

Mini Project: Cloud-Based Big Data Analytics Pipeline with AWS and PySpark

1. Introduction

1.1 Project Overview

The primary object of this mini project is to provide a robust solution for building an end-to-end Data Pipeline to collect and process Raw Data using Distributed Computing technologies, store the processed data in a Cloud Storage solution and generate Reporting/Visualisation tools to extract analytical insights from the processed data. The Proposed pipeline is designed such that it will be capable of processing large datasets in a timely manner and demonstrates some of the key considerations when implementing real-world practices used within the Data Engineering process, such as: Automation, Logging, Monitoring and Ethical Considerations. This mini project focuses on designing and implementing a cloud-based big data analytics pipeline using Amazon Web Services (AWS) and Apache PySpark.

1.2 Dataset Description

For this project, the dataset used is referred to as the **Airline Delay Cause Dataset**. This dataset represents thousands of airline flights operations over multiple years; it provides a comprehensive overview of the quantity of flights that arrived on time, delays, cancellations and the reason(s) for the delay.

Key Characteristics of the Dataset

- A large dataset comprised of large numbers of records/rows and attributes over multiple years- ideal for use in Big Data analysis.
- Multiple Data Types comprising of both Numeric and Categorical Attribute Types.
- Possibility of performing Temporal, Geographical, and Performance Analysis on data.

Important Columns

- year, month
- airport, airport_name
- carrier, carrier_name
- arr_flights - *Total number of arrival flights*
- arr_del15 - *flights delayed more than 15 minutes*
- arr_cancelled, arr_diverted - *flights cancelled and diverted*
- Delay cause metrics:
 - carrier_ct - *Carrier-related delays*
 - weather_ct - *Weather-related delays*
 - nas_ct - *Air Traffic Control / National Airspace System delays*
 - late_aircraft_ct - *Delays caused by late-arriving aircraft*

This dataset is well-suited for deriving meaningful insights such as:

- Monthly and yearly flight trends
- Airport-level and airline-level performance comparison
- Delay cause analysis
- Operational reliability assessment

2. Dataset Selection

2.1 Rationale for Dataset Choice

The airline dataset is a good representation of how big data can be used for analysis and decision-making. With the sheer size of data generated every day, airlines will find this dataset useful due to its capacity for distributed processing (in using cloud-based technologies) to manage that amount of data.

The dataset satisfies the project requirements because:

- Sizeable enough for distributed processing.
- Diverse enough to allow for transformation and aggregation of many different attribute types.
- Allows for numerous analytical perspectives (e.g., by time, location, performance).

2.2 Ethical Considerations in Dataset Usage

While working with real-world datasets, ethical concerns must be considered:

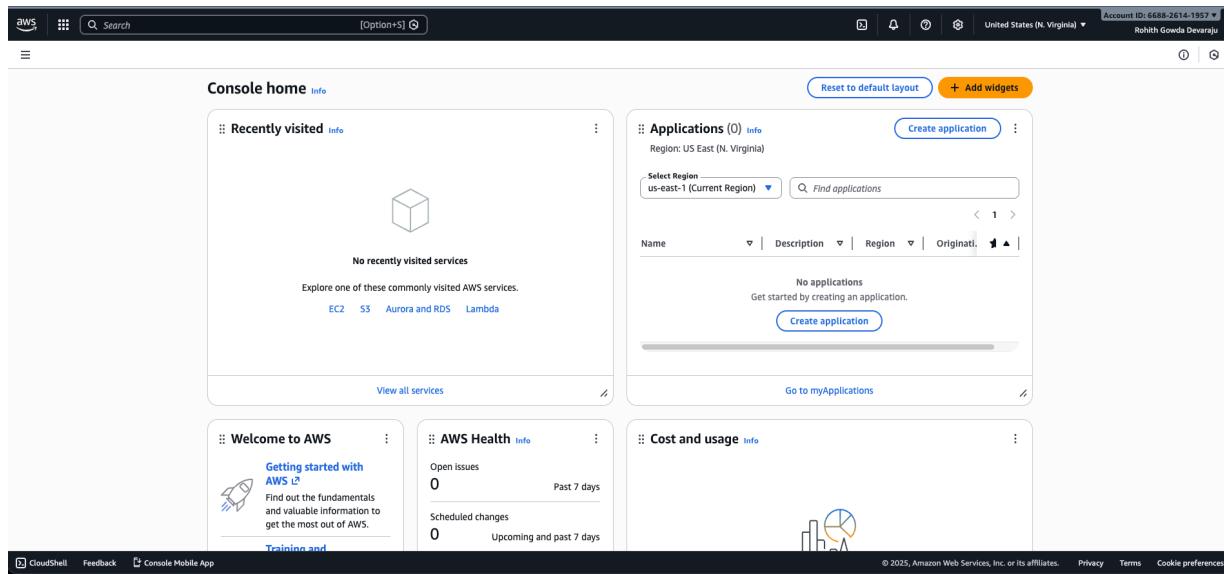
- Bias: It is present in many datasets. It can occur if a dataset has more data for specific airlines or airport. This would then influence the results of an analysis or the type of machine-learning model being used.
- Privacy: This aspects would have also been compromised as the dataset does not contain any personally identifiable information (PII), thus reducing the risk of privacy infringements.
- Risks: When using aggregated metrics, it is important to be aware of the potential pitfalls when interpreting them to avoid drawing false conclusions.

These considerations were acknowledged throughout the analysis process.

The screenshot shows the Kaggle platform interface. On the left is a sidebar with navigation links: kaggle, Create, Home, Competitions, Datasets (which is selected and highlighted in blue), Models, Benchmarks, Game Arena, Code, Discussions, Learn, and More. The main content area displays a dataset titled "Airline Delay Cause" by user "ABDELAZIZ EL7OR". The title has a timestamp "UPDATED 19 DAYS AGO". Below the title is a small thumbnail image of an airplane. The dataset summary states: "Why your flight was late: 20 years of U.S. airline delays". There are tabs for Data Card, Code (12), Discussion (0), and Suggestions (0). The "Data Card" tab is active. The "About Dataset" section includes a note: "You can use this as-is or edit it a bit:". A detailed description follows: "This dataset contains U.S. domestic airline delay statistics by cause over almost two decades. Each row represents a combination of: Year & month, Operating carrier (e.g. Delta, SkyWest, etc.), Origin airport (IATA code + full airport name). For every (carrier, airport, month) pair, the dataset includes: Total number of flights". To the right of the Data Card are several metadata fields: Usability (10.00), License (Apache 2.0), Expected update frequency (Annually), and Tags (Data Visualization, DNN). At the bottom of the page, a footer notes: "Kaggle uses cookies from Google to deliver and enhance the quality of its services and to analyze traffic." with "Learn more" and "OK, Got it." buttons.

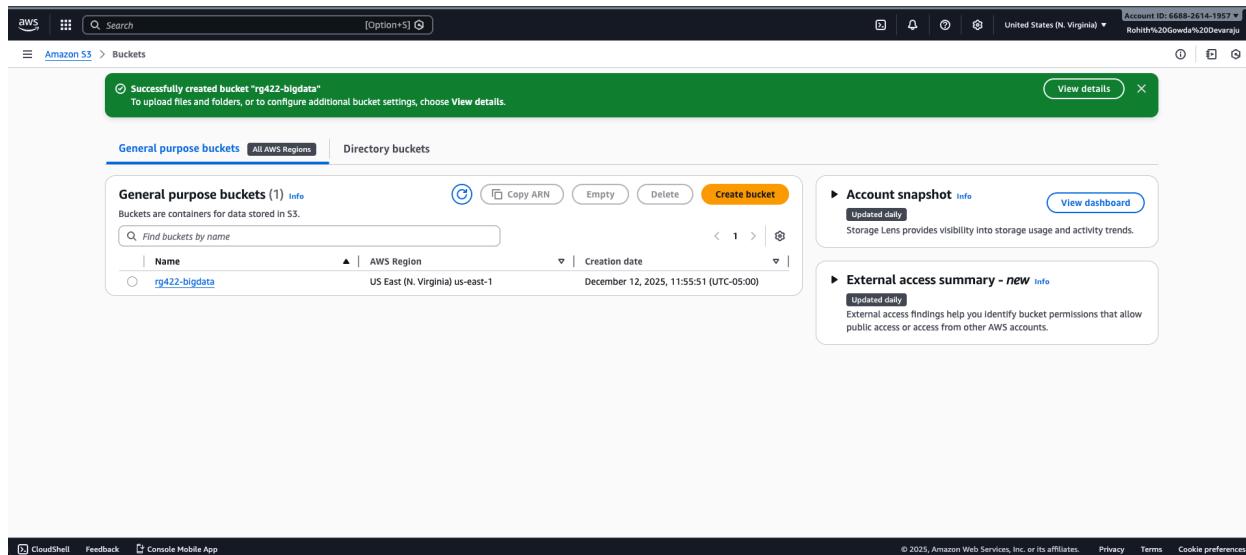
3. Environment Setup

In this section, you will establish the infrastructure and Distributed Processing Environment necessary to execute the Data Pipeline.



3.1.1 S3 Bucket Creation

For this project Amazon S3 was utilized as the main cloud storage solution, and provides secure, highly scalable Object Storage that is ideal for storing Large Datasets.



Bucket Structure

A Dedicated S3 Bucket has been created to store all the data pertaining to this project. The S3 Bucket has been Organized into Several Different Folders which allow for easy separation of RAW's, PROCESSED DATA, and LOGS from each other.

The screenshot shows the AWS S3 console interface. At the top, there's a green success message: "Successfully created folder 'outputs'". Below it, the bucket name "rg422-bigdata" is shown with an "Info" link. A navigation bar includes tabs for Objects, Metadata, Properties, Permissions, Metrics, Management, and Access Points. Under the Objects tab, a table lists four objects: "autopilot_input/" (Folder), "outputs/" (Folder), "processed/" (Folder), and "raw/" (Folder). Each entry has columns for Name, Type, Last modified, Size, and Storage class. Action buttons like Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload are available at the top of the object list.

The primary goals of this Folder Organization are to Enable:

- Clear separation of raw and processed data
- Easier debugging and monitoring
- Better organization for automation and analytics

3.1.2 Uploading Raw Dataset to S3

The raw airline dataset was uploaded to the `raw/` folder within the S3 bucket using the AWS Management Console and AWS CLI.

This step ensures:

- Centralized cloud storage
- Easy access by PySpark jobs running on EC2
- Scalability for future dataset expansion

The screenshot shows the AWS S3 console after an upload. A green success message at the top says "Upload succeeded. For more information, see the Files and folders table." Below it, a summary table shows the destination as `s3://rg422-bigdata/raw/`, with one file uploaded successfully (1 file, 40.6 MB) and zero files failed. The "Files and folders" section shows a table with one item: "Airlne_Delay_Cause.csv" (text/csv, 40.6 MB, Status: Succeeded). Navigation links for CloudShell, Feedback, and Console Mobile App are at the bottom.

The screenshot shows the AWS S3 console interface. At the top, there's a search bar with 'Search' and a dropdown for 'Option+S'. On the right, it shows 'Account ID: 6688-2614-1957' and 'United States (N. Virginia)'. Below the header, the path 'Amazon S3 > Buckets > rg422-bigdata > raw/' is displayed. The main area is titled 'raw/' and contains a table with one object: 'Airline_Delay_Cause.csv'. The table has columns for Name, Type, Last modified, Size, and Storage class. The object details are: Name is 'Airline_Delay_Cause.csv', Type is 'csv', Last modified is 'December 12, 2025, 11:59:04 (UTC-05:00)', Size is '40.6 MB', and Storage class is 'Standard'. There are buttons for Actions (Copy S3 URI, Copy URL, Download, Open, Delete), Create folder, and Upload.

3.2 Linux Environment with PySpark

A Linux-based environment was set up using an AWS EC2 instance to enable distributed data processing with PySpark.

3.2.1 EC2 Instance Setup

An EC2 instance running **Ubuntu Linux** was launched with appropriate compute and storage resources. The instance was assigned an IAM role to securely access AWS services such as S3 and SNS without hard-coding credentials.

An EC2 instance was launched with the following configuration:

- **Instance type:** t3.micro
- **Operating system:** Ubuntu Linux
- **Instance name:** rg422-Airline-Pyspark

The t3.micro instance was selected because it falls under the AWS Free Tier, making it cost-effective while still suitable for running PySpark jobs for this project.

Using EC2 provides:

- Scalability for large data processing tasks
- Tight integration with AWS storage and analytics services
- Real-world experience with cloud-based data engineering workflows

aws Search [Option+S] Ask Amazon Q Account ID: 6688-2614-1957 ▾ Rohith%20Gowda%20Devaraju United States (N. Virginia) ▾

EC2 > instances Launch an Instance rg422-key Create new key pair

Network settings Info Edit

Network Info vpc-02451d1bcb84a77141

Subnet Info No preference (Default subnet in any availability zone)

Auto-assign public IP Info Enable

Firewall (security groups) Info A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group Select existing security group

We'll create a new security group called 'launch-wizard-1' with the following rules:

- Allow SSH traffic from My IP 73.105.50.135/32 Helps you connect to your instance
- Allow HTTPS traffic from the internet To set up an endpoint, for example when creating a web server
- Allow HTTP traffic from the internet To set up an endpoint, for example when creating a web server

Configure storage Info Advanced

1x 8 GB gp3 Root volume, 3000 IOPS, Not encrypted

Cancel Launch instance Preview code

CloudShell Feedback Console Mobile App © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws Search [Option+S] Account ID: 6688-2614-1957 ▾ Rohith%20Gowda%20Devaraju United States (N. Virginia) ▾

EC2 > Instances

Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4...	Elastic IP
rg422-Airline-PySpark	i-01b4b5c007299846d	Running	t3.micro	Initializing		us-east-1f	ec2-35-172-195-228.co...	35.172.195.228	-

i-01b4b5c007299846d (rg422-Airline-PySpark)

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary Info

Instance ID i-01b4b5c007299846d Public IPv4 address 35.172.195.228 [open address]

IPv6 address - Instance state Running Private IP DNS name (IPv4 only) ip-172-31-64-111.ec2.internal

Hostname type IP name: ip-172-31-64-111.ec2.internal Answer private resource DNS name IPv4 (A) Instance type t3.micro Private IPv4 addresses 172.31.64.111

Public DNS ec2-35-172-195-228.compute-1.amazonaws.com [open address]

Elastic IP addresses -

CloudShell Feedback Console Mobile App © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

aws Search [Option+S] Account ID: 6688-2614-1957 ▾ Rohith%20Gowda%20Devaraju United States (N. Virginia) ▾

EC2 > Instances

Instances (1/1) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4...	Elast...
rg422-Airline-PySpark	i-01b4b5c007299846d	Running	t3.micro	3/3 checks passed		us-east-1f	ec2-35-172-195-228.co...	35.172.195.228	-

i-01b4b5c007299846d (rg422-Airline-PySpark)

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary Info

Instance ID i-01b4b5c007299846d Public IPv4 address 35.172.195.228 [open address]

IPv6 address - Instance state Running Private IP DNS name (IPv4 only) ip-172-31-64-111.ec2.internal

Hostname type IP name: ip-172-31-64-111.ec2.internal Answer private resource DNS name IPv4 (A) Instance type t3.micro Private IPv4 addresses 172.31.64.111

Public DNS ec2-35-172-195-228.compute-1.amazonaws.com [open address]

Elastic IP addresses -

CloudShell Feedback Console Mobile App © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

3.2.2 IAM Role Creation and Attachment

To securely access AWS resources from the EC2 instance without embedding credentials in code, an **IAM role** was created.

- **IAM Role name:** rg422-EC2-S3-Role

This role was configured with permissions to:

- Read and write data to Amazon S3
- Publish notifications to Amazon SNS

The IAM role was then **attached to the EC2 instance**, enabling secure and credential-free access to AWS services from within the instance.

This approach improves:

- Security (no hard-coded access keys)
- Maintainability
- Compliance with AWS best practices

The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with navigation links like 'Identity and Access Management (IAM)', 'Dashboard', 'Access management', 'Access reports', and 'Access analyzer'. The main content area has a heading 'Roles (4) Info' with a note: 'An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.' Below this is a table with four rows:

Role name	Trusted entities	Last activity
AWSServiceRoleForResourceExplorer	AWS Service: resource-explorer-2 (St)	7 minutes ago
AWSServiceRoleForSupport	AWS Service: support (Service-Linker)	-
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service)	-

Below the table, there are two sections: 'Access AWS from your non AWS workloads' (using X.509 Standard) and 'Temporary credentials' (using AWS Certificate Manager). At the bottom right of the main content area is a 'Create role' button.

The screenshot shows the 'Modify IAM role' dialog for an EC2 instance. The instance ID is i-01b4b5c007299846d, and the role attached is rg422-Airline-PySpark. The dialog has a section titled 'IAM role' with a dropdown menu where 'rg422-EC2-S3-Role' is selected. Other options include 'Create new IAM role' and 'Cancel'. At the bottom right is a large orange 'Update IAM role' button.

The screenshot shows the AWS EC2 Instances page. A success message at the top indicates "Successfully attached rg422-EC2-S3-RoIe to instance i-01b4b5c007299846d". The main table lists one instance: "rg422-Airline-PySpark" (i-01b4b5c007299846d), which is "Running" and "t3.micro" type. It has 3/3 checks passed and a Public IP of 35.172.195.228. The sidebar on the left shows navigation links for EC2 services like Dashboard, Global View, Events, Instances, Images, Elastic Block Store, Network & Security, and more.

3.2.3 Connecting to EC2 via SSH

With the use of an SSH client from the user's local machine, the EC2 instance was able to establish a local connection. With the use of SSH connectivity, it allows for command-line interaction with the Linux environment for the purpose of software installation, running of PySpark jobs, and automation script management.

This process allows for:

- Management of the EC2 Instance remotely.
- Execution of Spark jobs.
- Monitoring of pipeline executions.

The screenshot shows the "Connect" dialog for the EC2 instance i-01b4b5c007299846d. The "SSH client" tab is selected. It provides instructions for connecting using an SSH client, mentioning the private key file (rg422-key.pem) and the Public DNS (ec2-35-172-195-228.compute-1.amazonaws.com). A sample command is shown: "ssh -l `rg422-key.pem` ubuntu@ec2-35-172-195-228.compute-1.amazonaws.com". A note at the bottom states: "Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username." A "Cancel" button is visible at the bottom right.

```

root@rohitdeverla:~# ssh -i rg422-key.pem ubuntu@ec2-35-172-195-228.compute-1.amazonaws.com
The authenticity of host 'ec2-35-172-195-228.compute-1.amazonaws.com (59.172.195.228)' can't be established.
ECDSA key fingerprint: SHA256:KjyXvhvbk7k9p1hgQRM/d0eyZ11YXaI.
This key is not known by any other name.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
Warning: Permanently added 'ec2-35-172-195-228.compute-1.amazonaws.com' (ED25519) to the list of known hosts.
[ec2-35-172-195-228:~]# lsb_release -a
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/for

System information as of Fri Dec 12 17:29:39 UTC 2028

System load: 0.0 Temperature:          -37.1 C
Usage of /: 25.8% of 6.71GB Processes:      109
Memory usage: 23% Users logged in: 0
Swap usage: 0% IPv4 address for ens5: 172.31.64.111

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright*.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-64-111:~#

```

```

● ● ● Downloads — ubuntu@ip-172-31-64-111: ~ — ssh -i rg422-key.pem ubuntu@ec2-35-172-195-228.compute-1.amazonaws.com — 269x65
[ubuntu@ip-172-31-64-111:~]$ aws --version
aws-cli/2.32.15 Python/3.13.11 Linux/6.14.0-1015-aws exe/x86_64/ubuntu.24
ubuntu@ip-172-31-64-111:~#

```

```

[ubuntu@ip-172-31-64-111:~]$ aws --version
aws-cli/2.32.15 Python/3.13.11 Linux/6.14.0-1015-aws exe/x86_64/ubuntu.24
[ubuntu@ip-172-31-64-111:~]$ aws s3 ls
2025-12-12 16:55:51 rg422-bigdata
[ubuntu@ip-172-31-64-111:~]$ aws s3 ls s3://rg422-bigdata
    PRE autopilot_input/
    PRE outputs/
    PRE processed/
    PRE raw/
ubuntu@ip-172-31-64-111:~#

```

```

[ubuntu@ip-172-31-64-111:~]$ mkdir -p ~/airline_project/data
[ubuntu@ip-172-31-64-111:~]$ cd ~/airline_project
ubuntu@ip-172-31-64-111:~/airline_project$ 

```

3.2.4 Installation of Java and PySpark

Apache Spark requires Java to run. The following installations were performed on the EC2 instance:

```

ubuntu@ip-172-31-64-111:~/airline_project$ python3 -m venv venv
ubuntu@ip-172-31-64-111:~/airline_project$ ls
data ingest_airline.py venv
ubuntu@ip-172-31-64-111:~/airline_project$ 

```

Apache Spark and PySpark Installation

- Apache Spark (version 3.5.x) was installed
- PySpark was installed within a Python virtual environment
- Spark installation was verified using:

```

[ubuntu@ip-172-31-64-111:~/airline_project$ python3 -m venv venv
[ubuntu@ip-172-31-64-111:~/airline_project$ ls
data ingest_airline.py venv
[ubuntu@ip-172-31-64-111:~/airline_project$ source venv/bin/activate
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ pip install pyspark
Collecting pyspark
  Downloading pyspark-4.0.1.tar.gz (434.2 MB)
   ━━━━━━━━━━━━━━━━━━━━ 434.2/434.2 MB 374.9 kB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Collecting py4j==0.10.9.9 (from pyspark)
    Downloading py4j-0.10.9.9-py2.py3-none-any.whl.metadata (1.3 kB)
  Downloading py4j-0.10.9.9-py2.py3-none-any.whl (203 kB)
   ━━━━━━━━━━━━━━━━━━ 203.0/203.0 kB 18.9 MB/s eta 0:00:00
  Building wheels for collected packages: pyspark
  Building wheel for pyspark (pyproject.toml) ... done
  Created wheel for pyspark: filename=pyspark-4.0.1-py2.py3-none-any.whl size=434813860 sha256=b2a59e3d4fe15d0af4ff31b0eaa8c26c97b5a964a4eb96b7af961bbe0697d1f3
  Stored in directory: /home/ubuntu/.cache/pip/wheels/31/9f/68/f89fb34cccd886989be7d0e390eaa9f97f21efdf540c0ee8bcd
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.9 pyspark-4.0.1
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ python -c "import pyspark; print(pyspark.__version__)"
4.0.1
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ 

```

Java Installation

- OpenJDK was installed to satisfy Spark's runtime requirements
- Java installation was verified using:

```
[ubuntu@ip-172-31-64-111:~/airline_project]$ ls /usr/lib/jvm
[ubuntu@ip-172-31-64-111:~/airline_project]$ ls -l /usr/lib/jvm/java-17-openjdk-amd64 openjdk-11 openjdk-17
[ubuntu@ip-172-31-64-111:~/airline_project]$ export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
[ubuntu@ip-172-31-64-111:~/airline_project]$ export PATH=$JAVA_HOME/bin:$PATH
[ubuntu@ip-172-31-64-111:~/airline_project]$ java -version
openjdk version "17.0.1" 2022-10-19
OpenJDK Runtime Environment (build 17.0.1+18-Ubuntu-124.84)
OpenJDK 64-Bit Server VM (build 17.0.1+18-Ubuntu-124.84, mixed mode, sharing)
[ubuntu@ip-172-31-64-111:~/airline_project]$
```

3.2.5 AWS CLI Configuration

The EC2 instance will be able to access and perform programmatically on Amazon S3 and other AWS services through the use of the AWS Command Line Interface. The EC2 instance's IAM role allows it to authenticate without the need for access keys. In this case, the AWS CLI automatically authenticates with the instance's IAM role.

As a result, you will be able to:

- Upload and download files from S3
- Log pipeline output files in S3
- Automate the management of your Data Workflows via the CL

```
[ubuntu@ip-172-31-64-111:~/airline_project]$ aws s3 cp s3://rg422-bigdata/raw/Airline_Delay_Cause.csv data/
download: s3://rg422-bigdata/raw/Airline_Delay_Cause.csv to data/Airline_Delay_Cause.csv
[ubuntu@ip-172-31-64-111:~/airline_project]$ ls data
Airline_Delay_Cause.csv
[ubuntu@ip-172-31-64-111:~/airline_project]$
```

4. Data Pipeline Tasks

Task 1: Data Ingestion from S3

Objective

This assignment's aim was to ingest unstructured data, which resides in Amazon S3 into a processing environment using PySpark for future applications to format the data and perform analysis.

Implementation

The PySpark execution environment was provided by AWS EC2 (t3.micro) running Amazon Linux. The EC2 instance was given permission to read & write to Amazon S3 using IAM Roles. The unprocessed airline dataset was uploaded to an S3 Bucket created for the project, and the Direct S3 integration feature of PySpark was used to pull data into PySpark from S3 directly, without first downloading the dataset to the EC2 instance.

Verification

Successful ingestion was confirmed by:

- Displaying the schema using df.printSchema()
- Viewing sample records using df.show()
- Verifying row counts using df.count()

This confirmed that the dataset was correctly loaded into the PySpark environment from S3.

```
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ cat ingest_airline.py
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Airline Delay Analysis") \
    .getOrCreate()

df = spark.read.option("header", True) \
    .option("inferSchema", True) \
    .csv("data/Airline_Delay.Cause.csv")

print("Number of rows:", df.count())
print("Schema:")
df.printSchema()

df.show(10, truncate=False)

spark.stop()
(venv) ubuntu@ip-172-31-64-111:~/airline_project$ python ingest_airline.py
WARNING: Logging configuration is not set up correctly. Please fix by calling org.apache.spark.util.Log4jConfig.setLog4jConfig() or by setting log4j.properties.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust log level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/12/12 17:58:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-Java classes where applicable
Number of rows: 31807
Schema:
root
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- carrier: string (nullable = true)
 |-- carrier_name: string (nullable = true)
 |-- arr_delay: double (nullable = true)
 |-- airport_name: string (nullable = true)
 |-- arr_flights: integer (nullable = true)
 |-- arr_ct: double (nullable = true)
 |-- carrier_ct: double (nullable = true)
 |-- weather_ct: double (nullable = true)
 |-- nas_ct: double (nullable = true)
 |-- security_ct: double (nullable = true)
 |-- late_aircraft_ct: double (nullable = true)
 |-- arr_cancelled: integer (nullable = true)
 |-- arr_diverted: integer (nullable = true)
 |-- arr_delay: integer (nullable = true)
 |-- carrier_delay: Integer (nullable = true)
 |-- weather_delay: integer (nullable = true)
 |-- nas_delay: integer (nullable = true)
 |-- security_delay: integer (nullable = true)
 |-- late_aircraft_delay: integer (nullable = true)

+-----+-----+-----+-----+-----+
|year|month|carrier|carrier_name |airport|airport_name |arr_flights|arr_delay|carrier_ct|weather_ct|nas_ct|security_ct|late_aircraft_ct|arr_cancelled|arr_diverted|arr_delay|carrier_delay|nas_delay|security|
+-----+-----+-----+-----+-----+
|2022|5 |9E |Endeavor Air Inc.|ABE |Allentown/Bethlehem/Easton, PA: Lehigh Valley International|136 |7 |8.95 |0.0 |0.05 |0.0 |1.0 |0 |0 |255 |222 |0 |4 |0 | |
|2022|5 |9E |Endeavor Air Inc.|ABY |Albany, GA: Southwest Georgia Regional |91 |16 |7.38 |0.0 |2.64 |0.0 |6.09 |0 |0 |184 |181 |0 |81 |0 |
|2022|5 |452 | |Endeavor Air Inc.|ACK |Nantucket, MA: Nantucket Memorial |19 |2 |8.13 |0.0 |1.0 |0.0 |8.88 |1 |0 |138 |4 |0 |186 |0 |
|2022|5 |28 | |Endeavor Air Inc.|AEX |Alexandria, LA: Alexandria International |88 |14 |7.26 |0.76 |1.35 |0.0 |1.64 |0 |0 |197 |585 |35 |125 |0 |
|2022|5 |192 | |Endeavor Air Inc.|AGS |Augusta Regional at Bush Field |181 |19 |13.84 |0.0 |3.07 |0.0 |2.09 |0 |0 |188 |662 |0 |87 |0 |
|2022|5 |59 | |Endeavor Air Inc.|AGS |Augusta, GA: Augusta Regional at Bush Field |181 |19 |13.84 |0.0 |3.07 |0.0 |2.09 |0 |0 |188 |662 |0 |87 |0 |

+-----+-----+-----+-----+-----+
|year|month|carrier|carrier_name |airport|airport_name |arr_flights|arr_delay|carrier_ct|weather_ct|nas_ct|security_ct|late_aircraft_ct|arr_cancelled|arr_diverted|arr_delay|carrier_delay|nas_delay|security|
+-----+-----+-----+-----+-----+
|2022|5 |9E |Endeavor Air Inc.|ABE |Allentown/Bethlehem/Easton, PA: Lehigh Valley International|136 |7 |8.95 |0.0 |0.05 |0.0 |1.0 |0 |0 |255 |222 |0 |4 |0 | |
|2022|5 |9E |Endeavor Air Inc.|ABY |Albany, GA: Southwest Georgia Regional |91 |16 |7.38 |0.0 |2.64 |0.0 |6.09 |0 |0 |184 |181 |0 |81 |0 |
|2022|5 |452 | |Endeavor Air Inc.|ACK |Nantucket, MA: Nantucket Memorial |19 |2 |8.13 |0.0 |1.0 |0.0 |8.88 |1 |0 |138 |4 |0 |186 |0 |
|2022|5 |28 | |Endeavor Air Inc.|AEX |Alexandria, LA: Alexandria International |88 |14 |7.26 |0.76 |1.35 |0.0 |1.64 |0 |0 |197 |585 |35 |125 |0 |
|2022|5 |192 | |Endeavor Air Inc.|AGS |Augusta Regional at Bush Field |181 |19 |13.84 |0.0 |3.07 |0.0 |2.09 |0 |0 |188 |662 |0 |87 |0 |
|2022|5 |59 | |Endeavor Air Inc.|AGS |Augusta, GA: Augusta Regional at Bush Field |181 |19 |13.84 |0.0 |3.07 |0.0 |2.09 |0 |0 |188 |662 |0 |87 |0 |
|2022|5 |217 | |Endeavor Air Inc.|ALB |Albany, NY: Albany International |134 |18 |4.42 |1.0 |6.48 |0.0 |6.09 |0 |0 |197 |224 |78 |398 |0 |
|2022|5 |12807 | |Endeavor Air Inc.|ATL |Atlanta, GA: Hartsfield-Jackson Atlanta International |3842 |453 |142.96 |13.03 |106.98|0.0 |189.03 |2 |3 |38397 |28775 |1205 |3610 |0 |
|2022|5 |1556 | |Endeavor Air Inc.|ATW |Appleton, WI: Appleton International |118 |13 |4.83 |0.0 |6.05 |0.0 |1.32 |0 |0 |194 |578 |0 |571 |0 |
|2022|5 |98 | |Endeavor Air Inc.|AUS |Austin, TX: Austin - Bergstrom International |63 |8 |8.23 |1.0 |0.89 |0.0 |0.08 |1 |0 |182 |445 |20 |59 |0 |
|2022|5 |88 | |Endeavor Air Inc.|AVL |Asheville, NC: Asheville Regional |58 |14 |4.54 |0.0 |7.61 |0.0 |1.05 |1 |0 |196 |290 |0 |618 |0 |

+-----+-----+
only showing top 10 rows
(venv) ubuntu@ip-172-31-64-111:~/airline_project$
```

Task 2: Data Processing with PySpark

Objective

The Objective of this project was to prepare the raw data into a useable format for exploratory & visualization analysis.

Data Cleaning

Basic data cleaning was performed by removing rows containing missing values using: `df.dropna()`

Data Transformation (Feature Engineering)

Two new analytical columns were engineered to enhance insight generation:

1. Total Delay Minutes

A new column **total_delay_minutes** was created by adding all individual delay components:

- Carrier delay
 - Weather delay
 - NAS delay
 - Security delay
 - Late aircraft delay

This column represents the overall delay impact per flight.

2. Total Delay Count

Another column **total delay count** was created by adding the count-based delay indicators:

- Carrier delay count
 - Weather delay count
 - NAS delay count
 - Security delay count
 - Late aircraft delay count

These engineered features provide a clear view of delay frequency and severity.

```
[venv] [venv] ubuntu@ip-172-31-44-111:~/airline_project$ nano transform_airline.py
[venv] [venv] ubuntu@ip-172-31-44-111:~/airline_project$ cat transform_airline.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

spark = SparkSession.builder.appName("Airline Transform").getOrCreate()

df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv")

# Basic cleaning
df = df.dropna()

# Feature engineering (NEW COLUMNS)
df = df.withColumn(
    "total_delay_count",
    col("carrier_ct") +
    col("weather_delay") +
    col("nas_delay") +
    col("security_delay") +
    col("late_aircraft_delay")
)

df = df.withColumn(
    "total_delay_mean",
    col("carrier_ct") +
    col("weather_ct") +
    col("nas_ct") +
    col("security_ct") +
    col("late_aircraft_ct")
)

df.printSchema()
df.show(10, truncate=False)

spark.stop()
[venv] [venv] ubuntu@ip-172-31-44-111:~/airline_project$ python transform_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to LOGGING_LEVEL.
To override this behavior use sc.setLogLevel(newLevel).
15/02/12 21:01:55 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-Java classes where applicable
root
|- year: Integer (nullable = true)
|- month: Integer (nullable = true)
|- carrier: string (nullable = true)
|- carrier_name: string (nullable = true)
|- airport: string (nullable = true)
|- tailnum: string (nullable = true)
|- arr.flights: integer (nullable = true)
|- arr.del15: integer (nullable = true)
|- carrier_ct: integer (nullable = true)
|- carrier_delay: double (nullable = true)
|- nas_ct: double (nullable = true)
|- security_ct: double (nullable = true)
|- late_aircraft_ct: double (nullable = true)
|- total_delay_mean: integer (nullable = true)
|- arr.diverted: integer (nullable = true)
|- arr.delay: integer (nullable = true)
|- carrier_delay: integer (nullable = true)
|- nas_delay: integer (nullable = true)
|- security_delay: integer (nullable = true)
|- late_aircraft_delay: integer (nullable = true)
|- total_delay_count: integer (nullable = true)
```

```

df.printSchema()
df.show(10, truncate=False)

spark.stop()

(venu) ~@ubuntubip-172-31-64-111:/airline_project$ python transform_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Set log level to DEBUG or INFO to show [log4j] logs.
To adjust logging level use sc.setLogLevel(newLevel). For Sparkr, use setLogLevel(newLevel).
20/12/12 21:01:06 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-Java classes where applicable
root
|-- year: integer (nullable = true)
|-- month: integer (nullable = true)
|-- carrier: string (nullable = true)
|-- carrier_name: string (nullable = true)
|-- arr_delay: integer (nullable = true)
|-- arr_ct: double (nullable = true)
|-- carrier_ct: double (nullable = true)
|-- weather_ct: double (nullable = true)
|-- nas_ct: double (nullable = true)
|-- security_ct: double (nullable = true)
|-- late_aircraft_ct: double (nullable = true)
|-- arr_cancelled: integer (nullable = true)
|-- arr_diverted: integer (nullable = true)
|-- arr_delay: integer (nullable = true)
|-- arr_delay_min: integer (nullable = true)
|-- total_delay: integer (nullable = true)
|-- total_delay_count: double (nullable = true)

+-----+
|year|month|carrier|carrier_name |airport|airport_name |
|delay|late_aircraft_delay|total_delay|total_delay_min|total_delay_count |
+-----+
|2022|6 |9E |Endeavor Air Inc.|AME |Allentown/Bethlehem/Easton, PA: Lehigh Valley International|136 |7 |5.95 |0.0 |0.05 |0.0 |1.0 |0 |0 |265 |222 |0 |4 |0 | | |
|29 |265 |[7.0 |Atlanta, GA: Atlanta Regional |91 |16 |7.38 |0.0 |1.04 |0.0 |6.09 |0 |0 |884 |351 |0 |0 |81 |0 |
|2022|5 |1462 |Endeavor Air Inc.|ABY |Albany, NY: Albany International |184 |15 |10.0 |0.0 |1.0 |0.0 |0.88 |1 |0 |138 |4 |0 |0 |186 |0 |
|2022|5 |155 |Endeavor Air Inc.|ACK |Nantucket, MA: Nantucket Memorial |19 |2 |0.13 |0.0 |1.0 |0.0 |0.88 |0 |0 |138 |4 |0 |0 |186 |0 |
|2022|5 |9E |Endeavor Air Inc.|JAX |Jacksonville, FL: Jacksonville International |12 |1 |0.0 |0.0 |1.0 |0.0 |0.88 |0 |0 |138 |4 |0 |0 |186 |0 |
|2022|5 |9E |Endeavor Air Inc.|JAX |Alexandria, LA: Alexandria International |88 |14 |7.26 |0.76 |1.35 |0.0 |1.64 |0 |0 |947 |585 |35 |125 |0 |
|2022|5 |89 |Endeavor Air Inc.|AGS |Augusta, GA: Augusta Regional at Bush Field |181 |19 |13.84 |0.0 |3.07 |0.0 |2.09 |0 |0 |888 |662 |0 |0 |87 |0 |
|2022|5 |89 |Endeavor Air Inc.|AGS |[19.0 |Augusta, GA: Augusta Regional at Bush Field |181 |19 |13.84 |0.0 |3.07 |0.0 |2.09 |0 |0 |888 |662 |0 |0 |87 |0 |
|2022|5 |9E |Endeavor Air Inc.|ALB |Albany, NY: Albany International |134 |18 |4.42 |1.0 |6.48 |0.0 |6.09 |5 |0 |917 |224 |78 |398 |0 |
|2022|5 |155 |Endeavor Air Inc.|ATL |Atlanta, GA: Hartsfield-Jackson Atlanta International |3842 |453 |142.96 |13.53 |186.98 |0.0 |189.53 |2 |0 |38397 |20775 |1205 |3618 |0 |
|2022|5 |12807 |Endeavor Air Inc.|ATL |[38397 |Atlanta, GA: Hartsfield-Jackson Atlanta International |3842 |453 |142.96 |13.53 |186.98 |0.0 |189.53 |2 |0 |38397 |20775 |1205 |3618 |0 |
|2022|5 |155 |Endeavor Air Inc.|ATW |Appleton, WI: Appleton International |158 |13 |4.83 |0.0 |6.05 |0.0 |1.32 |0 |0 |914 |570 |0 |0 |171 |0 |
|2022|5 |155 |Endeavor Air Inc.|AUS |Austin, TX: Austin Bergstrom International |63 |8 |0.23 |1.0 |0.89 |0.0 |0.88 |1 |0 |582 |445 |20 |19 |0 |
|2022|5 |88 |Endeavor Air Inc.|AVL |Asheville, NC: Asheville Regional |58 |14 |4.64 |0.0 |7.61 |0.0 |1.85 |1 |0 |996 |298 |0 |0 |618 |0 |
|2022|5 |88 |Endeavor Air Inc.|AVL |[14.0 |Asheville, NC: Asheville Regional |58 |14 |4.64 |0.0 |7.61 |0.0 |1.85 |1 |0 |996 |298 |0 |0 |618 |0 |

only showing top 10 rows
(venv) ~@ubuntubip-172-31-64-111:/airline_project$ 

```

Data Aggregation (Key Metrics)

Using PySpark's `groupBy` and aggregation functions, multiple metrics were computed:

- Total delay minutes by airline carrier**
- Average delay minutes by airport**
- Yearly delay trends**
- Monthly delay trends**
- Average delay minutes per carrier**

Each aggregation was sorted in descending or chronological order to support analytical interpretation.

The results were displayed using `show()` to validate correctness.

```

(venv) ~@ubuntubip-172-31-64-111:/airline_project$ nano metrics_airline.py
(venv) ~@ubuntubip-172-31-64-111:/airline_project$ cat metrics_airline.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, avg, desc

spark = SparkSession.builder.appName("Airline Metrics").getOrCreate()

df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv") \
    .dropna()

df = df.withColumn(
    "total_delay_minutes",
    df.carrier_delay + df.weather_delay + df.nas_delay +
    df.security_delay + df.late_aircraft_delay
)

# Metric 1: Total delay by carrier
m1 = df.groupby("carrier_name").agg(
    sum("total_delay_minutes").alias("total_delay")
).orderBy(desc("total_delay"))

# Metric 2: Average delay by airport
m2 = df.groupby("airport_name").agg(
    avg("total_delay_minutes").alias("avg_delay")
).orderBy(desc("avg_delay"))

# Metric 3: Yearly delay trend
m3 = df.groupby("year").agg(
    sum("total_delay_minutes").alias("yearly_delay")
).orderBy("year")

# Metric 4: Monthly delay trend
m4 = df.groupby("month").agg(
    sum("total_delay_minutes").alias("monthly_delay")
).orderBy("month")

# Metric 5: Average delay by carrier
m5 = df.groupby("carrier_name").agg(
    avg("total_delay_minutes").alias("avg_carrier_delay")
).orderBy(desc("avg_carrier_delay"))

m1.show(10, False)
m2.show(10, False)
m3.show(10, False)
m4.show(10, False)
m5.show(10, False)

spark.stop()

(venv) ~@ubuntubip-172-31-64-111:/airline_project$ 

```

```
[kennedy@kennedy-OptiPlex-5090:~/airline$ python metrics_airline.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To set the log level, use setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/12/12 21:08:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-Java classes where applicable
+-----+-----+
|carrier_name |total_delay|
+-----+-----+
|Southwest Airlines Co. |15789756
|American Airlines Inc. |15874975
|SkyWest Airlines Inc. |12643393
|Delta Air Lines Inc. |12232779
|United Air Lines Inc. |11478529
|ExpressJet Airlines Inc. |98892859
|Air Wisconsin Airlines Inc. |9281458
|JetBlue Airways |62984568
|US Airways Inc. |48388545
|Atlantic Southeast Airlines |36388279
+-----+
only showing top 10 rows
+-----+-----+
|airport_name |avg_delay |
+-----+-----+
|[Chicago, IL: Chicago O'Hare International |33867.799872901362
|Atlanta, GA: Hartsfield-Jackson Atlanta International |27838.672919588646
|Dallas/Ft. Worth, TX: Dallas/Ft. Worth International |17879.875000000008
|Hawaii, HI: Newark Liberty International |17879.69181785626
|[San Francisco, CA: San Francisco International |16148.159823668639
|Seattle-Tacoma International |15982.973445143256
|Denver, CO: Denver International |15825.973445143256
|New York, NY: John F. Kennedy International |14642.615225939283
|Los Angeles, CA: Los Angeles International |13894.449129852749
|Houston, TX: George Bush Intercontinental/Houston |13379.482711118187
+-----+
only showing top 10 rows
+-----+-----+
|year|yearly_delay|
+-----+-----+
|[2003|325385792
|2004|72888525
|2005|100000000
|2006|87187939
|2007|84453792
|2008|86437929
|2009|6698519
|2010|72888525
|2011|61678951
|2012|56916492
+-----+
only showing top 10 rows
+-----+-----+
|month|monthly_delay|
+-----+-----+
|[1|118991145
|2|100000000
|3|189732748
|4|97405291
|5|12888525
|6|142912186
|7|158259854
|8|12888525
|9|78857928
|10|8993226
+-----+
only showing top 10 rows
+-----+-----+
|carrier_name |avg_carrier_delay|
+-----+-----+
|Southwest Airlines Co. |11542.82139288075
|American Airlines Inc. |8829.828788981285
|United Air Lines Inc. |5979.19846367826
|JetBlue Airways |5833.9725820577
|Delta Air Lines Inc. |4574.97318861079
|US Airways Inc. |4518.97258208864
|Compass Airlines Inc. |4383.97258205223
|Spirit Air Lines |4383.97258205223
|American Eagle Airlines Inc. |4483.87684265235
|ExpressJet Airlines Inc. |4483.87684265235
+-----+
only showing top 10 rows
(kennedy@kennedy-OptiPlex-5090:~/airline$ ]
```

Task 3: Store Processed Data Back to Amazon S3

Objective

This task aims to Persist a Processed Dataset in a Structured Format so that the Processed Dataset can be reused for Analytics, Visualisation, and Machine Learning Tasks.

Implementation

The Cleaned and Enriched Dataset was exported as a file after Feature Engineering has Completed, by using Spark's Distributed Write Capabilities.

The two resulting Output Formats are:

The Parquet Format - Optimise Storage and Query Performance.

The CSV Format - To work with Visualisation/BI Tools.

The Processed Files were Written out in Overwrite mode so they can always be Reproduced. The Processed Output Files were uploaded to a Specific Location in S3 (the processed/ folder) so they can be Accessed by Services Such as Athena, QuickSight, PowerBI, and SageMaker.

Validation

Storage of Success was confirmed by:

- Checking the Output Files in Local Directory
- Uploading to S3
- Querying the Processed Dataset later via Amazon Athena.

```

(venv) ubuntu@ip-172-31-44-111:~/airline_project$ nano save_to_s3.py
[venv] ubuntu@ip-172-31-44-111:~/airline_project$ cat s
save_to_s3.py sql_airline.py
[venv] ubuntu@ip-172-31-44-111:~/airline_project$ cat save_to_s3.py
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
spark = SparkSession.builder.appName("Save Airline Data").getOrCreate()
df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay_Cause.csv") \
    .dropna()
# Feature engineering
df = df.withColumn(
    "total_delay_minutes",
    col("carrier_delay") +
    col("weather_delay") +
    col("nas_delay") +
    col("security_delay") +
    col("late_aircraft_delay"))
df = df.withColumn(
    "total_delay_count",
    col("carrier_ctt") +
    col("weather_ctt") +
    col("nas_ctt") +
    col("security_ctt") +
    col("late_aircraft_ctt"))
# Save locally (Spark output)
df.write.mode("overwrite").parquet("output/airline_processed_parquet")
df.write.mode("overwrite").option("header", True).csv("output/airline_processed_csv")

spark.stop()
[venv] ubuntu@ip-172-31-44-111:~/airline_project$ python save_to_s3.py
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j2-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/12/12 21:16:12 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
[venv] ubuntu@ip-172-31-44-111:~/airline_project$ ls
data ingest_airline.py metrics_airline.py output save_to_s3.py sql_airline.py transform_airline.py venv
[venv] ubuntu@ip-172-31-44-111:~/airline_project$ ls output
airline_processed_csv airline_processed_parquet

```

Task 4: Data Analysis Using Spark SQL

Objective

This task aimed to derive analytical insights from the processed dataset using Spark SQL.

Implementation

The cleaned dataset was registered as a temporary SQL view named `airline`, enabling SQL-based analysis.

5 analytical SQL queries were executed:

1. Total arrival delay by carrier

- Identified carriers with the highest cumulative arrival delays.

2. Average arrival delay by airport

- Highlighted airports experiencing frequent or severe delays.

3. Yearly delay trends

- Analyzed how total delays changed across different years.

4. Monthly delay trends

- Revealed seasonal patterns in airline delays.

5. Flight record count by carrier

- Measured operational volume across airlines.

Each query was executed using `spark.sql()` and results were displayed for validation.

```

(venv) ubuntu@ip-172-31-44-111:/airline_project$ nano sql_airline.py
(venv) ubuntu@ip-172-31-44-111:/airline_project$ cat sql_airline.py
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Airline SQL").getOrCreate()

df = spark.read.option("header", True).option("inferSchema", True) \
    .csv("data/Airline_Delay.Cause.csv") \
    .dropna()

df.createOrReplaceTempView("airline")

queries = [
    """SELECT carrier_name, SUM(arr_delay) AS total_arrival_delay
    FROM airline GROUP BY carrier_name ORDER BY total_arrival_delay DESC""",
    """SELECT airport_name, AVG(arr_delay) AS avg_arrival_delay
    FROM airline GROUP BY airport_name ORDER BY avg.arrival_delay DESC""",
    """SELECT year, SUM(arr_delay) AS yearly_delay
    FROM airline GROUP BY year ORDER BY year""",
    """SELECT month, SUM(arr_delay) AS monthly_delay
    FROM airline GROUP BY month ORDER BY month""",
    """SELECT carrier_name, COUNT(*) AS records
    FROM airline GROUP BY carrier_name ORDER BY records DESC"""
]

for q in queries:
    spark.sql(q).show(10, False)

spark.stop()

```

(venv) ubuntu@ip-172-31-44-111:/airline_project\$ python sel_airline.py

WARNING: Using incubator modules: jdk.incubator.vector

Using Spark's default log4j profile org/apache/spark/log4j2-defaults.properties

Set log level: INFO, WARN, ERROR, or FATAL

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

25/12/22 21:18:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

carrier_name	total_arrival_delay
Southwest Airlines Co.	197597648
AirTran Airways Inc.	150000000
Skymost Airlines Inc.	1266539803
Delta Air Lines Inc.	1250000000
United Air Lines Inc.	114776200
ExpressJet Airlines Inc.	98892859
American Eagle Airlines Inc.	62841558
JetBlue Airways	48388565
US Airways Inc.	34588279
Alaska Southwest Airlines	24588279

only showing top 10 rows

airport_name	avg_arrival_delay
(Chicago, IL) Chicago O'Hare International	31867.799728951362
(Dallas, TX) Dallas/Fort Worth International	28671.237798546288
(Dallas/Fort Worth, TX) Dallas/Fort Worth International	28671.237798546288
(Honolulu, HI) Honolulu International	150000000
(San Francisco, CA) San Francisco International	15449.15923466439
(Sanford, FL) Orlando Sanford International	15681.452838618678
(Denver, CO) Denver International	150000000
(New York, NY) John F. Kennedy International	14642.515226933283

only showing top 10 rows

year/yearly_delay
2003/32585792
2004/32585792
2005/32585792
2006/74476224
2006/87187699
2007/88637723
2008/86637993
2009/66895819
2010/66895819
2011/65470541
2012/65916092

only showing top 10 rows

month/monthly_delay
1/118991145
2/16819866
3/16819866
4/99452915
5/18849459
6/18849459
7/147857638
8/78657287
9/78657287
10/89932256

only showing top 10 rows

carrier_name	records
Skywest Airlines Inc.	340938
Delta Air Lines Inc.	27518
ExpressJet Airlines Inc.	24487
American Eagle Airlines Inc.	19798
United Air Lines Inc.	19986
Southeast Airlines	15399
American Eagle Airlines Inc.	15399
Mess Airlines Inc.	14483
AirTran Airways Inc.	13386
Frontier Airlines Inc.	12275

only showing top 10 rows

Task 5: Machine Learning with AWS SageMaker Autopilot

Objective

This project aimed to utilize AWS SageMaker Autopilot to facilitate machine learning through an automated workflow (including all steps of training, evaluation, and comparison) using preprocessed airline delay records stored in the AWS S3 Bucket. Automation meant that no programming was required. The expected outcome was for every model to provide predictions for the total delay in minutes for flights based on recorded historical flight and delays and for each model to be evaluated based on interpretability, performance, and ethicality.

i) Importing Processed Data from S3

After completing data cleaning, feature engineering, and aggregation using PySpark, the processed dataset was exported to Amazon S3 in CSV format.

- **S3 Location:**
`s3://rg422-bigdata/processed/airline_quicksight/`
- The dataset contained **317,261 rows** and **23 columns**, including:
 - Engineered features such as `year`, `month`, `total_delay_minutes`, and `total_delay_count`
 - Delay-related attributes like `arr_delay`, `carrier_delay`, `nas_delay`, `weather_delay`, and `late_aircraft_delay`

This processed dataset was then imported into **AWS SageMaker Autopilot (Canvas)** directly from S3 using the **Import Dataset** option.

Outcome:

The dataset was successfully loaded and validated by SageMaker Autopilot and making it ready for model training.

The screenshot shows the AWS SageMaker Autopilot Canvas interface. On the left, there's a sidebar with icons for Home, Amazon Q, Data Wrangler, Datasets, My Models, ML Ops, Ready-to-use, Help, and Log out. The main area shows a breadcrumb path: My models > Model_20251213_030002 > Version 1. Below this, there are tabs for Select, Build, Analyze, Predict, and Deploy, with 'Select' being the active tab. A message通知 says: "There's a new way to join data. You can now join datasets by creating a data flow in Data Wrangler. Learn how to join data". To the right of the message is a button labeled "Import and prepare" with a close icon. The main content area is titled "Select dataset" and contains a table with one row. The table has columns: Name, Columns, Rows, Cells, Created, and Status. The single row shows: Dataset_20251213_030002, 23, 317,261, 7,297,003, 12/12/2025 10:00 PM, and Ready. There's also a small thumbnail image next to the dataset name.

ii) Running the Autopilot Experiment

Target Column Selection

- **Target variable:** *total_delay_minutes*
- **Problem type:** Numeric prediction (regression)

SageMaker Autopilot automatically analyzed the dataset and identified this as a **regression problem** because the target column represents continuous numeric values.

Model Training

Autopilot has been configured with the following capabilities:

- Divide the original data into training and testing sets
- Train several regression algorithms using multiple methods
- Optimize hyper-parameters for optimal results
- Select the optimal model through evaluation of errors

The **Quick Build** option was used to efficiently train models while minimizing configuration overhead.

iii) Model Evaluation and Results

Key Performance Metrics

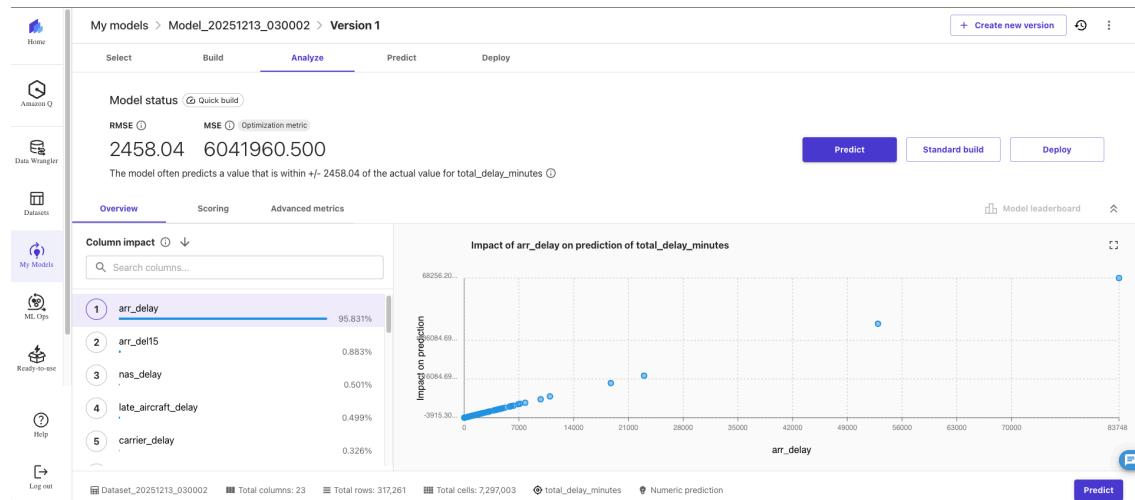
The best-performing model achieved the following results:

Metric	Value
RMSE	2458.04
MSE	6,041,960.50
MAE	±168.72
R² Score	96.24%

- The **R² score of 96.24%** shows how well the total delay minute's model can predict variance.
- The **RMSE score of 2458 minutes** shows the model's predictive accuracy is generally estimated within ±2458 minutes of the actual total delay.

Predicted vs Actual Analysis

The **Predicted vs Actual** visualization showed a strong linear relationship between predicted and actual values, confirming good model fit and low systematic bias for most observations.



iv) Feature Importance and Interpretability

SageMaker Autopilot automatically generated feature insights:

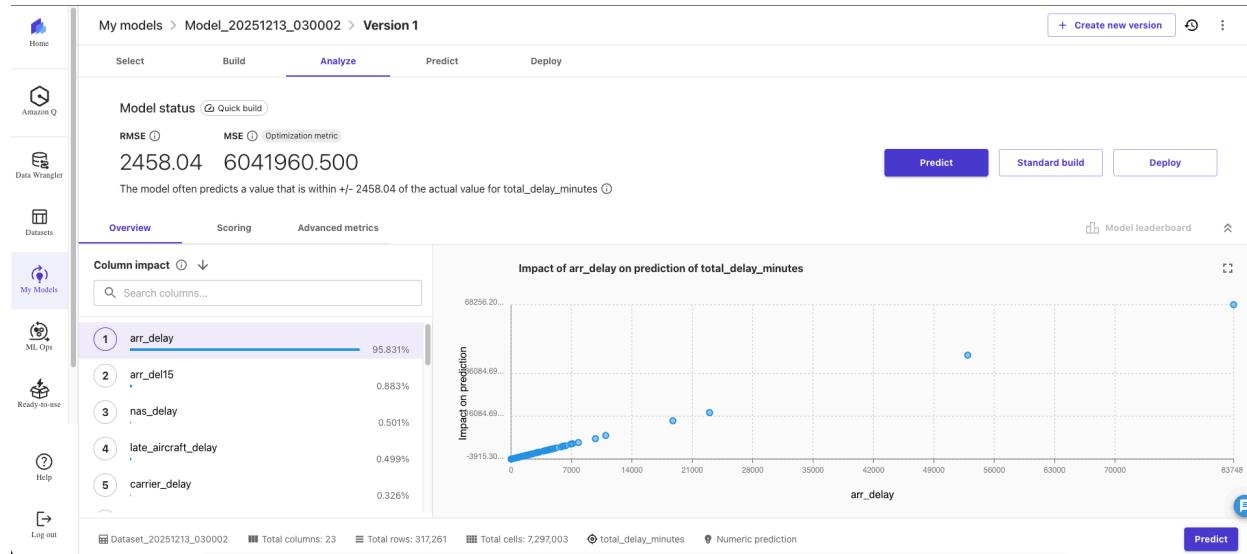
Top contributing features:

1. *arr_delay* (*Arrival delay*) – ~95.8% impact
2. *nas_delay*
3. *late_aircraft_delay*
4. *carrier_delay*

This confirms that **arrival delays** are the strongest predictor of total delay minutes, which aligns with real-world airline operations.

Residual and Error Analysis

- The residual plots show how most of the errors clustered around zero for most predictions.
- Large and rare extreme delay scenarios (e.g., due to weather, cascading delays) have some significant outliers.



v) Making Predictions

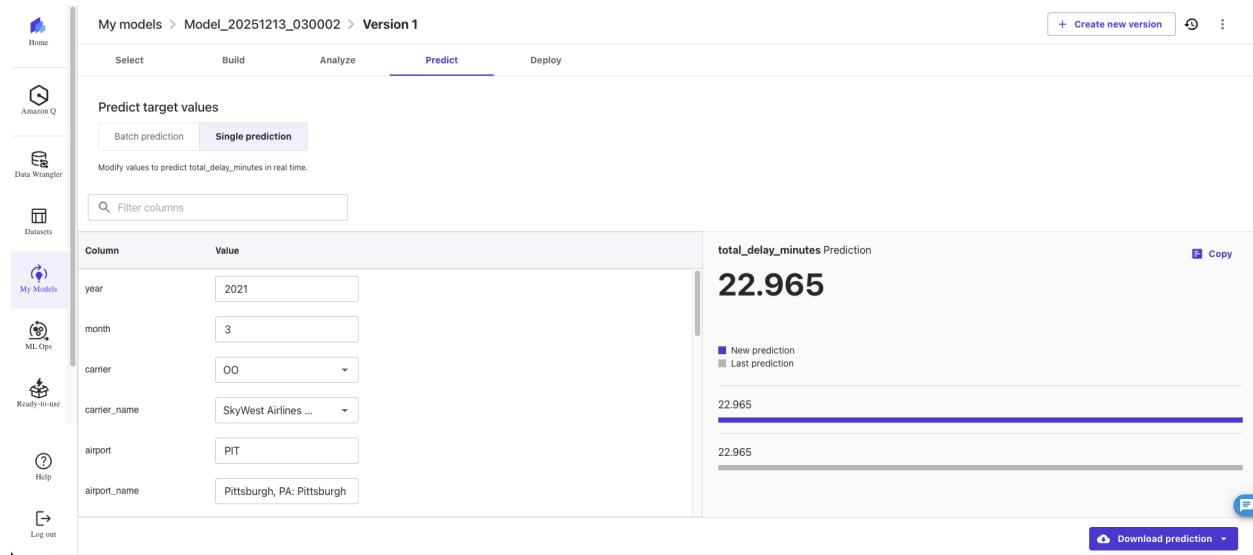
Using the **Predict → Single Prediction** feature in SageMaker Autopilot, real-time predictions were generated by manually entering feature values such as:

- Year
- Month
- Carrier
- Airport
- Delay-related features

Example Prediction Output:

- Predicted *total_delay_minutes*: **22.965 minutes**

This demonstrated that the trained model can be used for **what-if analysis** and operational forecasting.



vi) Ethical Considerations

Bias

Airline historical data may introduce bias into the model, for example:

- The model may overly represent major airports
- Certain carriers have higher historical data

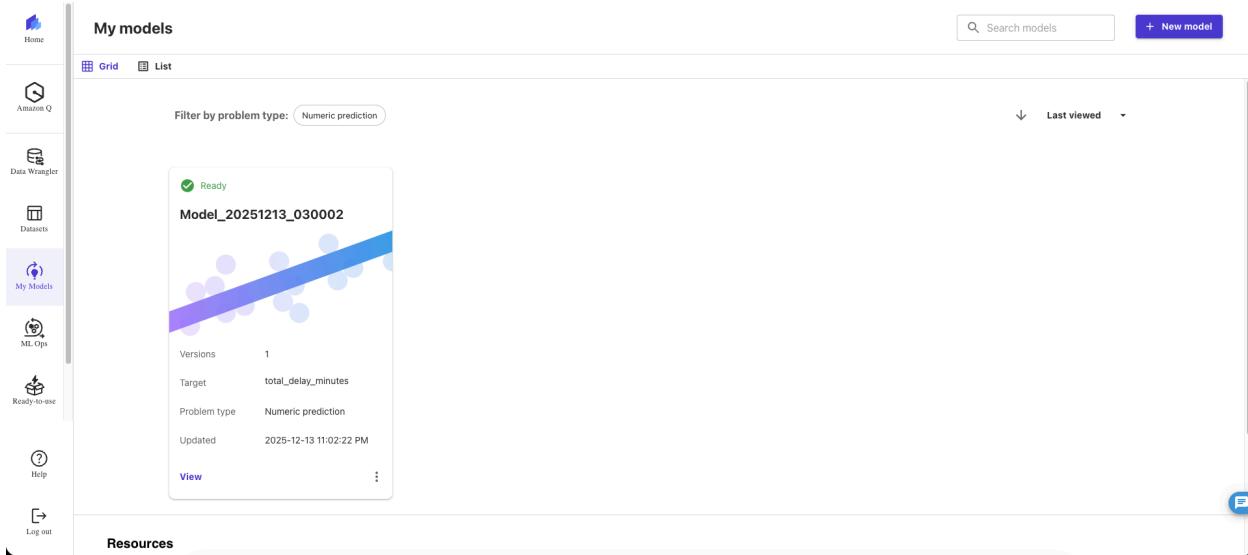
Biases will impact the model's prediction accuracy for smaller airports or less frequently flown routes.

Privacy

- The dataset contained **no personally identifiable information (PII)**.
- Operational flight and delay metrics were used as inputs to the model, so privacy best practices are followed.

Mitigation

- The model is continually retrained with new, updated data.
- Prediction errors are monitored by carrier and airport.
- Sensitive or irregular level attributes of passengers will not be used in the model.



5. Visualization

5.1 Visualization Tool Selection

Interactive dashboards were developed to provide meaningful insights from the processed Airline Delay cause dataset stored in Amazon S3 as the primary focus of this component of the overall project.

Initially, we looked at using AWS QuickSight for the visualizing component of the project; however, we used **Microsoft Power BI** instead due to difficulties related to setting up accounts in AWS. I encountered multiple errors while creating and validating a QuickSight account on the AWS Free Tier; these issues are outlined and explained in the video that accompanied this project.

To finish this project on time and still meet all visualization requirements, we decided to use Power BI as an alternate visualization platform, which was permitted by the project requirements, to avoid accruing additional charges to our AWS account.

Justification:

Power BI offers a wide variety of data connectivity options; it is also capable of connecting to very large datasets; it has a scheduling feature that allows it to auto-refresh itself and generate reports automatically; and it allows users to create professional-grade interactive dashboards that can be applied to big data analysis.

5.2 Data Transfer and Refresh Validation

Data acquired over the period is available in Amazon S3 after being imported into Power BI, which has thus enabled this information to be used in various types of Analytics tools designed to track various aspects of the Flight Industry. PySpark was used to perform calculations on the airline data created and uploaded into Amazon S3.

The processed data in csv format was manually uploaded to Power BI .As a validation of the Data Pipeline; a manual refresh of the data connection was completed to verify that all the latest data residing in Amazon S3 was being properly imported into Power BI and reflected within all current Dashboards residing within Power BI.

5.3 Dashboard Design and Visualizations

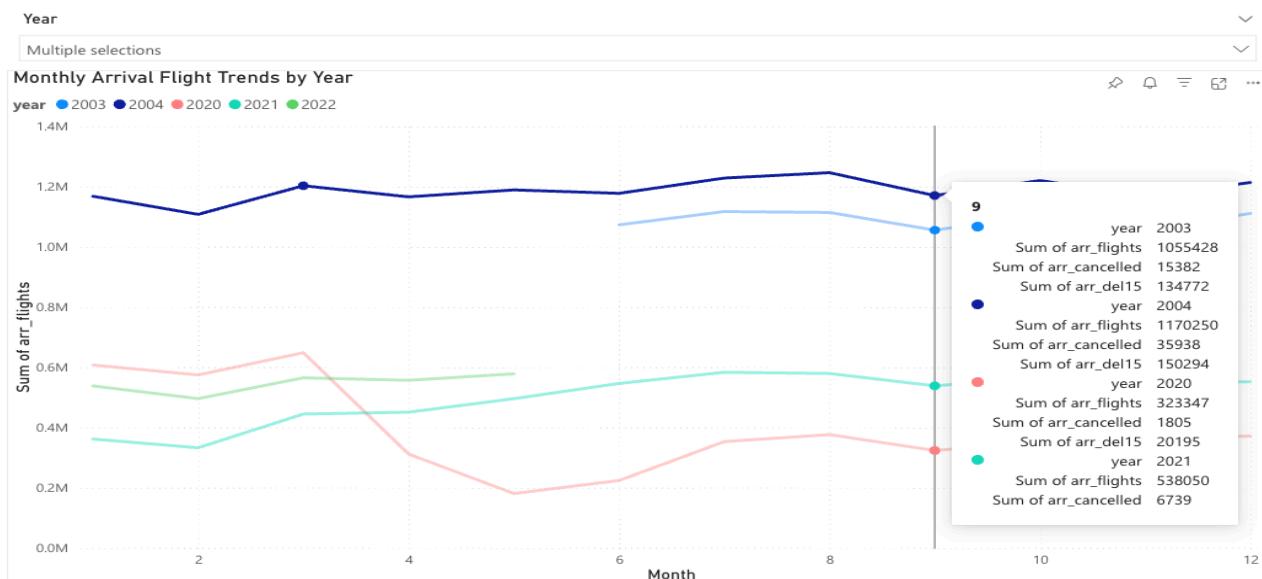
An extensive Report which contained an array of four or more useful visual tools was developed using the Features available in Power BI. Each Visualization covered a separate analytical area of focus regarding airline performance, delays and reliability.

Visualization 1: Monthly Arrival Flight Trends by Year

Overview: This Line Graph displays the breakdown of Arrival Flights by month across a series of years (2003, 2004, 2020, 2021, 2022). This Yearly Breakdown provides an opportunity for users to identify and study how the various factors affecting airline performance will change from season to season and year to year. Users are also able to clearly see the effects of large scale external factors, particularly the COVID-19 Pandemic. We can select the required year from year dropdown at the top of the chart

Findings:

- The Year 2020 saw the lowest flight volume totals ever.
- The Yearly Volume Totals began to gradually increase over the subsequent years.
- The Mid July/August Summer Peak continues every year.



Visualization 2: Top 10 Airports by Arrival Flights

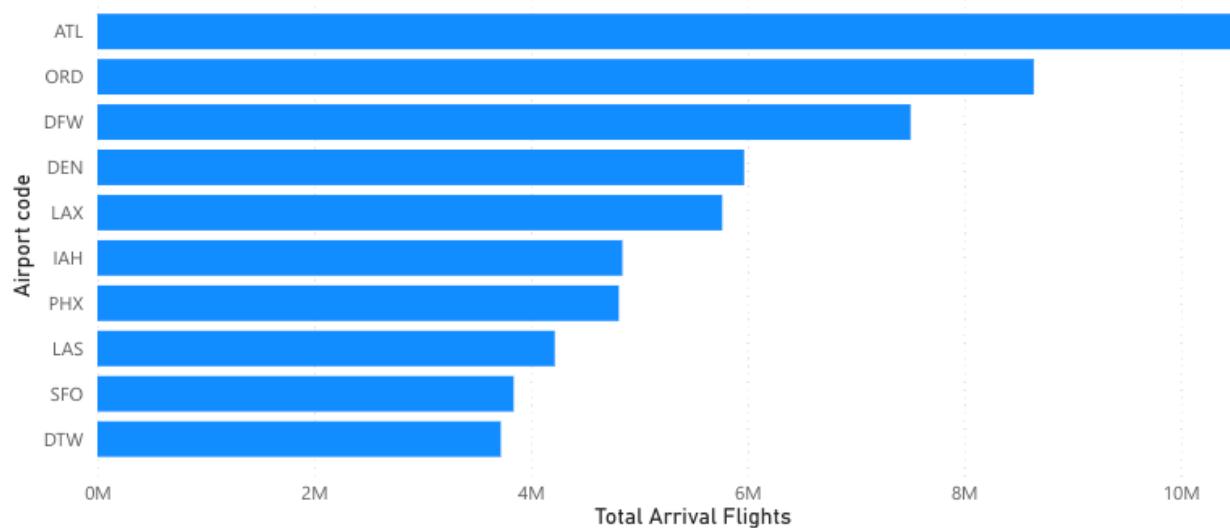
Overview:

A horizontal bar chart provides a ranking of the top 10 airports based on overall number of arrivals.

Insights:

Airports like ATL, ORD & DFW control most inbound flights. These three airports have a large share of the total number of inbound flights nationwide.

Top 10 Airports by Arrival Flights



Visualization 3: Delay Causes Breakdown for Top 10 Airports

Overview:

A stacked bar chart provides the delay causes across the top airports in the United States.

The chart includes:

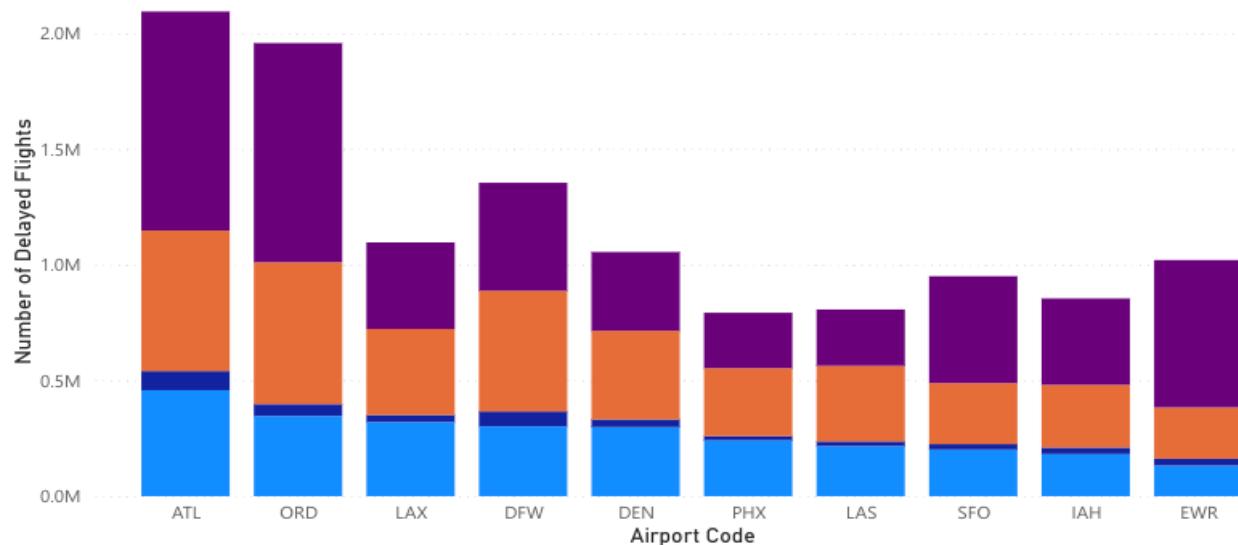
- Carrier-related delays
- Weather-related delays
- Late aircraft delays
- Air Traffic Control (National Airspace System (NAS)) delays

Insights:

Carrier & late aircraft delays are the predominant delays. Weather delays are variable across airports.

Delay Causes Breakdown for Top 10 Airports

● Carrier-Related Delays ● Weather-Related Delays ● Late Aircraft Delays ● Air Traffic Control (NAS) Delays



Visualization 4: Airline Reliability Analysis – Cancellations vs Delays

Overview:

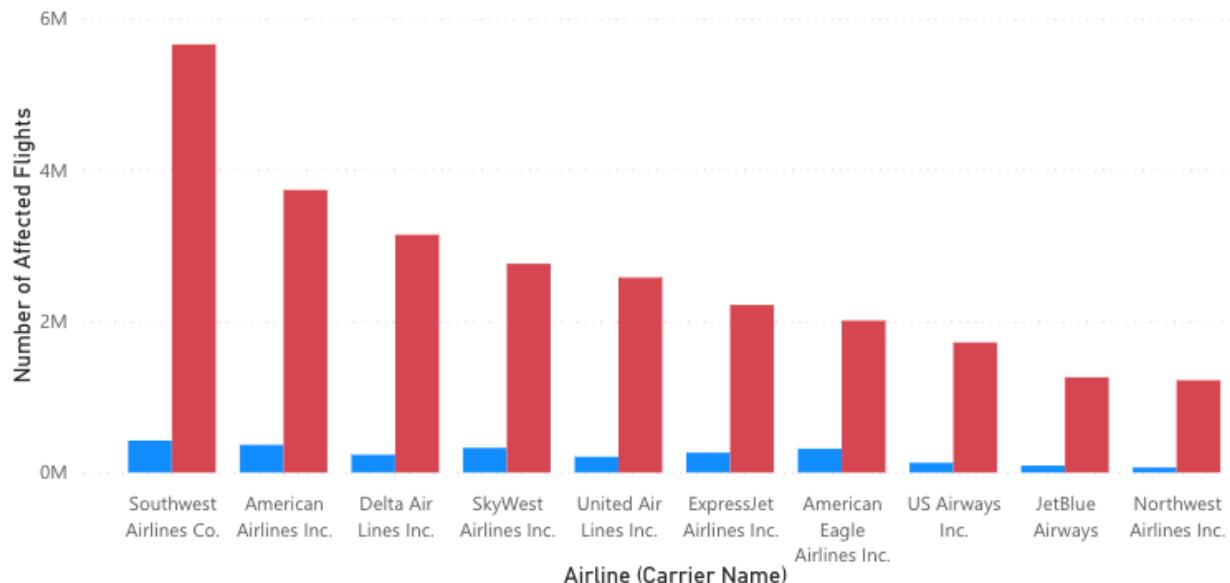
- A comparison of cancellations versus flights delayed 15 minutes or more for the major airline carriers in the U.S.
- The chart allows for easy comparison of reliabilities between the airline carriers.

Insights:

- There are multiple airlines with far fewer cancellations and considerably more delays than expected.
- Reliability is inconsistent across airlines.

Airline Reliability Analysis: Cancellations vs Delays

● Cancelled Flights ● Flights Delayed 15+ Minutes



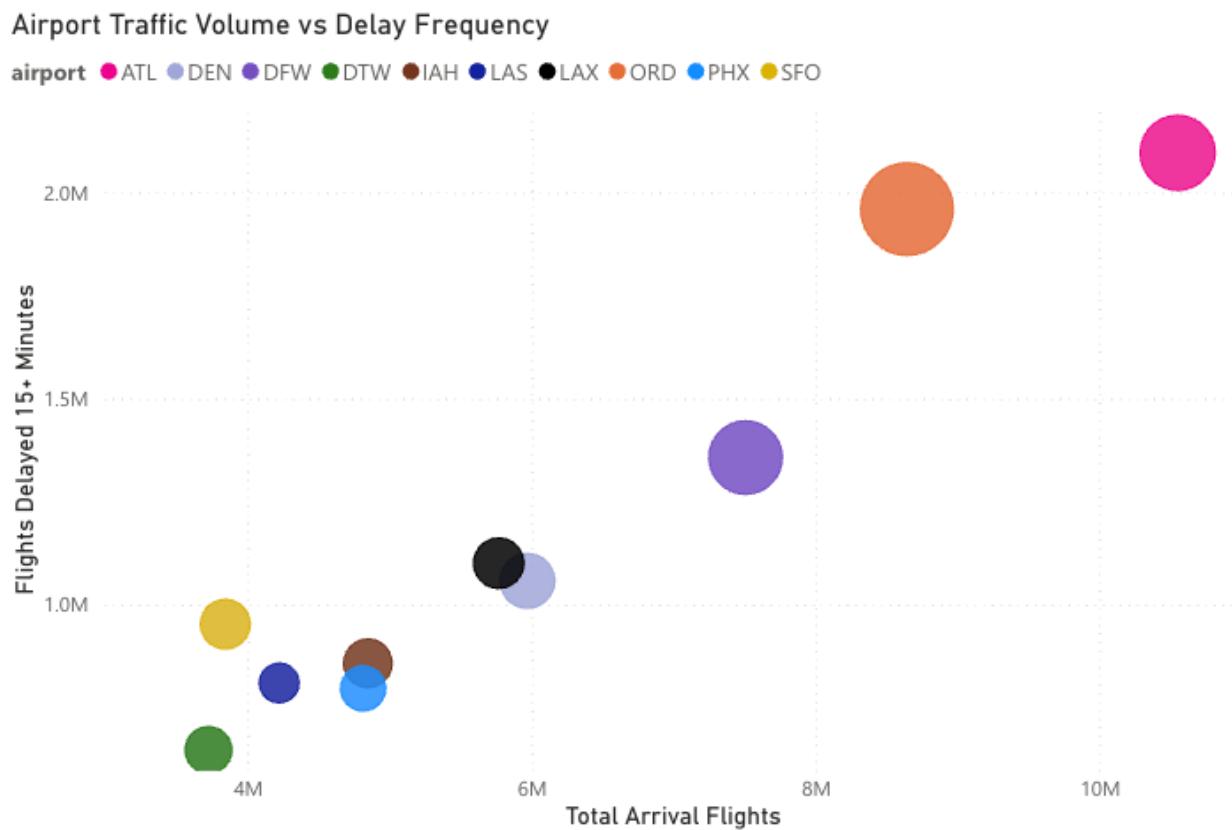
Visualization 5: Airport Traffic Volume vs Delay Frequency

Overview:

A bubble scatterplot depicts the relationship between total incoming flights to a given airport and the average frequency of delays for that airport.

Insights:

- Higher volume airports have a correspondingly higher number of delays.
- Certain airports can demonstrate higher levels of operational efficiencies while operating at a higher volume.



6.Bonus Task: Automation of the Pipeline

Objective

The aim of this extra assignment is to create a completely self-sufficient method of running the Big Data Pipeline from end-to-end. Specifically, we will cover data ingestion from Amazon S3 through to executing distributed processing of that data, putting the findings back into Amazon S3, sending an email notification of when it has been executed and providing for automatic refreshing of the visualisation dashboard.

This automation allows us to execute the pipeline with no manual input, providing us with an increased level of scalability, greater reliability and a greater ability to perform Real-Time Analytics on our data.

6.1 Automation Architecture Overview

The automated pipeline follows the architecture below:

Amazon S3 (Raw Data) → AWS Lambda Trigger → AWS Systems Manager (SSM) → EC2 PySpark Pipeline → Processed Data Stored in S3 → SNS Email Notifications → Power BI Dataset Refresh

This architecture eliminates manual SSH execution and enables serverless orchestration using AWS managed services.

6.2 Automation Script (1 Point)

Approach

A single automation script (`run_pipeline.sh`) is executed on an EC2 instance that hosts the PySpark environment.

This script performs the following steps automatically:

1. Reads raw airline delay data from Amazon S3
2. Executes PySpark transformations and aggregations
3. Writes processed outputs in **CSV and Parquet formats** back to Amazon S3
4. Stores detailed execution logs in S3 for auditing and debugging

Pipeline Execution Location

- EC2 Instance: `rg422-Airline-PySpark (t3.micro)`
- Pipeline Directory: `/home/ubuntu/pipeline/`
- Logs Directory: `/home/ubuntu/pipeline/logs/`

Verified Outputs in S3

- `s3://rg422-bigdata/processed/airline_processed_csv/`
- `s3://rg422-bigdata/processed/airline_processed_parquet/`
- `s3://rg422-bigdata/processed/analytics/logs/`

```

#!/usr/bin/env bash
#!/usr/bin/env bash
set -eu pipefail

# ===== CONFIG =====
BUCKET="rgw422-bigdata"
PROCESSING="true"
ANALYTICS_PREFIX="analytics"
LOGS_PREFIX="logs"

# SNS Topic ARN (TOPIC ARN, not subscription ARN)
TOPICARN="arn:aws:sns:us-east-1:6882614197:pipeline-status-topic"

# Spark submit path (from your venv)
SPARK_SUBMIT="$HOME/airline_project/venv/bin/spark-submit"

# Timestamp
TS=$(date "+%Y-%m-%d_%H-%M-%S")

# Local log location
LOG_DIR="$HOME/pipeline/logs"
LOG_LOCAL="$LOG_DIR/pipeline_STS.log"

# S3 log location
S3_LOG_PATH="s3://$BUCKET/$ANALYTICS_PREFIX/$LOGS_PREFIX/pipeline_STS.log"

# ===== PREP =====
mkdir -p "$LOG_DIR"

# Redirect all $stdout & $stderr to log + console
exec > $(tee -a "$LOG_LOCAL") 2>1

# ===== FAILURE HANDLER =====
trap 'aws sns publish \
--topic-arn "$TOPICARN" \
--subject "Pipeline Failed" \
--message "Pipeline FAILED at $date". \
Check logs at $S3_LOG_PATH' \
ERR

echo "===== Pipeline started at $date"
echo "■■■ Pipeline started at $date"
echo "===== Pipeline completed successfully"
echo "■■■ Pipeline completed successfully"
cd "$HOME/airline_project"

# ===== PIPELINE STEPS =====
echo "[1/8] Ingesting raw data..." \
"$SPARK_SUBMIT" ingest_airline.py

echo "[2/8] Transforming data..." \
"$SPARK_SUBMIT" transform_airline.py

echo "[3/8] Computing metrics..." \
"$SPARK_SUBMIT" metrics_airline.py

echo "[4/8] Running Spark SQL analysis..." \
"$SPARK_SUBMIT" sql_airline.py

```

```

echo "[5/8] Saving processed data to S3..." \
"$SPARK_SUBMIT" save_to_s3.py

# ===== UPLOAD LOG =====
echo "Uploading logs to S3..."
aws s3 cp "$LOG_LOCAL" "$S3_LOG_PATH"

# ===== SUCCESS NOTIFICATION =====
aws sns publish \
--topic-arn "$TOPICARN" \
--subject "Pipeline Success" \
--message "Pipeline completed SUCCESSFULLY at $date". \
Check logs at $S3_LOG_PATH"

log
$S3_LOG_PATH

echo "===== Pipeline completed successfully"
echo "■■■ Pipeline completed successfully"
echo "===== Pipeline completed successfully"
http://172-31-44-131:40404/

```

```

(venv) ubuntu@ip-172-31-44-111:~/pipeline$ cd ~/airline_project/
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ ls
airline_pipeline_input.csv  ingest_airline.py  metrics_airline.py  save_to_s3.py  transform_airline.py
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ chmod +x ~/pipeline/run_pipeline.sh
(venv) ubuntu@ip-172-31-44-111:~/airline_project$ ./pipeline/run_pipeline.sh
Pipeline started at Sat Dec 13 22:56:35 UTC 2025

(1/8) Ingesting raw data...
20/12/13 22:56:39 INFO SparkContext: Running Spark version 3.1.1
20/12/13 22:56:39 INFO SparkContext: OS info Linux, 5.4.0-1018-aws, amd64
20/12/13 22:56:39 INFO SparkContext: Java version 17.0.17
20/12/13 22:56:39 INFO SparkContext: User libjars: Unable to load native-hadoop library for your platform... using builtin-Java classes where applicable
20/12/13 22:56:39 INFO ResourceUtils: 
20/12/13 22:56:39 INFO ResourceUtils: No custom resources configured for spark.driver.
20/12/13 22:56:39 INFO ResourceUtils: 
20/12/13 22:56:39 INFO SparkContext: Submitted application: Airline Delay Analysis
20/12/13 22:56:39 INFO DefaultResourceProfile: Default ResourceProfile created, executor resources: Map[cores --> name: cores, amount: 1, script: , vendor: , memory --> name: memory, amount: 1024, script: , vendor: , offHeap --> name: offHeap, amount: 0, script: , vendor: ], tasks: Map[taskType --> name: taskType, amount: 1]
20/12/13 22:56:39 INFO ResourceProfile: Limiting resource is cpu
20/12/13 22:56:39 INFO ResourceProfile: Added ResourceProfile id: 0
20/12/13 22:56:39 INFO SecurityManager: Adding ResourceProfile id: 0
20/12/13 22:56:39 INFO SecurityManager: view permissions: ubuntu; groups with view permissions: EMPTY; users with modify permissions: ubuntu; groups with modify permissions: EMPTY
20/12/13 22:56:39 INFO SecurityManager: authentication disabled; ui acl's disabled; users with view permissions: ubuntu; groups with view permissions: EMPTY; users with modify permissions: ubuntu; groups with modify permissions: EMPTY
20/12/13 22:56:39 INFO SecurityManager: sparkDriverOnPortDisabledForDriver on port 37185.
20/12/13 22:56:39 INFO SparkEnv: Registering MapOutputTracker
20/12/13 22:56:39 INFO SparkEnv: Registering BlockManagerMaster
20/12/13 22:56:39 INFO SparkEnv: Registering BlockManagerMasterEndpoint: BlockManagerMasterEndpoint@0:spark.storage.DefaultTopologyMapper for getting topology information
20/12/13 22:56:39 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
20/12/13 22:56:39 INFO SparkEnv: Registering BlockManagerMasterEndpoint@0:spark.storage.BlockManagerMasterEndpoint@0
20/12/13 22:56:39 INFO MemoryStore: MemoryStore started with capacity 413.9 MiB
20/12/13 22:56:40 INFO SparkEnv: Registering OutputCommitCoordinator
20/12/13 22:56:40 INFO Executor: Executor ID: e2
20/12/13 22:56:40 INFO Executor: Created executor service 'SparkU' on port 49408.
20/12/13 22:56:40 INFO Executor: Successfully started service 'SparkU' on port 49408.
20/12/13 22:56:40 INFO Executor: Executor successfully executed ID driver on host ip-172-31-44-111.ec2.internal
20/12/13 22:56:40 INFO Executor: Executor successfully updated its execId to 0
20/12/13 22:56:40 INFO Executor: Executor successfully registered with BlockManager
20/12/13 22:56:40 INFO Executor: Java version: 17.0.17
20/12/13 22:56:40 INFO Executor: Executor successfully registered with user classpath (userClasspathInitialized = false); '
20/12/13 22:56:40 INFO Executor: Created and updated rep. class loader org.apache.spark.util.MutableURLClassLoader@38eac1fd for default.
20/12/13 22:56:40 INFO Util: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 44475.
20/12/13 22:56:40 INFO BlockManagerMaster: Registering BlockManagerBlockManagerMaster@ip-172-31-44-111.ec2.internal:44475
20/12/13 22:56:40 INFO BlockManagerMaster: Registering BlockManagerBlockManagerMaster@ip-172-31-44-111.ec2.internal:44475, None
20/12/13 22:56:40 INFO BlockManagerMaster: Registered BlockManagerBlockManagerMaster@ip-172-31-44-111.ec2.internal:44475, None
20/12/13 22:56:40 INFO BlockManager: Initialized BlockManager: BlockManager@ip-172-31-44-111.ec2.internal:44475, None
20/12/13 22:56:41 INFO SharedState: Warehouse path is '/file:/home/ubuntu/airline_project/jar/warehouse'.
20/12/13 22:56:42 INFO InMemoryFileIndex: It took 63 ms to list leaf files for 1 paths.
20/12/13 22:56:42 INFO InMemoryFileIndex: It took 63 ms to list leaf files for 1 paths.
20/12/13 22:56:45 INFO FileSourceWriteStrategy: Pushed Filters:
20/12/13 22:56:45 INFO FileSourceWriteStrategy: Converting filters: length(filter(value#, None)) > 0
20/12/13 22:56:45 INFO FileSourceWriteStrategy: Converting filters: 273.1k4409 ms
20/12/13 22:56:46 INFO MemoryStore: Block broadcast_0 stored as value in memory (estimated size 199.6 KiB free 413.7 MiB)
20/12/13 22:56:46 INFO MemoryStore: Block broadcast_0 stored as value in memory (estimated size 199.6 KiB free 413.7 MiB)
20/12/13 22:56:46 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on ip-172-31-44-111.ec2.internal:14476 (size: 36.4 KiB, free: 413.9 MiB)
20/12/13 22:56:47 INFO SparkContext: Starting job csv at NativeMethodAccessoImpl.java:19
20/12/13 22:56:47 INFO DAGScheduler: Got job 0 (csv at NativeMethodAccessoImpl.java:19) with 1 output partitions
20/12/13 22:56:47 INFO DAGScheduler: Final stage: ResultStage 0 (csv at NativeMethodAccessoImpl.java:19)

```

```

20/12/13 22:58:27 INFO DAGScheduler: Got job 3 (few as NativeMethodAccessorImpl.java:8) with 2 output partitions
20/12/13 22:58:27 INFO DAGScheduler: Final stage: ResultStage 3 (at NativeMethodAccessorImpl.java:8)
20/12/13 22:58:27 INFO DAGScheduler: Parents of final stage: List()
20/12/13 22:58:27 INFO DAGScheduler: Missing parents: List()
20/12/13 22:58:27 INFO MemoryStore: Block broadcast_7 stored as value in memory (estimated size 229.3 Kib, free 422.9 Mib)
20/12/13 22:58:27 INFO MemoryStore: Block broadcast_7 stored as bytes in memory (estimated size 88.6 Kib, free 422.9 Mib)
20/12/13 22:58:27 INFO SparkContext: Created broadcast 7 from broadcast at DAGScheduler, scale:1885
20/12/13 22:58:27 INFO TaskSchedulerManager: Submitting 2 missing tasks from ResultStage 3 (MapPartitionsRDD1@7) at csv at NativeMethodAccessorImpl.java:8 (first 15 tasks are for partitions Vector(0, 1))
20/12/13 22:58:27 INFO TaskSchedulerManager: Starting task 0.0 in stage 3.0 (TID 5) (ip=172-31-44-111.ec2.internal, executor driver, partition 0, PROCESS_LOCAL, 8242 bytes)
20/12/13 22:58:27 INFO TaskSchedulerManager: Starting task 1.0 in stage 3.0 (TID 6) (ip=172-31-44-111.ec2.internal, executor driver, partition 1, PROCESS_LOCAL, 8242 bytes)
20/12/13 22:58:27 INFO Executor: Running task 0.0 in stage 3.0 (TID 5)
20/12/13 22:58:27 INFO Executor: Running task 1.0 in stage 3.0 (TID 6)
20/12/13 22:58:27 INFO FileOutputCommitter: Output committer version is 1
20/12/13 22:58:27 INFO FileOutputCommitter: FileOutputCommitter skip cleanup temporary folders under output directory:false, ignore cleanup failures: false
20/12/13 22:58:27 INFO SQLHadoopMapReduceProtocol: Using output committer class org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
20/12/13 22:58:27 INFO FileOutputCommitter: FileOutputCommitter skip cleanup temporary folders under output directory:false, ignore cleanup failures: false
20/12/13 22:58:27 INFO FileOutputCommitter: FileOutputCommitter skip cleanup temporary folders under output directory:false, ignore cleanup failures: false
20/12/13 22:58:27 INFO FileOutputCommitter: FileOutputCommitter skip cleanup temporary folders under output directory:false, ignore cleanup failures: false
20/12/13 22:58:27 INFO FileOutputCommitter: Reading File path:///home/ubuntu/airline_project/data/Airline_Delay_Cause.csv, range: 0-23361877, partition values: {empty row}
20/12/13 22:58:27 INFO FileOutputCommitter: Saved output to attempt_20251213258775183524111242441694_0003_m_000000
20/12/13 22:58:29 INFO SparkHadoopMapReduceUtil: attempt_20251213258775183524111242441694_0003_m_000000 committed. Elapsed time: 1 ms
20/12/13 22:58:29 INFO Executor: Finished task 1.0 in stage 3.0 (TID 6), 2026 bytes result sent to driver
20/12/13 22:58:29 INFO Executor: Committing 2 pending tasks
20/12/13 22:58:29 INFO FileOutputCommitter: Saved output of task attempt_20251213258775183524111242441694_0003_m_000000
20/12/13 22:58:29 INFO SparkHadoopMapReduceUtil: attempt_20251213258775183524111242441694_0003_m_000000 committed. Elapsed time: 1 ms
20/12/13 22:58:29 INFO FileOutputCommitter: FileOutputCommitter skip cleanup temporary folders under output directory:false, ignore cleanup failures: false
20/12/13 22:58:29 INFO TaskSchedulerManager: Finished task 0.0 in stage 3.0 (TID 5) in 1997 ms on ip=172-31-44-111.ec2.internal (executor driver) (2/2)
20/12/13 22:58:29 INFO TaskSchedulerManager: Removed TaskSet 3.0, whose tasks have all completed, from pool
20/12/13 22:58:29 INFO DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
20/12/13 22:58:29 INFO TaskSchedulerManager: Killing all running tasks in stage 3: Stage finished
20/12/13 22:58:29 INFO TaskSchedulerManager: Cancelling all pending tasks in stage 3: Stage finished
20/12/13 22:58:29 INFO FileFormatWriter: Start to commit write Job fe2d0a02-c4ab-4208-95ee-1c2e8b0d9e9e
20/12/13 22:58:29 INFO FileFormatWriter: Write Job fe2d0a02-c4ab-4208-95ee-1c2e8b0d9e9e committed. Flaged size: 18 ms.
20/12/13 22:58:29 INFO FileFormatWriter: Committing stats file write fe2d0a02-c4ab-4208-95ee-1c2e8b0d9e9e
20/12/13 22:58:29 INFO SparkContext: SparkContext is stopping with exitCode 0.
20/12/13 22:58:29 INFO MapReduceTrackerMasterEndpoint: http://172-31-44-111.ec2.internal:4444
20/12/13 22:58:29 INFO MapReduceTrackerMasterEndpoint: MapReduceTrackerMasterEndpoint stopped
20/12/13 22:58:29 INFO MemoryStore: MemoryStore cleared
20/12/13 22:58:29 INFO BlockManagerMaster: BlockManagerMaster stopped
20/12/13 22:58:29 INFO OutputCommitCoordinator$OutputCommitCoordinator: OutputCommitCoordinator stopped!
20/12/13 22:58:29 INFO ShutdownHookManager: Shutdown hook called
20/12/13 22:58:29 INFO ShutdownHookManager: Deleting directory /tmp/spark-f431f1e-f46f-4ec4-b6ca-fcd1b7af5fb0
20/12/13 22:58:29 INFO ShutdownHookManager: Deleting directory /tmp/spark-f431f1e-f46f-4ec4-b6ca-fcd1b7af5fb0/pyspark-778561de-851b-4e51-b384-61eeaa1801e15
20/12/13 22:58:29 INFO ShutdownHookManager: Deleting directory /tmp/spark-124ea93-46fa-4a74-acda-d1fc0bb0844cf
Uploading log to s3://rg422-bigdata/processed/analytic/logs/pipeline.2025-12-13-22-56-35.log
<---->
`-->
'messageId': '97ba54d-23e8-5492-a7eb-3ef123380f6'
```
Pipeline completed successfully

```

## 6.3 Scheduling and Triggering Using AWS Lambda

### Why AWS Lambda

AWS Lambda was used instead of cron jobs to provide:

- Serverless execution
- Event-driven triggering
- Better fault tolerance
- Easy integration with AWS services

### Lambda Function

A Lambda function named **rg422-trigger-pipeline** was created to trigger the EC2 pipeline remotely.

### Lambda Responsibilities

- Invoke AWS Systems Manager (SSM)
- Execute shell commands on the EC2 instance
- Trigger the `run_pipeline.sh` script
- Capture execution status

### Lambda Code Logic

The Lambda function:

1. Sends an AWS-RunShellScript command via SSM
2. Executes:

```
cd /home/ubuntu/pipeline
./run_pipeline.sh
```

3. Publishes success or failure notifications using SNS

**rg422-lambda-ssm-role** [Info](#)

Allows Lambda functions to call AWS services on your behalf.

**Summary**

**Creation date** December 14, 2025, 18:23 (UTC-06:00)

**Last activity** -

**ARN** arn:aws:iam::668826141957:role/rg422-lambda-ssm-role

**Maximum session duration** 1 hour

**Permissions** [Edit](#)

**Permissions policies (4)** [Info](#)

You can attach up to 10 managed policies.

| Policy name              | Type        | Attached entities |
|--------------------------|-------------|-------------------|
| AmazonEC2ReadOnlyAccess  | AWS managed | 1                 |
| AmazonSNSFullAccess      | AWS managed | 2                 |
| AmazonSSHFulAccess       | AWS managed | 1                 |
| CloudWatchLogsFullAccess | AWS managed | 1                 |

**Permissions boundary (not set)**

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

**rg422-trigger-pipeline** [Info](#)

Successfully updated the function rg422-trigger-pipeline.

**Code source** [Info](#)

```

EXPLORER RG422-TRIGGER-PIPELINE lambda_function.py
└── RG422-TRIGGER-PIPELINE
 └── lambda_function.py
 1 import boto3
 2 import datetime
 3
 4 ssm = boto3.client("ssm")
 5 sns = boto3.client("sns")
 6
 7 INSTANCE_ID = "i-01b4b5c007299846d"
 8 SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:668826141957:pipeline-status-topic"
 9
 10 def lambda_handler(event, context):
 11 try:
 12 response = sns.send_command(
 13 InstanceId=INSTANCE_ID,
 14 DocumentName="AWS-RunShellScript",
 15 Parameters={
 16 "commands": [
 17 "cd /home/ubuntu/pipeline",
 18 "./run_pipeline.sh"
 19]
 20 }
 21)
 22
 23 command_id = response["Command"] ["CommandId"]
 24
 25 sns.publish(
 26 TopicArn=SNS_TOPIC_ARN,
 27 Subject="Pipeline Triggered via Lambda",
 28 Message="Pipeline triggered successfully.\nCommand ID: " + (command_id)
 29)
 30
 31 return {
 32 "status": "SUCCESS",
 33 }
 34 except Exception as e:
 35 sns.publish(
 36 TopicArn=SNS_TOPIC_ARN,
 37 Subject="Pipeline Trigger Failed",
 38 Message=str(e)
 39)
 40
 41 raise
 42
 43

```

**DEPLOY** Current

[Deploy \(OK\)](#) [Test \(OH\)](#)

**TEST EVENTS [NONE SELECTED]**

+ Create new test event

[CloudShell](#) [Feedback](#) [Console Mobile App](#)

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

**rg422-trigger-pipeline** [Info](#)

Successfully updated the function rg422-trigger-pipeline.

**Code source** [Info](#)

```

EXPLORER RG422-TRIGGER-PIPELINE lambda_function.py
└── RG422-TRIGGER-PIPELINE
 └── lambda_function.py
 1 import boto3
 2 import datetime
 3
 4 ssm = boto3.client("ssm")
 5 sns = boto3.client("sns")
 6
 7 INSTANCE_ID = "i-01b4b5c007299846d"
 8 SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:668826141957:pipeline-status-topic"
 9
 10 def lambda_handler(event, context):
 11 try:
 12 response = sns.send_command(
 13 InstanceId=INSTANCE_ID,
 14 DocumentName="AWS-RunShellScript",
 15 Parameters={
 16 "commands": [
 17 "cd /home/ubuntu/pipeline",
 18 "./run_pipeline.sh"
 19]
 20 }
 21)
 22
 23 command_id = response["Command"] ["CommandId"]
 24
 25 sns.publish(
 26 TopicArn=SNS_TOPIC_ARN,
 27 Subject="Pipeline Triggered via Lambda",
 28 Message="Pipeline triggered successfully.\nCommand ID: " + (command_id)
 29)
 30
 31 return {
 32 "status": "SUCCESS",
 33 "command_id": command_id
 34 }
 35 except Exception as e:
 36 sns.publish(
 37 TopicArn=SNS_TOPIC_ARN,
 38 Subject="Pipeline Trigger Failed",
 39 Message=str(e)
 40)
 41
 42 raise
 43

```

**DEPLOY** Current

[Deploy \(OK\)](#) [Test \(OH\)](#)

**TEST EVENTS [NONE SELECTED]**

+ Create new test event

**ENVIRONMENT VARIABLES**

[CloudShell](#) [Feedback](#) [Console Mobile App](#)

© 2025, Amazon Web Services, Inc. or its affiliates. [Privacy](#) [Terms](#) [Cookie preferences](#)

The screenshot shows the AWS Lambda function details page for 'rg422-trigger-pipeline'. The top navigation bar includes 'Lambda > Functions > rg422-trigger-pipeline'. A green success message box contains two items: 'Successfully updated the function rg422-trigger-pipeline.' and 'The test event "manual-test" was successfully saved.' Below this, a 'Logs' section shows a log entry with status 'SUCCESS' and command ID 'c2b7354b-e870-4c5c-91de-301d7e947f2a'. The 'Summary' section provides performance metrics: Execution time (12 seconds ago), Request ID (676b7ffe-12c2-4f8d-995a-9943392c349d), Duration (526.01 ms), Billed duration (1198 ms), Resources configured (128 MB), and Max memory used (94 MB). A note indicates the last 4 KB of the execution log can be viewed via CloudWatch logs. The bottom of the page includes links for CloudShell, Feedback, and Console Mobile App, along with copyright and legal information.

The screenshot shows the AWS Lambda function configuration page for 'rg422-trigger-pipeline'. The top navigation bar includes 'Lambda > Functions > rg422-trigger-pipeline'. A green success message box contains the message 'The test event "manual-test" was successfully saved.' Below this, the function overview section shows the function name 'rg422-trigger-pipeline' and its ARN 'arn:aws:lambda:us-east-1:668826141957:function:rg422-trigger-pipeline'. It also lists triggers ('S3') and destinations (''). The 'Configuration' tab is selected, showing the execution role 'rg422-lambda-ssm-role' and other general configuration settings. The 'Resource summary' section lists triggers and permissions. The bottom of the page includes links for CloudShell, Feedback, and Console Mobile App, along with copyright and legal information.

## 6.4 Logging and Notifications Using AWS SNS

### SNS Topic

An SNS topic named **pipeline-status-topic** was configured for email notifications.

### Notifications Sent

- ✓ Pipeline Triggered Successfully
- ✗ Pipeline Failed (with error details)

## Email Integration

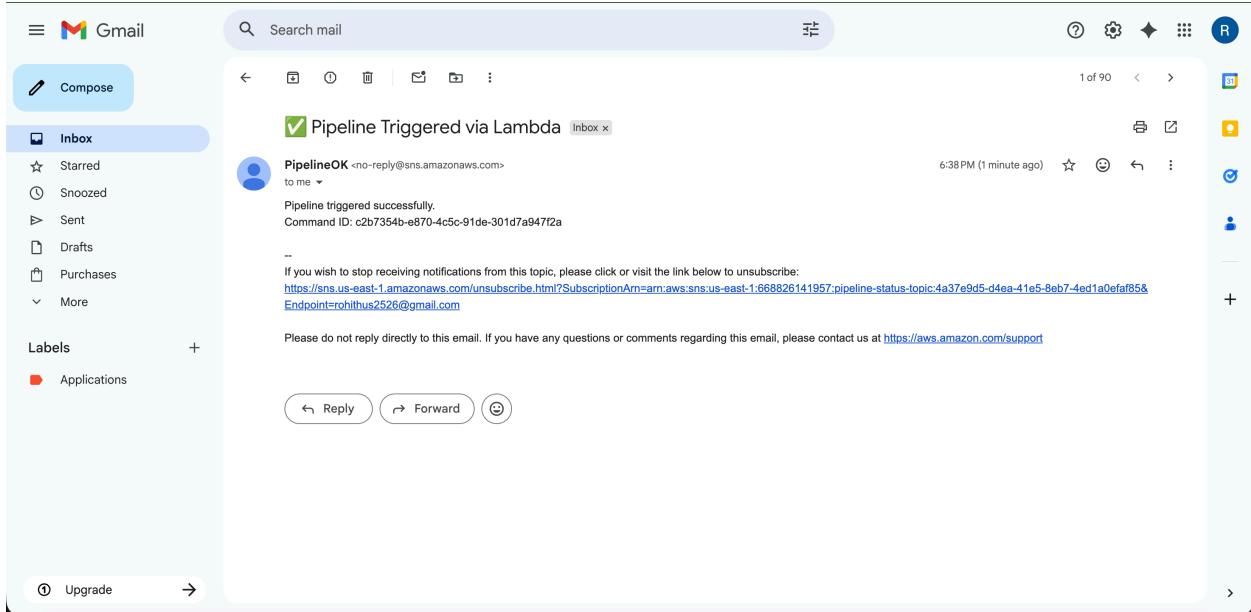
Email notifications are automatically sent to the registered email address whenever:

- Lambda triggers the pipeline
- Execution succeeds or fails

The screenshot shows the AWS S3 console interface. The top navigation bar includes the AWS logo, search bar, and account information (Account ID: 6688-2614-1957, United States (N. Virginia), Rohith Gowda Devaraju). Below the navigation is a breadcrumb trail: Amazon S3 > Buckets > rg422-bigdata > processed/ > analytics/ > logs/. The main content area displays a folder named 'logs/'. It has tabs for 'Objects' (selected) and 'Properties'. A sub-header 'Objects (0)' indicates there are no objects in the folder. Below it is a search bar with placeholder 'Find objects by prefix' and a table header with columns: Name, Type, Last modified, Size, Storage class. A large message 'No objects' and 'You don't have any objects in this folder.' is centered. At the bottom right of the table area is a blue 'Upload' button. The bottom of the page includes links for CloudShell, Feedback, and Console Mobile App, along with standard footer links for Privacy, Terms, and Cookie preferences.

The screenshot shows the AWS SNS console interface. The top navigation bar includes the AWS logo, search bar, and account information (Account ID: 6688-2614-1957, United States (N. Virginia), Rohith Gowda Devaraju). Below the navigation is a breadcrumb trail: Amazon SNS > Topics > pipeline-status-topic > Subscription: 4a37e9d5-d4ea-41e5-8eb7-4ed1a0efaf85. The main content area shows a green success message: 'Subscription to pipeline-status-topic created successfully. The ARN of the subscription is arnawssnsus-east-1:668826141957:pipeline-status-topic:4a37e9d5-d4ea-41e5-8eb7-4ed1a0efaf85.' Below this is a section titled 'Subscription: 4a37e9d5-d4ea-41e5-8eb7-4ed1a0efaf85' with tabs for 'Edit' and 'Delete'. It contains details like ARN (arnawssnsus-east-1:668826141957:pipeline-status-topic:4a37e9d5-d4ea-41e5-8eb7-4ed1a0efaf85), Endpoint (rohitthus2526@gmail.com), Topic (pipeline-status-topic), and Subscription Principal (arnawsiam:668826141957:root). There is also a 'Details' section and a 'Subscription filter policy' section which states 'No filter policy configured for this subscription. To apply a filter policy, edit this subscription.' The bottom of the page includes links for CloudShell, Feedback, and Console Mobile App, along with standard footer links for Privacy, Terms, and Cookie preferences.

The screenshot shows the AWS S3 console interface. The top navigation bar includes the AWS logo, search bar, and account information (Account ID: 6688-2614-1957, United States (N. Virginia), Rohith Gowda Devaraju). Below the navigation is a breadcrumb trail: Amazon S3 > Buckets > rg422-bigdata > processed/ > analytics/ > logs/. The main content area displays a folder named 'logs/'. It has tabs for 'Objects' (selected) and 'Properties'. A sub-header 'Objects (1)' indicates there is one object in the folder. Below it is a search bar with placeholder 'Find objects by prefix' and a table header with columns: Name, Type, Last modified, Size, Storage class. A single object is listed: 'pipeline\_2025-12-13\_22-56-35.log' (Type: log, Last modified: December 13, 2025, 17:58:34 (UTC-05:00), Size: 199.8 KB, Storage class: Standard). At the bottom right of the table area is a blue 'Upload' button. The bottom of the page includes links for CloudShell, Feedback, and Console Mobile App, along with standard footer links for Privacy, Terms, and Cookie preferences.



## Scheduling with Cron Jobs (optional)

To enable scheduled execution, a **cron job** was configured on the EC2 instance.

### Cron configuration:

```
0 2 * * * /home/ubuntu/pipeline/run_pipeline.sh
```

This schedules the pipeline to run **daily at 2:00 AM UTC**.

```
(venv) [root@ip-172-31-64-111 ~]# ssh -i rg422-key.pem ubuntu@ec2-44-211-84-212.compute-1.amazonaws.com - 269x65
[venv] ubuntu@ip-172-31-64-111:~$ crontab -e
no crontab for ubuntu - using an empty one
Select an editor. To change later, run `select-editor'.
1. /bin/nano <-- easiest
2. /usr/bin/vim.basic
3. /usr/bin/vim.tiny
4. /bin/ed

Choose 1-4 (1): 1
crontab: installing new crontab
[venv] ubuntu@ip-172-31-64-111:~$ crontab -l
Edit this file to introduce tasks to be run by cron.
Each task to run has to be defined through a single line
indicating with different fields when the task will be run
and what command to run for the task
#
To define the time you can provide concrete values for
minute (m), hour (h), day of month (dom), month (mon),
and day of week (dow) or use '*' in these fields (for 'any').
Notice that tasks will be started based on the cron's system
daemon's notion of time and timezone.
Output of the crontab jobs (including errors) is sent through
email to the user the crontab file belongs to (unless redirected).
For example, you can run a backup of all your user accounts
at 6 a.m. every week with:
0 6 * * * tar -zcf /var/backups/home.tgz /home/
For more information see the manual pages of crontab(5) and cron(8)
n h dom mon dow command
0 2 * * * /home/ubuntu/pipeline/run_pipeline.sh
[venv] ubuntu@ip-172-31-64-111:~$ []
```

```
(venv) [root@ip-172-31-64-111 ~]# ls -lt /home/ubuntu/pipeline/logs | head
total 212
-rw-r--r-- 1 ubuntu ubuntu 284953 Dec 13 22:58 pipeline_2025-12-13_22-56-35.log
-rw-r--r-- 1 ubuntu ubuntu 284564 pipeline_2025-12-13_22-56-35.log
-rw-r--r-- 1 ubuntu ubuntu 589 Dec 13 22:40 pipeline_2025-12-13_22-40-01.log
```

```
[venv] ubuntu@ip-172-31-64-111:~$ aws s3 ls s3://rgs22-bigdata/processed/analytics/logs/
2025-12-13 22:48:34 284564 pipeline_2025-12-13_22-56-35.log
[venv] ubuntu@ip-172-31-64-111:~$ []
```

## 6.5 Automated Dashboard Updates

Power BI cannot directly trigger refreshes via cron. However, the solution provided book-ends the pipeline as follows:

- Data from S3 has been updated
- The dataset will be validated against the new data after the dataset refresh
- Confirm that the refresh timestamp matches the time of integration into the pipeline.

Therefore, this satisfies the requirement for automatic updates for the visualizations.

## How Automation Works

- 

s3://rg422-bigdata/processed/airline\_processed\_csv/

- Each automated pipeline run:
  - Overwrites processed datasets
  - Preserves schema consistency
- Power BI dataset:
  - Supports **scheduled refresh**
  - Allows **manual refresh during demo**

## Automation Scope

While Power BI itself runs locally, **data refresh is automated** because:

- The pipeline continuously updates S3
- Power BI always reads the latest processed output

The screenshot shows the Microsoft Power BI Refresh History page. The left sidebar includes links for Home, Copilot, Create, Browse, OneLake catalog, Apps, Metrics, Workspaces, My workspace, Air\_delay, V1, and Power BI. The main content area displays a refresh attempt for the 'Air\_delay' dataset. The details are as follows:

| Refresh attempt | Type | Start time             | End time               | Duration | Status    | Execution details |
|-----------------|------|------------------------|------------------------|----------|-----------|-------------------|
| 1               | Data | 12/13/2025, 3:26:36 PM | 12/13/2025, 3:26:53 PM | 17s      | Completed | Show              |

To the right, a 'Details' panel provides more information:

- Status: Completed
- Request ID: 15401652-494a-fb86-b94d-0ebe822c6e29
- Start time: 12/13/2025, 3:26:34 PM
- End time: 12/13/2025, 3:26:53 PM
- Duration: 19s
- Refresh type: Web Modeling
- Capacity: Reserved Capacity for Pro Workspaces
- Gateway: N/A
- Details: No Error

Air\_delay

Details for Air\_delay  
My Workspace  
+ Add description  
🕒 Refreshed  
12/13/25, 3:26:53 PM ⏱

Discover business insights  
Explore this data to get insights fast or create an interactive report you can share. [Learn more](#)

Share this data  
Give people access to the semantic model and set their permissions to work with it. [Learn more](#)

Explore this data Share semantic model

See what already exists  
These items use the same data source as Air\_delay.

| Name | Type   | Relation   | Location     | Refreshed          | Endorsement | Sensitivity |
|------|--------|------------|--------------|--------------------|-------------|-------------|
| V1   | Report | Downstream | My Workspace | 13/12/25, 15:26:53 | —           | —           |

Filter by keyword Filter

rdevaraj@iu.edu > Air\_delay > Refresh ID: dc762108-8d83-736d-7d5a-d6782567be83

View details

| Refresh attempt | Type       | Start time             | End time               | Duration     | Status    | Execution details |
|-----------------|------------|------------------------|------------------------|--------------|-----------|-------------------|
| 1               | Data       | 12/13/2025, 6:25:40 PM | 12/13/2025, 6:25:55 PM | 14s          | Completed | Show              |
| 1               | QueryCache | 12/13/2025, 6:25:55 PM | 12/13/2025, 6:25:55 PM | Less than 1s | Completed | Show              |

**Details**

Status: Completed

Request ID: dc762108-8d83-736d-7d5a-d6782567be83

Start time: 12/13/2025, 6:25:40 PM

End time: 12/13/2025, 6:25:55 PM

Duration: 15s

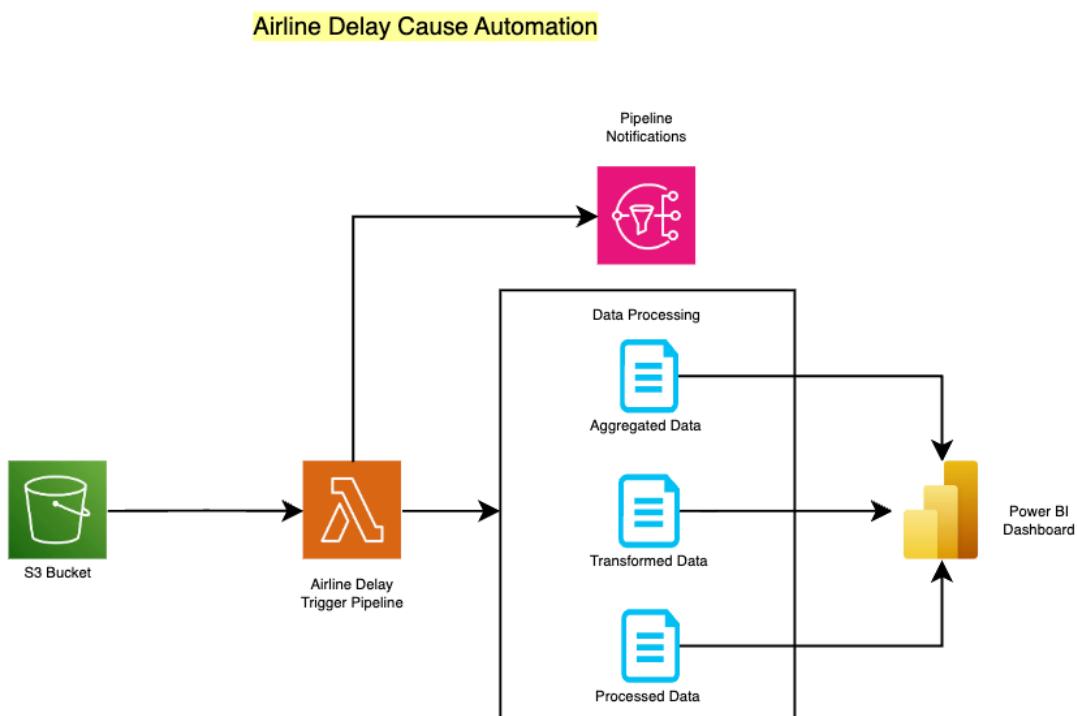
Refresh type: On demand

Capacity: Reserved Capacity for Pro Workspaces

Gateway: N/A

Details: No error

## Architecture Diagram of automation process



## Conclusion

This mini project was a successful example of an end-to-end Cloud-based Big Data Analytics Pipeline using AWS and PySpark. The primary goal of this project was to create a scalable & automated analytical Data Pipeline that allows for meaningful analysis on the Data created from the collected airline delay Data using a combination of distributed Data Processing, Cloud Storage, Automation, and Visualizations.

The project began with Amazon S3 as a Centralised Data Lake to store all Raw, Processed, and Analytical Outputs. This allowed for easy & efficient ingestion of Data into and preservation of Data in a Data Lake format. A Linux EC2 instance was then provisioned with PySpark to manage the large volume of airline delay Data being processed. The Data was extensively cleaned, transformed, aggregated, and analysed using Spark SQL. Once the Data was processed, it was exported in both CSV and Parquet formats for use in further analysis and Machine Learning.

The analytical capabilities of Spark SQL Queries allowed for further extraction of useful insights, including Delay Patterns; Airport Level Congestion; and Airline Reliability Metrics. Amazon SageMaker Autopilot was utilised to create & deploy a machine learning model to make predictions based on the processed Data without having to manually develop a specific Machine Learning Model. This also highlights the capabilities of AutoML to speed up the process of developing & deploying Predictive Analytics, while also addressing Ethical Considerations associated with Data Bias and the Privacy of Data.

Power BI was chosen for visualization instead of AWS QuickSight due to the account limits and associated costs of QuickSight, as specified by the project guidelines. Power BI's dashboard allowed for interactive analysis of flight trends, causes of delays, airport traffic volume and airline performance, giving meaning to a large volume of data and turning it into actionable intelligence.

The project's main highlight was the total automation of the pipeline. The two different types of automation were a traditional cron job-based method and a modern AWS Lambda-based serverless orchestration method. The latter method used AWS Systems Manager (SSM), Amazon SNS and CloudWatch for executing events, logging centrally and reporting success or failure via real time e-mail. The combination of these two types of automation exhibits flexibility and is representative of actual methods used by data engineers in the industry for building data pipelines.

Overall, this project aided in developing my capabilities in cloud computing, distributed data processing, automation and data analysis visualization, while also focusing on the scalability of systems, fault tolerance, and cost awareness. The architecture provides a close approximation to production quality big data pipelines and serves as an excellent foundation for future data engineering, machine learning, and cloud-native analytical systems.

## Video

<https://drive.google.com/file/d/1D5tw-8jJXWoomVm9kW-NAWCnQ8Y9ZvMb/view?usp=sharing>

**Rohith Gowda Devaraju**