SJSU SAN JOSÉ STATE UNIVERSITY

Department of Applied Data Science

DATA 225

Database Systems for Analytics

Instructor: Simon Shim

GROUP PROJECT REPORT

U.S. Climatological Data Analysis in Google Cloud Platform

Group 4

Group Members

Anshul Yadav

Sakshi Jain

Shreya Sree Matta

Veena Ramesh Beknal

INDEX

Abstract

The phrase ‘Under the weather’ has an interesting origin story. Meaning unwell or feeling worse than usual, the term under the weather is a nautical term from the days of old sailing ships. Any sailor who was feeling ill would be sent below deck to protect them from the weather. Being below deck, the sailor would literally be under the weather. Earth’s temperature has risen by an average of 0.14° Fahrenheit (0.08° Celsius) per decade since 1880, or about 2° F in total and the rate of warming since 1981 is more than twice as fast: 0.32° F (0.18° C) per decade [source]. As weather conditions affect all of us in multiple ways and as we’re beginning to experience the effects of climate change, it would be fair to say that we’re all ‘under the weather’ in some way! The role of weather data is critical in numerous applications, including predicting weather patterns, studying climate change, and understanding local weather conditions. This research project focuses on the U.S. Local Climatological Data obtained at a station level from the National Oceanic and Atmospheric Administration (NOAA). Through comprehensive analysis of historical data combined with near-real-time data, we would like to answer the questions listed below:

1. What is the trend of weather patterns over time across the US?
2. Are there zones/clusters of regions where climate change is more prominent than others?
3. What areas in California can utilities companies such as PG&E focus their efforts on to address the surge in demand ahead of time in a data-driven manner? (to reduce stress on the power grid in peak power consumption periods like winter and snowfall)

Solution Requirements

We propose using the clustering algorithm k-means to identify groups of regions with similar climate change magnitudes. By leveraging these techniques, we aim to contribute to the advancement of weather prediction and climatology. The analysis aims to quantify the impact of climate change, represent this in an intuitive visual manner, and help in predicting future weather conditions in the state of California. Our project shows the possibilities of utilizing the US climatological data to get useful insights through cutting-edge analytical techniques, statistical modeling, and visualizations.

Limitations

- The dataset’s scope might be constrained, missing important long-term patterns. The reliability of analyses is a function of the data quality.

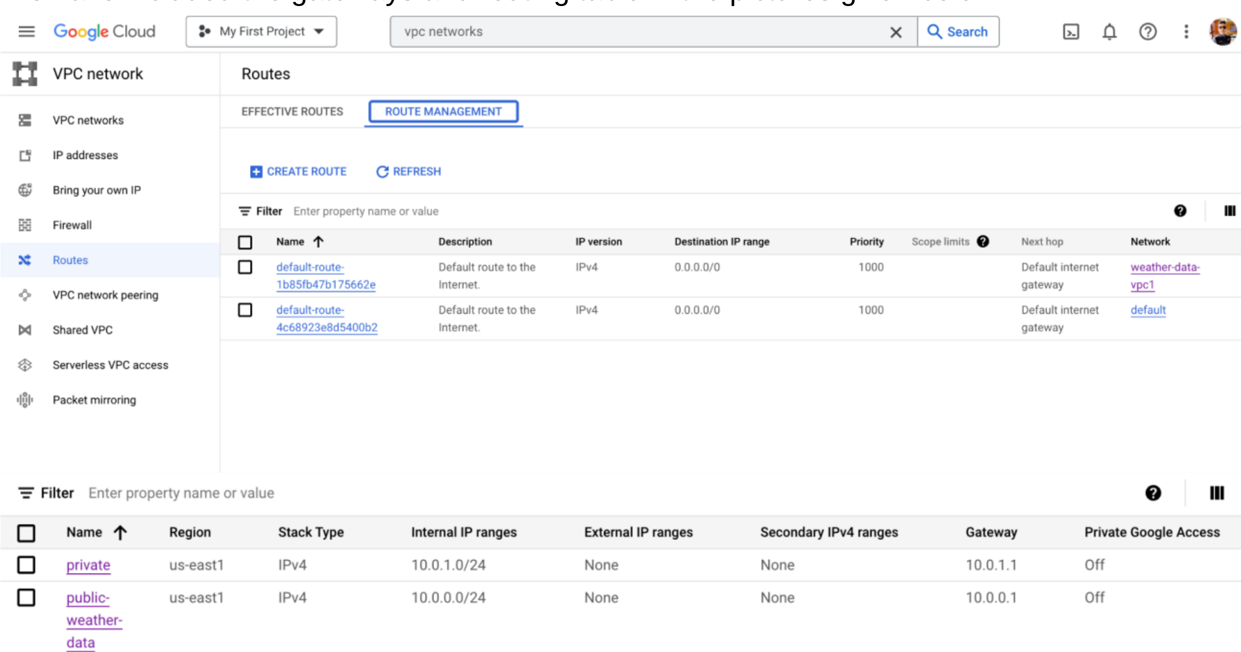
- Non-climatic elements that can affect weather patterns, such as urbanization, deforestation, and industrial activity, might not be taken into account in the analysis.
- The granularity of the analysis may vary depending on how frequently data is collected (daily, monthly, or annually, for example).
- This analysis does not consider the influence of other factors like deforestation and industrial activity that may also have an impact on weather patterns.

Data overview and VPC connection

To efficiently handle and retain our archived meteorological data, we made use of the Google Cloud Platform (GCP). Because a BigQuery instance can handle structured data well, a characteristic that many climates data sets share, we established a privately owned VPC network to house our entire data workflow, ensuring efficient and safe data handling. A private virtual private cloud (VPC) is needed because it provides a safe and private network environment necessary for handling meteorological data and removes the possibility of data manipulation by unauthorized users. This protects our database from any security threats and external exposure and also enhances the network speed.

In our VPC network, we added two subnets. The deployment of these subnets gives us comprehensive control over the traffic flow and administration within the VPC and contributes to the network's efficient organization.

We have included the gateways and routing table in the pictures given below.



Name	Description	IP version	Destination IP range	Priority	Scope limits	Next hop	Network
default-route-1b85fb47b175662e	Default route to the Internet.	IPv4	0.0.0.0/0	1000		Default internet gateway	weather-data-vpc1
default-route-4c68923e8d5400b2	Default route to the Internet.	IPv4	0.0.0.0/0	1000		Default internet gateway	default

Name	Region	Stack Type	Internal IP ranges	External IP ranges	Secondary IPv4 ranges	Gateway	Private Google Access
private	us-east1	IPv4	10.0.1.0/24	None	None	10.0.1.1	Off
public-weather-data	us-east1	IPv4	10.0.0.0/24	None	None	10.0.0.1	Off

Figure 1 Gateways and Routing table

As for the actual data that we're using for the project, there are 3 main datasets that are being used, the primary source of which is NOAA (National Oceanic and Atmospheric Administration):

- Historical weather data from NOAA's FTP server (link) - this contains data for weather stations from 180 countries from which we're downloading the CSV files but filtering for USA only our workflow
    - We have considered data from 2020 to 2023 and this data altogether amounts to ~1.5 GB
- Real-time weather data from NOAA's web services API (link)
- Weather station mapping for California stations (FIPS:06) from NOAA's web services API

From there, we're working with 3 weather metrics **Temperature** (TMIN, TMAX, TAVG), **Precipitation** (PRCP), and **Snowfall** (SNOW) captured for each day for each weather station

The below code snippet is used to extract the weather station details for all California stations:

```
token = {'token': '<token>'}

# Define API parameters for stations
stations_params = {
    'datasetid': 'GHCND',  # GHCND dataset
    'limit': 1000,          # Maximum number of results per page
}


# Define the URL for GHCND stations with FIPS:06 for California
url = f"https://www.ncei.noaa.gov/cdo-web/api/v2/stations?locationid=FIPS:06"

# Initialize an empty list to store all station data
all_station_data_california = []

# Make the initial API request to get the total number of stations
initial_response = requests.get(url, headers=token, params=stations_params)
initial_json = initial_response.json()

# Check if 'metadata' key is present in the response
if 'metadata' in initial_json and 'resultset' in initial_json['metadata']:
    total_stations = initial_json['metadata']['resultset']['count']

    # Determine the number of requests needed based on the total number of stations
    num_requests = -(-total_stations // stations_params['limit'])

    # Make multiple requests to get all stations
    for offset in range(1, num_requests + 1):
        stations_params['offset'] = offset
        response = requests.get(url, headers=token, params=stations_params)
        json_data = response.json()

        # Check if 'results' key is present in the response
        if 'results' in json_data:
            ghcnd_stations = json_data['results']
            all_station_data_california.extend([(station['id'], station['name']) for station in ghcnd_stations])

# Save all station data to a CSV file
csv_file_path = 'all_ghcnd_stations_california.csv'
with open(csv_file_path, 'w', newline='', encoding='utf-8') as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(['Station ID', 'Station Name'])  # Header row
    csv_writer.writerows(all_station_data_california)

print(f"All GHCND stations data for California saved to {csv_file_path}")
```

The real-time data API call is shown in the below screenshot:

```python
token = {'token': '<token>'}

# Date definitions
today_date = datetime.now()
day7_before_today = (today_date - timedelta(7)).strftime('%Y-%m-%d')

stations_params = {
    'limit': 1000, # Maximum number of results per page
    'offset': 1, # Starting point of the results
    #'startdate': day_before_yday,
    #'enddate': day_before_yday,
}

# Initialize an empty list to store all station data
data_california = []

# URL with FIPS:06 (California) for a date 7 days prior
url = f"https://www.ncei.noaa.gov/cdo-web/api/v2/data?datasetid=GHCND&locationid=FIPS:06&startdate=
{day7_before_today}&enddate={day7_before_today}"

station_data_response = requests.get(url, headers=token, params=stations_params)
# Capturing the initial JSON from the API response
initial_json = station_data_response.json()

# Check if 'metadata' key is present in the response
if 'metadata' in initial_json and 'resultset' in initial_json['metadata']:
    total_stations = initial_json['metadata']['resultset']['count']

    # Determine the number of requests needed based on the total number of stations
    # Multiple requests might be required since limit is 1000 per request
    num_requests = -(-total_stations // stations_params['limit'])

    # Make multiple requests to get all data points
    for offset in range(1, num_requests + 1):
        stations_params['offset'] = offset
        response = requests.get(url, headers=token, params=stations_params)
        json_data = response.json()
        #california_stations_data = json.loads(response.text)['results']

        # Check if 'results' key is present in the response
        if 'results' in json_data:
            california_stations_data = json_data['results']
            data_california.extend([(dt['date'], dt['datatype'],
                                     dt['station'], dt['value']) for dt in california_stations_data])

# Storing in a dataframe
data_califronia_df = pd.DataFrame(data_california)
column_names_california_df = ['date', 'metric', 'station','values']
data_califronia_df.columns = column_names_california_df

# Save all station data to a CSV file
csv_file_path = 'california_real_time.csv'
with open(csv_file_path, 'w', newline='', encoding='utf-8') as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(['Date', 'datatype', 'Station ID', 'Value'])  # Header row
    csv_writer.writerows(data_california)

print(f"All real-time data (7 days from date of run) for California saved to {csv_file_path}")
```

Please note that we are getting real-time data with a lag of 7 days to increase the coverage of data.

ETL Pipelines

We built an ETL pipeline on the Google Cloud Platform; this process necessitates thoughtful preparation and the appropriate GCP tool selection. We have extracted the data from the ftp server ([Index of /pub/data/ghcn/daily/by_year]) to BigQuery to store and analyze data. Data extraction from various sources is the first step in the ETL pipeline. After that, information is transformed using tools like (Dataflow or Data prep). After that, BigQuery is loaded with the converted data. The top focus is given to security, which is managed by stringent data encryption techniques and IAM. Cost-effectiveness and resource efficiency are the main goals of the entire procedure. This methodology satisfies the requirements of contemporary data administration and analysis by offering a reliable and adaptable way to manage a variety of data analytics jobs in GCP.

Scheduling using Apache Airflow

In our project, the ETL (Extract, Transform, Load) pipeline is coordinated using Apache Airflow. Utilizing the Directed Acyclic Graph (DAG) script architecture provided by Airflow, we can orchestrate the flow of data across specific tasks defined by us. By defining tasks and dependencies in DAG, it ensures that tasks are run in the correct order and at the right time. These Python-written DAG scripts ensure efficient execution for both real-time and archived data sources.
We have created 2 DAGs - one for historical data and one for real-time data as shown in the representation below.

Historical data workflow:

- For historical weather data, we're first extracting yearly data for the last 4 years from the NOAA CDO FTP site (link)
- These files contain weather data at a day level for weather stations from 180 countries, hence we use Python to filter for only US weather data and filter for only temperature, snow and precipitation metrics
- These files are then stored in Cloud storage buckets
- Through Airflow (implemented in GCP as Composer), we're able to schedule a DAG to pick up these files from the storage bucket, perform basic transformation like deduplication and some null value treatment and then write this to a pre-defined schema in BigQuery
- BigQuery is GCP's data warehousing tool which can be used to store and query massive amounts of data within seconds, we use standard SQL as the querying language of choice
- Queries written for specific analyses are then visualized in Looker, GCP's visualization tool

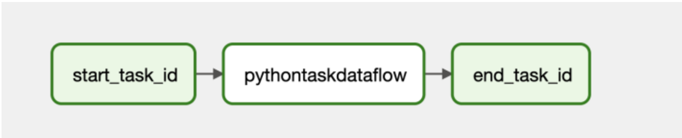


Figure 2 Historical DAG

Real-time data workflow:

- For real-time data, we're first extracting data lagged by 7 days from the current date (for better data coverage) NOAA CDO web services API (link) only for California stations by filtering for FIPS:06 criteria

- These raw data API extracts contain multiple weather parameters all appended horizontally, these need to be filtered for the key metrics of interest and pivoted to be unique at the weather station and day level - these will be done through Python
- These transformed dataframes are then stored in Cloud Storage buckets for backup purposes
- Through Airflow (implemented in GCP as Composer), we're able to schedule a DAG to pick up these files from the storage bucket, perform basic transformations like deduplication and some null value treatment, and then write this to a pre-defined schema in BigQuery
- BigQuery is GCP's data warehousing tool which can be used to store and query massive amounts of data within seconds, we use standard SQL as the querying language of choice
- Queries written for specific analyses with near real-time are then visualized in Looker, GCP's visualization tool

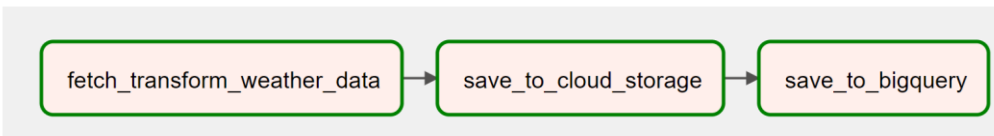


Figure 3 Real-time DAG

Figure 4 DAG Statistics

Data Warehousing

As explained above, the last step of Apache Airflow is to write data into BigQuery for the design and implementation of our project's Data Warehouse (DW). Our choice of BigQuery stems from its ability to work with both historical and real-time data sources at scale.
Our solution uses BigQuery's robust query engine to perform analytics directly using SQL queries. Creating tables that are part of our studies and analysis in the data warehouse allows us to have a single location for all of our analytical and reporting requirements. This strategy makes sure that we have a single source of all information, along with our data handling being consistent and easier access for different stakeholders.

Database Architecture

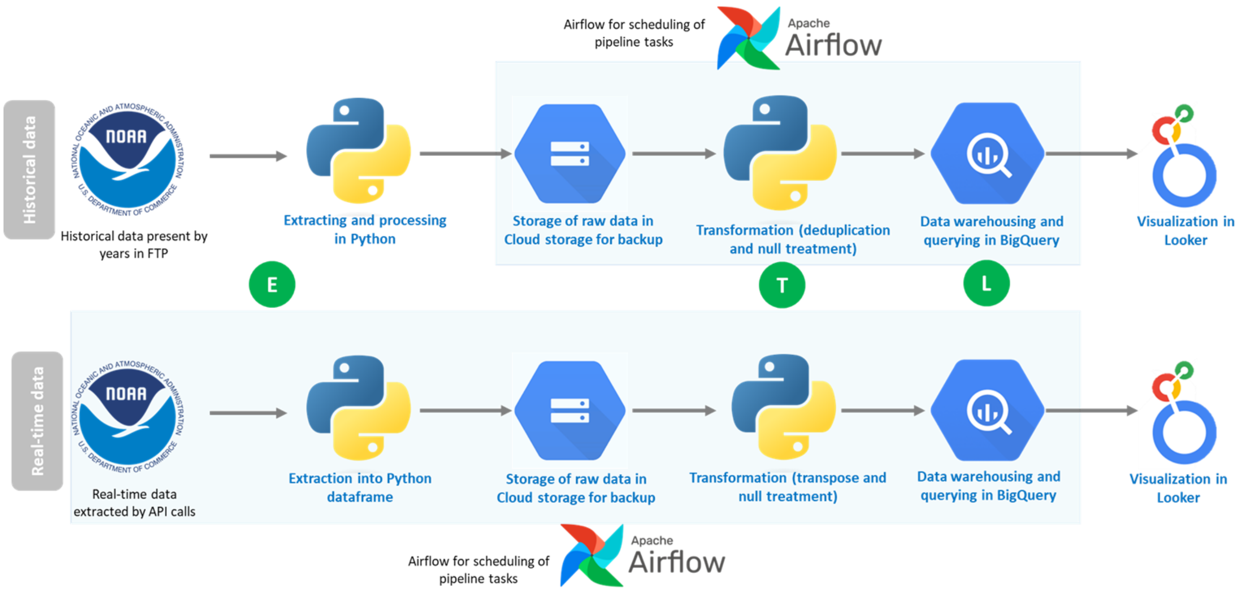


Figure 5 Architecture Diagram

Historical Data:
We have extracted historical data for the 2020–2023 four-year period. Meteorological data about station ID, minimum temperature, average temperature, maximum temperature, snowfall amount, precipitation amount, and temperature variance are all stored.

Mapping Stations:
Information about California's weather stations is presented in a table. It includes the station name, ID, state, and county.

Real-Time Data:
Our data source provides us with real-time data that we may access. In line with the historical data, we have preserved weather-related metrics such as station_id, TMIN (minimum temperature), TAVG (average temperature), TMAX (maximum temperature), SNOW (snowfall amount), and PRCP (precipitation amount).

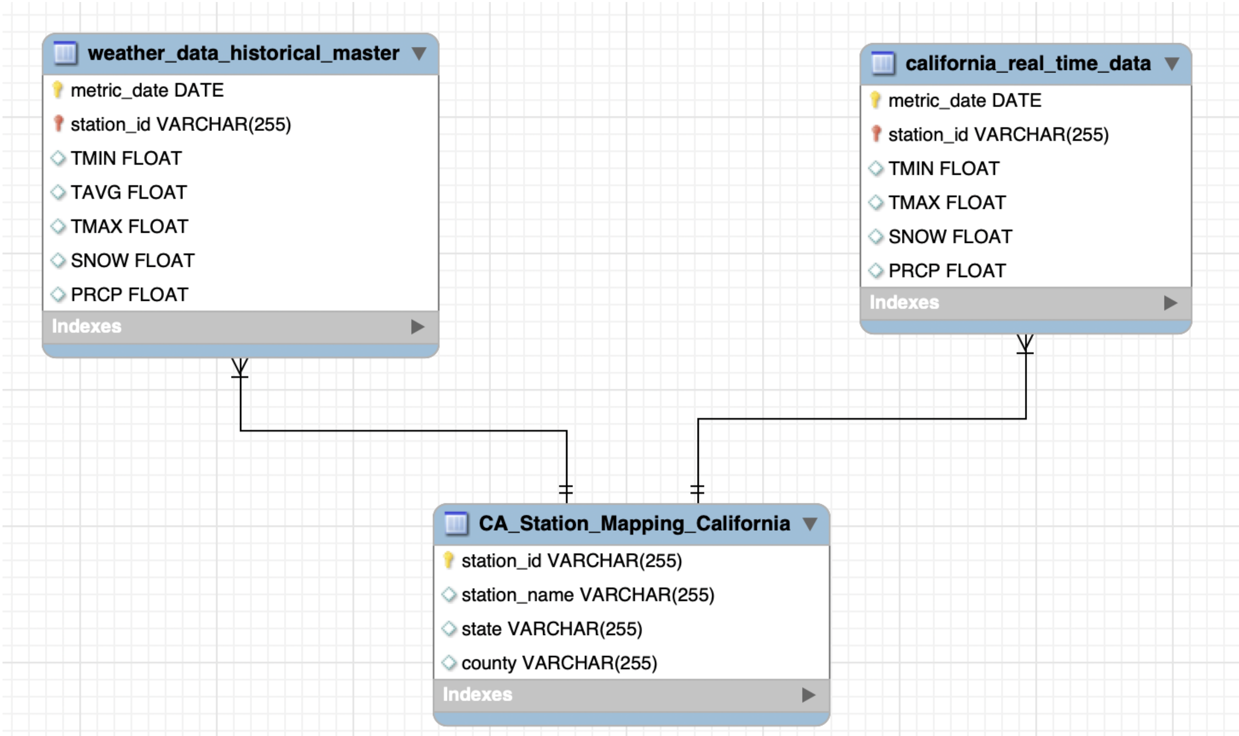


Figure 6 DB Schema

```
CREATE TABLE CA_Station_Mapping_California (
    station_id VARCHAR(255),
    station_name VARCHAR(255),
    state VARCHAR(255),
    county VARCHAR(255),
    PRIMARY KEY (station_id)
);

CREATE TABLE weather_data_historical_master (
    metric DATE,
    station_id VARCHAR(255),
    TMIN FLOAT,
    TAVG FLOAT,
    TMAX FLOAT,
    SNOW FLOAT,
    PRCP FLOAT,
    PRIMARY KEY (metric, station_id),
    FOREIGN KEY (station_id) REFERENCES CA_Station_Mapping_California(station_id)
);

CREATE TABLE california_real_time_data (
    metric DATE,
    station_id VARCHAR(255),
    TMIN FLOAT,
```

```
    TMAX FLOAT,
    SNOW FLOAT,
    PRCP FLOAT,
    PRIMARY KEY (metric, station_id),
    FOREIGN KEY (station_id) REFERENCES CA_Station_Mapping_California(station_id)
);
```

Functional Analysis

1. Overall temperature over time: calculates daily average temperature by converting numeric date format and grouping data by date.

2. Total precipitation across each station in California by year: determines annual total precipitation at each California station by joining weather and station data and grouping by year and station name.

3. Average temperature of non-California stations in December 2022 with no snow: Find the average temperature for non-California stations in December 2022 where no snow was reported, using left join and date filters.

4. Total precipitation and average temperature for selected stations in 2022: computes average temperature and total precipitation for 2022 at stations identified by state code, focusing on data within the year.

5. Yearly snowfall trends per Country: Calculates annual total snowfall for US by extracting year from date and summing up non-null snowfall values.

6. Heavy snow and rainy days analysis: determines the count of heavy rainy and snowy days by setting thresholds for precipitation and snowfall, grouped by date.

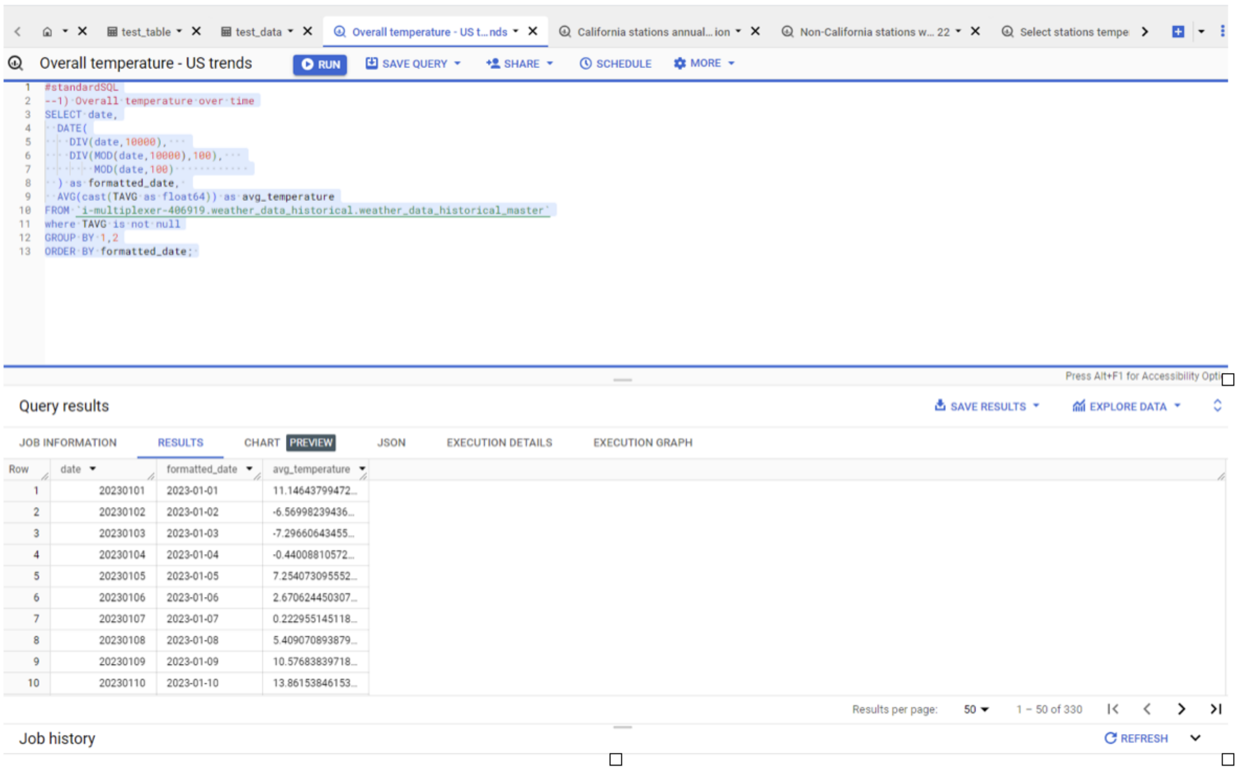
7. Rainfall recorded by each weather station by weekday: aggregates total precipitation for each weather station by day of the week, formatted from the numeric date.

Code snippets / Queries for ETL workflow

1) Overall temperature over time: calculates daily average temperature by converting numeric date format and grouping data by date

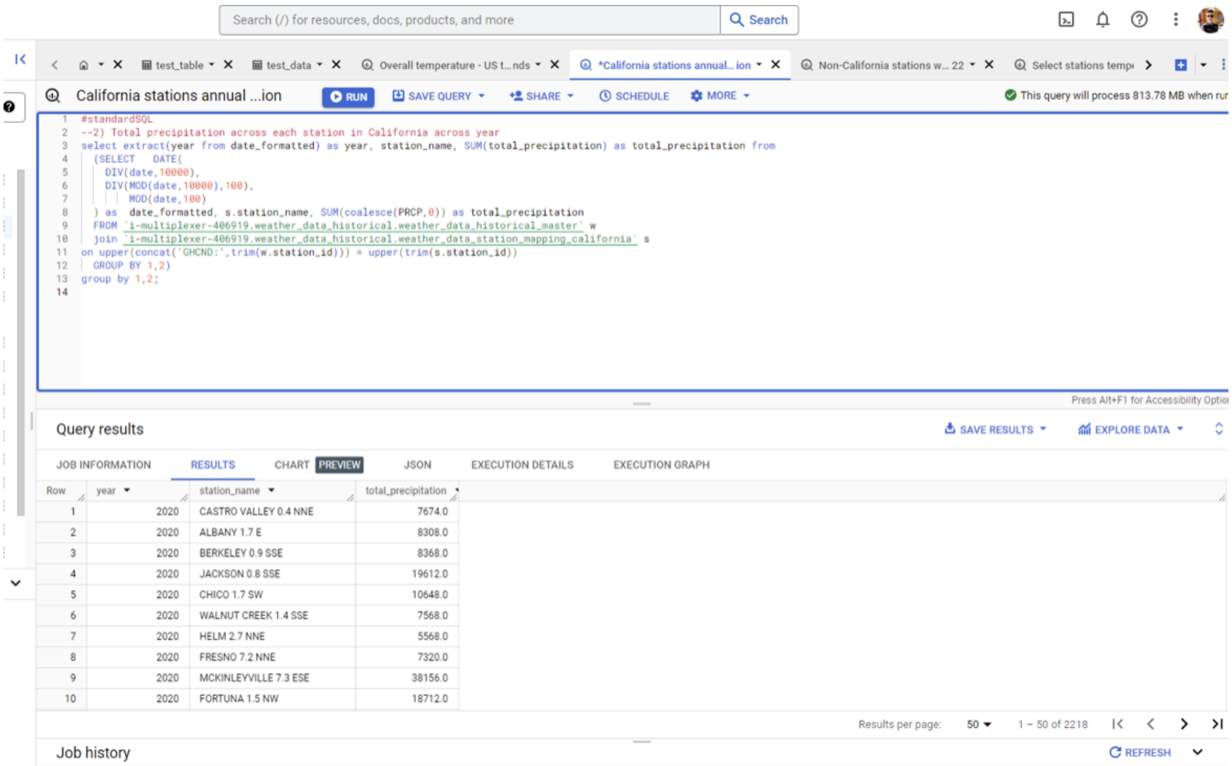
```
SELECT date,
  DATE(
    DIV(date,10000),
    DIV(MOD(date,10000),100),
        MOD(date,100)
  ) as formatted_date,
```

```
  AVG(cast(TAVG as float64)) as avg_temperature
FROM `i-multiplexer-406919.weather_data_historical.weather_data_historical_master`
where TAVG is not null
GROUP BY 1,2
ORDER BY formatted_date;
```



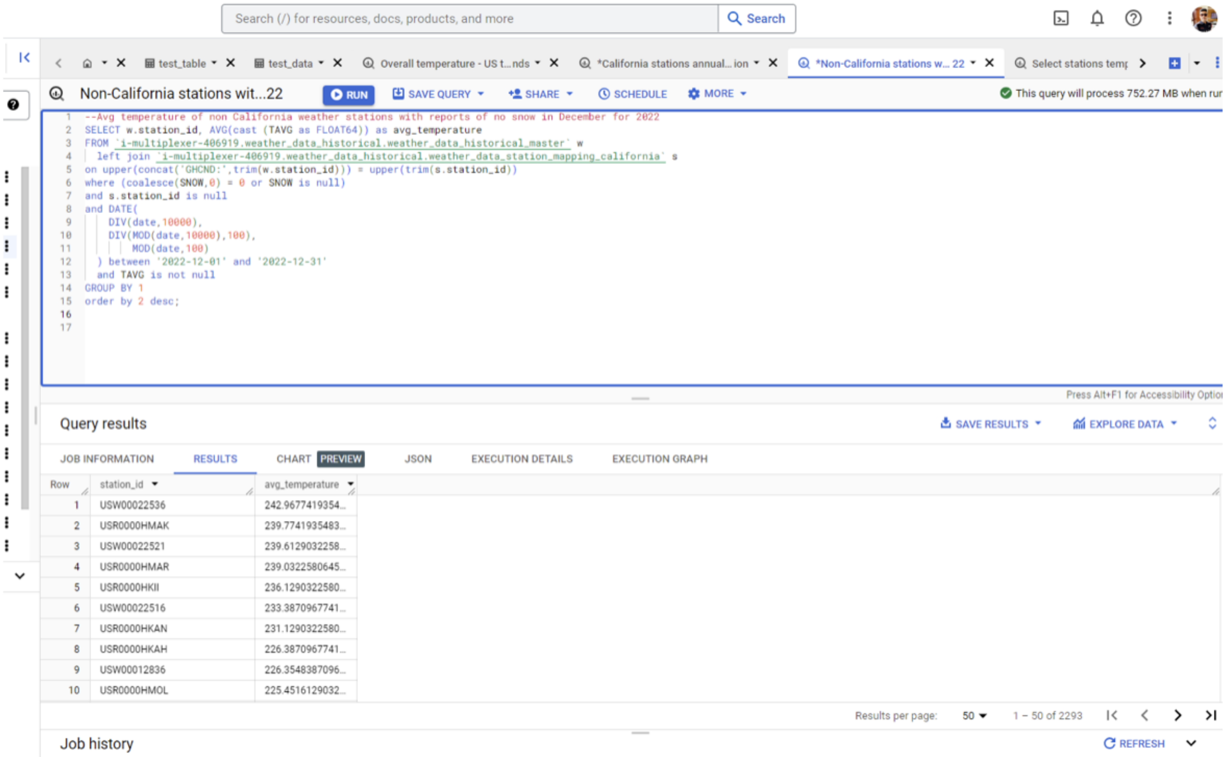
2) Total precipitation across each station in California by year: determines annual total precipitation at each California station by joining weather and station data and grouping by year and station name

```
        (SELECT   DATE(
          DIV(date,10000),
          DIV(MOD(date,10000),100),
             MOD(date,100)
        ) as  date_formatted, s.station_name, SUM(coalesce(PRCP,0)) as total_precipitation
        FROM `i-multiplexer-406919.weather_data_historical.weather_data_historical_master`
      w
        join `i-multiplexer-
      406919.weather_data_historical.weather_data_station_mapping_california` s
      on upper(concat('GHCND:',trim(w.station_id))) = upper(trim(s.station_id))
```



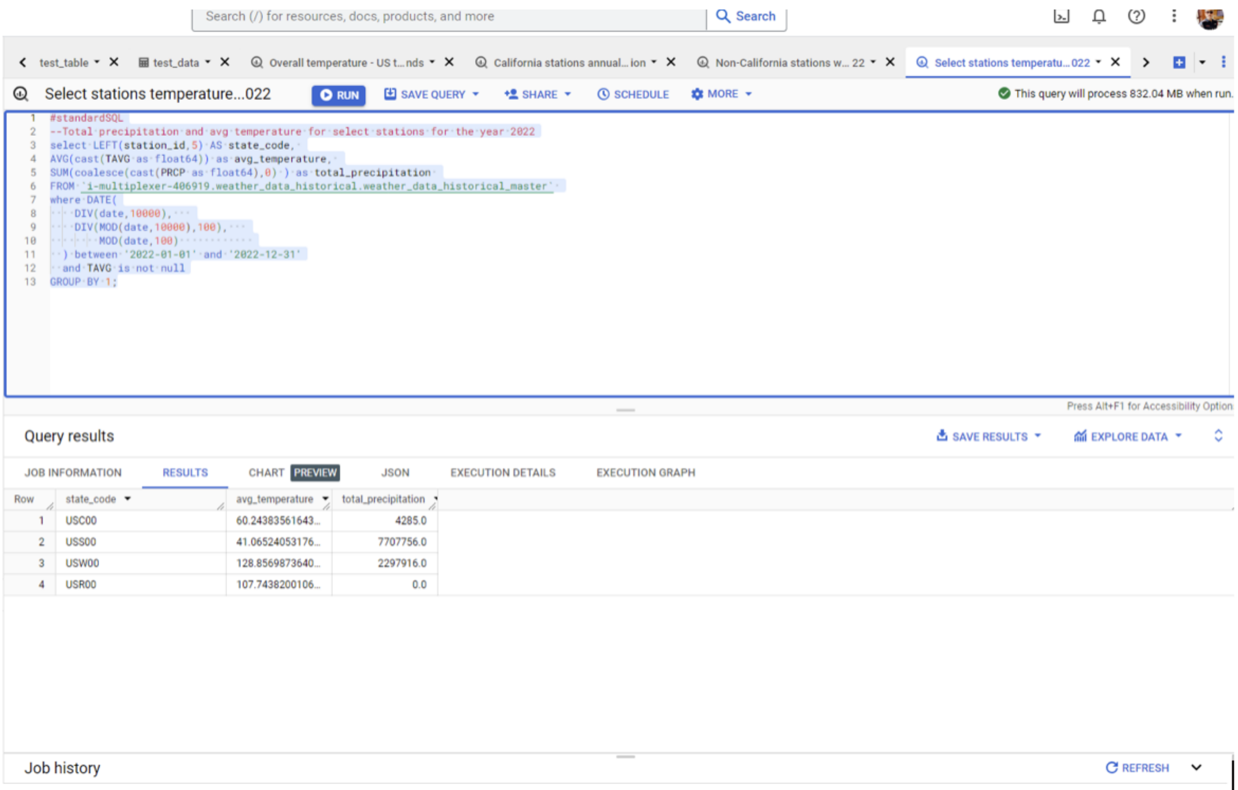
3)Average temperature of Non-California stations in December 2022 with No Snow: Finds average temperature for non-California stations in December 2022 where no snow was reported, using left join and date filters

```
SELECT w.station_id, AVG(cast (TAVG as FLOAT64)) as avg_temperature
FROM `i-multiplexer-406919.weather_data_historical.weather_data_historical_master` w
  left join `i-multiplexer-
406919.weather_data_historical.weather_data_station_mapping_california` s
on upper(concat('GHCND:',trim(w.station_id))) = upper(trim(s.station_id))
where (coalesce(SNOW,0) = 0 or SNOW is null)
and s.station_id is null
and DATE(
    DIV(date,10000),
    DIV(MOD(date,10000),100),
        MOD(date,100)
  ) between '2022-12-01' and '2022-12-31'
  and TAVG is not null
GROUP BY 1
order by 2 desc;
```



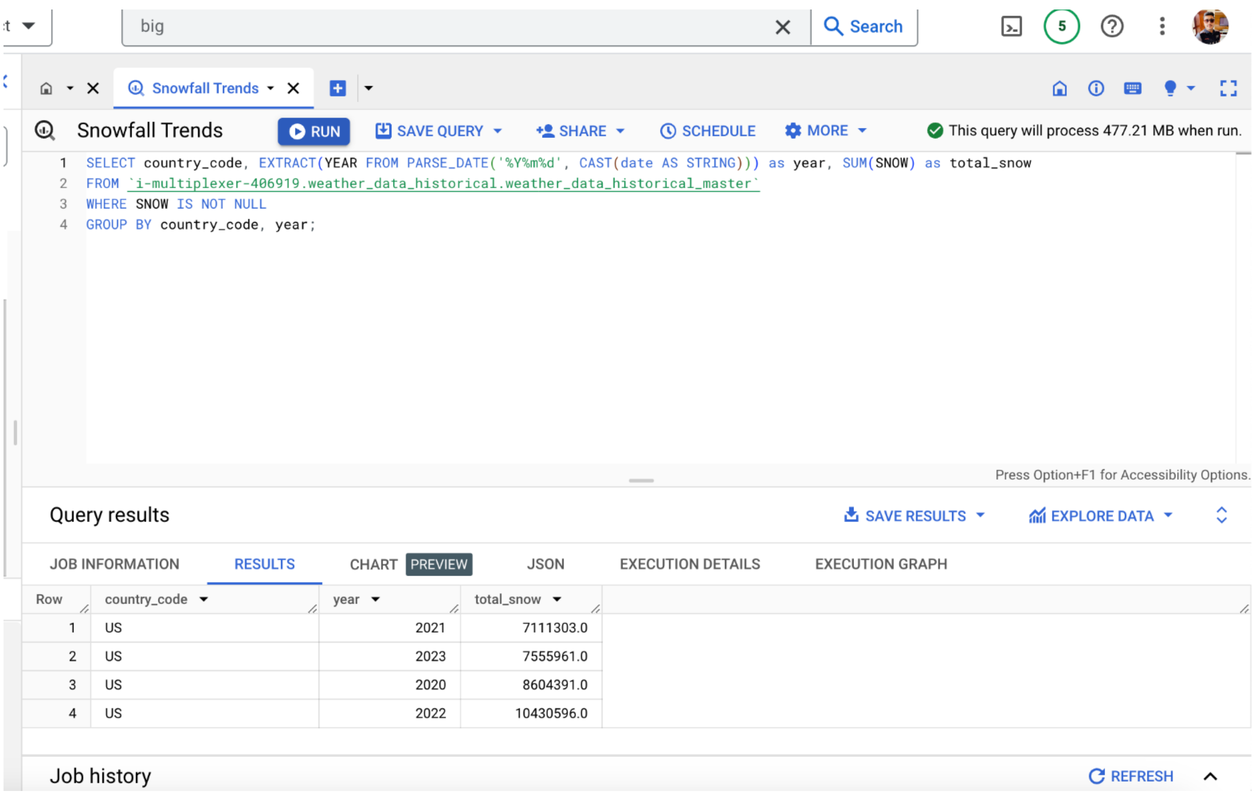
4) Total Precipitation and Average temperature for selected stations in 2022: computes average temperature and total precipitation for 2022 at stations identified by state code, focusing on data within the year

```
select LEFT(station_id,5) AS state_code,
AVG(cast(TAVG as float64)) as avg_temperature,
SUM(coalesce(cast(PRCP as float64),0) ) as total_precipitation
FROM `i-multiplexer-406919.weather_data_historical.weather_data_historical_master`
where DATE(
    DIV(date,10000),
    DIV(MOD(date,10000),100),
        MOD(date,100)
  ) between '2022-01-01' and '2022-12-31'
  and TAVG is not null
GROUP BY 1;
```



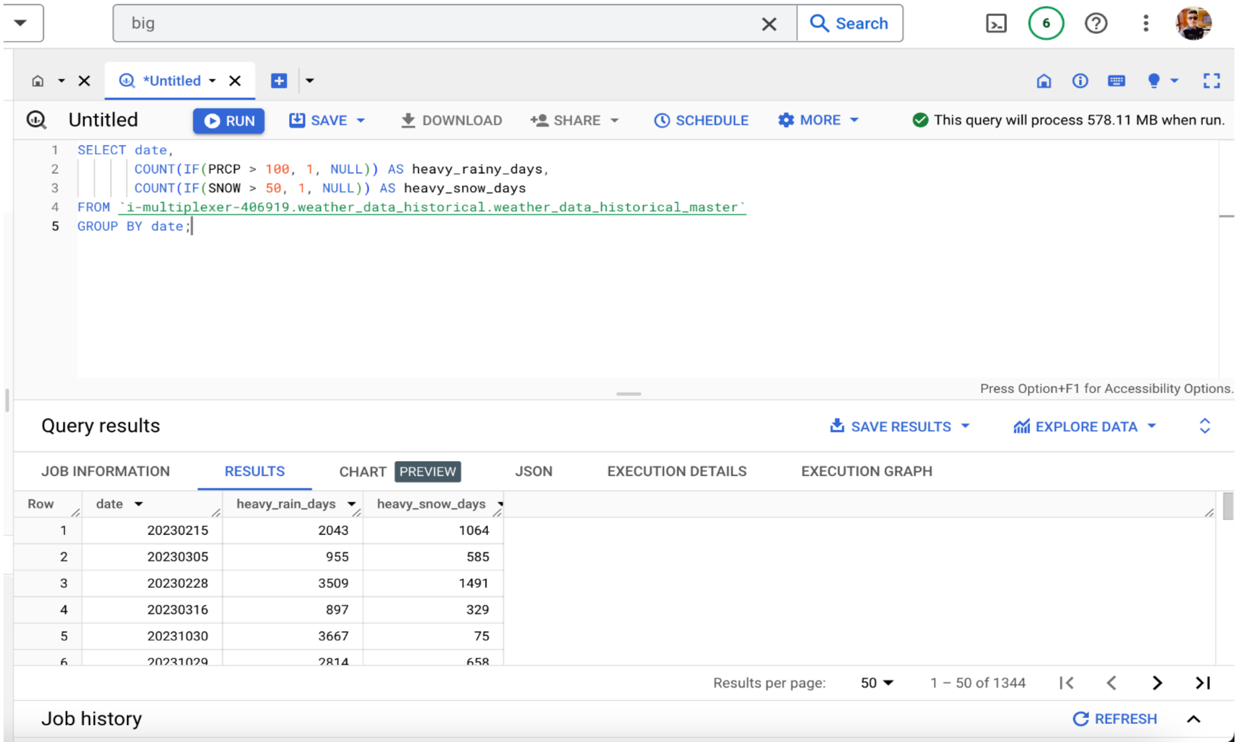
5)Yearly Snowfall Trends: Calculates annual total snowfall for US by extracting year from date and summing up non-null snowfall values

```
SELECT  EXTRACT(YEAR FROM PARSE_DATE('%Y%m%d', CAST(date AS
STRING))) as year, SUM(SNOW) as total_snow
FROM `i-multiplexer-406919.weather_data.weather_data_historical_master`
WHERE  SNOW IS NOT NULL AND country_code = 'US'
GROUP BY  year
```



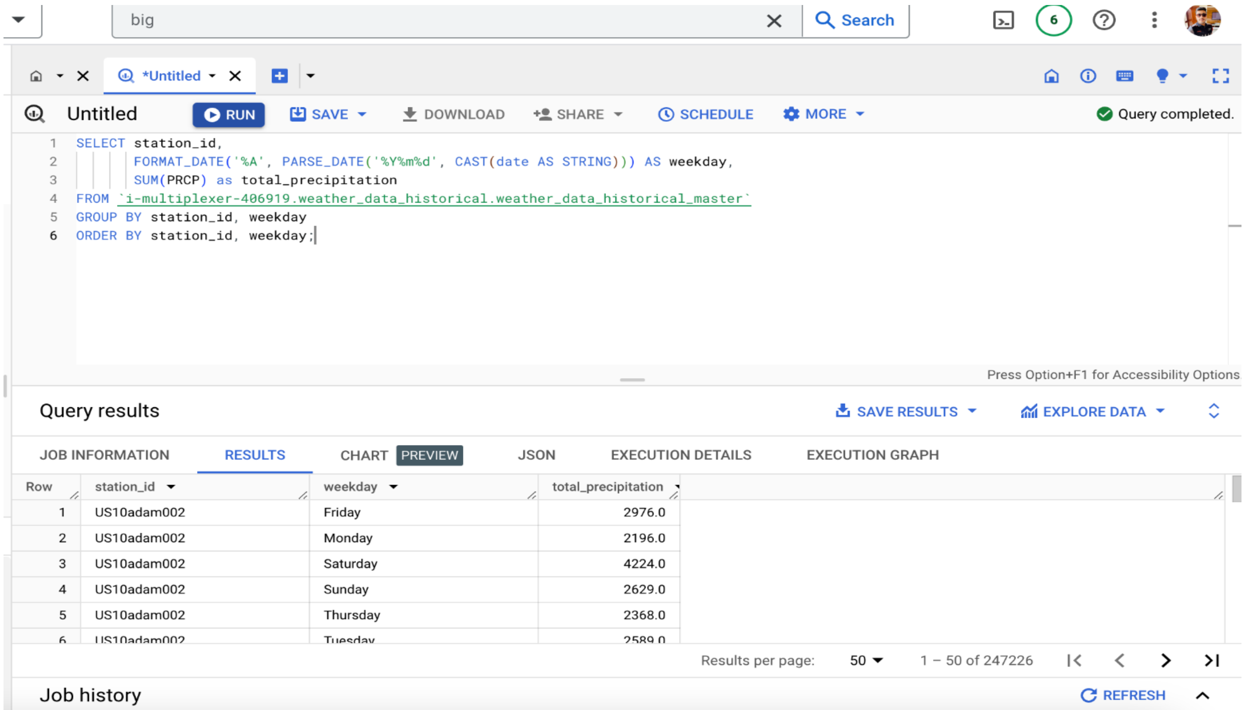
6)Heavy snow and rainy days analysis: determines the count of heavy rainy and snowy days by setting thresholds for precipitation and snowfall, grouped by date

```
SELECT date,
    COUNT(IF(PRCP > 100, 1, NULL)) AS heavy_rainy_days,
    COUNT(IF(SNOW > 50, 1, NULL)) AS heavy_snow_days
FROM `i-multiplexer-406919.weather_data_historical.weather_data_historical_master`
GROUP BY date;
```



7) Rainfall Recorded by each weather station by weekday: aggregates total precipitation for each weather station by day of the week, formatted from the numeric date.

```
SELECT station_id,
    FORMAT_DATE('%A', PARSE_DATE('%Y%m%d', CAST(date AS STRING))) AS weekday,
    SUM(PRCP) as total_precipitation
FROM `i-multiplexer-406919.weather_data_historical.weather_data_historical_master`
GROUP BY station_id, weekday
ORDER BY station_id, weekday;
```



Statistics and Machine Learning

Using the massive amount of weather data at our disposal, we were interested in 3 key data science use cases:

- Is there a difference between temperatures over the years
- Is temperature for a given year normally distributed
- Are there groups of stations in California that show similar extreme weather patterns (heavy rainfall and snow) that could help utility companies focus their efforts to ensure limited power disruptions during extreme weather events

Use case 1: Difference between temperature over the years

To do this, we first considered the median of TAVG (average temperature) across all stations for each day of the year for 2021 and 2022. We hypothesized that there would be a significant difference between these 2 years' temperatures. To test this, we ran a t-test with our null hypothesis being "There is no significant difference between the temperatures of 2021 and 2022", hence our alternative hypothesis becomes "There is a significant difference between the temperatures of 2021 and 2022". The code snippet for our t-test is shown below:

```python
# Converting date string to date format for 2022
weather_data_us_2022['clean_date'] = pd.to_datetime(weather_data_us_2022['date'], format='%Y%m%d')

# Converting date string to date format for 2021
weather_data_us_2021['clean_date'] = pd.to_datetime(weather_data_us_2021['date'], format='%Y%m%d')
```

```python
# Taking median temperature for all US for the year 2022
df_weather_temp_grouped_2022 = weather_data_us_2022[['clean_date','TAVG']].groupby('clean_date')[['TAVG']].median()
```

```python
# Taking median temperature for all US station for the year 2021
df_weather_temp_grouped_2021 = weather_data_us_2021[['clean_date','TAVG']].groupby('clean_date')[['TAVG']].median()
```

```python
_, p_value = ttest_ind(df_weather_temp_grouped_2022['TAVG'].head(282)
                       , df_weather_temp_grouped_2022['TAVG'])
print(f"p-value for independent t-test between 2022 and 2021 median temperature by day is: {p_value}")
```

```
p-value for independent t-test between 2022 and 2021 median temperature by day is: 0.0036704045290449705
```

The outcome was that the p-value for the independent t-test was lower than the critical value i.e. 0.05, indicating that we have sufficient evidence to reject the null hypothesis and conclude that **there is a significant difference between 2021 and 2022 temperatures.**

Use case 2: Distribution of temperature

We wanted to understand if median daily temperature across 2022 follows a normal distribution or not. To prove or disprove this mathematically, we used a Shapiro-Wilk test. In the Shapiro-Wilk test, null hypothesis = Sample is from the normal distributions. (p-value>0.05). Hence, if we obtain a p-value > 0.05, we accept the null hypothesis and can conclude that the data is normally distributed.

First, we visually inspected the histogram of the daily median temperature and the plot looked like the below chart:

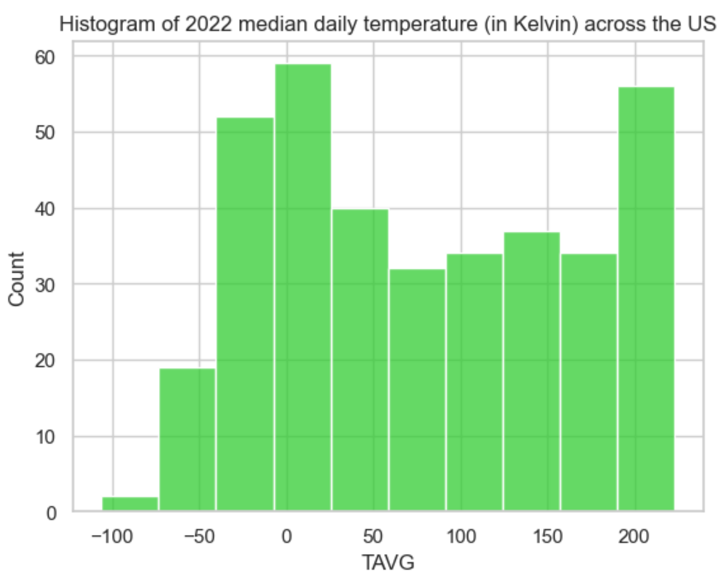


Figure 7 Histogram for 2022 median daily temperature

Through visual inspection of the data, we see a somewhat bimodal distribution of data and this doesn't look like a bell curve. We can confirm our suspicion by running the Shapiro-Wilk test.

```
shapiro_weather_data = shapiro(df_weather_temp_grouped_2022['TAVG'])
shapiro_weather_data
```

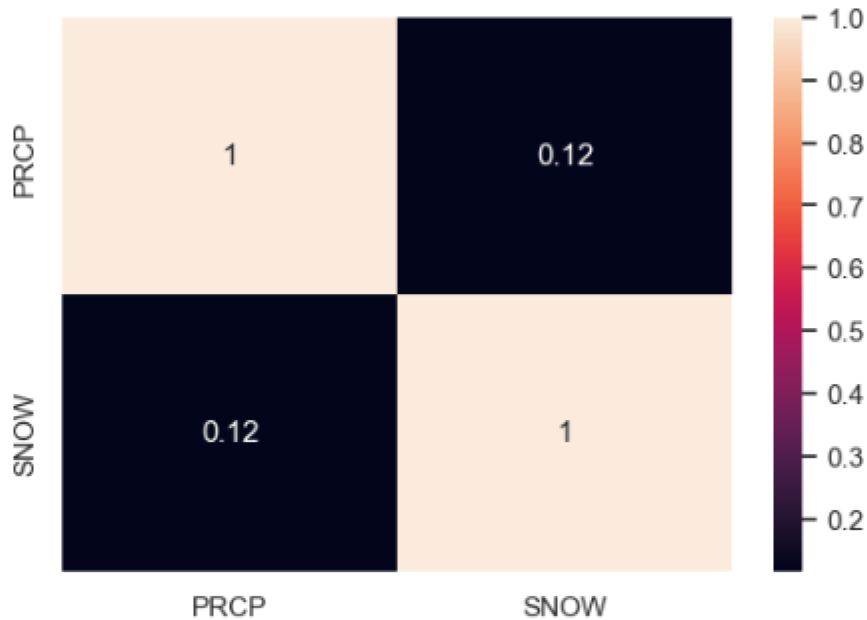
```
]: ShapiroResult(statistic=0.9386388063430786, pvalue=3.915789914543666e-11)
```

The p-value <0.05 implies that we reject the null hypothesis and can conclude that the data is NOT normally distributed.

Use case 3: Extreme weather data clustering for California stations
California is one of the most geographically diverse states in the US. Despite being a coastal state, there are several dry-desert-like areas but also snowy mountains and picturesque beaches. Hence, for such a unique region, utility companies like PG&E (California's largest utilities provider) need to understand weather patterns to be prepared for seamless power supply even in extreme conditions.
To do this, we propose a novel approach of using k-means clustering to identify clusters or groups of stations that have similar precipitation and snowfall conditions.
For this, we considered 756 weather stations in the state of California for which we had good coverage of precipitation (total precipitation for the year 2022) and snowfall (average yearly snowfall) data.
With this, we ran a kmeans clustering algorithm to understand the groups of stations that showed similar weather patterns. The steps are explained below.

- After aggregating 2022 data for 756 stations (sum of precipitation and average of snowfall), we first checked if these are correlated

```python
# Heatmap to illustrate relationships between all independent variables

# Setting size of figure with width 10 and height 8
plt.figure(figsize=(6,4))

# Calculating the correlation matrix on the numeric columns
corr_weather_2022 = df_weather_cali_grouped[['PRCP', 'SNOW']].corr()

# Plotting the heatmap
sns.heatmap(corr_weather_2022, annot=True)

# Displaying the heatmap
plt.show()

# No correlation observed between the variables
```



- Since the correlation was quite low, we can proceed with the next step of scaling the data - for this, we used StandardScaler from sci-kit learn

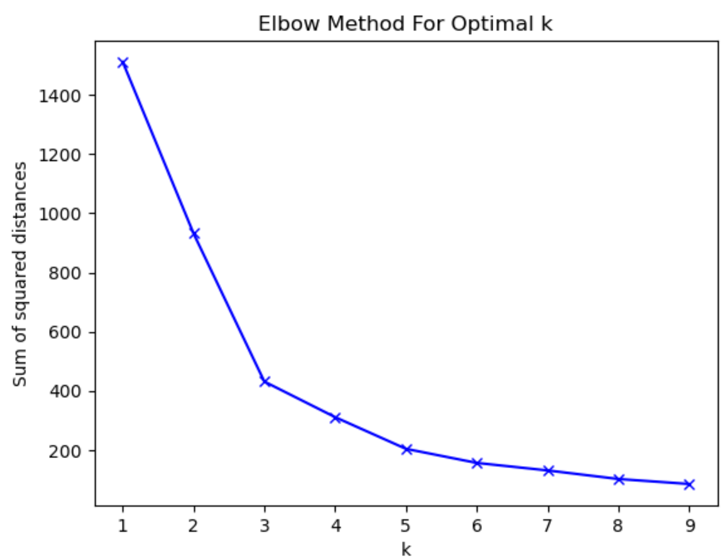
```python
# Select the relevant columns for clustering
weather_data_clustering = df_weather_cali_grouped[['PRCP', 'SNOW']]

# Correlation and remove correlated features
# No correlation

# Standardize the data
scaler = StandardScaler()
weather_data_scaled = scaler.fit_transform(weather_data_clustering)
```

- K-means is an unsupervised machine learning algorithm wherein we can cluster datapoints that have similar characteristics
- Since we don't know how many clusters we will get at the end, we first need to iterate through different values of k and find the optimal value - this is done through a method

called the elbow curve, wherein we plot the different values of k against the sum of squared distances (SSD) between each point from the respective cluster centroids

- Optimal value of k is chosen based on the “elbow” point or the point beyond which the rate of decrease of SSD is not as gradual

```
# Elbow curve to get optimal value of k
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Calculate sum of squared distances
ssd = []
K_value_range = range(1,10)
for k in K_value_range:
    km = KMeans(n_clusters=k)
    km = km.fit(weather_data_scaled)
    ssd.append(km.inertia_)

# Plot sum of squared distances / elbow curve
plt.plot(K_value_range, ssd, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum of squared distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



- From our iteration, we can say that the optimal value is 5 although this is subjective and open to interpretation
- With k=5, we can run kmeans as shown below to get the data labeled with 5 clusters

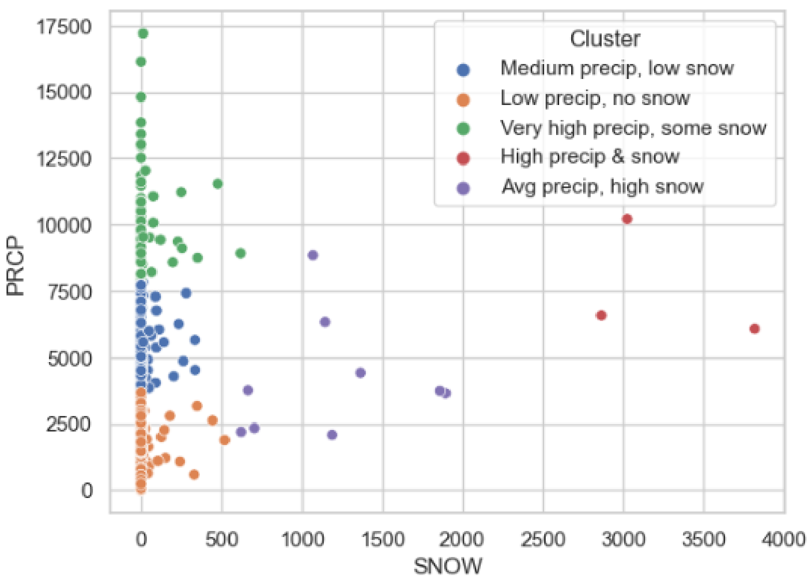
```
# Perform KMeans clustering
kmeans = KMeans(n_clusters=5)  # Choosing the number of clusters based on elbow plot
kmeans.fit(weather_data_scaled)

# Add the cluster labels to the original dataframe
df_weather_cali_grouped['cluster'] = kmeans.labels_

# Save the dataframe with cluster labels to a new CSV file
df_weather_cali_grouped.to_csv('weather_data_california_clustered.csv', index=False)
```

- Based on analyzing the clusters, we can observe that there each station can show 1 among 5 characteristics
    a. Low precipitation with no snow - dry habitable regions
    b. High precipitation and snow - extreme regions
    c. Very high precipitation with some snow - wet regions
    d. Average precipitation and high snow - snowy regions
    e. Medium precipitation with low snow - habitable regions

```
# Defining labels for clusters
kmeans_cluster_number = [
    df_weather_cali_grouped['cluster'] == 0,
    df_weather_cali_grouped['cluster'] == 1,
    df_weather_cali_grouped['cluster'] == 2,
    df_weather_cali_grouped['cluster'] == 3,
    df_weather_cali_grouped['cluster'] == 4
]

kmeans_labels = ['Low precip, no snow', 'High precip & snow', 'Very high precip, some snow',
                 'Avg precip, high snow', 'Medium precip, low snow']


# Creating a new label column based on mapping cluster number to label
df_weather_cali_grouped['cluster_label'] = np.select(kmeans_cluster_number, kmeans_labels)
```

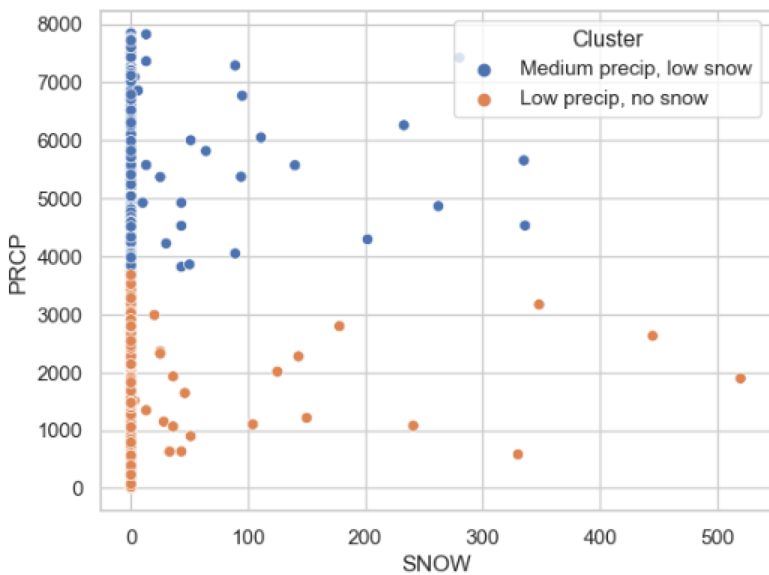
Visualizing the results, we can observe that clusters b, c, and d are outliers - these are the total 73 extreme regions that PG&E must focus on. However, they also skew our results and hence, we can visualize the clusters without these outlier points to get a better view of the data spread.

```python
# Set the style of the visualization
sns.set(style="whitegrid")

# Creating a bubble plot
bubble_plot = sns.scatterplot(data=df_weather_cali_grouped, x="SNOW", y="PRCP",
                              hue="cluster_label", legend="full", palette="deep")

# This is getting skewed by 3 clusters

# Find the handles and labels of the current legend
handles, labels = bubble_plot.get_legend_handles_labels()

plt.legend(title='Cluster')
# Show the plot
plt.show()
```



Visualization of the data without the outlier points:

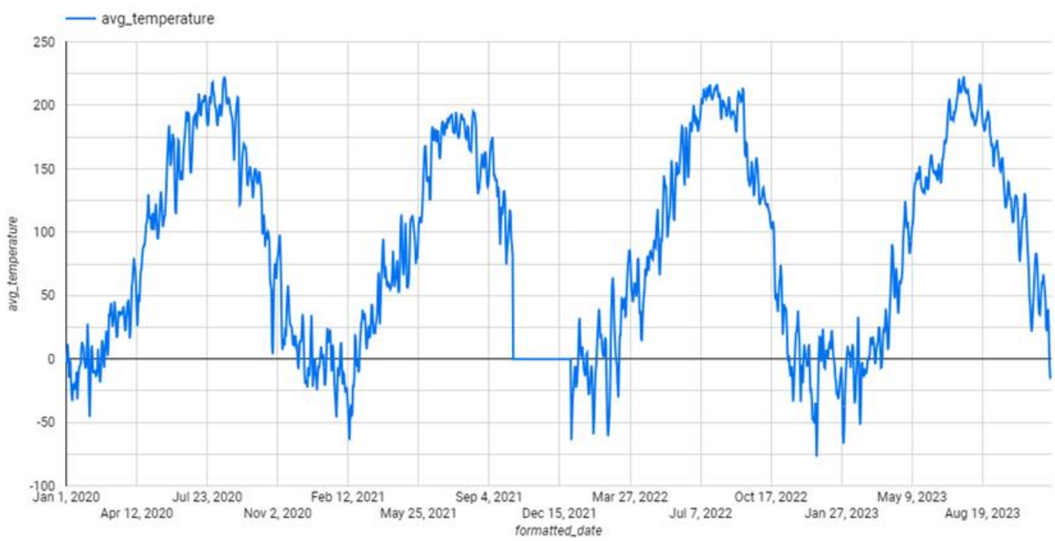
```python
# Set the style of the visualization
sns.set(style="whitegrid")

# Excluding outlier clusters
cluster_df = df_weather_cali_grouped.loc[df_weather_cali_grouped["cluster"] != 1]
cluster_df = cluster_df.loc[cluster_df["cluster"] != 2]
cluster_df = cluster_df.loc[cluster_df["cluster"] != 3]

# Creating a bubble plot
bubble_plot = sns.scatterplot(data=cluster_df, x="SNOW", y="PRCP",
                              hue="cluster_label", legend="full", palette="deep")

# This view is a bit more spread out after removing the clusters with extreme values

# Find the handles and labels of the current legend
handles, labels = bubble_plot.get_legend_handles_labels()

plt.legend(title='Cluster')
# Show the plot
plt.show()
```



Business Intelligence

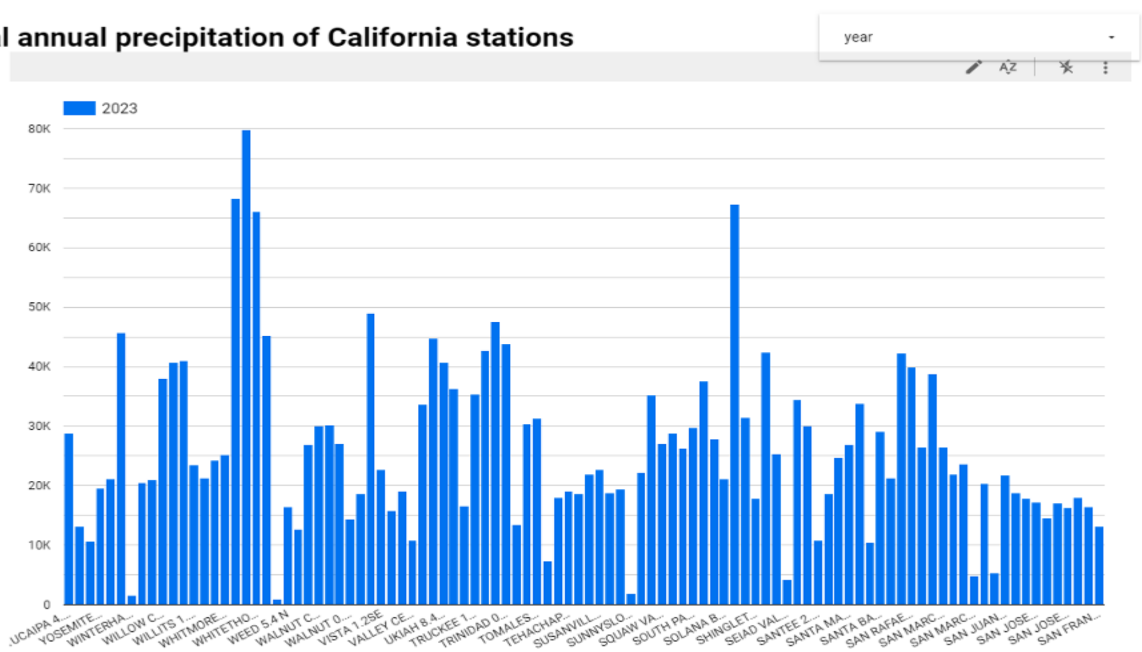
We have successfully completed business intelligence tasks in our project, concentrating on extracting the important observations, following thorough data processing and analysis done in the previous steps. We made the decision to combine our BigQuery searches with Looker from Google Cloud Platform in order to efficiently illustrate these observations. This choice is driven by Looker's strong data visualization features and its smooth BigQuery integration, which guarantee not only that our complex datasets are accessible but also understandable.

Historical average temperature trends across the last 3 years across the USA



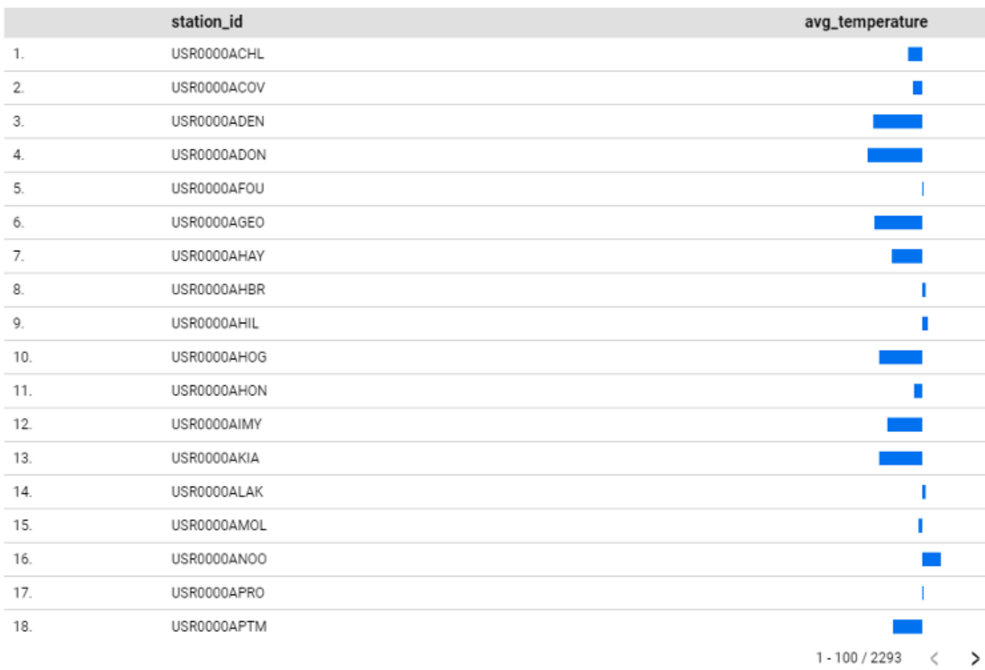
The line graph shows the variations in the average temperature in the US during a period of three years, starting in January 2020 and finishing in late August 2023. The data shows a recurring pattern in which the temperature rises in the middle of the year, which corresponds to the summer months and falls at the start and end of the year, which corresponds to the colder months of winter.

Total annual precipitation of California stations



The Y-axis of the graphic shows the total annual precipitation for each weather station, while the X-axis shows the various weather stations. The bars show how much precipitation was measured at each station; some had considerably larger totals than others, reflecting variations in the distribution of rainfall in California.

Avg temperature of non-California weather stations with reports of no snow in December 2022



	station_id	avg_temperature
1.	USR0000ACHL	
2.	USR0000ACOV	
3.	USR0000ADEN	
4.	USR0000ADON	
5.	USR0000AFOU	
6.	USR0000AGEO	
7.	USR0000AHAY	
8.	USR0000AHBR	
9.	USR0000AHIL	
10.	USR0000AHOG	
11.	USR0000AHON	
12.	USR0000AIMY	
13.	USR0000AKIA	
14.	USR0000ALAK	
15.	USR0000AMOL	
16.	USR0000ANOO	
17.	USR0000APRO	
18.	USR0000APTM	

1 - 100 / 2293

The X-axis shows the matching average temperatures for December 2022 for each weather station listed by station ID on the Y-axis. The average temperature recorded at each station is represented by the length of each bar, which shows a comparison of temperatures across different areas that did not report any snowfall during the month.

Avg temperature and total precipitation for 2022 for select state stations

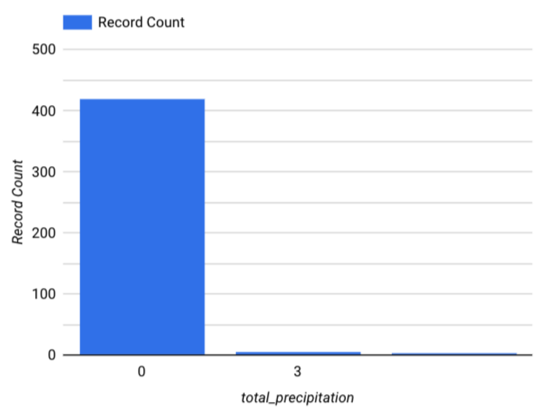
state_code	avg_temperature	total_precipitation
USW00	128.86	2,297,916
USR00	107.74	0
USC00	60.24	4,285
USS00	41.07	7,707,756

The chart has two sets of horizontal bars on the X-axis that show the average temperature and total precipitation for each state station, and a list of state codes on the Y-axis. The first set of bars represents the average temperature in degrees for the year 2022, and the second set depicts the total amount of precipitation in an undefined unit for the same period. The length of the bars varies, representing the variations in temperature and precipitation between the states.

Weekly Precipitation Totals by Station

	station	Record Count ▾
1.	GHCND:USC00040983	1
2.	GHCND:USC00044303	1
3.	GHCND:USC00044374	1
4.	GHCND:USC00045100	1
5.	GHCND:USC00040820	1
6.	GHCND:USC00041614	1
7.	GHCND:USC00041715	1
8.	GHCND:USC00042239	1
9.	GHCND:USC00043551	1
10.	GHCND:USC00043573	1
11.	GHCND:USC00043669	1

1 - 50 / 427 < >



This chart shows a portion of a report labeled "Weekly Precipitation Totals by Station," which lists different weather stations with a record count of '1' for every one of them. Even though each station only displays one record, the bar chart shows three records for precipitation overall, raising the possibility of a mismatch in the data depiction.

Conclusion and Recommendations

- Based on our exploration of both AWS and GCP, we can conclude that GCP is the easier platform to set up and use; while AWS had multiple options and services, there is a learning curve to get started on it

- We experimented with Cloud Dataflow for transformation and loading into BigQuery, it is a useful service for large-scale ETL jobs where there’s a continuous flow of data
- Looker is a good BI tool for basic visualization and reporting and integrates well within GCP but lacks the depth of functionality that Tableau or PowerBI offers
- Based on our analysis of weather data, we noticed that there was a significant difference between overall temperatures between 2021 and 2022, indicating that temperatures are increasing due to global warming
- Weather data follows a cyclical pattern and the trend of weather repeats over time, this helps in the predictability of weather patterns over time
- There are 73 stations with extreme weather conditions (snow and precipitation) that PG&E must prioritize to ensure seamless power supply during extreme weather events

Possible Future Enhancements

- Getting historical data for the last 20-30 years will give us richer insights into historical weather trends and patterns
- Real-time data can directly be extracted from APIs and ingested using GCP’s Cloud Pub/Sub service, the equivalent in AWS is Kinesis
- Advanced ML can be done in GCP through BigQuery ML and CloudML in GCP, equivalent in AWS is AWS Lambda and Sagemaker
- Cloud Dataproc can also be used in GCP for comprehensive data processing of large datasets - the equivalent in AWS is Glue
- A fully managed pipeline can be built in GCP using a combination of Pub/Sub, Cloud Storage, Cloud Dataflow/Dataprep, Dataproc, Apache Beam, and BigQuery although this would also come at a significant cost