# Data Sources

|  |
| --- |
|  |



<https://data.ca.gov/dataset/ground-water-water-quality-results>

<https://www.waterboards.ca.gov/water_issues/programs/water_quality_goals/docs/wq_goals_text.pdf>

<https://www.cityofwoodland.org/DocumentCenter/View/1067/Water-Quality-Report-PDF>



<https://www.census.gov/data/tables.html>

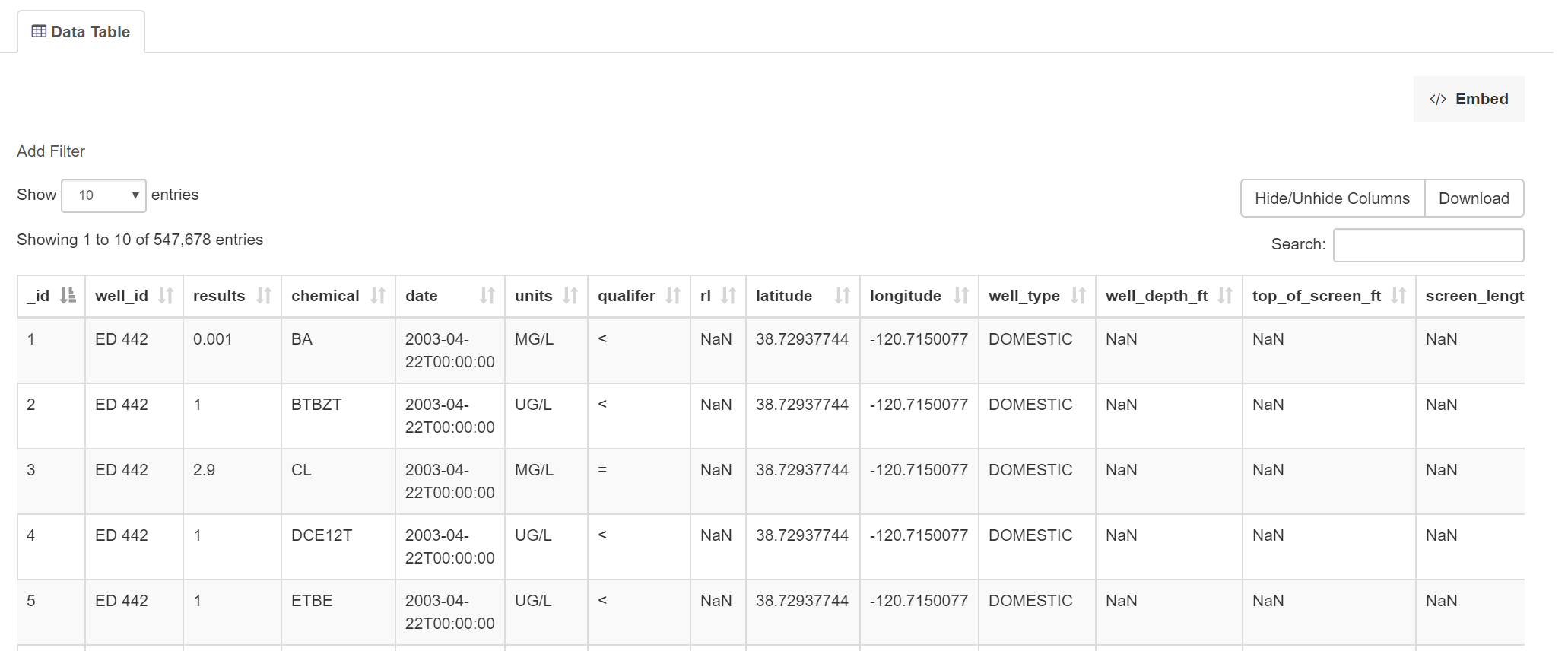


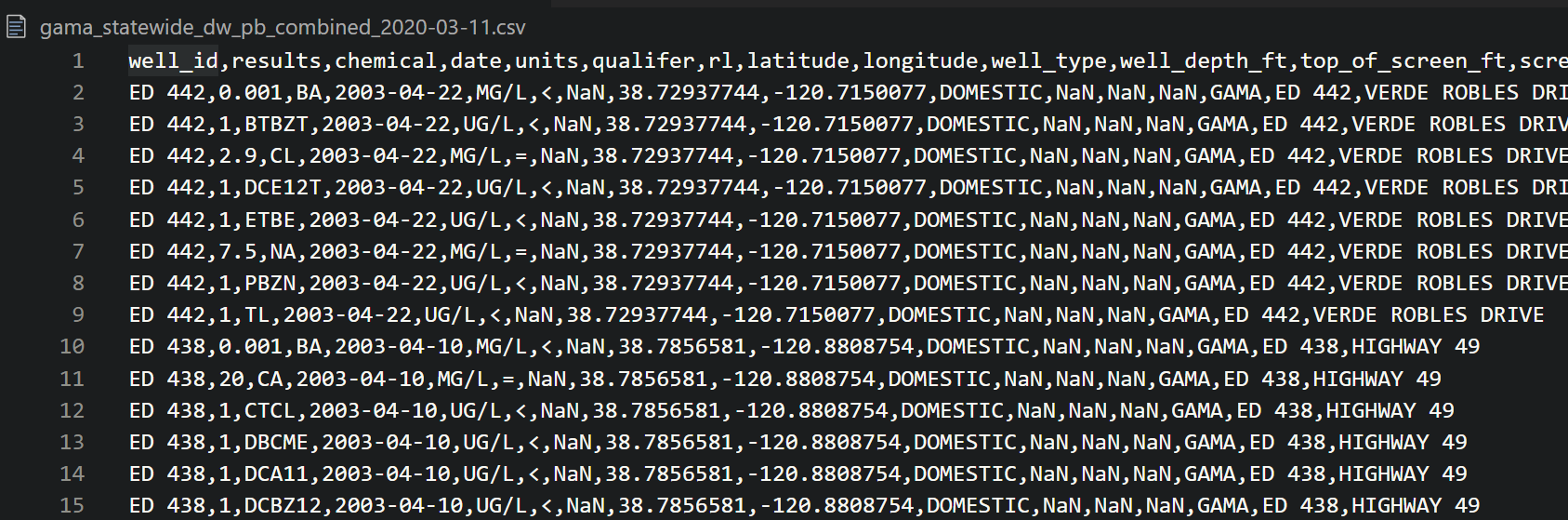
<https://droughtmonitor.unl.edu/Data/DataDownload/ComprehensiveStatistics.aspx>

# Database Creation

Initial download was from the California Open Data Portal, <https://data.ca.gov/>. This data was chosen because it contains the statewide groundwater quality data for all chemicals from the GAMA Domestic Well (DW) and Priority Basin (PB) programs. It is the most comprehensive statewide groundwater quality information publicly available.

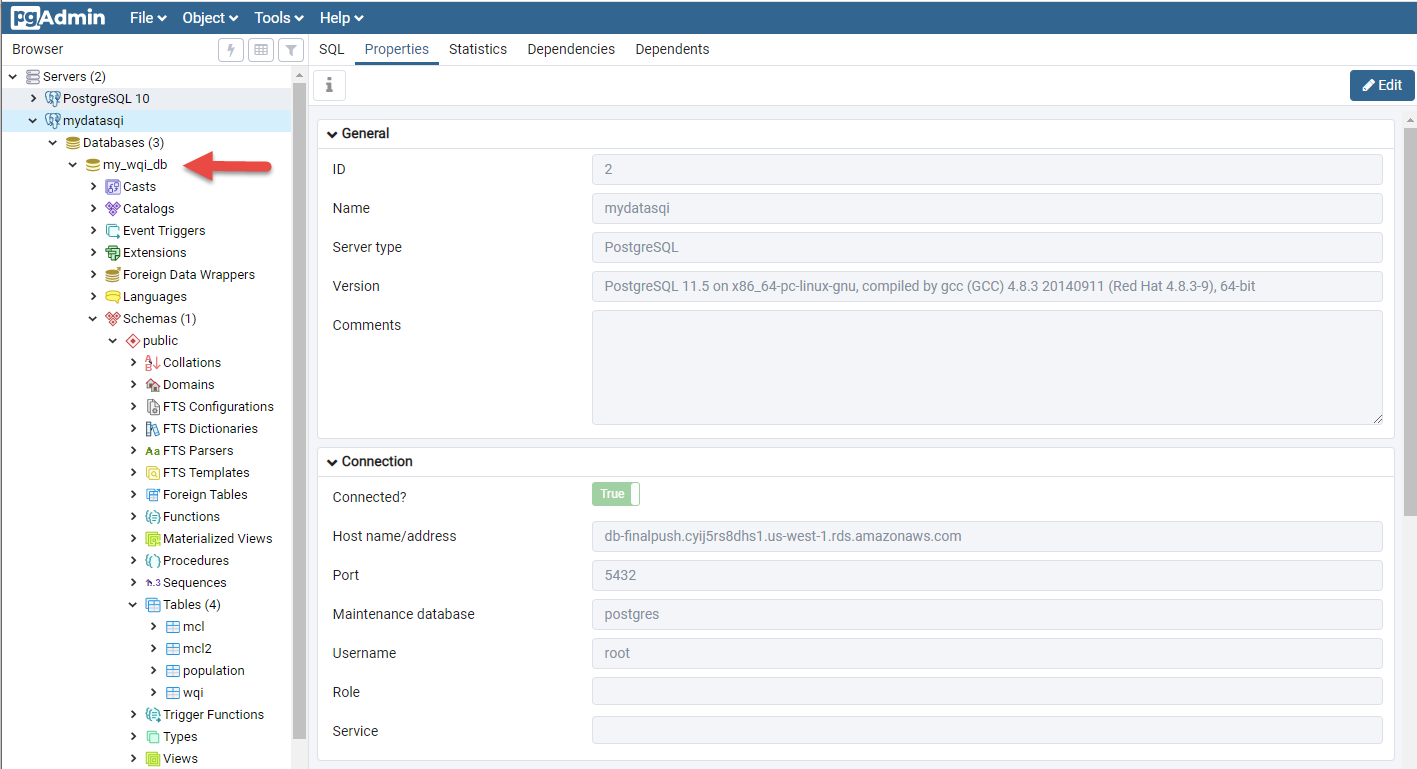
<https://data.ca.gov/dataset/81d1347c-891d-4f09-abc5-6eeb521b55d2/resource/5cef96fd-6f7b-4a83-ac83-aea62d437552/download/gama_statewide_dw_pb_combined_2020-03-11.csv>





After an extensive ETL process (data cleanup) of the water quality file was read to be added to a database.

We then created a PostgreSQL database instance on Amazon Web Services. We created the database schema in SQL and imported the CSV files into database. We also added the census information into it’s own table and added two additional tables for chemical contaminant levels, and one for drought conditions.

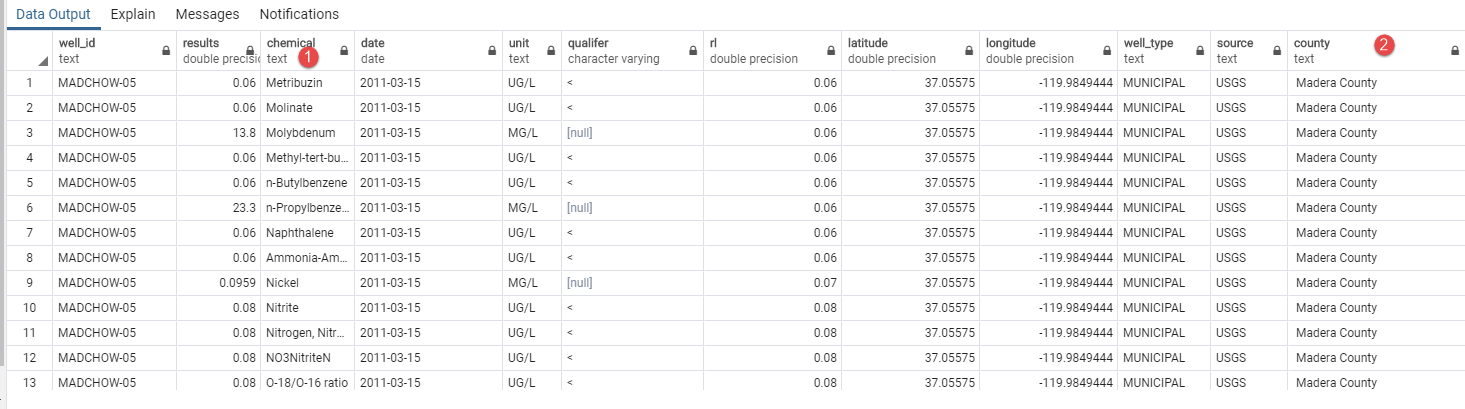


# Data Cleanup

Groundwater Quality Master File

We removed needless columns and NaN data.

We also inserted two new columns:



Chemical Column

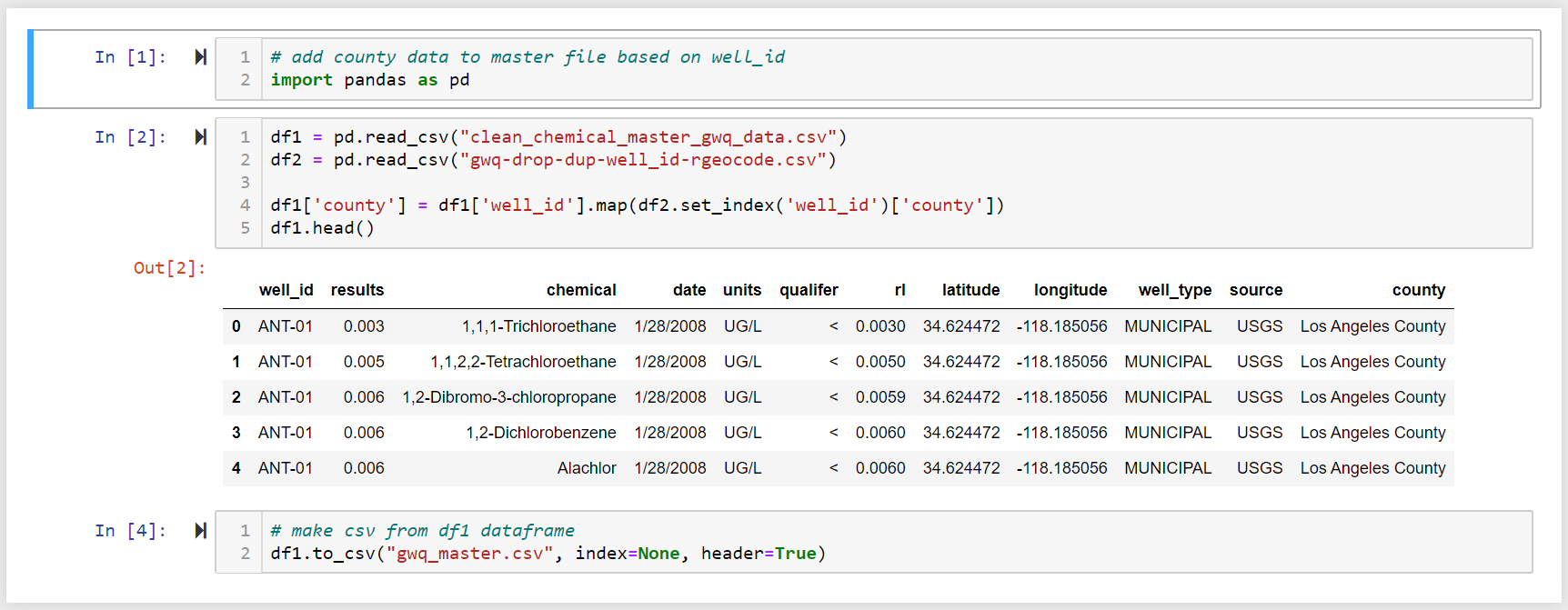
Updated to full Chemical name. We went to <https://pubchem.ncbi.nlm.nih.gov/> to find the chemical names then opened the .csv file in jupyter notebook and ran this command for each chemical.

for chem in data.chemical:   
print(chem)  
if chem == "MTBE":  
chem = "Methyl-tert-butyl ether"  
else if

County Column

Originally all we had was latitude and longitude, but we needed a county column to join with other tables in database. This was done with reverse geocoding, getting geographical information in a list associated with a latitude longitude location. We found a free geocoder, geopy.geocoders with an import called Nomanatim that we ran in Jupyter Notebook to accomplish this task.

It was found that the number of geocode requests had to be limited to successfully process the data and 547,678 rows in the original file was too many. Since there were multiple rows of the same well, which is in the same location and has the same lat/long data, we were able to drop duplicate rows with the same “well\_id”. This left me with a much easier to manage file with 4,085 rows of data, which was still too many, and the geocoder would time out. Through trial and error, we were able to find that about 400 rows would not time out the geocoder, so we split the 4,085 row file file into 10 separate files. With Pandas, WEwas then able to reverse geocode each file, produce 10 files with the county data, and concatenate those files into a new 4,085 row file that included county data as a new column. We then imported that new file and the master file into Pandas as separate dataframes and added all the county data to the master file.



Other cleanup tasks:

* Updating column names
* Changing data types to allow the merge to occur
* Ensuring equal amount of rows to each column.

# Tableau Connection

We connected our PostgreSQL database to Tableau. Here’s a screenshot of the data tab and tables. At this point, all project members had access to the AWS for our data reporting purposes.

