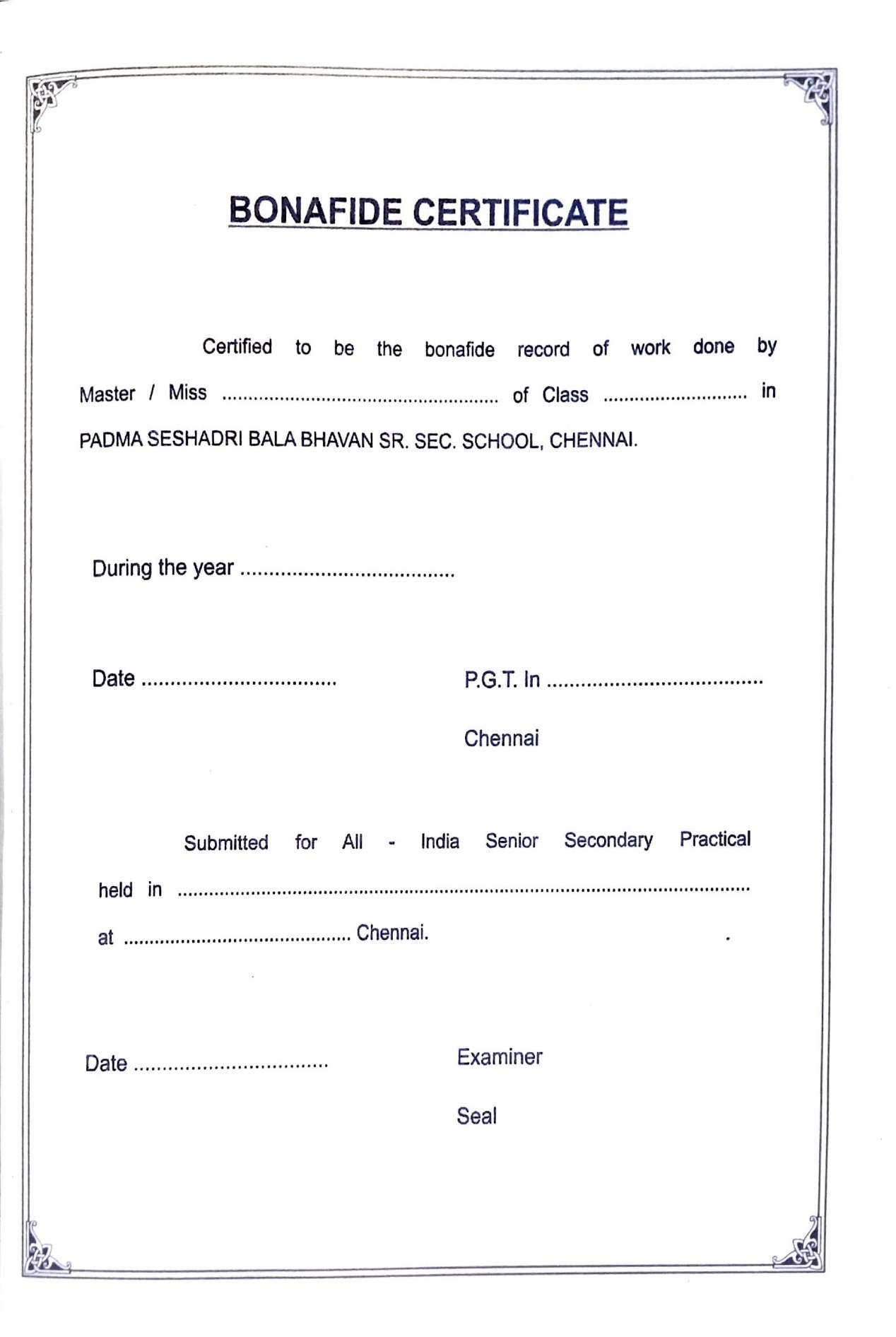
INDEX

|  |  |  |
| --- | --- | --- |
| S.NO | TITLE | PAGE NO. |
| **1.** | Case study |  |
| **2.** | MySQL Table Description |  |
| **3.** | Flowchart |  |
| **4.** | Program Code |  |
| **5.** | Sample Screenshot |  |
| **6.** | Limitations |  |
| **7.** | Scope for Improvement |  |
| **8.** | References |  |

**ACKNOWLEDGEMENT**

I would like to express my deep sense of gratitude towards my school Principal. Mrs. Vasanthi, as well as my computer science teacher Mrs. L.Kala for their valuable guidance and encouragement. I also thank my teachers for giving me this wonderful opportunity to explore new modules and libraries in Python .  
I would also like to thank my parents for their support and help rendered for completion of the project.

A special acknowledgement goes to my classmates who helped me in completing the project by exchanging interesting ideas and sharing their experience.



**CASE STUDY**

The current economic context in the world as well as the country is quite alarming with job losses, salary cuts, etc. impacting the lives of millions of people.

It is therefore imperative that families and individuals take their expenses seriously and start tracking them. Tracking expenses vs the income will help in the following ways:

(1) Maintaining control over finances

(2) Organizing finances

(3) Identifying wasteful expenditure and controlling them

(4) Improving saving potential

(5) Reduced financial stress

(6) Avoidance of debts by reducing impulse buying

(7) Staying focussed on financial goals

Given this background, we were motivated to develop a simple income/ expense tracker, that can be used by common people on a daily basis to keep tab of their personal finances.

The other objective is to present the financial data gathered in a graphical form, which can help in easy interpretation. Users of this application will be able to view category wise spends and this data in turn can be analysed further to identify categories where the spends are high and and come up with ways of reducing these.   
  
There are also graphs which tell you various kinds of information like how much money you have spent per month, how many times you have used a specific payment mode and where you have most of your money(the category).

In short the tracker is an financial tool that can help families improve their economic situation.

**MySQL TABLE DESCRIPTION:**

Created a database called Buckstracker

It contains 3 tables:

1. Login
2. Income
3. Expenditure

mysql> desc login;

This is the schema of the login table

+----------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+-------+

| USERNAME | varchar(30) | NO | PRI | NULL | |

| PASSWORD | varchar(30) | NO | UNI | NULL | |

+----------+-------------+------+-----+---------+-------+

mysql> desc income;

This is the schema of the income table

+-------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+---------------+------+-----+---------+-------+

| SNO | int | NO | PRI | NULL | |

| DATE | date | YES | | NULL | |

| NAME | varchar(30) | NO | | NULL | |

| sal | decimal(10,2) | NO | | NULL | |

| NOTES | varchar(30) | YES | | NULL | |

+-------+---------------+------+-----+---------+-------+

mysql> desc expenditure;

This is the schema of the expenditure table

+----------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+---------------+------+-----+---------+-------+

| SNO | decimal(5,0) | NO | PRI | NULL | |

| DATE | date | NO | | NULL | |

| NAME | varchar(30) | NO | | NULL | |

| CATEGORY | varchar(30) | NO | | NULL | |

| PAYMODE | varchar(30) | NO | | NULL | |

| COST | decimal(10,2) | NO | | NULL | |

| NOTES | varchar(30) | YES | | NULL | |

+----------+---------------+------+-----+---------+-------+

USE OF THE ABOVE TABLES

1. Login

Login is a table storing the username and password of the user. This is essential to check if the user enters password to the corresponding username.

2. Income

This table stores the income details of the user. Each record is each income/salary received by the user with the given date. i.e it will contain the date of credit/salary of the user along with his username, salary of credit amount and notes regarding the credit

3. Expenditure

This table stores details regarding debit/expense transactions of the user. Each record is each debit done by the user. It contains the date of transaction, the category of expenditure made, mode of payment by which transaction was done, the cost/ money debited and the notes regarding this transaction

**PROGRAM CODE**

Contains the following

1 main file : Buckstracker.py

with 2 modules : Graphs.py and treeview\_sql.py

Buckstracker.py:

from tkinter import \*

from tkinter import messagebox

import sys

from Graphs import \*

import mysql.connector as m

from treeview\_sql import \*

c=m.connect(host="localhost",user="root",passwd="mysql",database="buckstracker")

cur=c.cursor()

#Deletion of Account

def deleteaccfn():

global username

delete\_win=Tk()

delete\_win.geometry('350x125')

signdel=Label(delete\_win,text="DELETE ACCOUNT",font=("Verdana",13,"bold"),fg="navy").grid(row=0,column=1,sticky="w")

name=Label(delete\_win,text="Enter username : ",font=("Verdana",10),fg="navy").grid(row=1,column=0)

username=Entry(delete\_win)

username.grid(row=1,column=1)

pswd=Label(delete\_win,text="Enter password : ",font=("Verdana",10),fg="navy").grid(row=2,column=0)

password=Entry(delete\_win,show='\*')

password.grid(row=2,column=1)

#Checking for correct username/password

def checkdel():

a=username.get()

b=password.get()

if a!="" and b!="":

cur.execute("select \* from login where username='"+ a +"'")

data=cur.fetchall()

# Verifying username, then password to delete acc

if len(data)==0:

messagebox.showerror("Notice","You have not entered a registered username")

elif data[0][0]==a and data[0][1]==b:

cur.execute("delete from login where username= '" +a+ "' and password= '" +b+ "'")

messagebox.showinfo("Message","Successfully deleted account")

c.commit()

delete\_win.destroy()

else:

messagebox.showerror("Notice","Your password or username is not entered correctly")

elif a=="" and b=="":

messagebox.showerror("Notice","You have not entered a username or password")

label\_dummy=Label(delete\_win,text="").grid(row=4,column=0)

button\_del=Button(delete\_win,text="DELETE ACCOUNT",font=("Calibre",12,"bold"),fg="navy",command= checkdel)

button\_del.grid(row=5,column=1,sticky="w")

#Login Screen

def loginwindow():

global login,us

login=Tk()

login.geometry('400x175')

login.title("LOGIN SCREEN")

label1=Label(login,text="Do you want to login or signup or delete account ?",font=("Verdana",10)).place(x=50,y=0)

loginwindow()

#Login function

def signinfunc():

global us,signin,login,mainname

#us: String variable for username while mainame is value of us

signin=Tk()

signin.geometry("300x140")

signin.title("SIGNIN SCREEN")

sign=Label(signin,text="LOGIN",font=("Verdana",13,"bold"),fg="indian red").grid(row=0,column=1,sticky="w")

name=Label(signin,text="Enter username : ",font=("Verdana",10),fg="indian red").grid(row=1,column=0)

us=Entry(signin)

us.grid(row=1,column=1)

pswd=Label(signin,text="Enter password : ",font=("Verdana",10),fg="indian red").grid(row=2,column=0)

password=Entry(signin,show='\*')

password.grid(row=2,column=1)

empty=Label(signin,text="").grid(row=3,column=0)

mainname=us.get()

#Checking for correct username/password for login

def checklogin():

a=us.get()

b=password.get()

if a!="" and b!="":

cur.execute("select \* from login where username='"+ a +"'")

data=cur.fetchall()

if len(data)==0:

messagebox.showerror("Notice","You have not entered a registered username")

elif data[0][0]==a and data[0][1]==b:

messagebox.showinfo("Message","Successfully logged into account")

signin.destroy()

loginin(a)

else:

messagebox.showerror("Notice","Your password or username is not entered correctly")

elif a=="" and b=="":

messagebox.showerror("Notice","You have not entered a username or password")

button3=Button(signin,text="SIGN IN",font=("Calibre",12,"bold"),fg="indian red",command=checklogin)

button3.grid(row=4,column=1,sticky="w")

#Sign up function

def signupfunc():

global signup,login

signup=Tk()

signup.geometry("310x160")

signup.title("SIGNUP SCREEN")

sign=Label(signup,text="SIGN UP",font=("Verdana",13,"bold"),fg="steel blue").grid(row=0,column=1,sticky="w")

name=Label(signup,text="Enter username : ",font=("Verdana",10),fg="steel blue").grid(row=1,column=0)

user\_signup=Entry(signup)

user\_signup.grid(row=1,column=1)

pswd=Label(signup,text="Enter password : ",font=("Verdana",10),fg="steel blue").grid(row=2,column=0)

password\_signup=Entry(signup,show='\*')

password\_signup.grid(row=2,column=1)

cpswd=Label(signup,text="Confirm password : ",font=("Verdana",10),fg="steel blue").grid(row=3,column=0)

cpass\_signup=Entry(signup,show="\*")

cpass\_signup.grid(row=3,column=1)

empty=Label(signup,text="").grid(row=4,column=0)

#Adding account to SQL

def uploadsignup():

A=user\_signup.get()

B=password\_signup.get()

C=cpass\_signup.get()

if B==C and C!=None and A!=None:

cur.execute("select \* from login where username='"+A+"'")

d=cur.fetchall()

if len(d)==0:

messagebox.showinfo("Message","Successfully created an account. Please proceed to Login")

cur.execute("insert into login values('"+ A +"','"+ B +"')")

c.commit()

signup.destroy()

else:

messagebox.showerror("Notice","This Username already exists")

else:

messagebox.showerror("Notice","Your password or username is not entered correctly")

button3=Button(signup,text="SIGN UP",font=("Calibre",12,"bold"),fg="steel blue",command=uploadsignup)

button3.grid(row=5,column=1,sticky="w")

# Final buttons performing given functions

button1=Button(login,text=" LOGIN ",font=("Calibre",12,"bold"),fg="blue",command=signinfunc).pack(pady=22)

button2=Button(login,text="SIGNUP",font=("Calibre",12,"bold"),fg="green",command=signupfunc).pack(pady=4)

button3=Button(login,text="DELETE ACCOUNT",font=("Calibre",12,"bold"),fg="red",command=deleteaccfn).pack(pady=15)

#Information Options : Graph Reports or Income or Expense Data or Logout

def loginin(mainname\_1):

global signin,signup,login

root=Toplevel()

root.geometry('450x100')

root.title("Buckstracker")

label=Label(root,text="Buckstracker",anchor=CENTER,font=("Calibre",15,"bold"),fg="magenta3").grid(row=0,column=2)

b1=Button(root,text="Income",anchor=CENTER,font=("Calibre",12,"bold"),fg="green",command= lambda: Income\_Tree(mainname\_1))

b1.grid(row=1,column=1)

b2=Button(root,text="Expense",anchor=CENTER,font=("Calibre",12,"bold"),fg="red",command= lambda: Expense\_Tree(mainname\_1))

b2.grid(row=1,column=2)

b3=Button(root,text="Graph Reports",anchor=CENTER,font=("Calibre",12,"bold"),fg="blue",command= lambda : graphreports(mainname\_1))

b3.grid(row=1,column=3)

def logout(root,win):

root.destroy()

# function to logout and open Buckstracker Login window if it had been closed previously

if win.state()!="normal":

loginwindow()

Label\_Space=Label(root,text=" ")

Label\_Space.grid(row=1,column=4)

b4=Button(root,text="LOGOUT",anchor=CENTER,font=("Calibre",12,"bold"),fg="navy",command= lambda: logout(root,login))

b4.grid(row=1,column=5)

root.mainloop()

login.mainloop()

treeview\_sql.py:

from tkinter import \*

from tkinter import ttk

import mysql.connector as m

from tkinter import messagebox

from tkinter import ttk

conn=m.connect(host="localhost",user="root",passwd="mysql",database="buckstracker")

cur=conn.cursor()

Tree1\_count=0

Tree2\_count=0

def Income\_Tree(Tree1\_Username):

#Income Tree

Tree1=Tk()

Tree1.title("Income Tree View")

Tree1.geometry("720x675")

#Tree view Scroll frame

Tree1\_ScrollFrame=Frame(Tree1)

Tree1\_ScrollFrame.pack(pady=20)

#Tree View Scrollbar

Tree1\_Scroll=Scrollbar(Tree1\_ScrollFrame)

Tree1\_Scroll.pack(side=RIGHT,fill=Y)

inc\_tree=ttk.Treeview(Tree1\_ScrollFrame,yscrollcommand=Tree1\_Scroll.set,selectmode="browse")

Tree1\_Scroll.config(command=inc\_tree.yview)

inc\_tree['columns']=("SNo","Date","Name","Salary","Notes")

#Column formating

inc\_tree.column("#0",width=0,stretch=NO)

inc\_tree.column("SNo",anchor=CENTER,width=35,minwidth=20)

inc\_tree.column("Date",anchor=W,width=120,minwidth=20)

inc\_tree.column("Name",anchor=W,width=120,minwidth=20)

inc\_tree.column("Salary",anchor=W,width=120,minwidth=20)

inc\_tree.column("Notes",anchor=W,width=120,minwidth=20)

#Column Headings

inc\_tree.heading("#0",text="", anchor=W)

inc\_tree.heading("SNo",text="SNo", anchor=CENTER)

inc\_tree.heading("Date",text="Date of Income(YYYY-MM-DD)", anchor=W)

inc\_tree.heading("Name",text="Name", anchor=W)

inc\_tree.heading("Salary",text="Salary", anchor=W)

inc\_tree.heading("Notes",text="Extra Info", anchor=W)

inc\_tree.pack()

#Frame Creation MAIN PART

Tree1\_frame = Frame(Tree1)

Tree1\_frame.pack(pady=20)

#Labels

Tree1\_l0=Label(Tree1\_frame,text="SNo")

Tree1\_l0.grid(row=0,column=0)

Tree1\_l1=Label(Tree1\_frame,text="Date of Income(YYYY-MM-DD)")

Tree1\_l1.grid(row=0,column=1)

Tree1\_l2=Label(Tree1\_frame,text="Name")

Tree1\_l2.grid(row=0,column=2)

Tree1\_l3=Label(Tree1\_frame,text="Salary")

Tree1\_l3.grid(row=0,column=3)

Tree1\_l4=Label(Tree1\_frame,text="Extra Info")

Tree1\_l4.grid(row=0,column=4)

#Entry Box

Tree1\_e1=Entry(Tree1\_frame)

Tree1\_e1.grid(row=1,column=1)

Tree1\_e2=Label(Tree1\_frame,text=Tree1\_Username)

Tree1\_e2.grid(row=1,column=2)

Tree1\_e3=Entry(Tree1\_frame)

Tree1\_e3.grid(row=1,column=3)

Tree1\_e4=Entry(Tree1\_frame)

Tree1\_e4.grid(row=1,column=4)

#Showing Records

def Tree1\_Show():

global Tree1\_count

cur.execute("select \* from income where name='"+Tree1\_Username+"'")

Tree1\_data=cur.fetchall()

for x in Tree1\_data:

inc\_tree.insert(parent='',index='end',iid=Tree1\_count,text="",values=(x[0],x[1],x[2],x[3],x[4]))

Tree1\_count+=1

Tree1\_b5=Button(Tree1,text="Show existing table",command=Tree1\_Show)

Tree1\_b5.pack(pady=10)

#Showing SNo Req

def Tree1\_show\_sno():

global Tree1\_sno,Tree1\_f0

Tree1\_f0=Frame(Tree1\_frame)

Tree1\_f0.grid(row=1,column=0)

q\_sno="select max(sno)+1 from income"

cur.execute(q\_sno)

sno=cur.fetchall()

Tree1\_sno=str(sno[0][0])

sno\_Label=Label(Tree1\_f0,text=Tree1\_sno).pack()

Tree1\_b1\_1=Button(Tree1,text="Display SNo for new record",command=Tree1\_show\_sno)

Tree1\_b1\_1.pack(pady=20)

#Adding record

def add\_Tree1():

global Tree1\_count

Tree1\_q2="insert into income values('"+Tree1\_sno+"','"+Tree1\_e1.get()+"','"+Tree1\_Username+"','"+Tree1\_e3.get()+"','"+Tree1\_e4.get()+"')"

cur.execute(Tree1\_q2)

conn.commit()

inc\_tree.insert(parent='',index='end',iid=Tree1\_count,text="",values=(int(Tree1\_sno),Tree1\_e1.get(),Tree1\_Username,Tree1\_e3.get(),Tree1\_e4.get()))

Tree1\_count+=1

#Clearing data in the box and in frame Tree1\_f0

Tree1\_e1.delete(0,END)

Tree1\_e3.delete(0,END)

Tree1\_e4.delete(0,END)

Tree1\_f0.destroy()

Tree1\_b1=Button(Tree1,text="Add record",command=add\_Tree1)

Tree1\_b1.pack(pady=20)

#Select Record

def select\_Tree1():

#Clear entry boxes

Tree1\_e1.delete(0,END)

Tree1\_e3.delete(0,END)

Tree1\_e4.delete(0,END)

#Getting Record Number

Tree1\_selected=inc\_tree.focus()

#Getting the values

Tree1\_values=inc\_tree.item(Tree1\_selected,'values')

#Output to Entry boxes

Tree1\_e1.insert(0,Tree1\_values[1])

Tree1\_e3.insert(0,Tree1\_values[3])

Tree1\_e4.insert(0,Tree1\_values[4])

Tree1\_b2=Button(Tree1,text="Select a record",command=select\_Tree1)

Tree1\_b2.pack(pady=10)

#Update Record

def update\_Tree1():

#Getting Record Number

Tree1\_selected=inc\_tree.focus()

#Getting the values

Tree1\_values=inc\_tree.item(Tree1\_selected,'values')

#Save the item

Tree1\_q5="update income set Sno='"+Tree1\_values[0]+"', date='"+Tree1\_e1.get()+"', name = '"+Tree1\_Username+"', sal='"+Tree1\_e3.get()+"',notes='"+Tree1\_e4.get()+"'where sno='"+Tree1\_values[0]+"'"

cur.execute(Tree1\_q5)

conn.commit()

inc\_tree.item(Tree1\_selected,text='',values=(Tree1\_values[0],Tree1\_e1.get(),Tree1\_Username,Tree1\_e3.get(),Tree1\_e4.get()))

#Clear entry boxes

Tree1\_e1.delete(0,END)

Tree1\_e3.delete(0,END)

Tree1\_e4.delete(0,END)

Tree1\_b3=Button(Tree1,text="Update record",command=update\_Tree1)

Tree1\_b3.pack(pady=10)

#Remove 1 record

def remove1\_Tree1():

#Getting Record Number

Tree1\_selected=inc\_tree.focus()

#Getting the values

Tree1\_values=inc\_tree.item(Tree1\_selected,'values')

# Deletion in SQL

Tree1\_q3="delete from income where sno='"+Tree1\_values[0]+"'"

cur.execute(Tree1\_q3)

conn.commit()

Tree1\_q4="update income set sno=sno-1 where sno>= '"+Tree1\_values[0]+"'"

cur.execute(Tree1\_q4)

conn.commit()

Tree1\_d1=inc\_tree.selection()

inc\_tree.delete(Tree1\_d1)

messagebox.showinfo("Refrsh","Refreshing data in the viewing table")

for record in inc\_tree.get\_children():

inc\_tree.delete(record)

Tree1\_Show()

Tree1\_b4=Button(Tree1,text="Remove selected record",command=remove1\_Tree1)

Tree1\_b4.pack(pady=10)

Tree1.mainloop()

def Expense\_Tree(Tree2\_Username):

#Expense Tree

Tree2=Tk()

Tree2.title("Expense Tree View")

Tree2.geometry("975x675")

#Tree view Scroll frame

Tree2\_ScrollFrame=Frame(Tree2)

Tree2\_ScrollFrame.pack(pady=20)

#Tree View Scrollbar

Tree2\_Scroll=Scrollbar(Tree2\_ScrollFrame)

Tree2\_Scroll.pack(side=RIGHT,fill=Y)

exp\_tree=ttk.Treeview(Tree2\_ScrollFrame,yscrollcommand=Tree2\_Scroll.set,selectmode="browse")

Tree2\_Scroll.config(command=exp\_tree.yview)

#Column Variables Required

exp\_tree['columns']=("SNo","Date","Name","Category","Payment Mode","Cost","Notes")

#Column formating

exp\_tree.column("#0",width=0,stretch=NO)

exp\_tree.column("SNo",anchor=CENTER,width=35,minwidth=20)

exp\_tree.column("Date",anchor=W,width=120,minwidth=20)

exp\_tree.column("Name",anchor=W,width=120,minwidth=20)

exp\_tree.column("Category",anchor=W,width=120,minwidth=20)

exp\_tree.column("Payment Mode",anchor=W,width=120,minwidth=20)

exp\_tree.column("Cost",anchor=W,width=120,minwidth=20)

exp\_tree.column("Notes",anchor=W,width=120,minwidth=20)

#Column Headings

exp\_tree.heading("#0",text="", anchor=W)

exp\_tree.heading("SNo",text="SNo", anchor=CENTER)

exp\_tree.heading("Date",text="Date of Exp(YYYY-MM-DD)", anchor=W)

exp\_tree.heading("Name",text="Name", anchor=W)

exp\_tree.heading("Category",text="Category", anchor=W)

exp\_tree.heading("Payment Mode",text="Payment Mode", anchor=W)

exp\_tree.heading("Cost",text="Cost",anchor=W)

exp\_tree.heading("Notes",text="Extra Info", anchor=W)

exp\_tree.pack()

#Frame Creation MAIN PART

Tree2\_frame = Frame(Tree2)

Tree2\_frame.pack(pady=20)

#Labels

Tree2\_l0=Label(Tree2\_frame,text="SNo")

Tree2\_l0.grid(row=0,column=0)

Tree2\_l1=Label(Tree2\_frame,text="Date of Exp(YYYY-MM-DD)")

Tree2\_l1.grid(row=0,column=1)

Tree2\_l2=Label(Tree2\_frame,text="Name")

Tree2\_l2.grid(row=0,column=2)

Tree2\_l3=Label(Tree2\_frame,text="Category")

Tree2\_l3.grid(row=0,column=3)

Tree2\_l4=Label(Tree2\_frame,text="Payment Mode")

Tree2\_l4.grid(row=0,column=4)

Tree2\_l5=Label(Tree2\_frame,text="Cost")

Tree2\_l5.grid(row=0,column=5)

Tree2\_l6=Label(Tree2\_frame,text="Extra Info")

Tree2\_l6.grid(row=0,column=6)

#Entry Box

Tree2\_e1=Entry(Tree2\_frame)

Tree2\_e1.grid(row=1,column=1)

Tree2\_e2=Label(Tree2\_frame,text=Tree2\_Username)

Tree2\_e2.grid(row=1,column=2)

Tree2\_e3=StringVar()

Tree2\_e3.set("Select a category")

Tree2\_Option1=["Provisions","Medicine","Taxes","Travel","Education","Miscellaneous"]

Tree2\_Menu=OptionMenu(Tree2\_frame,Tree2\_e3,\*Tree2\_Option1)

Tree2\_Menu.grid(row=1,column=3)

Tree2\_e4=StringVar()

Tree2\_e4.set("Select payment mode")

Tree2\_Option2=['Cash','Credit Card','Netbanking','Cheque','BHIM UPI','PayTM','PhonePe','Samsung Pay','Airtel Money']

Tree2\_Menu=OptionMenu(Tree2\_frame,Tree2\_e4,\*Tree2\_Option2)

Tree2\_Menu.grid(row=1,column=4)

Tree2\_e5=Entry(Tree2\_frame)

Tree2\_e5.grid(row=1,column=5)

Tree2\_e6=Entry(Tree2\_frame)

Tree2\_e6.grid(row=1,column=6)

#Showing Records

def Tree2\_Show():

global Tree2\_count

cur.execute("select \* from expenditure where name='"+Tree2\_Username+"'")

Tree2\_data=cur.fetchall()

for x in Tree2\_data:

exp\_tree.insert(parent='',index='end',iid=Tree2\_count,text="",values=(x[0],x[1],x[2],x[3],x[4],x[5],x[6]))

Tree2\_count+=1

Tree2\_b5=Button(Tree2,text="Show existing table",command=Tree2\_Show)

Tree2\_b5.pack(pady=10)

#Showing Req SNo

def Tree2\_show\_sno():

global Tree2\_sno,Tree2\_f0

#Frame for Sno

Tree2\_f0=Frame(Tree2\_frame)

Tree2\_f0.grid(row=1,column=0)

q\_sno="select max(sno)+1 from expenditure"

cur.execute(q\_sno)

sno\_2=cur.fetchall()

Tree2\_sno=str(sno\_2[0][0])

sno\_Label=Label(Tree2\_f0,text=Tree2\_sno).pack()

Tree2\_b2\_1=Button(Tree2,text="Display SNo for new record",command=Tree2\_show\_sno)

Tree2\_b2\_1.pack(pady=20)

#Adding record

def add\_Tree2():

global Tree2\_count

Tree2\_q2="insert into expenditure values('"+Tree2\_sno+"','"+Tree2\_e1.get()+"','"+Tree2\_Username+"','"+Tree2\_e3.get()+"','"+Tree2\_e4.get()+"','"+Tree2\_e5.get()+"','"+Tree2\_e6.get()+"')"

cur.execute(Tree2\_q2)

conn.commit()

exp\_tree.insert(parent='',index='end',iid=Tree2\_count,text="",values=(Tree2\_sno,Tree2\_e1.get(),Tree2\_Username,Tree2\_e3.get(),Tree2\_e4.get(),Tree2\_e5.get(),Tree2\_e6.get()))

Tree2\_count+=1

Tree2\_e1.delete(0,END)

Tree2\_e5.delete(0,END)

Tree2\_e6.delete(0,END)

Tree2\_f0.destroy()

Tree2\_b1=Button(Tree2,text="Add record",command=add\_Tree2)

Tree2\_b1.pack(pady=20)

#Select Record

def select\_Tree2():

#Clear entry boxes

Tree2\_e1.delete(0,END)

Tree2\_e5.delete(0,END)

Tree2\_e6.delete(0,END)

#Getting Record Number

Tree2\_selected=exp\_tree.focus()

#Getting the values

Tree2\_values=exp\_tree.item(Tree2\_selected,'values')

#Output to Entry boxes

Tree2\_e1.insert(0,Tree2\_values[1])

Tree2\_e3.set(Tree2\_values[3])

Tree2\_e4.set(Tree2\_values[4])

Tree2\_e5.insert(0,Tree2\_values[5])

Tree2\_e6.insert(0,Tree2\_values[6])

Tree2\_b2=Button(Tree2,text="Select a record",command=select\_Tree2)

Tree2\_b2.pack(pady=10)

#Update Record

def update\_Tree2():

#Getting Record Number

Tree2\_selected=exp\_tree.focus()

#Getting the values

Tree2\_values=exp\_tree.item(Tree2\_selected,'values')

Tree2\_q5="update expenditure set Sno='"+Tree2\_values[0]+"', date='"+Tree2\_e1.get()+"', name='"+Tree2\_Username+"', category='"+Tree2\_e3.get()+"', paymode='"+Tree2\_e4.get()+"', cost='"+Tree2\_e5.get()+"', notes='"+Tree2\_e6.get()+"'where sno='"+Tree2\_values[0]+"'"

cur.execute(Tree2\_q5)

conn.commit()

exp\_tree.item(Tree2\_selected,text='',values=(Tree2\_values[0],Tree2\_e1.get(),Tree2\_Username,Tree2\_e3.get(),Tree2\_e4.get(),Tree2\_e5.get(),Tree2\_e6.get()))

#Clear entry boxes

Tree2\_e1.delete(0,END)

Tree2\_e5.delete(0,END)

Tree2\_e6.delete(0,END)

Tree2\_b3=Button(Tree2,text="Update record",command=update\_Tree2)

Tree2\_b3.pack(pady=10)

#Remove 1 record

def remove1\_Tree2():

#Getting Record Number

Tree2\_selected=exp\_tree.focus()

#Getting the values

Tree2\_values=exp\_tree.item(Tree2\_selected,'values')

Tree2\_d1=exp\_tree.selection()

exp\_tree.delete(Tree2\_d1)

# Deletion in SQL

Tree2\_q3="delete from expenditure where sno= '"+Tree2\_values[0]+"'"

cur.execute(Tree2\_q3)

conn.commit()

Tree2\_q4="update expenditure set sno=sno-1 where sno>= '"+Tree2\_values[0]+"'"

cur.execute(Tree2\_q4)

conn.commit()

messagebox.showinfo("Refresh","Refreshing data of viewing Table")

for record in exp\_tree.get\_children():

exp\_tree.delete(record)

Tree2\_Show()

Tree2\_b4=Button(Tree2,text="Remove selected record",command=remove1\_Tree2)

Tree2\_b4.pack(pady=10)

Tree2.mainloop()

Graphs.py

from tkinter import \*

from tkinter import messagebox,ttk

from matplotlib import pyplot as plt

from pymysql import \*

from random import \*

from datetime import \*

# Function for Checking the min value of categories in piechart

def explodepie(slices\_1):

cat\_min=min(slices\_1)

ind\_min=slices\_1.index(cat\_min)

t=()

for e in range(len(slices\_1)):

if e==ind\_min:

t=t+(0.1,)

else:

t=t+(0,)

return t

# Main function for this module

def graphreports(name\_1):

global enter\_year,enter\_month,options,options\_var,cursor,username,frame\_window

#enter\_month,enter\_year is user input of month,yr respectively

#options is the different graphs

# options\_var is tkinter variable for Menu

#frame\_window is extra pop up frame to input yr and month

username=name\_1

root=Tk()

root.title("Graph Reports")

root.geometry("500x250")

conn=connect(host='localhost',user='root',passwd='mysql',database='buckstracker')

cursor=conn.cursor()

options=[

'Expenditure in different categories of a month as Piechart(use when month has required data)',#Piechart

'Expenditure in different categories of a year as Piechart' ,#Piechart

'Income levels in a year as bar graph and line graph',#Bar graph or line graph

'Frequency of Payment mode in a year as Pie chart', # Pie chart

'Frequency payment mode in a month as Bar graph', # Horizontal Bar graph

'Frequency of expenditure in a month over a year as Scatter Plot',#Scatter plot

'Comparison of income and expenditure in a year over various months', # Bar graph

]

# Inserting above details into tkinter window

LABEL=Label(root,text="Enter the graph required from below")

LABEL.pack()

options\_var=StringVar()

graphopt\_menu=OptionMenu(root,options\_var,\*options)

graphopt\_menu.config(width=200)

graphopt\_menu.pack()

#This function is to display required month or year options depending upon selected graph option

def otheropt(root\_window,var\_graphopt,conn\_1,cursor\_1,opt):

global enter\_month,enter\_year,graphopt\_menu,month\_dropmenu,year\_dropmenu,frame\_window

# var\_graphopt is the Tkinter variable of Graph Option menu

frame\_window=Frame(root\_window)

frame\_window.pack(pady=20)

resp\_1=var\_graphopt.get()

enter\_month=IntVar()

enter\_year=IntVar()

enter\_month.set(0)

enter\_year.set(2010)

e1=[no1 for no1 in range(1,13)]

e2=[no2 for no2 in range(2010,2031)]

label\_1=Label(frame\_window,text="Enter the month and year required ")

label\_2=Label(frame\_window,text="Enter year required")

month\_dropmenu=OptionMenu(frame\_window,enter\_month,\*e1)

if resp\_1==opt[0] or resp\_1==opt[4]:

month\_dropmenu.pack()

label\_1.pack()

else:

label\_2.pack()

year\_dropmenu=OptionMenu(frame\_window,enter\_year,\*e2)

year\_dropmenu.pack()

# Checking for availability of data

s3=''

try:

cursor\_1.execute("select year(date) from expenditure group by year(date) having count(\*)>=10 ")

conn\_1.commit()

yr=cursor\_1.fetchall()

for i in yr:

s3+=str(i[0])+','

except:

pass

if s3=='':

s3="NONE"

s1="Note that we have constructive data for year"

s2=" so please choose accordingly"

lab=Label(frame\_window,text=s1+s3+s2)

lab.pack()

# Function which displays graphs based on selected option

def graph(opt,var\_1,c,cur,name,window,enter\_2,enter\_1=0):

# name is username, c is the mysql connection, cur is the cursor,var\_1 is the graph options

window.destroy()

response\_graphopt=var\_1.get()

month\_input=enter\_1.get()

year\_input=enter\_2.get()

# Expenditure in different categories of a month as Piechart

if response\_graphopt==opt[0]:

slices=[]

category=[]

query= "select sum(cost),category from expenditure where month(date) = '"+ str(month\_input)+"' and year(date) = '" + str(year\_input) + "'and name= '" +name+ "'group by category , month(date)"

cur.execute(query)

c.commit()

data=cur.fetchall()

sum=0

if data!=():

for i in data:

slices.append(i[0])

category.append(i[1])

sum+=i[0]

finalcol=['blue','cyan','green','yellow','red','orange','brown','magenta','white']

finalexp=explodepie(slices)

# Creating Piechart

plt.pie(slices,labels=category,colors=finalcol,startangle=0,shadow=True,radius=1.0,autopct='%1.1f%%',explode=finalexp)

plt.title("Expenditure over different categories over a month with total expenditure = "+str(sum))

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

#Expenditure in different categories of a year as Piechart

elif response\_graphopt==opt[1]:

slices=[]

category=[]

query="select sum(cost), category from expenditure where year(date) = '"+ str(year\_input)+ "'and name = '"+name+"'group by category,year(date)"

cur.execute(query)

c.commit()

data=cur.fetchall()

sum=0

if data!=():

for i in data:

slices.append(i[0])

category.append(i[1])

sum+=i[0]

finalcol=['blue','cyan','green','yellow','red','orange','brown','magenta','white']

finalexp=explodepie(slices)

plt.pie(slices,labels=category,colors=finalcol,startangle=0,shadow=True,radius=1.0,autopct='%1.1f%%',explode=finalexp)

plt.title("Expenditure over different categories over a year with total expenditure= "+str(sum))

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

#Income levels in a year as bar graph and line graph

elif response\_graphopt==opt[2]:

query="select sum(sal),month(date) from income where year(date)= '"+str(year\_input)+ "' and name = '"+name+"' group by month(date) order by month(date)"

cur.execute(query)

c.commit()

data=cur.fetchall()

x=['January','February','March','April','May','June','July','August','September','October','November','December']

sal=[]

if data!=():

for i in data:

sal.append(i[0])

plt.bar(x,sal,color='green',label='Income per month')

plt.plot(x,sal,color='black',label='Income trend')

plt.legend()

plt.title("Monthly income of "+str(year\_input))

powr= str(len(str(data[0][0]//10)))

plt.xlabel("Months")

plt.ylabel("Monthly income in the power of "+powr)

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

#Frequency of Payment mode in a year as Pie chart

elif response\_graphopt==opt[3]:

slices=[]

category=[]

query="select count(paymode),paymode from expenditure where year(date) = '"+ str(year\_input) + "' and name = '" +name+ "'group by paymode,year(date)"

cur.execute(query)

c.commit()

data=cur.fetchall()

sum1=0

for i in data:

slices.append(i[0])

category.append(i[1])

sum1+=i[0]

if data!=():

finalcol=['blue','cyan','green','yellow','red','orange','brown','magenta','white']

finalexp=explodepie(slices)

plt.pie(slices,labels=category,colors=finalcol,startangle=0,shadow=True,radius=1.0,autopct='%1.1f%%',explode=finalexp)

plt.title("Expenditure by payment modes over a year of total="+str(sum1))

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

# Frequency payment mode in a month as Bar graph

elif response\_graphopt==opt[4]:

l=['BHIM UPI','PayTM','Airtel Money','PhonePe','Samsung Pay','Cash','Credit Card',

'Netbanking','Cheque']

x=[]

y=[]

query="select count(paymode) , paymode from expenditure where month(date)= '" + str(month\_input) + "' and year(date)= '" + str(year\_input) + "' and name = '" +name+ "'group by paymode, month(date) order by paymode asc"

cur.execute(query)

c.commit()

data=cur.fetchall()

sum=0

for i in data:

y.append(i[0])

x.append(i[1])

sum+=i[0]

if data!=():

plt.barh(x,y,color='red',label='Payment modes')

plt.xlabel('Payment Modes')

plt.ylabel('Number of times of usage')

plt.title("Frequency of diffent payment modes from total="+str(sum))

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

#Variation of frequency of expenditure in a month over a year as Scatter Plot

elif response\_graphopt==opt[5]:

query="select sum(cost),week(date),month(date), category from expenditure where year(date)='" +str(year\_input)+ "' and name = '" +name+ "' group by category,month(date),week(date);"

cur.execute(query)

c.commit()

d1=cur.fetchall()

cat1x=[]

cat1y=[]

cat2x=[]

cat2y=[]

cat3x=[]

cat3y=[]

cat4x=[]

cat4y=[]

cat5x=[]

cat5y=[]

cat6x=[]

cat6y=[]

sum=0

for i in d1:

sum=sum+i[0]

date\_2=date(int(year\_input),int(i[2]),1)

# Calculating week no of 1st of every month in Python

w1=date.isocalendar(date\_2)

# Calculating week no from beginning of month to check variation of payment in a month Since mysql counts weeks bit differently must add 2

week=int(i[1])-w1[1]+2

if week==6:

week=5

if i[3].lower()=='Provisions'.lower():

cat1x.append(week)

cat1y.append(i[0])

elif i[3].lower()=='Medicine'.lower():

cat2x.append(week)

cat2y.append(i[0])

elif i[3].lower()=='Taxes'.lower():

cat3x.append(week)

cat3y.append(i[0])

elif i[3].lower()=='Travel'.lower():

cat4x.append(week)

cat4y.append(i[0])

elif i[3].lower()=='Education'.lower():

cat5x.append(week)

cat5y.append(i[0])

elif i[3].lower()=='Miscellaneous'.lower():

cat6x.append(week)

cat6y.append(i[0])

if d1!=():

plt.title("Frequency of expenditure in a category per week of total="+str(sum))

plt.scatter(cat1x,cat1y,label="Provisions",color='green')

plt.scatter(cat2x,cat2y,label="Medicine",color='cyan')

plt.scatter(cat3x,cat3y,label="Taxes",color='red')

plt.scatter(cat4x,cat4y,label="Travel",color='blue')

plt.scatter(cat5x,cat5y,label="Education",color='black')

plt.scatter(cat6x,cat6y,label="Miscellaneous",color='orange')

plt.legend()

plt.xlabel("Weeks of Month")

plt.ylabel("Expenditure in Rupees")

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

#Comparison of income and expenditure in a year over various months()

elif response\_graphopt==opt[6]:

query1= "select sum(sal),month(date) from income where year(date)='" +str(year\_input)+ "' and name = '" +name+ "' group by month(date)"

query2="select sum(cost),month(date) from expenditure where year(date)='" +str(year\_input)+ "' and name = '" +name+ "' group by month(date)"

cur.execute(query1)

c.commit()

data1=cur.fetchall()

cur.execute(query2)

c.commit()

data2=cur.fetchall()

x\_axis=[]

y\_axis1=[]

y\_axis2=[]

sum1=sum2=0

if data1!=() and data2!=():

for i in data1:

x\_axis.append(i[1])

y\_axis1.append(i[0])

sum1+=i[0]

for j in data2:

y\_axis2.append(j[0])

sum2+=j[0]

plt.bar(x\_axis,y\_axis1,color='red',label='Income in month of total ='+str(sum1))

plt.bar(x\_axis,y\_axis2,color='blue',label='Expenditure in month of total ='+str(sum2))

plt.title("Income and expenditure comparison")

plt.legend()

plt.xlabel("Month No")

plt.ylabel("Money in Rupees")

plt.show()

else:

messagebox.showerror("ERROR","There is no data in the given year/month")

submit\_b1=Button(frame\_window,text='Submit',command= lambda: graph(options,var\_graphopt,conn,cursor,username,frame\_window,enter\_year,enter\_month) )

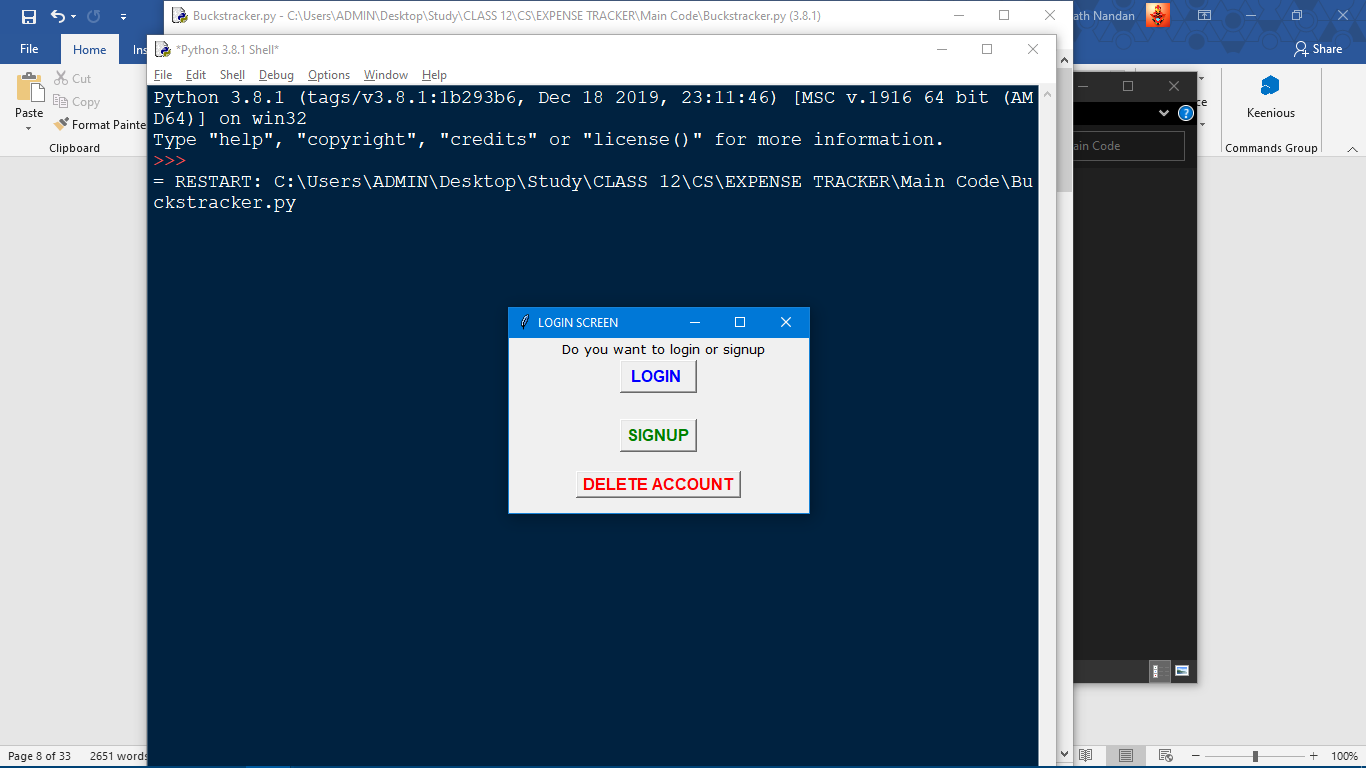
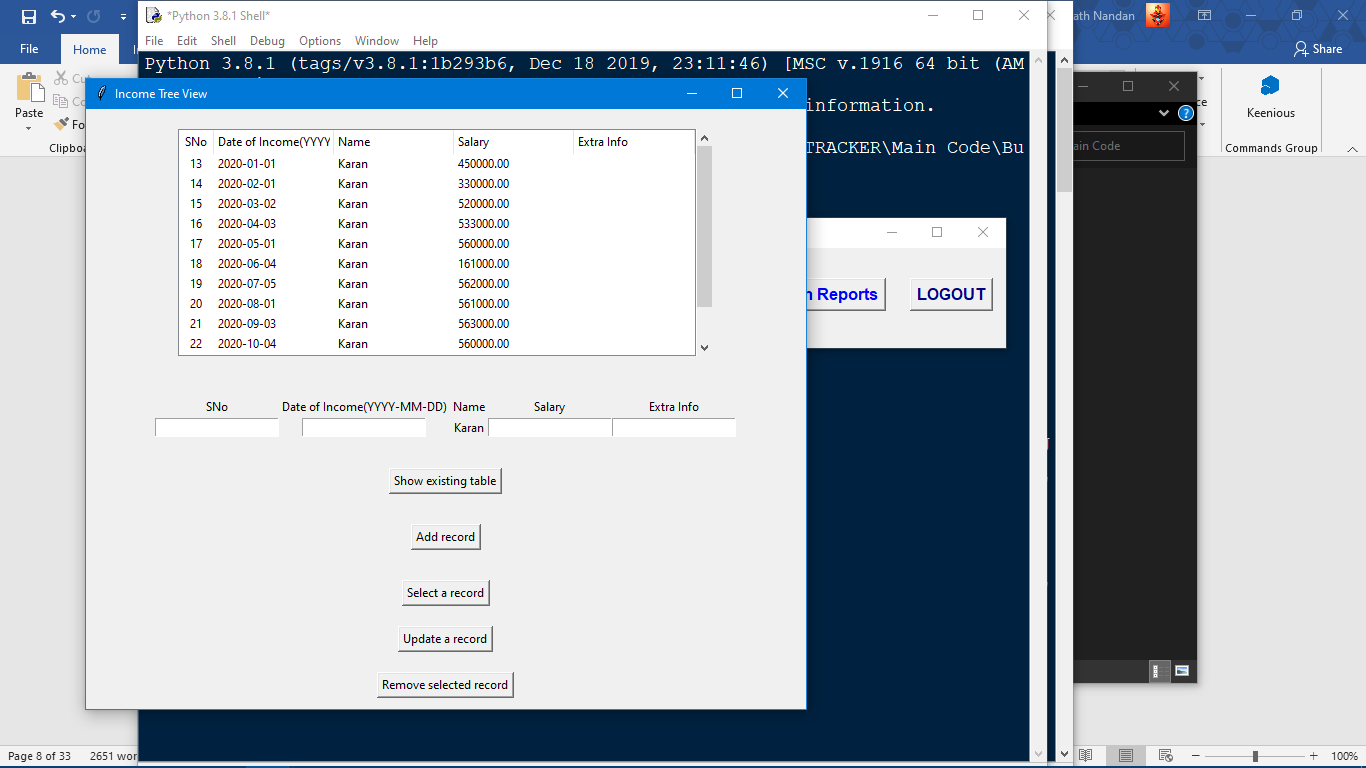
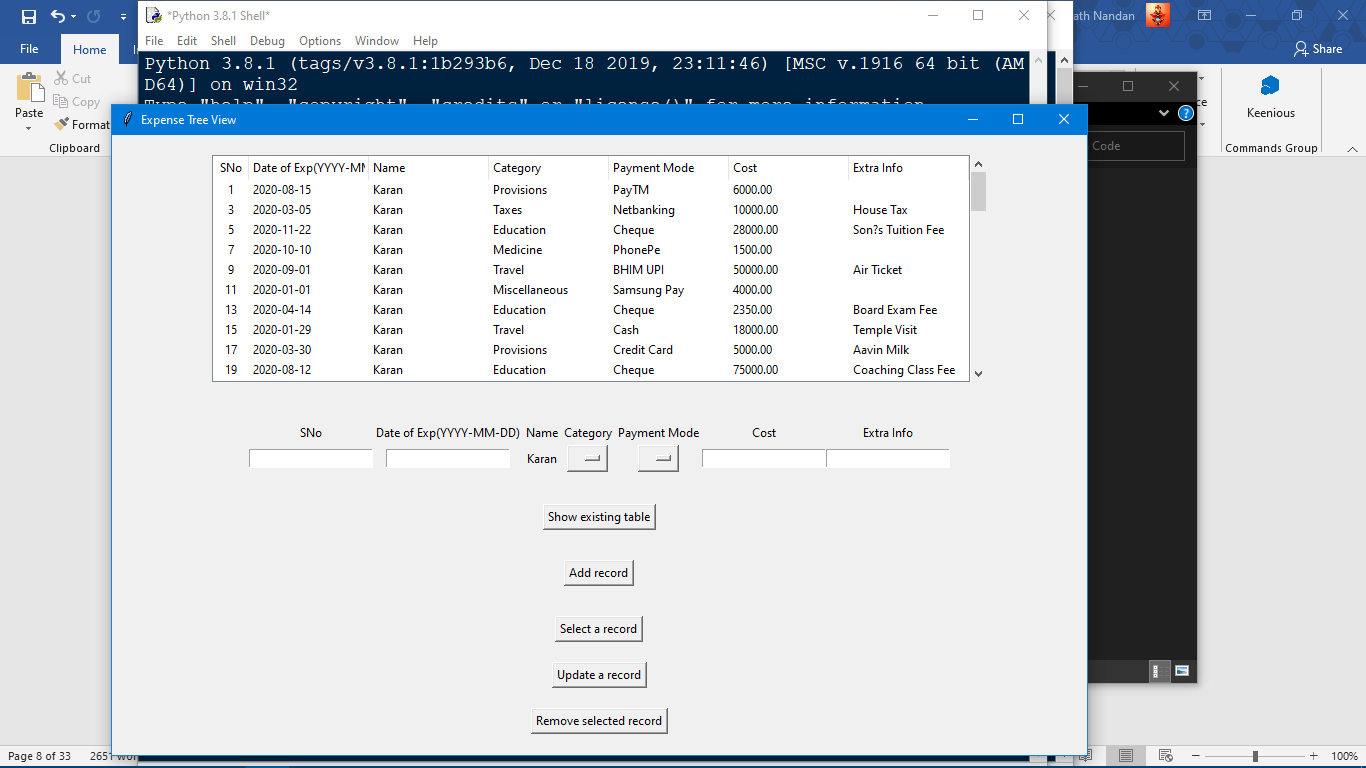
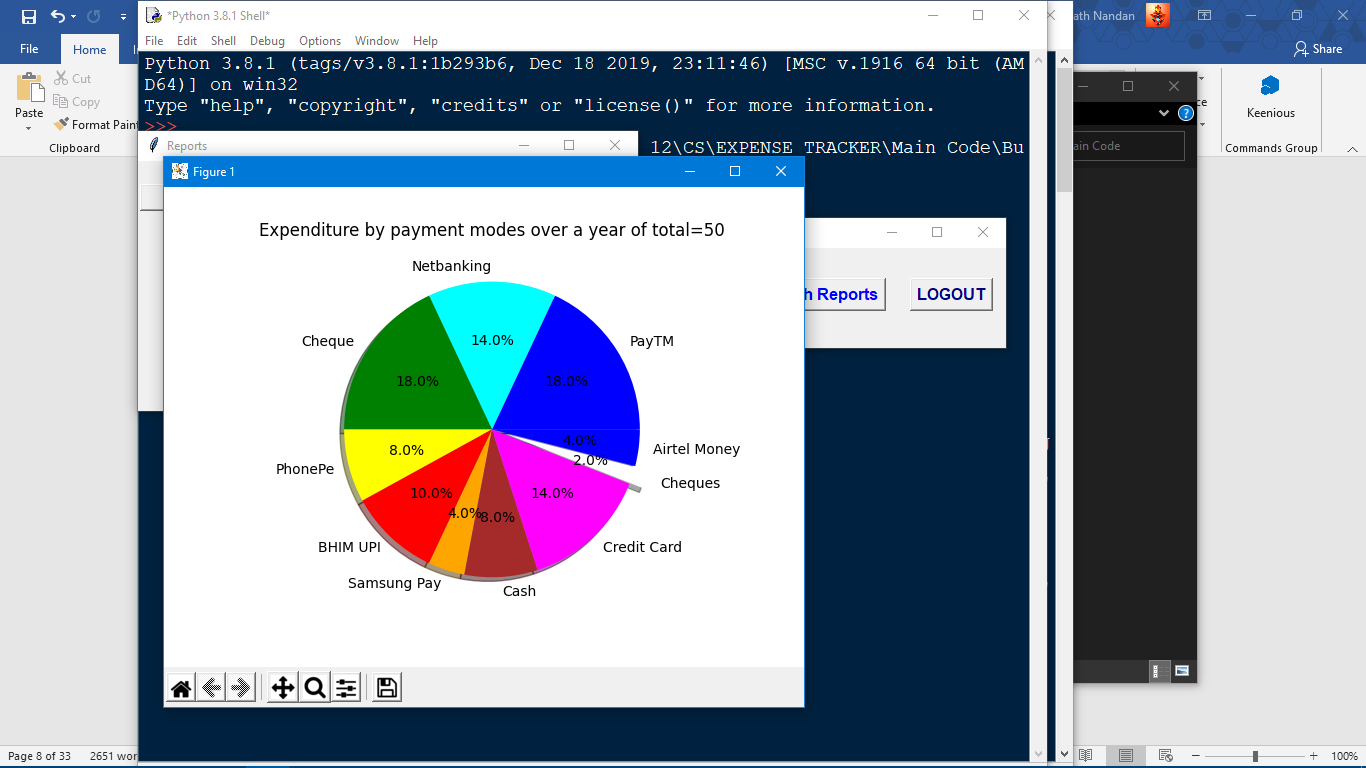
submit\_b1.pack()

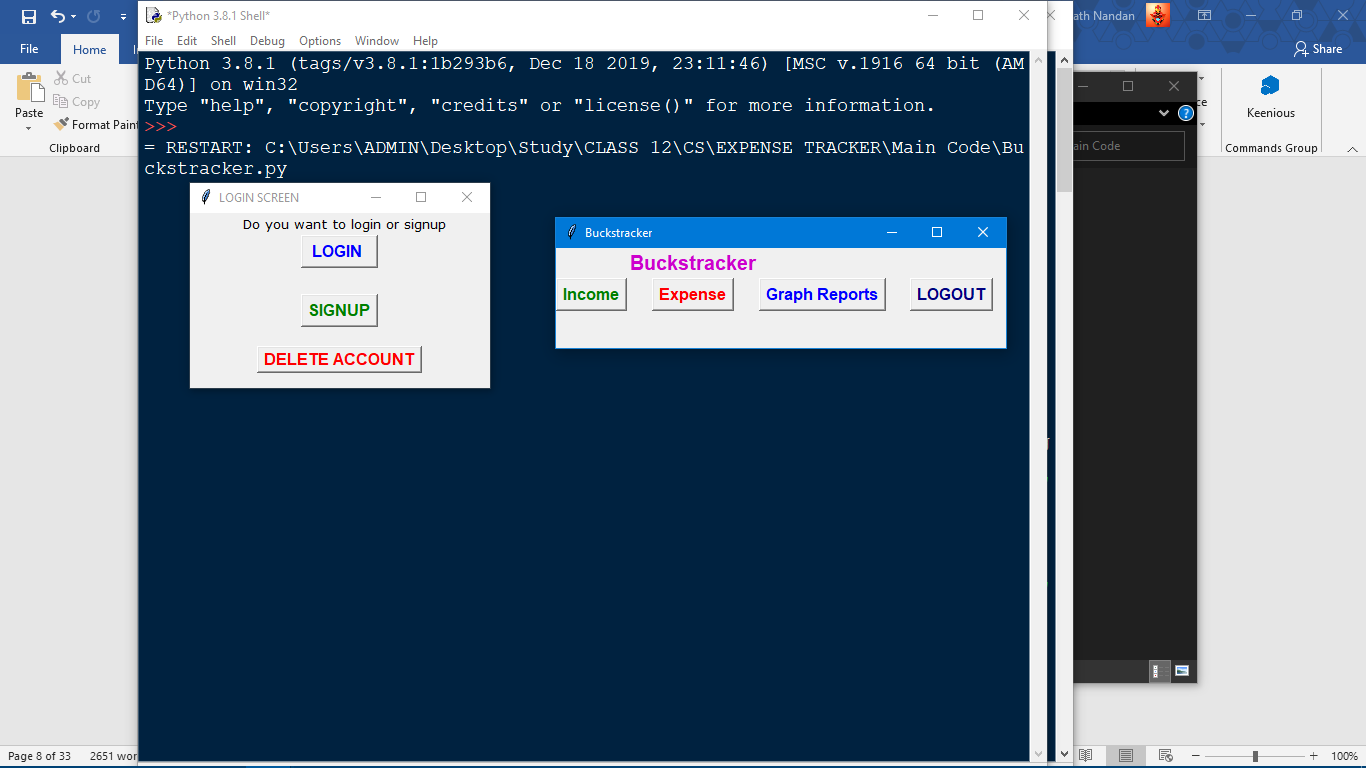
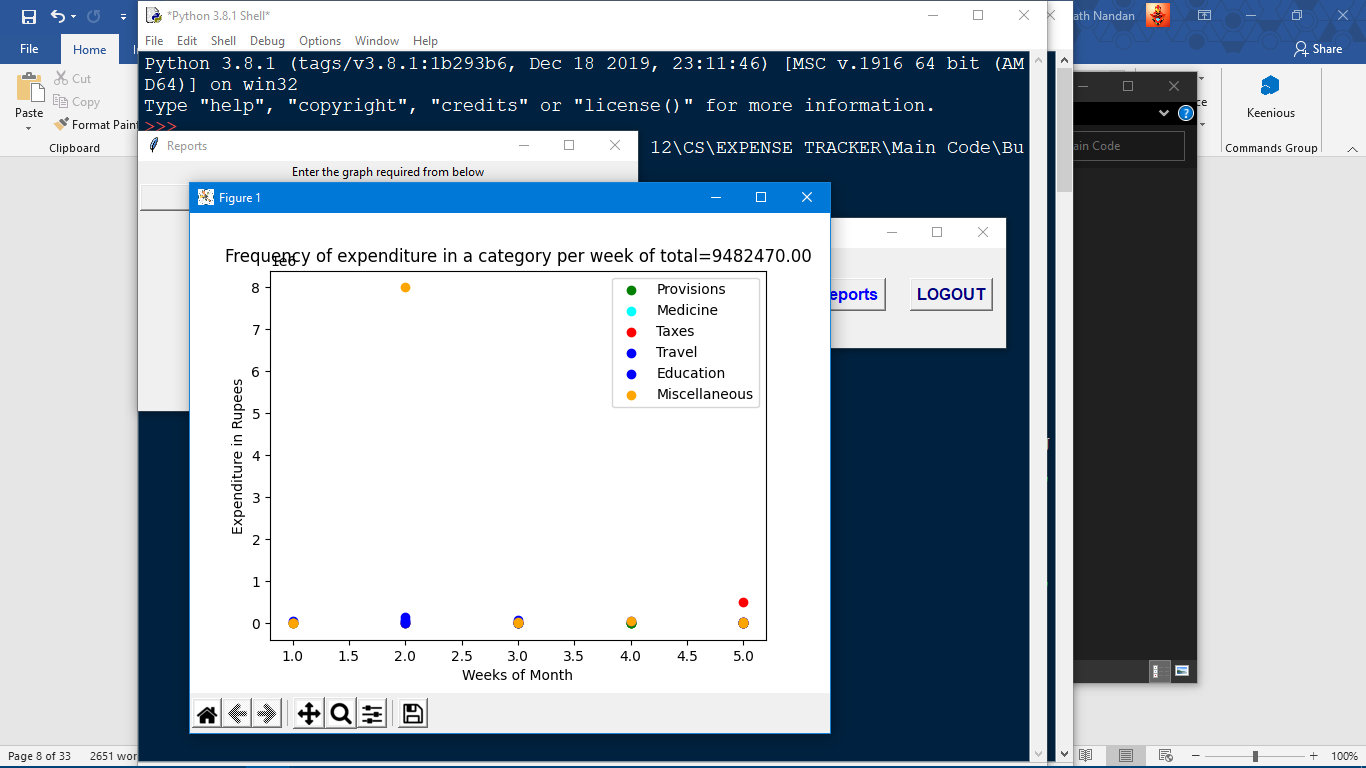
submit\_b2=Button(root,text="Submit option",command= lambda: otheropt(root,options\_var,conn,cursor,options))

submit\_b2.pack()

root.mainloop()

**SAMPLE SCREENSHOTS**





**LIMITATIONS**

1. The project doesn’t look very appealing or colourful. It has all the default colours and backgrounds

2. We can’t add, delete or update multiple records at the same time.

3. The payment modes and categsories are fixed. Users can’t add a new category or payment mode of their choice.

4. We don’t have in depth analysis and statistics of the user as the software does not collect more info such as the user’s occupation, economic status, education, etc.

5. For some of the graphs large amounts of data required to have a good analysis by the graph otherwise the graph would seem so primitive and boring.

**SCOPE FOR IMPROVEMENT**

1. The menu options do not display the selected option. There seems to be some a setback while the functions are imported from the module. If we store the other code in the same python file, it uses up a lot of memory and takes time to run the program.

2. Colours, background images and graphic designs could not be include due to time constraint and application of more complex code.

3. We could expand the database for more payment modes, different reports by graphs and include more categories of transaction

4. So far we have used only 1 table of income and expenditure for all users using the same database i.e same system. In future we can use separate tables for each user to distinguish between them

**REFERENCES**

1. [Treeview - Python Tkinter GUI Tutorial #116 - YouTube](https://www.youtube.com/watch?v=YTqDYmfccQU)
2. [Add And Remove Records From Treeview - Python Tkinter GUI Tutorial #117 - YouTube](https://www.youtube.com/watch?v=rtR5wHXPKZ4&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV&index=117)
3. [Treeview Scrollbar - Python Tkinter GUI Tutorial #120 - YouTube](https://www.youtube.com/watch?v=-rVA37OVDs8&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV&index=120)
4. [Treeview Update Records - Python Tkinter GUI Tutorial #121 - YouTube](https://www.youtube.com/watch?v=lKiNlSs_cms&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV&index=121)
5. [Python GUI's With TKinter - YouTube](https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV)
6. [Treeview - tkinter - Python documentation - Kite](https://www.kite.com/python/docs/tkinter.ttk.Treeview)
7. <https://matplotlib.org/tutorials/index.html>
8. https://youtu.be/yZTBMMdPOww
9. CBSE Class 12 Computer Science Textbook
10. Computational Thinking with Python by Sumita Arora