**Technical Write-up report for ETL-Project**

Downloaded Data from RedFin in CSVs.

Check for duplicates using MLS Listing as Primary Key, since that is unique

**For new construction listings, it didn’t have a MLS # so we deleted those from our dataset.**

<http://www.city-data.com/zips/94102.html>

94102, 94103, 94104, 94105, 94107, 94108, 94109, 94110, 94111, 94112, 94114, 94115, 94116, 94117, 94118, 94121, 94122, 94123, 94124, 94127, 94129, 94130, 94131, 94132, 94133, 94134, 94158.  
  
Read more: <http://www.city-data.com/zipmaps/San-Francisco-California.html>

Ran the data by zip code

**Analyzing Search Engine Results Pages**

For our project, we reviewed search engine result pages data related to travel. Specifically, our data consists of search terms, like “flights to hong kong” and the various websites, like Expedia & Travel Advisor, that had the quickest search result queries, which is under Search Time.

At first, we attempted to load the data in Postgres but the data included escape characters that prevented the import. Therefore, we pulled the data into a Juypter Notebook to analyze the data.

We are going to be sharing a way to get SERP data and have it in a DataFrame (table / csv / excel sheet) for analysis, and in an automated way.

We will be using the programming language Python, so there will be some coding involved.

**Importing the data**

We were able to import the flight\_tickets .csv files into postgres. We created the database and tables to support the import of the data. There were escape characters that initially posed a problem in importing the data into Postgres however, there was an exception to be selected which eased the import.

Also included with the flight\_tickets.csv file was a SERP\_flights.csv that we analyzed. The data was not easy to clean up and we could not use it further however, we wanted to mention that as it caused some delay on our end.

We finally found a SQLite database on flights data and some files on Airbnb data and RedFin realestate data that we could work with.

**Handling the data**

We will be using Postgres, Jupyter Notebook and Python packages for our work:

* [pandas](https://pandas.pydata.org/): For data manipulation, reshaping, merging, sorting, etc.
* Sqlite3: To create, query and update the database
* Postgres – loading the data into the database
* Matplotlib for plotting graphs and charts

**flights\_tickets/Serp\_flights data**

A few notes on the different columns available:

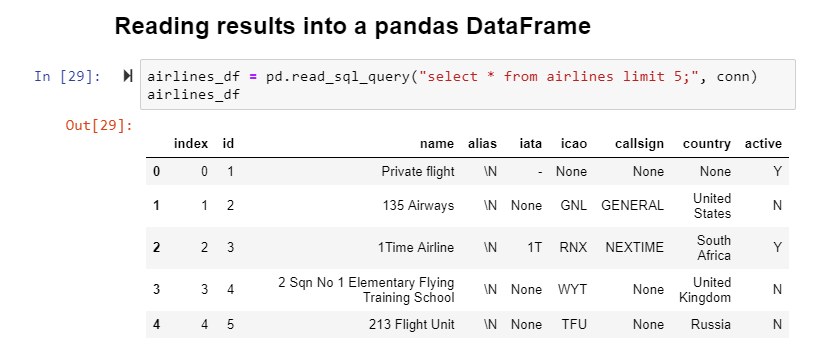
"queryTime" is the time that the query was run (when the request was created). This is different from "searchTime" which is the amount of time it took Google to run the query (usually less than one second). Most of the main columns will always be there, but if you pass different parameters you will have more or less columns. For example, you would have columns describing the images, in case you specify the type of search to be "image"

**The dataset**

For the flights\_tickets and serp\_flights, we obtained datasets from Kaggle. We were able to obtain .csv files with recent data.

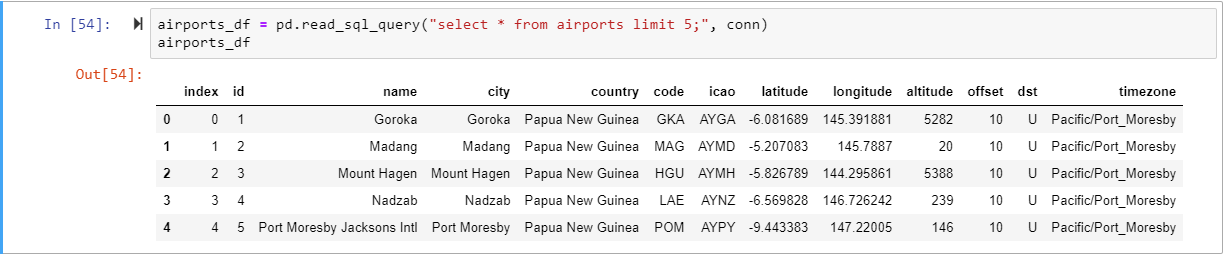
For all of the SQLite database analysis, we found a database called flights, that contains three tables – airlines, airports and routes.

Airlines data:



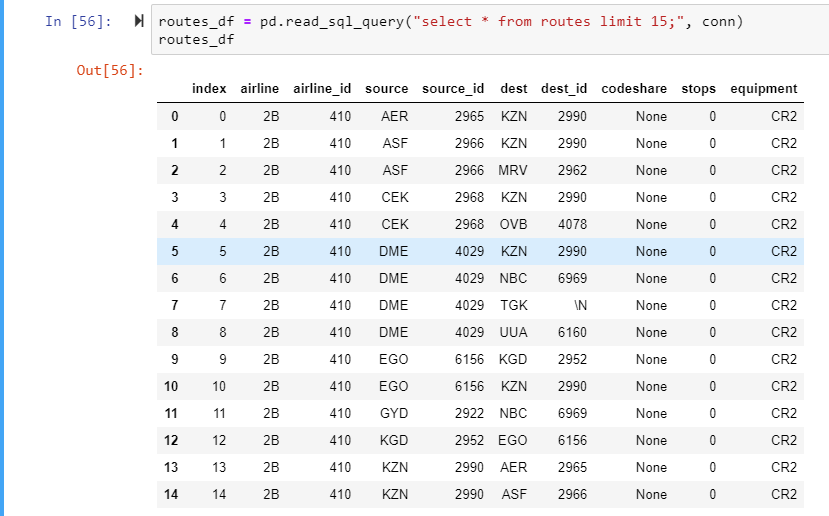
Each row is a different airline, and each column is a property of that airline, such as name, country, etc. Each airline also has a unique id, so we can easily look it up with the id as required.

Here are a few rows from the Airport table:



Each row corresponds to an airport, and contains information on the location of the airport. Each airport also has a unique id, so we can run queries with the same.

Routes table:



Each route contains an airline\_id, which is the id of the airline that flies the route, as well as the source\_id, which is the id of the airport that the route originates from, and dest\_id, which is the id of the destination airport for the flight.

Querying database rows in Python

The read\_sql\_query function would read the results of a SQL query directly into a pandas DataFrame. It automatically reads in the names of the headers from the table. It creates a DataFrame, so we can quickly explore the data. This function gives us the advantage to manipulate the columns.

Modifying database rows

We used sqlite to modify a SQLite database by inserting, updating, and or deleting rows. First we inserted a new row in the airlines table. We specified 9 values to insert, one for each column in the airlines table. This added a new row to the table.

Passing parameters into a query

We added a few columns of data to the airline table, updated the rows and passed queries to view the inserted data. The inserted columns of data were added to the airlines table.

Next we moved on to creating tables, we created a new table daily\_flights and inserted 6 columns of data. We performed a query to check the updated table with the inserted row, then performed a query to delete the data. We also queried to check whether the data from the table was deleted.

Next we moved on to creating a table in daily\_flights and loaded the data into DataFrames and loaded the same data into the SQL database. We created a table to represent each daily flight on a route with the following columns:

* id — integer
* departure — date, when the flight left the airport
* arrival — date, when the flight arrived at the destination
* number — text, the flight number
* route\_id — integer, the id of the route the flight was flying

We also performed some queries to alter tables by adding columns to existing tables within airlines and daily\_flights.

Creating tables with Pandas, we created a DataFrame first to export it to a SQL table.

We created the DataFrame with date time values to be entered into the daily\_flights table. Then, we’ll be able to call the [to\_sql](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_sql.html) method to convert df to a table in a database. We set the keep\_exists parameter to replace to delete and replace any existing tables named daily\_flights. We can ten verify that everything worked by querying the database.

Altering tables with Pandas

Using our airline example, we decided to add an airplanes field to the airlines table that indicates how many airplanes each airline owns. Alter queries are immediately executed and do not require the commit command. All the columns are set to null in SQLite (which translates to None in Python) because there are not any values for the column yet.

This was our technical report with working with data in a SQLite database using Python and Pandas.