Flight Cost Analysis and Application

Matt Anderson, Cai John, Rojae Johnson, Andrey Karnauch, and Matthew Kramer

Abstract— **There exists a need for an application that helps travelers determine the most cost-effective airport within a user-defined driving radius to depart from. Many of the current resources available to users are either locked behind a price barrier, only deal with live flight data, or do not allow for comparisons across different airports within an area. To bridge this gap, we have built a tool based on historical data of domestic flights in the United States. The tool allows users to specify their location, a distance they are willing to drive, and a destination airport. It creates a series of charts for them to observe historical averages (including gas costs of driving to certain airports, if desired) and make an informed decision based on these outputs.**

I. Objective

Our team set out to create an application that advises users on lowest price points for airline travel from a user-defined region to the city of destination. The region is an area centered on a user-defined location that may include multiple airports. The goal is to determine which is the most cost effective airport to depart from while accounting for travel costs to the airports that are not the most proximal. This allows users to effectively plan budget travel based around their location, willingness to drive, travel dates and destination.

II. Motivation and Existing Work

This project is motivated by a gap in existing algorithmic tools for travelers. Currently, multiple resources exist for travelers to determine which airline is most cost-effective for travel between a set departure and destination airport. An example of this is Kayak [1], a service that queries several different airfare providers to determine the cheapest ticket prices for an origin and destination airport. A resource such as Kayak performs effectively for users seeking cheap tickets for the immediate future, but it falls short for users looking to plan flights far in advance based on historical trends or users willing to drive to obtain cheaper flight costs. Another service that addresses the shortcomings of Kayak is Faredetective [2], which allows users to specify an origin and destination airport and displays a trend of historical ticket prices for that trip. However, Faredetective falls short when it comes to smaller airports by providing outdated prices (e.g. the only historical data it has on flights from Knoxville (TYS) to New York (JFK) is from 2009). Furthermore, neither of these resources address the desire to view ticket costs across dynamic departure airports near a user. This is desirable as there can be significant price differences between neighboring airports. For example, searching for flights from Knoxville to Boston, we see a variation of over $120* depending on if you depart from Knoxville, Chattanooga, or Nashville. Clearly, if based in Knoxville it is most convenient to depart from this airport, but if driving to Nashville costs $30 the resulting savings is $90, a non-trivial amount to students and budget travelers. Thus this tool is motivated by and directed at facilitating reductions in travel cost for budget travelers, who may be unable to travel without these reductions.

*Current airfare prices as of November 2018 on Google Flights

III. Data Approach

A. Data Set

Our tool utilizes historical flight data provided by the Bureau of Transportation Statistics (BTS)[3]. The flight data is limited to domestic flights within the United States. Furthermore, the data does not specify departure or arrival dates for flights, but it does provide the yearly quarter of when the flight took place. Given that most airfare data sets are private or locked behind pricey APIs, the BTS data set gives some of the best insight into historical airfares that our team could find. The two tables with the relevant information for our application are **DB1BTicket** and **DB1BMarket**, and both of the tables have "no missing values or *some* missing values," where *some* is less than .005% according to the BTS table descriptions.

B. Data Size

The BTS website allows for quarterly downloads, with each quarter providing data on around six million flights. To keep the historical prices as recent as possible, our tool utilizes BTS data from each quarter from 2015 - 2018 (at the time of this application, only 2 quarters were available for 2018), with a total of 14 quarters. For efficiency purposes, rather than using six million rows for each quarter, we randomly sampled 400,000 flights from each quarter, which gave us a final data set consiting of around six million flights total.

C. Data Processing

To download 14 quarters of data (each consisting of **two** CSV files as dicussed in section A), a simple retrieval script was written that utilizes HTTP requests. The result was a directory consisting of 28 CSV files (14 from each table). The next step was to filter out all columns in our data set that were not relevant to the application, which allowed us to eliminate over half of the columns. From there, another script was written to perform the main data processing. This script loads the two related CSV files into a pandas dataframe, drops all irrelevant columns, and then performs a join on the two dataframes. The resulting dataframe is written out to a final CSV file for that specific year and quarter. This process was repeated for each year and quarter (14 times). The final step of data processing was moving the cleaned and joined data into a database.

IV. Platform Approach

To have a central database and uniform development environment, our team decided to build the application using Google Cloud (GC) services. The three services we ended up using were Google Cloud's Compute Engine, Cloud SQL, and Bucket storage. Furthermore, we settled on Python as the programming language for the application due to the wide variety of packages it offers along with the fact that each team member had or wanted some experience with the language.

A. *Compute Engine*

Google Cloud's Compute Engine allowed us to created an instance for the whole team to work out of. Each member was given their own home directory, and all Python packages were installed globally. This was a huge benefit due to the fact that there was no trouble setting up a local development environment for every team member. The biggest drawback of GC Compute Engine was some members having to learn VI/VIM as the text/code editor. However, the majority of the team was comfortable and actually prefered to develop in VI/VIM.

B. *Cloud SQL*

Google Cloud's SQL database provided a central data store for our team that was populated with the processed data mentioned in section III. This was a sufficient solution as the data took the form of a relational data set, allowing for easy database creation and population. Furthermore, the Cloud SQL instance served as a "local" database for our Compute Engine instance. This mitigated the latency of retrieving data from the database by not having to worry about connecting through a proxy or public IP address.

C. *Storage*

Google Cloud's Storage provided a central repository (bucket) for any supplementary files associated with the project. It was specifically utilized as a place to store all of the cleaned CSV data to prevent exceeding storage limits on the Compute Engine instance. Furthermore, since development was done on the Compute Engine instance, any graphical outputs by the tool were stored in a bucket for immediate viewing without having to perform any excess work (e.g. using scp to a local machine).

V. Tools Approach

The data analysis within the application does not require any advanced machine learning or statistical models because the requirement of displaying historical airfare trends to a user utilizes basic statistics and aggregation functions. However, we had to utilize a variety of tools and packages to reach our goal of dynamic airport selection. To obtain the geographical coordinates of both our user and airports, our application uses the **Geopy** [4] library in conjunction with its Geocode module and Google's Geocoding API. This allows the application to receive any form of address data (including airport codes) and return the assumed location and its coordinates. From there, the **GeoBases** [5] library returns all IATA-coded airports within a user-specified mile radius and location. Furthermore, if the user wants to include the cost of driving from their location to the origin airport in the trend calculations, the **Googlemaps** [6] library calculates driving distance between the user's location and each airport. Once this information is collected, the application carries out analysis through either aggregate SQL queries or basic analysis using the popular **numpy** library. Lastly, all plotting and visualization is performed using **matplotlib**.

VI. Time-line

To stay on pace with our project goals, we utilized GitHub Issue tracking to create sprint milestones. From there, issues were created for each sprint to break up the work for the two week intervals.

A. *Sprint 1*

This sprint was used to set up our Google Cloud environment. This included both setting up the Compute Engine and Cloud SQL instances. From there, we analyzed our data set to exclude unnecessary columns and used that information to create our database schema in Cloud SQL.

B. *Sprint 2*

This sprint was used to populate the database and execute basic queries on it using Python. The first part of this included creating the two data extraction scripts: downloading and cleaning. After all the data was cleaned and compiled, the Cloud SQL database was populated. From there, the data analysis team learned how to communicate with the database using Python and MySQLdb and executed basic queries against it.

C. *Sprint 3*

This sprint was used to shift focus from data processing to data analysis. This included writing modular functions that took origin and destination airports and returned average airfare costs for those flights. The average costs were also plotted using matplotlib. Furthermore, we explored potential APIs to aid with the dynamic airport generation. After deciding on Geopy and GeoBases, the function for obtaining all airports near a user was written.

D. *Sprint 4*

This sprint was used to finalize the output of the application and generate more results. We took the modular functions and made them respond to user-input rather than hard-coded values. Furthermore, three more charts were added to the application's output.

VII. Results

The resulting application meets all of the main goals described in sections I and II and leaves the decision making and interpretation up to the user. The application is CLI-driven, allowing the user to specify their **location**, how far (**miles**) they are willing to drive to reach an airport, their **destination**, and whether or not they would like to include **gas costs** in the calculations. The outputs of the application are heavily focused around average ticket prices and are tiered in terms of generality. Figure 1 displays the average cost of tickets per passenger from the airports within the users specified radius. These average prices are not tied to any specific destination and reflect general trends from these origin airports to any other airport in the United States. The Figure 2 output builds upon the first one and specifies the same ticket averages, but incorporates the specific destination into the analysis. This allows a user to see how ticket prices vary across airports with flights directly to their location.

Fig. 1: Average Cost per Airport in General

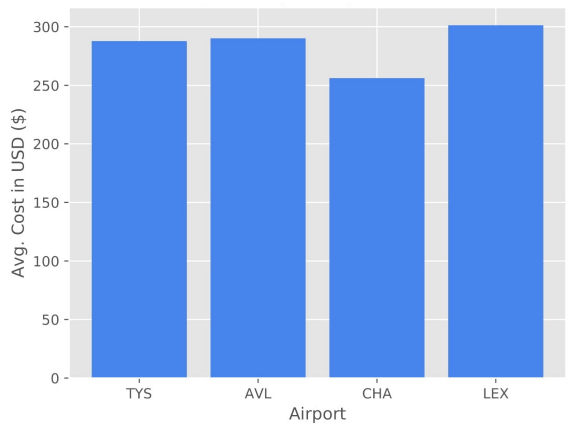
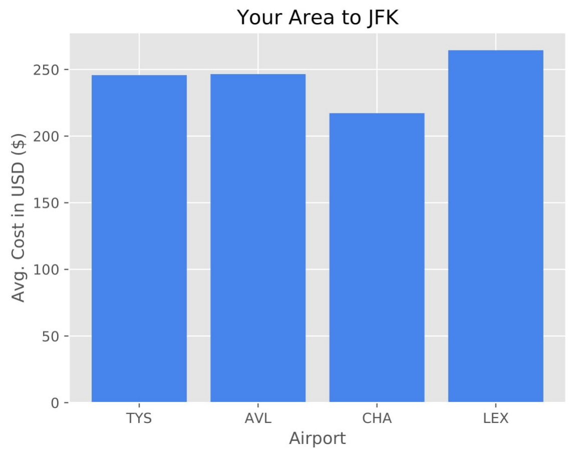
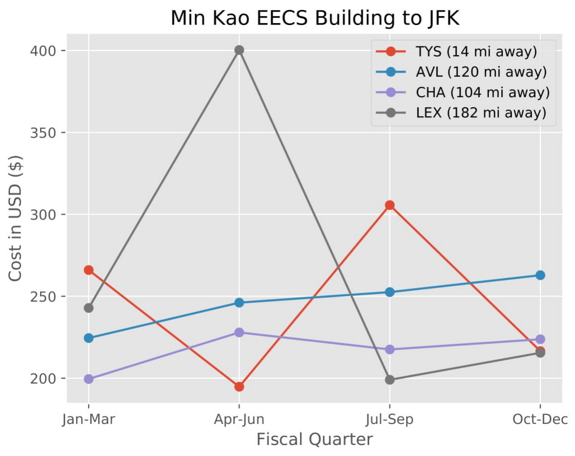


Fig. 2: Average Cost to Destination per Airport



Lastly, in order to give the user a chance to plan ahead based on time of year, the application displays quarterly ticket trends in Figure 3. This provides users with insight into the averages seen in Figures 1 and 2. For example, a user might look at Figure 2 and decide that flying out of CHA is the most cost effective option. But if the user intends to fly out between the months of April and June, Figure 3 shows that the user might be able to save even more money by flying out of TYS. Together, these graphical outputs allow users to make decisions based on their own needs and interpretations.

Fig. 3: Quarterly Breakdown of Ticket Prices



VIII. Future Work and Limitations

A. *Stretch Goals - Future Work*

Based on our original proposal for this project, there were a couple main stretch goals that we failed to meet that would greatly benefit this application. The first goal revolves around implementing a Graphical User Interface via a website rather than a CLI-driven (Command Line Interface) application. In conjunction with this, we feel that the user would also benefit from recommendations from the application itself, rather than leaving it up to the user to interpret the trends. One benefit of this goal obviously comes in the form of a cleaner and more intuitive interface for the user. Furthermore, it also increases the accessibility aspect by making it publicly available via a website rather than a script inside a GitHub repository. The second stretch goal we had was to incorporate live airfare data into our database to provide trends for the immediate future, rather than relying solely on historical data.

B. *Limitations*

Throughout the development process, our team came across two other pieces of future work that would greatly benefit the application. The first one boils down to sampling more than 400,000 flights from each quarter. Given that BTS provides more than 5 million flights from each quarter, the application should utilize all of those to ensure all potential airport combinations are included. Figure 4 below shows how the current data set does not contain any flights from PIB to LAX. However, Figure 5 shows that flights out of PIB to other airports do exist. Since we know that PIB does in fact do flights to LAX, this problem exists due to the small sampling of the original data set. Utilizing more of the original data set would result in more records, decreasing the chance of missing airport combinations, and

Fig. 4: Average Cost per Airport to LAX

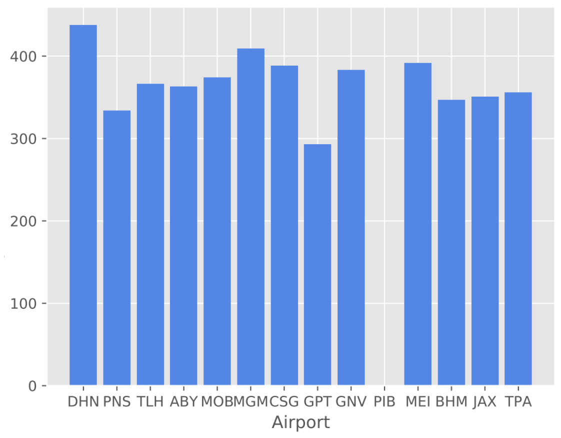
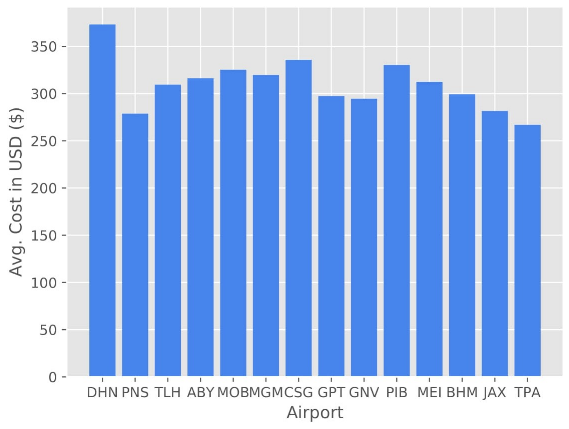


Fig. 5: Average Cost per Airport in General



increasing overall accuracies in terms of mean ticket prices. The introduction of more entries into the database comes with its own limitation as well. The speed of the current queries take around one second to complete, and this would definitely increase if we scaled the data set size from the current 5.4 million to around 54 million rows. Solutions to this problem come in the form of either minimizing the number of queries the application currently performs or indexing the database to speed up query times.

IX. Project Management

A. *Andrey Karnauch*

Andrey was the team lead and was initially responsible for planning out the four main sprints and organizing the project's GitHub repository. He was also the most familiar with the data set, so he spent most of his time working on all steps of the data collection, cleaning, and insertion. Andrey created the Cloud SQL Database instance along with the actual database schema. He wrote the automatic download script to collect all desired quarters of data. From there, he worked on the **unzip_and_merge.py** script to clean the data and export it to the GC Storage bucket. Andrey also performed the insertion of cleaned data into the SQL database. In terms of data analysis, Andrey helped with the Geopy, GeoBases, and Googlemaps APIs because he had worked with them before. He created the function that calculates driving distance and gas costs to each airport.

B. *Matthew Kramer*

Matthew was heavily involved with the data analytics for this project. He set up the base python script that the program was run on for easy editing and addition. He was responsible for pulling relevant data that will be used during the analysis section of the project. This included two types of data retrieval based on what was most needed by another function. He assisted the team with data cleaning and deciding which data was relevant to use for our analysis. He worked on some of the functionality within the analysis Python script and assisted with some of the analysis.

C. *Cai John*

Cai was a member of the data analytics team. Matthew and Cai worked together to define a modular structure for the analytics pipeline building off functions that query the database. This design structure was chosen to avoid queries in every function. Building off the results of the analysis, Cai took responsibility for creating the visualizations that would be presented to the user, aiming for easy and efficient interpretation. He also contributed by giving input on the data cleaning steps and guiding the direction of the project from its onset.

D. *Rojae Johnson*

Rojae was responsible for setting up and managing the Gcloud instance as well as making sure that any relevant libraries were installed and available to everyone. This includes handling API access and permissions to Google services and making sure any keys were allocated to the programs when necessary. Once this was complete, he helped troubleshoot any issues that resulted from ssh access to the server. On the coding side he provided the starting points and proof of concepts for many of the scripts that the project was built from including the interfacing with the SQL server through python scripts and accessing Google APIs.

E. *Matt Anderson*

Matt helped with the data cleaning and deciding what data the team would use for calculations and analysis.

References

[1] Search Flights, Hotels and Rental Cars, `www.kayak.com`.
[2] "Fare History Charts". Discount Airfares and Airline Tickets, `www.faredetective.com`.
[3] Bureau of Transportation Statistics, `www.transtats.bts.gov/tables.asp?db_id=125&DB_Name=`
[4] "Geopy." PyPI, `pypi.org/project/geopy`.
[5] "GeoBases." PyPI, `pypi.org/project/GeoBases`.
[6] "Googlemaps/Google-Maps-Services-Python." GitHub, 13 Nov. 2018, `github.com/googlemaps/google-maps-services-python`.