

PRESIDENCY SCHOOL

BANGALORE SOUTH



Salary Management System

Subject: COMPUTER SCIENCE

DONE BY:

V. S. Ezhilan

Anirudh Kavle

G Aditya Raman

XII 'B'

2020-2021

CERTIFICATE

Name: Anirudh .R. Kavle

Class: 12th 'B'

Exam No: _____

This is certified to be the bonafide work of the student in the computer science laboratory during the academic year 2020-2021.

TEACHER INCHARGE

EXAMINER'S SIGNATURE

PRINCIPAL

Date: _____

Institution Rubber stamp

ACKNOWLEDGEMENT

I wish to express my deep gratitude and sincere thanks to all my teachers for encouragement and the management for providing all facilities to successfully complete the project work.

I extend my sincere thanks to my principal, Mrs. J Bhuvaneshwari and my Computer Science teacher, Mrs. Tamil Selvi whose valuable guidance helped me not only successfully complete the project but also appreciate the beauty of the computer science.

I extend my gratitude to my parents and classmates for their valuable support and time.

INDEX

S.No	Topic	Page No.
1.	System Hardware and Software Specifications	5
2.	Project Synopsis	6
3.	Design Work	9
4.	Coding	12
5.	Output	31
6.	Further Development Area	45
7.	Bibliography	46

SYSTEM SOFTWARE AND HARDWARE **SPECIFICATIONS**

SOFTWARE

The software used to run the program are :

- Windows 10.0.
- Anaconda.
- MySql 8.0.20.0.

HARDWARE

The hardware used to run the project are :

- 2.11 GHz Dual Core Processor.
- 8.00 GB RAM.
- 64-bit operating system, x64-based processor.

PROJECT SYNOPSIS

Salary Management System is an application that deals with everything related Money Management and Budgeting.

Our application uses simple tools and techniques to manage your money more effectively, conveniently and securely.

The application uses MySQL as the database which is also protected by a password to ensure your data is safe.

The application also uses smart budgeting techniques to auto calculate and make a budget for you without any intervention.

We can also show you detailed analysis through bar graphs and pie chart to get a detailed assessment of your expenditure and savings. These mesmerizing graphs and pie charts were made using matplotlib library with handpicked themes to enhance the key features.

The beautiful and minimal designs were made using tkinter library to not only make salary management user friendly but also fun and engaging.

Our program can be broadly divided into 5 features and functionalities:-

1. Creation of tables in the respective database on any system just by entering the correct details on our login page. These tables cover all the categories namely-
 - Child Care – to keep track of money being spent for children.
 - Health Care – to keep track of money being spent on health care.
 - Housing – to keep track of money being spent on insurance, maintenance and rent and also the income from other houses.
 - Transportation – to track and analyze money being spent on commute and other general travel expenses.
 - Living expenses – which cover day to day expenses like groceries, entertainment etc.
 - Miscellaneous – to track the money spent on pets, hobbies, gifts, donations and to cover any other category not mentioned in the previous categories.

These databases are therefore created by our program automatically without any user interference on the click of 1 button.

2. Entering values onto these tables, we have taken immense care in ensuring we boil down to every sub-category under our main category to ensure that user can get maximum coverage on their expenditure and therefore also see a clearer and more accurate analysis. Here are the categories along with their sub-categories with each asking the respective months also:

Category	Sub-Category
Child Care	Tuition, Other expenses for children
Health Care	Medical Insurance, Medical Expenses
Housing	Income from houses, Rent Paid, House maintenance, House Insurance and Other Spending on Houses.
Transportation	Vehicle loan, Insurance, Maintenance, Fuel Expenses, Other transportation Expenses.
Living Expenses	Groceries, Clothing, Entertainment, Other living expenses
Miscellaneous	Pet care, Hobbies, Gifts and donations, Vacation, Other miscellaneous expenses.

3. Analysis of data using data visualization-this is the crux of our program, all the data entered for 12 months in a year can be individually viewed and assessed by the user. Using matplotlib library we analyzed the data from the user and used appropriate legends and labels for both bar graphs and pie charts.

We begin by asking the user for which month he/she would like to view analysis of and proceed to display the table menu, and after the user chooses one and also chooses to view bar graph or pie chart we display them accordingly in the python output.

4. Budgeting or should we say ‘Smart Budgeting, after having done some sophisticated research on this matter, we discovered the most popular and best budgeting technique to help our users and to reduce manual labor. We call it ‘Automatic Budgeting’ or ‘Autobudget’. Using a pre-written set of code we create a tailored budget for the user without his/her help. And these details are then displayed to user. If the user is unhappy with our autobudget we have also included an option for manual budget to enter their own values and make the required changes.
5. Tracking budget-which is the application of making budget in the first place, here we ask the user for which month he/she would like to track budget for and then show a comparison between their budget and whether they remained under it or exceeded it, we also tell the user by how much money they defied their budget.

DESIGN WORK

Libraries Used

1. Tkinter : Tkinter is Python's de-facto standard GUI (Graphical User Interface) package. GUI is nothing but a desktop app that provides you with an interface that helps you to interact with the computers and enriches your experience of giving a command (command-line input) to your code. They are used to perform different tasks in desktops, laptops, and other electronic devices, etc.

Methods used are :

- geometry() : This method is used to set the dimensions of the Tkinter window as well as it is used to set the position of the main window on the user's desktop.
- Frame() : It works like a container, which is responsible for arranging the position of other widgets. It uses rectangular areas in the screen to organize the layout and to provide padding of these widgets.
- grid() : This geometry manager organizes widgets in a table-like structure in the parent widget.
- pack() : This geometry manager organizes widgets in blocks before placing them in the parent widget.
- Label() : This widget implements a display box where you can place text or images. The text displayed by this widget can be updated at any time you want.
- Button() : The Button widget is used to add buttons in a Python application. These buttons can display text or images that convey the purpose of the buttons. You can attach a function or a method to a button which is called automatically when you click the button.
- Entry() : The Entry widget is used to accept single-line text strings from a user.

2. mysql.connector : mysql.connector is an interface for connecting to a MySQL database server from Python. Using MySQL.connector library you can connect MySQL databases from within Python.

Methods used are :

- `connect()` : Establish a connection to the MySQL database.
- `cursor()` : Creates a cursor object. This is the object you use to interact with the database.
- `execute()` : This method executes the given database operation (query or command). The parameters found in the tuple are bound to the variables in the operation.
- `fetchall()` : The `fetchall()` method retrieves all (remaining) rows of a query result, returning them as a sequence of sequences.
- `commit()`: This method commits the current transaction.

3. `matplotlib.pyplot` : Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

Methods used are :

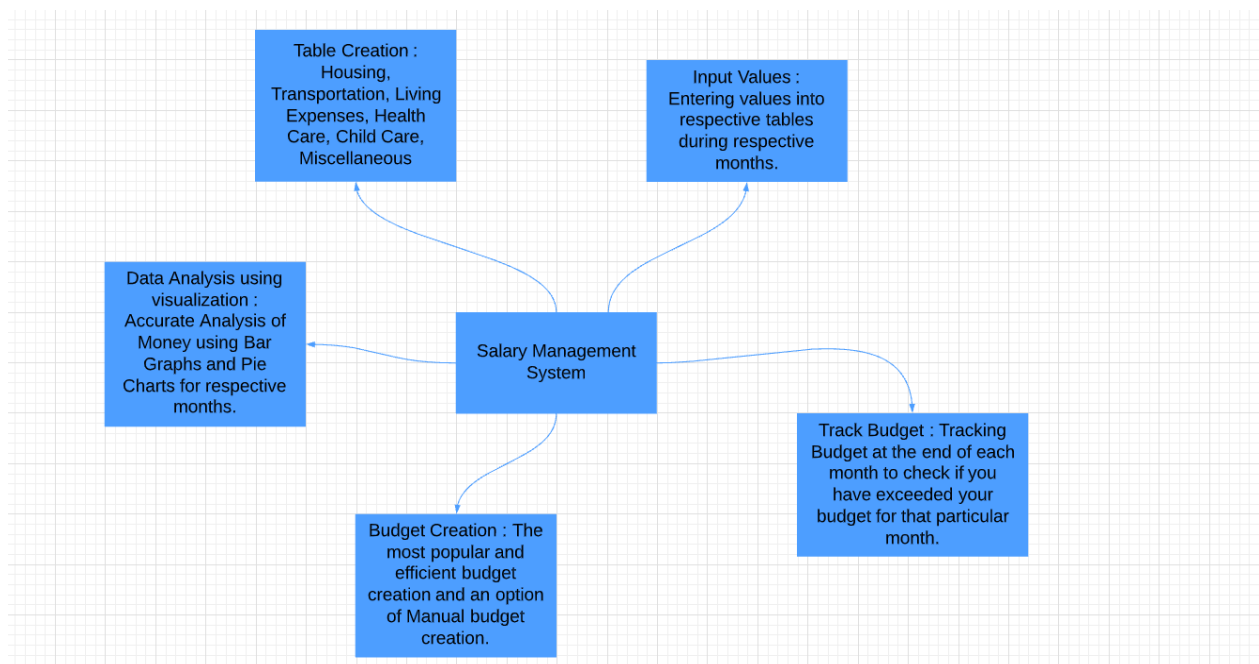
- `bar()` : Make a bar plot.
- `show()` : Display a figure.
- `xlabel()` : Set the label for the x-axis.
- `ylabel()` : Set the label for the y-axis.
- `pie()` : Make a pie chart.
- `tight_layout()` : Used to automatically adjust subplot parameters to give specified padding.
- `style.use()` : Used to create and use custom styles.

4. `functools` : The `functools` module is for higher-order functions that act on or return other functions. In general, any callable object can be treated as a function for the purposes of this module.

Method used are :

- `Partial()` : A partial function is an original function for particular argument values. Objects created by `partial()` have three read-only attributes:

- ✓ `partial.func` – It returns the name of parent function along with hexadecimal address.
- ✓ `partial.args` – It returns the positional arguments provided in partial function.
- ✓ `partial.keywords` – It returns the keyword arguments provided in partial function.



CODING

```
import mysql.connector as ctr
from matplotlib import pyplot as plt
import tkinter as tk
from functools import partial
from tkinter import ttk
from tkinter import messagebox

mycon = mycursor = plot_month = track_month = ""

necessities_ab = 0.0
wants_ab = 0.0
savings_ab = 0.0
monthly_aftertax_income = 0.0

manual_necessities = 0.0
manual_wants = 0.0
manual_savings = 0.0

cnd_for_budget_creation = ""

track_cnd = ""

def connect(password, database):
    global mycon
    global mycursor
    psd = password.get()
    db = database.get()
    mycon = ctr.connect(host = "localhost", user = "root", password = psd, database = db)
    if mycon.is_connected():
        print("Successfully connected to MySQL")
        print()
    mycursor = mycon.cursor()
    Menu()

def Login():

    tkwindow = tk.Tk()
    tkwindow.geometry('400x150')
    tkwindow.title('Salary Management System Login')

    tk.Label(tkwindow, text="Password").grid(row=0, column=0)
    password = tk.StringVar()
```

```

tk.Entry(tkwindow, textvariable=password, show='*').grid(row=0, column=1)

tk.Label(tkwindow, text="Database").grid(row=1, column=0)
database = tk.StringVar()
tk.Entry(tkwindow, textvariable=database).grid(row=1, column=1)

validateLogin1 = partial(connect, password, database)

tk.Button(tkwindow, text = "Login", command = validateLogin1).grid(row=2, column=0)

tkwindow.mainloop()

def Create_Tables():
    mycursor.execute("Create table Housing(Month varchar(9), Income_from_Houses float,
    Rent_Paid float, House_Maintainance float, House_Insurance float,
    Other_Spending_on_houses float)")
    mycursor.execute("Create table Transportation(Month varchar(9), Vehicle_Loan float,
    Vehicle_Insurance float, Vehicle_Maintainance float, Fuel_Expenses float,
    Other_Transportation_Expenses float)")
    mycursor.execute("Create table Living_Expenses(Month varchar(9), Groceries float,
    Clothing float, Entertainment float, Other_Living_Expenses float)")
    mycursor.execute("Create table Health_Care(Month varchar(9), Medical_Insurance float,
    Medical_Expenses float)")
    mycursor.execute("Create table Child_Care(Month varchar(9), Tuition float,
    Other_Expenses_for_Child float)")
    mycursor.execute("Create table Miscellaneous(Month varchar(9), Pet_Care float, Hobbies
    float, Gifts_and_Donations float, Vacation float, Other_Miscellaneous_Expenses float)")
    print("All the tables have been created")
    print()

def makeform(root, fields):
    entries = []
    for field in fields:
        row = tk.Frame(root)
        lab = tk.Label(row, text=field)
        ent = tk.Entry(row)
        row.pack(side=tk.TOP, fill=tk.X, padx=5, pady=5)
        lab.pack(side=tk.LEFT)
        ent.pack(side=tk.RIGHT, fill=tk.X)
        entries.append((field, ent))
    return entries

def Housing():
    root = tk.Tk()
    root.title('Housing Table')

```

```

fields = 'Month', 'Income from Houses', 'Rent Paid', 'House Maintainance', 'House
Insurance', 'Other Spending on Houses'
ents = makeform(root, fields)
root.bind(lambda event, e=ents: Housing_Values(e))
tk.Button(root, text='Submit', command=(lambda e=ents:
Housing_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
root.mainloop()

def Housing_Values(entries):
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    a = tuple(value_list)
    h1, h2, h3, h4, h5, h6 = a
    if float(h2)<0 or float(h3)<0 or float(h4)<0 or float(h5)<0 :
        messagebox.showinfo("Invalid Input!!", "Please enter positive values")
    else :
        mycursor.execute("insert into Housing values('{}',{},{},{},{},{})".format(h1, float(h2),
float(h3), float(h4), float(h5), float(h6)))
        mycon.commit()
        print("Values were added to the Housing table")

def Transportation():
    root = tk.Tk()
    root.title("Transportation Table")
    fields = 'Month', 'Vehicle Loan', 'Vehicle Insurance', 'Vehicle Maintainance', 'Fuel
Expenses', 'Other Transportation Expenses'
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: Transportation_Values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
Transportation_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    root.mainloop()

def Transportation_Values(entries):
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    a = tuple(value_list)
    t1, t2, t3, t4, t5, t6= a
    if float(t2)<0 or float(t3)<0 or float(t4)<0 or float(t5)<0 or float(t6)<0:
        messagebox.showinfo("Invalid Input!!", "Please enter positive values")
    else :
        mycursor.execute("insert into Transportation values('{}',{},{},{},{},{})".format(t1,
float(t2), float(t3), float(t4), float(t5), float(t6)))

```

```

mycon.commit()
print("Values were added to the Transportation table")

def Living_Expenses():
    root = tk.Tk()
    root.title('Living Expenses Table')
    fields = 'Month', 'Groceries', 'Clothing', 'Entertainment', 'Other Living Expenses'
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: Living_Expenses_Values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
Living_Expenses_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    root.mainloop()

def Living_Expenses_Values(entries):
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    a = tuple(value_list)
    l1, l2, l3, l4, l5 = a
    if float(l2)<0 or float(l3)<0 or float(l4)<0 or float(l5)<0 :
        messagebox.showinfo("Invalid Input!!", "Please enter positive values")
    else :
        mycursor.execute("insert into Living_Expenses values('{}',{},{},{},{})".format(l1,
float(l2), float(l3), float(l4), float(l5)))
        mycon.commit()
        print("Values were added to the Living Expenses Table")

def Health_Care():
    root = tk.Tk()
    root.title('Health Care Table')
    fields = 'Month', 'Medical Insurance', 'Medical Expenses'
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: Health_Care_Values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
Health_Care_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    root.mainloop()

def Health_Care_Values(entries):
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    a = tuple(value_list)
    hc1, hc2, hc3 = a
    if float(hc2)<0 or float(hc3)<0 :

```

```

        messagebox.showinfo("Invalid Input!!", "Please enter positive values")
    else :
        mycursor.execute("insert into health_care values('{ }',{ },{ })".format(hc1, float(hc2),
float(hc3)))
        mycon.commit()
        print("Values were added to the Health Care Table")

def Child_Care():
    root = tk.Tk()
    root.title('Child Care Table')
    fields = 'Month', 'Tuition', 'Other Expenses for Child'
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: Child_Care_Values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
Child_Care_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    root.mainloop()

def Child_Care_Values(entries):
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    a = tuple(value_list)
    cc1, cc2, cc3 = a
    if float(cc2)<0 or float(cc3)<0 :
        messagebox.showinfo("Invalid Input!!", "Please enter positive values")
    else :
        mycursor.execute("insert into child_care values('{ }',{ },{ })".format(cc1, float(cc2),
float(cc3)))
        mycon.commit()
        print("Values were added to the Child Care Table")

def Miscellaneous():
    root = tk.Tk()
    root.title('Miscellaneous Table')
    fields = 'Month', 'Pet Care', 'Hobbies', 'Gifts and Donations', 'Vacation', 'Other
Miscellaneous Expenses'
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: Miscellaneous_Values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
Miscellaneous_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    root.mainloop()

def Miscellaneous_Values(entries):
    value_list = []
    for entry in entries:

```



```

        value = entry[1].get()
        value_list.append(value)
    a = tuple(value_list)
    mv1, mv2, mv3, mv4, mv5, mv6 = a
    if float(mv2)<0 or float(mv3)<0 or float(mv4)<0 or float(mv5)<0 or float(mv6)<0:
        messagebox.showinfo("Invalid Input!!", "Please enter positive values")
    else :
        mycursor.execute("insert into miscellaneous values('{}',{},{},{},{},{})".format(mv1,
float(mv2), float(mv3), float(mv4), float(mv5), float(mv6)))
        mycon.commit()
        print("Values were added to the Miscellaneous Table")

```

def Menu():

```

    tkwindow = tk.Tk()
    tkwindow.geometry('1000x500')
    tkwindow.title('Main Menu')
    tk.Label(tkwindow, text='Main Menu', font = ('Arvo', 30, 'bold')).pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Create tables', command = Create_Tables, font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Input values', command = Table_Menu, font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Check Plots', command = Which_Month_Plots, font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Create budget', command = MAI, font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Track budget', command = Which_Month_Track_Budget,
font=('Arvo', 20), foreground='Green').pack(fill = tk.BOTH, expand = True)
    tkwindow.mainloop()

```

def Table_Menu():

```

    tkwindow = tk.Tk()
    tkwindow.geometry('1000x500')
    tkwindow.title("Table Menu")
    tk.Label(tkwindow, text='Table Menu', font = ('Arvo', 30, 'bold')).pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Housing', command = Housing,font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Transportation', command = Transportation,font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Living Expenses', command = Living_Expenses,font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)

```

```

tk.Button(pane,text='Health Care', command = Health_Care,font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Child Care', command = Child_Care,font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Miscellaneous', command = Miscellaneous,font=('Arvo', 20),
foreground='Green').pack(fill = tk.BOTH, expand = True)
tkwindow.mainloop()

def Which_Month_Plots():

    tkwindow = tk.Tk()
    tkwindow.title('Choosing Month for Plots')
    tkwindow.geometry('500x250')
    tk.Label(tkwindow, text = "Monthly Analysis", background = 'green', foreground
="white", font = ("Times New Roman", 15)).grid(row = 0, column = 1)
    tk.Label(tkwindow, text = "Select the Month :", font = ("Times New Roman",
10)).grid(column = 0, row = 5, padx = 10, pady = 25)
    monthchoosen = ttk.Combobox(tkwindow, width = 27, values=('January', 'February',
'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'))
    monthchoosen.grid(column = 1, row = 5)

    def selectmonth_action():
        global plot_month
        plot_month = monthchoosen.get()
        if plot_month == "":
            messagebox.showinfo("NO INPUT", "Kindly choose something")
        elif plot_month not in ['January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December']:
            messagebox.showinfo("WRONG INPUT", "Kindly use the dropdown")
        else:
            print(plot_month,"is chosen for checking Plots")
    ttk.Button(tkwindow, text="Get Value", command = selectmonth_action).grid(column = 3,
row = 5)
    ttk.Button(tkwindow, text="Submit", command = Plot_Menu).grid(column = 3, row = 6)
    tkwindow.mainloop()

def Plot_Menu():

    tkwindow = tk.Tk()
    tkwindow.geometry('1000x500')
    tkwindow.title("Plot Menu")
    tk.Label(tkwindow, text = 'Plot Menu', font = ('Arvo', 30, 'bold')).pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Housing',font=('Arvo', 20), foreground='Green', command =
Plot_Housing).pack(fill = tk.BOTH, expand = True)

```

```

tk.Button(pane,text='Transportation',font=('Arvo', 20), foreground='Green',command =
Plot_Transportation).pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Living Expenses',font=('Arvo', 20), foreground='Green',command =
Plot_Living_Expenses).pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Health Care', font=('Arvo', 20), foreground='Green',command =
Plot_Health_Care).pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Child Care',font=('Arvo', 20), foreground='Green',command =
Plot_Child_Care).pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Miscellaneous',font=('Arvo', 20), foreground='Green', command =
Plot_Miscellaneous).pack(fill = tk.BOTH, expand = True)
tkwindow.mainloop()

```

```

def Plot_Housing():
    tkwindow = tk.Tk()
    tkwindow.geometry('500x250')
    tkwindow.title("Plots for Housing Table")
    k = tk.Label(tkwindow, text = 'Data Analysis', font = ('Arvo', 40, 'bold'))
    k.pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Bar Graph',font=('Arvo', 20),foreground='Green', command =
bg_Housing).pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Pie Chart',font=('Arvo', 20),foreground='Green', command =
pg_Housing).pack(fill = tk.BOTH, expand = True)
    tkwindow.mainloop()

```

```

def Plot_Transportation():
    tkwindow = tk.Tk()
    tkwindow.geometry('500x250')
    tkwindow.title("Plots for Transportation Table")
    k = tk.Label(tkwindow, text = 'Data Analysis', font = ('Arvo', 40, 'bold'))
    k.pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Bar Graph',font=('Arvo', 20),foreground='Green', command =
bg_Transportation).pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Pie Chart',font=('Arvo', 20),foreground='Green', command =
pg_Transportation).pack(fill = tk.BOTH, expand = True)
    tkwindow.mainloop()

```

```

def Plot_Living_Expenses():
    tkwindow = tk.Tk()
    tkwindow.geometry('500x250')
    tkwindow.title("Plots for Living Expenses Table")
    k = tk.Label(tkwindow, text = 'Data Analysis', font = ('Arvo', 40, 'bold'))
    k.pack()

```

```

pane = tk.Frame(tkwindow)
pane.pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Bar Graph',font=('Arvo', 20),foreground='Green', command =
bg_Living_Expenses).pack(fill = tk.BOTH, expand = True)
tk.Button(pane,text='Pie Chart',font=('Arvo', 20),foreground='Green', command =
pg_Living_Expenses).pack(fill = tk.BOTH, expand = True)
tkwindow.mainloop()

```

```

def Plot_Health_Care():
    tkwindow = tk.Tk()
    tkwindow.geometry('500x250')
    tkwindow.title("Plots for Health Care Table")
    k = tk.Label(tkwindow, text = 'Data Analysis', font = ('Arvo', 40, 'bold'))
    k.pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Bar Graph',font=('Arvo', 20),foreground='Green', command =
bg_Health_Care).pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Pie Chart',font=('Arvo', 20),foreground='Green', command =
pg_Health_Care).pack(fill = tk.BOTH, expand = True)
    tkwindow.mainloop()

```

```

def Plot_Child_Care():
    tkwindow = tk.Tk()
    tkwindow.geometry('500x250')
    tkwindow.title("Plots for Child Care Table")
    k = tk.Label(tkwindow, text = 'Data Analysis', font = ('Arvo', 40, 'bold'))
    k.pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Bar Graph',font=('Arvo', 20),foreground='Green', command =
bg_Child_Care).pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Pie Chart',font=('Arvo', 20),foreground='Green', command =
pg_Child_Care).pack(fill = tk.BOTH, expand = True)
    tkwindow.mainloop()

```

```

def Plot_Miscellaneous():
    tkwindow = tk.Tk()
    tkwindow.geometry('500x250')
    tkwindow.title("Plots for Miscellaneous Table")
    k = tk.Label(tkwindow, text = 'Data Analysis', font = ('Arvo', 40, 'bold'))
    k.pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Bar Graph',font=('Arvo', 20),foreground='Green', command =
bg_Miscellaneous).pack(fill = tk.BOTH, expand = True)

```

```
tk.Button(pane,text='Pie Chart',font=('Arvo', 20),foreground='Green', command =
pg_Miscellaneous).pack(fill = tk.BOTH, expand = True)
tkwindow.mainloop()
```

```
def bg_Housing():
    Category = Expenditure = []
    mycursor.execute("select * from housing where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, ifh_plot, efh_plot, house_maintenance_plot, house_insurance_plot,
other_spending_on_house_plot) = i
        Category = ["Income", "Rent Paid", "Maintenance", "Insurance", "Other Spending"]
        Expenditure = [ifh_plot, efh_plot, house_maintenance_plot, house_insurance_plot,
other_spending_on_house_plot]
        plt.style.use("ggplot")
        plt.bar(Category, Expenditure, color = ['r','b','g','black','yellow'], width = [0.5, 0.5, 0.5,
0.5, 0.5])
        plt.xlabel("Category")
        plt.ylabel("Expenditure")
        plt.title("Housing")
        plt.show()
```

```
def pg_Housing():
    mycursor.execute("select * from housing where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, ifh_plot, efh_plot, house_maintenance_plot, house_insurance_plot,
other_spending_on_house_plot) = i
        slices = [ifh_plot, efh_plot, house_maintenance_plot, house_insurance_plot,
other_spending_on_house_plot]
        labels = ["Income", "Rent Paid", "Maintenance", "Insurance", "Other Spending"]
        plt.style.use("ggplot")
        plt.pie(slices, labels=labels, shadow=True, startangle=90, autopct='%1.1f%%',
wedgeprops={'edgecolor': 'black'})
        plt.title("Housing")
        plt.tight_layout()
        plt.show()
```

```
def bg_Transportation():
    Category = Expenditure = []
    mycursor.execute("select * from transportation where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, vehicle_loan_plot, vehicle_insurance_plot, vehicle_maintenance_plot,
fuel_expenses_plot, other_te_plot) = i
        Category = ["Loan", "Insurance", "Maintenance", "Fuel", "Other"]
        Expenditure = [vehicle_loan_plot, vehicle_insurance_plot, vehicle_maintenance_plot,
fuel_expenses_plot, other_te_plot]
        plt.style.use("ggplot")
```

```

plt.bar(Category, Expenditure, color = ['r','b','g','black','yellow'], width = [0.5, 0.5, 0.5, 0.5, 0.5])
plt.xlabel("Category")
plt.ylabel("Expenditure")
plt.title("Transportation")
plt.show()

```

```

def pg_Transportation():
    mycursor.execute("select * from transportation where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, vehicle_loan_plot, vehicle_insurance_plot, vehicle_maintainance_plot,
fuel_expenses_plot, other_te_plot) = i
        slices = [vehicle_loan_plot, vehicle_insurance_plot, vehicle_maintainance_plot,
fuel_expenses_plot, other_te_plot]
        labels = ["Loan", "Insurance", "Maintainance", "Fuel", "Other"]
        plt.style.use("ggplot")
        plt.pie(slices, labels=labels, shadow=True, startangle=90, autopct='%1.1f%%',
wedgeprops={'edgecolor': 'black'})
        plt.title("Transportation")
        plt.tight_layout()
        plt.show()

```

```

def bg_Living_Expenses():
    Category = Expenditure = []
    mycursor.execute("select * from living_expenses where Month =
'{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, groceries_plot, clothing_plot, entertainment_plot, other_le_plot) = i
        Category = ["Groceries", "Clothing", "Entertainment", "Other"]
        Expenditure = [groceries_plot, clothing_plot, entertainment_plot, other_le_plot]
        plt.style.use("ggplot")
        plt.bar(Category, Expenditure, color = ['r','b','g','black'], width = [0.5, 0.5, 0.5, 0.5])
        plt.xlabel("Category")
        plt.ylabel("Expenditure")
        plt.title("Living Expenses")
        plt.show()

```

```

def pg_Living_Expenses():
    mycursor.execute("select * from living_expenses where Month =
'{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, groceries_plot, clothing_plot, entertainment_plot, other_plot) = i
        slices = [groceries_plot, clothing_plot, entertainment_plot, other_plot]
        labels = ["Groceries", "Clothing", "Entertainment", "Other"]
        plt.style.use("ggplot")

```

```

plt.pie(slices, labels=labels, shadow=True, startangle=90, autopct='%1.1f%%',
wedgeprops={'edgecolor': 'black'})
plt.title("Living Expenses")
plt.tight_layout()
plt.show()

def bg_Health_Care():
    Category = Expenditure = []
    mycursor.execute("select * from health_care where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, medical_insurance_plot, medical_expenses_plot) = i
        Category = ["Medical Insurance", "Medical Expenses"]
        Expenditure = [medical_insurance_plot, medical_expenses_plot]
        plt.style.use("ggplot")
        plt.bar(Category, Expenditure, color = ['r','b'], width = [0.5, 0.5])
        plt.xlabel("Category")
        plt.ylabel("Expenditure")
        plt.title("Health Care")
        plt.show()

def pg_Health_Care():
    mycursor.execute("select * from health_care where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, medical_insurance_plot, medical_expenses_plot) = i
        slices = [medical_insurance_plot, medical_expenses_plot]
        labels = ["Medical Insurance", "Medical Expenses"]
        plt.style.use("ggplot")
        plt.pie(slices, labels=labels, shadow=True, startangle=90, autopct='%1.1f%%',
wedgeprops={'edgecolor': 'black'})
        plt.title("Health Care")
        plt.tight_layout()
        plt.show()

def bg_Child_Care():
    Category = Expenditure = []
    mycursor.execute("select * from child_care where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, tuition_plot, other_child_expense_plot) = i
        Category = ["Tuition", "Other"]
        Expenditure = [tuition_plot, other_child_expense_plot]
        plt.style.use("ggplot")
        plt.bar(Category, Expenditure, color = ['r','b'], width = [0.5, 0.5])
        plt.xlabel("Category")
        plt.ylabel("Expenditure")
        plt.title("Child Care")
        plt.show()

```

```

def pg_Child_Care():
    mycursor.execute("select * from child_care where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, tuition_plot, other_child_expense_plot) = i
        slices = [tuition_plot, other_child_expense_plot]
        labels = ["Tuition", "Other"]
        plt.style.use("ggplot")
        plt.pie(slices, labels=labels, shadow=True, startangle=90, autopct='%1.1f%%',
wedgeprops={'edgecolor': 'black'})
        plt.title("Child Care")
        plt.tight_layout()
        plt.show()

def bg_Miscellaneous():
    Category = Expenditure = []
    mycursor.execute("select * from miscellaneous where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, pet_care_plot, hobbies_and_sports_plot, gifts_and_donations_plot,
vacation_plot, other_miscellaneous_plot) = i
        Category = ["Pet Care", "Hobbies", "Gifts/Donations", "Vacation", "Other"]
        Expenditure = [pet_care_plot, hobbies_and_sports_plot, gifts_and_donations_plot,
vacation_plot, other_miscellaneous_plot]
        plt.style.use("ggplot")
        plt.bar(Category, Expenditure, color = ['r','b','g','black','yellow'], width = [0.5, 0.5, 0.5,
0.5,0.5])
        plt.xlabel("Category")
        plt.ylabel("Expenditure")
        plt.title("Miscellaneous")
        plt.show()

def pg_Miscellaneous():
    mycursor.execute("select * from miscellaneous where Month = '{}'.format(plot_month))
    for i in mycursor.fetchall():
        (extra_month, pet_care_plot, hobbies_and_sports_plot, gifts_and_donations_plot,
vacation_plot, other_miscellaneous_plot) = i
        slices = [pet_care_plot, hobbies_and_sports_plot, gifts_and_donations_plot,
vacation_plot, other_miscellaneous_plot]
        labels = ["Pet Care", "Hobbies", "Gifts and Donations", "Vacation", "Other"]
        plt.style.use("ggplot")
        plt.pie(slices, labels=labels, shadow=True, startangle=90, autopct='%1.1f%%',
wedgeprops={'edgecolor': 'black'})
        plt.title("Miscellaneous")
        plt.tight_layout()
        plt.show()

```



```

def MAI():
    root = tk.Tk()
    root.title("Monthly After-Tax Salary")
    fields = ('Enter the monthly after-tax salary',)
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: MAI_values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
MAI_values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    tk.Button(root, text = "Proceed", command = Budget_Menu).pack(side=tk.LEFT, padx=5,
pady=5)
    root.mainloop()

```

```

def MAI_values(entries):
    global necessities_ab, wants_ab, savings_ab, track_cnd, monthly_aftertax_income
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    for i in value_list:
        monthly_aftertax_income = float(i)
    necessities_ab = 0.5 * monthly_aftertax_income
    wants_ab = 0.3 * monthly_aftertax_income
    savings_ab = 0.2 * monthly_aftertax_income
    print("Monthly After Tax Income :",monthly_aftertax_income)
    track_cnd = "Auto Budget"

```

```

def Budget_Menu():
    tkwindow = tk.Tk()
    tkwindow.geometry('1000x500')
    tkwindow.title("Budget Menu")
    tk.Label(tkwindow, text = 'Budget Menu', font = ('Arvo', 30, 'bold')).pack()
    pane = tk.Frame(tkwindow)
    pane.pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Automatic Budget', font=('Arvo', 20),foreground='Green', command =
budget_desc).pack(fill = tk.BOTH, expand = True)
    tk.Button(pane,text='Manual Budget', font=('Arvo', 20),foreground='Green', command =
Manual_Budget).pack(fill = tk.BOTH, expand = True)
    tkwindow.mainloop()

```

```

def budget_desc():
    tkwindow = tk.Tk()
    tkwindow.geometry('1000x400')
    tkwindow.title("Auto Budget Description")
    T = tk.Text(tkwindow, height =10, width = 200)
    l = tk.Label(tkwindow, text = "Attention")
    l.config(font =("Calibri", 20))

```

Fact = """"This is the most popular way of budgeting, in this rule of budgeting, you spend roughly 50% of your after-tax money on necessities, no more than 30% on wants, and at least 20% on savings. We like the simplicity of this plan. Over the long term, someone who follows these guidelines will have manageable debt, room to indulge occasionally, and savings to pay irregular or unexpected expenses and retire comfortably."""

```
b1 = tk.Button(tkwindow, text = "Next", command = Auto_Budget_Desc)
l.pack()
T.pack()
b1.pack()
T.insert(tk.END, Fact)
tkwindow.mainloop()
```

```
def Auto_Budget_Desc():
    global cnd_for_budget_creation
    print()
    print("According to the autobudget : ")
    print("Amount of money you should be spending on necessities - ₹",necessities_ab)
    print("Amount of money you should be spending on wants - ₹",wants_ab)
    print("Amount of money you should be saving - ₹",savings_ab)
    print()
    cnd_for_budget_creation = "Budget Created"
```

```
def Manual_Budget():
    root = tk.Tk()
    root.title("Manual Budget")
    fields = 'Necessities', 'Wants', 'Savings'
    ents = makeform(root, fields)
    root.bind(lambda event, e=ents: Manual_Budget_Values(e))
    tk.Button(root, text='Submit', command=(lambda e=ents:
Manual_Budget_Values(e))).pack(side=tk.LEFT, padx=5, pady=5)
    root.mainloop()
```

```
def Manual_Budget_Values(entries):
    global manual_necessities, manual_wants, manual_savings, track_cnd,
cnd_for_budget_creation
    value_list = []
    for entry in entries:
        value = entry[1].get()
        value_list.append(value)
    manual_necessities = float(value_list[0])
    manual_wants = float(value_list[1])
    manual_savings = float(value_list[2])
    total_sum = manual_necessities + manual_wants + manual_savings
    track_cnd = "Manual Budget"
    if total_sum != monthly_aftertax_income :
```

```

        messagebox.showinfo("Invalid input", "The total amount assigned doesn't add up to the
monthly after-tax income.")
    print()
    else:
        print("According to the manual budget : ")
        print("Amount of money you should be spending on necessities - ₹",manual_necessities)
        print("Amount of money you should be spending on wants - ₹",manual_wants)
        print("Amount of money you should be saving - ₹",manual_savings)
        print("You're all done, we will help you keep track of this budget and let you know as
you progress.")
        print()
        cnd_for_budget_creation = "Budget Created"

```

```

def Necessities(tm):
    mycursor.execute("select tuition from child_care where month = '{ }'".format(tm))
    rn1 = mycursor.fetchall()
    mycursor.execute("select medical_insurance from health_care where month =
'{' }'".format(tm))
    rn2 = mycursor.fetchall()
    mycursor.execute("select income_from_houses, rent_paid, house_insurance from housing
where month = '{ }'".format(tm))
    rn3 = mycursor.fetchall()
    mycursor.execute("select groceries from living_expenses where month = '{ }'".format(tm))
    rn4 = mycursor.fetchall()
    mycursor.execute("select vehicle_insurance, vehicle_loan from transportation where
month = '{ }'".format(tm))
    rn5 = mycursor.fetchall()
    rn = rn1[0][0] + rn2[0][0] - rn3[0][0] + rn3[0][1] + rn3[0][2] + rn4[0][0] + rn5[0][0] +
rn5[0][1]
    return rn

```

```

def Wants(tm):
    mycursor.execute("select * from miscellaneous where month = '{ }'".format(tm))
    rw1 = mycursor.fetchall()
    mycursor.execute("select vehicle_maintainance, fuel_expenses,
other_transportation_expenses from transportation where month = '{ }'".format(tm))
    rw2 = mycursor.fetchall()
    mycursor.execute("select clothing, entertainment, other_living_expenses from
living_expenses where month = '{ }'".format(tm))
    rw3 = mycursor.fetchall()
    mycursor.execute("select house_maintainance, other_spending_on_houses from housing
where month = '{ }'".format(tm))
    rw4 = mycursor.fetchall()
    mycursor.execute("select medical_expenses from health_care where month =
'{' }'".format(tm))
    rw5 = mycursor.fetchall()

```

```

mycursor.execute("select other_expenses_for_child from child_care where month =
'{}'".format(tm))
rw6 = mycursor.fetchall()
rw = rw1[0][1] + rw1[0][2] + rw1[0][3] + rw1[0][4] + rw1[0][5] + rw2[0][0] + rw2[0][1] +
rw2[0][2] + rw3[0][0] + rw3[0][1] + rw3[0][2] + rw4[0][0] + rw4[0][1] + rw5[0][0] +
rw6[0][0]
return rw

def Savings(tm):
    rn_total = Necessities(tm)
    rw_total = Wants(tm)
    rs = monthly_aftertax_income - rn_total - rw_total
    return rs

def Which_Month_Track_Budget():

    if cnd_for_budget_creation == "Budget Created" :

        tkwindow = tk.Tk()
        tkwindow.title('Combobox')
        tkwindow.title('Choosing Month for Tracking Budget')
        tkwindow.geometry('500x250')

        tk.Label(tkwindow, text = "Monthly Analysis", background = 'green', foreground
="white", font = ("Times New Roman", 15)).grid(row = 0, column = 1)
        tk.Label(tkwindow, text = "Select the Month :", font = ("Times New Roman",
10)).grid(column = 0, row = 5, padx = 10, pady = 25)
        monthchoosen = ttk.Combobox(tkwindow, width = 27,
values=('January','February','March','April','May','June','July','August','September','October','
November', 'December' ))
        monthchoosen.grid(column = 1, row = 5)

        def selectmonth_action():
            global track_month
            track_month = monthchoosen.get()
            if track_month == "":
                messagebox.showinfo("NO INPUT", "Kindly choose something")
            elif track_month not in ['January', 'February', 'March', 'April', 'May', 'June', 'July',
'August', 'September', 'October', 'November', 'December']:
                messagebox.showinfo("WRONG INPUT", "Kindly use the dropdown")
            else:
                print(track_month,"is chosen for Tracking Budget")
                print()
            ttk.Button(tkwindow, text="Get Value", command = selectmonth_action).grid(column =
3, row = 5)

```

```

        ttk.Button(tkwindow, text="Submit", command = Track_Budget).grid(column = 3, row
= 6)
        tkwindow.mainloop()

    else :

        messagebox.showinfo("ERROR", "Please Create Budget in order to Track Budget")

def Track_Budget():

    global track_cnd

    if track_cnd == "Auto Budget" :
        Track_Auto_Budget()

    if track_cnd == "Manual Budget" :
        Track_Manual_Budget()

def Track_Auto_Budget():

    real_necessities = real_wants = real_savings = 0.0
    real_necessities = Necessities(track_month)
    real_wants = Wants(track_month)
    real_savings = Savings(track_month)
    track_necessities, track_wants, track_savings = necessities_ab, wants_ab, savings_ab
    if real_necessities > track_necessities:
        print("You have exceeded the limit on the amount to be spent on necessities")
        print("Amount exceeded :",real_necessities - track_necessities)
        print()
    else:
        print("You have remained under the budget on necessities")
        print()

    if real_wants > track_wants:
        print("You have exceeded the limit on the amount to be spent on wants")
        print("Amount exceeded :",real_wants - track_wants)
        print()
    else:
        print("You have remained under the budget on wants")
        print()

    if real_savings > track_savings:
        print("You have exceeded the limit on the amount to be saved")
        print("Amount exceeded :",real_savings - track_savings)
        print()
    else:

```

```

    print("You have remained under the budget on for amount saved")
    print()

def Track_Manual_Budget():

    real_necessities = real_wants = real_savings = 0.0
    real_necessities = Necessities(track_month)
    real_wants = Wants(track_month)
    real_savings = Savings(track_month)
    track_necessities, track_wants, track_savings = manual_necessities, manual_wants,
manual_savings
    if real_necessities > track_necessities:
        print("You have exceeded the limit on the amount to be spent on necessities")
        print("Amount exceeded :",real_necessities - track_necessities)
        print()
    else:
        print("You have remained under the budget on necessities")
        print()

    if real_wants > track_wants:
        print("You have exceeded the limit on the amount to be spent on wants")
        print("Amount exceeded :",real_wants - track_wants)
        print()
    else:
        print("You have remained under the budget on wants")
        print()

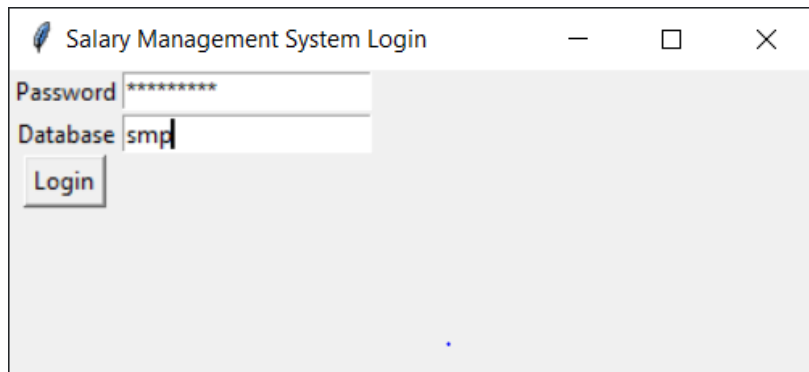
    if real_savings > track_savings:
        print("You have exceeded the limit on the amount to be saved")
        print("Amount exceeded :",real_savings - track_savings)
        print()
    else:
        print("You have remained under the budget on for amount saved")
        print()

Login()

```

OUTPUT

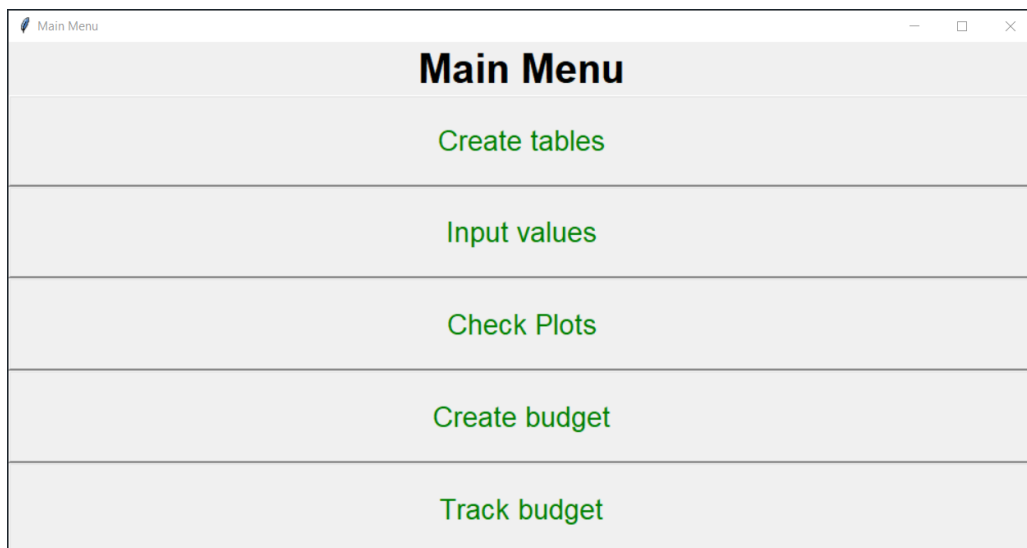
Login Page



A screenshot of a web application window titled "Salary Management System Login". The window has a light gray background and a white title bar with standard minimize, maximize, and close buttons. It contains two input fields: "Password" with a masked value "*****" and "Database" with the value "smp". Below these fields is a "Login" button. The window is centered on the screen.

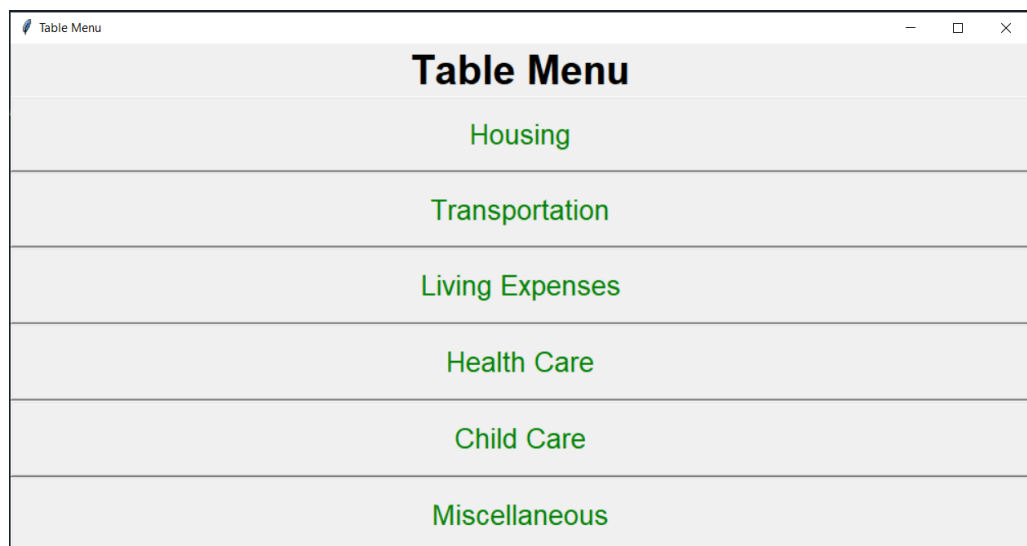
Successfully connected to MySQL

Main Menu



A screenshot of a web application window titled "Main Menu". The window has a light gray background and a white title bar with standard minimize, maximize, and close buttons. It displays a list of six menu items in green text, each on a separate line: "Create tables", "Input values", "Check Plots", "Create budget", and "Track budget". The window is centered on the screen.

Table Menu



A screenshot of a web application window titled "Table Menu". The window has a light gray background and a white title bar with standard minimize, maximize, and close buttons. It displays a list of six menu items in green text, each on a separate line: "Housing", "Transportation", "Living Expenses", "Health Care", "Child Care", and "Miscellaneous". The window is centered on the screen.

Entering Values into Housing Table

Housing Table	
Month	
Income from Houses	
Rent Paid	
House Maintainance	
House Insurance	
Other Spending on Houses	
<input type="button" value="Submit"/>	

Housing Table	
Month	January
Income from Houses	1000
Rent Paid	2000
House Maintainance	3000
House Insurance	4000
Other Spending on Houses	5000
<input type="button" value="Submit"/>	

Values were added to the Housing table

Entering Values into Transportation Table

Transportation Table	
Month	
Vehicle Loan	
Vehicle Insurance	
Vehicle Maintainance	
Fuel Expenses	
Other Transportation Expenses	
<input type="button" value="Submit"/>	

Transportation Table	
Month	January
Vehicle Loan	1000
Vehicle Insurance	2000
Vehicle Maintainance	3000
Fuel Expenses	4000
Other Transportation Expenses	5000
<input type="button" value="Submit"/>	

Values were added to the Transportation table

Entering Values into Living Expenses Table

Living Expense...	
Month	
Groceries	
Clothing	
Entertainment	
Other Living Expenses	
<input type="button" value="Submit"/>	

Living Expense...	
Month	January
Groceries	1000
Clothing	2000
Entertainment	3000
Other Living Expenses	4000
<input type="button" value="Submit"/>	

Values were added to the Living Expenses Table

Entering Values into Health Care table

Health Car...	
Month	<input type="text"/>
Medical Insurance	<input type="text"/>
Medical Expenses	<input type="text"/>
<input type="button" value="Submit"/>	

Health Car...	
Month	January
Medical Insurance	1000
Medical Expenses	2000
<input type="button" value="Submit"/>	

Values were added to the Health Care Table

Entering values into Child Care table

Child Care Table	
Month	<input type="text"/>
Tuition	<input type="text"/>
Other Expenses for Child	<input type="text"/>
<input type="button" value="Submit"/>	

Child Care Table	
Month	January
Tuition	1000
Other Expenses for Child	2000
<input type="button" value="Submit"/>	

Values were added to the Child Care Table

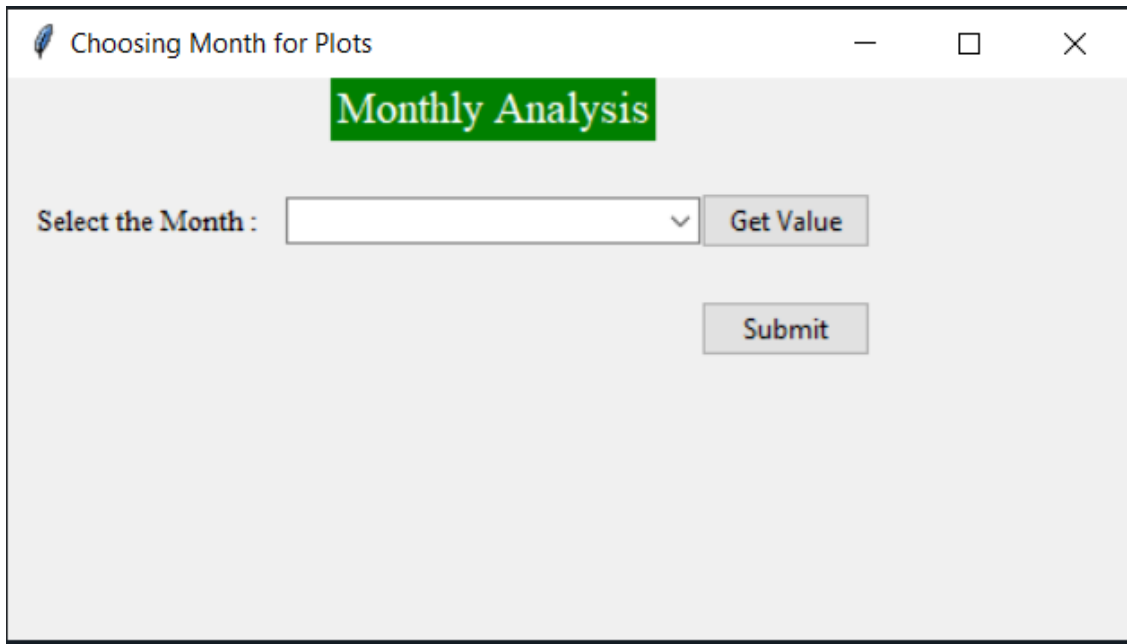
Entering Values into Miscellaneous table

Miscellaneous Table	
Month	<input type="text"/>
Pet Care	<input type="text"/>
Hobbies	<input type="text"/>
Gifts and Donations	<input type="text"/>
Vacation	<input type="text"/>
Other Miscellaneous Expenses	<input type="text"/>
<input type="button" value="Submit"/>	

Miscellaneous Table	
Month	January
Pet Care	1000
Hobbies	2000
Gifts and Donations	3000
Vacation	4000
Other Miscellaneous Expenses	5000
<input type="button" value="Submit"/>	

Values were added to the Miscellaneous Table

Choosing Month for Plots



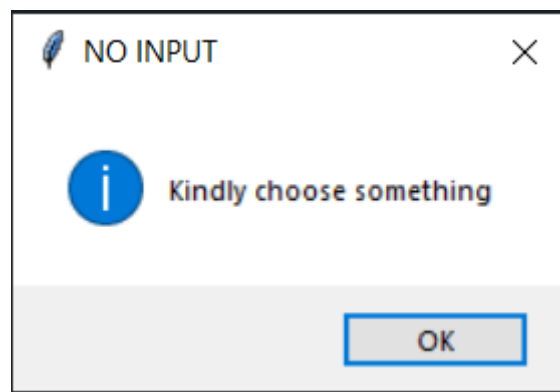
Choosing Month for Plots

Monthly Analysis

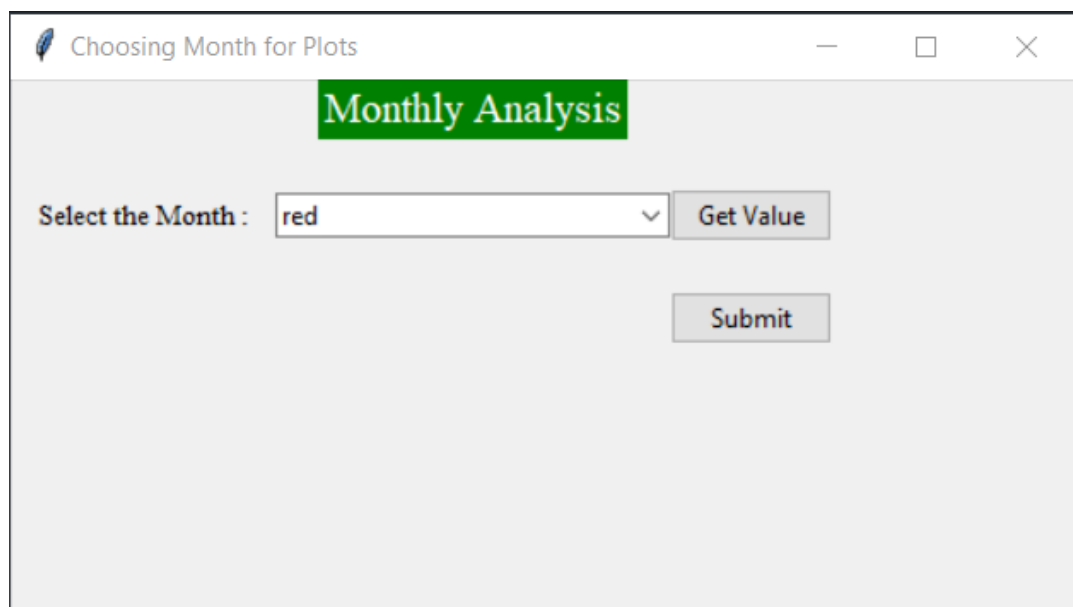
Select the Month : Get Value

Submit

Getting value without selecting any month.



Not choosing a valid month.

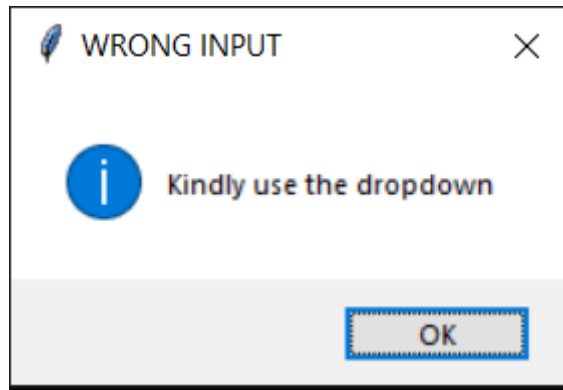


Choosing Month for Plots

Monthly Analysis

Select the Month : red Get Value

Submit



Selecting proper month.

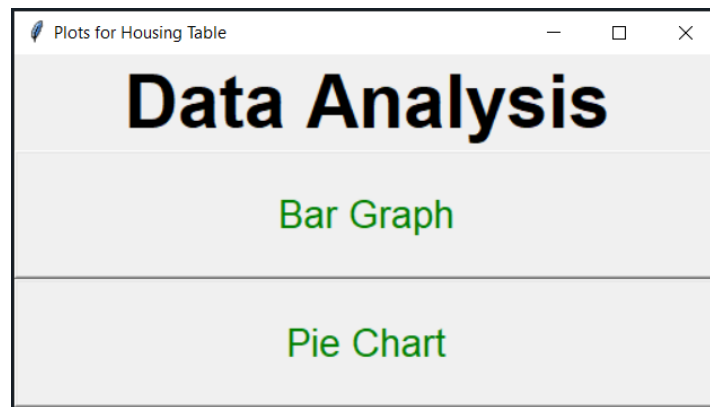
A window titled "Choosing Month for Plots" with a green header "Monthly Analysis". It features a dropdown menu labeled "Select the Month:" with "January" selected, a "Get Value" button, and a "Submit" button.

January is chosen for checking Plots

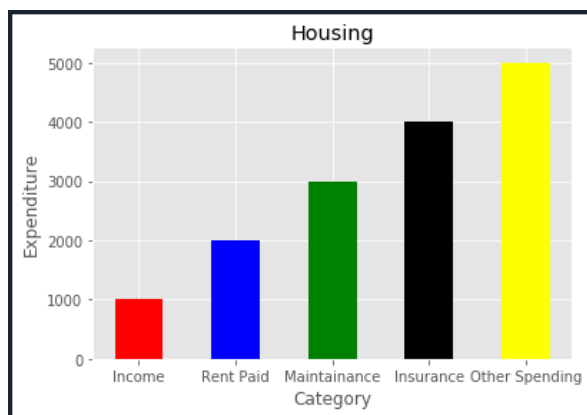
Plot Menu after submitting the month.

A window titled "Plot Menu" showing a list of categories: Housing, Transportation, Living Expenses, Health Care, Child Care, and Miscellaneous.

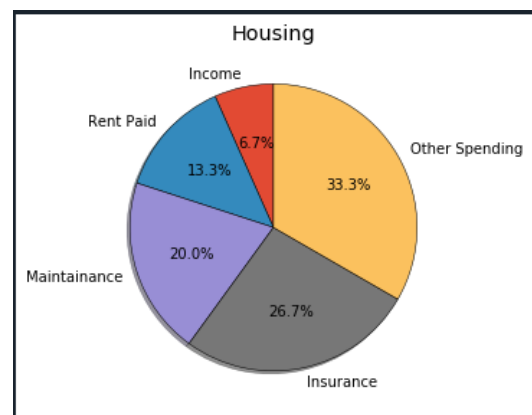
Clicking Housing in Plot Menu.



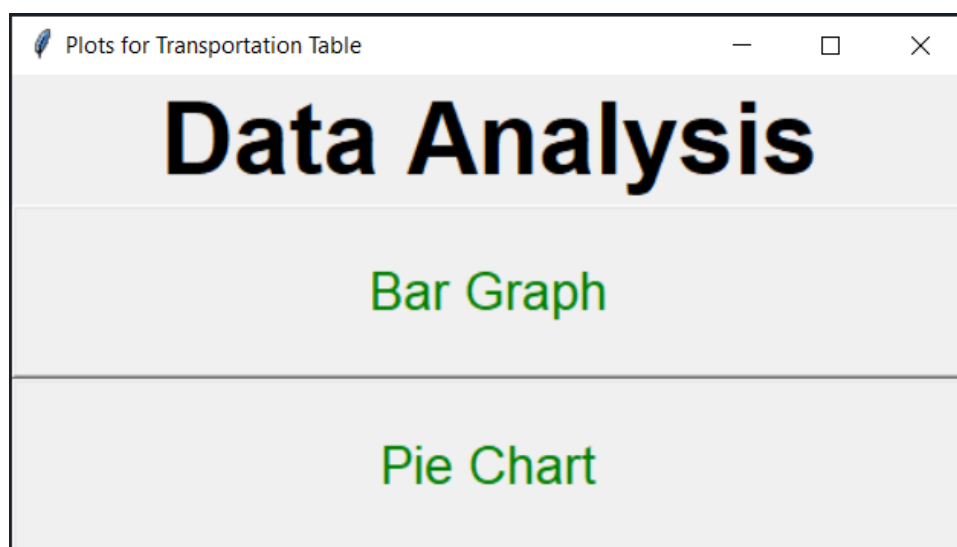
Bar Graph



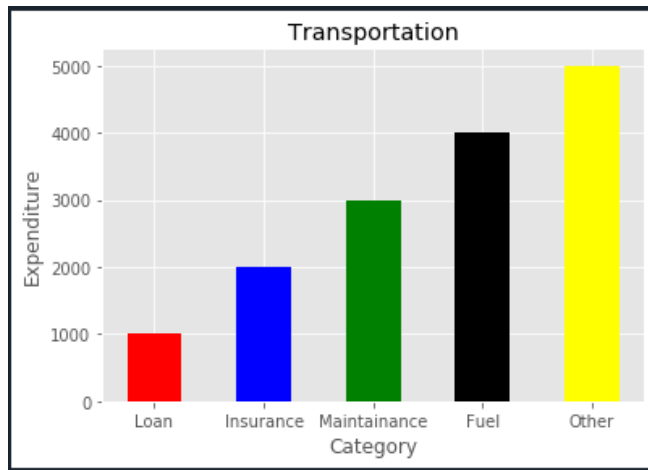
Pie Chart



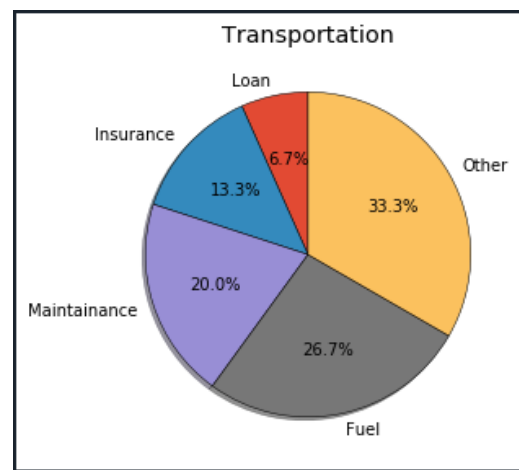
Clicking Transportation in Plot Menu



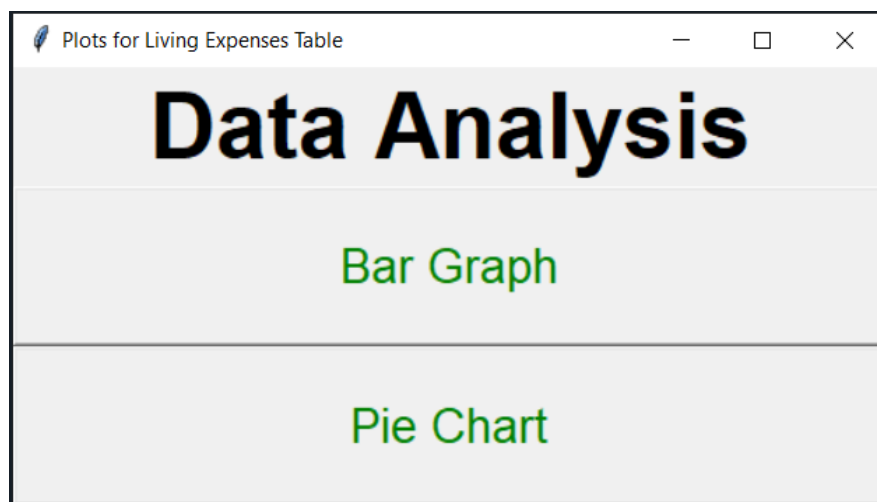
Bar Graph



Pie Chart



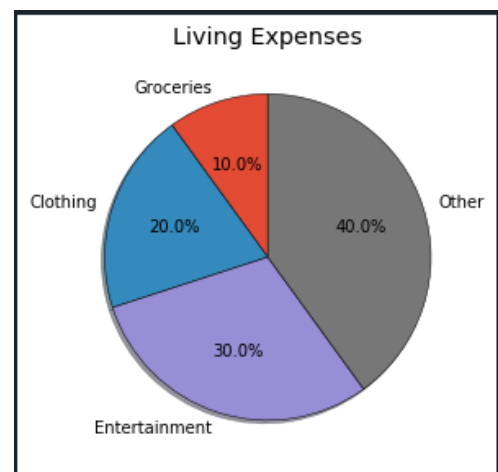
Clicking Living Expenses in Plot Menu



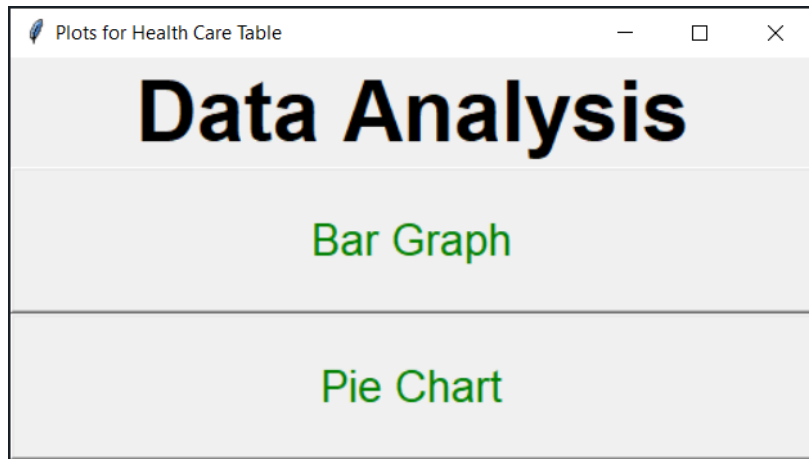
Bar Graph



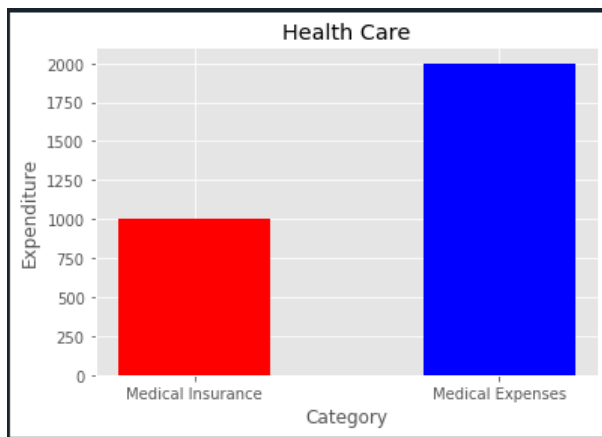
Pie Chart



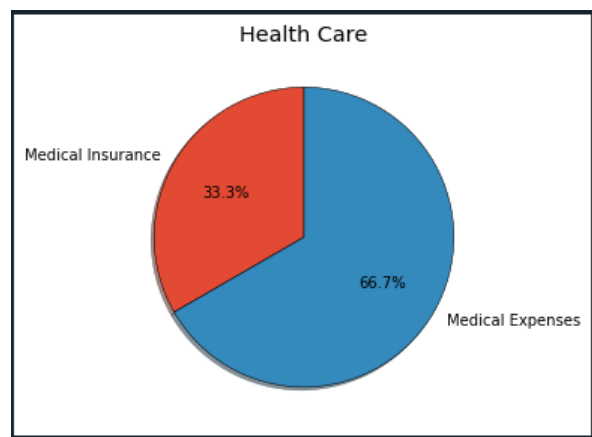
Clicking Health Care in Plot Menu



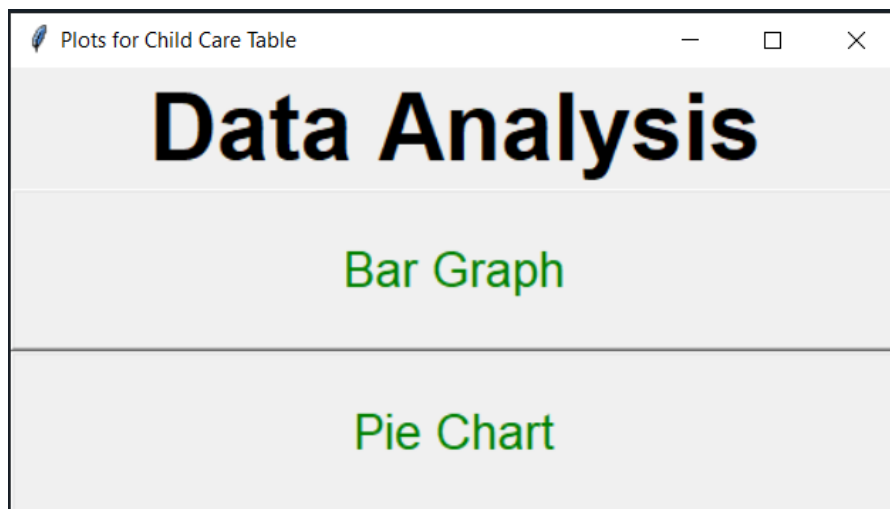
Bar Graph



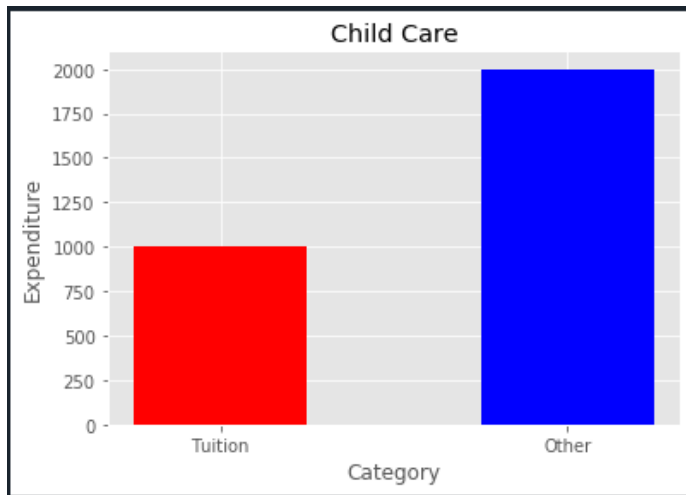
Pie Chart



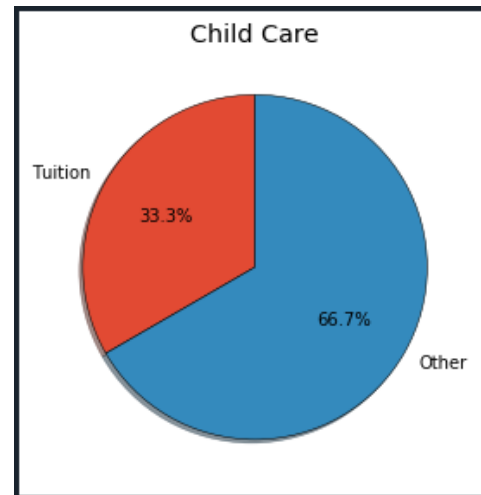
Clicking Child Care in Plot Menu



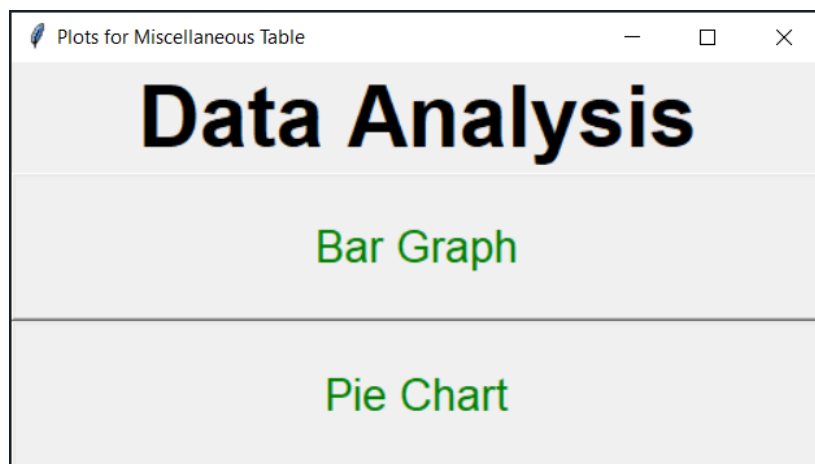
Bar Graph



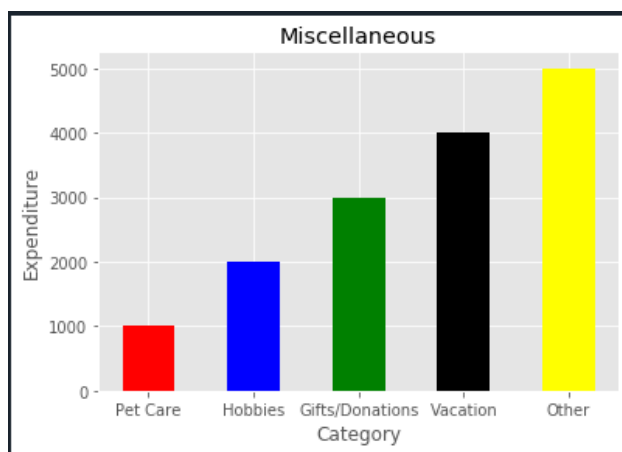
Pie Chart



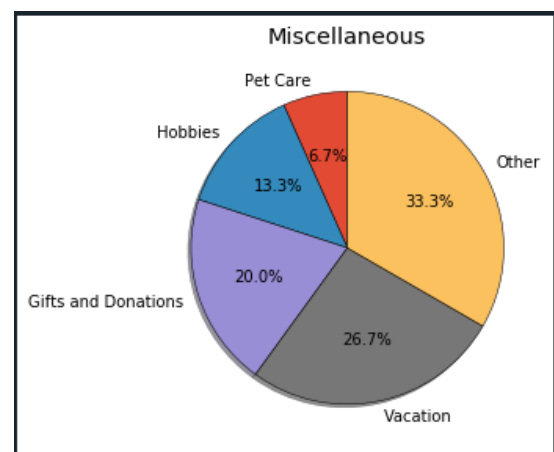
Clicking Miscellaneous in Plot Menu



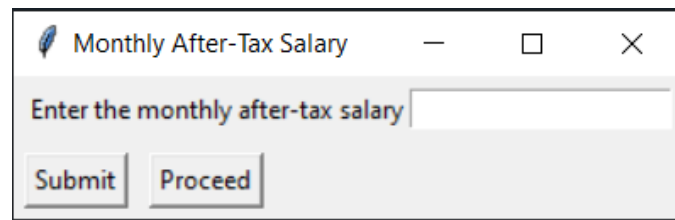
Bar Graph



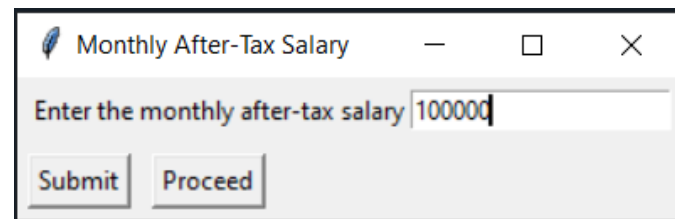
Pie Chart



Creating Budget



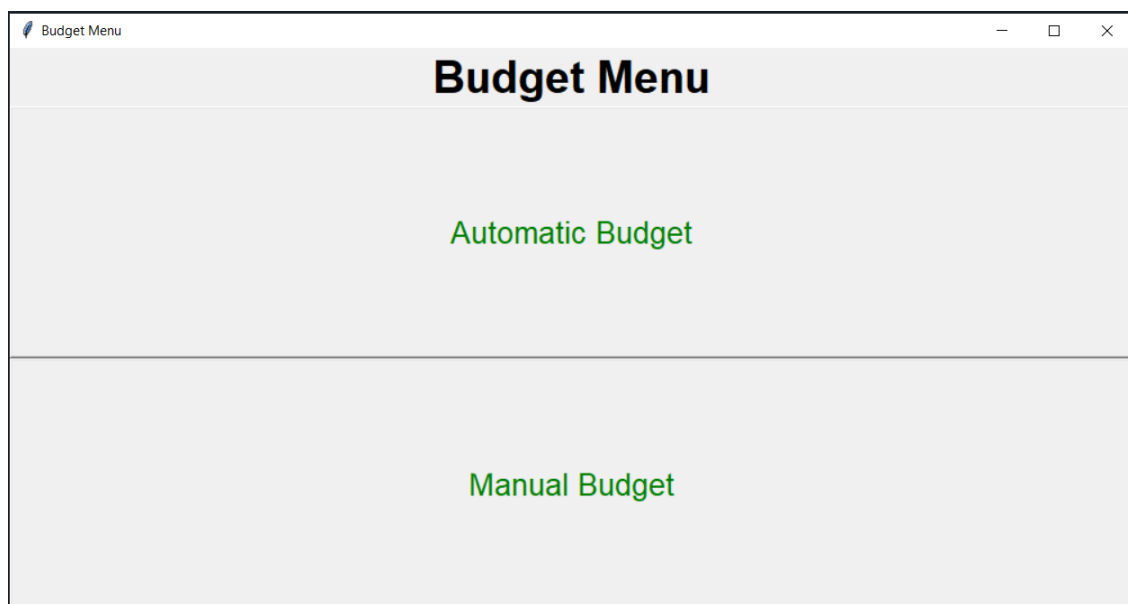
A small window titled "Monthly After-Tax Salary" with a feather icon, minus, maximize, and close buttons. It contains a text input field with the placeholder "Enter the monthly after-tax salary" and two buttons: "Submit" and "Proceed".



A small window titled "Monthly After-Tax Salary" with a feather icon, minus, maximize, and close buttons. The text input field now contains the value "100000". The "Submit" and "Proceed" buttons are still present.

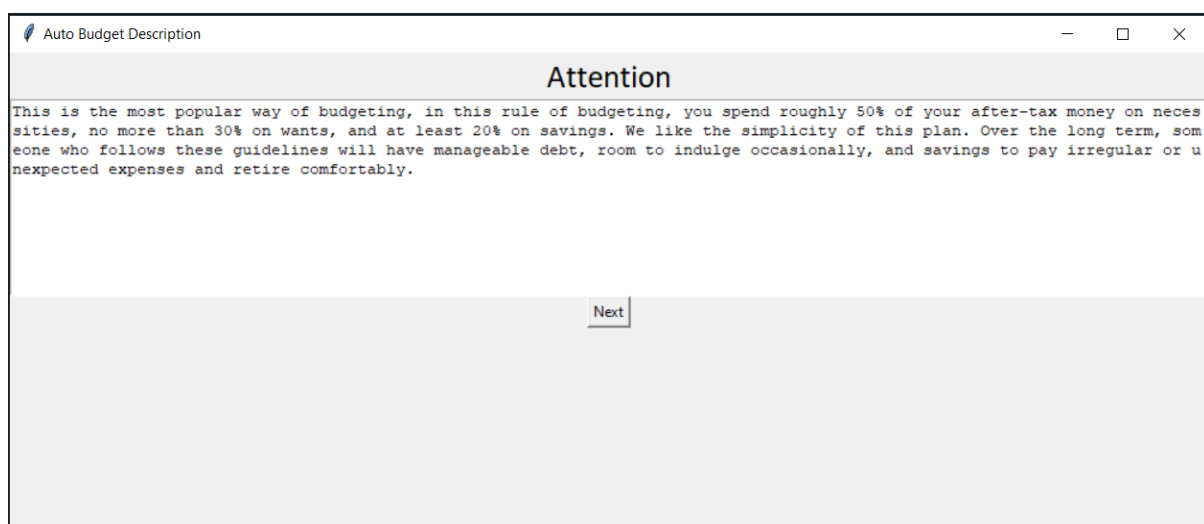
Monthly After Tax Income : 100000.0

Budget Menu after clicking proceed.



A window titled "Budget Menu" with a feather icon and window controls. It displays two options in green text: "Automatic Budget" and "Manual Budget".

Auto Budget Description after clicking Auto Budget in Budget Menu.

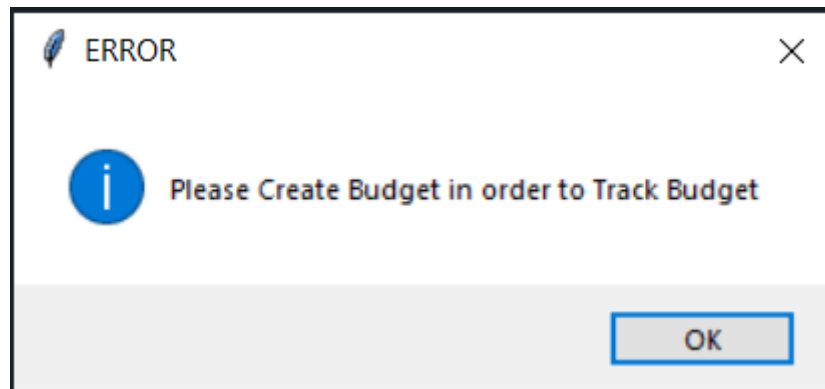


A window titled "Auto Budget Description" with a feather icon and window controls. It has a header "Attention" and a paragraph of text: "This is the most popular way of budgeting, in this rule of budgeting, you spend roughly 50% of your after-tax money on necessities, no more than 30% on wants, and at least 20% on savings. We like the simplicity of this plan. Over the long term, someone who follows these guidelines will have manageable debt, room to indulge occasionally, and savings to pay irregular or unexpected expenses and retire comfortably." At the bottom, there is a "Next" button.

Auto Budget Calculation for Necessities, Wants and Savings after clicking next in the description.

```
According to the autobudget :  
Amount of money you should be spending on necessities - ₹ 50000.0  
Amount of money you should be spending on wants - ₹ 30000.0  
Amount of money you should be saving - ₹ 20000.0
```

Clicking Track Budget in Main Menu without Creating Budget.



Tracking Budget after setting Budget as Auto Budget.

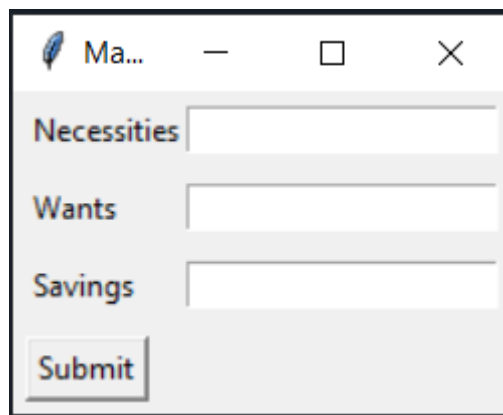
A dialog box titled "Choosing Month for Tracking Budget" with standard window controls. The main content area has a green header "Monthly Analysis". Below it, there is a label "Select the Month :" followed by a dropdown menu showing "January" and a "Get Value" button. At the bottom right, there is a "Submit" button.

```
January is chosen for Tracking Budget
```

Python Output after tracking budget with auto budget.

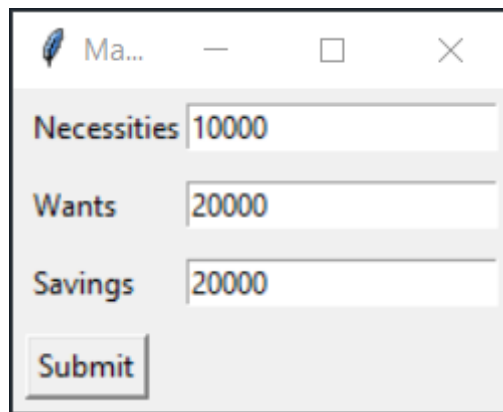
```
January is chosen for Tracking Budget  
  
You have remained under the budget on necessities  
  
You have exceeded the limit on the amount to be spent on wants  
Amount exceeded : 18000.0  
  
You have exceeded the limit on the amount to be saved  
Amount exceeded : 21000.0
```

Window to manually enter values for budget after clicking the Manual Budget in Budget Menu.

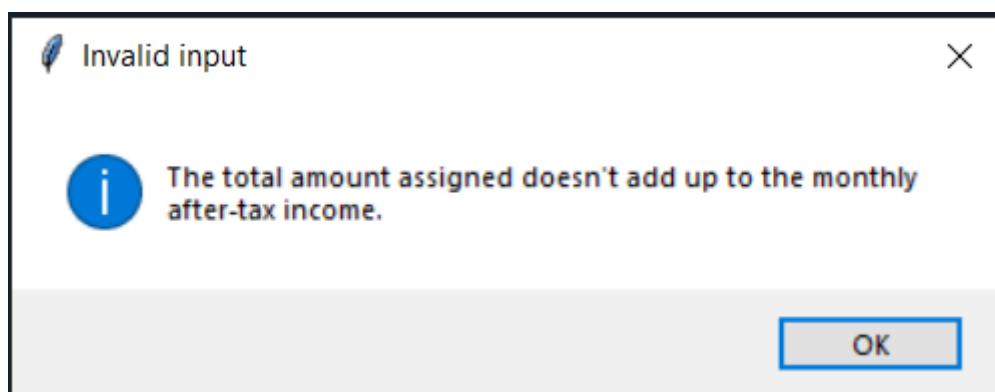


A screenshot of a window titled 'Ma...' with standard window controls. It contains three text input fields labeled 'Necessities', 'Wants', and 'Savings'. Below these fields is a 'Submit' button.

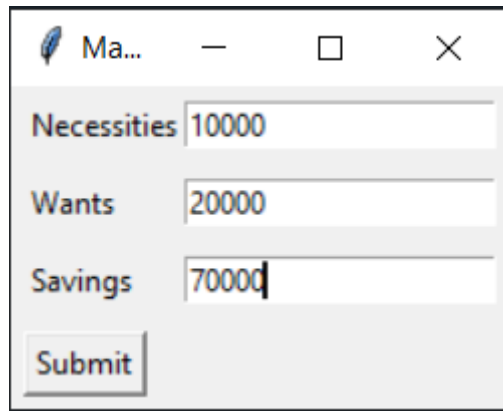
Entering Values which do not add up to the Monthly after-tax salary.



A screenshot of the same 'Manual Budget' window. The 'Necessities' field contains '10000', the 'Wants' field contains '20000', and the 'Savings' field contains '20000'. The 'Submit' button is still visible.

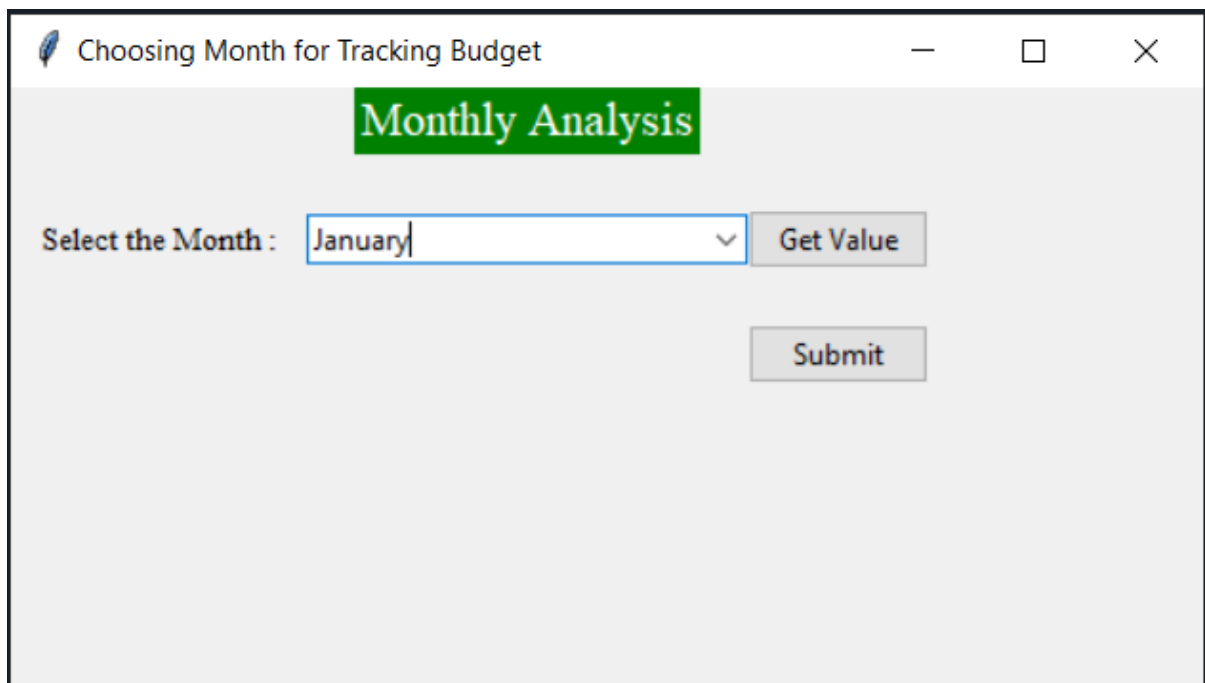


Entering valid values which adds up to the monthly after-tax salary.



A screenshot of a web application window titled "Ma...". It contains three input fields: "Necessities" with the value "10000", "Wants" with the value "20000", and "Savings" with the value "70000". Below these fields is a "Submit" button.

Tracking Budget after setting Budget as Manual Budget.



A screenshot of a web application window titled "Choosing Month for Tracking Budget". It features a green header "Monthly Analysis". Below this, there is a label "Select the Month :" followed by a dropdown menu showing "January" and a "Get Value" button. At the bottom right, there is a "Submit" button.

January is chosen for Tracking Budget

Manual Budget details for Necessities, Wants and Savings after clicking submit in the window.

```
According to the manual budget :  
Amount of money you should be spending on necessities - ₹ 10000.0  
Amount of money you should be spending on wants - ₹ 20000.0  
Amount of money you should be saving - ₹ 70000.0  
You're all done, we will help you keep track of this budget and let you know as you progress.
```

Python Output after tracking budget with manual budget.

January is chosen for Tracking Budget

You have exceeded the limit on the amount to be spent on necessities
Amount exceeded : 1000.0

You have exceeded the limit on the amount to be spent on wants
Amount exceeded : 28000.0

You have remained under the budget on for amount saved

MySQL Output

```
mysql> select * from child_care;
+-----+-----+-----+
| Month | Tuition | Other_Expenses_for_Child |
+-----+-----+-----+
| January | 1000 | 2000 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from health_care;
+-----+-----+-----+
| Month | Medical_Insurance | Medical_Expenses |
+-----+-----+-----+
| January | 1000 | 2000 |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from housing;
+-----+-----+-----+-----+-----+-----+
| Month | Income_from_Houses | Rent_Paid | House_Maintainance | House_Insurance | Other_Spending_on_houses |
+-----+-----+-----+-----+-----+-----+
| January | 1000 | 2000 | 3000 | 4000 | 5000 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from living_expenses;
+-----+-----+-----+-----+-----+
| Month | Groceries | Clothing | Entertainment | Other_Living_Expenses |
+-----+-----+-----+-----+-----+
| January | 1000 | 2000 | 3000 | 4000 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from miscellaneous;
+-----+-----+-----+-----+-----+-----+
| Month | Pet_Care | Hobbies | Gifts_and_Donations | Vacation | Other_Miscellaneous_Expenses |
+-----+-----+-----+-----+-----+-----+
| January | 1000 | 2000 | 3000 | 4000 | 5000 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> select * from transportation;
+-----+-----+-----+-----+-----+-----+
| Month | Vehicle_Loan | Vehicle_Insurance | Vehicle_Maintainance | Fuel_Expenses | Other_Transportation_Expenses |
+-----+-----+-----+-----+-----+-----+
| January | 1000 | 2000 | 3000 | 4000 | 5000 |
+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

FURTHER DEVELOPMENT AREA

- **Advanced Accessibility :** An app can be create that goes with the software which could provide cloud storage and easy SMS notifications about any updates.
- **High Data Security :** To have a better security system to handle and manage sensitive user data.
- **Inbuilt Compliance :** The software could calculate salaries, tax deductions, incentives and bonuses etc by itself, sort out the issues related to payments and deductions and records it digitally into its database.
- **Database Creation :** A database can be created in MySQL through python itself so as to input values from the user.

BIBLIOGRAPHY

- ✓ <https://www.geeksforgeeks.org/>
- ✓ https://www.tutorialspoint.com/python/python_gui_programming.htm
- ✓ <https://stackoverflow.com>
- ✓ <https://github.com>
- ✓ <https://docs.python.org/3/library/tkinter.html>
- ✓ https://www.python-course.eu/python_tkinter.php
- ✓ https://www.w3schools.com/python/matplotlib_bars.asp
- ✓ https://www.tutorialspoint.com/matplotlib/matplotlib_bar_plot.htm