APPENDIX B

```
# Import the sqlite3 module for working with SQLite databases
import sqlite3

# Import necessary modules for the GUI interface
from tkinter import *
from tkinter import messagebox
import tkinter as tk

# Import the matplotlib module for creating plots
import matplotlib.pyplot as plt

# Import the tempfile module for working with temporary files
import tempfile

# Import the sys module for interacting with the Python interpreter
import sys

# Import the subprocess module for running external commands
import subprocess
```

loginpage()

```
def loginpage():
  # Get the username and password entered by the user
  username = e1.get()
  password = e2.get()

  if username == "" and password == "":
    messagebox.showinfo("", "Try typing Admin for both username and password")

  elif username == "Admin" and password == "Admin":
    # If the credentials are correct, call the secondwindow function and destroy the root window
    secondwindow()
```

```
        root.destroy()

    else:
        # Clear the username and password fields
        e1.delete(0, END)
        e2.delete(0, END)
        messagebox.showerror('ERROR', 'Invalid credentials')


# Define functions to handle button hover effects
def on_enter(btn):
    btn.config(bg="#e81753")


def on_leave(btn):
    btn.config(bg="#E9765B")


# Style dictionary for buttons
newlook = {"bg": "#E9765B", "fg": "#ffffff",
           "borderwidth": 0}
root = Tk()
root.configure(bg='#2d2d2d')
root.title("Loginfrontpage")
root.winfo_screenwidth()
root.winfo_screenheight()
root.geometry("450x400")

l1 = Label(root, font=("comic sans ms", 15), bg="#2d2d2d", fg="#E9765B", text="Username")
l1.place(x=100, y=208)
l2 = Label(root, font=("comic sans ms", 15), bg="#2d2d2d", fg="#E9765B", text="Password")
l2.place(x=100, y=288)
login_name = StringVar()
e1 = Entry(root, font=("comic sans ms", 12), bg="#2d2d2d", fg="White",
textvariable=login_name)
e1.place(x=250, y=215, height=22, width=125)
login_data = StringVar()
e2 = Entry(root, font=("comic sans ms", 12), bg="#2d2d2d", fg="White",
textvariable=login_data, show="*")
e2.place(x=250, y=297, height=22, width=125)
```

```
b1 = Button(root, text="Login", font=("comic sans ms", 10, "bold"), width=12,
command=loginpage, **newlook)
b1.place(x=270, y=350)
b3 = Button(root, text="Exit Window", font=("comic sans ms", 10, "bold"), width=12,
command=root.destroy, **newlook)
b3.place(x=95, y=350)
Label(root, width=60, font=("comic sans ms", 30), bg="#2d2d2d", fg="#E9765B",
text="LOGIN").place(x=-500,
                                                                    y=100)
# Bind button hover effects to each button
b1.bind("<Enter>", lambda event, btn=b1: on_enter(btn))
b1.bind("<Leave>", lambda event, btn=b1: on_leave(btn))
b3.bind("<Enter>", lambda event, btn=b3: on_enter(btn))
b3.bind("<Leave>", lambda event, btn=b3: on_leave(btn))
```

secondwindow()

```
def secondwindow():
  gui = Tk()
  gui.title("My Window")
  gui.geometry("300x250")
  gui.configure(bg="#2c2f33")

  btn_style = {"bg": "#E9765B", "fg": "#ffffff",
          "borderwidth": 0}

  def tracker_command():
    trackerwindow()
    gui.destroy()

  expense_tracker_btn = tk.Button(gui, text="Expense Tracker", width=20, height=2,
command=tracker_command,
                        **btn_style)
  expense_tracker_btn.pack(pady=10)

  def visualize_command():
    visualize()
    gui.destroy()
```

```
  visualize_btn = tk.Button(gui, text="Visualize", width=20, height=2,
command=visualize_command, **btn_style)
  visualize_btn.pack(pady=10)

  def todolist_command():
    todolist()
    gui.destroy()

  todo_list_btn = tk.Button(gui, text="todo List", width=20, height=2,
command=todolist_command, **btn_style)
  todo_list_btn.pack(pady=10)

  def closeapp_command():
    gui.destroy()

  close_app_btn = tk.Button(gui, text="Close", width=20, height=2,
command=closeapp_command, **btn_style)
  close_app_btn.pack(pady=10)

  def on_enter(btn):
    btn.config(bg="#e81753")

  def on_leave(btn):
    btn.config(bg="#E9765B")

  expense_tracker_btn.bind("<Enter>", lambda event, btn=expense_tracker_btn: on_enter(btn))
  expense_tracker_btn.bind("<Leave>", lambda event, btn=expense_tracker_btn: on_leave(btn))
  visualize_btn.bind("<Enter>", lambda event, btn=visualize_btn: on_enter(btn))
  visualize_btn.bind("<Leave>", lambda event, btn=visualize_btn: on_leave(btn))
  todo_list_btn.bind("<Enter>", lambda event, btn=todo_list_btn: on_enter(btn))
  todo_list_btn.bind("<Leave>", lambda event, btn=todo_list_btn: on_leave(btn))
  close_app_btn.bind("<Enter>", lambda event, btn=close_app_btn: on_enter(btn))
  close_app_btn.bind("<Leave>", lambda event, btn=close_app_btn: on_leave(btn))
```

todolist()

```
def todolist():
  newlook = {"bg": "#E9765B", "fg": "#ffffff",
        "borderwidth": 0}
  conn = sqlite3.connect('tasks.db')
```

```
c = conn.cursor()

c.execute('''
        CREATE TABLE IF NOT EXISTS tasks (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            task TEXT,
            completed INTEGER
        )
    ''')
conn.commit()

bg_color = '#2d2d2d'
fg_color = '#ffffff'
btn_bg_color = '#444444'
btn_fg_color = '#ffffff'

gui = tk.Tk()
gui.title('todo List')
gui.geometry('500x400')
gui.config(bg=bg_color)

def add_task():
    task = task_entry.get()
    if task:
        c.execute('INSERT INTO tasks (task, completed) VALUES (?, ?)', (task, 0))
        conn.commit()
        task_listbox.insert(tk.END, task)
        task_entry.delete(0, tk.END)

def delete_task():
    selection = task_listbox.curselection()
    if selection:
        task = task_listbox.get(selection)
        c.execute('DELETE FROM tasks WHERE task=?', (task,))
        conn.commit()
        task_listbox.delete(selection)

def complete_task():
    selection = task_listbox.curselection()
    if selection:
```

```
        task = task_listbox.get(selection)
        c.execute('UPDATE tasks SET completed=1 WHERE task=?', (task,))
        conn.commit()
        task_listbox.itemconfig(selection, fg='grey')

    task_entry = tk.Entry(gui, bg=bg_color, fg=fg_color)
    task_entry.pack(fill=tk.X, padx=10, pady=10)

    add_button = tk.Button(gui, text='Add', width=16, command=add_task, **newlook)
    add_button.pack(padx=10, pady=5)
    add_button.bind("<Enter>", lambda event, a=add_button: on_touch(a, '#e81753'))
    add_button.bind("<Leave>", lambda event, a=add_button: on_alone(a, '#E9765B'))

    task_listbox = tk.Listbox(gui, bg=bg_color, fg=fg_color, selectbackground=btn_bg_color,
                              selectforeground=btn_fg_color)
    task_listbox.pack(fill=tk.BOTH, padx=10, pady=10)

    delete_button = tk.Button(gui, text='Delete', width=16, command=delete_task, **newlook)
    delete_button.pack(side=tk.LEFT, padx=10, pady=5)
    delete_button.bind("<Enter>", lambda event, a=delete_button: on_touch(a, '#e81753'))
    delete_button.bind("<Leave>", lambda event, a=delete_button: on_alone(a, '#E9765B'))

    complete_button = tk.Button(gui, text='Complete', width=16, command=complete_task,
**newlook)
    complete_button.pack(side=tk.RIGHT, padx=10, pady=5)
    complete_button.bind("<Enter>", lambda event, a=complete_button: on_touch(a, '#e81753'))
    complete_button.bind("<Leave>", lambda event, a=complete_button: on_alone(a, '#E9765B'))
    back_button = tk.Button(gui, text="Back", width=16, command=lambda: (secondwindow(),
gui.destroy()), **newlook)
    back_button.pack(side="bottom")
    back_button.bind("<Enter>", lambda event, a=back_button: on_touch(a, '#e81753'))
    back_button.bind("<Leave>", lambda event, a=back_button: on_alone(a, '#E9765B'))

    close_button = tk.Button(gui, text='Close', width=16, command=gui.destroy, **newlook)
    close_button.pack(side=tk.BOTTOM, padx=5, pady=5)
    close_button.bind("<Enter>", lambda event, a=close_button: on_touch(a, '#e81753'))
    close_button.bind("<Leave>", lambda event, a=close_button: on_alone(a, '#E9765B'))

    c.execute('SELECT task, completed FROM tasks')
    tasks = c.fetchall()
```

```
    for task in tasks:
        task_listbox.insert(tk.END, task[0])
        if task[1] == 1:
            task_listbox.itemconfig(tk.END, fg='grey')
```

trackerwindow()

```
def trackerwindow():
    def connectdata():
        conn = sqlite3.connect("Trackerdata.db")
        cur = conn.cursor()
        cur.execute(
            "CREATE TABLE IF NOT EXISTS Trackertable(Number INTEGER "
            "PRIMARY KEY,"
            "itemname TEXT,date TEXT,cost TEXT)")
        conn.commit()
        conn.close()

    connectdata()

    def datainput(itemname, date, cost):
        conn = sqlite3.connect("Trackerdata.db")
        cur = conn.cursor()
        cur.execute("INSERT INTO Trackertable VALUES(NULL,?,?,?)", (itemname, date, cost))
        conn.commit()
        conn.close()

    def view():
        conn = sqlite3.connect("Trackerdata.db")
        cur = conn.cursor()
        cur.execute("SELECT * FROM Trackertable")
        rows = cur.fetchall()
        conn.commit()
        conn.close()
        return rows

    def insertitems():
        item = vari1_itemname.get()
        date = vari1_date.get()
        cost = vari1_cost.get()
```

```
    replace = cost.replace('.', '', 1)
    count = date.count('-')
    if item == "" or date == "" or cost == "":
        messagebox.showerror('ERROR', "Fill in Something.. a name?.. maybe a number or
date?")
    elif len(date) != 10 or count != 2:
        messagebox.showerror('ERROR', " Maybe try using this format dd-mm-yyyy")
    elif not replace.isdigit():
        messagebox.showerror('ERROR', "Just use numbers ")
    else:
        datainput(item, date, cost)
        e1.delete(0, END)
        e2.delete(0, END)
        e3.delete(0, END)
    box1.delete(0, END)
    viewallitems()

  def update():

    date = vari1_date.get()
    count = date.count('-')
    selected_row = box1.curselection()
    if not selected_row:
        messagebox.showerror('ERROR', "Select a row to update")
        return

    itemname = vari1_itemname.get()
    date = vari1_date.get()
    cost = vari1_cost.get()

    if not all([itemname, date, cost]):
        messagebox.showerror('ERROR', "Fill in all fields")
        return

    try:
        cost = int(cost)
    except ValueError:
        messagebox.showerror('ERROR', "Cost must be a number")
        return
```

```
        if len(date) != 10 or count != 2:
            messagebox.showerror('ERROR', "Use dd-mm-yyyy format for date")
            return

        selected_num = box1.get(selected_row).split()[0]

        conn = sqlite3.connect("Trackerdata.db")
        cur = conn.cursor()
        cur.execute("UPDATE Trackertable SET itemname=?, date=?, cost=? WHERE Number=?",
                    (itemname, date, cost, selected_num))
        conn.commit()
        conn.close()

        viewallitems()
        e1.delete(0, END)
        e2.delete(0, END)
        e3.delete(0, END)

    def search(itemname="", date="", cost=""):
        conn = sqlite3.connect("Trackerdata.db")
        cur = conn.cursor()
        cur.execute("SELECT *FROM Trackertable WHERE itemname=? OR date=? OR cost=?",
(itemname, date, cost))
        rows = cur.fetchall()
        conn.commit()
        conn.close()
        return rows

    def search_item():
        box1.delete(0, END)
        box1.insert(END, "No.   NAME     DATE      COST")
        rows = search(vari1_itemname.get(), vari1_date.get(), vari1_cost.get())
        if len(rows) == 0:
            messagebox.showerror("Error", "Item not found in database.")
            viewallitems()
        else:
            for row in rows:
                j = str(row[0])
                k = str(row[1])
                l = str(row[2])
```

```
            m = str(row[3])
            o = j + "     " + k + "    " + l + "    " + m
            box1.insert(END, o)
    e1.delete(0, END)
    e2.delete(0, END)
    e3.delete(0, END)

def delete():
    selected_row = box1.curselection()
    if not selected_row:
        messagebox.showerror('ERROR', "Select a row to delete")
        return

    selected_num = box1.get(selected_row).split()[0]

    conn = sqlite3.connect("Trackerdata.db")
    cur = conn.cursor()
    cur.execute("DELETE FROM Trackertable WHERE Number=?", (selected_num,))
    conn.commit()
    conn.close()

    box1.delete(selected_row)
    e1.delete(0, END)
    e2.delete(0, END)
    e3.delete(0, END)
    messagebox.showinfo('SUCCESS', "Selected item deleted")

def deletealldata():
confirm = messagebox.askyesno("Confirmation", "Are you sure you want to delete all data?")
if confirm:
    conn = sqlite3.connect("Trackerdata.db")
    cur = conn.cursor()
    cur.execute("DELETE FROM Trackertable")
    conn.commit()
    conn.close()
    box1.delete(0, END)
    messagebox.showinfo('Successful', 'Vanished!!')
    viewallitems()
```

```
  def report():
      gui = Tk()
      gui.title("Report")
      gui.configure(bg="#2c2f33")
      gui.geometry("200x150")
      newlook = {"bg": "#E9765B", "fg": "#ffffff",
                 "borderwidth": 0}

      def expensereport():
          printdata()

      expense_report_btn = tk.Button(gui, text="Expense report", width=19,
command=expensereport, **newlook)
      expense_report_btn.pack(pady=10)
      expense_report_btn.bind("<Enter>", lambda event, a=expense_report_btn: on_touch(a,
'#e81753'))
      expense_report_btn.bind("<Leave>", lambda event, a=expense_report_btn: on_alone(a,
'#E9765B'))

      def total():
          totalsum()

      total_btn = tk.Button(gui, text="Total Spent", width=16, command=total, **newlook)
      total_btn.pack(pady=10)
      total_btn.bind("<Enter>", lambda event, a=total_btn: on_touch(a, '#e81753'))
      total_btn.bind("<Leave>", lambda event, a=total_btn: on_alone(a, '#E9765B'))

      def back():
          gui.destroy()
          trackerwindow()

      back_btn = tk.Button(gui, text="Back", width=16, command=back, **newlook)
      back_btn.pack(pady=10)
      back_btn.bind("<Enter>", lambda event, a=back_btn: on_touch(a, '#e81753'))
      back_btn.bind("<Leave>", lambda event, a=back_btn: on_alone(a, '#E9765B'))

  def printdata():
      gui = Tk()
      gui.title("Print Data")
      gui.geometry("400x400")
```

```
    data_listbox = Listbox(gui)
    data_listbox.pack(expand=True, fill=BOTH)

    rows = view()
    for row in rows:
        data_listbox.insert(END, row)

    print_button = Button(gui, text="Print", command=lambda: print_to_printer(data_listbox))
    print_button.pack()
    close_button = Button(gui, text="Close", command=gui.destroy)
    close_button.pack()

    def print_to_printer(data_listbox):

        temp_file = tempfile.NamedTemporaryFile(delete=False)

        for item in data_listbox.get(0, END):
            temp_file.write(bytes(str(item) + "\n", "utf-8"))

        temp_file.close()

        if sys.platform == "win32":
            subprocess.call(["notepad.exe", "/p", temp_file.name])
        else:
            messagebox.showerror("Printing is not supported on this platform.")

def totalsum():
    conn = sqlite3.connect("Trackerdata.db")
    cur = conn.cursor()
    cur.execute("SELECT SUM(cost) FROM Trackertable")
    gui = Tk()
    gui.configure(bg='#2d2d2d')
    gui.title("Amount Spent")
    gui.geometry("350x300")
    tsum = cur.fetchone()
    newlook = {"bg": "#E9765B", "fg": "#ffffff",
               "borderwidth": 0}
    k = str(tsum[0])
    Label(gui, width=60, font=("comic sans ms", 20), bg="#2d2d2d", fg="Green",
```

```
            text="TOTAL SPENT " + k + "$", ).place(x=-325,
                                                   y=100)

        close_btn = Button(gui, text="Close", font=("comic sans ms", 16, "bold"),
                           width=14, command=gui.destroy, **newlook)
        close_btn.place(x=70, y=230)

        close_btn.bind("<Enter>", on_enter)
        close_btn.bind("<Leave>", on_leave)
        conn.commit()
        conn.close()
        return tsum

    def viewallitems():
        box1.delete(0, END)
        box1.insert(END, "No.   NAME     DATE      COST")
        for row in view():
            j = str(row[0])
            k = str(row[1])
            l = str(row[2])
            m = str(row[3])
            n = j + "     " + k + "    " + l + "    " + m
            box1.insert(END, n)

    def on_enter(e):
        e.widget['background'] = '#e81753'

    def on_leave(e):
        e.widget['background'] = '#E9765B'

    gui = Tk()
    newlook = {"bg": "#E9765B", "fg": "#ffffff",
               "borderwidth": 0}
    gui.title("EXPENSE TRACKER")
    gui.configure(bg='#2d2d2d')
    gui.geometry("1000x600")

    Label(gui, width=60, height=7, font=("comic sans ms", 35), bg="#2d2d2d",
text="").place(x=450, y=60)
```

```
  Label(gui, width=100, height=10, font=("comic sans ms", 35), bg="#2d2d2d",
text="").place(x=-455, y=410)
  Label(gui, font=("comic sans ms", 15), bg='#2d2d2d', fg="#E9765B",
text="Name").place(x=10, y=80)
  vari1_itemname = StringVar()
  e1 = Entry(gui, font=("comic sans ms", 15), bg="#2d2d2d", fg="White",
textvariable=vari1_itemname)
  e1.place(x=150, y=80, height=27, width=165)
  Label(gui, font=("comic sans ms", 17), bg='#2d2d2d', fg="#E9765B", text="Date").place(
     x=10, y=130)
  vari1_date = StringVar()
  e2 = Entry(gui, font=("comic sans ms", 17), bg="#2d2d2d", fg="White",
textvariable=vari1_date)
  e2.place(x=150, y=130, height=27, width=165)
  Label(gui, font=("comic sans ms", 17), bg='#2d2d2d', fg="#E9765B",
text="Cost($)").place(x=10, y=180)
  vari1_cost = StringVar()
  e3 = Entry(gui, font=("comic sans ms", 15), bg="#2d2d2d", fg="White",
textvariable=vari1_cost)
  e3.place(x=150, y=180, height=27, width=165)

  scroll = Scrollbar(gui)
  scroll.place(x=435, y=100, height=396, width=20)
  box1 = Listbox(gui, height=12, width=34, selectbackground="grey", highlightthickness=5,
            font=("Times New Roman", 20)
            , bg='#E9765B', fg='White', yscrollcommand=scroll.set)
  box1.place(x=455, y=100)
  scroll.config(command=box1.yview)
  Label(gui, width=60, font=("comic sans ms", 35), fg="#E9765B", bg="#2d2d2d",
text="EXPENSE  TRACKER").place(x=-360,
                                                                    y=0)
  add_item_btn = Button(gui, text="Add", font=("comic sans ms", 13), width=30,
command=insertitems, **newlook)
  add_item_btn.place(x=60, y=250)

  add_item_btn.bind("<Enter>", on_enter)
  add_item_btn.bind("<Leave>", on_leave)

  searchs_item_btn = Button(gui, text="Search", font=("comic sans ms", 13), width=30,
command=search_item, **newlook)
```

```
   searchs_item_btn.place(x=60, y=300)

   searchs_item_btn.bind("<Enter>", on_enter)
   searchs_item_btn.bind("<Leave>", on_leave)

   view_item_btn = Button(gui, text="View all", font=("comic sans ms", 13), width=30,
command=viewallitems,
                          **newlook)
   view_item_btn.place(x=60, y=400)

   view_item_btn.bind("<Enter>", on_enter)
   view_item_btn.bind("<Leave>", on_leave)

   update_btn = Button(gui, text="Update", font=("comic sans ms", 13), width=30,
                       command=update, **newlook)
   update_btn.place(x=60, y=450)

   update_btn.bind("<Enter>", on_enter)
   update_btn.bind("<Leave>", on_leave)
   withno_item_btn = Button(gui, text="Delete Selected", font=("comic sans ms", 13),
command=delete, **newlook)
   withno_item_btn.place(x=40, y=350)

   withno_item_btn.bind("<Enter>", on_enter)
   withno_item_btn.bind("<Leave>", on_leave)

   alldet_item_btn = Button(gui, text="Delete all", font=("comic sans ms", 13), width=13,
command=deletealldata,
                         **newlook)
   alldet_item_btn.place(x=250, y=350)

   alldet_item_btn.bind("<Enter>", on_enter)
   alldet_item_btn.bind("<Leave>", on_leave)

   Report_item_btn = Button(gui, text="Report", font=("comic sans ms", 13), width=25,
                            command=lambda: (report(), gui.destroy()),
                            **newlook)
   Report_item_btn.place(x=450, y=530)

   Report_item_btn.bind("<Enter>", on_enter)
```

```
  Report_item_btn.bind("<Leave>", on_leave)

  closer_item_btn = Button(gui, text="Close", font=("comic sans ms", 13), width=10,
command=gui.destroy, **newlook)
  closer_item_btn.place(x=850, y=530)

  closer_item_btn.bind("<Enter>", on_enter)
  closer_item_btn.bind("<Leave>", on_leave)

  Back_item_btn = Button(gui, text="Back", font=("comic sans ms", 13), width=10,
                    command=lambda: (secondwindow(), gui.destroy()), **newlook)
  Back_item_btn.place(x=60, y=530)

  Back_item_btn.bind("<Enter>", on_enter)
  Back_item_btn.bind("<Leave>", on_leave)
```

visualize()

```
def on_touch(button, color):
  button['bg'] = color


def on_alone(button, color):
  button['bg'] = '#E9765B'


def visualize():
  newlook = {"bg": "#E9765B", "fg": "#ffffff",
          "borderwidth": 0}
  gui = tk.Tk()
  gui.title("Visualize")
  gui.configure(bg='#2d2d2d')
  gui.geometry("300x300")

  pie_chart_button = tk.Button(gui, text="Pie Chart", height=1, width=18,
command=visualize_pie_chart, **newlook)
  pie_chart_button.pack(side="left")
  pie_chart_button.bind("<Enter>", lambda event, b=pie_chart_button: on_touch(b, '#e81753'))
```

```
  pie_chart_button.bind("<Leave>", lambda event, b=pie_chart_button: on_alone(b, '#E9765B'))

  bar_graph_button = tk.Button(gui, text="Bar Graph", height=1, width=18,
command=visualize_bar_graph, **newlook)
  bar_graph_button.pack(side="right")
  bar_graph_button.bind("<Enter>", lambda event, b=bar_graph_button: on_touch(b, '#e81753'))
  bar_graph_button.bind("<Leave>", lambda event, b=bar_graph_button: on_alone(b,
'#E9765B'))

  back_button = tk.Button(gui, text="  Back  ", command=lambda: (secondwindow(),
gui.destroy()), **newlook)
  back_button.pack(side="bottom")
  back_button.bind("<Enter>", lambda event, b=back_button: on_touch(b, '#e81753'))
  back_button.bind("<Leave>", lambda event, b=back_button: on_alone(b, '#E9765B'))

  close_button = tk.Button(gui, text="  Close  ", command=gui.destroy, **newlook)
  close_button.pack(side=tk.BOTTOM, padx=5, pady=10)
  close_button.bind("<Enter>", lambda event, b=close_button: on_touch(b, '#e81753'))
  close_button.bind("<Leave>", lambda event, b=close_button: on_alone(b, '#E9765B'))


def visualize_pie_chart():
  global cur, conn
  try:
    conn = sqlite3.connect("Trackerdata.db")
    cur = conn.cursor()
    cur.execute("SELECT cost, itemname FROM Trackertable")
    rows = cur.fetchall()
    conn.commit()
  except sqlite3.Error as e:
    messagebox.showerror('error', f"Error connecting to database: {e}")
    return

  finally:
    cur.close()
    conn.close()

  if not rows:
    messagebox.showerror('error', "No data found in the database.")
    return
```

```
    costs = []
    products = []
    for row in rows:
        products.append(row[1])
        costs.append(int(row[0]))

    plt.pie(costs, labels=products, autopct='%1.1f%%', startangle=90)
    plt.axis('equal')
    plt.title('Distribution of expenses')
    plt.show()


def visualize_bar_graph():
    global conn, cur
    try:
        conn = sqlite3.connect("Trackerdata.db")
        cur = conn.cursor()
        cur.execute("SELECT cost, itemname FROM Trackertable")
        rows = cur.fetchall()
        conn.commit()
    except sqlite3.Error as e:
        messagebox.showerror('error', f"Error connecting to database: {e}")
        return

    finally:
        cur.close()
        conn.close()

    if not rows:
        messagebox.showerror('error', "No data found in the database.")
        return

    costs = []
    products = []
    for row in rows:
        products.append(row[1])
        costs.append(int(row[0]))

    plt.bar(products, costs)
```

```
    plt.title('Distribution of expenses')
    plt.xlabel('Items')
    plt.ylabel('Cost')
    plt.show()


root.resizable(False, False)
root.mainloop()
```