

**BINUS UNIVERSITY**

**Assignment Cover Letter**

**(Individual Work)**

**BINUS INTERNATIONAL**

Student Information: Surname Given Names Student ID Number

1. Jevon Danaristo 2440043591

Course Code : COMP6056

Class : L1BC

Major : Computer Science

Title of Assignment : Driver Attendance Log

Type of Assignment : Final Project

Submission Pattern

Due Date : Submission Date:

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating** BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality** By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

(Name of Student)

Jevon Danaristo

**“Driver Attendance Log”**

**Name: Jevon Danaristo**

**ID: 2440043591**

1. **Program Description**

Driver Attendance Log is an application that records data of drivers activity then processed the data into useful information and calculate the driver’s wage. The main purpose of this application is to monitor the activity of individual driver and the overall productivity of the entire group. The application is built entirely in python and its library.

1. **Program Structure**

The application is made of several page class that contains widgets, a main class and a drop-down class.

1. **Application Site Map**

Menu Page

Salary Page

Variable Page

Read Page

Data Page

Load Data

Variables

Add Data

Read Log

Save Variable

Read Activity

Read Salary

Load Data

Save Data

Read Production

1. **Menu Page**

**Description**

Menu is the first page to be displayed when running the application. Menu is used to navigate to the other pages.

**Interface**



**Source Code (interface)**

class StartPage(tk.Frame):

    def \_\_init\_\_(self,parent,frameName,butonName,destination):

        tk.Frame.\_\_init\_\_(self,parent)

        self.configure(bg="gray")

        self.button1 = tk.Button(self, height=2, width=15, text ="Read data",

command=lambda: frameName.show\_frame("pageIDRead"))

        self.button1.pack(anchor="w", padx=(10,10), pady=(10,10))

        self.button2 = tk.Button(self, height=2, width=15, text ="Add data",

command=lambda: frameName.show\_frame("pageIDAdd"))

        self.button2.pack(anchor="w", padx=(10,10), pady=(10,10))

        self.button3 = tk.Button(self, height=2, width=15, text="Add variable",command=lambda: frameName.show\_frame("pageIDDel"))

        self.button3.pack(anchor="w", padx=(10,10), pady=(10,10))

        self.button4 = tk.Button(self, height=2, width=15,

text="Driver's Salary",command=lambda: frameName.show\_frame("pageIDPay"))

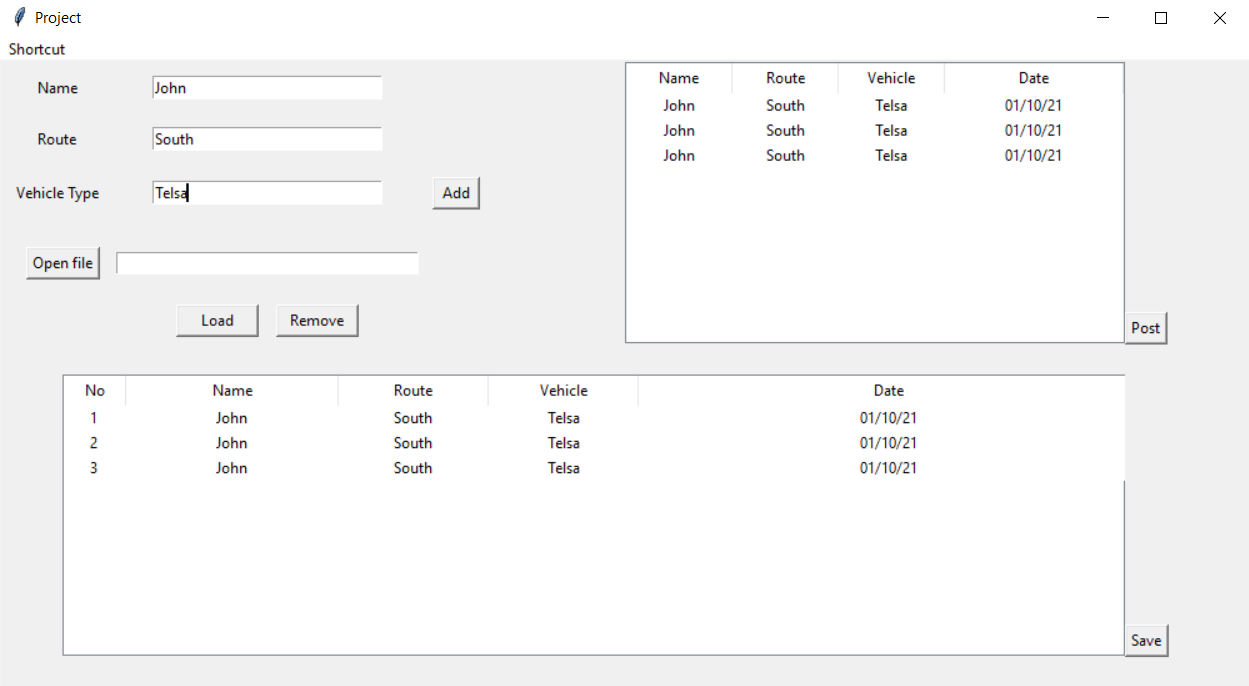
        self.button4.pack(anchor="w", padx=(10,10), pady=(10,10))

1. **Add Data Page**

**Description**

Page to input data of driver’s name, vehicle type, and route taken by the driver. The data will be added to the draft display before being posted into the last display and saved.

**Interface**

****

**Source Code (interface)**

class AddPage(tk.Frame):

    def \_\_init\_\_(self,parent,frameName):

        tk.Frame.\_\_init\_\_(self,parent)

        self.mainframe = Frame(self)

        self.mainframe.place(height=500,width=1000)

        # self.mainframe.configure(bg="red")

        # 1st Frame, data entry

        self.frame1 = Frame(self.mainframe)

        self.frame1.place(relx=0,rely=0,height=250,width=500)

        self.filepath = ""

        self.configure(bg="gray")

        self.label1 = Label(self.frame1, text="Name")

        self.label1.grid(column=0,row=0,padx=(10,10),pady=(10,10))

        self.entry1 = Entry(self.frame1, width=30)

        self.entry1.grid(column=1, row=0,padx=(10,10),pady=(10,10))

        self.label2 = Label(self.frame1, text="Route")

        self.label2.grid(column=0, row=1,padx=(10,10),pady=(10,10))

        self.entry2 = Entry(self.frame1, width=30)

        self.entry2.grid(column=1,row=1,padx=(10,10),pady=(10,10))

        self.label3 = Label(self.frame1,text="Vehicle Type")

        self.label3.grid(column=0,row=2,padx=(10,10),pady=(10,10))

        self.entry3 = Entry(self.frame1, width=30)

        self.entry3.grid(column=1,row=2,padx=(10,10),pady=(10,10))

        self.buttonpath = tk.Button(self.frame1,text="Open file",command=lambd a: Main.windowsDialog(self))

        self.buttonpath.grid(column=0,row=3,padx=(10,0),pady=(20,10))

        self.listBox = Listbox(self.frame1,width=40,height=1)

        self.listBox.grid(column=1,row=3,padx=(0,0),pady=(20,10))

        self.buttonadd = Button(self.frame1,text="Add", width=4, command=lambda: LD.post(self,

        self.entry1.get(),

        self.entry2.get(),

        self.entry3.get()

        )) # Passing all the entry into post function in LoadData File

        self.buttonadd.grid(column=3,row=2, padx=(10,10), pady=(10,10))

        self.buttonload = Button(self.frame1,text="Load", width=8, command=self.Load)

        self.buttonload.grid(column=1,row=4 ,padx=(0,80) ,pady=(10,10))

        self.buttonremove = Button(self.frame1,text="Remove", width=8, command=self.Delete)

        self.buttonremove.grid(column=1,row=4 ,padx=(80,0) ,pady=(10,10))

    # 2nd Frame, Draft table

        self.frame2 = Frame(self.mainframe)

        self.frame2.place(relx=0.5,rely=0,height=250,width=500)

        self.displayDraft = ttk.Treeview(self.frame2, column=("Name","Vehicle","Route","Date"))

        self.displayDraft.place(relheight=0.9, relwidth=0.8)

        self.displayDraft.bind('<Delete>', lambda x: self.delete(self, "displayDraft")) # Bind Delete key to delete function

        self.button = Button(self.frame2,text="Post", command=self.transfer)

        self.button.place(relx=0.8,rely=0.8)

    # 3rd Frame, Actual table

        self.frame3 = Frame(self.mainframe)

        self.frame3.place(relx=0.05, rely=0.5, height=250, width=1000)

        self.display = ttk.Treeview(self.frame3, column=("Name","Vehicle","Route","Date"))

        self.display.place(relheight=0.9, relwidth=0.85)

        self.savebutton = Button(self.frame3, text="Save", command=self.Save)

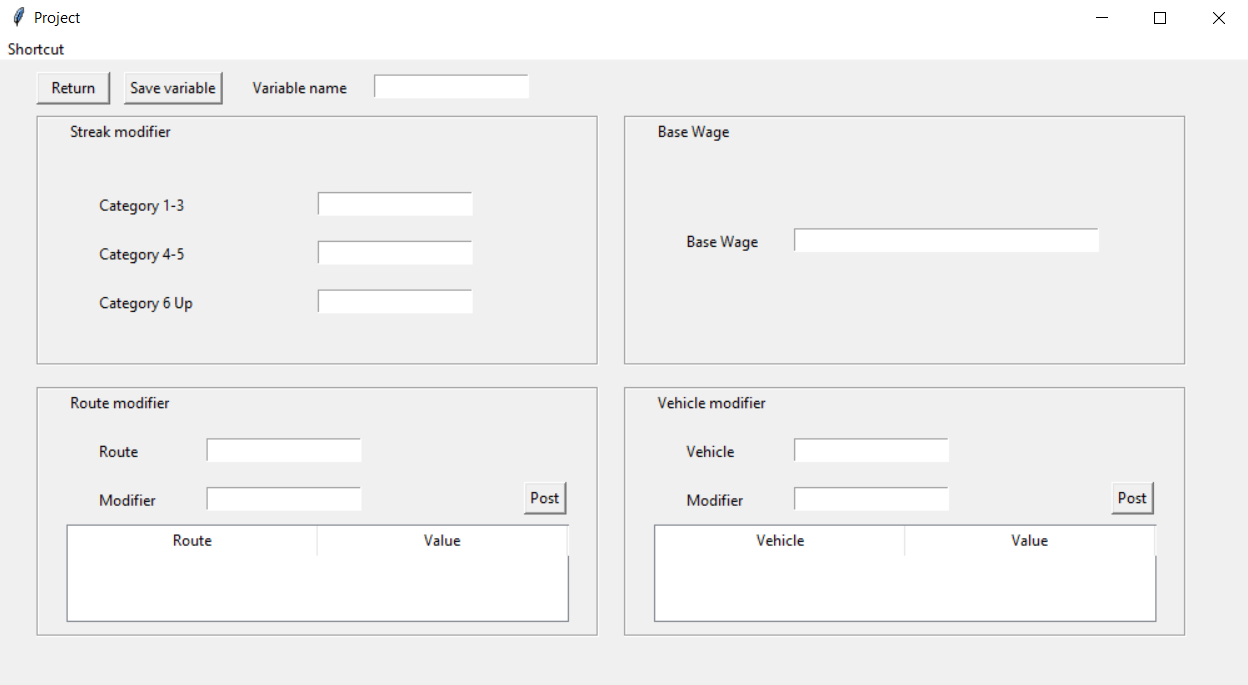
        self.savebutton.place(rely=0.8,relx=0.85)

1. **Add Variable Page**

**Description**

The page to add data to determine driver’s wage. In this page there are entries that will take user input as the modifiers of driver’s wage. The data then will be saved into the system to be used later in the driver’s salary page

**Interface**

****

**Source Code (interface)**

class DelPage(tk.Frame):

    def \_\_init\_\_(self,parent,frameName):

        self.delete = AddPage.delete

        tk.Frame.\_\_init\_\_(self,parent)

        self.mainframe = Frame(self)

        self.mainframe.place(height=500,width=1000)

        # self.mainframe.configure(bg="red")

        self.returnbutton = Button(self.mainframe,text="Return",command=lambda: frameName.show\_frame("pageIDStart"), width=7)

        self.returnbutton.place(relx=0.03,rely=0.015)

        self.savebutton = Button(self.mainframe,text="Save variable", command=lambda: SaveVar(self))

        self.savebutton.place(relx=0.1, rely=0.015)

        self.savenameL = Label(self.mainframe, text="Variable name")

        self.savenameL.place(relx=0.2, rely=0.02)

        self.savenameE = Entry(self.mainframe)

        self.savenameE.place(relx=0.3, rely=0.02)

        # Frame for streak variable

        self.frameStreak= Frame(self.mainframe,borderwidth=2, relief="groove")

        self.frameStreak.place(relx=0.03, rely= 0.085, relwidth=0.45, relheight=0.4)

        self.label = Label(self.frameStreak,text="Streak modifier")

        self.label.place(relx=0.05,rely=0)

        self.label2 = Label(self.frameStreak, text="Category 1-3")

        self.label2.place(relx=0.1, rely=0.3)

        self.label3 = Label(self.frameStreak, text="Category 4-5")

        self.label3.place(relx=0.1, rely=0.5)

        self.label4 = Label(self.frameStreak, text="Category 6 Up")

        self.label4.place(relx=0.1, rely=0.7)

        self.streakA = Entry(self.frameStreak)

        self.streakA.place(relx=0.5, rely=0.3)

        self.streakB = Entry(self.frameStreak)

        self.streakB.place(relx=0.5, rely=0.5)

        self.streakC = Entry(self.frameStreak)

        self.streakC.place(relx=0.5, rely=0.7)

        # Frame for base wage

        self.frameWage = Frame(self.mainframe,borderwidth=2, relief="groove")

        self.frameWage.place(relx=0.50, rely=0.085, relwidth=0.45, relheight=0.4)

        self.label = Label(self.frameWage,text="Base Wage")

        self.label.place(relx=0.05,rely=0)

        self.label2 = Label(self.frameWage,text="Base Wage")

        self.label2.place(rely=0.45,relx=0.1,)

        self.baseWage = Entry(self.frameWage, width=40)

        self.baseWage.place(rely=0.45,relx=0.3)

        # Frame for route variable

        self.frameRoute = Frame(self.mainframe,borderwidth=2, relief="groove")

        self.frameRoute.place(relx=0.03, rely=0.52, relwidth=0.45, relheight=0.4)

        self.label = Label(self.frameRoute,text="Route modifier")

        self.label.place(relx=0.05,rely=0)

        self.label1 = Label(self.frameRoute, text="Route")

        self.label1.place(relx=0.1, rely=0.2)

        self.route = Entry(self.frameRoute)

        self.route.place(relx=0.3, rely=0.2)

        self.label2 = Label(self.frameRoute, text="Modifier")

        self.label2.place(relx=0.1, rely=0.4)

        self.routeM = Entry(self.frameRoute)

        self.routeM.place(relx=0.3, rely=0.4)

        self.buttonPost = Button(self.frameRoute,text="Post",command=lambda: LD.varPostR(self, self.route.get(), self.routeM.get()))

        self.buttonPost.place(relx=0.87, rely=0.38)

        self.varDispR = ttk.Treeview(self.frameRoute, column=("Route","Modifier"))

        self.varDispR.place(relx=0.05, rely=0.55, relheight=0.4, relwidth=0.9)

        self.varDispR.heading('#0', text='Route')

        self.varDispR.heading('#1', text='Value')

        self.varDispR.column('#0', anchor='w')

        self.varDispR.column('#1', anchor='w')

        self.varDispR.bind('<Delete>', lambda x: AddPage.delete(self, "blank", "varDispR"))

        # Frame for vehicle

        self.frameVehicle = Frame(self.mainframe,borderwidth=2, relief="groove")

        self.frameVehicle.place(relx=0.50, rely=0.52, relwidth=0.45, relheight=0.4)

        self.label = Label(self.frameVehicle,text="Vehicle modifier")

        self.label.place(relx=0.05,rely=0)

        self.label1 = Label(self.frameVehicle, text="Vehicle")

        self.label1.place(relx=0.1, rely=0.2)

        self.vehicle = Entry(self.frameVehicle)

        self.vehicle.place(relx=0.3, rely=0.2)

        self.label2 = Label(self.frameVehicle, text="Modifier")

        self.label2.place(relx=0.1, rely=0.4)

        self.vehicleM = Entry(self.frameVehicle)

        self.vehicleM.place(relx=0.3, rely=0.4)

        self.buttonPost = Button(self.frameVehicle,text="Post", command=lambda: LD.varPostV(self, self.vehicle.get(), self.vehicleM.get()))

        self.buttonPost.place(relx=0.87, rely=0.38)

        self.varDispV = ttk.Treeview(self.frameVehicle, column=("Vehicle","Modifier"))

        self.varDispV.place(relx=0.05, rely=0.55, relheight=0.4, relwidth=0.9)

        self.varDispV.heading('#0', text='Vehicle')

        self.varDispV.heading('#1', text='Value')

        self.varDispV.column('#0', anchor='w')

        self.varDispV.column('#1', anchor='w')

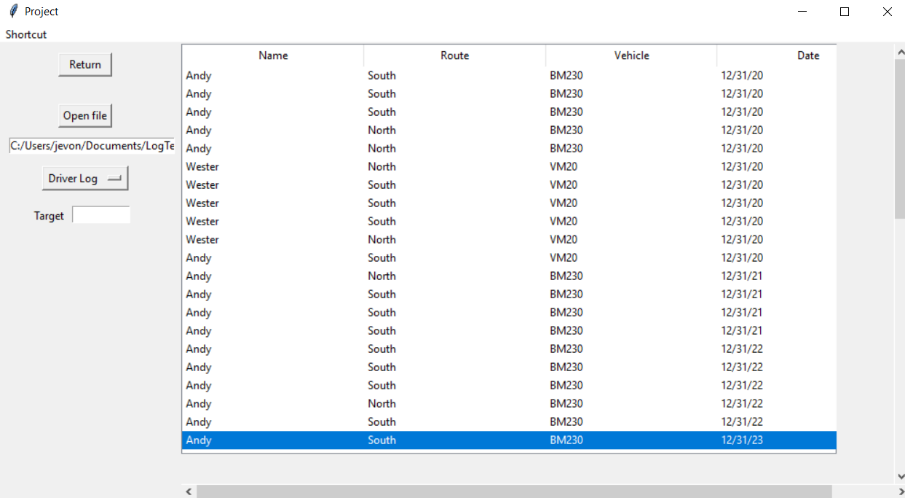
        self.varDispV.bind('<Delete>', lambda x: AddPage.delete(self, "blank", "varDispV"))

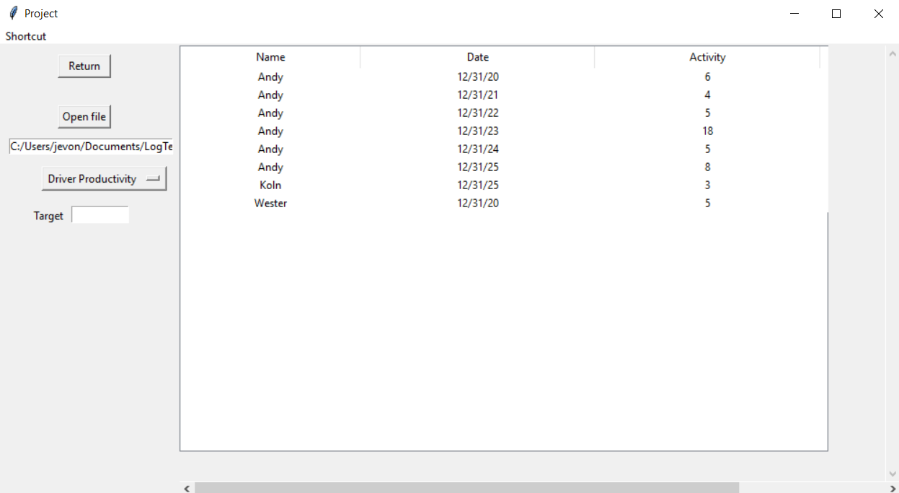
1. **Read Data Page**

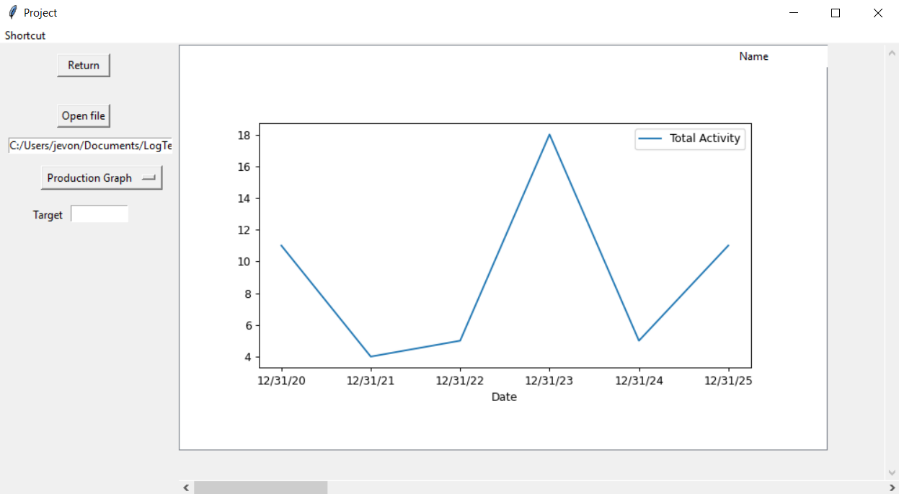
**Description**

The page to read data in three different mode. The first display mode isDriver logwhere it will display complete raw log from a file. The second display mode is driver productivity, This page will display the activity of each individual driver in a day. The last display mode, production graph, is a visualization data of the total activity for each day with a line graph.

**Interface**

****

****

****

**Source Code (interface)**

class ReadPage(tk.Frame):

    def \_\_init\_\_(self,parent,frameName):

        tk.Frame.\_\_init\_\_(self,parent)

        self.mainframe = Frame(self)

        # self.mainframe.configure(bg='red')

        self.mainframe.place(height=500, width=1000)

        self.filepath = ""

        # 1st Frame, interactables

        self.frame1 = Frame(self.mainframe)

        self.frame1.grid(column="0",row="0", ipadx=10, ipady=10,sticky=N)

        self.button = tk.Button(self.frame1,text="Return",command=lambda: frameName.show\_frame("pageIDStart"), width=7)

        self.button.pack(anchor="w", pady=(10,30),padx=(65,10))

        self.buttonpath = tk.Button(self.frame1,text="Open file",command=lambda: Main.windowsDialog(self))

        self.buttonpath.pack(anchor="w",padx=(65,10),pady=(0,10))

        self.listBox = Listbox(self.frame1,width=30,height=1)

        self.listBox.pack(anchor="w",padx=(10,10))

        # Dropdown menu for display option

        self.variable = StringVar(self)

        self.option = ['Driver Log','Driver Productivity','Production Graph']

        self.variable.set('Display Option')

        # Arguments for dropdown menu

        self.optionDP = Frame(self.mainframe)

        self.optionDP.place(relx=0.02, rely=0.35, height=30, width=150)

        self.targetL = tk.Label(self.optionDP, text="Target")

        self.targetL.place(relx=0.1, rely=0.1)

        self.targetE = tk. Entry(self.optionDP, width=10)

        self.targetE.place(relx=0.4, rely=0.1)

        Main.CreateToolTip(self.targetE, text=f"""Takes number of activity desired per choosen interval""")

        popupMenu = OptionMenu(self.frame1, self.variable, \*self.option)

        popupMenu.pack(anchor = "w", padx=(45,10), pady=(10,10))

        def change\_dropdown(\*args):

            LD.Load\_excel\_data(self)

        self.variable.trace('w', change\_dropdown)

        # 2nd Frame, data display

        self.frame2 = Frame(self.mainframe)

        self.frame2.place(relx=0.2, rely=0,width=800, height=500)

        self.display = ttk.Treeview(self.frame2)

        self.display.place(relheight=0.9, relwidth=0.9)

        self.scrollx1 = Scrollbar(self.frame2, orient='horizontal', command=self.display.xview)

        self.scrolly2 = Scrollbar(self.frame2, orient='vertical', command=self.display.yview)

        self.display.configure(xscrollcommand=self.scrollx1.set, yscrollcommand=self.scrolly2.set)

        self.scrollx1.pack(side="bottom", fill="x")

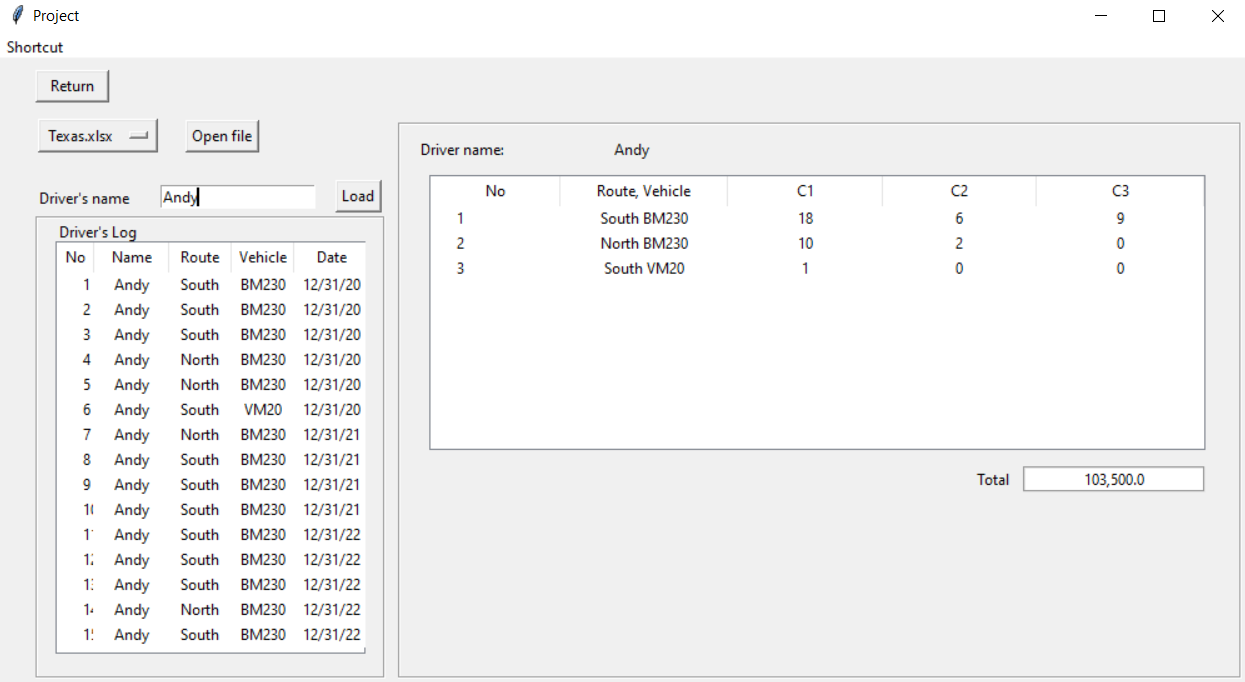
        self.scrolly2.pack(side="right", fill="y")

1. **Driver’s Salary**

**Description**

The driver’s salary page is the page to display the comprehensive information of a driver. This page will display the entire driver’s log and information that will determine the driver’s salary.

**Interface**

****

**Source Code (interface)**

class PayPage(tk.Frame):

    def \_\_init\_\_(self,parent,frameName):

        tk.Frame.\_\_init\_\_(self,parent)

        self.mainframe = Frame(self)

        self.mainframe.place(relx=0, rely=0, width=1000, height=500)

        self.returnbutton = Button(self.mainframe,text="Return",command=lambda: frameName.show\_frame("pageIDStart"), width=7)

        self.returnbutton.place(relx=0.03,rely=0.015)

        arr = os.listdir('Driver-Log-Recorder---Final-Project/Saved variable')

        self.variable = StringVar(self)

        self.option = arr

        self.dropdown = OptionMenu(self.mainframe, self.variable, \*self.option)

        self.dropdown.place(relx= 0.03, rely=0.090, relwidth=0.1)

        def OnChange(\*args):

            LD.Setpath(self)

            arr = os.listdir('Driver-Log-Recorder---Final-Project/Saved variable')

            print(arr)

            menu = self.dropdown["menu"]

            menu.delete(0, "end")

            for string in arr:

                menu.add\_command(label=string,

                                command=lambda value=string: self.variable.set(value))

        self.variable.trace('w', OnChange)

        self.variable.set('Variable')

        self.driverNameL = Label(self.mainframe, text="Driver's name")

        self.driverNameL.place(relx=0.03, rely=0.2)

        self.driverNameE = Entry(self.mainframe)

        self.driverNameE.place(relx=0.13, rely=0.2)

        self.driverNameB = Button(self.mainframe, text="Load", command= lambda: LD.LoadPay(self, self.driverNameE.get()))

        self.driverNameB.place(relx= 0.27, rely=0.191)

        self.buttonpath = tk.Button(self.mainframe,text="Open file",command=lambda: Main.windowsDialog2(self))

        self.buttonpath.place(relx=0.15, rely=0.096)

        # Log

        self.framelog = Frame(self.mainframe, borderwidth=2, relief="groove")

        self.framelog.place(relx=0.03, rely=0.25, width=280, height=370)

        self.logL = Label(self.framelog, text="Driver's Log")

        self.logL.place(relx=0.05,rely=0)

        self.display = ttk.Treeview(self.framelog, column=("Route", "Vehicle", "Date"))

        self.display.place(relx=0.05, rely=0.05, relwidth=0.9, relheight=0.9)

        # Paycheck

        self.framepay = Frame(self.mainframe, borderwidth=2, relief="groove")

        self.framepay.place(relx=0.32, rely=0.1, relwidth=0.675, relheight=0.89)

1. **Library Used**

The following are libraries that is used in this project:

* Pandas 1.1.5

To calculate the data and filtering the data by group

* Matplotlib 3.3.3

To make data visualization of the data in form of graph

* Numpy

To convert the data frame into a nested list form for easier use

* Tkinter

To make the user interface

* Ast

To help the process of loading data into the desired form

* Os

To get the directories of files needed and to get the list of saved files

1. **Problems On the Way**

Common issue that I have is that the form of data that I get from entries or saved cannot be directly processed into the system. Either because of the form of data is incompatible or the data somehow changed along the way. One example of this is when I saved the data from treeview into excel. In the process it turned the list into string and I only realized the problem later in the project. It is not hard to fix it however take some time to find it. Another issue that I have is my own inconsistency. After a while I will have another habit whether it is to change the double quote into single quote, naming the variable with short abbreviation into naming the variable in full detailed name, and changing from camel case into snake case. It does not affect how the code works but it looks messy to me.

1. **Application Algorithm**
2. **Posting data into treeview**

To display data I mainly use treeview since it support screen scroll, a really convenient feature to use when attempting to display large data. To transfer data from entries into treeview, I made a button with command that pass all the entries value into a function.

self.buttonadd = Button(self.frame1,text="Add", width=4, command=lambda: LD.post(self,

        self.entry1.get(),

        self.entry2.get(),

        self.entry3.get()

        ))

The function then will check if all data is filled or not, if one of the inputs is empty the function will raise an error and stop.

def post(self,name,route,vehicle):

    ''' Posting data from draft treeview into final treeview'''

    if not name.strip(): #Raise error if recieved empty string or space only input

        tk.messagebox.showerror("Empty Entry","Name Entry is empty")

        return None

    if not vehicle.strip():

        tk.messagebox.showerror("Empty Entry","vehicle Entry is empty")

        return None

    if not route.strip():

        tk.messagebox.showerror("Empty Entry","Route Entry is empty")

        return None

After checking, the function will make a headers and column for the treeview

    # Your common display insertion (Draft from add data display)

    self.displayDraft['show'] = 'headings'

    self.displayDraft.heading('#1', text='Name')

    self.displayDraft.heading('#2', text='Route')

    self.displayDraft.heading('#3', text='Vehicle')

    self.displayDraft.heading('#4', text='Date')

    self.displayDraft.column('#1', anchor="center", width=85)

    self.displayDraft.column('#2', anchor="center", width=85)

    self.displayDraft.column('#3', anchor="center", width=85)

    self.displayDraft.column('#4', anchor="center", width=143)

Finally it will insert the data into the treeview

    self.displaycontent = self.displayDraft

    self.displaycontent.insert("",index="end",text=f" ",value=(name,route,vehicle,datetime.datetime.now().strftime("%x")))

1. **Saving data to excel file**

* Log data

My method to retain the data after closing the application is to save it in excel file extension. The data that will be saved are displayed in treeview so the initial process will be taking all the data from the treeview.

values = []

    for child in self.display.get\_children():

        values.append(self.display.item(child)["values"] ) # appending the value of treeview, ["values"] will append the value without datatype information

After having the data in form of a list I turn the data into a dataframe since pandas has excelwriter, making things easier to save the data. Then tkinter asksaveasfilename function will get the file path that the user take.

df = pd.DataFrame(values)

    df.columns = ["Name", "Route", "Vehicle", "Date"]

    filepath = asksaveasfilename(defaultextension=".xlsx")

    df.to\_excel(filepath, index=False, header=True)

* Variable data

Saving variable data is completely different since some variables have the modifier as their value and the data must purposely be saved in the application directory.

for child in self.varDispV.get\_children():

        vechileList.append([self.varDispV.item(child)["text"],self.varDispV.item(child)["values"]])

    for child in self.varDispR.get\_children():

        routeList.append([self.varDispR.item(child)["text"],self.varDispR.item(child)["values"]])

    my\_dict = ({

    "Category A": CatA,

    "Category B": CatB,

    "Category C": CatC,

    "Vehicle": vechileList,

    "Route": routeList,

    "Base wage": baseWage

    })

df = pd.DataFrame({ key:pd.Series(value) for key, value in my\_dict.items() })

    print(df)

    path = f"Driver-Log-Recorder---Final-Project/Saved variable/{userfilename}.xlsx"

    df.to\_excel(path)

Some variables only consist of single data while others can have multiple amount of data, to be able to turn those data into data frame without having error because different data length I put all the data into dictionary

1. **Calculating the salary**

To find the of a driver salary, the system need to do three things. First is to know how many combination of variable exist and how many occurrence there are for that combination. Second is to count the streak. And finally is to calculate the salary with simple mathematical operation.

    for liste in tuples:

        row = '-'.join(liste) # Making space character in variable name impossible

        if not row in datastore:

            datastore[row] = 1

        elif row in datastore:

                datastore[row] += 1

For every different combination of route-vehicle-name-date the system will make a new key in the dictionary and give it a value of 1, if the combination already exist it will add 1 to the value. The value will serve as a counter.

for data in datastore.items():

        new\_id = (data[0].split("-"))

        new\_id.append(data[1])

        datastore2.append(new\_id)

After done with the first step then the data will be turned into a neater form of list.

for data in datastore2:

        if (data[1],data[2]) not in datastore3:

            datastore3[data[1],data[2]]={'C1':0, 'C2':0,'C3':0} # < --- Each combination will have that nested dictionary

The data from the list then will be turned into a key (again) and were given nested dictionary containing all streak category inside them.

for data in datastore2:

        print(data[4])

        if data[4] <= 3:

            datastore3[data[1],data[2]]['C1']+=int(data[4]) # if the occurrence count is <= 3 add the value to C1

        elif data[4] > 3 and data[4] <= 5:

            datastore3[data[1],data[2]]['C1']+=3                # if the occurrence is > 3 and <= 5 add 3 to C1 (maximum value C1 could get)

            datastore3[data[1],data[2]]['C2']+=int(data[4]-3)   # and subtract 3 (taken for C1 maximum value)

        elif data[4] > 5:

            datastore3[data[1],data[2]]['C1']+=3                # if the occurrence is > 5 follow the same pattern for C1

            datastore3[data[1],data[2]]['C2']+=2                # add 2 to C1 (maximum value C2 could get)

            datastore3[data[1],data[2]]['C3']+=int(data[4]-5)   # and subtract 5 (taken for C1 and C2 maximum value)

To be able to count the streak a simple for loop data with conditional statement were made. The complete data then will be displayed as an information in the treeview.

for combination in datastore3.items():

        # print(row.values)

        self.payDisplay.insert("",index="end",text=(len(self.payDisplay.get\_children())+1),value=(combination[0],combination[1]['C1'],

        combination[1]['C2'],

        combination[1]['C3']))

    # Paycheck calculation

    C1 = dfvar.loc[0,'Category A']

    C2 = dfvar.loc[0,'Category B']

    C3 = dfvar.loc[0,'Category C']

    Base = dfvar.loc[0,'Base wage']

        # Taking the data from variable xlsx file to be matched with data from display

    vehicleData = {}

    for row in dfvar['Vehicle']:

        try:

            exstring = ast.literal\_eval(row) # Make list that somehow turned into string list again

            vehicleData[exstring[0]] = exstring[1][0]

        except ValueError:

            pass

    routeData = {}

    for row in dfvar['Route']:

        try:

            exstring = ast.literal\_eval(row)

            routeData[exstring[0]] = exstring[1][0]

        except ValueError:

            pass

    displayData = []

    newdata = []

    for child in self.payDisplay.get\_children():

        displayData.append(self.payDisplay.item(child)["values"])

The variable data then will be loaded into the system along with the previous data that was displayed in a treeview.

totalmoney = []

    for data in displayData:

        x = data[0].split()

        newdata.append([x,data[1],data[2],data[3]])

        print(newdata)

    for datavar in newdata: # Base wage x streak count(C1 or C2 or C3) x Route modifier x Vehicle modifier x streak count(C1 or C2 or C3)

        totalmoney.append(float(Base)\*float(C1)\*float(routeData[datavar[0][0]])\*float(vehicleData[datavar[0][1]])\*datavar[1])

        totalmoney.append(float(Base)\*float(C2)\*float(routeData[datavar[0][0]])\*float(vehicleData[datavar[0][1]])\*datavar[2])

        totalmoney.append(float(Base)\*float(C3)\*float(routeData[datavar[0][0]])\*float(vehicleData[datavar[0][1]])\*datavar[3])

    self.Money = ("{:,}".format(sum(totalmoney))) # Summing it and adding coma to the total money

The data will be processed with a mathematical operation. The result then will be appended to a list to be summed