**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're going to be sharing a way to get SERP data and have it in a DataFrame (table / csv / excel sheet) for analysis, on a large scale, and in an automated way.

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

**Importing the data**

**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

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

A few notes on the different columns available:

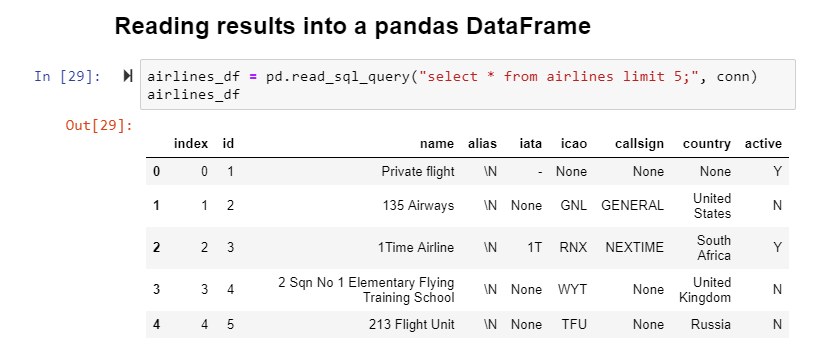
"queryTime" is the time that the query was run (when the request was created). This is different from "searchTime" which is the amount of time it took Google to run the query (usually less than one second). Most of the main columns will always be there, but if you pass different parameters you will have 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.

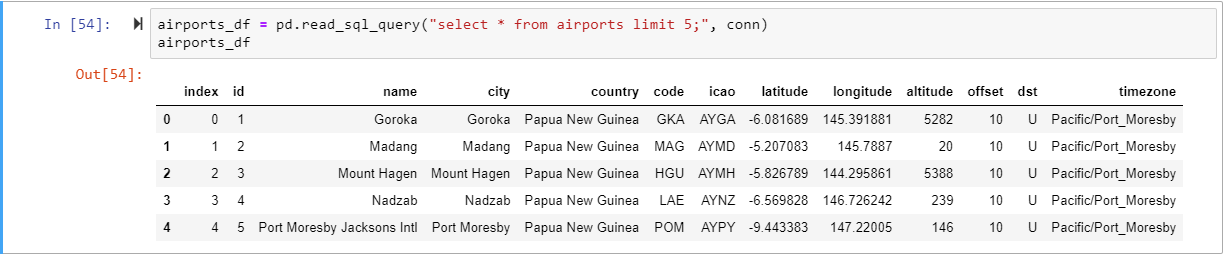
For all of the SQLite database, we found a database called flights, that contains three tables – airlines, airports and routes.

Airlines data:



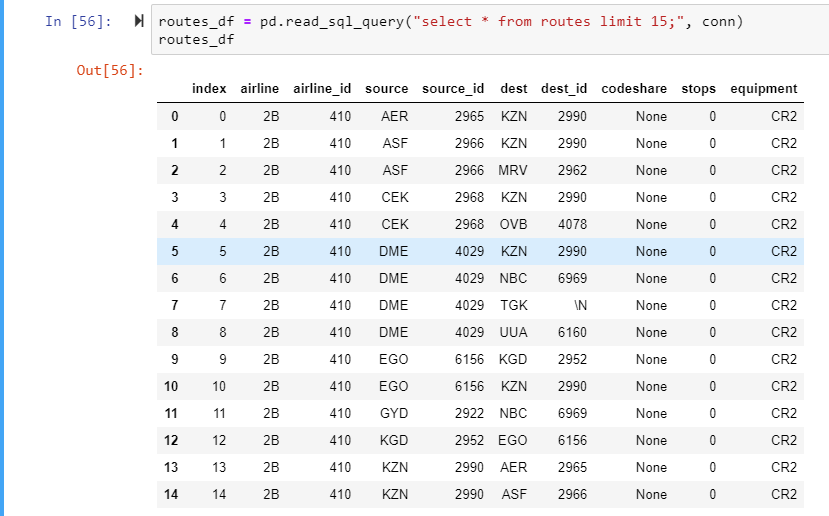
Each row is a different airline, and each column is a property of that airline, such as name, country, etc. Each airline also has a unique id, so we can easily look it up with the id as required.

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.