GRADED ASSESSMENT DATA PIPELINE PROJECT REPORT: WEEK 3

Report on Data Pipeline Project

Faisal Elawar
Faysal.elwar@lau.edu

Contents

Step1 Extracting the Data from GitHub

Extracted the Data from

https://raw.githubusercontent.com/falawar7/AAI_634O/refs/heads/main/Week3/Project/sales_data_FEB2025.csv

named the DF as sales data

sales_data.head()

	date	product_id	sales_amount	store_location
0	2/1/2025	P003	450	Los Angeles
1	2/1/2025	P002	950	New York
2	2/1/2025	P002	950	Houston
3	2/2/2025	P001	150	Seattle
4	2/2/2025	P004	600	New York

sales_data.info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63 entries, 0 to 62
Data columns (total 4 columns):
 #  Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0  date          63 non-null     object
 1  product_id    63 non-null     object
 2  sales_amount  63 non-null     int64
 3  store_location  63 non-null    object
dtypes: int64(1), object(3)
memory usage: 2.1+ KB
```

Step 2: Fetch Weather Data from the API

set the OpenWeatherMap API to fetch weather data for each store location on the corresponding transaction date. API Setup:

- api: "https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}
- api_key:3cb535b45b20509643dcb1f06587f284
- check one of the City as a sample Data

```
# API KEY
api_key="3cb535b45b20509643dcb1f06587f284"
#cehcking Temp , Humididty , weather condition for New York
temp, humidity, description = fetch_weather_data('New York', '2025-01-01', api_key)
print(f"Temp: {temp}, Humidity: {humidity}, Weather: {description}")
```

```
Temp: 1.0300000000000296, Humidity: 87, Weather: broken clouds
```

Step 3: Combine Weather Data with Sales Data

- Loop through each row of the sales_data dataframe,
- call the function and update the dataframe with weather data
- This method will add Temp: converted from Fahrenheit to Celsius , humidity , weather description from API weather fetching the data to store location within each date

```
def fetch_weather_data(city, date, api_key):
    base_url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}"
    response = requests.get(base_url)
    #print(response)
    data = response.json()
    #print(data)
    # Extract temperature, humidity, and weather description
    temperature = data['main']['temp'] - 273.15 # Convert from Kelvin to Celsius
    humidity = data['main']['humidity']
    weather_description = data['weather'][0]['description']
    return temperature, humidity, weather_description
```

- used the code as below:

```
# Loop through each row of the sales_data dataframe, call the function and update the dataframe with weather data
for index, row in sales_data.iterrows():
    temp, humidity, description = fetch_weather_data(row["store_location"], row["date"],api_key)
    sales_data.at[index, "Temperature (°C)"] = temp
    sales_data.at[index, "Humidity (%)"] = humidity
    sales_data.at[index, "Weather Description"] = description
```

Step 4: Load the Integrated Data into MongoDB

Load the Data to MongoDB as test to verify the records imported

```
[33]: #adding the records
sales_data_dict = sales_data.to_dict("records")
collection.insert_many(sales_data_dict)

#print the total added records
print(f"Total records added: {len(sales_data_dict)}")
```

```
Total records added: 63
```

dataengineeringcluster.61mrj.mongodb.net > sales_db > sales_weather_test

Open MongoDB shell

Documents 63 | Aggregations | Schema | Indexes 1 | Validation

Your pipeline is currently empty. Need help getting started? Generate aggregation

Explain | Export | Run | Options

Untitled | SAVE | + CREATE NEW | EXPORT TO LANGUAGE | PREVIEW | STAGES | TEXT | WIZARD

63 Documents in the collection

Preview of documents

```
_id: ObjectId('67a8c195f715e8c89b53a403')
date : "2/1/2025"
product_id : "P003"
sales_amount : 450
store_location : "Los Angeles"
Temperature (°C) : 12.400000000000034
Humidity (%) : 30
Weather Description : "broken clouds"
```

```
_id: ObjectId('67a8c195f715e8c89b53a404')
date : "2/1/2025"
product_id : "P002"
sales_amount : 950
store_location : "New York"
Temperature (°C) : 1.0300000000000296
Humidity (%) : 87
Weather Description : "broken clouds"
```

```
_id: ObjectId('67a8c195f715e8c89b53a405')
date : "2/1/2025"
product_id : "P002"
sales_amount : 950
store_location : "Houston"
Temperature (°C) : 21.170000000000016
Humidity (%) : 83
Weather Description : "overcast clouds"
```

The Complete ETL Process

Here is a more detailed summary of the steps involved in the ETL pipeline:

1. **Extract (extract_data):**

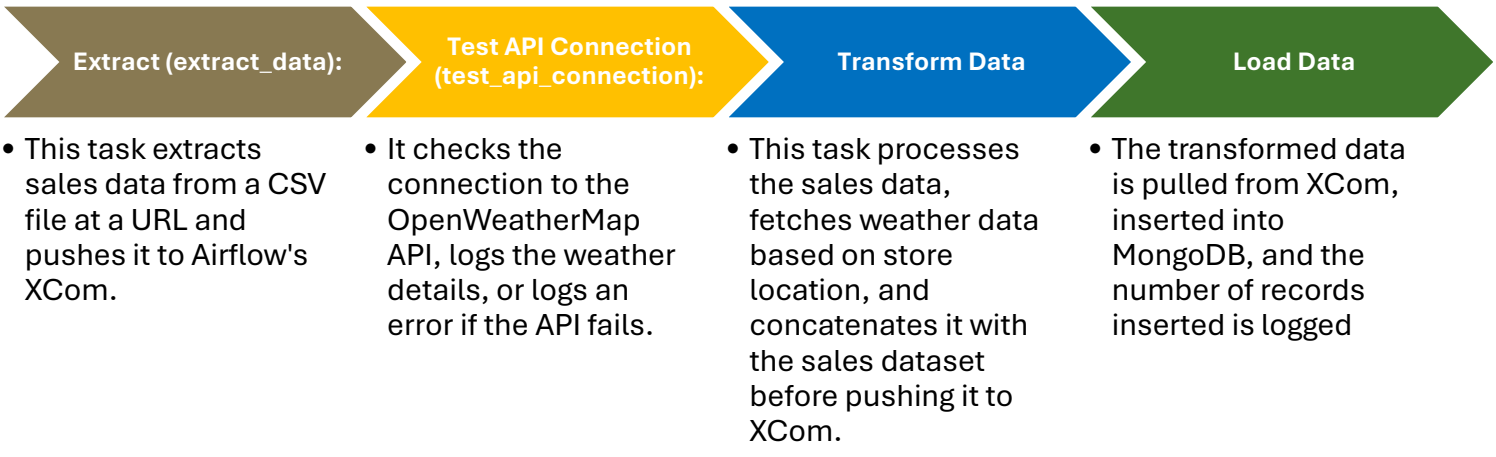
   - This task reads the sales data from a CSV file located at some URL. Sales data includes but is not limited to store locations and sales figures.
   - The data gets pushed to Airflow's XCom so it can be accessed by subsequent tasks.

2. **Test API Connection (test_api_connection):**
   - his task checks the connection to the OpenWeatherMap API. It will make a call to retrieve weather data for some sample city, for instance, New York.
   - Once the connection is received, it logs the temperature, humidity, and description of the weather. In case of API failure, it logs an error.

3. **Transform (transform_data):**
   - This task processes sales data pulled from XCom.
   - It fetches weather data for each store, namely temperature, humidity, and description of the weather, according to its location, which is based on the city.
   - It then concatenates the weather data as new columns to the sales dataset and pushes the transformed data to XCom for use in the final task.

4. **Load (load_data):**
   - It pulls the transformed sales data, now enriched with weather details, from XCom.
   - The task connects to MongoDB and inserts data into a collection, sales_weather, within the sales_db database.
   - It logs the number of records that were successfully inserted into MongoDB.

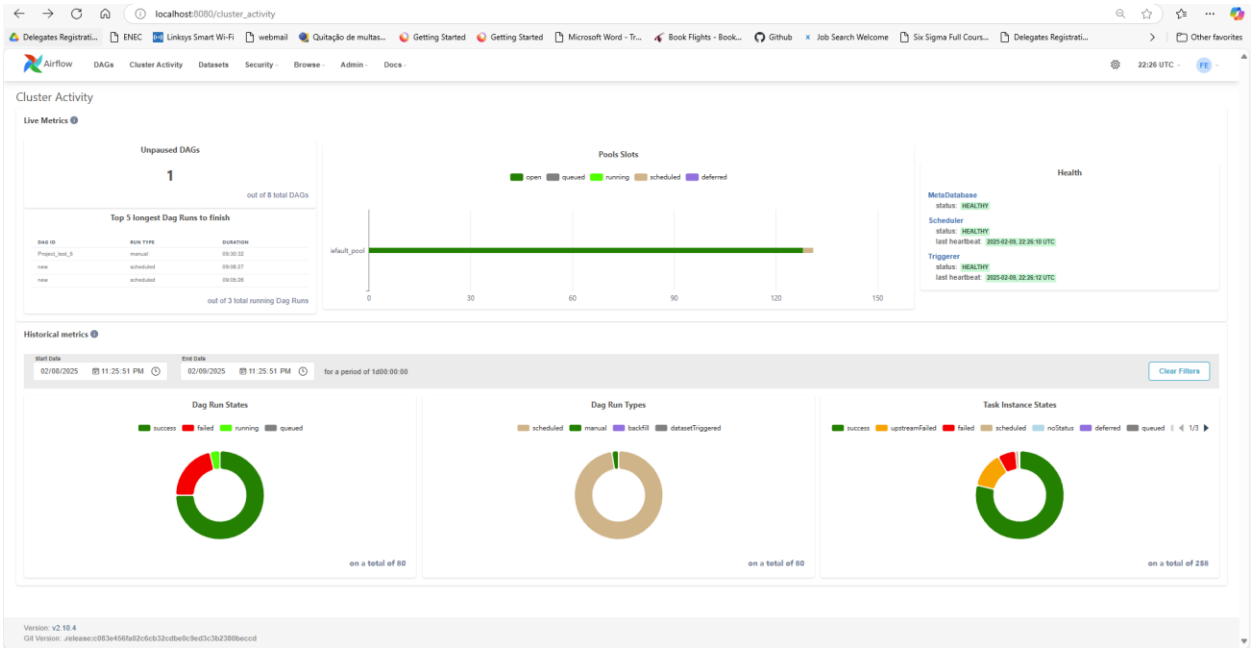
Figure of the ETL Tasks



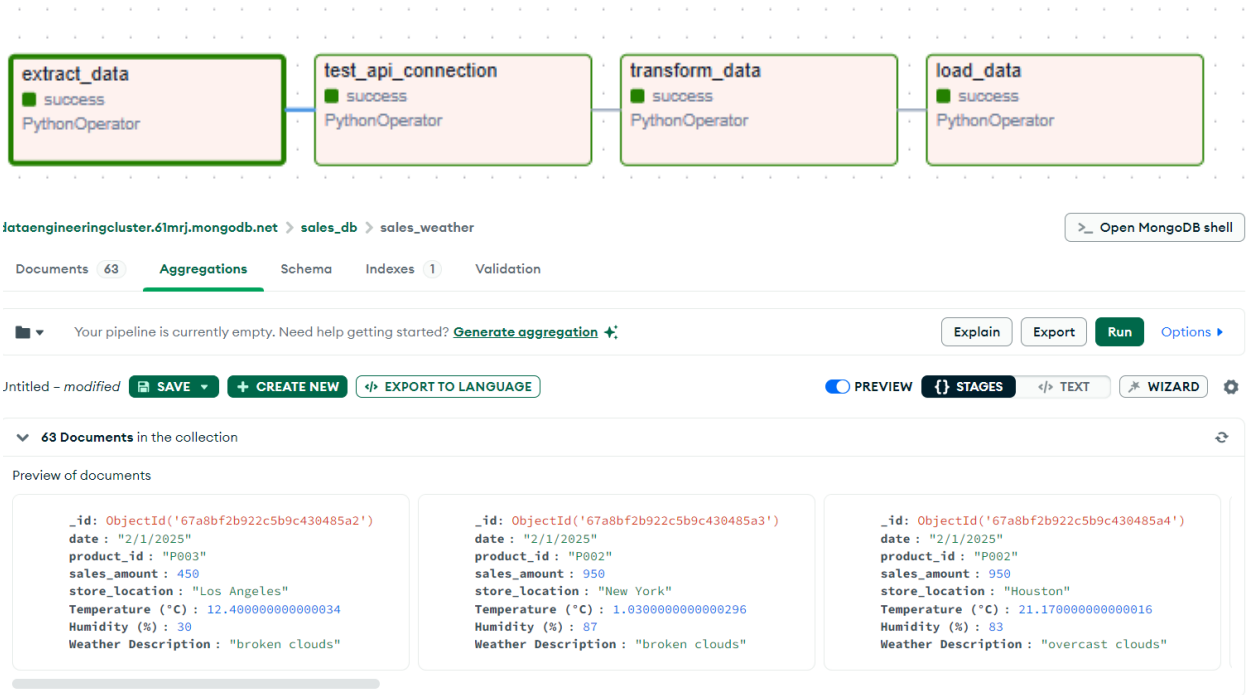
DAG is scheduled to run daily at 6:00 AM starting on 08 FEB 2025 for One DAY

It ensures that the entire ETL process-extraction, testing of APIs, transformation, and loading into MongoDB-is done in an automated and efficient Glow, Logs are generated at each step and errors for every step are registered in a log file to trace those errors.

Cluster Activity

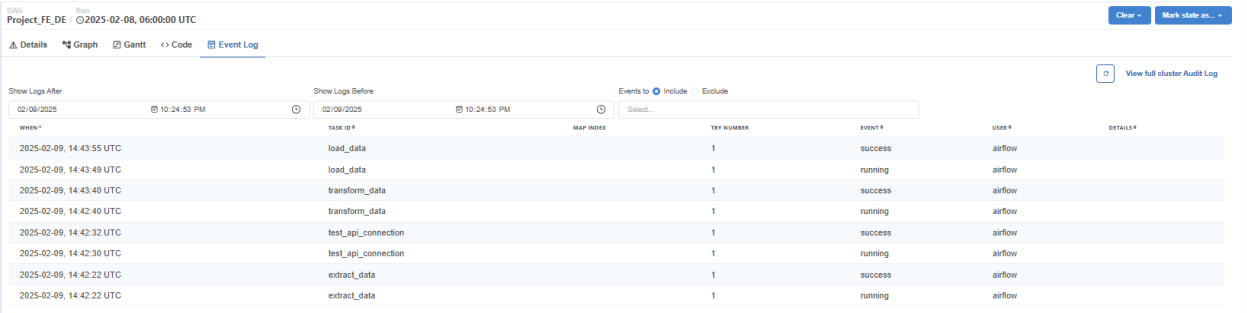


Verify the Loaded Data in **DAG and MongoDB**

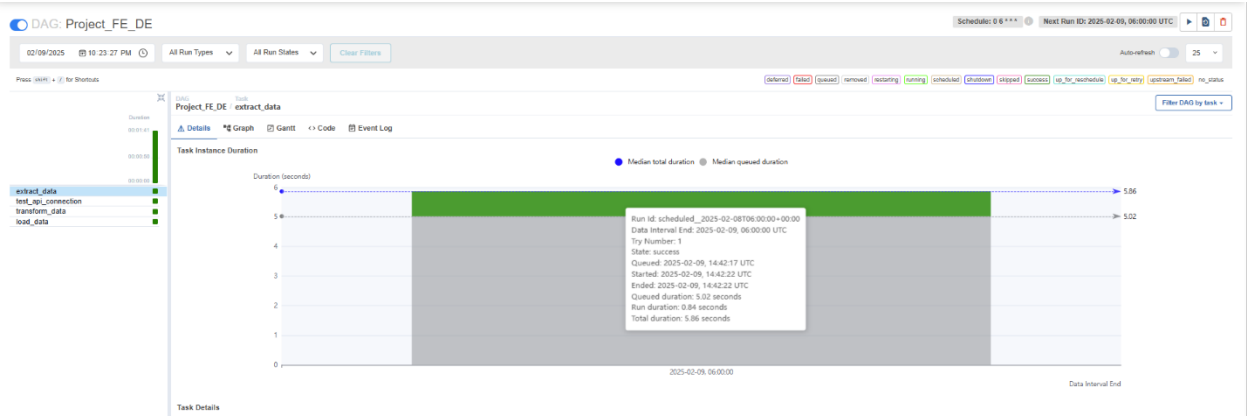


Graded Assessment Data Pipeline Project REPORT: Week 3

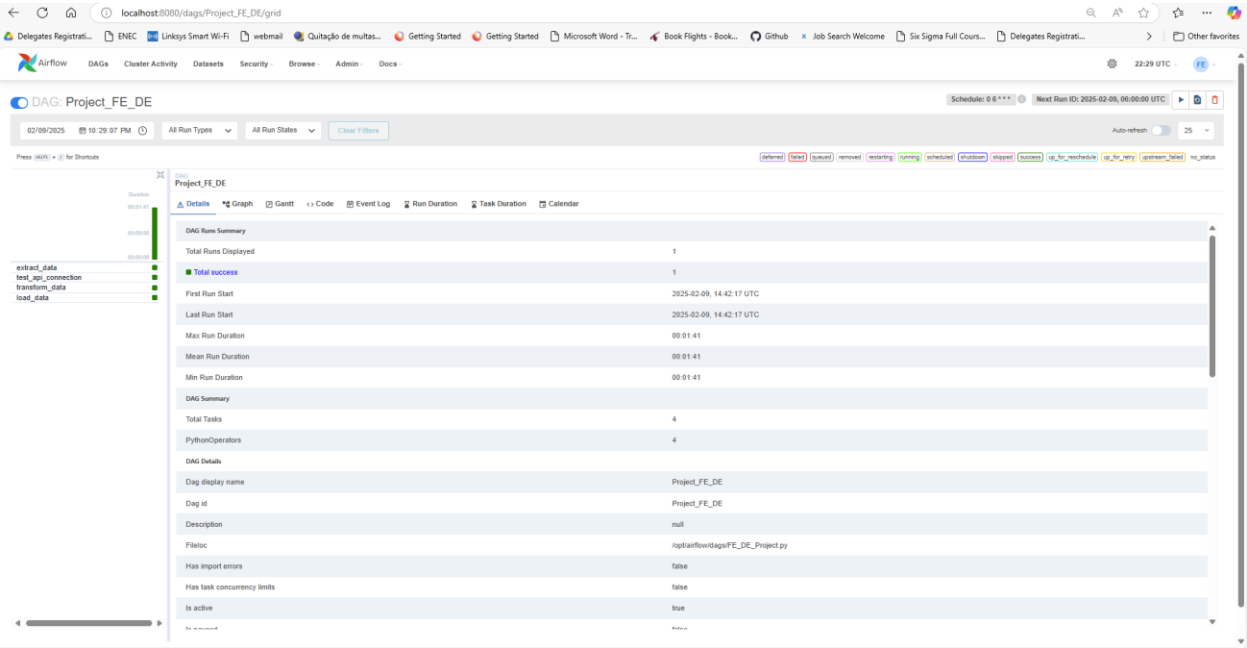
Tasks Status



Duration : Total Tasks 5.86s



Details of the DAG Scheduled



Connecting to MongDB and getting the Loaded Data on collection sales_weather

[44]:
```
# Connect to MongoDB
client = MongoClient("mongodb+srv://faysalelawar:pb6LB2kBPQ5Be5vN@dataengineeringcluster.61mrj.mongodb.net/?retryWrites=true&w=majority&appName=DataEngin
db = client["sales_db"]
collection = db["sales_weather"]

# Fetch data from MongoDB
df = pd.DataFrame(list(collection.find({}, {"_id": 0})))  # Convert cursor to list first

# Print column names to verify
print(df.columns)
```

```
Index(['date', 'product_id', 'sales_amount', 'store_location',
       'Temperature (°C)', 'Humidity (%)', 'Weather Description'],
      dtype='object')
```

Printing the Head for 15 rows.

[54]:
```
df.head(15)
```

[54]:

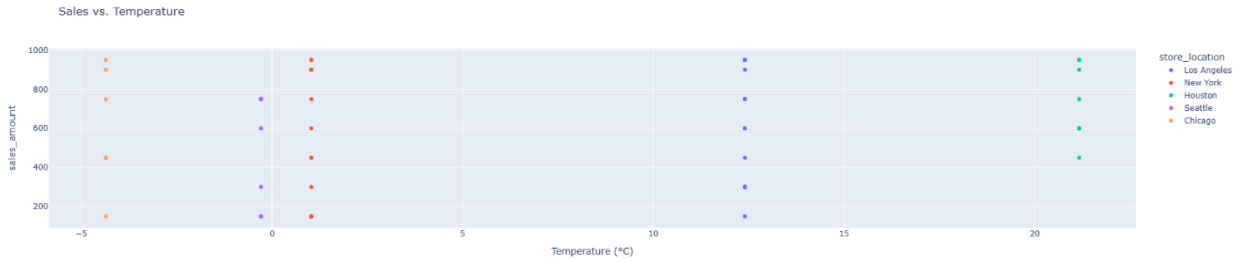
	date	product_id	sales_amount	store_location	Temperature (°C)	Humidity (%)	Weather Description
0	2025-02-01	P003	450	Los Angeles	12.40	30.0	broken clouds
1	2025-02-01	P002	950	New York	1.03	87.0	broken clouds
2	2025-02-01	P002	950	Houston	21.17	83.0	overcast clouds
3	2025-02-02	P001	150	Seattle	-0.29	95.0	overcast clouds
4	2025-02-02	P004	600	New York	1.03	87.0	broken clouds
5	2025-02-02	P005	750	Los Angeles	12.40	30.0	broken clouds
6	2025-02-02	P001	900	Los Angeles	12.40	30.0	broken clouds
7	2025-02-02	P001	900	Chicago	-4.36	58.0	overcast clouds
8	2025-02-02	P002	300	Seattle	-0.29	95.0	overcast clouds
9	2025-02-02	P005	750	New York	1.03	87.0	broken clouds
10	2025-02-03	P002	300	New York	1.03	87.0	broken clouds
11	2025-02-03	P005	750	Houston	21.17	83.0	overcast clouds
12	2025-02-04	P002	950	Chicago	-4.36	58.0	overcast clouds
13	2025-02-04	P001	150	Los Angeles	12.40	30.0	broken clouds
14	2025-02-04	P003	450	Houston	21.17	83.0	overcast clouds

Plotting on Dash

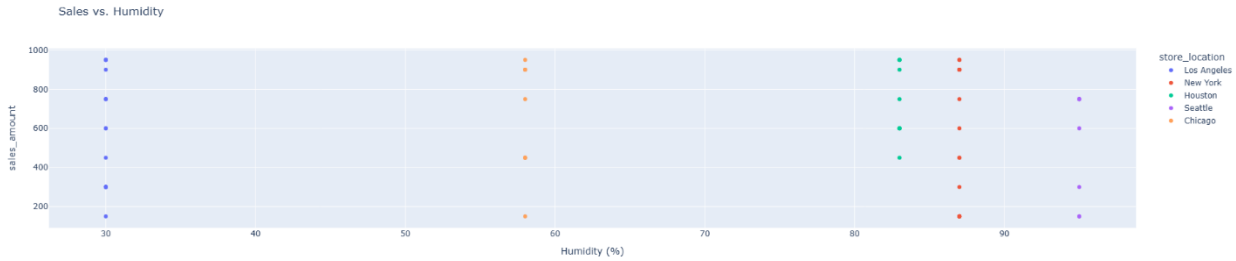
The Dash app visualizes the relationship between sales and weather tha have been loaded and stored in MongoDB.

The Plots as follows:

1. **Scatter Plot: Sales vs. Temperature**: Displays the relationship between sales and temperature for different store locations.

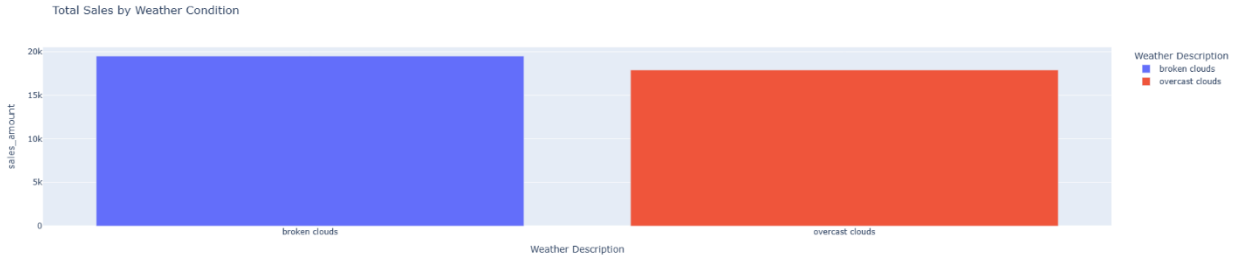


2. **Scatter Plot: Sales vs. Humidity**: Shows how sales correlate with humidity levels across various stores.

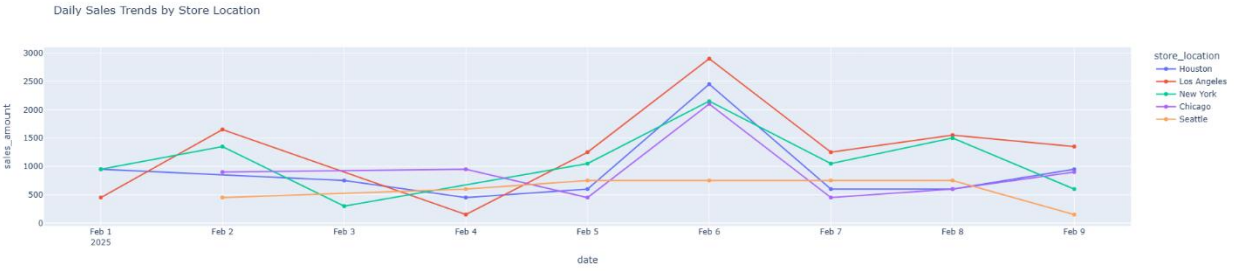


3. **Bar Chart: Total Sales per Store Location**: Visualizes total sales by store location.

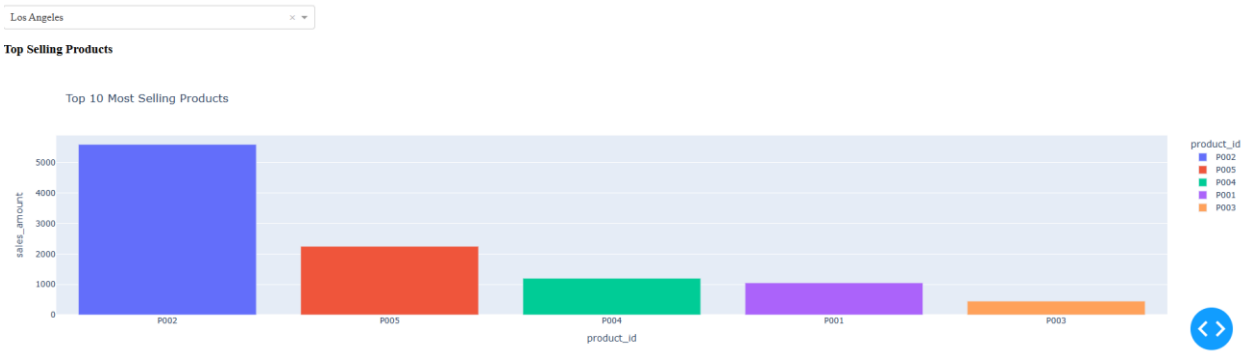
4. **Bar Chart: Total Sales by Weather Condition**: Aggregates sales data by weather description to show how different weather conditions impact sales.



5. **Line Chart: Daily Sales Trends by Store Location**: Tracks daily sales trends over time for each store, with hover data showing weather and sales details.



6. **Bar Chart: Top 10 Most Selling Products**: Displays the top-selling products, including average weather data (temperature and humidity) for each product. With an added drop done button for store location.



Links to Download :

1. Video Link on the Summary of Dash & ETL
   Project_final_vid.mp4

2. Sales Data used : AAI_634O/Week3/Project/sales_data_FEB2025.csv at main · falawar7/AAI_634O
3. My Notebook : AAI_634O/Week3/Project/Faysal_Elawar_Project_final_Rev1.ipynb at main · falawar7/AAI_634O
4. DAG Apache Airflow Scheduler: AAI_634O/Week3/Project/FE_W3_Apache_Airflow_Project.py at main · falawar7/AAI_634O
5. Logs for Dag AAI_634O/Week3/Project/dag_id=Project_FE_DE.rar at main · falawar7/AAI_634O
6. JSON file exported from MongoDB but I didn't use it :
   AAI_634O/Week3/Project/sales_db.sales_weather_without_id.json at main · falawar7/AAI_634O
7. Dash App : AAI_634O/Week3/Project/FE_W3_Project_Dash.py at main · falawar7/AAI_634O

_____Thank You_____