Repository link - https://github.com/rrajkuma/SI206_wn23_take-flight

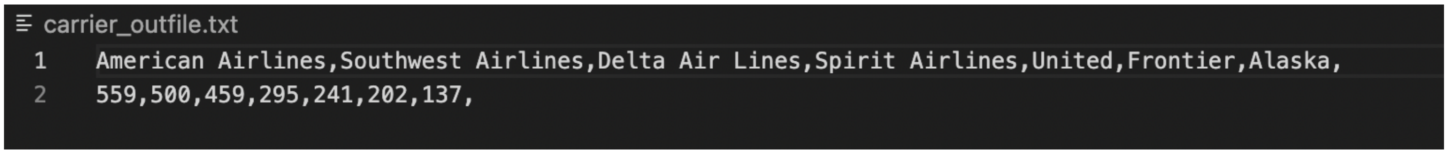
- The goals for your project including what APIs/websites you planned to work with and what data you planned to gather
  - We planned on grabbing data on the total number of airplane routes offered in a year within the continental US and the number of flights flown per route from aviationstack, and information about flight statuses for flights of interest from aerodatabox. Then, we planned to grab the data from the flight radar API to check which flights are the most frequently tracked.
  - We intended to see the frequency of unfavorable flight statuses for each route and carrier, and if these flights are regularly tracked. We would then explore any trends between flight delays and tracking numbers.

- The goals that were achieved including what APIs/websites you actually worked with and what data you did gather
  - We ultimately used aerodatabox for grabbing data on the total number of airplane routes offered in a year, and the number of flights per route. This will include international flights, since the API doesn't allow us to remove international flights from consideration, and manually removing them from the database would involve us searching each individual IATA code to determine if it belongs in the continental US. We also started data collection by looking at flights from 5 airports of interest: Detroit (DTW), Phoenix (PHX), Atlanta (ATL), Seattle (SEA), and Chicago (ORD), since aerodatabox returns data on the routes departing from an airport of interest, the carriers servicing those routes, and the average number of flights on that route. From there, we can load the data into the appropriate tables, and calculate what we need later.
  - For the second api we used flightradar to grab data on the most frequently tracked flights, collecting information on the flight name, the number of clicks and the from and to city as well. Then creating a visualization of the top departure cities that are tracked on flightradar

- The problems that you faced
  - Aviationstack only allows free access to real-time data, while the free plan on aerodatabox allows users to access real-time and historical data, albeit with a cap on the amount of requests the user can make. As a result, we limited our API usage to what aerodatabox can offer.
  - While aerodatabox has information on flight delays/statuses, it was difficult to aggregate enough data on unfavorable statuses in the timeframe given and the limitations on the API itself. One way to collect status data is to search for it via flight number and date, but this approach returns data for only one specific flight at a time. Aggregating enough data through this method would max out the number of requests we can make. The other method allows us to search for flight statuses by IATA code (essentially searching by airport rather than by flight), but due to the sheer scale of a commercial airport, only returns statuses within a maximum of 12 hours from the current time. This would not be indicative of year-round performance, and would therefore be harder to reconcile with the price data scraped from other websites.
  - We had to write separate SQL queries in order to pull the IATA code that corresponds to a certain id, rather than making it a subquery since the subqueries assigned the source_id and dest_id to DTW and ATL respectively, regardless of the values of the ids themselves. We ultimately made a query to select the source and destination ids, number of flights per route, and carrier first, and then made separate queries to replace the ids with their source and destination IATA codes.
  - Difficult to visualize large batches of data with legible labels.

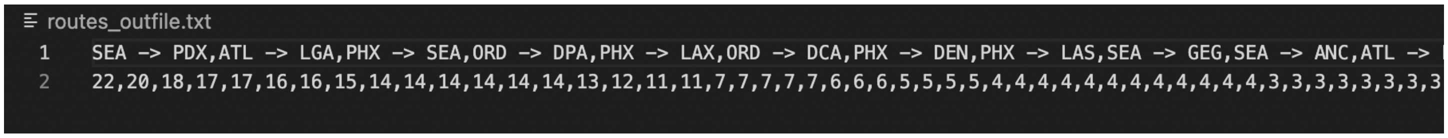
- The calculations from the data in the database (i.e. a screenshot)

Largest carriers in the the cities of interest



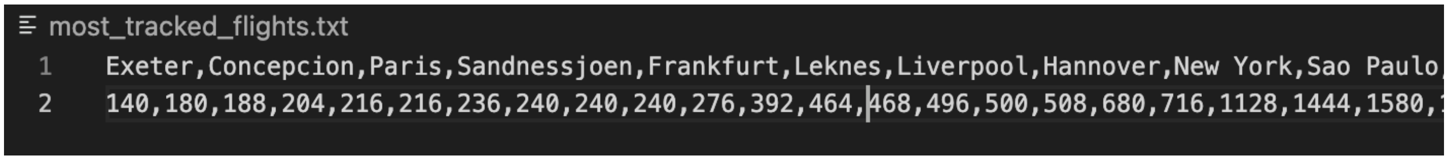
```
carrier_outfile.txt
American Airlines,Southwest Airlines,Delta Air Lines,Spirit Airlines,United,Frontier,Alaska,
559,500,459,295,241,202,137,
```

Frequency of routes



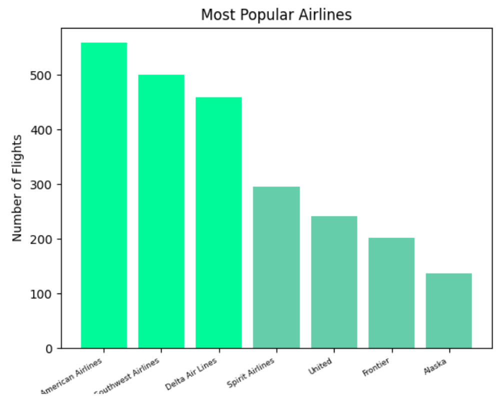
```
routes_outfile.txt
SEA -> PDX,ATL -> LGA,PHX -> SEA,ORD -> DPA,PHX -> LAX,ORD -> DCA,PHX -> DEN,PHX -> LAS,SEA -> GEG,SEA -> ANC,ATL ->
22,20,18,17,17,16,16,15,14,14,14,14,14,14,13,12,11,11,7,7,7,7,7,6,6,6,5,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3
```

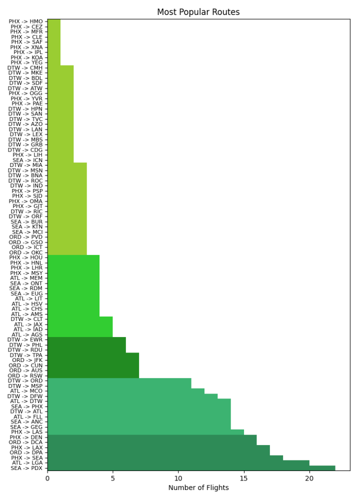
Frequency of tracking of departure cities

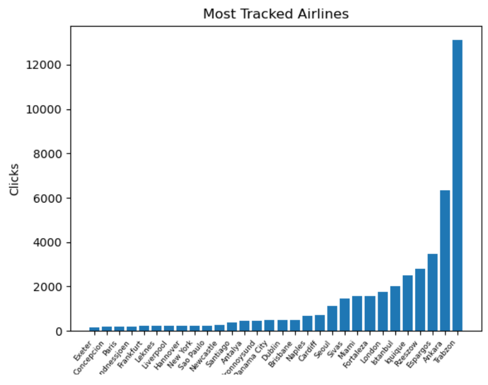


```
most_tracked_flights.txt
Exeter,Concepcion,Paris,Sandnessjoen,Frankfurt,Leknes,Liverpool,Hannover,New York,Sao Paulo,
140,180,188,204,216,216,236,240,240,240,276,392,464,468,496,500,508,680,716,1128,1444,1580,
```

- The visualization that you created (i.e. screenshot or image file)







- Instructions for running your code
  - First run the file flight_data_get.py to gather data from the aerodatabox API into the flight database.
  - After this run flight_data_visualize.py which will take in the data from the airports, carrier and routes table from the flight database and visualize the data as two png files, the first is the largest

    carriers between the cities of interest and the second is the most popular routes store the calculations in a txt file.
  - Then run the file most_tracked_flights.py to gather data from the flightradar API and store it in the flight database.
  - Lastly run the visualise_tracked_flights.py file which takes the data from the tracked_flight table in the flight database and visualizes the data as a png file, the top departing cities (from_cities) with the highest number of clicks and stores the calculation in a txt file.

- Documentation for each function that you wrote. This includes describing the input and output for each function

  - flight_data_get.py
    - get_API_info(url, params)
      - Grabs information using a url and key for aerodatabox.
      - Returns a dictionary with the data from aerodatabox.
    - open_database(db_name)
      - Takes in a string for a filename to save the database to, and creates the airports, routes, and carriers tables in the database. It also adds the first 5 airports to the table, which is found in the IATA_list in main.
      - Returns a connection object to the database, as well as a cursor object.
    - aero_data_into_dict(data_dict, start, end)
      - Takes in the flight_dict from get_API_info for a given airport and a start and endpoint for data collection from aero_dict_into_database to stay within the 25 row limit per run outlined in the rubric.
      - Returns a dict that stores IATA code, name, city, the average number of flights to that airport, and carriers.
    - helper_planes_into_database(plane_list, cur, conn)
      - Takes in a list of planes, a cursor object, and a connection object into the database, and adds the planes into the carriers table, if it doesn't already exist.
      - This returns nothing.
    - aero_dict_into_database(data_dict, port, cur, conn)
      - Takes in the dict from aero_data_into_dict, and loads the data into the appropriate tables given a IATA code for an airport, a cursor, and a database connection object.
      - This determines the rows to grab from the dictionary from aero_data_into dict, and adds the airports and routes to the list if they don't already exist. This will also call helper_planes_into_database for the carriers.
      - This returns nothing.
    - main()
      - Takes in nothing, and returns nothing.
      - Calls all the functions in order to get data and add to the database.
      - Has a list of source airports to grab data from in IATA_list, and will save the results from get_API_info into a json file to prevent maxing out requests to the API itself. This is capable of checking if there's a new airport in IATA_list, so it knows whether to read the data from a file, or to request data to make a new file.

- most_tracked_flights.py
    - get_API_info(url, params)
        - Takes information from the url and key for flightradar’s API
        - Returns dictionary with collected data from flightradar
    - open_database(db_name)
        - Creates a database with the filename taken in as a string and creates the tracked_flights table.
        - Returns connection with database and cursor object
    - flight_into_dict(data_dict, start, end)
        - Takes in the dict from get_API_info with starting and ending point to stay within the row limit
        - Returns a nested dictionary with the flight id as keys and the information of the flight name, number of clicks, the from city and to city in the inner dictionary.
    - flight_dict_into_database(data_dict, cur, conn)
        - Takes in the values from the dictionary and inserts it into the flights database under the tracked_flights table.
        - Returns none
    - main()
        - No arguments
        - Calling the other functions to input tracked flight data into the flights database.
        - Returns none

- flight_data_visualize.py
    - load_flights(database)
        - Takes in a string with the name of our database.
        - Returns a cursor object.
    - parse_carrier_data(cur)
        - Pulls data from flights to visualize the largest carriers between the cities of interest in the data_get file using the cursor object. The SQL query is responsible for the original calculation, but the data is modified further since aerodatabox considers “American” and “American Airlines” as two different carriers, among others.
        - Doesn’t return anything, but will produce a png file for the visualization, and a text file for the data used to create it.
    - parse_route_data(cur)
        - Pulls data from flights to visualize the most popular routes in the flights database.The SQL query handles calculations for this visualization without any further modifications.
        - Doesn’t return anything, but will produce a png file for the visualization, and a text file for the data used to create it.
    - main()
        - Takes in nothing, and returns nothing.
        - Calls all the functions in order to get data and add to the database.

- visualise_tracked_flights.py
    - load_flights(database):
        - Argument takes in the name of the database
        - Returns cursor
    - visualise_all_from_cities(cur):
        - Takes data from the tracked flights table in the flights database to visualize the top from cities with the highest number of clicks. It sums up the number of clicks if the cities are repeated.
        - Created a png of the visualization and saves it to most_tracked_airlines_chart.png and puts the values of from_city_list and from_clicks_list in a txt file, most_tracked_flights.txt
        - Returns none

- You must also clearly document all resources you used.

Date	Issue	Location of Resource	Result
4/10/23	N/A	https://rapidapi.com/aedbx-aedbx/api/aerodatabox/	Initial request script and documentation
4/10/23	N/A	https://rapidapi.com/apidojo/api/flight-radar1/	
4/12/23	Aggregate functions in SQL weren't adding up the correct columns and filtering data I didn't need.	Canvas page for SI 564: SQL and Databases.	Fixed the issue- when using aggregate functions like SUM() or AVG(), we filter information using GROUP BY clauses, not WHERE clauses.
4/17/23	Labels on vertical bar chart are illegible	https://www.pythoncharts.com/matplotlib/rotating-axis-labels/	Fixed the issue- the labels are more legible after tilting them by 30 degrees.
4/17/23	Labels on bar chart are cramped, and also illegible	https://www.freecodecamp.org/news/matplotlib-figure-size-change-plot-size-in-python/	Partial fix- changing the plot size spaced out the labels, but 88 data points is still a lot to graph
4/17/23	The bars blend in together and make it difficult to stand out	https://stackoverflow.com/questions/33476401/color-matplotlib-bar-chart-based-on-value and https://matplotlib.org/stable/gallery/color/named_colors.html	Fixed- allows data to be colored if it meets a certain threshold, and doesn't look garish when doing so