Forecast Blackout – Predicting Power Outages in California

Course No.: DATA226

Course Name: Data Warehouse and Pipeline

M.S Data Analytics

SAN JOSE STATE UNIVERSITY

SAN JOSE, CA

PROJECT REPORT

GROUP-4:

Aishwarya Iyer (018277948)

Arya Mehta (018292885)

Keith Rajesh Gonsalves (018326945)

Prajwal Dambalkar (018318196)

Pukhraj Rathkanthiwar (018274997)

May 2025

Acknowledgment

We would like to express our deepest gratitude to all those who supported and guided us throughout the course of this project.

Our sincere thanks go to Prof. Keeyong Han for his exceptional mentorship, insightful feedback, and constant encouragement. His expertise and support were instrumental in shaping the direction and success of this work.

We are also thankful to the Department of Applied Data Science for providing us with the academic resources, infrastructure, and collaborative environment that made this research possible.

We are especially grateful to our families and friends for their unwavering support, patience, and motivation during this journey. Their encouragement helped us stay focused and committed through every phase of the project.

This project has been a significant learning experience, offering us the opportunity to apply theoretical knowledge to a real-world problem while growing both individually and as a team. We are proud of what we have accomplished together and thankful for the opportunity to collaborate on work that aims to make a meaningful impact.

Table of Contents

Figure Table

Abstract

California power outages become increasingly disruptive with global warming and rising energy demands. This project tried to predict such outages by merging outage history information and real-time weather forecasts. With data engineering tools like Apache Airflow, dbt, and Snowflake, we developed a robust data pipeline to ingest, transform, and predict power outages. We also created interactive Tableau dashboards to see peak risk times and plan accordingly. Our pipeline offers data consistency, idempotency and automation to allow constant operation in a production grade system.

1. Introduction

California is experiencing routine power outages that threaten public health, safety, and economic well-being. Such outages are typically caused by natural causes such as storms and wildfires caused by adverse weather conditions. Surprisingly, not much predictive analytics is used to control outages as risks increase. This paper suggests an end-to-end pipeline that connects outage histories to real-time weather data to identify and forecast outage-prone areas. We leverage current tools like Airflow for orchestration, dbt for transformation, Snowflake as a data warehouse, and Tableau for business intelligence. Our project shows how public datasets can be operationalized to manage critical infrastructure.

2. Problem Statement

California lacks a good early-warning system for weather-related power outages. Outages cause loss of communication, damage to perishable goods, and disruption of services such as emergency response and education. Delayed mitigation action due to the lack of predictive information increases public risk. We aim to fill this gap by creating a data-driven forecasting system that integrates forecasted and historical weather data with historical patterns of outages.

3. Requirements and Specification

This project has several technical and operational requirements:

• Robust consumption of historical outage data from the state of California's open data repository.

• Open-Meteo weather API subscription for real-time and forecasted weather.

• Centralized, horizontally scalable data warehouse for storing structured data (Snowflake).

• Automated orchestration layer with Airflow for processing daily data fetch, transformation, and loading processes.

• SQL transformation layers with dbt using proper testing, documentation, and snapshots.

• Dashboarding software (Tableau) to display live outage risk analysis to stakeholders.

• GitHub version control for tracing development and change.

The system should further be modular, fault-tolerant, and schedulable in production.

4. System Architecture

There are five layers to the system: data source layer, extraction layer, transformation layer, storage layer, and visualization layer. At the source level, we gather power outage and hourly weather forecast data. This is pulled via Python scripts and Airflow DAGs. The transformation occurs via dbt with cleaned and joined tables for downstream forecasting. Forecasted data is stored in a Snowflake table and is then used to create Tableau dashboards.

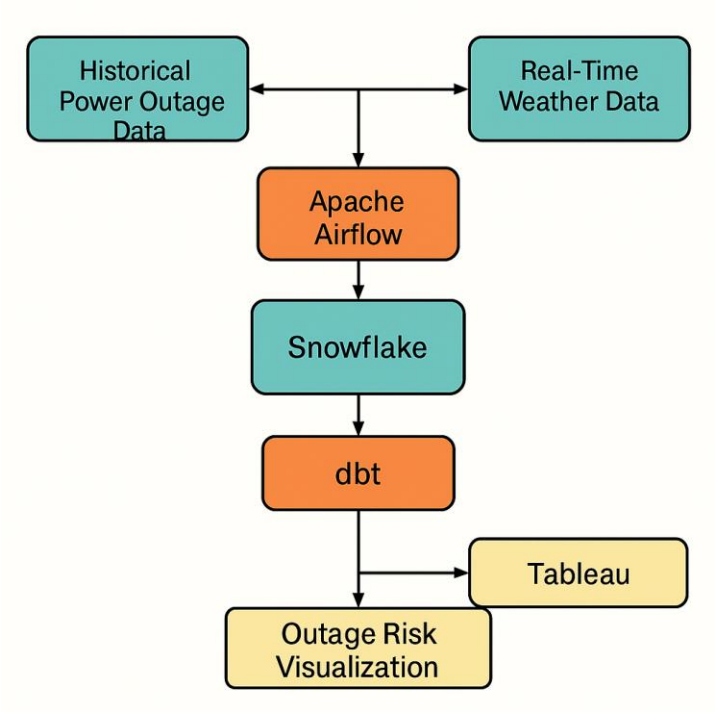


Fig 1: System Architecture

5. Database Table Structure

Every table within the project has a specific function in data analysis and storage:

1. raw.outage_data:

- incident_id (Primary Key), start_time, latitude, longitude, outage_type, cause
- Notes: All outages that have been reported; used as base table.

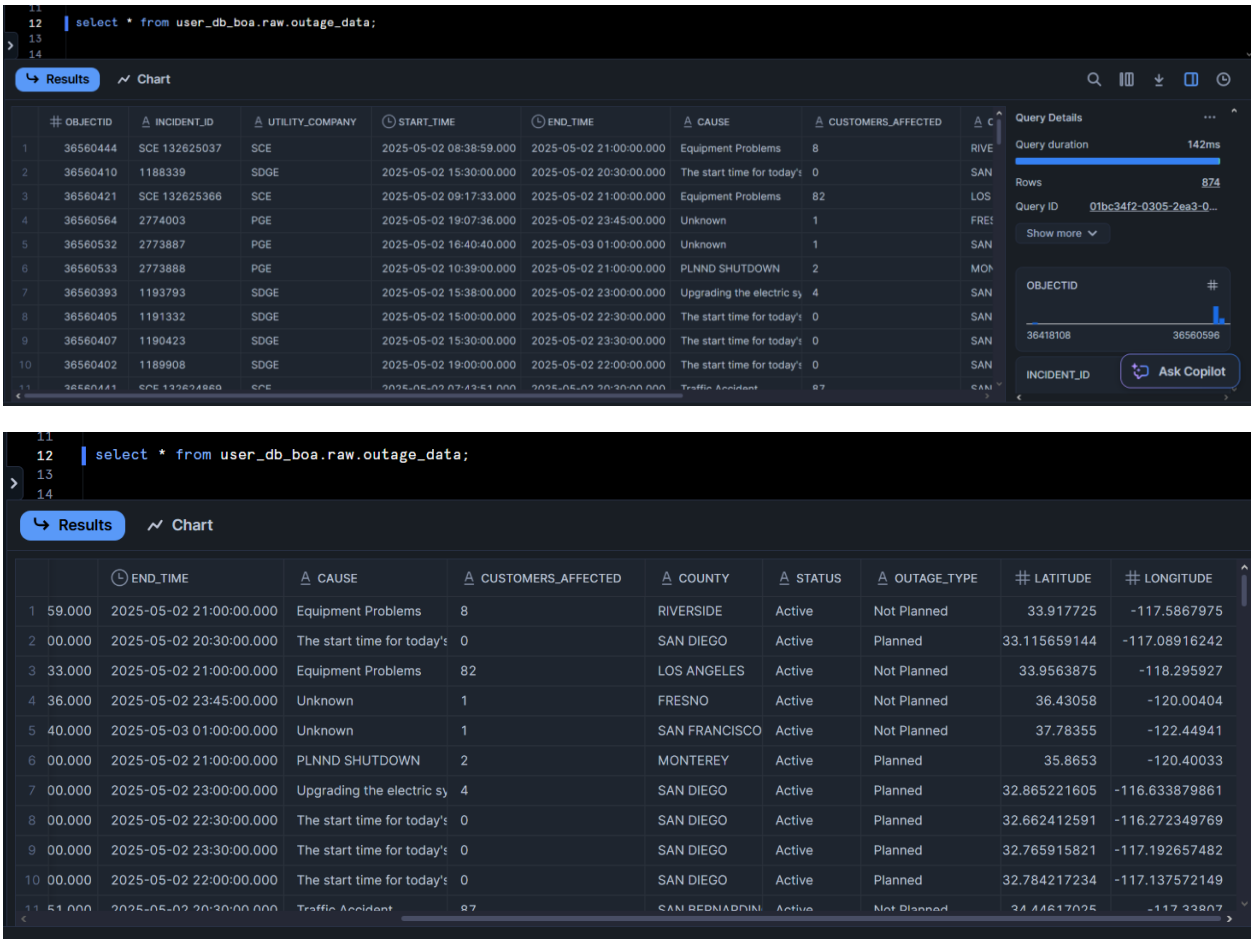
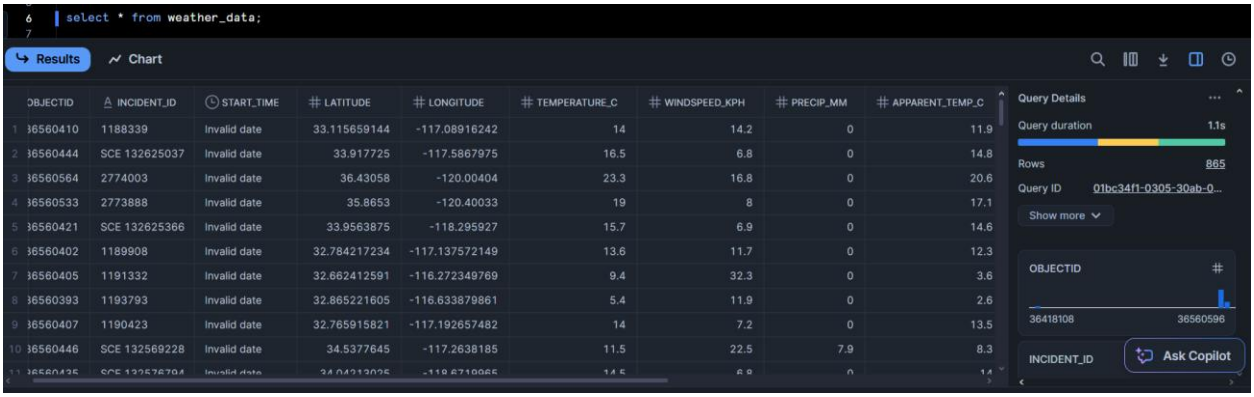


Fig 2: Outage table on snowflake

2. raw.weather_data:

- incident_id (Foreign Key), temperature_c, windspeed_kph, humidity_pct, pressure_hpa, timestamp

- Notes: Hourly weather readings by incident and location.



```
6  select * from weather_data;
7
```

Results | Chart

	TEMPERATURE_C	WINDSPEED_K	PRECIP_MM	APPARENT_TEMP_C	HUMIDITY_PCT	PRESSURE_HPA	WEATHER_SOURCE
1	14	14.2	0	11.9	75	984.7	open-meteo
2	16.5	6.8	0	14.8	56	986.5	open-meteo
3	23.3	16.8	0	20.6	38	998.5	open-meteo
4	19	8	0	17.1	46	953.6	open-meteo
5	15.7	6.9	0	14.6	68	1001.4	open-meteo
6	13.6	11.7	0	12.3	85	996.2	open-meteo
7	9.4	32.3	0	3.6	78	890.1	open-meteo
8	5.4	11.9	0	2.6	100	881	open-meteo
9	14	7.2	0	13.5	86	1007.2	open-meteo
10	11.5	22.5	7.9	8.3	91	906.9	open-meteo
11	14.5	6.8	0	14	83	994.8	open-meteo

Fig 4: Weather table on snowflake

3. int_outage_weather_joined:

- Combined table joining weather and outage information on incident_id + coordinates.

- Constraints: Does not include planned outages, performs timezone normalization.

```
22  select * from analytics.int_outage_weather_joined;
```

Results | Chart

	INCIDENT_ID	LATITUDE	LONGITUDE	OUTAGE_TYPE	START_TIME	COUNTY	UTILITY_COMPANY	CUSTOMERS_AFFECTED
1	SCE 132625037	33.917725	-117.5867975	Not Planned	2025-05-02 08:38:59.000	RIVERSIDE	SCE	8
2	SCE 132625366	33.9563875	-118.295927	Not Planned	2025-05-02 09:17:33.000	LOS ANGELES	SCE	82
3	2774003	36.43058	-120.00404	Not Planned	2025-05-02 19:07:36.000	FRESNO	PGE	1
4	2773887	37.78355	-122.44941	Not Planned	2025-05-02 16:40:40.000	SAN FRANCISCO	PGE	1
5	SCE 132624869	34.44617025	-117.33807	Not Planned	2025-05-02 07:43:51.000	SAN BERNARDIN	SCE	87
6	2774012	36.97135	-120.06677	Not Planned	2025-05-02 19:18:00.000	MADERA	PGE	203
7	2773775	35.33962	-119.25396	Not Planned	2025-05-02 15:35:05.000	KERN	PGE	1
8	2773950	36.76004	-119.71496	Not Planned	2025-05-02 17:53:39.000	FRESNO	PGE	1
9	2774021	37.97449	-122.30805	Not Planned	2025-05-02 19:30:59.000	CONTRA COSTA	PGE	1
10	2774039	38.11595	-121.35056	Not Planned	2025-05-02 19:52:08.000	SAN JOAQUIN	PGE	2
11	SCE 132625515	33.68857625	-117.855611	Not Planned	2025-05-02 09:46:41.000	ORANGE	SCE	91

```
select * from analytics.int_outage_weather_joined;
```

Results | Chart

_COMPANY	CUSTOMERS_AFFECTED	TEMPERATURE_C	WINDSPEED_KPH	PRECIP_MM	APPARENT_TEMP_C	HUMIDITY_PCT	PRESSURE_HPA
	8	23	20.8	0	20.1	45	991.5
	82	18.2	16.6	0	16.6	69	1007.2
	1	30.5	9.8	0	28.7	24	1002.7
	1	13.3	23.1	0	10	81	1002.2
	87	22.8	18.2	0	19.4	36	901.6
	203	30	13.5	0	28	27	1000.6
	1	29.5	13.7	0	27.5	30	998.5
	1	30.5	9.4	0	29.6	30	998.5
	1	16.7	20.9	0	14.2	67	1002.1
	2	28	21	0	25.4	33	1009.6
	91	19.6	13.5	0	18.2	60	1011.8

Fig 5: outage and weather joined table on snowflake

5. stg_outage_data

- Filtered out from the raw table to show unplanned outage types

```
select * from analytics.stg_outage_data;
```

Results | Chart

INCIDENT_ID	START_TIME	COUNTY	LATITUDE	LONGITUDE	UTILITY_COMPANY	CUSTOMERS_AFFECTED	OUTAGE_TYPE
SCE 132625037	2025-05-02 08:38:59.000	RIVERSIDE	33.917725	-117.5867975	SCE	8	Not Planned
SCE 132625366	2025-05-02 09:17:33.000	LOS ANGELES	33.9563875	-118.295927	SCE	82	Not Planned
2774003	2025-05-02 19:07:36.000	FRESNO	36.43058	-120.00404	PGE	1	Not Planned
2773887	2025-05-02 16:40:40.000	SAN FRANCISCO	37.78355	-122.44941	PGE	1	Not Planned
SCE 132624869	2025-05-02 07:43:51.000	SAN BERNARDIN	34.44617025	-117.33807	SCE	87	Not Planned
2774012	2025-05-02 19:18:00.000	MADERA	36.97135	-120.06677	PGE	203	Not Planned
2773775	2025-05-02 15:35:05.000	KERN	35.33962	-119.25396	PGE	1	Not Planned
2773950	2025-05-02 17:53:39.000	FRESNO	36.76004	-119.71496	PGE	1	Not Planned
2774021	2025-05-02 19:30:59.000	CONTRA COSTA	37.97449	-122.30805	PGE	1	Not Planned
2774039	2025-05-02 19:52:08.000	SAN JOAQUIN	38.11595	-121.35056	PGE	2	Not Planned
SCE 132625515	2025-05-02 09:46:41.000	ORANGE	33.68857625	-117.855611	SCE	91	Not Planned

Query Details
Query duration 746ms
Rows 484
Query ID 01bc34f5-0305-3178-0...
Show more
INCIDENT_ID
100% filled
START_TIME
Ask Copilot

Fig 6: outage view on snowflake

6. stg_weather_data

- Filtered out to extract the important weather features.

```
select * from analytics.stg_weather_data;
```

Results | Chart

INCIDENT_ID	LATITUDE	LONGITUDE	TEMPERATURE_C	WINDSPEED_KPH	PRECIP_MM	APPARENT_TEMP_C	HUMIDITY_PCT	PR
1188339	33.115659144	-117.08916242	14	14.2	0	11.9	75	
SCE 132625037	33.917725	-117.5867975	16.5	6.8	0	14.8	56	
2774003	36.43058	-120.00404	23.3	16.8	0	20.6	38	
2773888	35.8653	-120.40033	19	8	0	17.1	46	
SCE 132625366	33.9563875	-118.295927	15.7	6.9	0	14.6	68	
1189908	32.784217234	-117.137572149	13.6	11.7	0	12.3	85	
1191332	32.662412591	-116.272349769	9.4	32.3	0	3.6	78	
1193793	32.865221605	-116.633879861	5.4	11.9	0	2.6	100	
1190423	32.765915821	-117.192657482	14	7.2	0	13.5	86	
SCE 132569228	34.5377645	-117.2638185	11.5	22.5	7.9	8.3	91	
SCE 132576794	34.04213025	-118.6719965	14.5	6.8	0	14	83	

Query Details
Query duration 436ms
Rows 865
Query ID 01bc34f7-0305-2c67-0...
Show more
INCIDENT_ID
100% filled
LATITUDE
Ask Copilot

Fig 7: Weather view on snowflake

7. county_weather_forecasts:

- Has per-county, per-hour forecast with risk flags applied.

- Fields: county, timestamp, temperature_c, windspeed_kph, humidity_pct, pressure_hpa, outage_risk_flag.

```
17  select * from analytics.county_weather_forecasts;
18
19
```

Results | Chart

	ds	county	temperature_c	windspeed_kph	precip_mm	apparent_temp_c	humidity_pct	pressure_hpa	forecast_outage
1	2025-05-02 20:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
2	2025-05-02 21:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
3	2025-05-02 22:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
4	2025-05-02 23:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
5	2025-05-03 00:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
6	2025-05-03 01:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
7	2025-05-03 02:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
8	2025-05-03 03:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
9	2025-05-03 04:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
10	2025-05-03 05:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE
11	2025-05-03 06:00:00	ALAMEDA	19.2	18.2	0	17.3	61	1010.2	TRUE

Fig 8: Forecasted table on snowflake

6. Airflow Pipeline

Apache Airflow controls the pipeline of data ingestion and transformation. The Airflow DAG executes every day and comprises weather data retrieval, outage data retrieval, data transformation, and loading into Snowflake tasks. Tasks have been made modular and have been done as Airflow's @task decorator for legibility and testability. Connection and variables were utilized to safely store API keys and Snowflake credentials.

ETL Process

The ETL (Extract, Transform, Load) process begins with the extraction of two major sources: a historical dataset of power outages from California's public data portal and real-time weather data from the Open-Meteo API. Python scripts extract outage metadata (incident_id, start_time, location) and hourly weather measurements (temperature, wind speed, humidity).

Secondly, the data goes through Python and Pandas via unit conversion into standard values (Fahrenheit into Celsius), exclusion of null or missing values, and timestamp normalization. Finally, the clean data sets are merged with spatial-temporal keys and loaded into Snowflake for warehousing. This whole process is managed and scheduled daily using Apache Airflow so that there could be automation as well as reproducibility.

a. ETL Outage Dag

```
import uuid
```

```
from airflow import DAG # type: ignore
from airflow.decorators import task # type: ignore
from airflow.providers.snowflake.hooks.snowflake import SnowflakeHook # type: ignore
from datetime import datetime, timedelta
import json
import urllib.request
import pandas as pd # type: ignore
from snowflake.connector.pandas_tools import write_pandas # type: ignore
import requests # type: ignore
from pyproj import Transformer # type: ignore


# Get a Snowflake cursor
def return_snowflake_conn():
    hook = SnowflakeHook(snowflake_conn_id='snowflake_conn')  # Update conn_id if needed
    conn = hook.get_conn()
    return conn, conn.cursor()

# Extract Task
@task
def extract_power_outage_data():
    OUTAGE_URL = (
        "https://services.arcgis.com/BLN4oKB0N1YSgvY8/ArcGIS/"
        "rest/services/Power_Outages_%28View%29/FeatureServer/0/query"
    )

    params = {
        "where": "1=1",
        "outFields": "*",
        "returnGeometry": "true",
        "f": "json"
    }

    try:
        response = requests.get(OUTAGE_URL, params=params, timeout=30)
        response.raise_for_status()
        data = response.json()

        features = data.get("features", [])
        transformer = Transformer.from_crs("epsg:3857", "epsg:4326", always_xy=True)

        records = []
        for feature in features:
            record = feature.get("attributes", {})
            geometry = feature.get("geometry", {})

            # Convert x/y to lon/lat
            x = geometry.get("x") or record.get("x")
            y = geometry.get("y") or record.get("y")

            if x is not None and y is not None:
                try:
                    lon, lat = transformer.transform(x, y)
                    record["longitude"] = lon
                    record["latitude"] = lat
                except Exception as conv_err:
```

```
                print(f" ⚠️ Coordinate conversion failed for x={x}, y={y}: {conv_err}")
                record["longitude"] = None
                record["latitude"] = None
            else:
                record["longitude"] = None
                record["latitude"] = None

            records.append(record)

        print(f" ✅ Extracted {len(records)} records from ArcGIS API with lat/lon.")
        return records

    except Exception as e:
        print(" ❌ Failed to extract data:", e)
        return []

@task
def transform_data(records):
    if not records:
        print(" ⚠️ No records to transform.")
        return []

    df = pd.DataFrame(records)
    print(" 📌 Raw columns:", df.columns.tolist())

    # Define mapping from ArcGIS to internal schema
    column_mapping = {
        'OBJECTID': 'objectid',
        'IncidentId': 'incident_id',
        'UtilityCompany': 'utility_company',
        'StartDate': 'start_time',
        'EstimatedRestoreDate': 'end_time',
        'Cause': 'cause',
        'ImpactedCustomers': 'customers_affected',
        'County': 'county',
        'OutageStatus': 'status',
        'OutageType': 'outage_type',
        'longitude': 'longitude',
        'latitude': 'latitude'
    }

    df = df[[col for col in column_mapping if col in df.columns]]
    df = df.rename(columns=column_mapping)

    # Convert timestamps
    df['start_time'] = pd.to_datetime(df['start_time'], unit='ms')
    df['end_time'] = pd.to_datetime(df['end_time'], unit='ms')

    df['start_time'] = df['start_time'].dt.floor('S')
    df['end_time'] = df['end_time'].dt.floor('S')

    # Clean and fill null values
    string_cols = ['utility_company', 'cause', 'county', 'status']
    for col in string_cols:
        if col in df.columns:
```

```
            df[col] = df[col].fillna("Unknown")

    # Convert numeric columns safely
    df['customers_affected'] = pd.to_numeric(df['customers_affected'],
errors='coerce').fillna(0).astype(int)

    # Print example
    print(" ✅ Transformed sample record:\n", df.head(1).to_dict(orient='records'))

    # Convert datetime columns to string for XCom compatibility
    if 'start_time' in df.columns:
        df['start_time'] = df['start_time'].astype(str)
    if 'end_time' in df.columns:
        df['end_time'] = df['end_time'].astype(str)

    df.columns = [col.upper() for col in df.columns]  # for Snowflake compatibility
    print(df.head(1).to_dict(orient='records'))  # Print first record for debugging
    return df.to_dict(orient='records')

# Load Task
@task
def load_to_snowflake(records):
    if not records:
        print(" ⚠️ No records to load.")
        return

    print(" 🔍 Keys in first record:")
    print(records[0].keys() if records else "No records returned.")

    print(" 🔍 First record received for loading:")
    print(records[0] if records else "No records!")


    df = pd.DataFrame(records)

    conn, cursor = return_snowflake_conn()  # unified session
    print(" 🔗 Connected to Snowflake.")

    staging_table = "USER_DB_BOA.RAW.OUTAGE_STAGE"
    target_table = "USER_DB_BOA.RAW.OUTAGE_DATA"

    try:
        cursor.execute("BEGIN")

        # cursor.execute("CREATE DATABASE IF NOT EXISTS USER_DB_BOA;")

        # Use the database before creating schemas
        cursor.execute("USE DATABASE USER_DB_BOA;")

        cursor.execute("CREATE SCHEMA IF NOT EXISTS raw;")
        # cursor.execute("CREATE SCHEMA IF NOT EXISTS analytics;")

        # Step 1: Create staging table if not exists
        cursor.execute(f"""
            CREATE TABLE IF NOT EXISTS {staging_table} (
```

```
            OBJECTID            NUMBER,
            INCIDENT_ID         STRING,
            UTILITY_COMPANY     STRING,
            START_TIME          DATETIME,
            END_TIME            DATETIME,
            CAUSE               STRING,
            CUSTOMERS_AFFECTED  STRING,
            COUNTY              STRING,
            STATUS              STRING,
            OUTAGE_TYPE         STRING,
            LATITUDE            FLOAT,
            LONGITUDE           FLOAT
        );
    """)

    # Step 2: Truncate before reload
    cursor.execute(f"TRUNCATE TABLE {staging_table}")

    # Step 3: Load into staging
    # Fix datetime issues
    for time_col in ['START_TIME', 'END_TIME']:
        if time_col in df.columns:
            df[time_col] = pd.to_datetime(df[time_col], errors='coerce')
            df[time_col] = df[time_col].dt.strftime('%Y-%m-%d %H:%M:%S')  # convert to
proper string
            df[time_col] = df[time_col].where(df[time_col].notnull(), None)

    success, nchunks, nrows, _ = write_pandas(conn, df, table_name=staging_table.split('.')[-1],
schema="RAW")
    print(f"✅ Loaded {nrows} records into staging table.")

    # Step 4: Create target if not exists
    cursor.execute(f"""
        CREATE TABLE IF NOT EXISTS {target_table} (
            OBJECTID            NUMBER PRIMARY KEY,
            INCIDENT_ID         STRING,
            UTILITY_COMPANY     STRING,
            START_TIME          DATETIME,
            END_TIME            DATETIME,
            CAUSE               STRING,
            CUSTOMERS_AFFECTED  STRING,
            COUNTY              STRING,
            STATUS              STRING,
            OUTAGE_TYPE         STRING,
            LATITUDE            FLOAT,
            LONGITUDE           FLOAT
        );
    """)

    # Step 5: MERGE into final table
    cursor.execute(f"""
        MERGE INTO {target_table} tgt
        USING {staging_table} src
        ON tgt.OBJECTID = src.OBJECTID
        WHEN MATCHED THEN UPDATE SET
            INCIDENT_ID = src.INCIDENT_ID,
```

```
                UTILITY_COMPANY = src.UTILITY_COMPANY,
                START_TIME = src.START_TIME,
                END_TIME = src.END_TIME,
                CAUSE = src.CAUSE,
                CUSTOMERS_AFFECTED = src.CUSTOMERS_AFFECTED,
                COUNTY = src.COUNTY,
                STATUS = src.STATUS,
                OUTAGE_TYPE = src.OUTAGE_TYPE,
                LATITUDE = src.LATITUDE,
                LONGITUDE = src.LONGITUDE
            WHEN NOT MATCHED THEN INSERT (
                OBJECTID, INCIDENT_ID, UTILITY_COMPANY,
                START_TIME, END_TIME, CAUSE, CUSTOMERS_AFFECTED,
                COUNTY, STATUS, OUTAGE_TYPE, LATITUDE, LONGITUDE
            )
            VALUES (
                src.OBJECTID, src.INCIDENT_ID, src.UTILITY_COMPANY,
                src.START_TIME, src.END_TIME, src.CAUSE, src.CUSTOMERS_AFFECTED,
                src.COUNTY, src.STATUS, src.OUTAGE_TYPE, src.LATITUDE, src.LONGITUDE
            );
        """)

        cursor.execute("COMMIT")
        print(f"✅ Merge complete. {nrows} records upserted into final table.")

    except Exception as e:
        cursor.execute("ROLLBACK")
        print("❌ Merge failed:", e)
        raise e
    finally:
        cursor.close()

# DAG Definition
with DAG(
    dag_id='California_Power_Outage_ETL_PowerOutage',
    start_date=datetime(2025, 4, 28),
    schedule_interval='*/15 * * * *',  # Every 15 minutes
    catchup=False,
    tags=['ETL', 'PowerOutage']
) as dag:

    raw_data = extract_power_outage_data()
    transformed_data = transform_data(raw_data)
    load_to_snowflake(transformed_data)
```

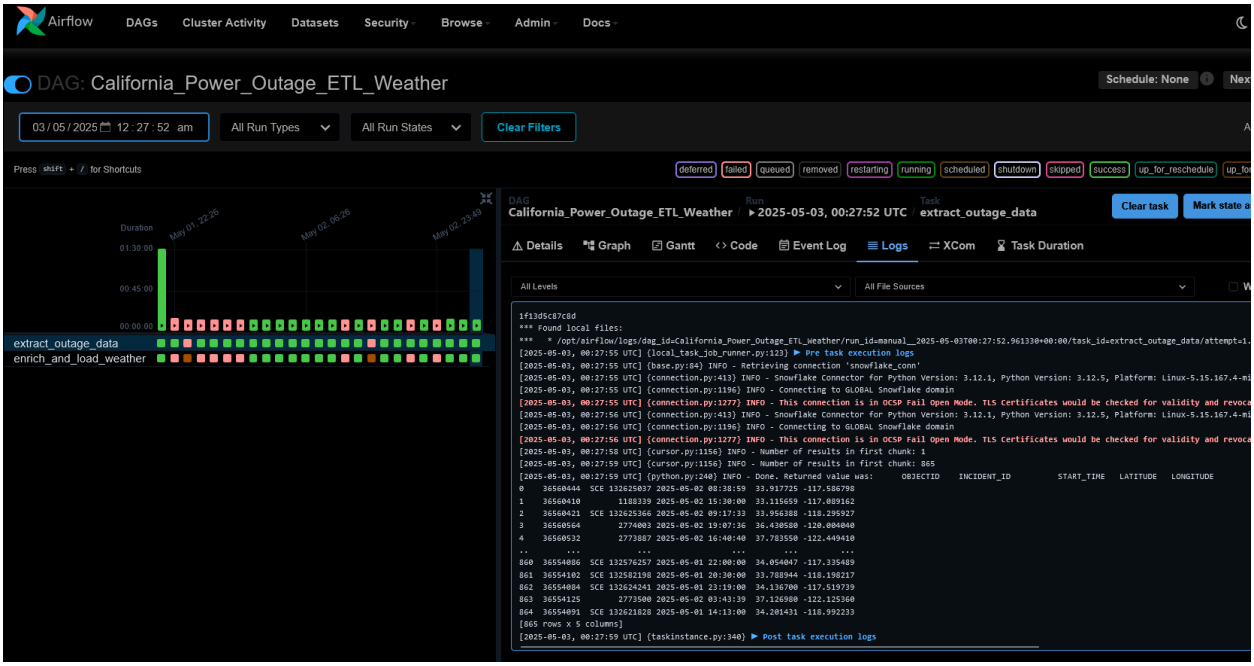


Fig 9: Airflow UI for outage extract logs

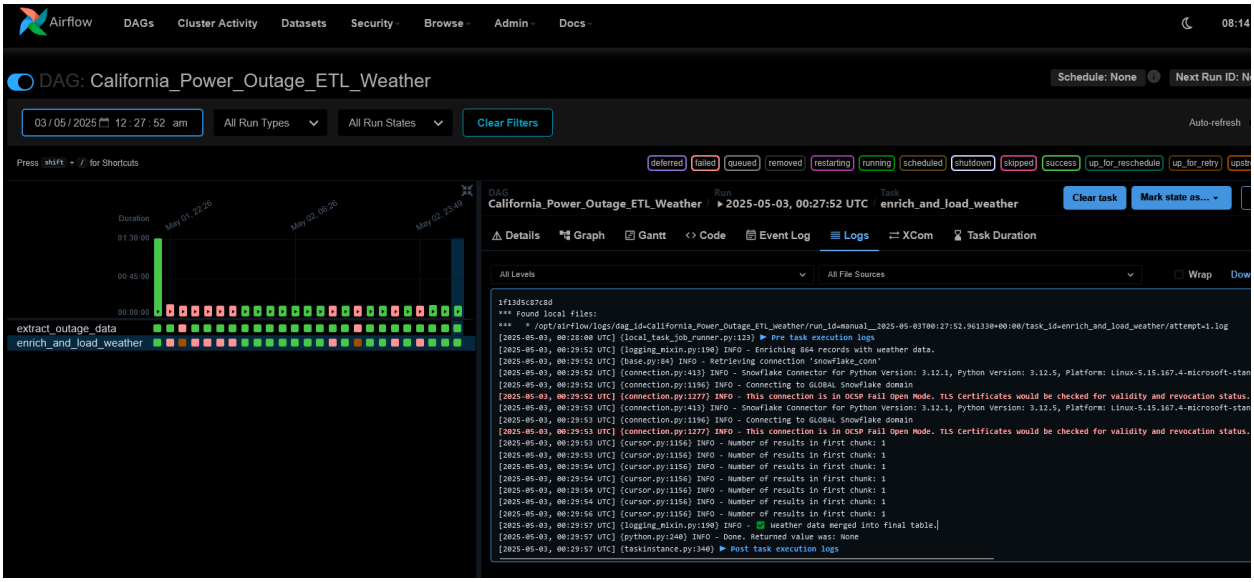


Fig 10: Airflow UI outage load logs

b. ETL Weather Dag

```
from airflow import DAG
from airflow.decorators import task
from airflow.utils.dates import days_ago
from airflow.providers.snowflake.hooks.snowflake import SnowflakeHook
from snowflake.connector.pandas_tools import write_pandas
```

```
import requests
import pandas as pd
from datetime import datetime, timedelta
from concurrent.futures import ThreadPoolExecutor, as_completed
import time

SNOWFLAKE_CONN_ID = "snowflake_conn"
SF_DATABASE = "USER_DB_CHIPMUNK"
SF_SCHEMA = "RAW"
SF_WEATHER_TABLE = "WEATHER_DATA"

OPEN_METEO_BASE_URL = "https://api.open-meteo.com/v1/forecast"


def get_snowflake_cursor():
    hook = SnowflakeHook(snowflake_conn_id=SNOWFLAKE_CONN_ID)
    return hook.get_conn(), hook.get_conn().cursor()
def fetch_weather(lat, lon, dt):
    if pd.isnull(dt):
        return None

    if isinstance(dt, pd.Timestamp):
        dt = dt.to_pydatetime()

    dt = dt.replace(microsecond=0)  # remove microseconds

    dt_hour = dt.strftime("%Y-%m-%dT%H:%M:%S")  # ISO format preferred by many APIs

    params = {
        "latitude": lat,
        "longitude": lon,
        "hourly":
"temperature_2m,apparent_temperature,relative_humidity_2m,surface_pressure,windspeed_10
m,precipitation",
        "start": dt_hour,
        "end": dt_hour,
        "timezone": "UTC"
    }

    headers = {"User-Agent": "weather-data-pipeline/1.0"}

    for attempt in range(3):
        try:
            response = requests.get(OPEN_METEO_BASE_URL, params=params,
headers=headers, timeout=10)
            if response.status_code == 200:
                data = response.json()
                if "hourly" in data and data["hourly"].get("time"):
                    return {
                        "temperature": data["hourly"]["temperature_2m"][0],
                        "apparent_temp": data["hourly"]["apparent_temperature"][0],
                        "humidity": data["hourly"]["relative_humidity_2m"][0],
                        "pressure": data["hourly"]["surface_pressure"][0],
                        "windspeed": data["hourly"]["windspeed_10m"][0],
                        "precip": data["hourly"]["precipitation"][0],
                    }
```

```
            time.sleep(1)
        except Exception as e:
            time.sleep(2)

    return None

#           INCIDENT_ID     STRING,
def create_weather_table_if_needed(cursor):
    cursor.execute(f"""
        CREATE TABLE IF NOT EXISTS {SF_DATABASE}.{SF_SCHEMA}.{SF_WEATHER_TABLE} (
            OBJECTID            NUMBER PRIMARY KEY,
            INCIDENT_ID         STRING,
            START_TIME          DATETIME,
            LATITUDE            FLOAT,
            LONGITUDE           FLOAT,
            TEMPERATURE_C       FLOAT,
            APPARENT_TEMP_C     FLOAT,
            HUMIDITY_PERCENT    FLOAT,
            PRESSURE_HPA        FLOAT,
            WINDSPEED_KPH       FLOAT,
            PRECIP_MM           FLOAT,
            WEATHER_SOURCE      STRING,
            ENRICHED_AT         DATETIME
        );
    """)

def query_outages():
    conn, cursor = get_snowflake_cursor()

    # Step 1: Check if weather_data table exists
    check_table_sql = f"""
        SELECT COUNT(*)
        FROM {SF_DATABASE}.INFORMATION_SCHEMA.TABLES
        WHERE TABLE_NAME = 'WEATHER_DATA' AND TABLE_SCHEMA = '{SF_SCHEMA}';
    """
    cursor.execute(check_table_sql)
    table_exists = cursor.fetchone()[0] > 0

    # Step 2: Write conditional query based on existence
    if table_exists:
        query = f"""
            SELECT OBJECTID, INCIDENT_ID,
                START_TIME, LATITUDE, LONGITUDE
            FROM {SF_DATABASE}.{SF_SCHEMA}.OUTAGE_DATA
            WHERE LATITUDE IS NOT NULL
              AND LONGITUDE IS NOT NULL
              AND START_TIME IS NOT NULL
              AND OBJECTID NOT IN (
                SELECT OBJECTID FROM {SF_DATABASE}.{SF_SCHEMA}.WEATHER_DATA
              );
        """
    else:
        query = f"""
```

```
            SELECT OBJECTID, INCIDENT_ID,
                START_TIME, LATITUDE, LONGITUDE
            FROM {SF_DATABASE}.{SF_SCHEMA}.OUTAGE_DATA
            WHERE LATITUDE IS NOT NULL
              AND LONGITUDE IS NOT NULL
              AND START_TIME IS NOT NULL;
        """

    # Step 3: Execute and return results
    cursor.execute(query)
    rows = cursor.fetchall()
    columns = [desc[0] for desc in cursor.description]
    return pd.DataFrame(rows, columns=columns)


def load_weather_to_snowflake(df):
    conn, cursor = get_snowflake_cursor()

    # Ensure schema exists
    cursor.execute(f"USE DATABASE {SF_DATABASE}")
    cursor.execute(f"CREATE SCHEMA IF NOT EXISTS {SF_SCHEMA}")

    # Create main + staging tables
    create_weather_table_if_needed(cursor)
    cursor.execute(f"""
        CREATE TABLE IF NOT EXISTS
{SF_DATABASE}.{SF_SCHEMA}.WEATHER_STAGE LIKE
{SF_DATABASE}.{SF_SCHEMA}.{SF_WEATHER_TABLE};
    """)

    cursor.execute(f"TRUNCATE TABLE
{SF_DATABASE}.{SF_SCHEMA}.WEATHER_STAGE")

    # # Ensure proper datetime format
    # df["START_TIME"] = pd.to_datetime(df["START_TIME"], errors="coerce")
    # df["ENRICHED_AT"] = pd.to_datetime(df["ENRICHED_AT"], errors="coerce")

    # print(df[['START_TIME', 'ENRICHED_AT']].dtypes)
    # print(df[['START_TIME', 'ENRICHED_AT']].head())

    # Load into staging
    write_pandas(
        conn,
        df,
        table_name="WEATHER_STAGE",
        schema=SF_SCHEMA
    )

    # MERGE from staging to weather table
    cursor.execute(f"""
        MERGE INTO {SF_DATABASE}.{SF_SCHEMA}.{SF_WEATHER_TABLE} tgt
        USING {SF_DATABASE}.{SF_SCHEMA}.WEATHER_STAGE src
        ON tgt.OBJECTID = src.OBJECTID
        WHEN MATCHED THEN UPDATE SET
            INCIDENT_ID     = src.INCIDENT_ID,
            START_TIME      = src.START_TIME,
```

```
            LATITUDE       = src.LATITUDE,
            LONGITUDE      = src.LONGITUDE,
            TEMPERATURE_C  = src.TEMPERATURE_C,
            APPARENT_TEMP_C = src.APPARENT_TEMP_C,
            HUMIDITY_PERCENT= src.HUMIDITY_PERCENT,
            PRESSURE_HPA   = src.PRESSURE_HPA,
            WINDSPEED_KPH  = src.WINDSPEED_KPH,
            PRECIP_MM      = src.PRECIP_MM,
            WEATHER_SOURCE  = src.WEATHER_SOURCE,
            ENRICHED_AT    = src.ENRICHED_AT
        WHEN NOT MATCHED THEN INSERT (
            OBJECTID, INCIDENT_ID, START_TIME, LATITUDE, LONGITUDE,
            TEMPERATURE_C,          APPARENT_TEMP_C,          HUMIDITY_PERCENT,
PRESSURE_HPA,
            WINDSPEED_KPH, PRECIP_MM,
            WEATHER_SOURCE, ENRICHED_AT
        )
        VALUES (
            src.OBJECTID,       src.INCIDENT_ID,       src.START_TIME,       src.LATITUDE,
src.LONGITUDE,
            src.TEMPERATURE_C,    src.APPARENT_TEMP_C,    src.HUMIDITY_PERCENT,
src.PRESSURE_HPA,
            src.WINDSPEED_KPH, src.PRECIP_MM,
            src.WEATHER_SOURCE, src.ENRICHED_AT
        );
    """)

    cursor.close()
    print("✅ Weather data merged into final table.")


def enrich_weather_records(df):
    df = df[df['START_TIME'].notna()]  #       Skip null start times
    enriched = []

    def enrich_row(row):
        weather = fetch_weather(row['LATITUDE'], row['LONGITUDE'], row['START_TIME'])
        if weather:
            return {
                "OBJECTID": row['OBJECTID'],
                "INCIDENT_ID": row['INCIDENT_ID'],
                "START_TIME":              row['START_TIME'].replace(microsecond=0)              if
pd.notnull(row['START_TIME']) else None,
                "LATITUDE": row['LATITUDE'],
                "LONGITUDE": row['LONGITUDE'],
                "TEMPERATURE_C": weather["temperature"],
                "APPARENT_TEMP_C": weather["apparent_temp"],
                "HUMIDITY_PERCENT": weather["humidity"],
                "PRESSURE_HPA": weather["pressure"],
                "WINDSPEED_KPH": weather["windspeed"],
                "PRECIP_MM": weather["precip"],
                "WEATHER_SOURCE": "open-meteo",
                "ENRICHED_AT": datetime.utcnow().replace(microsecond=0)
            }
        return None
```

```
    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = {executor.submit(enrich_row, row): row for _, row in df.iterrows()}
        for future in as_completed(futures):
            result = future.result()
            if result:
                enriched.append(result)

    return pd.DataFrame(enriched)


default_args = {"owner": "airflow", "retries": 1}

with DAG(
    dag_id="California_Power_Outage_ETL_Weather",
    default_args=default_args,
    schedule_interval=None,
    start_date=days_ago(1),
    catchup=False,
    tags=["weather", "enrichment"]
) as dag:

    @task()
    def extract_outage_data():
        return query_outages()

    @task()
    def enrich_and_load_weather(outages_df):
        df = enrich_weather_records(outages_df)
        if df.empty:
            print("No weather data to load.")
        if not df.empty:
            print(f"Enriching {len(df)} records with weather data.")
            load_weather_to_snowflake(df)

    outages = extract_outage_data()
    enrich_and_load_weather(outages)
```

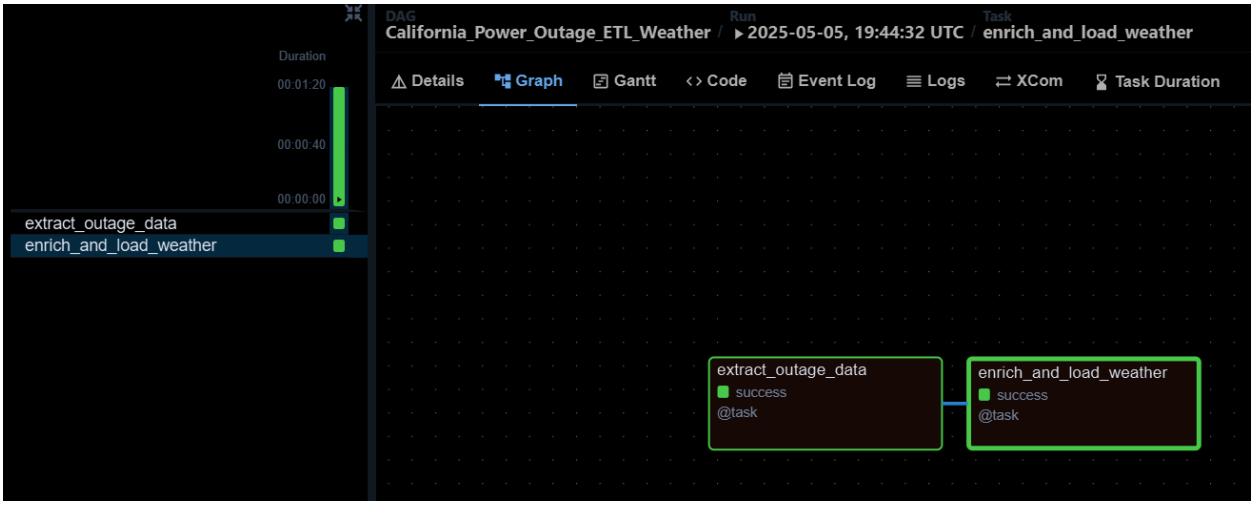


Fig 11: Airflow UI for weather dag

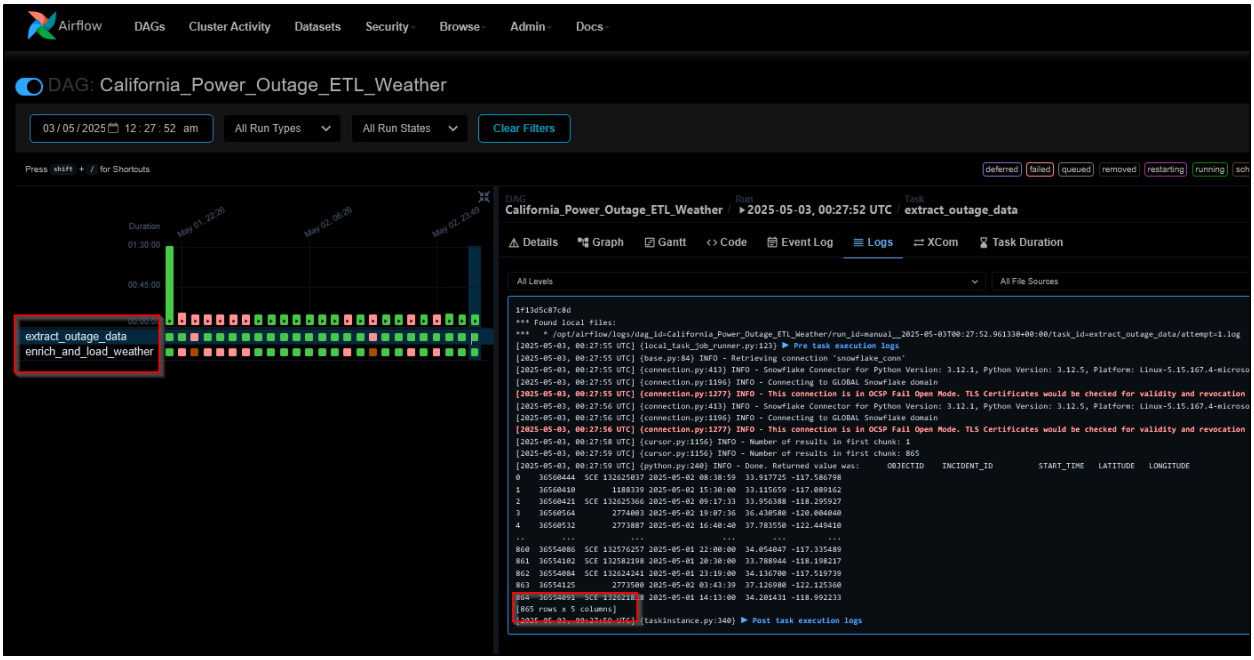


Fig 12: Airflow UI logs for weather dag

ELT Process

In the ELT (Extract, Load, Transform) process, raw data is first loaded into Snowflake staging tables. dbt (data build tool) is used to perform SQL-based transformations within the warehouse.

stg_outage_data and stg_weather_data are staging models for raw tables.

int_outage_weather_joined is the intermediate model that joins the staging models on keys incident_id, latitude, and longitude. It also filters out planned outages and unnecessary columns.

The final model is implemented as a table and tested using dbt tests for nulls and referential integrity.

This modular strategy allows for easy maintenance and testing with high data quality and schema consistency.

c. ELT Merged table Dag

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from airflow.utils.dates import days_ago
from airflow.models import Variable

default_args = {
   'owner': 'airflow',
   'start_date': days_ago(1),
   'retries': 1,
}

with DAG(
```

```
    dag_id='ELT_Outage_Weather_Join',
    default_args=default_args,
    schedule_interval='@daily',
    catchup=False,
    tags=['dbt', 'elt'],
) as dag:

    run_dbt = BashOperator(
        task_id='run_dbt_transformations',
        bash_command=(
            'export PATH="$PATH:/home/airflow/.local/bin" && '
            'cd "$DBT_PROJECT_DIR" && '
                    'dbt run --profiles-dir . --models stg_outage_data stg_weather_data
int_outage_weather_joined && '
            'dbt test --profiles-dir .'
        ),
        env={
            'DBT_PROJECT_DIR': '/opt/airflow/dags/forecastblackout',
            'DBT_ACCOUNT': 'sfedu02-ksb65579',
            'DBT_USER': Variable.get("snowflake_username"),
            'DBT_PASSWORD': Variable.get("snowflake_password"),
            'DBT_DATABASE': 'USER_DB_CHIPMUNK',
            'DBT_SCHEMA': 'analytics',
            'DBT_ROLE': 'TRAINING_ROLE',
            'DBT_WAREHOUSE': 'CHIPMUNK_QUERY_WH',
        },
    )
```

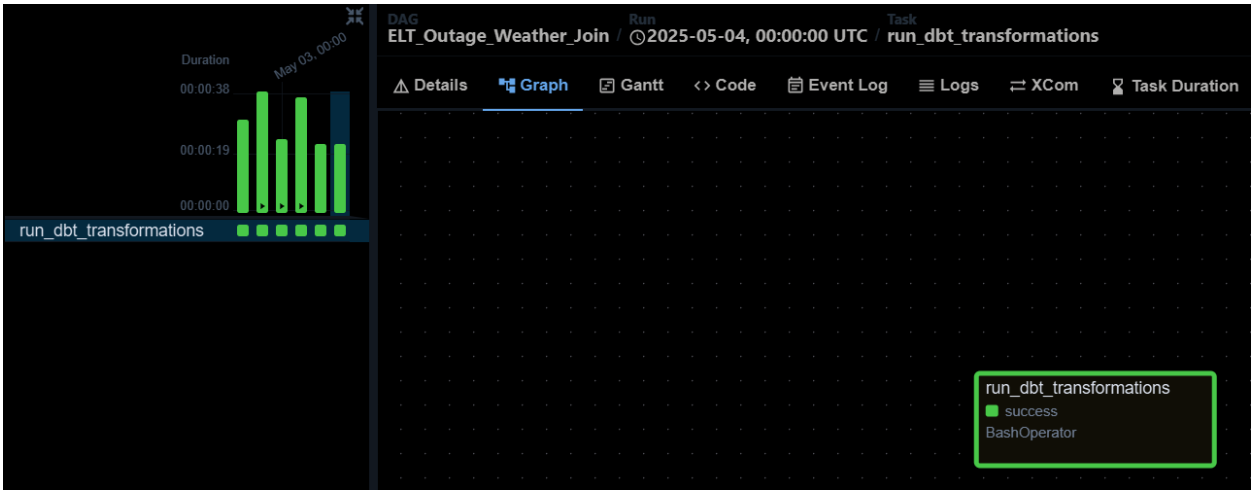


Fig 13: Airflow UI graph for final merged table

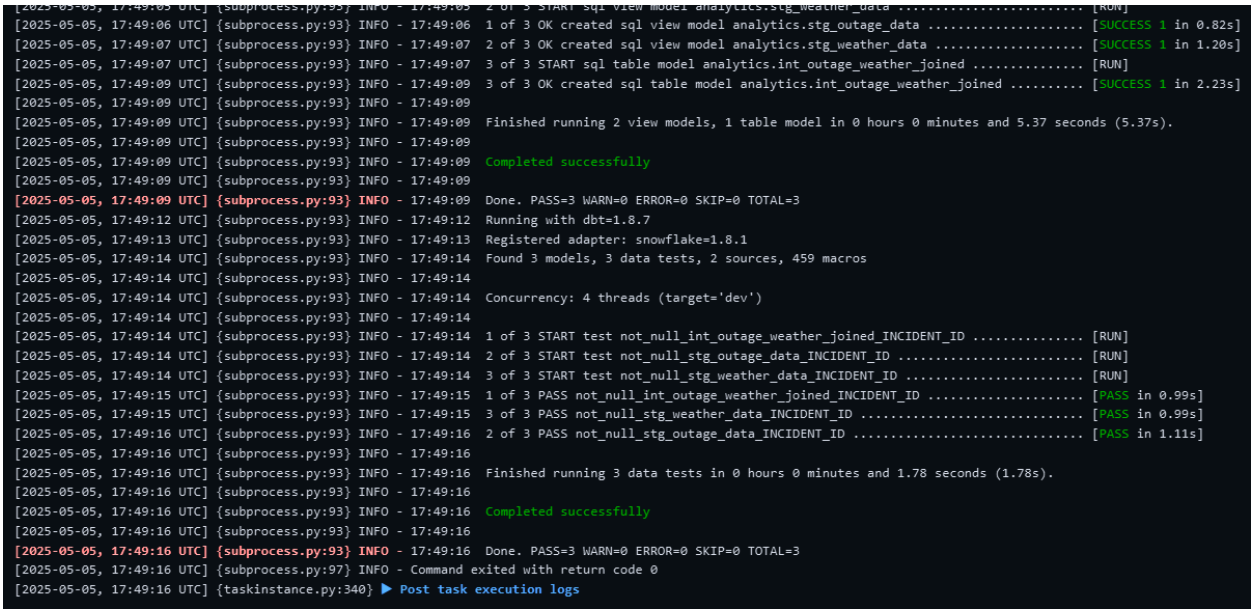


Fig 14: Airflow UI logs for final merged table

---

d. Model/stg_outage_data.sql

```
with base as (
    select
        INCIDENT_ID,
        START_TIME,
        COUNTY,
        LATITUDE,
        LONGITUDE,
        UTILITY_COMPANY,
        CUSTOMERS_AFFECTED,
        OUTAGE_TYPE,
        STATUS
    from {{ source('user_db_boa', 'outage_data') }}
    where OUTAGE_TYPE != 'Planned'
)

select * from base
```

---

e. Model/stg_weather_data.sql

```
with base as (
    select
        INCIDENT_ID,
        LATITUDE,
        LONGITUDE,
        TEMPERATURE_C,
        WINDSPEED_KPH,
```

```
        PRECIP_MM,
        APPARENT_TEMP_C,
        HUMIDITY_PCT,
        PRESSURE_HPA
    from {{ source('user_db_chipmunk', 'weather_data') }}
)

select * from base
```

---

f. Model/int_outage_weather_joined.sql

```
select
    o.INCIDENT_ID,
    o.LATITUDE,
    o.LONGITUDE,
    o.OUTAGE_TYPE,
    o.START_TIME,
    o.COUNTY,
    o.UTILITY_COMPANY,
    o.CUSTOMERS_AFFECTED,
    w.TEMPERATURE_C,
    w.WINDSPEED_KPH,
    w.PRECIP_MM,
    w.APPARENT_TEMP_C,
    w.HUMIDITY_PCT,
    w.PRESSURE_HPA
from {{ ref('stg_outage_data') }} o
left join {{ ref('stg_weather_data') }} w
  on o.INCIDENT_ID = w.INCIDENT_ID
```

---

g. Schema.yml

```
version: 2

sources:
  - name: user_db_boa
    schema: RAW
    database: USER_DB_BOA
    tables:
      - name: outage_data
        description: "Raw outage data from Snowflake"

  - name: user_db_chipmunk
```

```
    schema: RAW
    database: USER_DB_CHIPMUNK
    tables:
      - name: weather_data
        description: "Raw weather data from Snowflake"

models:
  - name: stg_outage_data
    description: "Staging model for outage data from USER_DB_BOA.RAW.OUTAGE_DATA"
    columns:
      - name: INCIDENT_ID
        description: "Unique identifier for each outage event"
        tests:
          - not_null
          #- unique

      - name: OUTAGE_TYPE
        description: "Type of the outage (Planned or Unplanned)"
      - name: START_TIME
        description: "Outage start timestamp"

  - name: stg_weather_data
                description: "Staging model for weather data from
USER_DB_CHIPMUNK.RAW.WEATHER_DATA"
    columns:
      - name: INCIDENT_ID
        description: "Foreign key to the outage event"
        tests:
          - not_null

      - name: TEMPERATURE_C
        description: "Temperature in Celsius"
      - name: WINDSPEED_KPH
        description: "Wind speed in kilometers per hour"
      - name: PRECIP_MM
        description: "Precipitation in millimeters"
      - name: APPARENT_TEMP_C
        description: "Apparent temperature in Celsius"
      - name: HUMIDITY_PCT
        description: "Relative humidity in percent"
      - name: PRESSURE_HPA
        description: "Atmospheric pressure in hectopascals"

  - name: int_outage_weather_joined
    description: "Joined and cleaned dataset of outages and weather conditions"
    columns:
      - name: INCIDENT_ID
        description: "Primary key for joined data"
        tests:
          - not_null
```

```
      - name: LAT
        description: "Latitude for the event"
      - name: LON
        description: "Longitude for the event"
      - name: TEMPERATURE_C
        description: "Current or forecasted temperature at outage location"
```

---

h. Dbt_project.yml

```
# Name your project! Project names should contain only lowercase characters
# and underscores. A good package name should reflect your organization's
# name or the intended use of these models
name: 'forecastblackout'
version: '1.0.0'

# This setting configures which "profile" dbt uses for this project.
profile: 'forecastblackout'

# These configurations specify where dbt should look for different types of files.
# The `model-paths` config, for example, states that models in this project can be
# found in the "models/" directory. You probably won't need to change these!
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

clean-targets:         # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"

# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models

# In this example config, we tell dbt to build all models in the example/
# directory as views. These settings can be overridden in the individual model
# files using the `{{ config(...) }}` macro.
models:
  forecastblackout:
    staging:
      materialized: view
    intermediate:
      materialized: table
```

Forecasting Process

The forecasting module uses the cleaned and merged dataset (int_outage_weather_joined) to make future outage predictions:

Extraction & Preprocessing: Snowflake is loaded into a Pandas DataFrame. Timestamps are read in, and data are grouped by county and resampled hourly. Missing values are interpolated.

Forecasting: Persistence model used—using the most recent value for each county and extrapolating ahead 24 hours.

Risk Calculation: Threshold rules are applied to signal potential outage risks. For example:

- Wind speed > 35 kph
- Temperature < 0°C or > 30°C
- Humidity < 30% or > 70%
- Pressure outside 1000–1010 hPa range

Loading Results: The final forecast with risk flags is written back to a Snowflake table (COUNTY_WEATHER_FORECASTS) for Tableau visualization.

i. Forecasting DAG

```
from airflow import DAG
from airflow.decorators import task
from airflow.providers.snowflake.hooks.snowflake import SnowflakeHook
from datetime import datetime, timedelta
import pandas as pd
from snowflake.connector.pandas_tools import write_pandas

with DAG(
    dag_id="county_weather_persistence_forecast",
    start_date=datetime(2025, 5, 2),
    schedule_interval="@daily",
    catchup=False,
    default_args={
        "retries": 3,
        "retry_delay": timedelta(minutes=2),
    },
    tags=["weather", "persistence", "forecast"],
) as dag:

    @task
    def extract_weather_data() -> pd.DataFrame:
        """Extract historical weather data from Snowflake"""
        try:
            hook = SnowflakeHook(snowflake_conn_id="Snowflake_Connect")
            sql = """
                SELECT
                  START_TIME as ds,
                  COUNTY,
                  TEMPERATURE_C,
```

```
                WINDSPEED_KPH,
                PRECIP_MM,
                APPARENT_TEMP_C,
                HUMIDITY_PCT,
                PRESSURE_HPA
            FROM user_db_chipmunk.analytics.int_outage_weather_joined
            WHERE START_TIME IS NOT NULL
        """
        df = hook.get_pandas_df(sql)
        df.columns = df.columns.str.lower()
        df['ds'] = pd.to_datetime(df['ds'], errors='coerce')
        return df.dropna(subset=['ds'])
    except Exception as e:
        raise ValueError(f"Data extraction failed: {str(e)}")

@task
def preprocess_data(df: pd.DataFrame) -> dict:
    """Prepare data for persistence model"""
    try:
        county_data = {}
        for county, group in df.groupby('county'):
            # Resample to hourly and forward fill
            resampled = (
                group.set_index('ds')
                .resample('1H')
                .first()
                .ffill()
            )
            county_data[county] = resampled.reset_index()
        return county_data
    except Exception as e:
        raise ValueError(f"Preprocessing failed: {str(e)}")

@task
def generate_persistence_forecast(county_data: dict) -> pd.DataFrame:
    """Generate forecasts using persistence model (last observation carried forward)"""
    try:
        all_forecasts = []
        forecast_hours = 24  # 24-hour forecast

        for county, data in county_data.items():
            if data.empty:
                continue

            last_record = data.iloc[-1]
            forecast_dates = pd.date_range(
                start=last_record['ds'] + timedelta(hours=1),
                periods=forecast_hours,
                freq='H'
            )
```

```
                forecast_df = pd.DataFrame({
                    'ds': forecast_dates,
                    'county': county,
                    'temperature_c': last_record['temperature_c'],
                    'windspeed_kph': last_record['windspeed_kph'],
                    'precip_mm': last_record['precip_mm'],
                    'apparent_temp_c': last_record['apparent_temp_c'],
                    'humidity_pct': last_record['humidity_pct'],
                    'pressure_hpa': last_record['pressure_hpa']
                })
                all_forecasts.append(forecast_df)

            return pd.concat(all_forecasts, ignore_index=True)
        except Exception as e:
            raise ValueError(f"Forecast generation failed: {str(e)}")

    @task
    def calculate_outage_risk(forecast_df: pd.DataFrame) -> pd.DataFrame:
        """Calculate outage risk flags"""
        try:
            forecast_df['forecast_outage'] = (
                (forecast_df['windspeed_kph'] > 35) |
                (forecast_df['precip_mm'] > 0.5) |
                ((forecast_df['temperature_c'] < 0) | (forecast_df['temperature_c'] > 30)) |
                ((forecast_df['humidity_pct'] < 30) | (forecast_df['humidity_pct'] > 70)) |
                ((forecast_df['pressure_hpa'] < 1000) | (forecast_df['pressure_hpa'] > 1010))
            )
            return forecast_df
        except Exception as e:
            raise ValueError(f"Risk calculation failed: {str(e)}")

    @task
    def load_results(df: pd.DataFrame):
        """Load results to Snowflake"""
        try:
            # Convert datetime to string for Snowflake
            df['ds'] = df['ds'].astype(str)

            hook = SnowflakeHook(snowflake_conn_id="Snowflake_Connect")
            conn = hook.get_conn()

            write_pandas(
                conn,
                df,
                table_name="COUNTY_WEATHER_FORECASTS",
                schema="ANALYTICS",
                database="USER_DB_CHIPMUNK",
                auto_create_table=True,
                overwrite=True
            )
        except Exception as e:
```

```
        raise ValueError(f"Data load failed: {str(e)}")

    # DAG workflow
    raw_data = extract_weather_data()
    processed_data = preprocess_data(raw_data)
    forecasts = generate_persistence_forecast(processed_data)
    risk_assessment = calculate_outage_risk(forecasts)
    load_results(risk_assessment)
```

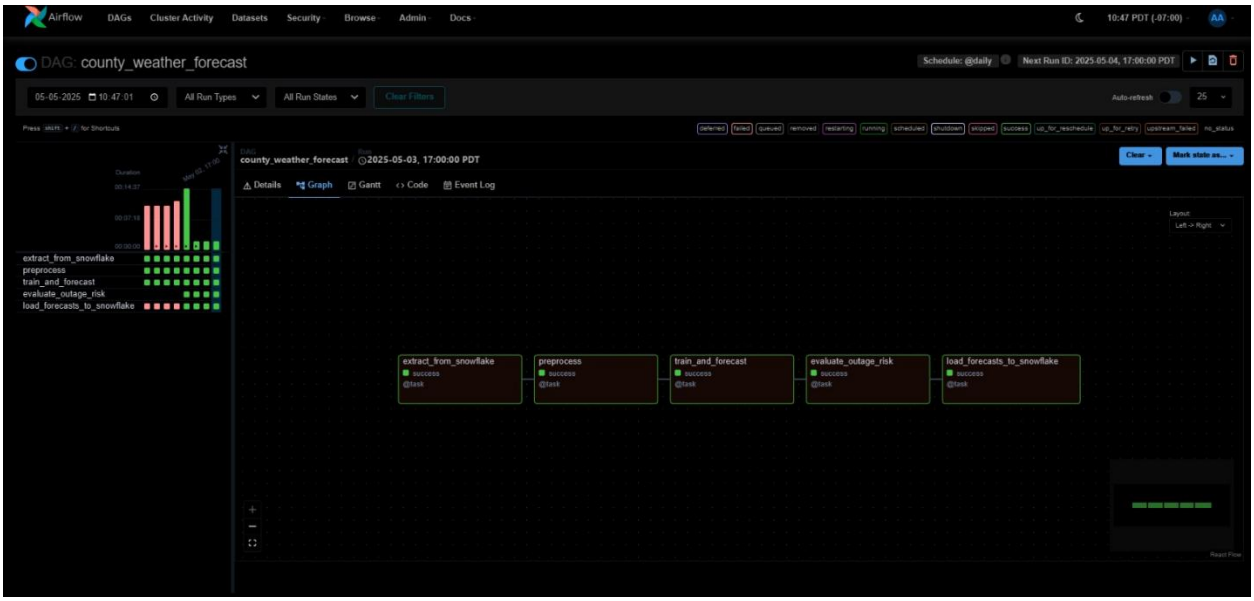


Fig 15: Airflow UI graph for Forecasting dag

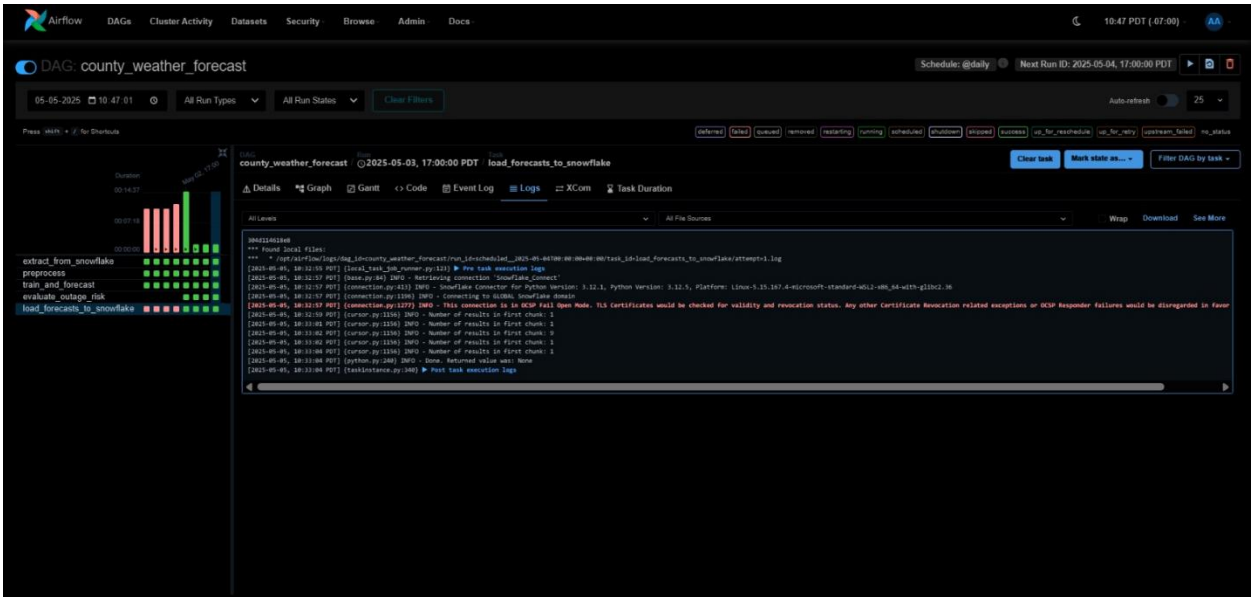


Fig 16: Airflow UI logs for Forecasting

7. Tableau Dashboard

The Forecast Blackout dashboard suite is designed to produce an in-depth and interactive summary of the California weather-condition power-outage relationship. The dashboards are intended to be put into use within utility providers, emergency management organizations, and policy planners to track and respond to outage risks triggered by weather anomalies ahead of time.

Dashboard 1: Outage Forecast Overview

This dashboard shows an executive summary of the forecasted power outage situations across California:

Leading metrics display highest temperature and wind speed during outages.

Total Forecasted Outage Hours: Time series graph of the way forecasted outage hours are trending over time.

Most Affected Counties: Horizontal bar graph of counties forecasted to have most outage hours.

Weather Factor Impact: The bar graph quantifies the relative contribution of various weather factors—pressure, humidity, temperature, and wind—to outage frequency so that environmental monitoring can be optimally prioritized.

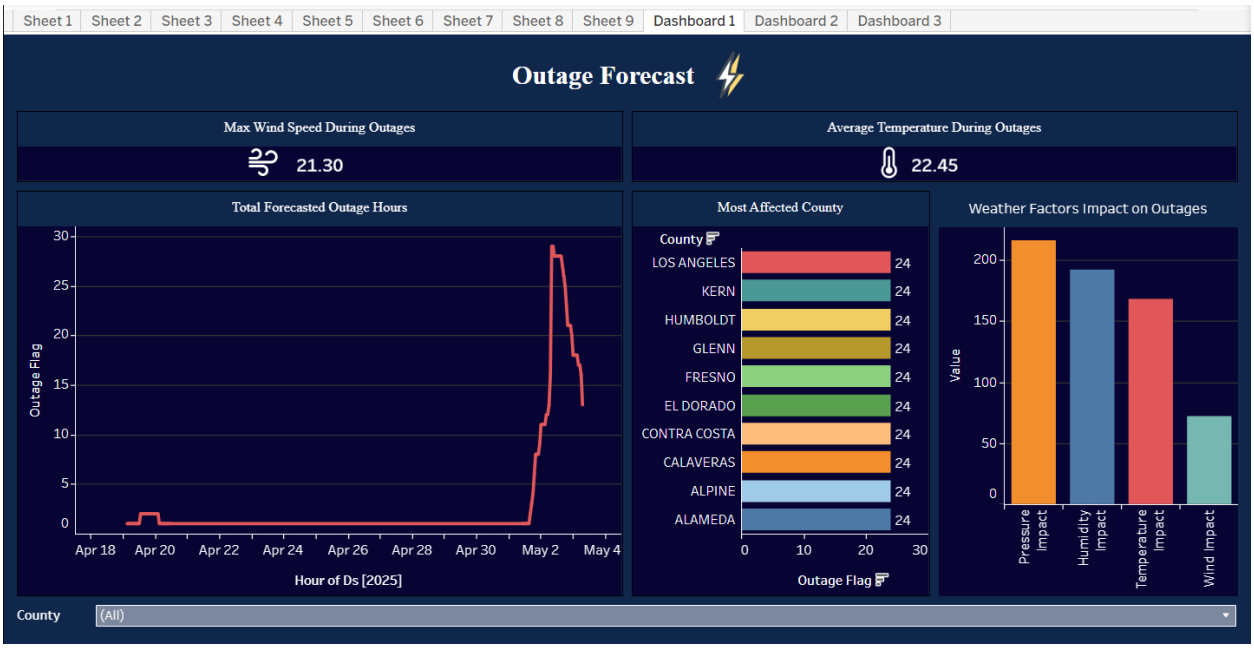


Fig 17: Dashboard showing overall outage forecast, most affected counties, and impact of weather factors on outages.

Dashboard 2: Weather-Outage Correlation (Part 1)

This dashboard investigates wind speed and temperature's correlation with power outages over time:

Wind Speed vs Outage: Plots average wind speed (line chart) against outage flag counts, allowing visualization of at which wind speeds outages typically happen.

Temperature vs Outage: Displays the effect of average temperature on the frequency of outages, giving insight into potential thermal thresholds that increase grid exposure.

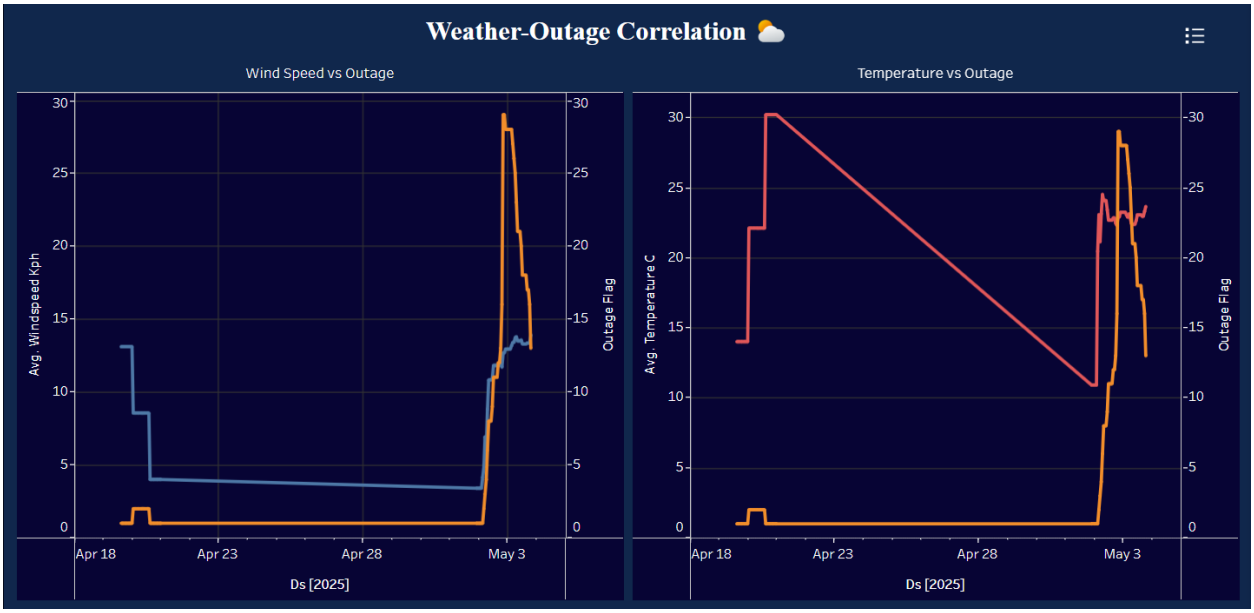


Fig 18: Dashboard showing correlation between wind speed, temperature, and outage occurrences over time.

Dashboard 3: Weather-Outages Correlation (Part 2)

This section continues analyzing correlation of weather with weather, atmospheric pressure, and humidity:

Pressure vs Outage: Shows a trend of mean pressure levels over time versus outage flags. It shows low-pressure systems, which are typically associated with storms.

Humidity vs Outage: Shows the correlation of relative humidity and power outages. Very low and very high humidity can stress the power lines and transformers.

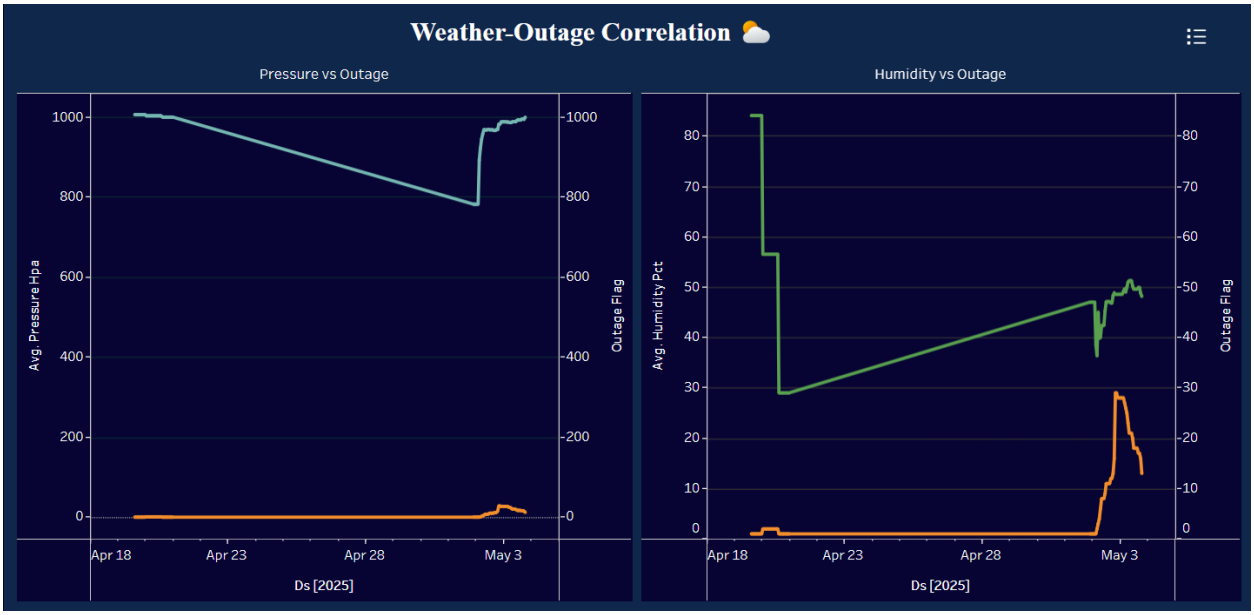


Fig 19: Dashboard showing correlation between atmospheric pressure, humidity, and outage frequency.

8. Github Repository

All code is maintained in a public GitHub repository at: https://github.com/puks0618/DW-Project

The structure includes:

• ETL/: Python ETL DAG files

• ELT/: Python ELT DAG file and DBT files

• Forecast/: Python Forecast DAG file

• tableau_exports/: Dashboards and images

9. Conclusion

We developed and deployed a stable pipeline that predicts power outage risk in California from live weather data. Supported by orchestration using Apache Airflow, transformation with dbt, Snowflake as the data warehouse, and Tableau for visualization, the project is a blueprint for data-led disaster preparedness. The pipeline is production-ready and has reliability maintained through transaction-based loading and modular design.

10. Future Work

Although this project represents a basic solution, a number of extensions can enhance its usefulness:

• Replace rule-based forecasting with machine learning models.

• Offer support for SMS/email alerting through Airflow alerts.

• Add more detailed location data (grid block or zipcode).

• Create Tableau dashboards for mobile optimization.

• Continuously ingest and label future outages for training.

11. References

[1] Open-Meteo API. https://open-meteo.com/

[2] Power Outage Dataset. https://data.ca.gov/dataset/power-outage-incidents

[3] Apache Airflow. https://airflow.apache.org/

[4] dbt Docs. https://docs.getdbt.com/

[5] Snowflake. https://www.snowflake.com/

[6] Tableau. https://www.tableau.com/