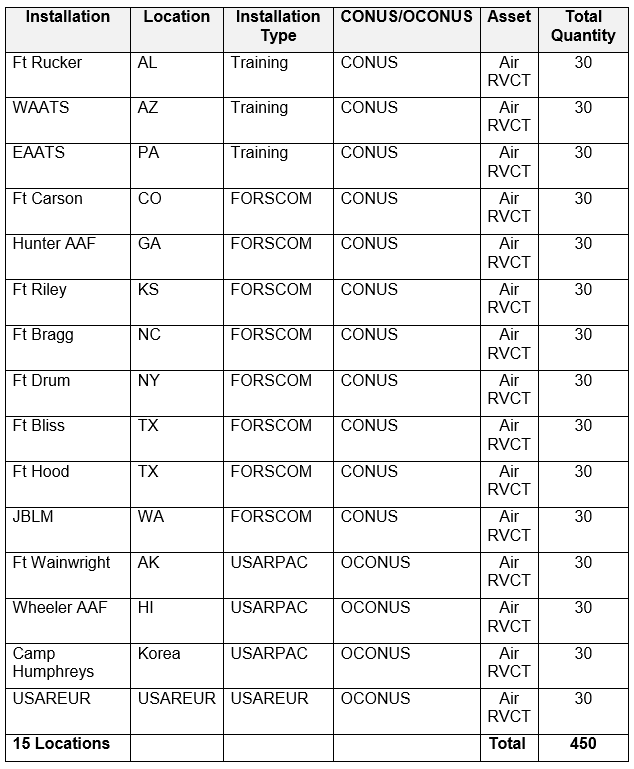
**TeamBAM\_ETL\_Project**

**ACME Defense Customer Locations**

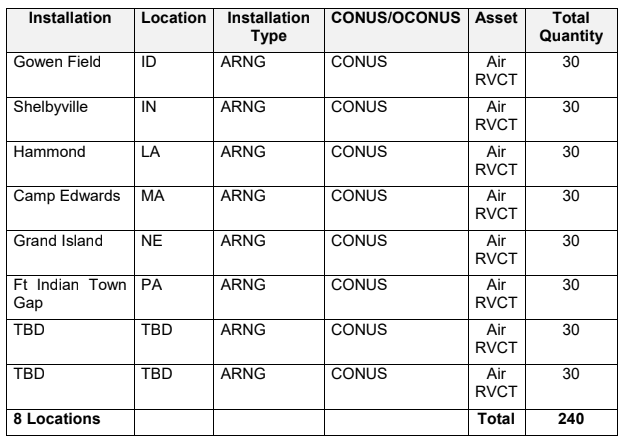
OBJECTIVE:

The ACME Defense Company needs to create a database of all military sites it will provide services to. In order to do this, they require an API that provides lists of sites by which customer organization it belongs to (command), region in which it is located, and type of RVCT product required to be serviced. In each case, the output will be a list of all the filtered sites, their specific location in latitude and longitude coordinates, and the quantity of RVCT products by type requiring service.

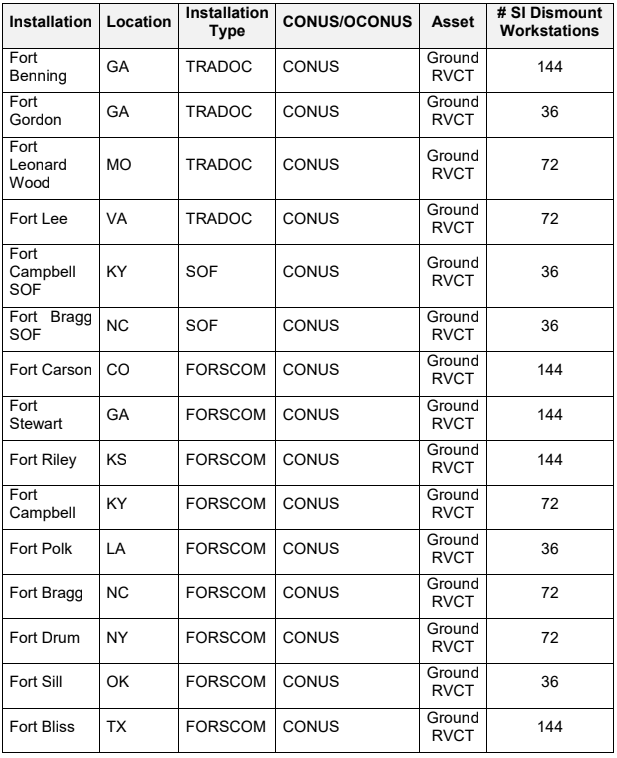
In order to do its job, ACME needs specific location information for each customer location identified in the following tables (from the PDF file).

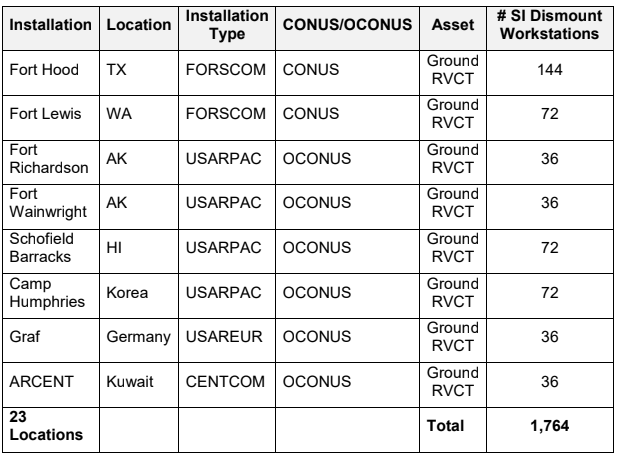


**Table 1: Maximum Active Component Air RVCT and Semi Immersive Persistent Training Capability**

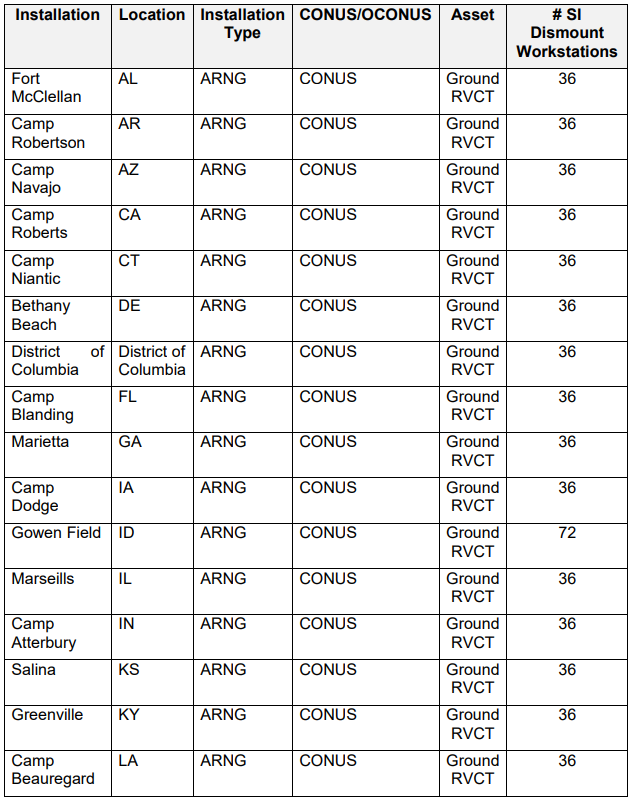
****

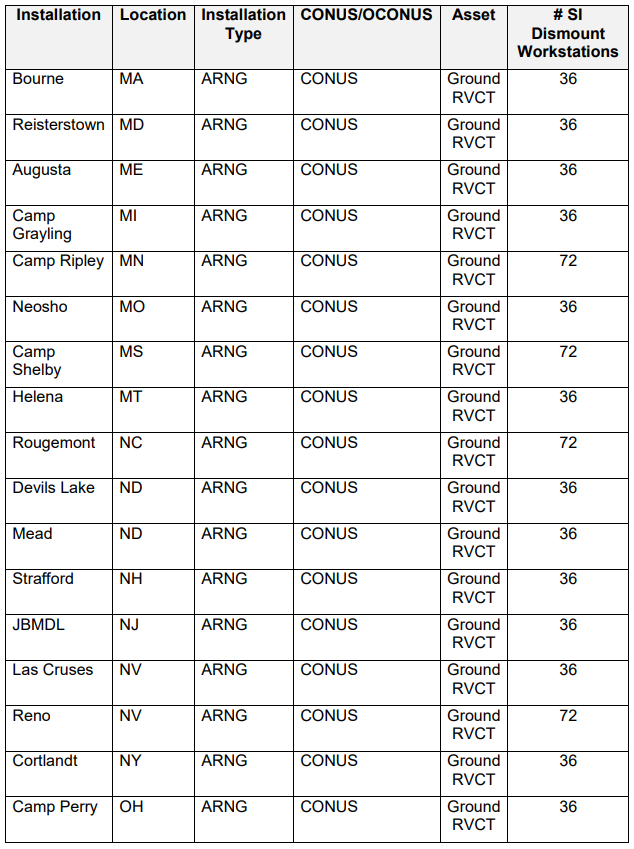
**Table 2: Maximum Air RVCT Guard and Reserve Capability**

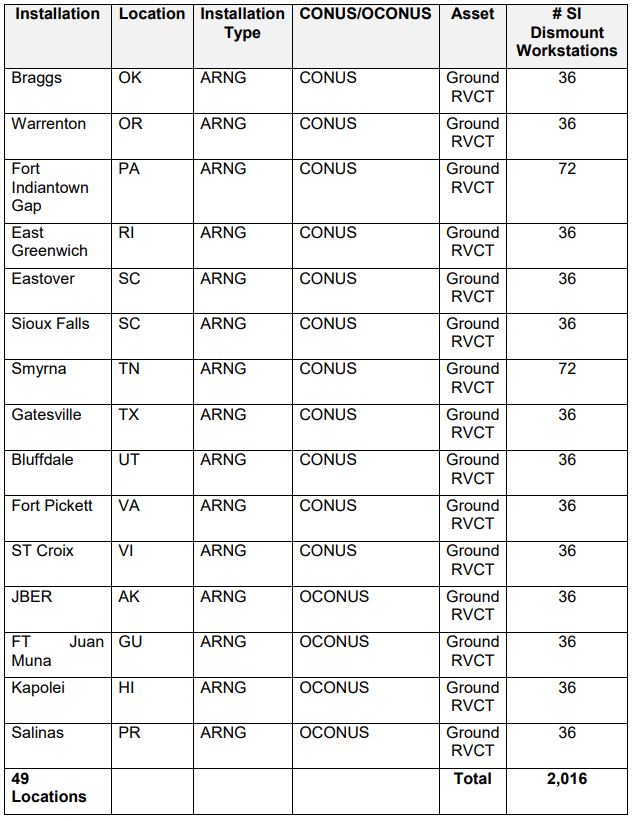
****

****

**Table 3: Maximum Active Component Ground RVCT and Semi Immersive Dismounted Soldier Workstation Training Capability**

****

****

****

**Table 4: Maximum Ground RVCT and Semi-immersive Dismounted Soldier Army Reserve and Army National Guard Capability**

Unfortunately, the only location data provided by the customer was in the following map. So we will need to scrape the specific location data from the World Wide Web.

A picture containing text, map

Description automatically generated

**Diagram 1: Military Installations**

APPROACH:

The tables of customer location and product information were dumped from a pdf file to a text file. ACME has taken the first step to convert the data to a CSV file but it has table formatting that needs to be cleaned up.

Team BAM has been contracted to extract the supplied base data, correlate it with location information (lat,long), and load it into an SQL database they can then use to query bases by state and region, and return the location information accordingly.

TEAM TASKS:

1. EXTRACT & TRANSFORM 1: Rob will use Python and Pandas to extract the data from the CSV file and export all entries out to correctly formatted CSV files for each column.
2. EXTRACT & TRANSFORM 2: Rob will import all CSV file data into a dataframe and export all entries out to a formatted CSV.
3. EXTRACT & TRANSFORM 3: Mike will use web scraping (Beautiful Soup) via a Google Search URL for each base location (name/state) into the dataframe, using Base Name and State. He will then convert the dataframe to a dictionary and export as 'Base\_Locations.csv'.
4. LOAD 1: Lillian will import the Base\_Locations.csv into MySQL and create the SQL queries required to recreate the Base\_Locations table.
5. LOAD 2: Lillian will create a query for showing all sites (base name, lat/long, type, quantity) by either state, command, region, or type.
6. LOAD 3: Lillian will recreate the Base\_Locations table and populate a SQLite database that Flask can use from the repo directory.
7. REPORT: Rob will create a Flask based website to provide all of the results pulled from the SQLite database file using Lillian�s queries now in Python.
8. REPORT: Team will compile the final report to include the following;

a) Sample data imports

b) Formatting challenges and methods

c) Output data format

d) Jupyter python code

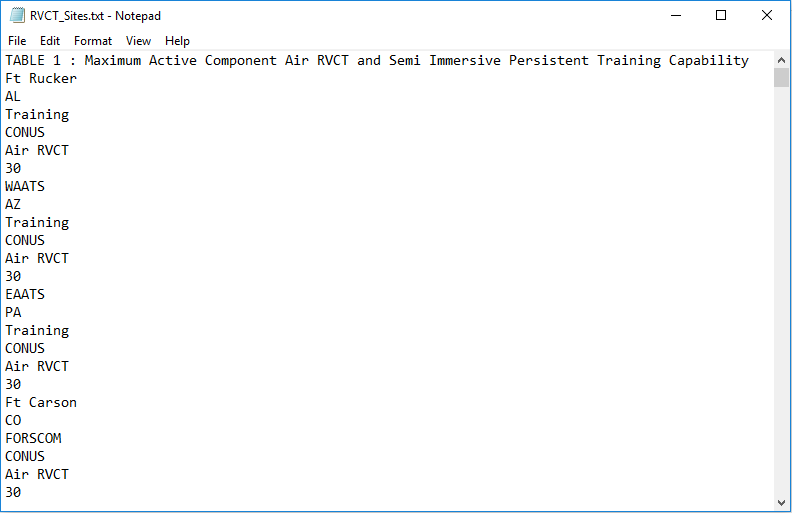
e) Google Search scrape method - search on '{ base\_name } {state} lat long' and extract latitude, longitude from tag div class="Z0LcW"

f) MySQl recreate schema SQL - show create table 'base\_location'

1. BONUS: Flask based API for search by state, command, region, or type
2. BONUS: Use PyPDF2 to extract table data from the pdf file

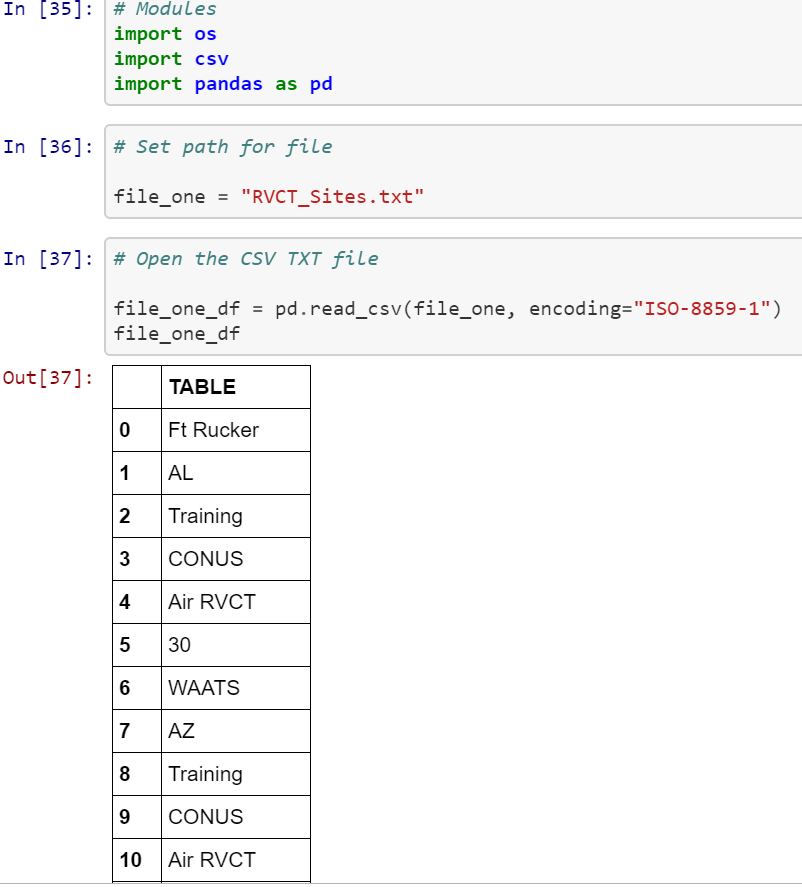
TASK DESCRIPTIONS:

1. EXTRACT & TRANSFORM 1: Rob will use Python and Pandas to extract the data from the CSV file and export all entries out to correctly formatted CSV files for each column.

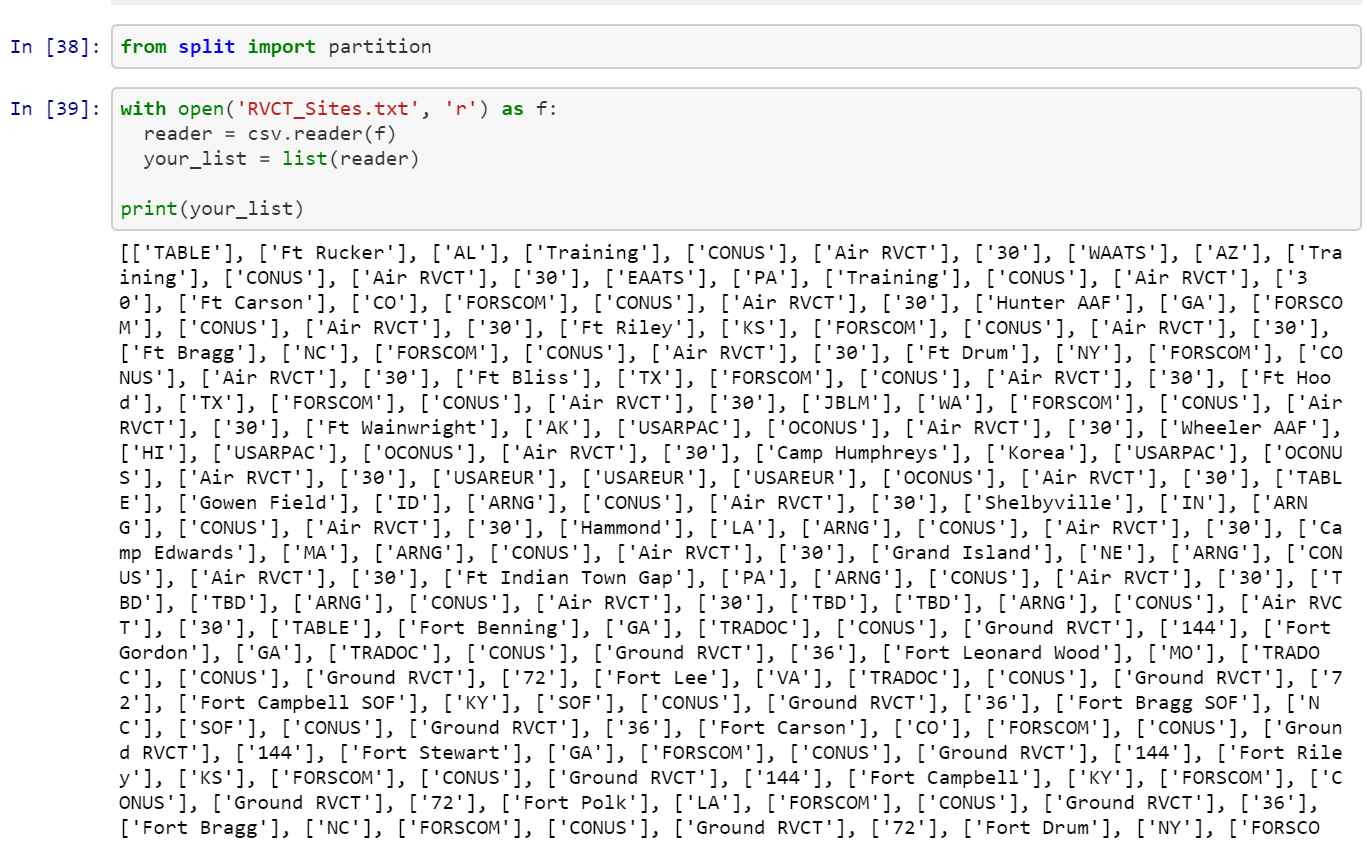


Observation: Data extracted manually from a PDF file places all columns and cells into a single column. The text is unstructured, there are no column labels, and table names are inserted at the beginning of each table of data extracted. Each base name consists or either a Fort, a Camp, a city name, or an abbreviation. This might make transformation and scraping of location data difficult later.

Method: Use Jupyter Notebook (Base\_Data.ipynb), and python, pandas, and CSV, to load the txt file as a CSV, compare the data order with the table structures provided in the PDF (Tables 1-4).



Now load the entire CVS into a list from which we can iteratively process.

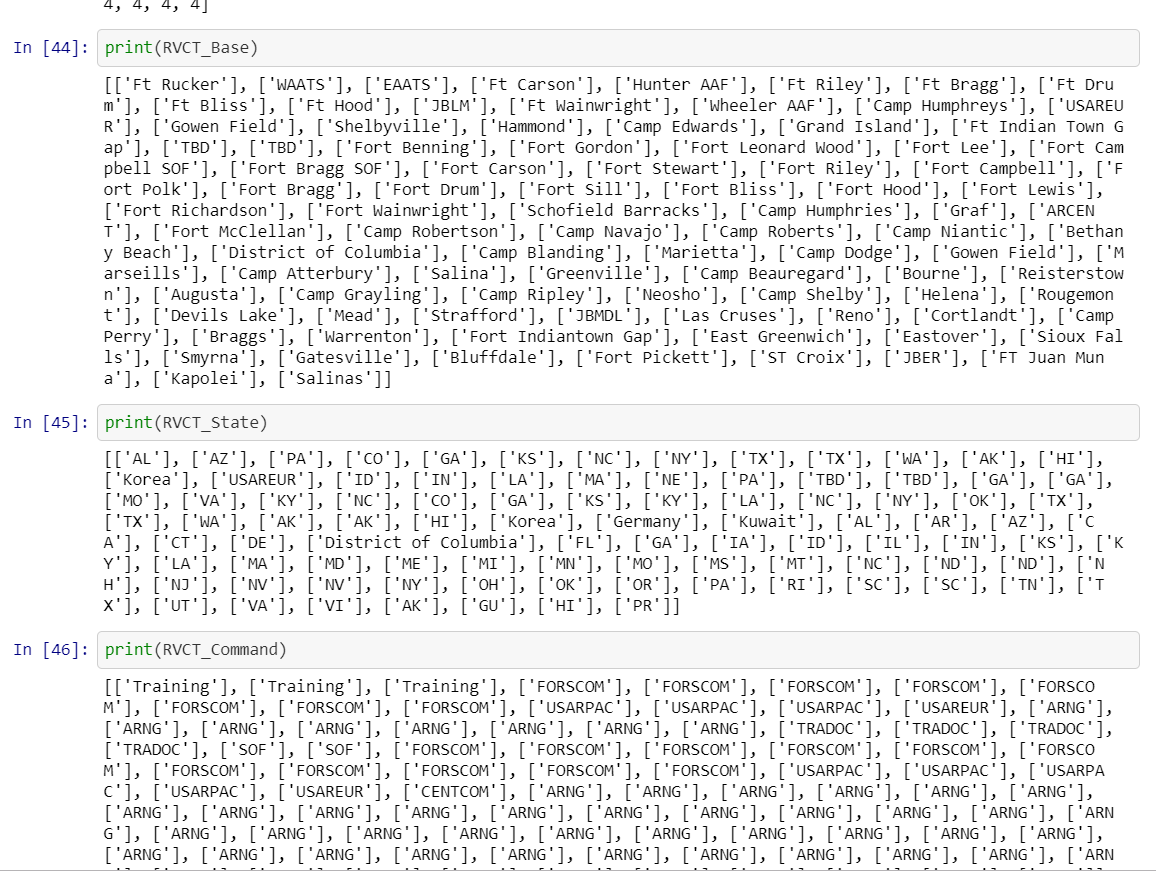


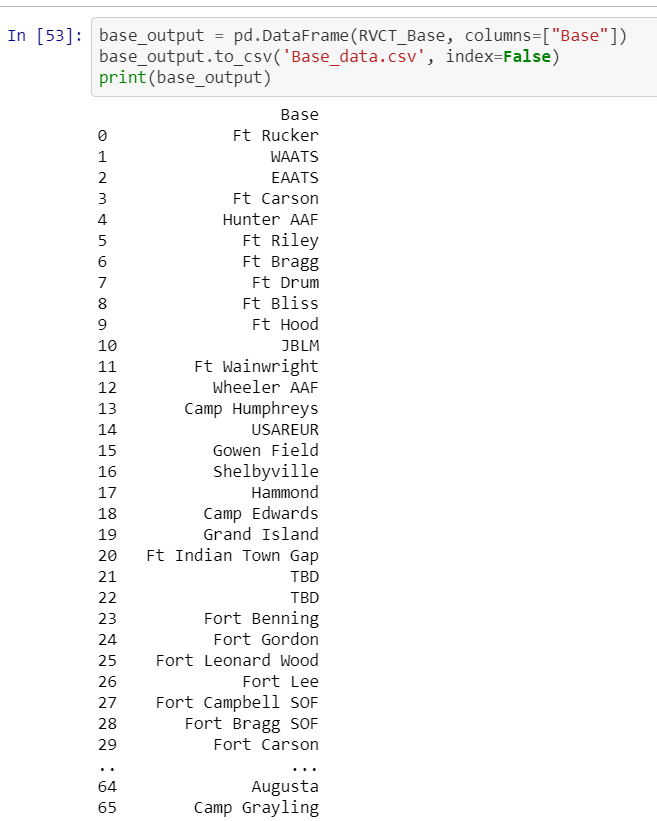
We simplify the transformation of data by replacing all table text with simply ‘TABLE’, and count the number of tables to

confirm we can count each table as it occurs. We confirm there are four tables. We can now begin to pass each list element into a unique column list with itertools.



Now we can confirm that each column is complete and ready for export to separate and/or consolidated CSV file/s in a structured format.





1. EXTRACT & TRANSFORM 2: Rob will import all CSV file data into a dataframe and export all entries out to a formatted CSV.

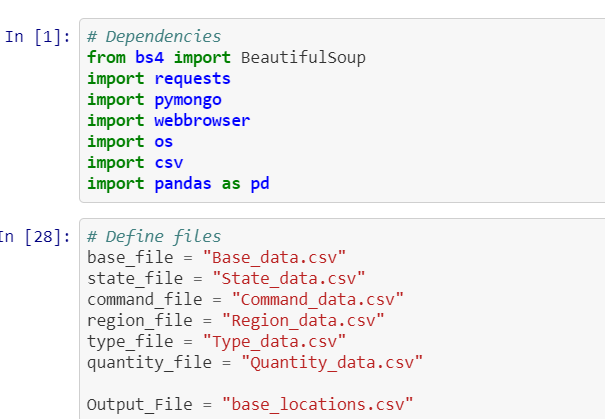
Now we use a new Jupyter Notebook file (Base\_Locations.ipynb) to read each column of CSV data into dataframes, and prepare to aggregate the data

into a new dataframe with column headers, but also including the location data ‘Latitude’ and ‘Longitude’ into which we will be scraping.



1. EXTRACT & TRANSFORM 3: Mike will use web scraping (Beautiful Soup) via a Google Search URL for each base location (name/state) into the dataframe, using Base Name and State. He will then convert the dataframe to a dictionary and export as 'Base\_Locations.csv'.

Mike now takes the same jupyter notebook file (Base\_Locations.ipynb) and attempts to scrape the location data from Google, now including BeautifulSoup, requests, and webbrowser



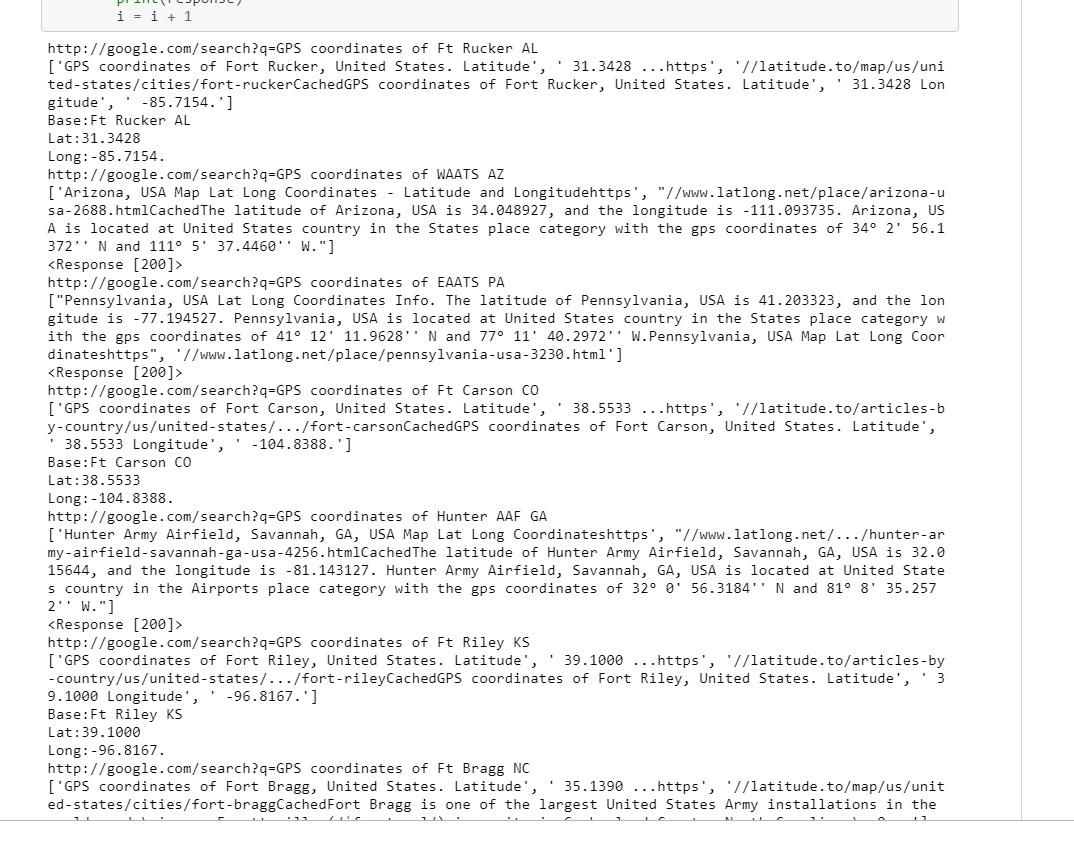
Now we use a loop to try to scrape the GPS coordinates for each base and state passed into a Google search URL



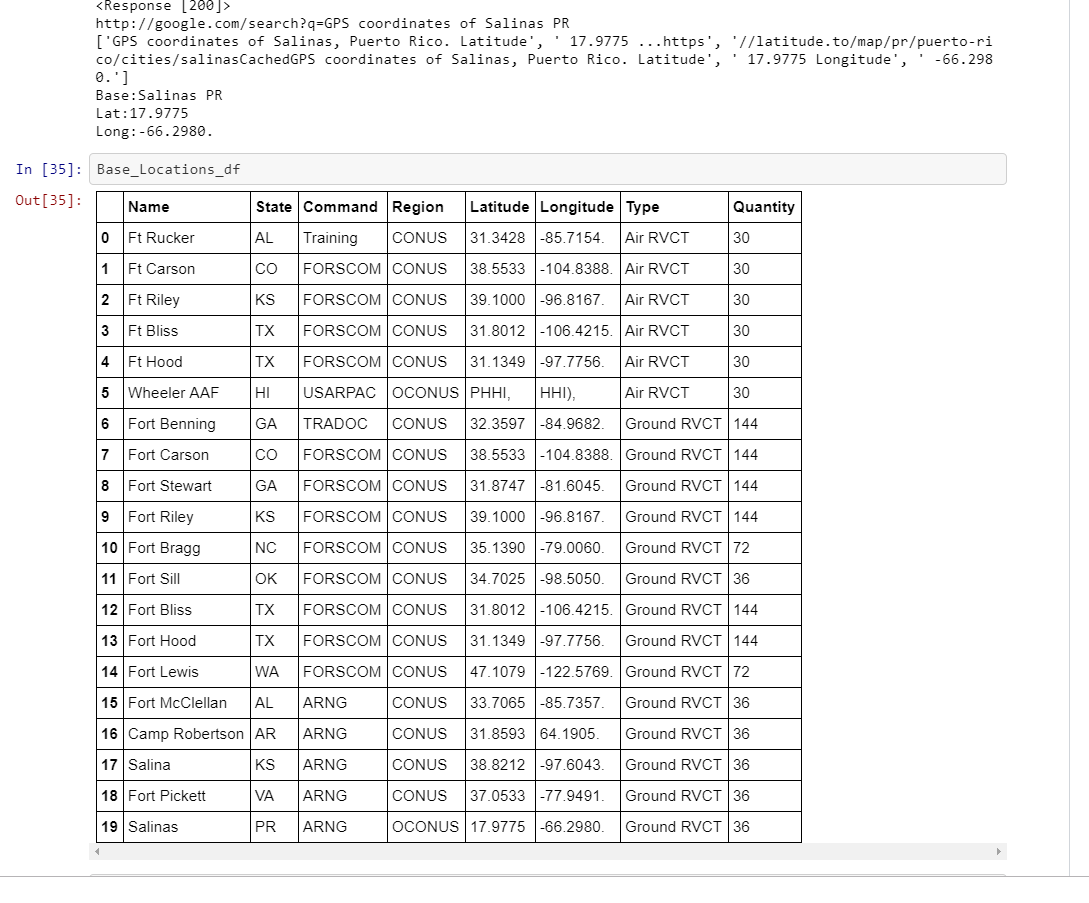
At this point Mike discovers that not all location data is scraped from Google Search. But we do manage scrape for all ‘Fort’ bases.

In the following screen dump, we are printing the URL created for each base, the first list of data scraped from the response, the base name and state,

and finally the Latitude and Longitude collected from splitting the response string.



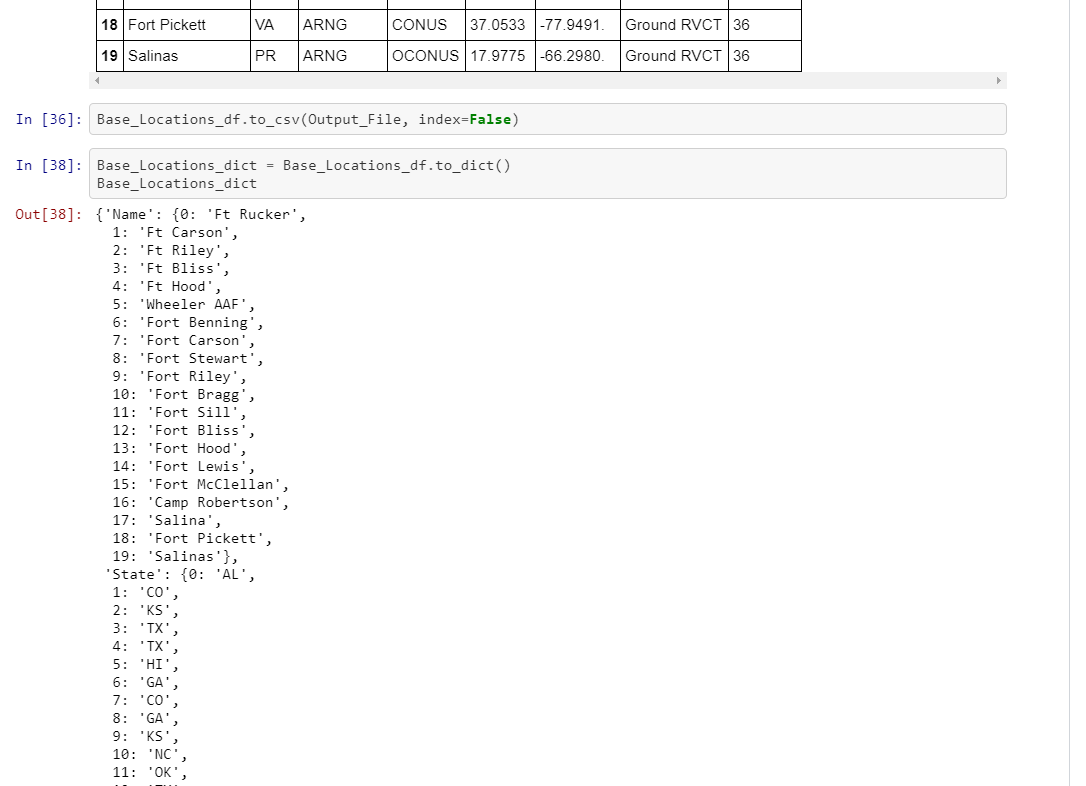
Once the scrap loop completes passing all base names, the results are stored in a single dataframe and output to check how many were parsed successfully.



At this point, Mike encounters his first significant challenge. For each URL passed, the returned Google search HTML is not exactly the same structure.

This explains why we only collected 19 of the 93 locations into the dataframe. In order to provide input to Lillian’s task, we went ahead and exported this data

to a dictionary and out to a new CSV file (base\_locations.csv).



While Lillian moved forward with loading this data into MySQL, Mike continued to try to scrape based on span class = ‘st’, and then div class="Z0LcW".

Subsequently, Mike discovered two more challenges. These included an error not passing %20 in each space within the URL string, and a persistent

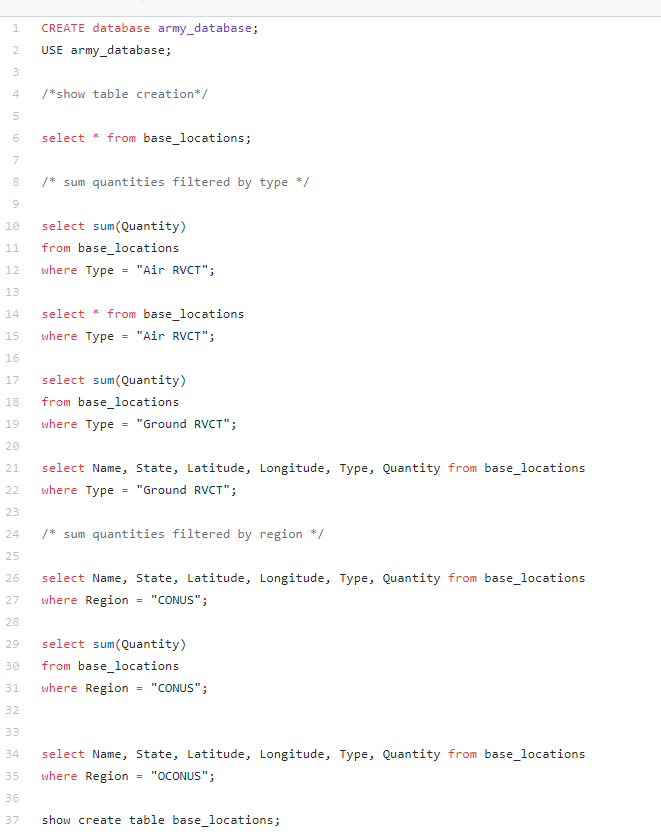
denial of service (error 503) by Google after running the python more than twice per computer in a day. This prevented our team from successfully

scraping the remaining data by the original submission date. Our TA did find a work around but time did not permit to investigate and test an

alternative approach or use an API.

1. LOAD 1: Lillian will import the Base\_Locations.csv into MySQL and create the SQL queries required to recreate the Base\_Locations table.

Lillian imported the base\_locations.csv file directly into MySQL so she can script and test her SQL queries to extract the required reports by RVCT type and region.



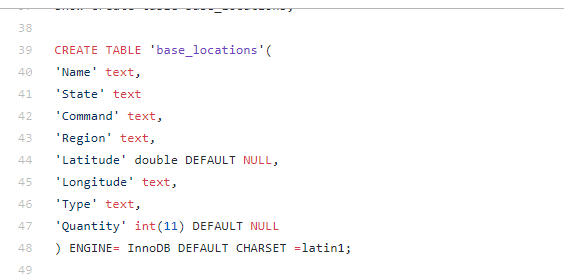
1. LOAD 2: Lillian will create a query for showing all sites (base name, lat/long, type, quantity) by either state, command, region, or type.

Then she completed the set of queries to complete all of the required reports (by command).



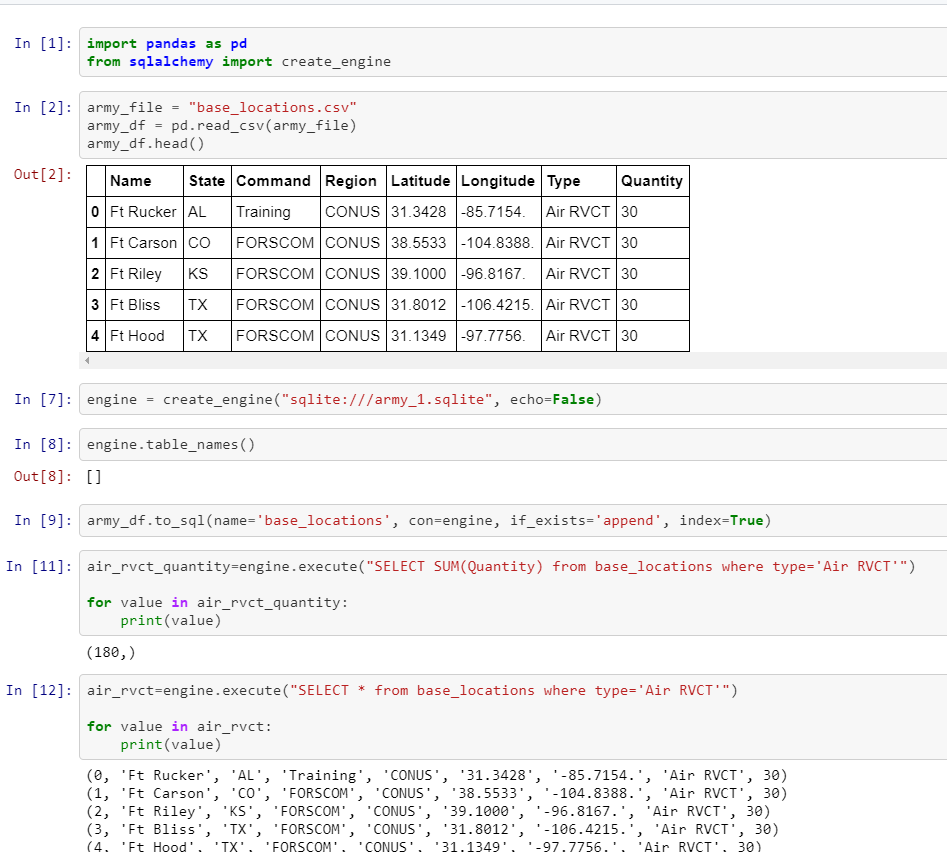
1. LOAD 3: Lillian will recreate the Base\_Locations table and populate a SQLite database that Flask can use from the repo directory.

She also used show create table *base\_locations* (table name) to copy the query code required to rebuild the database in SQLite (for the FLASK python file).



With the queries complete and tested, Lillian then migrated the SQL scripts to a python in order to recreate the tables and data within an SQLite database that Flask can use.

To test her code, she starts in Jupyter Notebook (army\_sqlite.ipynb) and creates the SQLite file (army\_1.sqlite).



1. REPORT: Rob will create a Flask based website to provide all of the results pulled from the SQLite database file using Lillian’s queries now in Python.

Whilst Mike and Lillian were working on the scrape and SQL tasks, Rob had already started on Flask code in python (army.py) using SQLalchemy and Session to create an API and web pages for

running the queries against the SQLite database.



The Flask Routes all map the specific queries to web pages in the templates folder.



Then each python query is pasted into each respective route.



In HTML, we now create a nav bar to add each query page link.



Each link is to the API route, except the Report, which uses a link to the Github Project repository Docs folder to show this report as HTML.



1. REPORT: Team will compile the final report (this document).

REPORTED CHALLENGES:

Rob:

1. Initially the pdf table data was copied into a text file and imported as a CSV. Rather than being copied directly into its column format, this process placed all data in series into a single column, including table titles. (ACME\_Customer\_Sites.csv)
2. The data had to be transformed into a structured CSV files using python. (Base\_Data.ipynb)
3. These were then used to initiate the coordinate data scrape from Google. (Base\_Locations.ipynb)
4. The final task was to take Lillian’s SQL code and place it into the Flask python file so the web pages could run queries. (army.py)
5. Various challenges ensued but in time were rectified. These included an issue with flask html pages not displaying image files placed in the repo, and some effort to ensure all API data output was organized into an HTML table on each web page. Thes issues were resolved using HTML Doc type declarations, external image links, and the Github repo doc folder.

Mike:

1. We had problems identifying the proper coding for the html tag we were attempting to single out. It ended up being: ("div", attrs={"class" : "Z0LcW"}):.Several factors led to inconsistent scraping, including an initial lack of space padding (%20) in the URL, and variable HTML formats in the Google search responses (depending on keywords).
2. When attempting to scrape from Google we kept getting “503” errors meaning we were blacklisted by Google because attempting to scrape Google from a dynamically run search violates the terms of use policy. We then had to wait a full day to attempt again.
3. Whilst we did not acquire all of the required location data, we did export a set of consolidated data for loading into the database. (base\_locations.csv)

Lillian:

1. The first task of loading was performed by a CSV file import into MySQL. (army.sql)
2. One of the challenges was to create the queries for the SQLite database. (army\_sqlite.ipynb)
3. Also I accidentally recreated the SQLite database every time I ran it so I kept adding the same data to the table. Oops.