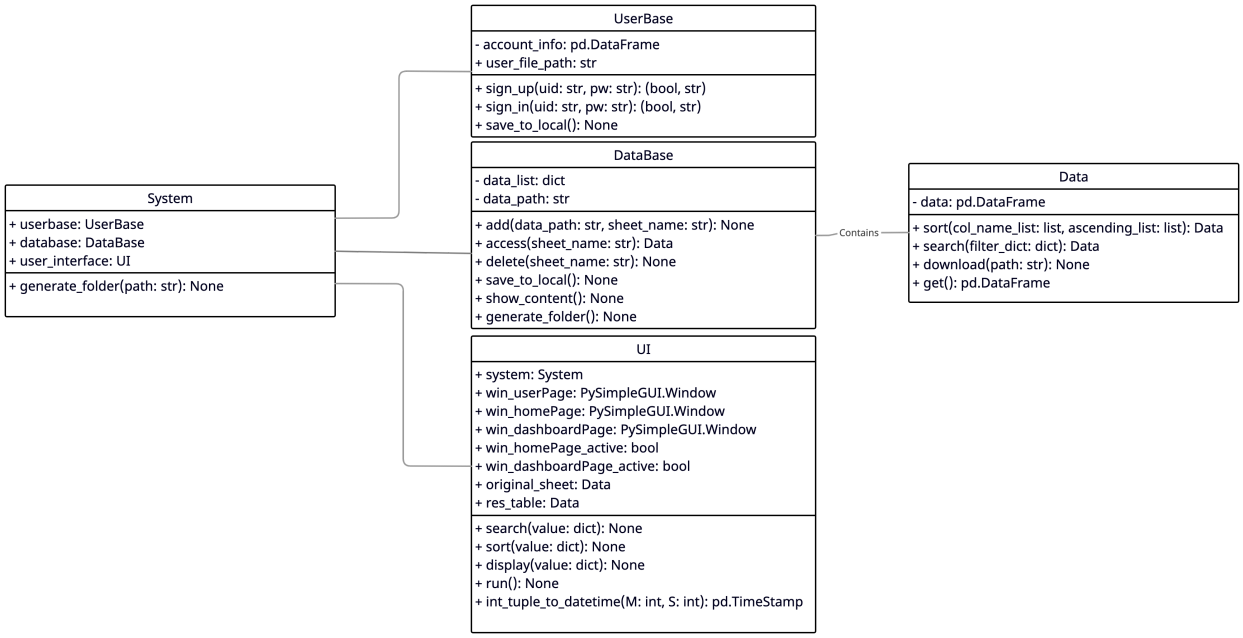
Inquiry System for Taiwan Traffic Data

Authors:

WANG Zihan 20801541, XIE Wenyu 12247636, CHEN Zuozhi 20797609

1. UML and Design Patterns

1.1 UML Class Diagram



1.2 Class Explanation

We explain our class design here. Note that the classes shown in this section have ONLY the framework and the necessary code. The detailed codes are in the appendix.

System Class

In [ ]:
```
class System:

    def __init__(self) -> None:
        ...
        self.userbase = UserBase(lib_path)
        self.database = DataBase(data_path)

        self.user_interface = UI(self)
        self.user_interface.run()

        self.userbase.save_to_local()
        self.database.save_to_local()

    @staticmethod
    def generate_folder(path: str) -> None:
        ...
```

The System class serves as the main function of this program. It contains a constructor and a member function "generate_folder".

- The constructor is the process of our program. It first initializes three objects of UserBase (stores user info), DataBase (stores data) and UI (initialized by passing the current System object, so that the UI can access the userbase and the database). Then, the UI runs to accept input and return output. Finally, when the UI program exits, the userbase and the database save themselves to local library, to memorize the log-in info and the sheet info.
- The "generate_folder" function generates a |directory if it not exists.

UserBase Class

In [ ]:
```
class UserBase:

    def __init__(self, lib_path) -> None:
        self.user_file_path = os.path.join(lib_path, 'account_info.csv')
        if os.path.exists(self.user_file_path):
            self.__account_info = pd.read_csv(self.user_file_path, index_col=0)
            self.__account_info = self.__account_info.astype(str)
```

```
            self.__account_info.reset_index(drop=True, inplace=True)
        else:
            self.__account_info = pd.DataFrame(columns=['uid', 'password'])

    def sign_up(self, uid: str, pw: str) -> tuple:
        ...
    def sign_in(self, uid: str, pw: str) -> tuple:
        ...
    def save_to_local(self):
        ...
```

The UserBase class is used to handle the sign-in, sign-up and the user info storage problems. We use a DataFrame called the account_info to store the user info.

- The constructor loads the local user file info if any.
- The "sign-up" function adds user info into the account_info attribute.
- The "sign-in" function checks the user info with the account_info attribute.
- The "save_to_local" function saves the account_info attribute to local.

DataBase Class

In [ ]:
```
class DataBase:
    def __init__(self, data_path) -> None:
        self.__data_list = {}
        self.__data_path = data_path
        file_list = os.listdir(self.__data_path)
        file_list = [f for f in file_list if f != '.DS_Store']
        if len(file_list) > 0:
            for f in file_list:
                self.add(os.path.join(self.__data_path, f), f.rstrip('.csv'))

    def add(self, data_path: str, sheet_name: str):
        ...
    def access(self, sheet_name: str) -> Data:
        ...
    def delete(self, sheet_name: str):
        ...
    def save_to_local(self):
        ...
    def show_content(self):
        ...
    @staticmethod
    def generate_folder(path: str) -> None:
        ...
```

The DataBase class is used to store and manage the data sheets added by users. We use a dictionary called "data_list" to store the (sheet_name, Data) pair.

- The constructor loads the local data sheet file info if any.
- The "add" function adds a data sheet into the database.
- The "access" function accesses a data sheet in the database.
- The "delete" function deletes a data sheet from the database.
- The "save_to_local" function saves all sheets in the database to local library.
- The "show_content" function inquires all sheet names inside the database.

Data Class

In [ ]:
```
class Data:

    def __init__(self, df: pd.DataFrame = pd.DataFrame()) -> None:
        self.__data = df.copy()
        ...
    def sort(self, col_name_list: list, ascending_list: list):
        ...
    def search(self, filter_dict: dict):
        ...
    def download(self, path):
        ...
    def get(self):
        ...
```

The Data class is used to store a single data sheet. We use a pandas DataFrame called "data" to store the data sheet.

- The "sort" function sorts data.
- The "search" function searches data.
- The "download" function saves data to local.
- The "get" function returns the private member "data".

UI Class

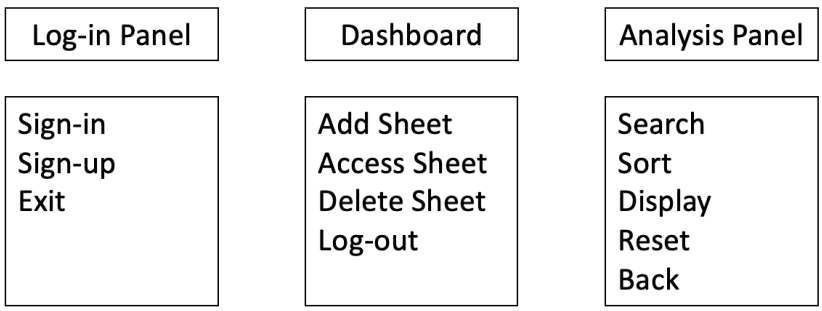
In [ ]:
```
class UI:

    def __init__(self, system) -> None: #system:System obj
        self.system = system
        ...
    def search(self, vals2):
```

```
    ...
def sort(self,vals2):
    ...
def display(self, num_row: int):
    ...
def run(self):
    ...
@staticmethod
def int_tuple_to_datetime(M, S):
    ...
```

The UI class is used to generate a GUI for user to use. We use PySimpleGUI to create this class.

- The "search" function copes with the action when user presses "search" button.
- The "sort" function copes with the action when user presses "sort" button.
- The "display" function copes with the action when user presses "display" button.
- The "run" function starts the process of UI.
- The "int_tuple_to_datetime" function changes integers to datetime.

1.3 User Interface Structure



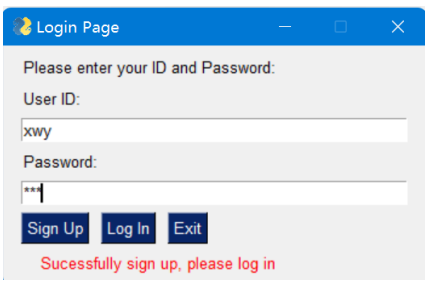
2. Inquiry System Showcase

We build up a powerful system through **pySimpleGUI** Package , which enables us to conduct a workflow of a standard database. There are three panels for this system: Sign in/ Sign up panel, Dashboard panel, analysis panel for user to search or sort and the display . We will provide some showcases in this part.

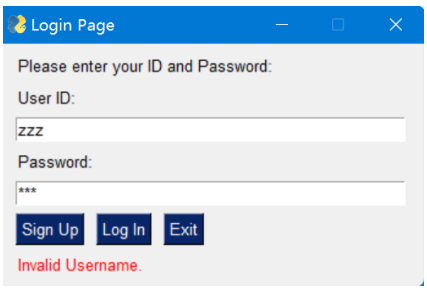
2.1 Sign up and Log in

At the login panel, the user should first choose to sign up in this system which will store the user information in our cache files. Then, the user choose to log in and the system will check the input name and password. If the inputs are matched, the user will enter the dashboard panel; if either user id or password is wrong, warning message will show up:

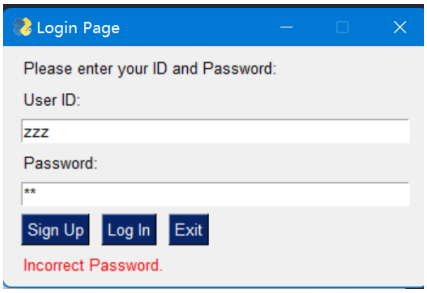
- Sign up



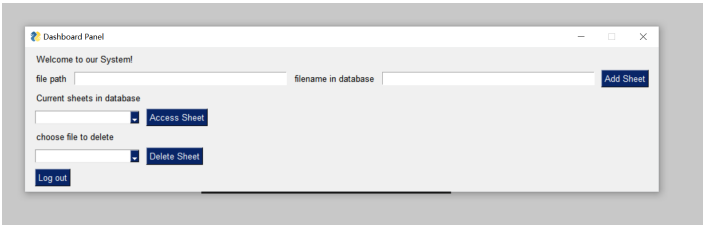
- log in with invalid user name



- log in with incorrect password



- log in successfully, enter the dashboard page



2.2 Dashboard

In the dashboard panel, the user has four selections as follows:

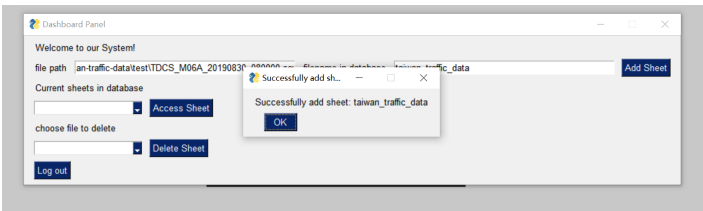
- Add Sheet: add a new sheet to the database. Required input: the absolute path of the csv file, and a sheet name in the database
- Delete Sheet: delete an existing sheet in the database
- Access Sheet: choose an existing sheet to access
- Log out: log out and return to the Sign up/ Log in panel

1). Add Sheet

In this example, we can add a new sheet in the database and the file will be stored in the directory /lib/data after exit the system . In this case, user can directly access the file without adding again, functioned as the real "database" .

- to add a new sheet, the user need to input the abolute path of the csv file as the file path, and the sheet name as the file

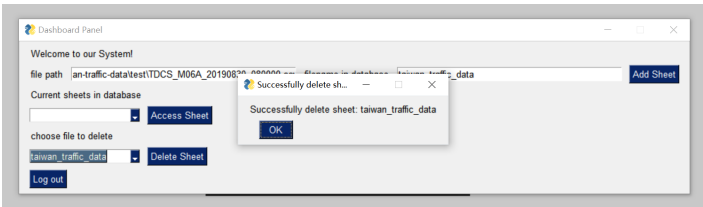
- click the button Add Sheet, we will add the sheet into database



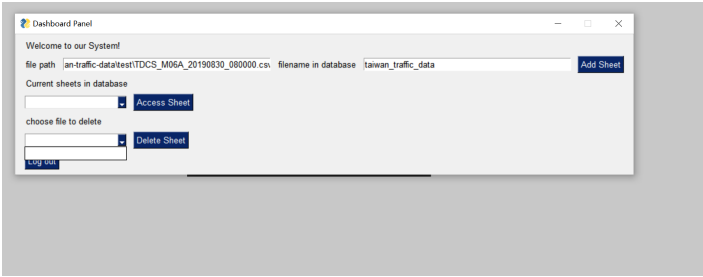
2). Delete Sheet

In this example, we can delete an existing file in the database. After deletion, the sheet will be removed from the "database" can not be accessed.

- To delete a sheet, the user needs to pull down the combo and select one sheet to delete



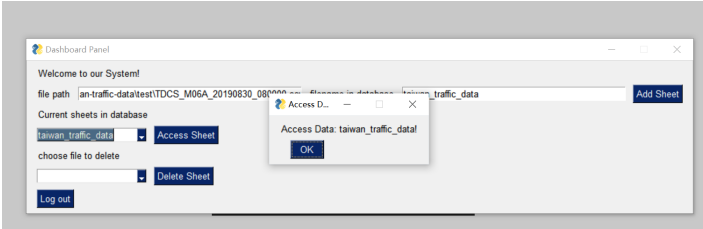
Now we see nothing in the combo after deletion



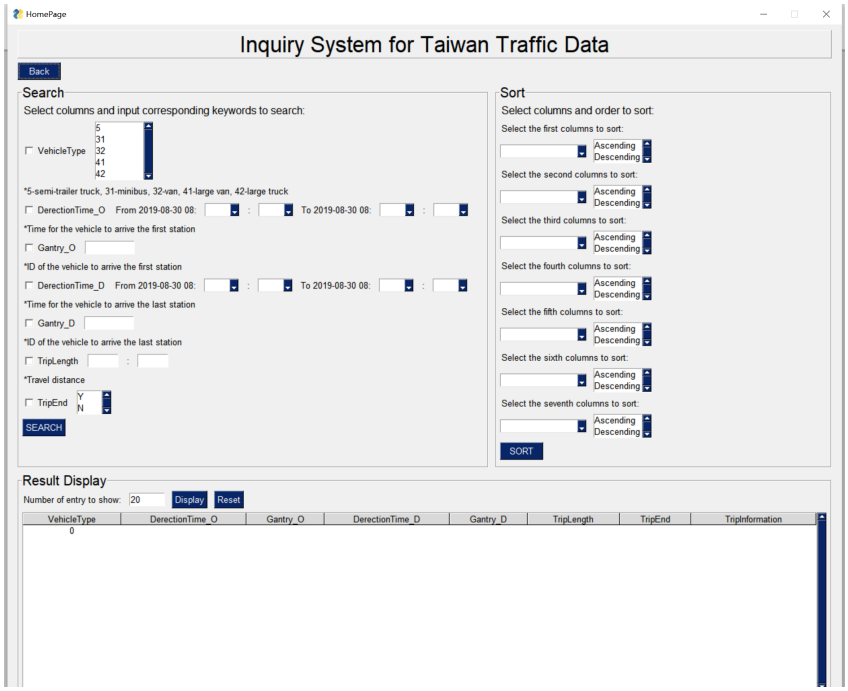
3). Access Sheet

In this case, we choose the taiwan traffic data sheet to access, which is the file added before.And then it leads us to the Analysis panel.

- To access a sheet, the user just pull down the combo and select the sheet he wants to access



- Click ok, we can enter the dashboard



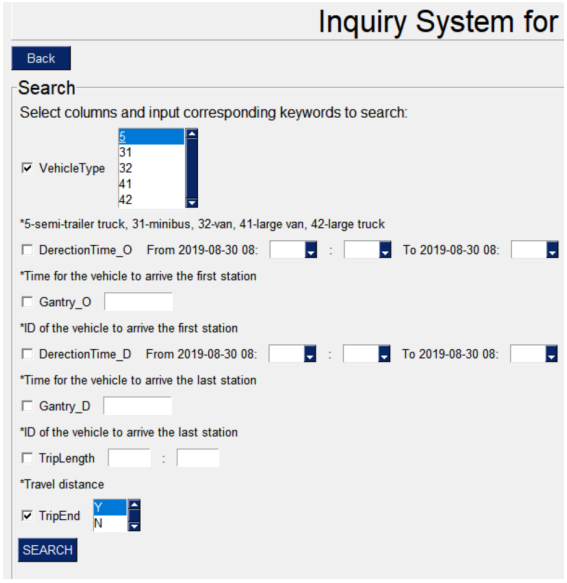
2.3 Analysis -Search & Sort

In the analysis panel, we provide two methods for our users.

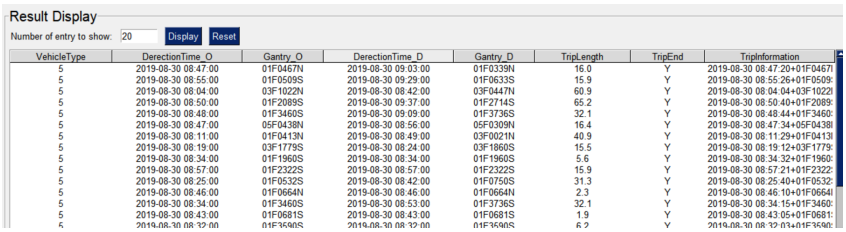
- Search: The user can search data he wants in the sheet with specific conditions. To be specific, the user should first choose a column (click the checkbox) and then set the filter condition through checkbox or input box provided in our system.
- Sort: The user can both sort the whole sheet or the output searched before with priority order. The system supports the users to sort the sheet with multiple columns and different ranking type(ascending/Descending)

Example

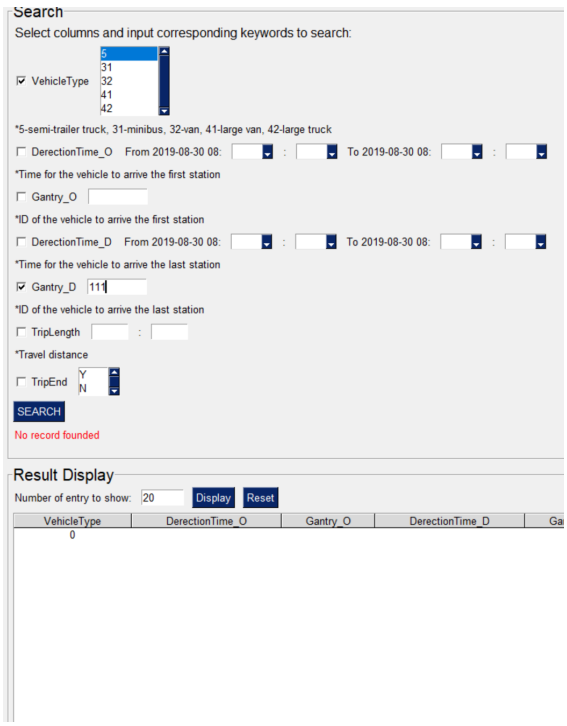
- Search. In this example, we want to search rows satisfying VehicleType == 5 and TripEnd == Y. We choose the VehicleType by clicking the combo and the TripEnd by select from the listbox.



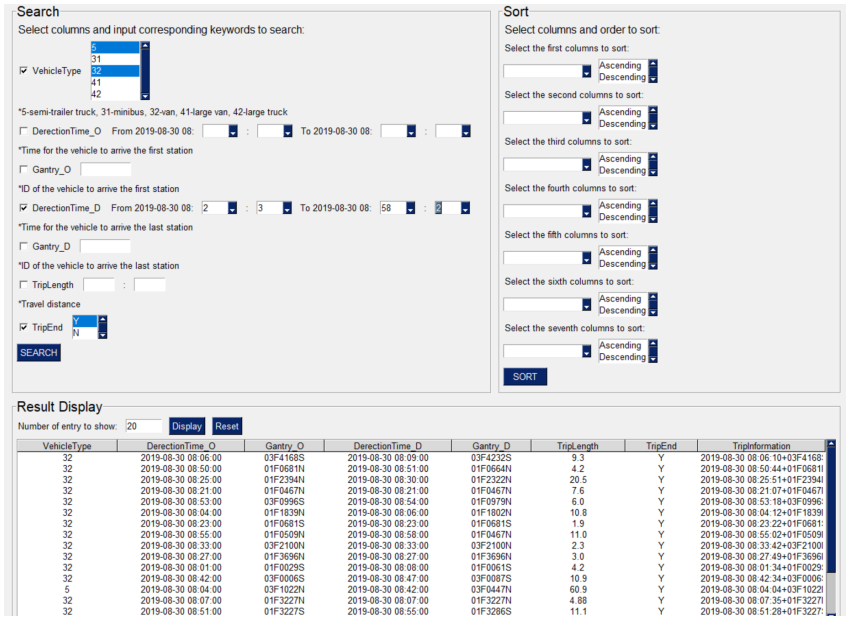
- Click the SEARCH button



If no record is founded by such filter conditions, a warning message will show up and table will be cleared:

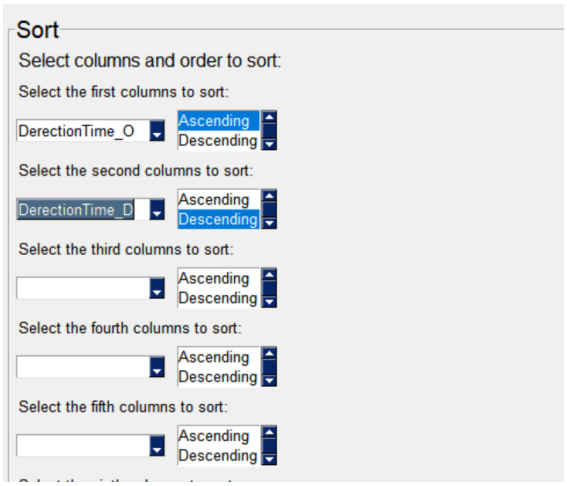


- Search again with another filter condition:

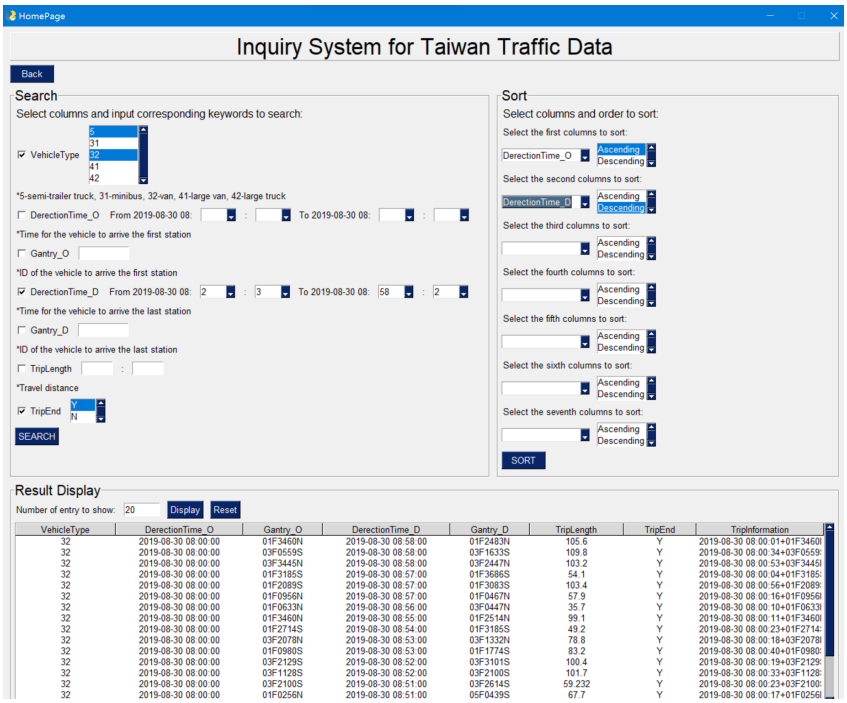


Now we want to continuously sort the above results by Derection_Time_O with Ascending order and Derection_Time_D with dscending order. In this case, we choose the column we want to sort and select the sorting rank type respectively.

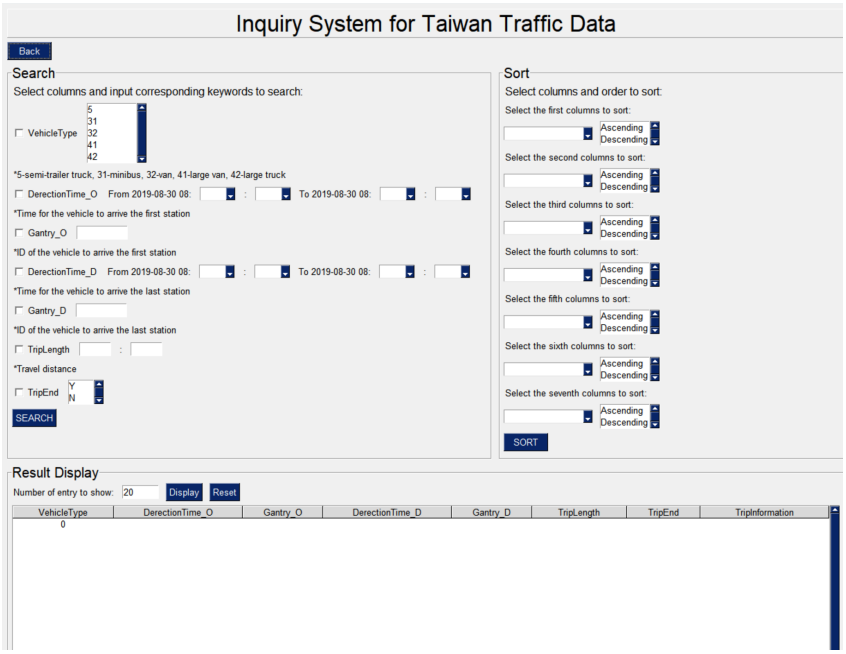
- choose sort condition



- click the SORT Button



- Reset: Click the reset button to reset the pages

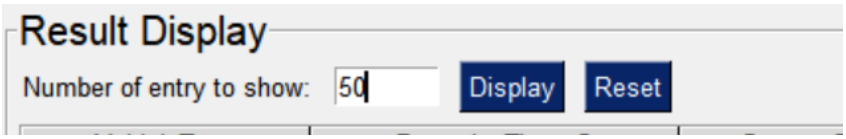


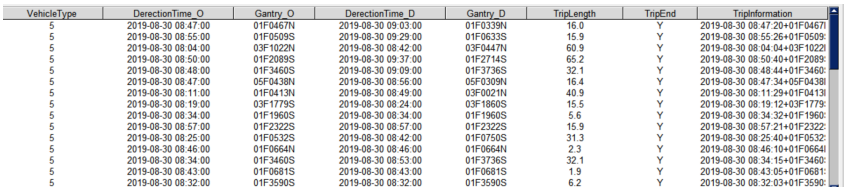
- click Back Button to return to the dashboard page
- click log out to return to the Login page
- click exit to end the program

2.4 Display

For the display part, the user can choose the number of rows to show in the panel.

- Modify the 'Number of entry to show' to set the number of rows we want to observe in the page
- click Display Button





VehicleType	DerectionTime_O	Gantry_O	DerectionTime_D	Gantry_D	TripLength	TripEnd	TripInformation
5	2019-08-30 08:47:00	01F0467N	2019-08-30 09:03:00	01F0339N	16.0	Y	2019-08-30 08:47:20+01F0467
5	2019-08-30 08:55:00	01F0509S	2019-08-30 09:29:00	01F0633S	15.9	Y	2019-08-30 08:55:26+01F0509
5	2019-08-30 08:04:00	03F1022N	2019-08-30 08:42:00	03F0447N	60.9	Y	2019-08-30 08:04:04+03F1022
5	2019-08-30 08:50:00	01F2089S	2019-08-30 09:37:00	01F2714S	65.2	Y	2019-08-30 08:50:40+01F2089
5	2019-08-30 08:48:00	01F3460S	2019-08-30 09:09:00	01F3736S	32.1	Y	2019-08-30 08:48:44+01F3460
5	2019-08-30 08:47:00	05F0438N	2019-08-30 08:56:00	05F0309N	16.4	Y	2019-08-30 08:47:34+05F0438
5	2019-08-30 08:11:00	01F0413N	2019-08-30 08:49:00	03F0021N	40.9	Y	2019-08-30 08:11:29+01F0413
5	2019-08-30 08:19:00	03F1779S	2019-08-30 08:24:00	03F1860S	15.5	Y	2019-08-30 08:19:12+03F1779
5	2019-08-30 08:34:00	01F1960S	2019-08-30 08:34:00	01F1960S	5.6	Y	2019-08-30 08:34:32+01F1960
5	2019-08-30 08:57:00	01F2322S	2019-08-30 08:57:00	01F2322S	15.9	Y	2019-08-30 08:57:21+01F2322
5	2019-08-30 08:25:00	01F0532S	2019-08-30 08:42:00	01F0750S	31.3	Y	2019-08-30 08:25:40+01F0532
5	2019-08-30 08:46:00	01F0664N	2019-08-30 08:46:00	01F0664N	2.3	Y	2019-08-30 08:46:10+01F0664
5	2019-08-30 08:34:00	01F3460S	2019-08-30 08:53:00	01F3736S	32.1	Y	2019-08-30 08:34:15+01F3460
5	2019-08-30 08:43:00	01F0681S	2019-08-30 08:43:00	01F0681S	1.9	Y	2019-08-30 08:43:05+01F0681
5	2019-08-30 08:32:00	01F3590S	2019-08-30 08:32:00	01F3590S	6.2	Y	2019-08-30 08:32:03+01F3590

Appendix

Github Link

https://github.com/Bai0-0/Design-the-inquiry-system-for-Taiwan-traffic-data.git

Source Code

In [ ]:
```
import os
from os.path import dirname, abspath, join
import pandas as pd

# from database import DataBase
# from userbase import UserBase
# from user_interface import UI

class System:

    def __init__(self) -> None:

        root_path = dirname(abspath(__file__))
        lib_path = join(root_path, 'lib')
        data_path = join(root_path, 'lib', 'data')
        self.test_path = join(root_path, 'test')
```

```
        self.generate_folder(lib_path)
        self.generate_folder(data_path)

        self.userbase = UserBase(lib_path)
        self.database = DataBase(data_path)

        self.database.add(join(self.test_path, 'TDCS_M06A_20190830_080000.csv'), 'taiwan_traffic_data')
        self.user_interface = UI(self)
        self.user_interface.run()

        self.userbase.save_to_local()
        self.database.save_to_local()



    @staticmethod
    def generate_folder(path: str) -> None:
        if not os.path.exists(path):
            os.makedirs(path)
        return None
```

In [ ]:

```
import shutil

class Data:

    def __init__(self, df: pd.DataFrame = pd.DataFrame()) -> None:
        self.__data = df.copy()
        self.__data.reset_index(drop = True, inplace=True)


        if not self.__data.empty:
            if type(self.__data.DerectionTime_O[0]) == str:
                self.__data.DerectionTime_O = pd.to_datetime(self.__data.DerectionTime_O)
                self.__data.DerectionTime_D = pd.to_datetime(self.__data.DerectionTime_D)

    def sort(self, col_name_list: list, ascending_list: list):
        """Sort data

        Args:
            col_name_list (list): column names to sort by
            ascending_list (list): True for ascending and False for descending

        Returns:
            Data
        """
        assert len(col_name_list) == len(ascending_list), 'Length not match for input arguments'
        res = self.__data.sort_values(col_name_list, ascending=ascending_list)
        return Data(res)

    def search(self, filter_dict: dict):
        """Search data

        Args:
            filter_dict (dict): filter dict

        Returns:
            Data
        """

        # f1: list[int]
        f1 = filter_dict.get('VehicleType', None)
        # f2: tuple(datetime1, datetime2)
        f2 = filter_dict.get('DerectionTime_O', None)
        # f3: str
        f3 = filter_dict.get('Gantry_O', None)
        # f4: tuple(datetime1, datetime2)
        f4 = filter_dict.get('DerectionTime_D', None)
        # f5: str
        f5 = filter_dict.get('Gantry_D', None)
        # f6: tuple(int1, int2)
        f6 = filter_dict.get('TripLength', None)
        # f7: list[str]
        f7 = filter_dict.get('TripEnd', None)

        # if (f3 is not None) and (f3 not in set(self.__data.Gantry_O)):
        #     return (0, 'Gantry_O', f3)

        # if (f5 is not None) and (f5 not in set(self.__data.Gantry_D)):
        #     return (0, 'Gantry_D', f5)

        res = self.__data.copy()
        if f1 is not None:
            res = res[res.VehicleType.isin(f1)]

        if f2 is not None:
            res = res[(f2[0] <= res.DerectionTime_O) & (res.DerectionTime_O <= f2[1])]

        if f3 is not None:
            res = res[res.Gantry_O == f3]

        if f4 is not None:
            res = res[(f4[0] <= res.DerectionTime_D) & (res.DerectionTime_D <= f4[1])]

        if f5 is not None:
```

```
            res = res[res.Gantry_D == f5]

        if f6 is not None:
            res = res[(f6[0] <= res.TripLength) & (res.TripLength <= f6[1])]

        if f7 is not None:
            res = res[res.TripEnd.isin(f7)]

        return Data(res)

    def download(self, path):
        self.__data.to_csv(path, index=False)

    def get(self):
        return self.__data


class DataBase:
    def __init__(self, data_path) -> None:
        self.__data_list = {}
        self.__data_path = data_path
        file_list = os.listdir(self.__data_path)
        file_list = [f for f in file_list if f != '.DS_Store']
        if len(file_list) > 0:
            for f in file_list:
                self.add(os.path.join(self.__data_path, f), f.rstrip('.csv'))

    def add(self, data_path: str, sheet_name: str):
        """Add sheet to the database

        Args:
            data_path (str): path of data sheet
            sheet_name (str): sheet name
        """
        df = pd.read_csv(data_path)
        self.__data_list[sheet_name] = Data(df)

    def access(self, sheet_name: str) -> Data:
        """Access sheet in the database

        Args:
            sheet_name (str): sheet name

        Returns:
            Data: data sheet
        """
        if sheet_name in self.__data_list:
            return self.__data_list[sheet_name]
        else:
            print('Data sheet does not exist in the database.')

    def delete(self, sheet_name: str):
        """Delete sheet from the database

        Args:
            sheet_name (str): sheet name
        """
        if sheet_name in self.__data_list:
            self.__data_list.pop(sheet_name)
            return True
        else:
            return False
            print()

    def save_to_local(self):
        shutil.rmtree(self.__data_path)
        self.generate_folder(self.__data_path)
        for k in self.__data_list:
            path = os.path.join(self.__data_path, k + '.csv')
            self.__data_list[k].download(path)

    def show_content(self):
        return self.__data_list.keys()

    @staticmethod
    def generate_folder(path: str) -> None:
        if not os.path.exists(path):
            os.makedirs(path)
        return None
```

In [ ]:

```
class UserBase:

    def __init__(self, lib_path) -> None:
        self.user_file_path = os.path.join(lib_path, 'account_info.csv')
        if os.path.exists(self.user_file_path):
            self.__account_info = pd.read_csv(self.user_file_path, index_col=0)
            self.__account_info = self.__account_info.astype(str)
            self.__account_info.reset_index(drop=True, inplace=True)
        else:
            self.__account_info = pd.DataFrame(columns=['uid', 'password'])

    def sign_up(self, uid: str, pw: str) -> tuple:
        """Sign up
```

```
        Args:
            uid (str): user name
            pw (str): password

        Returns:
            tuple: (True) for success
        """
        if uid not in set(self.__account_info.uid):
            tmp = pd.DataFrame({'uid': [uid], 'password': [pw]})
            self.__account_info = pd.concat([self.__account_info, tmp])
            self.__account_info.reset_index(drop=True, inplace=True)
        else:
            self.__account_info.loc[self.__account_info.uid == uid, 'password'] = pw

        return (True)

    def sign_in(self, uid: str, pw: str) -> tuple:
        """Sign in

        Args:
            uid (str): user name
            pw (str): password

        Returns:
            tuple: (True, '') for success, (False, Error) for failure
        """
        if uid in set(self.__account_info.uid):
            pw_list = list(self.__account_info.loc[self.__account_info.uid == uid, 'password'])
            if pw in pw_list:
                return (True, '')
            else:
                print('Incorrect Password.')
                return (False, 'Incorrect Password.')
        else:
            print('Invalid Username.')
            return (False, 'Invalid Username.')

    def save_to_local(self):
        if os.path.exists(self.user_file_path):
            os.remove(self.user_file_path)
        self.__account_info = self.__account_info.astype(str)
        self.__account_info.to_csv(self.user_file_path)
```

In [ ]:

```
import PySimpleGUI as sg
import pandas as pd
from pandas import Timestamp
# from database import Data, DataBase
import copy

class UI:

    @staticmethod
    def int_tuple_to_datetime(M, S):

        strm = str(M)
        strs = str(S)
        if len(strm) < 2:
            strm = '0' + strm
        if len(strs) < 2:
            strs = '0' + strs
        date = '2019-08-30 08:' + strm + ':' + strs

        return pd.to_datetime(date)

    def __init__(self, system) -> None: # system:System

        self.system = system

        '''-------------Layout design-------------------'''

        sg.theme('default')

        header = ("VehicleType",'DerectionTime_O','Gantry_O','DerectionTime_D',
                  'Gantry_D','TripLength',"TripEnd",'TripInformation')
        header_sort = ("VehicleType",'DerectionTime_O','Gantry_O','DerectionTime_D',
                       'Gantry_D','TripLength',"TripEnd",'TripInformation', '/')
        vehicle_list = ('5','31','32','41','42')

        layout_userPage =[[sg.Text("Please enter your ID and Password:")],
                          [sg.Text("User ID:")],
                          [sg.Input(key = "userID")],
                          [sg.Text("Password:")],
                          [sg.Input(key = "pwd", password_char='*')],
                          [sg.Button("Sign Up"), sg.Button('Log In'), sg.Button('Exit')],
                          [sg.Text('Sucessfully sign up, please log in', k = '-end_signup-',
                                   text_color = 'red', visible=False),
                           sg.Text(' ',k = '-warn_user-', text_color='red')]]

        frame_search =sg.Frame(title ='Search', font = ("Helvetica", 15), size = [700,570],
                    layout =[[sg.Text('Select columns and input corresponding keywords to search:',
                                      font = ("Helvetica", 12))],

                    #Column 1
                    [sg.Checkbox('VehicleType', k='-CB_Vehicle-'),sg.Listbox(size=(10,5),values = vehicle_list,
```

```
                select_mode = 'multiple',key = "-LB_Vehicle-")],
                [sg.Text('*5-semi-trailer truck, 31-minibus, 32-van, 41-large van, 42-large truck')],

                #Column 2
                [sg.Checkbox('DerectionTime_O', k='-CB_TimeO-'), sg.Text("From 2019-08-30 08:"),
                sg.Combo(values=[i for i in range(60)],size =(5,5), key = '-From_OMin-'),sg.Text(':'),
                sg.Combo(values=[i for i in range(60)],size =(5,5), key = '-From_OSec-'),
                sg.Text('To 2019-08-30 08:'), sg.Combo(values=[i for i in range(60)],size =(5,5),
                                                       key = '-To_OMin-'),
                sg.Text(':'),sg.Combo(values=[i for i in range(60)],size =(5,5), key = '-To_OSec-')],
                [sg.Text('*Time for the vehicle to arrive the first station')],

                #Col 3
                [sg.Checkbox('Gantry_O', k='-CB_GO-'),sg.Input(key = '-Input_GO-',size = [10,1])],
                [sg.Text('*ID of the vehicle to arrive the first station')],

                #col4
                [sg.Checkbox('DerectionTime_D', k='-CB_TimeD-'), sg.Text("From 2019-08-30 08:"),
                sg.Combo(values=[i for i in range(60)],size =(5,5), key = '-From_DMin-'),sg.Text(':'),
                sg.Combo(values=[i for i in range(60)],size =(5,5), key = '-From_DSec-'),
                sg.Text('To 2019-08-30 08:'), sg.Combo(values=[i for i in range(60)],size =(5,5),
                                                       key = '-To_DMin-'),
                sg.Text(':'),sg.Combo(values=[i for i in range(60)],size =(5,5), key = '-To_DSec-')],
                [sg.Text('*Time for the vehicle to arrive the last station')],

                #col5
                [sg.Checkbox('Gantry_D', k='-CB_GD-'),sg.Input(key = '-Input_GD-',size = [10,1])],
                [sg.Text('*ID of the vehicle to arrive the last station')],

                #col6
                [sg.Checkbox('TripLength', k='-CB_TripLen-'),sg.Input(key = '-Input_TripLen1-',size = [6,1]),
                           sg.Text(':'),sg.Input(key = '-Input_TripLen2-',size = [6,1])],
                [sg.Text('*Travel distance')],

                #col7
                [sg.Checkbox('TripEnd', k='-CB_TripE-'),
                sg.Listbox(values = ('Y','N'),size = [5,2],select_mode = 'multiple',key = "-LB_TripE-")],
                [sg.Button('SEARCH',key = "-Search-",size = [7,1])],
                [sg.Text('No record founded',text_color = 'red', k = '-warning-',visible = False)]
                ])

        frame_sort =sg.Frame(title ='Sort',font = ("Helvetica", 15), size = [500,570],
                             element_justification = 'left', layout =[
            [sg.Text('Select columns and order to sort:',font = ("Helvetica", 12))],
            [sg.Text('Select the first columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_1-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_1-')],
            [sg.Text('Select the second columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_2-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_2-')],
            [sg.Text('Select the third columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_3-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_3-')],
            [sg.Text('Select the fourth columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_4-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_4-')],
            [sg.Text('Select the fifth columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_5-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_5-')],
            [sg.Text('Select the sixth columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_6-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_6-')],
            [sg.Text('Select the seventh columns to sort:')],
            [sg.Combo(values=header_sort,key='-Sort_Col_7-',enable_events = True),
            sg.Listbox(size=(10,2), values=['Ascending', 'Descending'],key='-Sorting_Order_7-')],
            [sg.Button('SORT', key = '-Sort-',size = [7,1])],
            [sg.Text(k='search_output_list')]])

        layout_res = [[sg.Text('Number of entry to show:'), sg.Input(default_text = '20',
                                                                     k = '-Input_head-',size=[7,1]),
                       sg.Button('Display', k = '-display-'), sg.Button('Reset', k = '-clear-')],
                      [sg.Table([[0]], headings = header, num_rows = 20,
                                size = [1150, 30], expand_x = True, expand_y = True,
                      justification='center', k = '-res-')]]

        frame_res = sg.Frame(title='Result Display',  font = ("Helvetica", 15),
                             size = [1210,330],layout = layout_res)

        self.layout_homePage = [[sg.Text('Inquiry System for Taiwan Traffic Data',
                                         font = ("Helvetica", 25), expand_x = True,
                           justification = 'center', relief=sg.RELIEF_RIDGE)],
                           [sg.Button('Back', size = [7,1])],
                           [frame_search, frame_sort],
                           [frame_res]]
        self.layout_dashboardPage = [[sg.Text('Welcome to our System!')],

                       [sg.Text('file path'),sg.InputText(key = '-Input_Path-'),
                        sg.Text('filename in database'),
                       sg.InputText(key = '-Input_Name-'),sg.Button("Add Sheet",
                                                                    key = '-Action_add-')],
                       [sg.Text('Current sheets in database')],
                       [sg.Combo(values = list(self.system.database.show_content()),
                                 key='-Sheet_To_Access-', enable_events=True),
                       sg.Button('Access Sheet', key = '-Action_access-')],
                       [sg.Text('choose file to delete')],
```

```
                    [sg.Combo(values = list(self.system.database.show_content()),
                              key='-Sheet_To_Delete-', enable_events=True),
                    sg.Button('Delete Sheet', key = '-Action_delete-')],
                    [sg.Button('Log out', key = '-Log_out-')]]

        self.win_userPage = sg.Window('Login Page', layout = layout_userPage)
        self.win_homePage_active = False
        self.win_dashboardPage_active = False

    def search(self, vals2):

        '''
        store self.res_table  continously, reset to original when click '-clear-' button

        '''

        self.win_homePage['-warning-'].update(visible = False)

        filter_dict = { }

        if vals2['-CB_Vehicle-'] == True:

            filter_dict['VehicleType'] = [int(x) for x in vals2['-LB_Vehicle-']]

        if vals2['-CB_TimeO-'] == True:
            filter_dict['DerectionTime_O'] = (self.int_tuple_to_datetime(
                vals2['-From_OMin-'], vals2['-From_OSec-']),
                                              self.int_tuple_to_datetime(
                                                  vals2['-To_OMin-'], vals2['-To_OSec-']))
        if vals2['-CB_GO-'] == True:
            filter_dict['Gantry_O'] = str(vals2['-Input_GO-'])

        if vals2['-CB_TimeD-'] == True:
            filter_dict['DerectionTime_D'] = (self.int_tuple_to_datetime(
                vals2['-From_DMin-'], vals2['-From_DSec-']),
                                              self.int_tuple_to_datetime(
                                                  vals2['-To_DMin-'], vals2['-To_DSec-']))

        if vals2['-CB_GD-'] == True:
            filter_dict['Gantry_D'] = str(vals2['-Input_GD-'])

        if vals2['-CB_TripLen-'] == True:
            filter_dict['TripLength'] = (float(vals2['-Input_TripLen1-']),float(vals2['-Input_TripLen2-']))

        if vals2['-CB_TripE-'] == True:
            filter_dict['TripEnd'] = vals2['-LB_TripE-']

        self.res_table = Data(self.origin_sheet.get().copy())  # reset self.res_table

        self.res_table = self.res_table.search(filter_dict)

        self.display(int(vals2['-Input_head-']))

    def sort(self,vals2): #working_sheet: Data
        self.win_homePage['-warning-'].update(visible = False)

        ascending_order = []
        col_name = []

        for i in range(1,8):
            col = f'-Sort_Col_{i}-'
            sort_order = f'-Sorting_Order_{i}-'
            if vals2[col] and vals2[col] != '/':
                col_name.append(vals2[col])
                if len(vals2[sort_order])>0:
                    ascending_order.append(False if vals2[sort_order][0] == 'Descending' else True)
                else:
                    ascending_order.append(True)

        self.res_table = self.res_table.sort(col_name, ascending_order)
        self.display(int(vals2['-Input_head-']))

    def display(self, num_row: int):

        self.res_table_data = self.res_table.get().copy()

        if len(self.res_table_data) == 0: # empty search result

            self.win_homePage['-warning-'].update(visible = True) #show warning message
            self.win_homePage['-res-'].update([[0]])

        else:
            self.res_table_data['DerectionTime_D'] = self.res_table_data['DerectionTime_D'].astype(str)
            self.res_table_data['DerectionTime_O'] = self.res_table_data['DerectionTime_O'].astype(str)

            temp = self.res_table_data.head(num_row)
            temp = temp.values.tolist()

            self.win_homePage['-res-'].update(temp)
            self.win_homePage['-warning-'].update(visible = False)


    def run(self):
```

```
        while True:
            ev1, vals1 = self.win_userPage.read()

            if ev1 == sg.WIN_CLOSED or ev1 == 'Exit' or ev1 == None:
                break

            '''-------------step 1: login page manipulation-----------'''
            if not self.win_dashboardPage_active and ev1 == 'Sign Up':
                print(vals1['userID'], vals1['pwd'])
                self.win_userPage['-warn_user-'].update('')
                self.system.userbase.sign_up(vals1['userID'],vals1['pwd'])
                self.win_userPage['-end_signup-'].update(visible = True)


            if not self.win_dashboardPage_active and ev1 == 'Log In':
                self.win_userPage['-end_signup-'].update(visible = False)

                temp = self.system.userbase.sign_in(vals1['userID'],vals1['pwd'])

                if temp[0] is False:  # (False, message)

                    self.win_userPage['-warn_user-'].update(temp[1])

                else: #sucessfully log in

                    #create home page
                    self.win_dashboardPage_active = True
                    temp_layout = copy.deepcopy(self.layout_dashboardPage)

                    self.win_dashboardPage = sg.Window('Dashboard Panel', temp_layout)
                    #hide user page
                    self.win_userPage.hide()
                    self.win_userPage['-warn_user-'].update(' ')


                    while self.win_dashboardPage_active:
                        ev3, vals3 = self.win_dashboardPage.read()

                        if ev3 == '-Action_access-':
                            if not vals3['-Sheet_To_Access-']:
                                sg.popup('Please select a sheet!')
                                continue
                            else:
                                data_sheet = vals3['-Sheet_To_Access-']

                                self.origin_sheet = self.system.database.access(data_sheet)
                                self.res_table = Data(self.origin_sheet.get().copy())
                                sg.popup(f'Access Data: {data_sheet}!')

                                self.win_homePage_active = True
                                self.win_dashboardPage.hide()
                                temp_layout = copy.deepcopy(self.layout_homePage)

                                self.win_homePage = sg.Window('HomePage', temp_layout, finalize=True)

                                while self.win_homePage_active == True:

                                    ev2, vals2 = self.win_homePage.read()


                                    if ev2 == '-Search-': #press search button
                                        self.search(vals2)

                                    if ev2 == '-Sort-':

                                        self.sort(vals2)

                                    if ev2 == '-display-':

                                        self.display(int(vals2['-Input_head-']))

                                    if ev2 == '-clear-': #rebuild homepage
                                        self.win_homePage.close()
                                        self.res_table = Data(self.origin_sheet.get().copy())
                                        temp_layout = copy.deepcopy(self.layout_homePage)
                                        self.win_homePage = sg.Window('HomePage', temp_layout, finalize=True)

                                    if ev2 == sg.WIN_CLOSED or ev2 == None or ev2 == 'Back':
                                        #Close homepage and back to login page
                                        self.win_homePage_active  = False
                                        self.win_homePage.close()
                                        self.win_dashboardPage.un_hide()  #back to login page
                                        break

                        elif ev3 == '-Action_add-':
                            file_path = vals3['-Input_Path-']
                            file_name = vals3['-Input_Name-']
                            self.system.database.add(file_path,file_name)
                            sg.popup(f'Successfully add sheet: {file_name}')


                            self.win_dashboardPage['-Sheet_To_Delete-'].update(values =list(
                                self.system.database.show_content())  )
                            self.win_dashboardPage['-Sheet_To_Access-'].update(values =list(
                                self.system.database.show_content())  )
```

```
                    elif ev3 == '-Action_delete-':
                        file_name = vals3['-Sheet_To_Delete-']
                        deleted = self.system.database.delete(file_name)
                        if deleted:
                            sg.popup(f'Successfully delete sheet: {file_name}')
                        else:
                            sg.popup('Data sheet does not exist in the database.')

                        self.win_dashboardPage['-Sheet_To_Delete-'].update(values =list(
                            self.system.database.show_content())  )
                        self.win_dashboardPage['-Sheet_To_Access-'].update(values =list(
                            self.system.database.show_content())  )

                    elif ev3 == '-Log_out-' or ev3 == sg.WIN_CLOSED or ev3== None:
                        self.win_dashboardPage_active = False
                        self.win_dashboardPage.close()
                        self.win_userPage.un_hide()  #back to login page

        self.win_userPage.close()
        print("End")
```

In [ ]:

```
''' run the main program here'''

if __name__ == '__main__':
    system = System()
```