Project Instructions

Your project is to create a module named `sakaydb` for managing ride-hailing data. This is to be done by LT. Your code should be commited on a private repo in GitHub with repo name `sakaydb`, then submit this notebook via the `Assignments` tab of `nbgrader`. The project is due on September 7, 2022, 11:59PM. Only one member of the LT should submit---there will be a penalty for submissions from multiple members of an LT. The module should be at the top-level directory of this repo. Grant read access to this repo to Damian Dailisan and Michael Dorosan (GH account `temetski` and `mikedataCrunch`). **Do not write your code on this notebook nor submit it along with this notebook**. Just specify your repo url in the cell below.

Only the following packages may be used for the implementation:

- Python standard libraries
- Numpy (but not scipy)
- Pandas
- Matplotlib

The `SakayDB` class

The module should contain one class named `SakayDB`. It should have the following specifications:

Initialization

The class initializer should accept a string `data_dir` which is the directory path to where the data files are located. This path should be stored in the `data_dir` attribute of the object.

Data persistence

Data are stored in the CSV files `trips.csv`, `drivers.csv`, and `locations.csv` in `data_dir`. The CSV files begin with a column header and the columns of each CSV file are:

`trips.csv`:

- `trip_id`: an integer assigned to trip
- `driver_id`: an integer assigned to the driver
- `pickup_datetime`: datetime of dropoff as string with format "hh:mm:ss,DD-MM-YYYY"
- `dropoff_datetime`: datetime of dropoff as string with format "hh:mm:ss,DD-MM-YYYY"
- `passenger_count`: number of passengers as integer
- `pickup_loc_id`: an integer assinged to to the pickup location
- `dropoff_loc_id`: an integer assinged to to the dropoff location
- `trip_distance`: distance in meters as float
- `fare_amount`: total fare amount as float

`drivers.csv`:

- `driver_id`: an integer assigned to the driver
- `last_name`: last name of driver
- `given_name`: given name of driver

`locations.csv`:

- `location_id`: an integer assigned to the location
- `loc_name`: zone location name

Exception

The class has the associated exception `SakayDBError` which is a `ValueError`.

Features

Adding a single trip to the database

Create a method `add_trip` that accepts the following parameters:

- `driver`: trip driver as a string in `Last name, Given name` format
- `pickup_datetime`: datetime of pickup as string with format "hh:mm:ss,DD-MM-YYYY"
- `dropoff_datetime`: datetime of dropoff as string with format "hh:mm:ss,DD-MM-YYYY"
- `passenger_count`: number of passengers as integer
- `pickup_loc_name`: zone as a string, (e.g., Pine View, Legazpi Village)
- `dropoff_loc_name`: zone as a string, (e.g., Pine View, Legazpi Village)
- `trip_distance`: distance in meters (float)
- `fare_amount`: amount paid by passenger (float)

The method should append the trip data to the end of `trips.csv`, if it exists, or creates it, otherwise.

The `trip_id` is *last* `trip_id` *in the file* + 1, or `1` if there's no trip in the file yet.

The `driver_id` is the corresponding `driver_id` in `drivers.csv` based on the case-insensitive matches of `given_name` and `last_name`. It should append the driver in `drivers.csv` if the driver is not yet there. The `driver_id` of a new driver is _last `driver_id` in the file_ + 1, or `1` if there's no driver in the file yet. The method should return the `trip_id` or raise a `SakayDBError` exception if the trip is already in `trips.csv`. A trip is said to be in `trips.csv` if there is a trip that matches the `driver` (case-insensitive), `pickup_datetime`, `dropoff_datetime`, `passenger_count`, `pickup_loc_id`, `dropoff_loc_id`, `trip_distance` and `fare_amount`.

Adding trips in the database

Create a method `add_trips` that accepts a list of trips in the form of dictionaries with the following keys:

- `driver`: trip driver as a string in `Last name, Given name` format
- `pickup_datetime`: datetime of pickup as string with format "hh:mm:ss,DD-MM-YYYY"
- `dropoff_datetime`: datetime of dropoff as string with format "hh:mm:ss,DD-MM-YYYY"
- `passenger_count`: number of passengers as integer
- `pickup_loc_name`: zone as a string, (e.g., Pine View, Legazpi Village)
- `dropoff_loc_name`: zone as a string, (e.g., Pine View, Legazpi Village)
- `trip_distance`: distance in meters (float)
- `fare_amount`: amount paid by passenger (float)

The method should add each trip to the database. It returns a list of the `trip_ids`s of successfully added trips. If a trip is already in the database, skip it and print: `Warning: trip index {i} is already in the database. Skipping...`

If a trip has invalid or incomplete information, skip it and print `Warning: trip index {i} has invalid or incomplete information. Skipping...` instead. The *trip index* is the zero-based index of the trip in the passed list of trips.

Deleting a trip in the database

Create a method `delete_trip` that accepts the `trip_id` to delete then removes it from `trips.csv`. It will raise a `SakayDBError` if the `trip_id` is not found.

Searching for trips in the database

Create a method `search_trips` that accepts the following keyword arguments:

- `key`: string, can be any of the ff: `driver_id`, `pickup_datetime`, `dropoff_datetime`, `passenger_count`, `trip_distance`, `fare_amount`. If dict, search can be done with multiple keys
- `range` : for range search
  - Case 1: tuple like (*value*, `None`) sorts by `key` (chronological or ascending) returns all entries from *value*, begin inclusive
  - Case 2: tuple like (`None`,*value*) sorts by `key` (chronological or ascending) returns all entries up to *value*, end inclusive
  - Case 3: tuple like (*value1*, *value2*) sorts by `key` and returns values between *value1* and *value2*, end inclusive.
- `exact` : for single value search. Some *value* with data type and format conforming to that of `key`
- `dict_` : search using intersection of multiple `key` and `range` / `exact` pairs inputed as a dictionary.

This method should raise a `SakayDBError` when the following are not satisfied:

- Invalid input to `key` i.e. not in listed keys above
- Invalid input to `range` i.e. tuples with sizes greater than 2, either *values1* or *value2* not of and not in `key`.
- When either *value1* or *value2* is beyond the datetime range available in the database in any of Cases 1 to 3 for `range`
- Invalid input to `exact` i.e. data type or format not algined with values in `key`

The method should return a `pd.DataFrame` of all the entries aligned with search key and values.

Exporting data

Create a method `export_data` that returns all of the trips in the database as a pandas data frame with the following columns:

- `driver_lastname`: trip driver last name as string, capitalize first letter of each word in lastname
- `driver_givenname`: trip driver last name as string, capitalize first letter of each word in lastname
- `pickup_datetime`: datetime of pickup as string with format "hh:mm:ss,DD-MM-YYYY"
- `dropoff_datetime`: datetime of dropoff as string with format "hh:mm:ss,DD-MM-YYYY"
- `passenger_count`: number of passengers as integer
- `pickup_loc_name`: zone as a string, (e.g., Pine View, Legazpi Village)
- `dropoff_loc_name`: zone as a string, (e.g., Pine View, Legazpi Village)
- `trip_distance`: distance in meters (float)
- `fare_amount`: id of the trip's director

Sort the rows by the corresponding `trip_id` of each trip.

Generating statistics

Create a method `generate_statistics` that returns a dictionary depending on the `stat` parameter passed to it:

- `trips`: key is day name (e.g., Monday), value is the average number of trips with pick-ups for that day name in the entire dataset
- `passenger`: key is each unique `passenger_count`, value is another dictionary with day name (e.g., `Monday`) as key, and value is the average number of trips with pick-ups for that day name in the entire dataset
- `driver`: key is driver name following the format `Last name, Given name`, value is another dictionary with day name as key and average number of trips of that driver for that day name as value
- `all`: keys are `trips`, `passenger` and `driver`, values are the corresponding `stat` dictionaries returned by those keywords

The `stat` values are case-sensitive and the method should raise `SakayDBError` if the passed `stat` is unknown.

Plotting statistics

Create a method `plot_statistics` that returns a matplotlib `Axes` depending on the `stat` parameter passed to it:

- `trips`: bar plot of the average number of trips per day name
- `passenger`: scatter plot with x-axis as the day of the week, y-axis is the number of passengers, and the size of the dot in the plot representing the average number of trips.
- `driver`: Plot only the 5 drivers with the most trips. Scatter plot with x-axis as the day of the week, y-axis is the name of the driver (sorted by last name alphabetically, top to bottom), and the size of the dot in the plot representing the average number of trips of that driver for that day.

The `stat` values are case-sensitive and the method should raise `SakayDBError` if the passed `stat` is unknown.

Generate Origin-Destination Matrix

Create a method `generate_odmatrix` that takes in a `range` input parameter and returns a `pandas.DataFrame` with the `trips.csv` `pickup_loc_name` as the row names (dataframe index) and `dropoff_loc_name` as the columns. The values for each row-column combination is the number of trips that occured within the `range` specified.

- `range`: should funciton like that of `search_trips` but only uses the `pickup_datetime` as *key*. Hence, `range` should only take in a tuple of datetime strings.

Input errors to the `range` parameter should be handled like that of `search_trips`.

Grading guide [Still To Change]

- The project has a highest possible score of 150 points.
- Each cell with an assert statement is worth 10 pts. Successfully passing all of the tests in a cell will earn you the entire 10 pts. Failure to pass any of the test in the cell, including hidden tests, will earn no point. No partial points will be given thus make sure that you run and pass all the visible tests in the test suite before submitting.
- Successful git cloning is worth 15 pts. Successful importing of the module is worth 5 pts.
- If the module fails to clone or import, the professor will attempt to make it work but will merit additional deductions up to 10% of highest possible score.
- Methods should have a sensible docstring. The professor will deduct up to a total of 15 pts for missing, misleading or nonsensible docstrings. If you reasonably follow the numpy docstring format then you will likely not receive any deductions.
- The code should follow PEP8. The professor will run your python codes through pycodestyle and will deduct a point up to a total of 15 points for every instance of PEP8 violation (including warning).

In [1]:
```
# THIS IS THE ONLY CELL THAT YOU WILL MODIFY IN THIS NOTEBOOK.
# Store the SSH clone URL for your `sakaydb` repo as a string
git_repo_url = ''
```