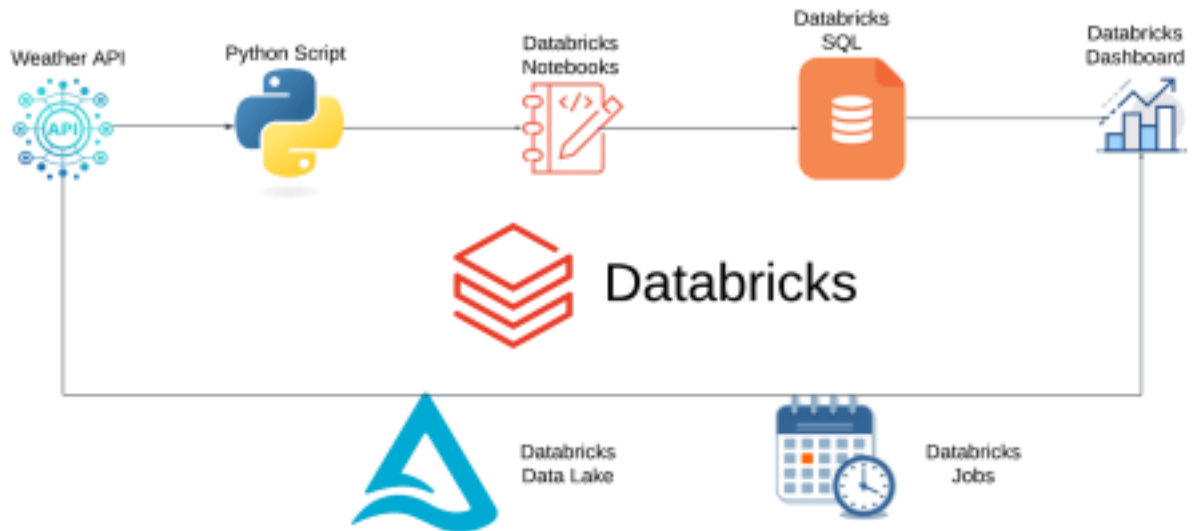


Workflow



Data Ingestion

```
# Install necessary libraries
import requests
import datetime

# API setup
API_KEY = 'c1d0f294ee92ed3c62e99d697f6f615e'
CITY = 'West Bengal'
URL = f"http://api.openweathermap.org/data/2.5/weather?q={CITY}&appid={API_KEY}"
```

Data Validation And Error Handling

```
# Function to fetch weather data
def fetch_weather_data():
    response = requests.get(URL)
    if response.status_code == 200:
        data = response.json()
        return {
            'city': data['name'],
            'temperature': data['main']['temp'],
            'description': data['weather'][0]['description'],
            'timestamp': datetime.datetime.now().isoformat()
        }
    else:
        print(f"Error: {response.status_code}")
        return None
```

```
4 days ago (10) 7 Python

# Fetch and display weather data
weather_data = fetch_weather_data()
print(weather_data)

{'city': 'West Bengal', 'temperature': 301.86, 'description': 'broken clouds', 'timestamp': '2024-11-06T05:00:28.129831'}
```

Data Storage

```
4 days ago (30s) 9

from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.getOrCreate()

# Convert weather data to DataFrame
weather_df = spark.createDataFrame([weather_data])

# Save the DataFrame to Delta Lake
weather_df.write.format("delta").mode("append").save("/mnt/delta/weather_data")

(3) Spark Jobs

weather_df: pyspark.sql.dataframe.DataFrame
city: string
description: string
temperature: double
timestamp: string
```

Data Preprocessing and Cleaning

```
4 days ago (8s) 11 Python

weather_df = spark.read.format("delta").load("/mnt/delta/weather_data")

# Clean data: drop null values
cleaned_df = weather_df.dropna()

# Show cleaned data
cleaned_df.show()

(4) Spark Jobs

weather_df: pyspark.sql.dataframe.DataFrame
Schema Details History
city: string
description: string
temperature: double
timestamp: string

cleaned_df: pyspark.sql.dataframe.DataFrame
city: string
description: string
temperature: double
timestamp: string
```

Data Analysis and Aggregation

```
Scala
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._
val weatherDataPath = "/mnt/delta/weather_data"
val rawData = spark.read.format("delta").load(weatherDataPath)
rawData.show()
val cleanedData = rawData
  .select(
    col("city"),
    col("temperature").cast("double"),
    col("description"),
    col("timestamp").cast("timestamp")
  )
  .na.drop() // Drop rows with any missing values

// Calculate the average temperature per city
val averageTemp = cleanedData.groupBy("city").agg(avg("temperature").alias("avg_temperature"))

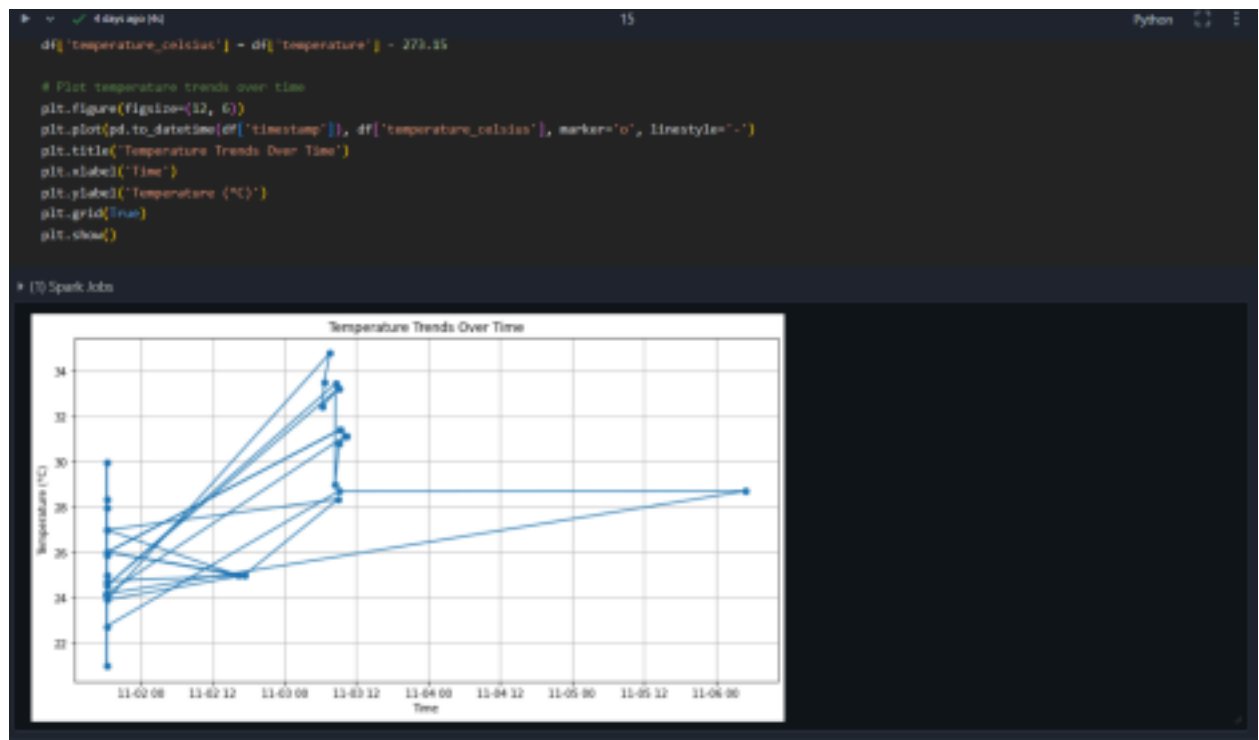
// Display the results
averageTemp.show()
```

▶ (3) Spark Jobs

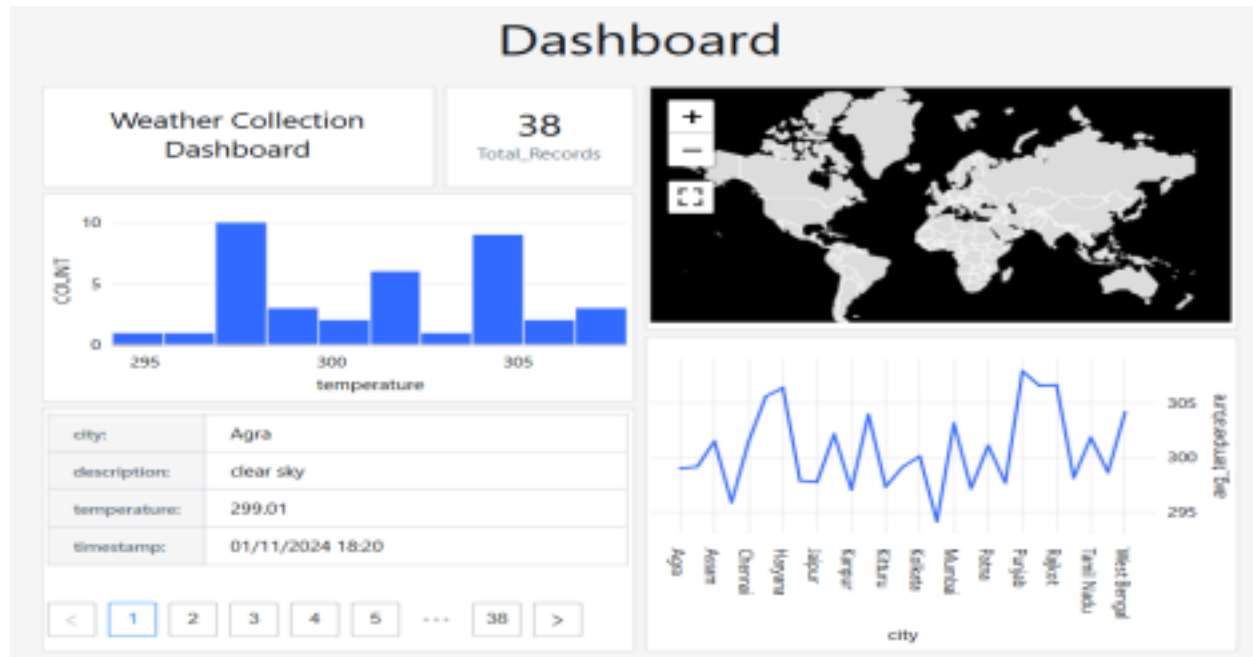
- ▶ `rawData`: org.apache.spark.sql.DataFrame = (city: string, description: string ... 2 more fields)
- ▶ `cleanedData`: org.apache.spark.sql.DataFrame = (city: string, temperature: double ... 2 more fields)
- ▶ `averageTemp`: org.apache.spark.sql.DataFrame = (city: string, avg_temperature: double)

city	description	temperature	timestamp
------	-------------	-------------	-----------

Data Visualization



Building Report Using Dashboard



Monitoring Pipeline

```

def monitor_delta_table():
    try:
        # Track the current record count in Delta table
        logging.info("Fetching Delta table record count...")
        record_count = spark.read.format("delta").load("/mnt/delta/weather_data").count()
        logging.info(f"Current record count in Delta table: {record_count}")

        # Track the schema of the Delta table
        logging.info("Fetching Delta table schema...")
        schema = spark.read.format("delta").load("/mnt/delta/weather_data").schema
        logging.info(f"Delta table schema: {schema}")

    except Exception as e:
        logging.error(f"Error occurred while monitoring Delta table: {str(e)}")

# Run the monitor function
monitor_delta_table()

# [D] Spark logs
2024-11-06 05:02:00,100 - INFO - Fetching Delta table record count...
2024-11-06 05:02:00,105 - INFO - Current record count in Delta table: 38
2024-11-06 05:02:00,107 - INFO - Fetching Delta table schema...
2024-11-06 05:02:00,108 - INFO - Delta table schema: StructType([StructField('city', StringType(), True), StructField('description', StringType(), True), StructField('temperature', DoubleType(), True), StructField('timestamp', StringType(), True)])
  
```

Automate Collection

```

spark = SparkSession.builder().getOrCreate()

# Function to fetch weather data
def collect_weather_data():
    response = requests.get(URL)
    if response.status_code == 200:
        data = response.json()
        weather_data = {
            'city': data['name'],
            'temperature': data['main']['temp'],
            'description': data['weather'][0]['description'],
            'time': datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }

        # Convert weather data to DataFrame
        weather_df = spark.createDataFrame([weather_data])

        # Save the DataFrame to Delta Lake
        weather_df.write.format("delta").mode("append").save("/mnt/delta/weather_data")
        print("Weather data collected and stored successfully.")

    else:
        print(f"Error: {response.status_code}")

# Schedule the job to run at regular intervals
scheduler = CronScheduler(spark, collect_weather_data)

# Keep the scheduler running
while True:
    scheduler.run_pending()
    time.sleep(5)

```