Weather Data Analytics Pipeline Using Snowflake, Airflow, dbt, and Tableau

Deeksha Chauhan, Gayatri Patil, Nikhil Swami, Shweta Shinde
Department Of Applied Data Science, San Jose State University
San Jose, California, USA 95192 - 0250

Abstract:

This report discusses how to implement an end-to-end data analytics pipeline using the theme of weather data analytics. Such a pipeline will use Snowflake as the data warehouse, Apache Airflow for orchestration, dbt for data transformation, and Tableau for visualization. The project shows that all these tools can come together to build a robust analytics solution that is scalable, works on historical weather data, and generates insights into various weather metrics and analyses.

Keywords—
ETL- Extract, Transform, Load
ELT- Extract, Load, Transform
DAG - Directed Acyclic Graph
DBT- Data Build Tool

Problem Statement:

In today's data-driven world, analyzing weather patterns and their impact on various sectors is crucial. However, processing and analyzing vast amounts of weather data presents several challenges:

1. Data Integration: Weather data often comes from multiple sources and in various formats, making consolidation difficult.
2. Scalability: As historical weather data accumulates, traditional processing methods struggle to keep up.
3. Data Transformation: Raw weather data requires significant transformation to derive meaningful metrics and insights.
4. Automation: Manual data processing is time-consuming and error-prone, leading to delays in analysis.
5. Visualization: Presenting complex weather data in an easily understandable format is challenging but essential for quick interpretation.
6. Maintainability: Updating and maintaining data pipelines as requirements evolve can be complex.
7. Real-time Insights: There's a growing need for near-real-time weather data processing to support timely decision-making.

To address these challenges, we need an end-to-end data analytics solution that can efficiently ingest, process, transform, and visualize weather data. This project aims to develop such a solution using Snowflake, Apache Airflow, dbt, and Tableau.

I. Introduction

Weather data is crucial for various sectors, including agriculture, energy, transportation, and urban planning. This project aims to create an efficient, scalable data pipeline that can ingest, process, and visualize weather data, providing valuable insights for decision-making.

Objectives:

1. Implement an ETL process to populate raw weather data tables in Snowflake using Airflow
1. Develop an ELT process using dbt to create abstract tables with calculated weather metrics
2. Visualize key weather analytics using Tableau
3. Integrate all components into a cohesive, automated pipeline

The project utilizes Snowflake for data storage, Apache Airflow for orchestration, dbt for data transformation, and Tableau for visualization, leveraging each tool's strengths to create a comprehensive analytics solution.

II. System Architecture

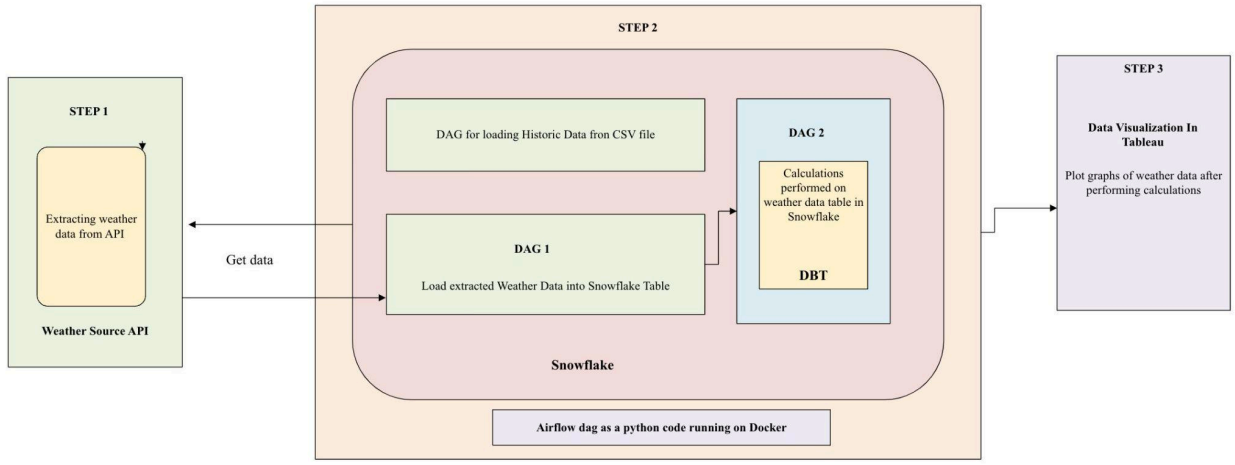


Fig. 1: High Level system diagram

The system architecture consists of four main components:

1. *Snowflake:* Acts as the central data warehouse, storing both raw and transformed data.
2. *Apache Airflow:* Orchestrates the entire pipeline, scheduling and executing both ETL and ELT processes.
3. *dbt:* Performs data transformations within Snowflake, creating abstract tables with calculated metrics.
4. *Tableau:* Connects to Snowflake to visualize the transformed data, providing interactive dashboards for analysis.

Data flows through the system as follows:

1. Raw stock data is ingested into Snowflake using Airflow-scheduled ETL processes.

2. dbt models, triggered by Airflow, transform the raw data into abstract tables within Snowflake.
3. Tableau connects to Snowflake, reading from the abstract tables to create visualizations.

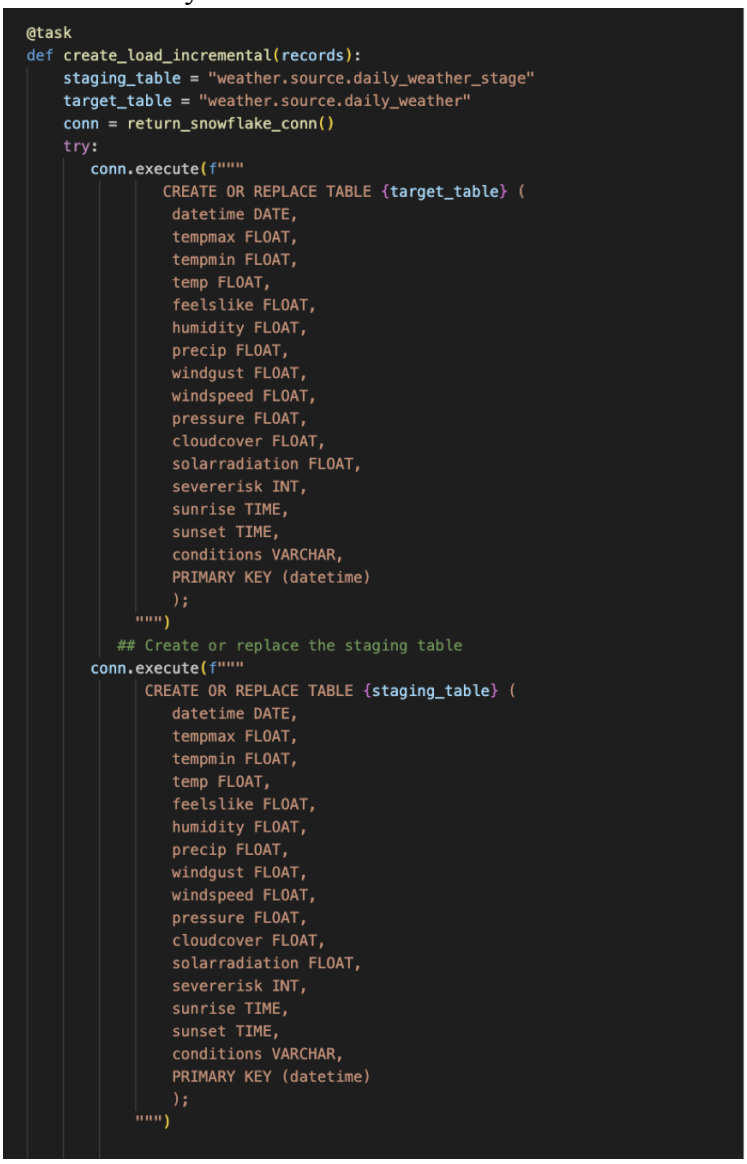
III. ETL Process

The ETL process for our weather data analytics pipeline consists of two main tasks:

- Historical Data Loading
- Recent Data Fetching

Historical Data Loading

We use the csv_historical_data_etl.py script to load historical weather data from January 1, 2020, to October 31, 2024, from a CSV file. This process ensures we have a substantial baseline of historical data for analysis.



```
@task
def create_load_incremental(records):
    staging_table = "weather.source.daily_weather_stage"
    target_table = "weather.source.daily_weather"
    conn = return_snowflake_conn()
    try:
       conn.execute(f"""
              CREATE OR REPLACE TABLE {target_table} (
               datetime DATE,
               tempmax FLOAT,
               tempmin FLOAT,
               temp FLOAT,
               feelslike FLOAT,
               humidity FLOAT,
               precip FLOAT,
               windgust FLOAT,
               windspeed FLOAT,
               pressure FLOAT,
               cloudcover FLOAT,
               solarradiation FLOAT,
               severerisk INT,
               sunrise TIME,
               sunset TIME,
               conditions VARCHAR,
               PRIMARY KEY (datetime)
               );
          """)
         ## Create or replace the staging table
       conn.execute(f"""
            CREATE OR REPLACE TABLE {staging_table} (
               datetime DATE,
               tempmax FLOAT,
               tempmin FLOAT,
               temp FLOAT,
               feelslike FLOAT,
               humidity FLOAT,
               precip FLOAT,
               windgust FLOAT,
               windspeed FLOAT,
               pressure FLOAT,
               cloudcover FLOAT,
               solarradiation FLOAT,
               severerisk INT,
               sunrise TIME,
               sunset TIME,
               conditions VARCHAR,
               PRIMARY KEY (datetime)
               );
          """)
```

Fig. 2a: Python code to load historical data from csv file

This script reads the CSV file, processes the data, and uploads it to Snowflake, creating a solid foundation for our weather analysis.

```
    # Insert records into the staging table
    for r in records:
        datetime = r['datetime']
        tempmax = r['tempmax']
        tempmin = r['tempmin']
        temp = r['temp']
        feelslike = r['feelslike']
        humidity = r['humidity']
        precip = r['precip']
        windgust = r['windgust']
        windspeed = r['windspeed']
        pressure = r['pressure']
        cloudcover = r['cloudcover']
        solarradiation = r['solarradiation']
        severerisk = r['severerisk']
        sunrise = r['sunrise']
        sunset = r['sunset']
        conditions = r['conditions']
        #insert_sql = f"INSERT INTO {staging_table} (datetime, tempmax, tempmin, temp, feelslike, humidity, precip, windspeed, cloudcover, conditions) VALUES ('{datetime}',{tempmax}, {tempmin}, {temp}, {feelslike}, {humidity}, {precip},{windspeed}, {cloudcover},{conditions})"
        #conn.execute(insert_sql) # Execute within the with block

        insert_sql = f"""INSERT INTO {staging_table} (datetime, tempmax, tempmin, temp, feelslike, humidity, precip, windgust, windspeed,pressure, cloudcover, solarradiation,
                    severerisk, sunrise, sunset, conditions
                    ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s) """
        conn.execute(insert_sql, ( datetime, tempmax, tempmin, temp, feelslike, humidity, precip, windgust, windspeed,pressure, cloudcover, solarradiation, severerisk, sunrise, sunset, conditions))

    conn.execute("COMMIT;")

    # perform UPSERT
    upsert_sql = f"""
        MERGE INTO {target_table} AS target
        USING {staging_table} AS stage
        ON target.datetime = stage.datetime
        WHEN MATCHED THEN
            UPDATE SET
                target.datetime = stage.datetime,
                target.tempmax = stage.tempmax,
                target.tempmin = stage.tempmin,
                target.temp = stage.temp,
                target.feelslike = stage.feelslike,
                target.humidity = stage.humidity,
                target.precip = stage.precip,
                target.windgust = stage.windgust,
                target.windspeed = stage.windspeed,
                target.pressure = stage.pressure,
                target.cloudcover = stage.cloudcover,
                target.solarradiation = stage.solarradiation,
                target.severerisk = stage.severerisk,
                target.sunrise = stage.sunrise,
                target.sunset = stage.sunset,
                target.conditions = stage.conditions
        WHEN NOT MATCHED THEN
            INSERT ( datetime, tempmax, tempmin, temp, feelslike, humidity, precip,  windgust, windspeed, pressure, cloudcover, solarradiation,  severerisk, sunrise, sunset,  conditions)
            VALUES (stage.datetime, stage.tempmax, stage.tempmin, stage.temp, stage.feelslike, stage.humidity, stage.precip, stage.windgust, stage.windspeed, stage.pressure, stage.cloudcover, stage.solarradiation, stage.severerisk,stage.sunrise, stage.sunset,  stage.conditions)
            """

    conn.execute(upsert_sql)
    #Commit the change
    conn.execute("COMMIT;")
    print(f"Stage Table {staging_table}, Target table create '{target_table}', Data loaded successfully in both the tables using Incremental Load ")
except Exception as e:
    conn.execute("ROLLBACK;")
    print(e)
    raise e
```

Fig. 2b: Python code to load historical data from csv file

Github link for Historical data ETL DAG code:
https://github.com/patilgayatri22/DataWarehouse_FinalProject/blob/main/DataWarehouseProject/dags/csv_historical_data_etl.py

Recent Data Fetching:

For more recent data, from November 1, 2024, to the current date, we use the project_etl_1.py script. This script fetches data from the Visual Crossing Weather API and loads it into Snowflake.

This script fetches the most recent weather data and uses a MERGE statement to insert or update records in Snowflake, ensuring our dataset remains current.

By combining these two ETL processes, we create a comprehensive weather dataset that spans historical data and includes up-to-date information. This approach allows for both long-term trend analysis and current weather insights, providing a robust foundation for our weather analytics pipeline.

Github link for Recent data ETL DAG code:

https://github.com/patilgayatri22/DataWarehouse_FinalProject/blob/main/DataWarehouseProject/dags/project_ETL_pipeline1.py

IV. ELT Process with dbt

The dbt project structure consists of models for calculating various weather metrics:

dbt code link:
https://github.com/patilgayatri22/DataWarehouse_FinalProject/tree/main/DataWarehouseProject/proj

```
proj/
├── analyses/
├── models/
│   ├── input/
│   │   └── daily_weather.sql
│   └── output/
│       ├── cumulative_precipitation.sql
│       ├── daily_changes.sql
│       ├── daylight_duration.sql
│       ├── pressure_drops.sql
│       ├── solar_efficiency.sql
│       └── temperature_range.sql
├── snapshots/
│   ├── snapshot.daily_changes.sql
│   ├── snapshot.daylight_duration.sql
│   ├── snapshot.pressure_drops.sql
│   ├── snapshot.solar_efficiency.sql
│   └── snapshot.temperature_range.sql
└── seeds/
```

Fig. 4: dbt Folder structure (main files)

Key transformations implemented:

1. temperature_range:

   Tracks daily temperature variations, including min/max ranges and rolling averages.

```sql
WITH temperature_range AS (
    SELECT
        DATETIME,
        TEMPMAX,
        TEMPMIN,
        (TEMPMAX - TEMPMIN) AS TEMP_RANGE,
        CONDITIONS
    FROM {{ ref('daily_weather') }}
)
SELECT
    DATETIME,
    TEMPMAX,
    TEMPMIN,
    TEMP_RANGE,
    CONDITIONS
FROM temperature_range
ORDER BY DATETIME DESC
```

Figure 5 : Code to calculate temperature range

2. solar_efficiency:

   Analyzes solar radiation patterns accounting for cloud cover impact.

```sql
WITH weather_data AS (
    SELECT
        DATETIME,
        solarradiation,
        cloudcover,
        CASE
            WHEN cloudcover > 0 THEN solarradiation / cloudcover
            ELSE NULL
        END AS solar_efficiency
    FROM {{ ref('daily_weather') }}
)
SELECT
    DATETIME,
    solarradiation,
    cloudcover,
    solar_efficiency
FROM weather_data
```

Figure 6 : Code to calculate solar efficiency

3. Daily_changes:

   Monitors day-over-day changes in key weather metrics.

```sql
WITH weather_data AS (
    SELECT
        datetime,
        temp,
        humidity,
        LAG(temp) OVER (ORDER BY datetime) AS prev_temp,
        LAG(humidity) OVER (ORDER BY datetime) AS prev_humidity
    FROM {{ ref('daily_weather') }}
),
calculated_changes AS (
    SELECT
        datetime,
        temp,
        prev_temp,
        ((temp - prev_temp) / NULLIF(prev_temp, 0)) * 100 AS temp_change,
        humidity,
        prev_humidity,
        ((humidity - prev_humidity) / NULLIF(prev_humidity, 0)) * 100 AS humidity_change
    FROM weather_data
)
SELECT
    datetime,
    temp,
    prev_temp,
    temp_change,
    humidity,
    prev_humidity,
    humidity_change
FROM calculated_changes
ORDER BY datetime
```

Figure 7: Code to calculate daily changes

4. Daylight_duration:

Calculates daily sunlight hours and tracks seasonal changes.

```sql
WITH daylight_hours_data AS (
    SELECT
        datetime,
        CAST(CONCAT(datetime, ' ', sunrise) AS TIMESTAMP) AS sunrise_ts,
        CAST(CONCAT(datetime, ' ', sunset) AS TIMESTAMP) AS sunset_ts,
        DATEDIFF('second',
                 CAST(CONCAT(datetime, ' ', sunrise) AS TIMESTAMP),
                 CAST(CONCAT(datetime, ' ', sunset) AS TIMESTAMP)
        ) / 3600.0 AS daylight_hours
    FROM {{ ref('daily_weather') }}
)
SELECT
    datetime,
    sunrise_ts AS sunrise,
    sunset_ts AS sunset,
    daylight_hours
FROM daylight_hours_data
ORDER BY datetime DESC
```

Figure 8: Code to calculate daily duration

5. Cumulative_precipitation:

Maintains running totals and moving averages of precipitation.

```sql
WITH cumulative_precip_data AS (
    SELECT
        datetime,
        precip,
        SUM(precip) OVER (ORDER BY datetime ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cumulative_precip
    FROM {{ ref('daily_weather') }}
)
SELECT
    datetime,
    precip,
    cumulative_precip
FROM cumulative_precip_data
ORDER BY datetime DESC
```

Figure 9 : Code to calculate cumulative precipitation

6. Pressure_drops:

   Identifies significant weather changes by analyzing temperature, humidity, and wind patterns.

```sql
WITH weather_data AS (
    SELECT
        datetime,
        pressure,
        LAG(pressure) OVER (ORDER BY datetime) AS prev_pressure,
        (pressure - LAG(pressure) OVER (ORDER BY datetime)) AS pressure_change,
        CASE
            WHEN (pressure - LAG(pressure) OVER (ORDER BY datetime)) < -2 THEN 'Yes'
            ELSE 'No'
        END AS pressure_drop
    FROM {{ ref('daily_weather') }}
)
SELECT
    datetime,
    pressure,
    prev_pressure,
    pressure_change,
    pressure_drop
FROM weather_data
```

Figure 10 : Code to calculate pressure drop

Integration of dbt with Airflow:

```
Follows the standard dbt seed, run, and test pattern.
"""

from pendulum import datetime

from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.hooks.base import BaseHook


DBT_PROJECT_DIR = "/opt/airflow/dbt"


conn = BaseHook.get_connection('snowflake_conn')
with DAG(
    "BuildELT_dbt",
    start_date=datetime(2024, 10, 14),
    description="A sample Airflow DAG to invoke dbt runs using a BashOperator",
    schedule=None,
    catchup=False,
    default_args={
        "env": {
            "DBT_USER": conn.login,
            "DBT_PASSWORD": conn.password,
            "DBT_ACCOUNT": conn.extra_dejson.get("account"),
            "DBT_SCHEMA": conn.schema,
            "DBT_DATABASE": conn.extra_dejson.get("database"),
            "DBT_ROLE": conn.extra_dejson.get("role"),
            "DBT_WAREHOUSE": conn.extra_dejson.get("warehouse"),
            "DBT_TYPE": "snowflake"
        }
    },
) as dag:
    dbt_run = BashOperator(
        task_id="dbt_run",
        bash_command=f"/home/airflow/.local/bin/dbt run --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}",
    )

    dbt_test = BashOperator(
        task_id="dbt_test",
        bash_command=f"/home/airflow/.local/bin/dbt test --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}",
    )

    dbt_snapshot = BashOperator(
        task_id="dbt_snapshot",
        bash_command=f"/home/airflow/.local/bin/dbt snapshot --profiles-dir {DBT_PROJECT_DIR} --project-dir {DBT_PROJECT_DIR}",
    )

    # print_env_var = BashOperator(
    #     task_id='print_aa_variable',
    #     bash_command='echo "The value of AA is: $DBT_ACCOUNT,$DBT_ROLE,$DBT_DATABASE,$DBT_WAREHOUSE,$DBT_USER,$DBT_TYPE,$DBT_SCHEMA"'
    # )

    dbt_run >> dbt_test >> dbt_snapshot
```

Fig. 11: Python code for ELT DAG

This DAG runs the dbt models daily, ensuring that the abstract tables in Snowflake are updated with the latest calculations.

TABLE STRUCTURES

Table - Daily Weather

NAME ↑	TYPE
CLOUDCOVER	Float
CONDITIONS	Varchar
DATETIME	Date
FEELSLIKE	Float
HUMIDITY	Float
PRECIP	Float
PRESSURE	Float
SEVERERISK	Number
SOLARRADIATION	Float
SUNRISE	Time
SUNSET	Time
TEMP	Float
TEMPMAX	Float
TEMPMIN	Float
WINDGUST	Float
WINDSPEED	Float

Table - Temperature Range

NAME ↑	TYPE
CONDITIONS	Varchar
DATETIME	Date
TEMPMAX	Float
TEMPMIN	Float
TEMP_RANGE	Float

Table - Daily changes

NAME ↑	TYPE
DATETIME	Date
HUMIDITY	Float
HUMIDITY_CHANGE	Float
PREV_HUMIDITY	Float
PREV_TEMP	Float
TEMP	Float
TEMP_CHANGE	Float

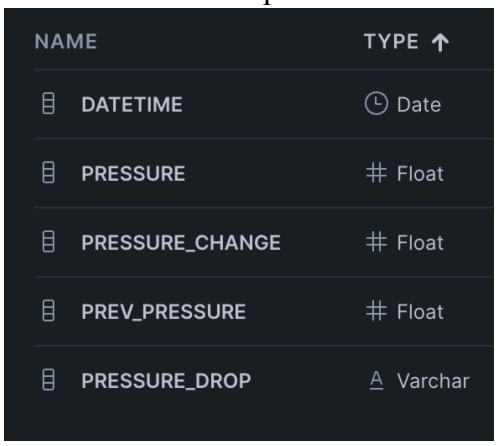
Table- Daylight Duration

NAME ↑	TYPE
DATETIME	Date
DAYLIGHT_HOURS	Number
SUNRISE	Timestamp_NTZ
SUNSET	Timestamp_NTZ

Table - Solar Efficiency

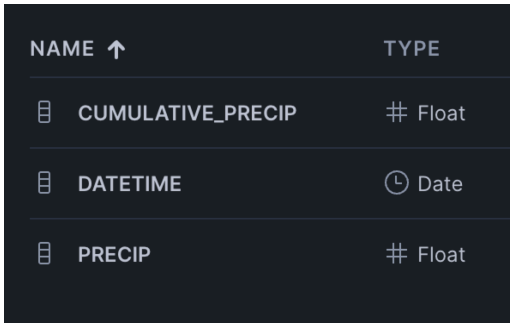
NAME ↑	TYPE
CLOUDCOVER	Float
DATETIME	Date
SOLARRADIATION	Float
SOLAR_EFFICIENCY	Float

Table - Pressure Drops



NAME	TYPE
DATETIME	Date
PRESSURE	Float
PRESSURE_CHANGE	Float
PREV_PRESSURE	Float
PRESSURE_DROP	Varchar

Table- Cumulative precipitation



NAME	TYPE
CUMULATIVE_PRECIP	Float
DATETIME	Date
PRECIP	Float

V. Airflow Data Pipelines

Airflow is a platform used to automate and manage workflows by organizing tasks into Directed Acyclic Graphs (DAGs). It enables users to schedule tasks, monitor progress, and ensure data is processed in the correct order or in parallel, making it ideal for automating ETL processes in data pipelines.

ETL Dag-

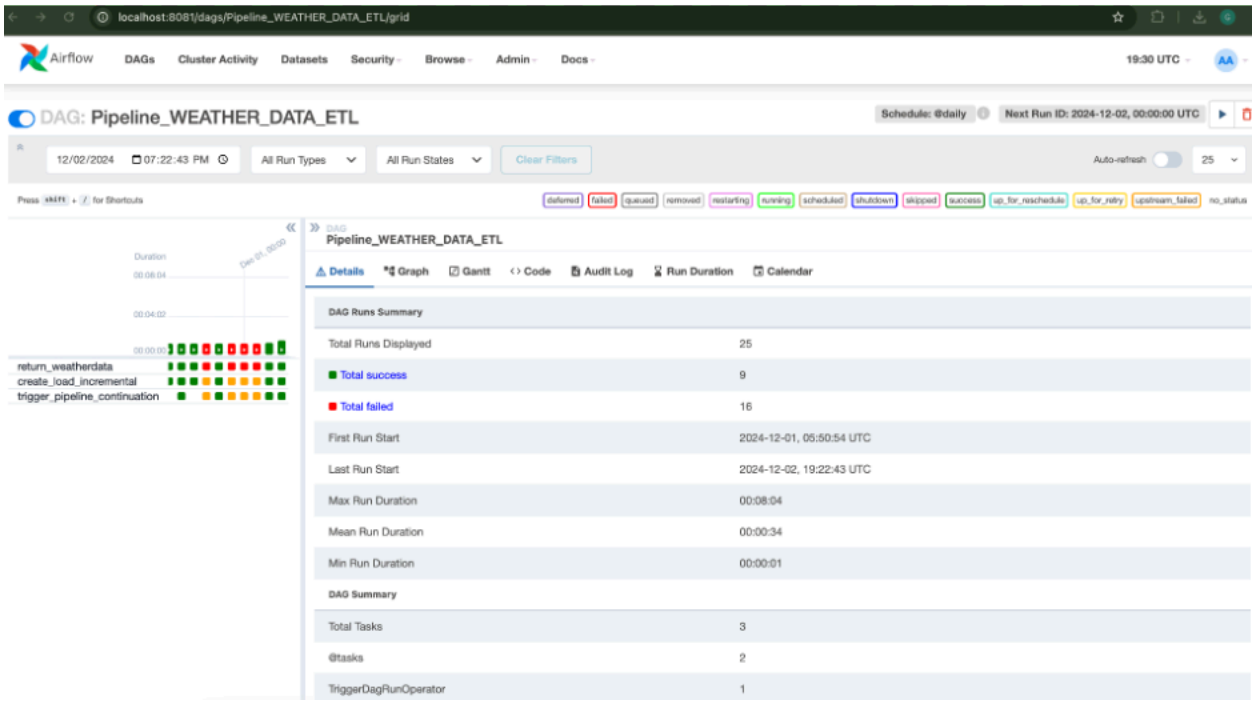


Fig. 12: Airflow ETL DAG

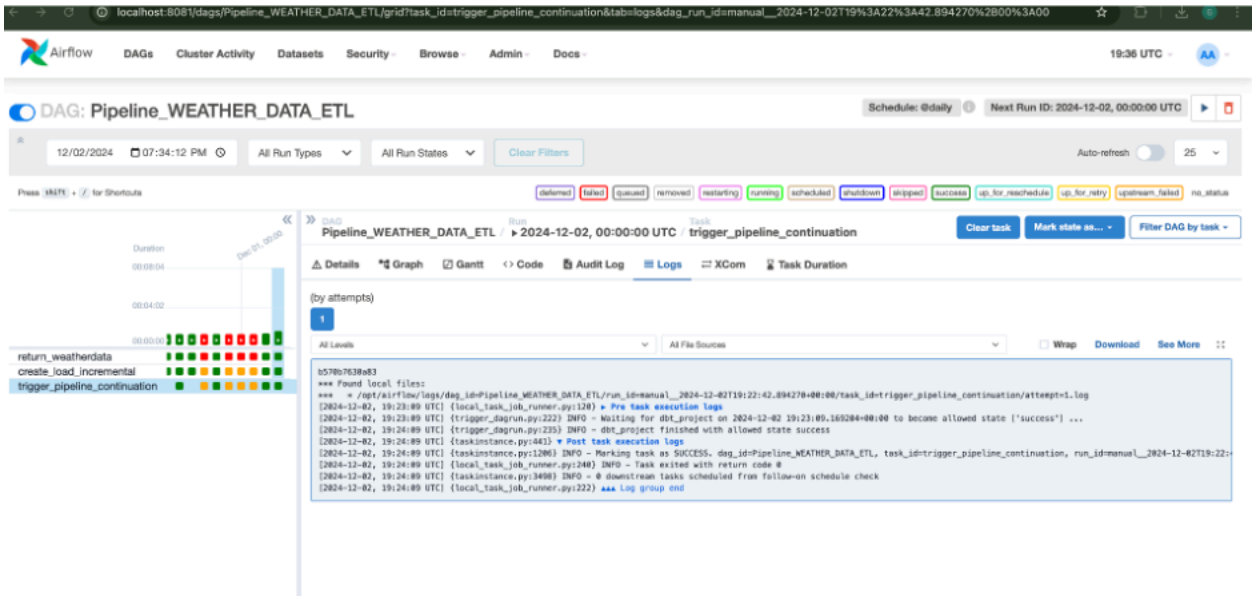


Fig 13 : Airflow ETL Dag with Logs

ELT Dag -

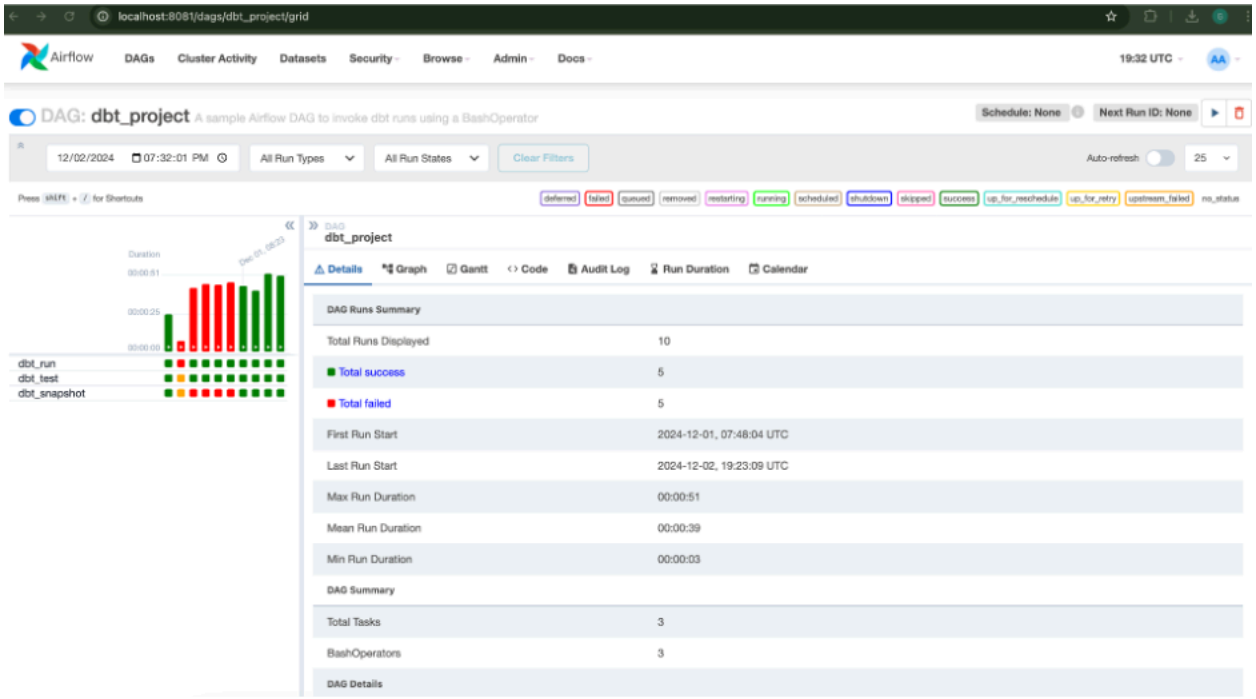


Fig 14 : Airflow ELT Dag

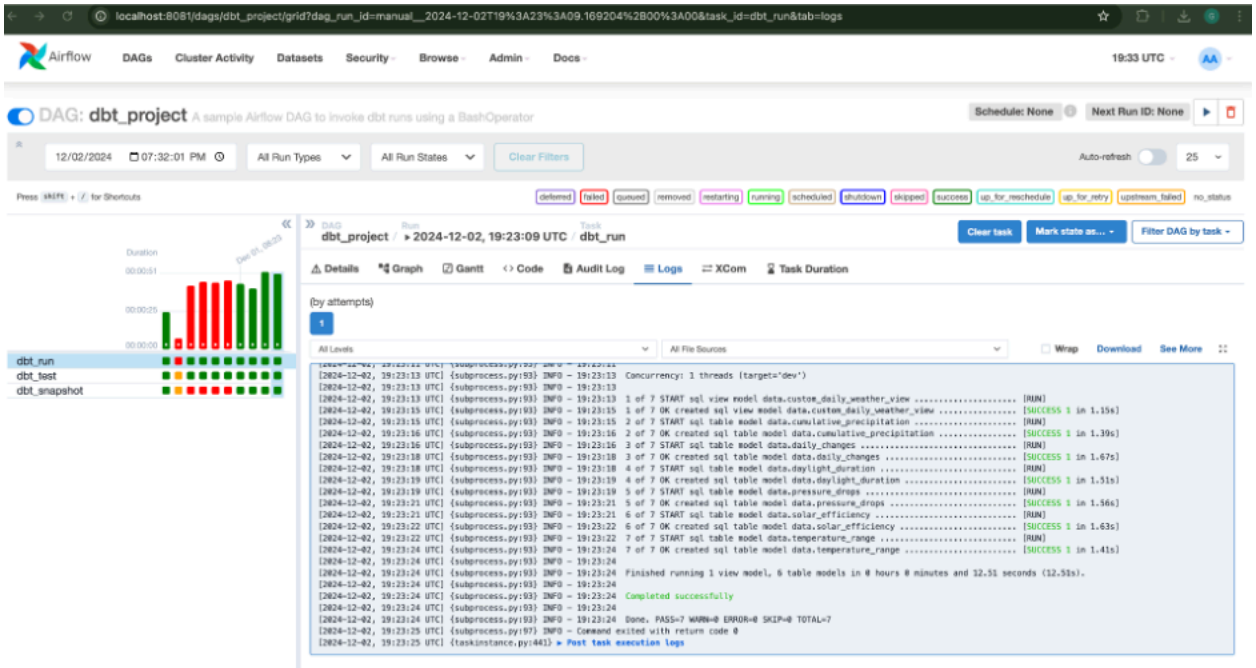


Fig 15a : Airflow ETL Dag with Logs

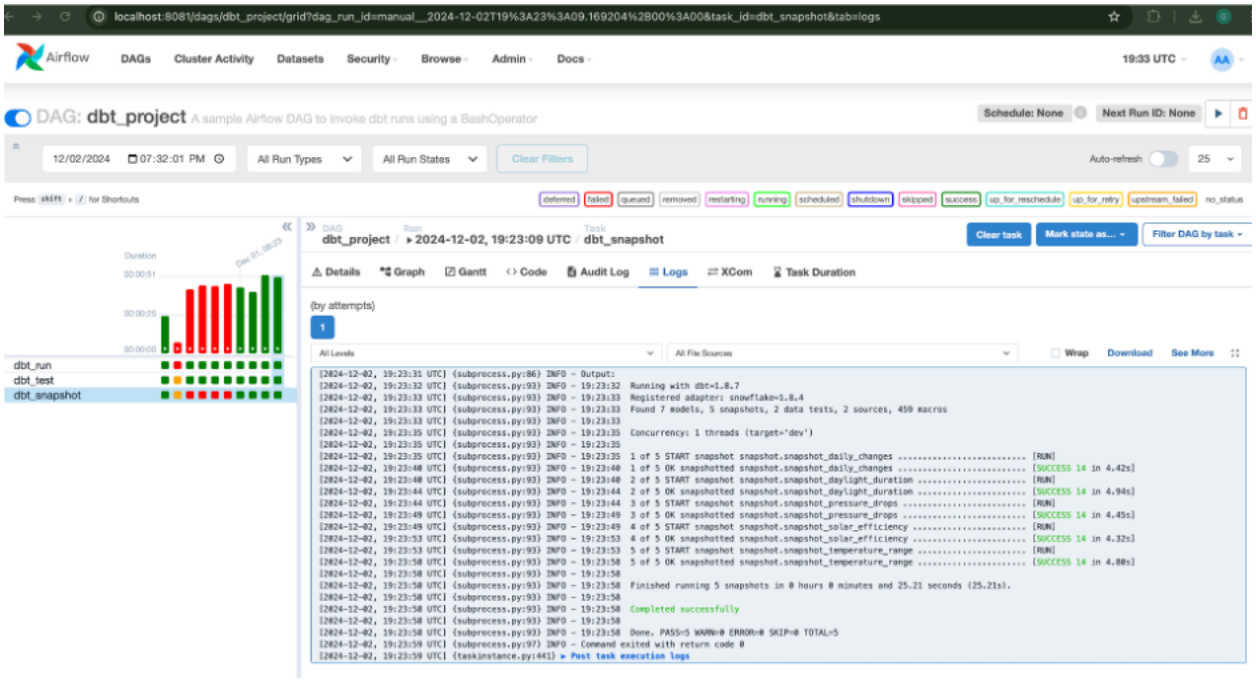


Fig 15b : Airflow ETL Dag with Logs

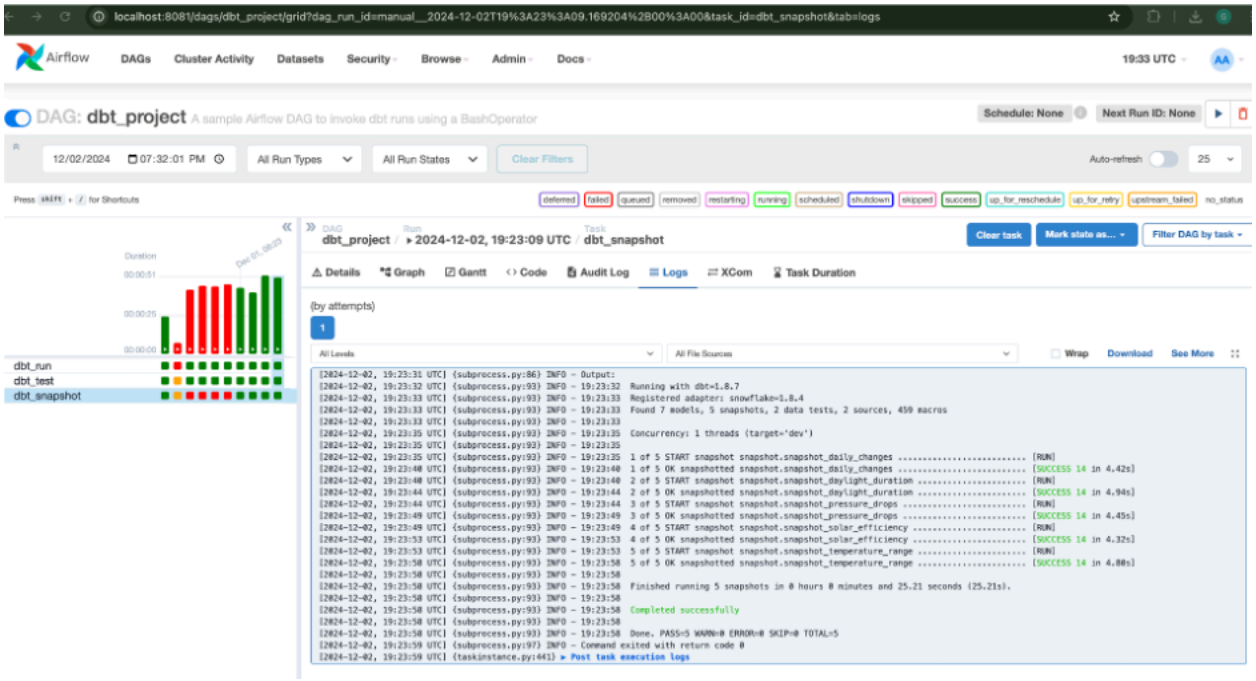


Fig 15c : Airflow ETL Dag with Logs

DBT screenshots from terminal-

```
gayatripatil@Gayatris-MacBook-Air proj % dbt run
04:51:12  Running with dbt=1.8.7
04:51:13  Registered adapter: snowflake=1.8.4
04:51:13  Found 7 models, 2 data tests, 2 sources, 459 macros
04:51:13
04:51:15  Concurrency: 1 threads (target='dev')
04:51:15
04:51:15  1 of 7 START sql view model data.custom_daily_weather_view ..... [RUN]
04:51:16  1 of 7 OK created sql view model data.custom_daily_weather_view ..... [SUCCESS 1 in 1.50s]
04:51:16  2 of 7 START sql table model data.cumulative_precipitation ..... [RUN]
04:51:18  2 of 7 OK created sql table model data.cumulative_precipitation ..... [SUCCESS 1 in 1.69s]
04:51:18  3 of 7 START sql table model data.daily_changes ..... [RUN]
04:51:19  3 of 7 OK created sql table model data.daily_changes ..... [SUCCESS 1 in 1.76s]
04:51:19  4 of 7 START sql table model data.daylight_duration ..... [RUN]
04:51:22  4 of 7 OK created sql table model data.daylight_duration ..... [SUCCESS 1 in 2.16s]
04:51:22  5 of 7 START sql table model data.pressure_drops ..... [RUN]
04:51:23  5 of 7 OK created sql table model data.pressure_drops ..... [SUCCESS 1 in 1.29s]
04:51:23  6 of 7 START sql table model data.solar_efficiency ..... [RUN]
04:51:25  6 of 7 OK created sql table model data.solar_efficiency ..... [SUCCESS 1 in 2.16s]
04:51:25  7 of 7 START sql table model data.temperature_range ..... [RUN]
04:51:27  7 of 7 OK created sql table model data.temperature_range ..... [SUCCESS 1 in 1.52s]
04:51:27
04:51:27  Finished running 1 view model, 6 table models in 0 hours 0 minutes and 13.45 seconds (13.45s).
04:51:27
04:51:27  Completed successfully
04:51:27
04:51:27  Done. PASS=7 WARN=0 ERROR=0 SKIP=0 TOTAL=7
gayatripatil@Gayatris-MacBook-Air proj %
```

Fig 16 : DBT Run

```
gayatripatil@Gayatris-MacBook-Air proj % dbt snapshot
06:17:26  Running with dbt=1.8.7
06:17:26  Registered adapter: snowflake=1.8.4
06:17:27  Found 7 models, 2 data tests, 5 snapshots, 2 sources, 459 macros
06:17:27
06:17:30  Concurrency: 1 threads (target='dev')
06:17:30
06:17:30  1 of 5 START snapshot snapshot.snapshot_daily_changes ..... [RUN]
06:17:32  1 of 5 OK snapshotted snapshot.snapshot_daily_changes ..... [SUCCESS 1 in 2.73s]
06:17:32  2 of 5 START snapshot snapshot.snapshot_daylight_duration ..... [RUN]
06:17:35  2 of 5 OK snapshotted snapshot.snapshot_daylight_duration ..... [SUCCESS 1 in 2.30s]
06:17:35  3 of 5 START snapshot snapshot.snapshot_pressure_drops ..... [RUN]
06:17:37  3 of 5 OK snapshotted snapshot.snapshot_pressure_drops ..... [SUCCESS 1 in 1.77s]
06:17:37  4 of 5 START snapshot snapshot.snapshot_solar_efficiency ..... [RUN]
06:17:38  4 of 5 OK snapshotted snapshot.snapshot_solar_efficiency ..... [SUCCESS 1 in 1.92s]
06:17:38  5 of 5 START snapshot snapshot.snapshot_temperature_range ..... [RUN]
06:17:41  5 of 5 OK snapshotted snapshot.snapshot_temperature_range ..... [SUCCESS 1 in 2.44s]
06:17:41
06:17:41  Finished running 5 snapshots in 0 hours 0 minutes and 14.39 seconds (14.39s).
06:17:41
06:17:41  Completed successfully
06:17:41
06:17:41  Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
gayatripatil@Gayatris-MacBook-Air proj %
```

Fig 16 : DBT Snapshot

```
gayatripatil@Gayatris-MacBook-Air proj % dbt test
06:23:31  Running with dbt=1.8.7
06:23:31  Registered adapter: snowflake=1.8.4
06:23:31  Found 7 models, 5 snapshots, 2 data tests, 2 sources, 459 macros
06:23:31
06:23:33  Concurrency: 1 threads (target='dev')
06:23:33
06:23:33  1 of 2 START test not_null_temperature_range_datetime ..... [RUN]
06:23:34  1 of 2 PASS not_null_temperature_range_datetime ..... [PASS in 0.76s]
06:23:34  2 of 2 START test unique_temperature_range_datetime ..... [RUN]
06:23:34  2 of 2 PASS unique_temperature_range_datetime ..... [PASS in 0.76s]
06:23:34
06:23:34  Finished running 2 data tests in 0 hours 0 minutes and 3.04 seconds (3.04s).
06:23:34
06:23:34  Completed successfully
06:23:34
06:23:34  Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
gayatripatil@Gayatris-MacBook-Air proj %
```

Fig 16 : DBT Test

VI. Snowflake screenshots for all tables

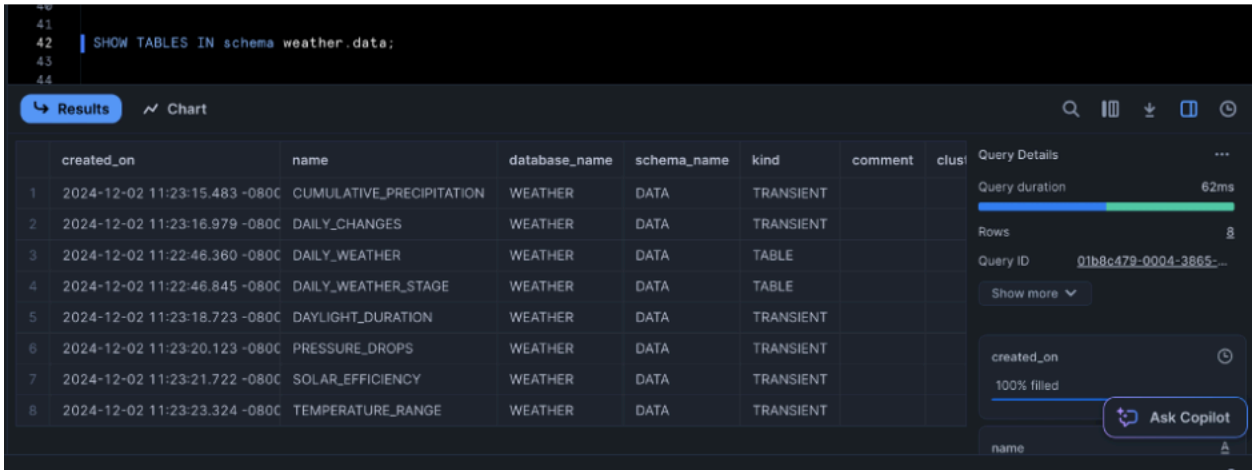


Fig 17a: All Tables Created in the Snowflake

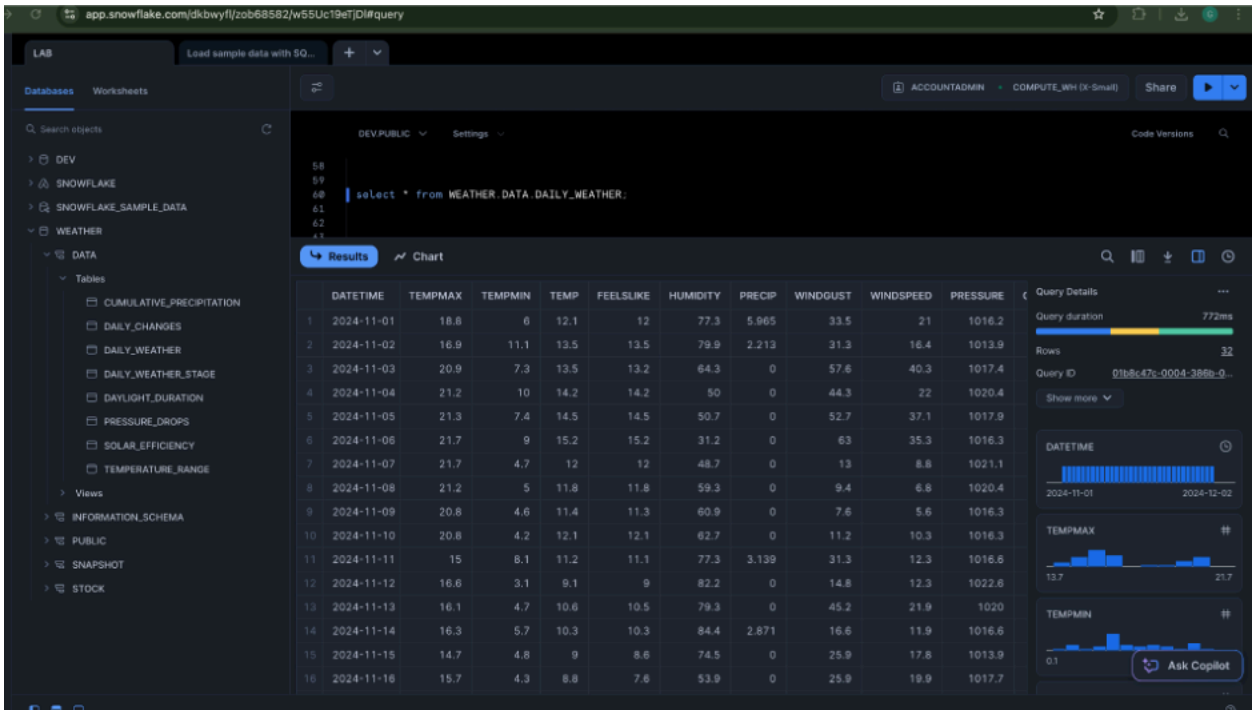


Fig 17b: Table Daily_Weather

Tables Created through DBT run -

Fig 17c: Table Temperature Range

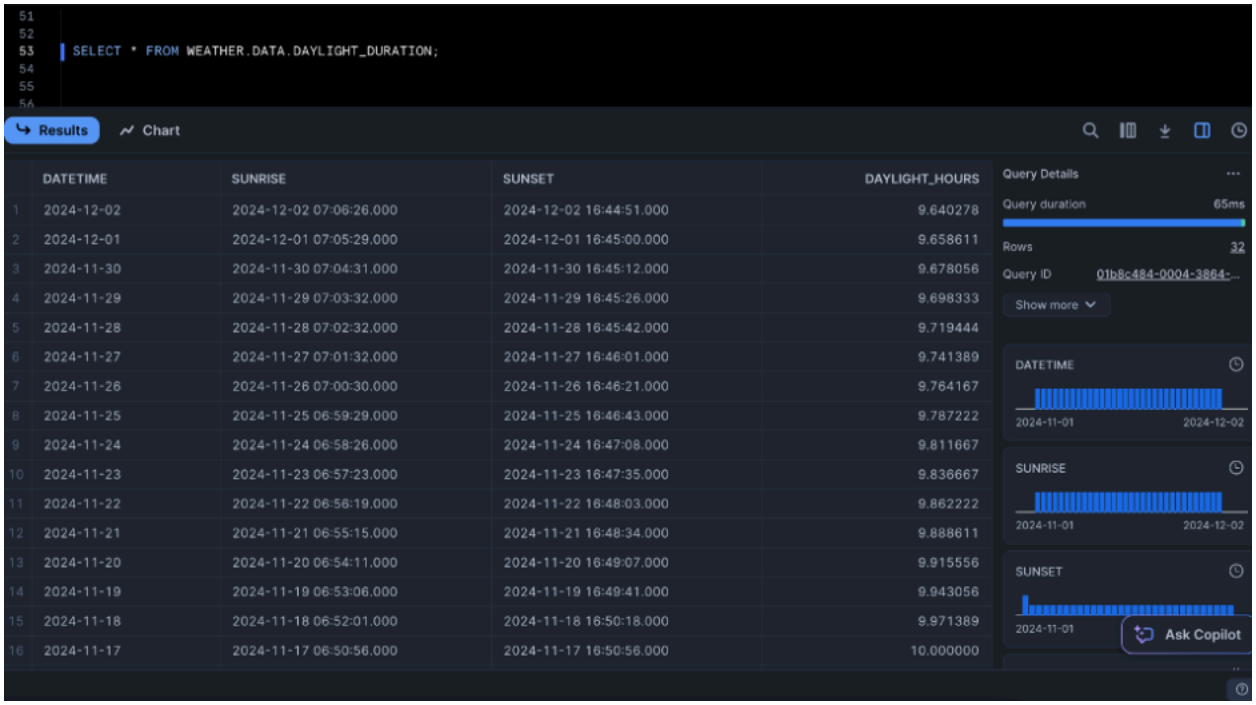
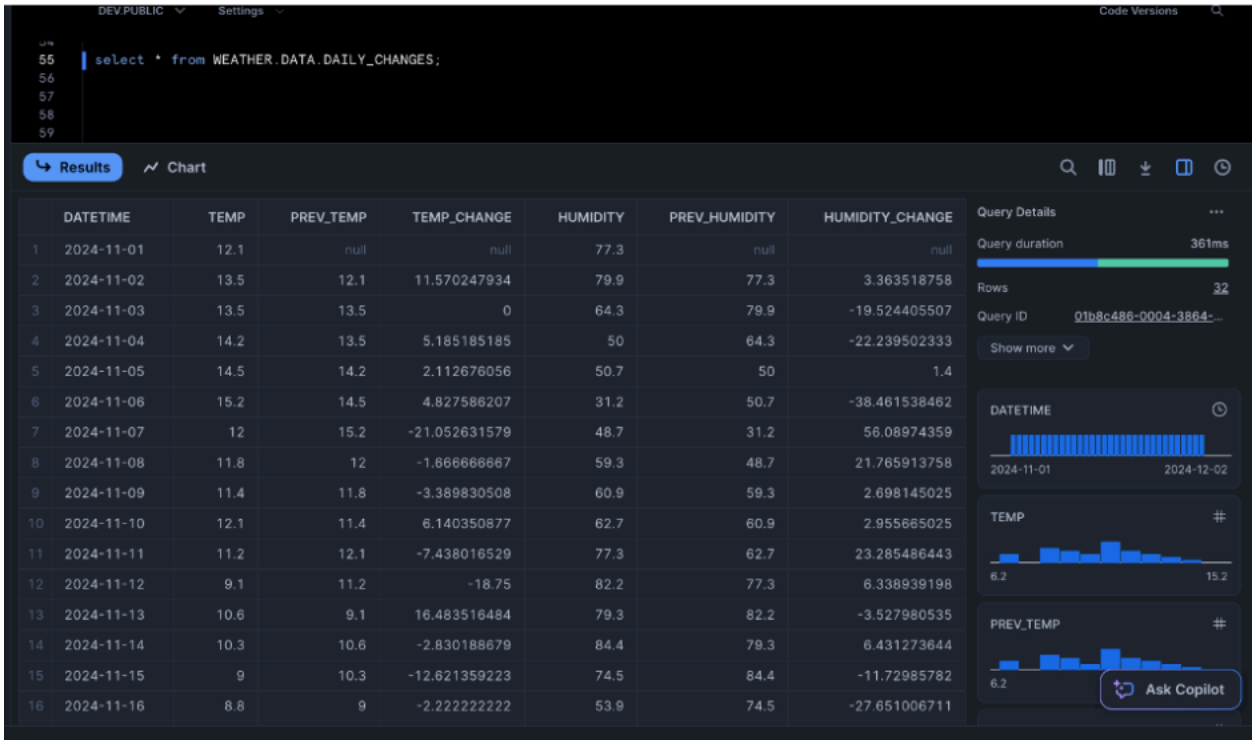


Fig 17d: Table Daylight Duration

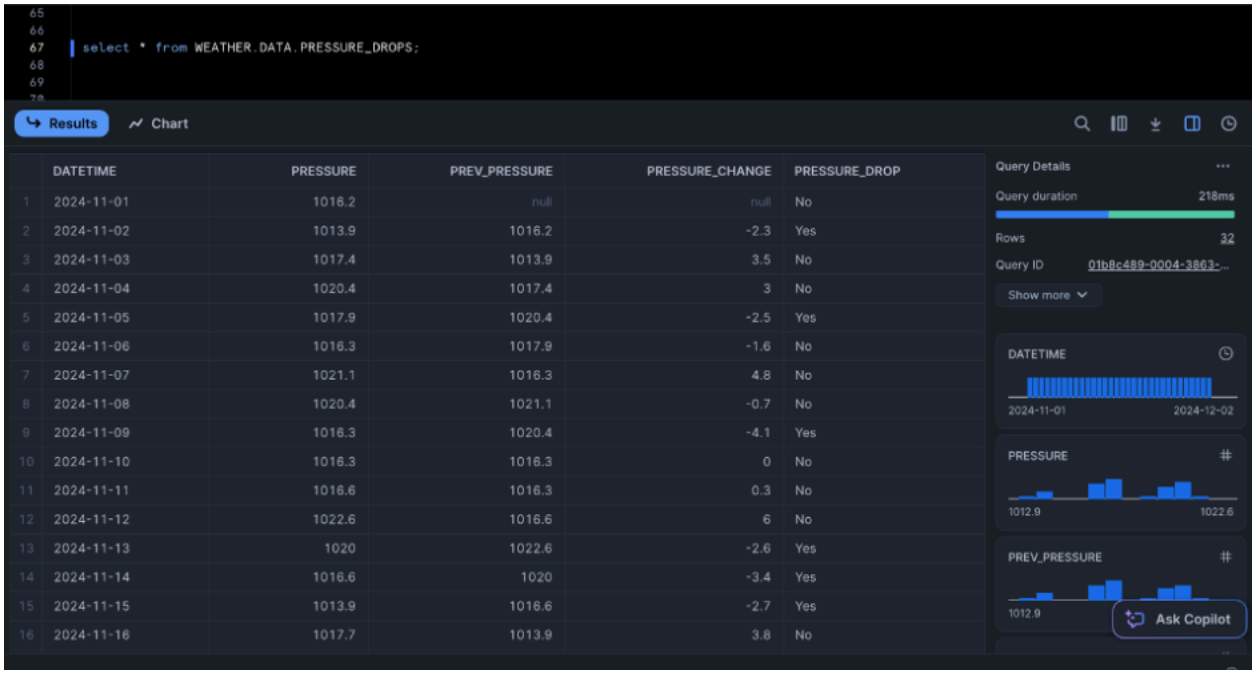


	DATETIME	TEMP	PREV_TEMP	TEMP_CHANGE	HUMIDITY	PREV_HUMIDITY	HUMIDITY_CHANGE
1	2024-11-01	12.1	null	null	77.3	null	null
2	2024-11-02	13.5	12.1	11.570247934	79.9	77.3	3.363518758
3	2024-11-03	13.5	13.5	0	64.3	79.9	-19.524405507
4	2024-11-04	14.2	13.5	5.185185185	50	64.3	-22.239502333
5	2024-11-05	14.5	14.2	2.112676056	50.7	50	1.4
6	2024-11-06	15.2	14.5	4.827586207	31.2	50.7	-38.461538462
7	2024-11-07	12	15.2	-21.052631579	48.7	31.2	56.08974359
8	2024-11-08	11.8	12	-1.666666667	59.3	48.7	21.765913758
9	2024-11-09	11.4	11.8	-3.389830508	60.9	59.3	2.698145025
10	2024-11-10	12.1	11.4	6.140350877	62.7	60.9	2.955665025
11	2024-11-11	11.2	12.1	-7.438016529	77.3	62.7	23.285486443
12	2024-11-12	9.1	11.2	-18.75	82.2	77.3	6.338939198
13	2024-11-13	10.6	9.1	16.483516484	79.3	82.2	-3.527980535
14	2024-11-14	10.3	10.6	-2.830188679	84.4	79.3	6.431273644
15	2024-11-15	9	10.3	-12.621359223	74.5	84.4	-11.72985782
16	2024-11-16	8.8	9	-2.222222222	53.9	74.5	-27.651006711

Fig 17e: Table Daily Changes

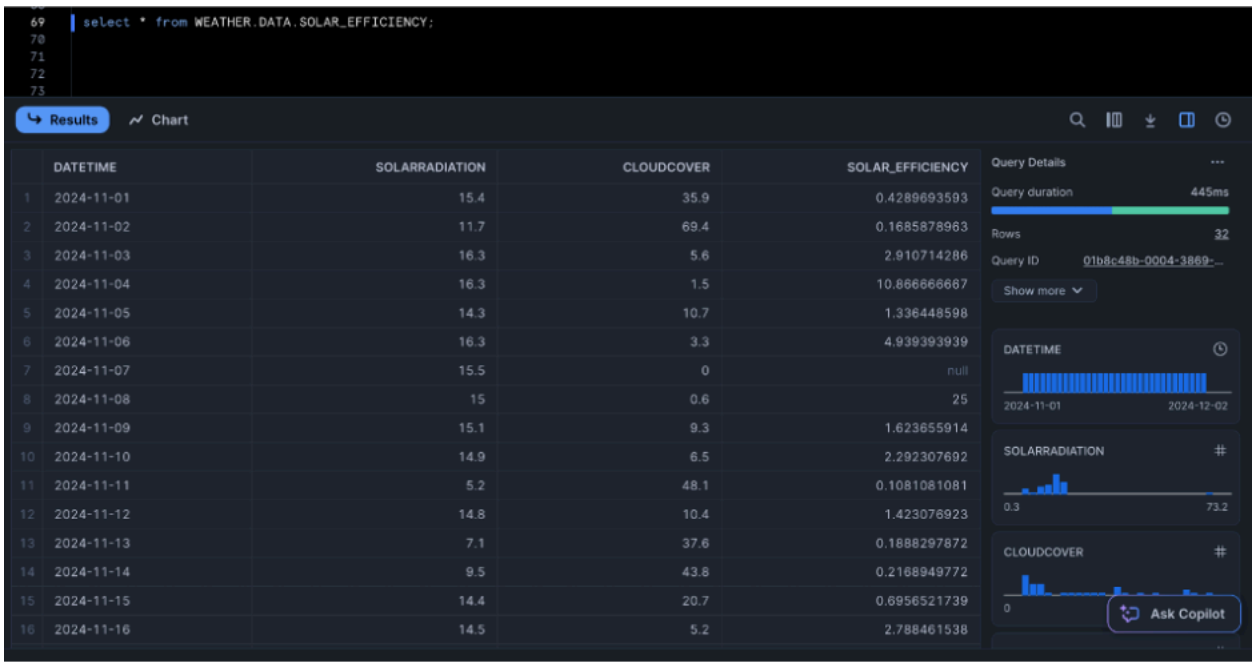
	DATETIME	PRECIP	CUMULATIVE_PRECIP
1	2024-12-02	0	82.131
2	2024-12-01	0	82.131
3	2024-11-30	0	82.131
4	2024-11-29	0	82.131
5	2024-11-28	0	82.131
6	2024-11-27	0	82.131
7	2024-11-26	2.856	82.131
8	2024-11-25	2.496	79.275
9	2024-11-24	0	76.779
10	2024-11-23	4.569	76.779
11	2024-11-22	43.525	72.21
12	2024-11-21	6.178	28.685
13	2024-11-20	8.087	22.507
14	2024-11-19	0	14.42
15	2024-11-18	0	14.42
16	2024-11-17	0.232	14.42

Fig 17f: Table Cumulative Precipitation



	DATETIME	PRESSURE	PREV_PRESSURE	PRESSURE_CHANGE	PRESSURE_DROP
1	2024-11-01	1016.2	null	null	No
2	2024-11-02	1013.9	1016.2	-2.3	Yes
3	2024-11-03	1017.4	1013.9	3.5	No
4	2024-11-04	1020.4	1017.4	3	No
5	2024-11-05	1017.9	1020.4	-2.5	Yes
6	2024-11-06	1016.3	1017.9	-1.6	No
7	2024-11-07	1021.1	1016.3	4.8	No
8	2024-11-08	1020.4	1021.1	-0.7	No
9	2024-11-09	1016.3	1020.4	-4.1	Yes
10	2024-11-10	1016.3	1016.3	0	No
11	2024-11-11	1016.6	1016.3	0.3	No
12	2024-11-12	1022.6	1016.6	6	No
13	2024-11-13	1020	1022.6	-2.6	Yes
14	2024-11-14	1016.6	1020	-3.4	Yes
15	2024-11-15	1013.9	1016.6	-2.7	Yes
16	2024-11-16	1017.7	1013.9	3.8	No

Fig 17g: Table Pressure Drops



	DATETIME	SOLARRADIATION	CLOUDCOVER	SOLAR_EFFICIENCY
1	2024-11-01	15.4	35.9	0.4289693593
2	2024-11-02	11.7	69.4	0.1685878963
3	2024-11-03	16.3	5.6	2.910714286
4	2024-11-04	16.3	1.5	10.866666667
5	2024-11-05	14.3	10.7	1.336448598
6	2024-11-06	16.3	3.3	4.939393939
7	2024-11-07	15.5	0	null
8	2024-11-08	15	0.6	25
9	2024-11-09	15.1	9.3	1.623655914
10	2024-11-10	14.9	6.5	2.292307692
11	2024-11-11	5.2	48.1	0.1081081081
12	2024-11-12	14.8	10.4	1.423076923
13	2024-11-13	7.1	37.6	0.1888297872
14	2024-11-14	9.5	43.8	0.2168949772
15	2024-11-15	14.4	20.7	0.6956521739
16	2024-11-16	14.5	5.2	2.788461538

Fig 17h: Table Solar Efficiency

DBT Snapshots screens -

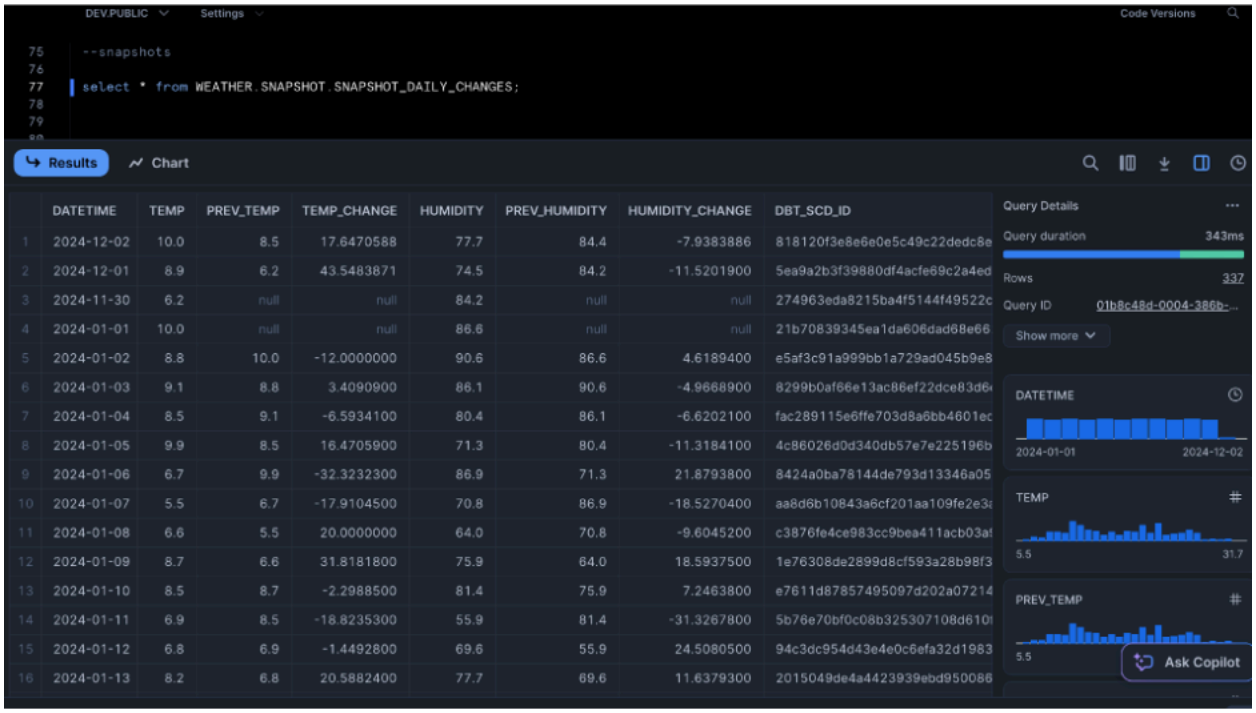


Fig 18a: Snapshot Daily Changes

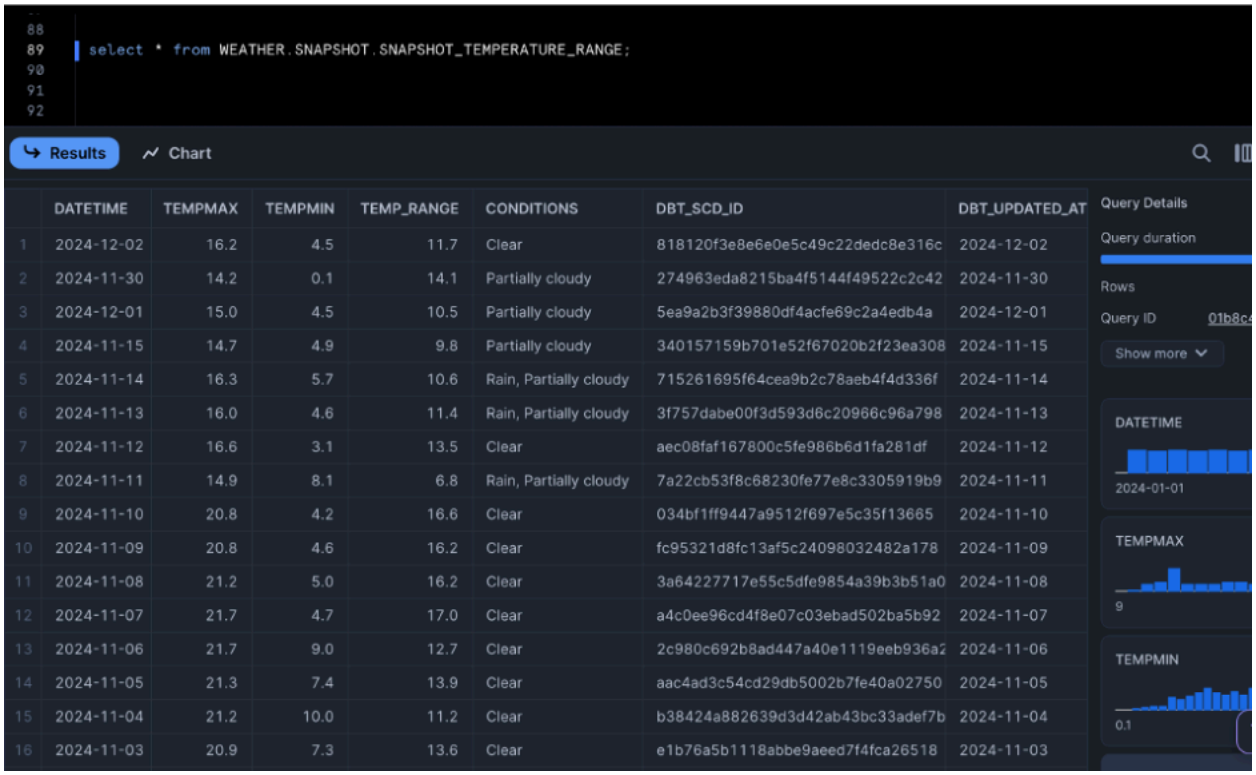


Fig 18b: Snapshot Temperature Range

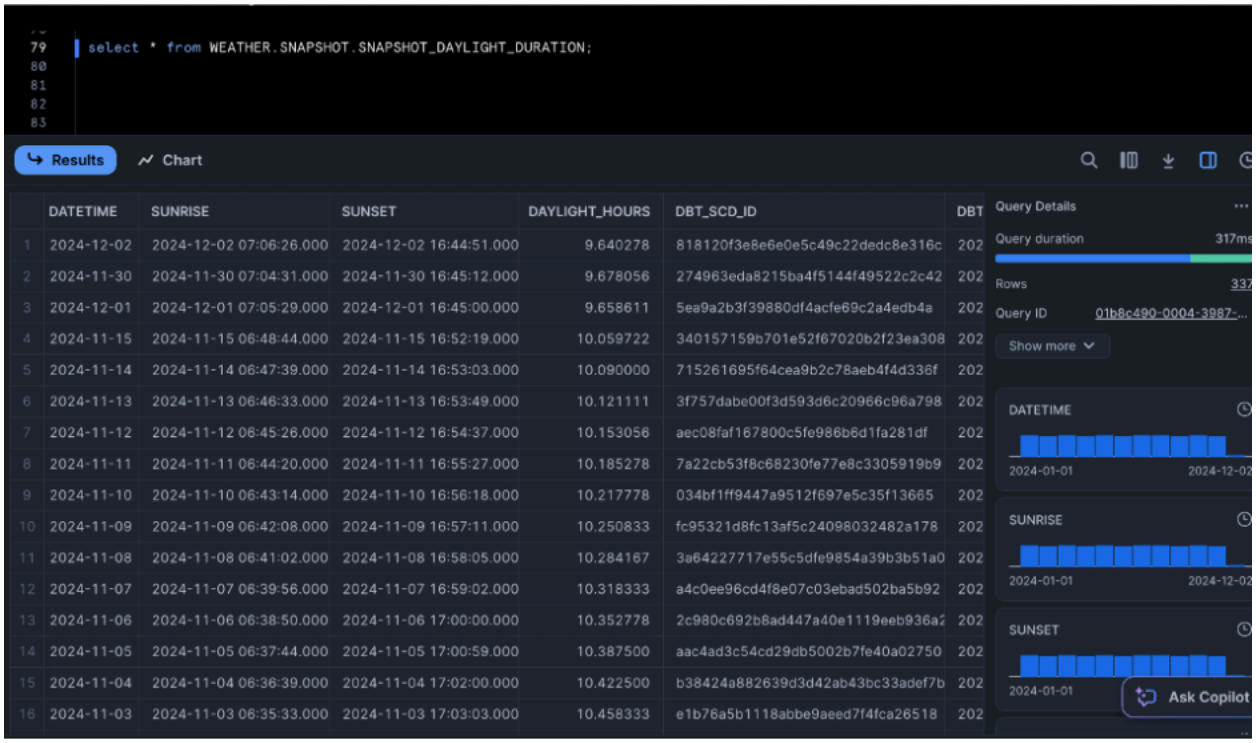


Fig 18c: Snapshot Daylight Duration

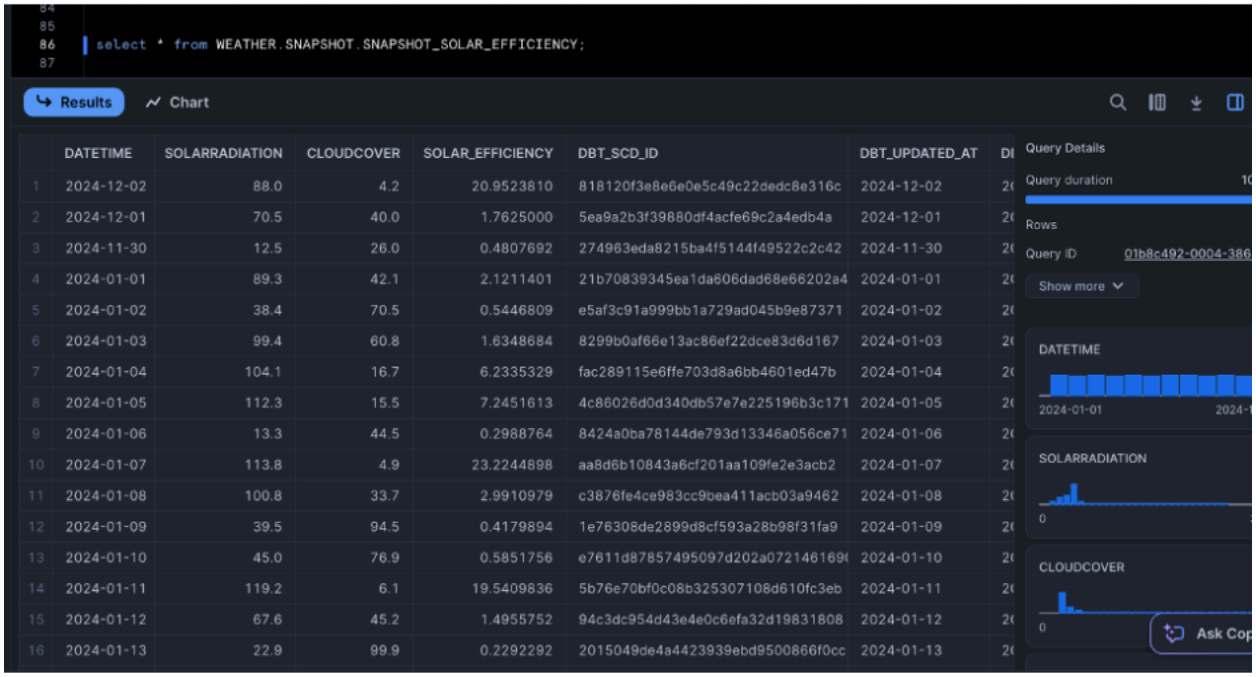


Fig 18d: Snapshot Solar Efficiency

Fig 18e: Snapshot Pressure Drops

VII. Data Visualization

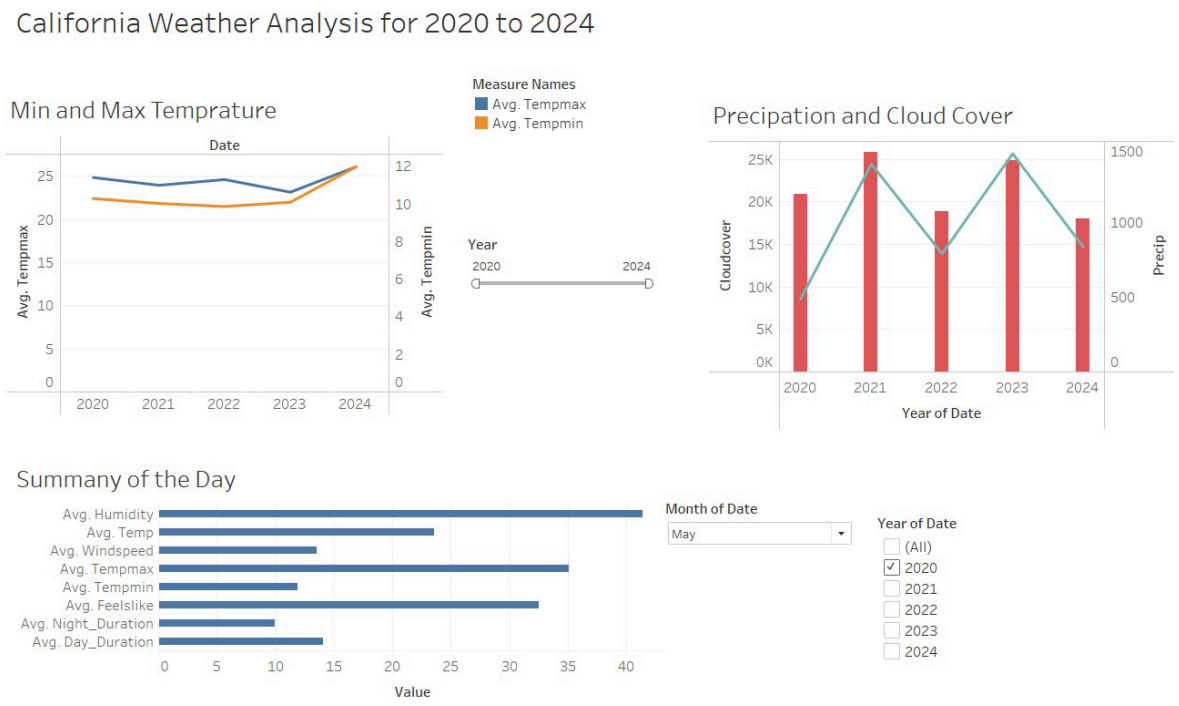


Figure 19: Historical data analysis dashboard

The dashboard contains three distinct visualizations of California's weather data:

1. Min and Max Temperature Chart
    - Shows a gradual convergence of maximum and minimum temperatures from 2020 to 2024, with both lines meeting around 2024 at approximately 25°C
    - Maximum temperatures remain relatively stable around 25°C while minimum temperatures show an upward trend from about 10°C to 12°C

2. Precipitation and Cloud Cover Chart
    - Displays a cyclical pattern with peak precipitation levels in 2021 and 2023, shown by red bars reaching approximately 25K units
    - Cloud cover (blue line) correlates with precipitation patterns, showing highest coverage during the same peak years and declining in 2024
3. Summary of the Day Chart
    - Presents eight key weather metrics for May 2020, with Average Humidity showing the highest value at around 40 units
    - Shows the relationship between day/night duration and temperature metrics, with average temperature sitting at approximately 25 units while day duration is about 15 units

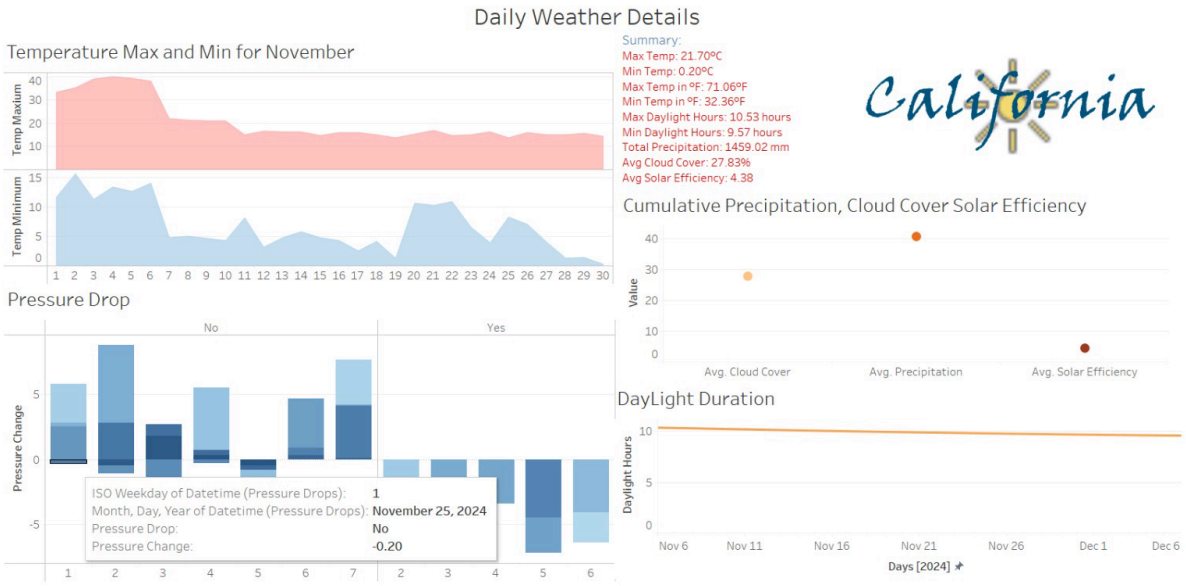


Figure 20: Recent data analysis dashboard

The dashboard displays current weather data for California with four key visualizations:

1. Temperature Max and Min for November
    - Shows daily temperature variations with maximum temperatures (pink area) ranging from 40°C to 15°C, with a notable drop after day 7

- Minimum temperatures (blue area) fluctuate between 0-15°C, showing several peaks around days 3-5 and 20-22

2. Pressure Drop Analysis
    - Displays pressure changes categorized as "Yes" and "No", with the largest negative pressure change of -0.20 recorded on November 25, 2024
    - Shows more frequent pressure variations in the first week of the month, with smaller fluctuations in subsequent weeks
3. Cumulative Precipitation, Cloud Cover & Solar Efficiency
    - Reveals current measurements with cloud cover at 27.83%, total precipitation at 1459.02mm, and solar efficiency at 4.38
    - Demonstrates the relationship between these metrics with precipitation showing the highest value among the three parameters
4. Daylight Duration Trend
    - Shows a gradual decrease in daylight hours from 10.53 (maximum) to 9.57 (minimum) hours from early November to early December
    - Displays a consistent downward trend in daylight hours, typical of the approaching winter season

VIII. Results and Discussion

The implemented pipeline successfully processes daily weather data, calculates key metrics, and presents them in an easily digestible format. The use of Airflow ensures reliable scheduling and execution of both ETL and ELT processes, while dbt enables version-controlled, modular data transformations.

Scalability Considerations:

- The pipeline can easily accommodate additional locations by adding them to the ETL process.
- Snowflake's scalability allows for efficient processing of large volumes of historical weather data.

Limitations and future improvements:

- The current implementation focuses on daily data. Incorporating hourly could provide more granular insights.
- Additional weather metrics and predictive models could be incorporated to provide more comprehensive analysis and forecasting capabilities.

IX. GitHub repository link

https://github.com/patilgayatri22/DataWarehouse_FinalProject/tree/main/DataWarehouseProject

X. *Conclusion*

This project demonstrates the successful integration of modern data tools to create an end-to-end analytics pipeline for weather data. By leveraging Snowflake, Airflow, dbt, and Tableau, we've created a scalable, maintainable solution that can provide valuable insights for weather analysis and its impact on various sectors.

The modular nature of the pipeline allows for easy expansion and modification, making it adaptable to changing requirements and additional data sources. This approach to data analytics can be applied to various domains beyond weather analysis, showcasing the versatility of the chosen tools and architecture.

Acknowledgement

The Data Warehouse project of building an end-to-end Weather Data Analytics using Snowflake, Airflow, dbt, Tableau was an enriching experience because of the guidance and support of professor Keeyong Hun and TA Revanth Kumar Bondada. We would like to extend our gratitude towards them for their insights and expertise which turned out to be instrumental in helping us navigate the complexities of the project. Thank you for fostering a collaborative and engaging learning environment that encouraged us to explore and innovate. We greatly appreciate your dedication and encouragement.

References

[1] Professor Keeyong Hun, DATA226 Lecture Notes Week 8, Week 9, Week 10, Week 11
[2] Visual Crossing Weather Data Services API as data source, https://www.visualcrossing.com/weather/weather-data-services