

# Flight Delay Prediction using Big Data and Machine Learning

Mounika Veeramachaneni

Rolano Rebelo

Srilekha Sridasyam

Sai Sahithi Gundapaneni

CSCE 5300: Introduction to Big Data and Data Science

# Motivation

In the airlines sector, flight delays are a major and frequent occurrence that cost airlines a lot of money and which lead to annoyed passengers, delayed and cancelled plans. Our goal is to create a prediction model that can precisely predict flight delays by utilizing Big Data and Data Science techniques like Spark and Machine learning. This will allow travelers and airlines to minimize the effects of interruptions and maximize their travel arrangements which is a win-win situation.

# Significance

This project is significant because it has the potential to use machine learning and big data tools to solve an actual issue facing the aviation industry.

We can find patterns, variables, and trends that lead to flight delays by utilizing big data components to process massive amounts of historical flight data and applying sophisticated machine learning techniques to analyze the trends and patterns.

The project's outcome has the potential to enable airlines to boost customer satisfaction, optimize resource allocation, and increase operational efficiency.

# Objective and Goals

To develop a predictive model that accurately forecasts flight delays using historical flight data

Data Collection: Collect past flight performance data and form the dataset

Data Preprocessing: Cleaning and preprocessing the data to make it suitable for modelling using AWS Glue and Spark on Amazon EMR

Feature Engineering: Identifying relevant features, as well as modifying and creating new ones that will help us predict the flight delay accurately

Model Development: Training and testing several ML models like logistic regression, decision trees, SVM to predict delays

Model Evaluation: Assessing the model's performance using metrics such as accuracy, precision, recall, and F1-score

Data Visualization: Using libraries like Matplotlib and SparkSQL to visualize the data

# Frameworks/Tools

Amazon S3: This is used to store the dataset which is in CSV format

AWS Glue: This is used for ETL which is extract the data from S3 and convert it into a Glue table or a schema that can be used by other AWS services like EMR for preprocessing, Athena for querying the data, etc.

Amazon EMR: Spark on EMR is used to preprocess the data

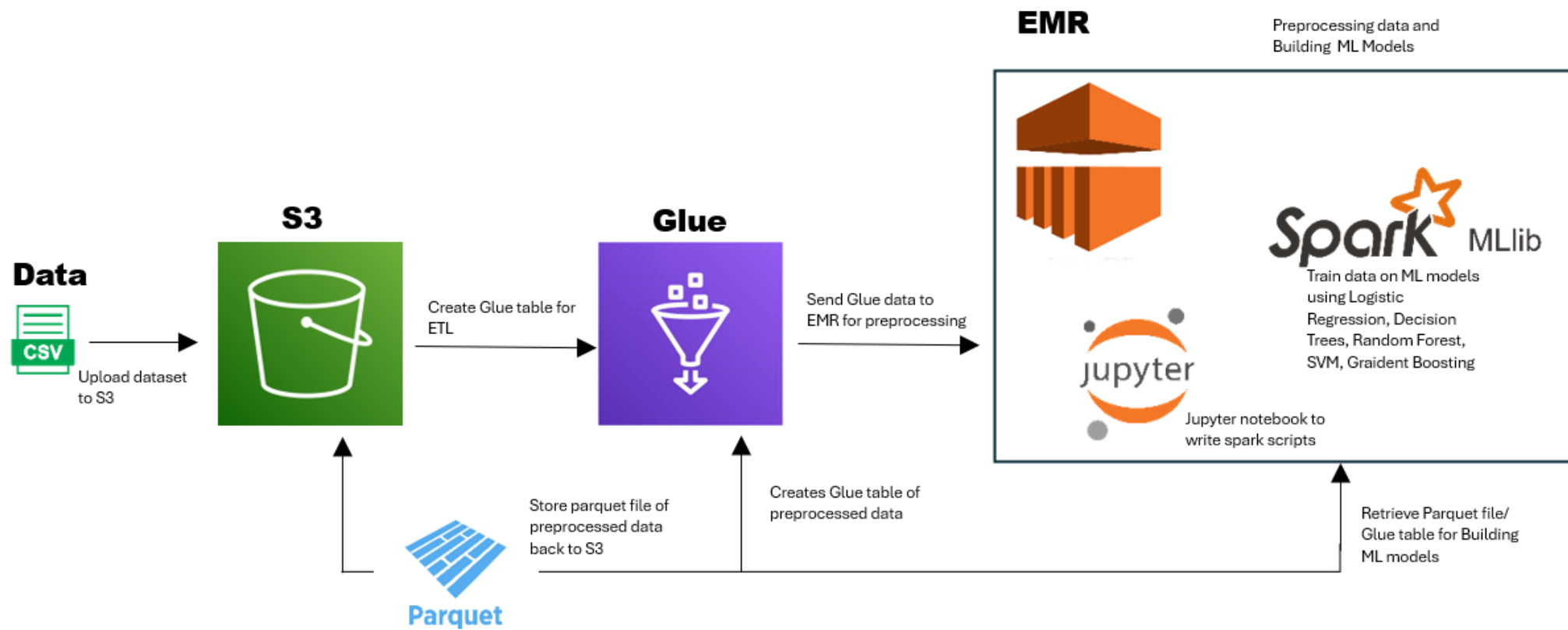
Jupyter Notebook: This is used to write the Spark script

SparkMlib: This is used to build, train and test different ML models as well as evaluate each one of them

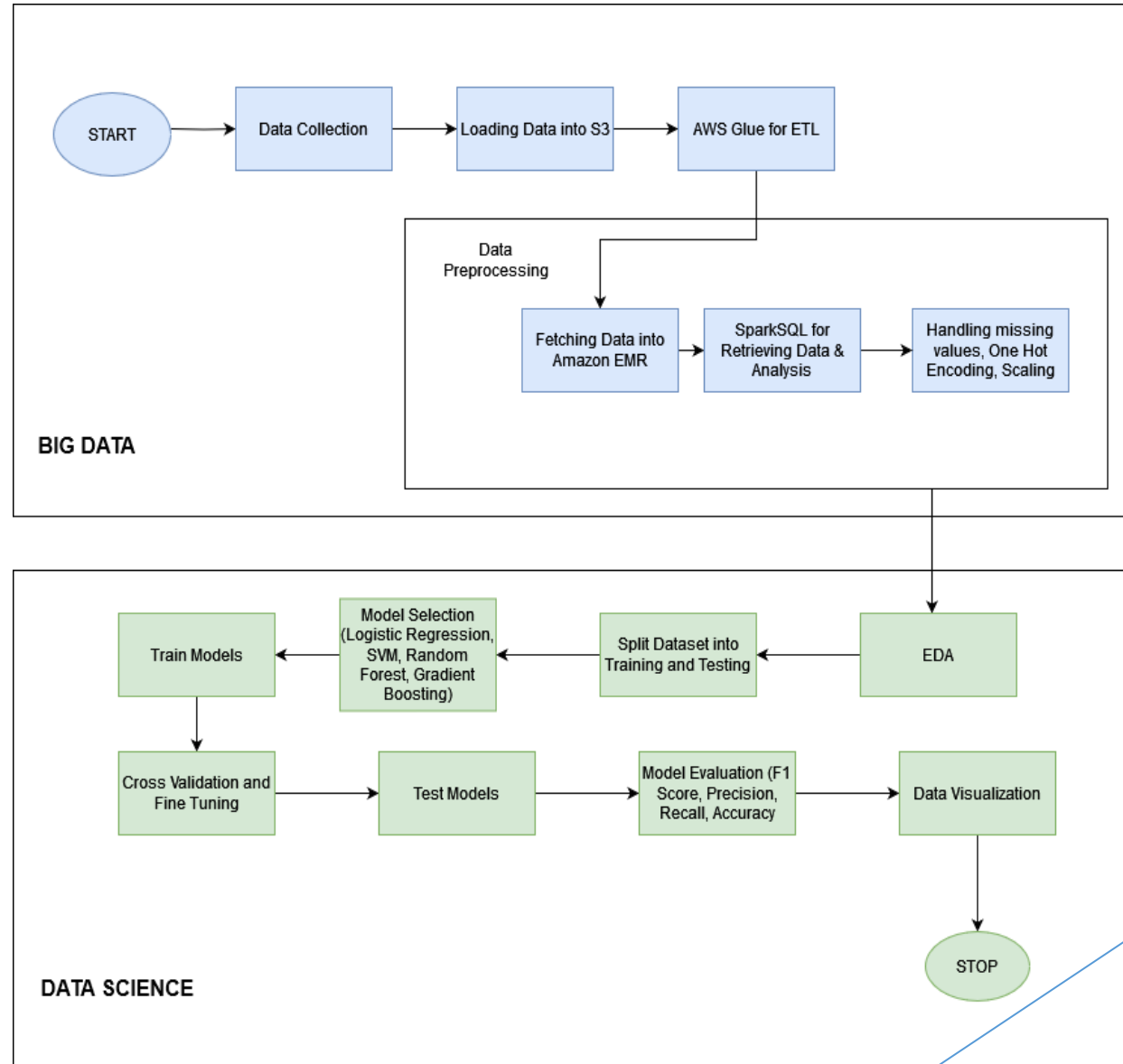
SparkSQL: This is used to retrieve and store data from and to S3 and Glue from EMR as well as query the dataframe to visualize it

Matplotlib: This is used for data visualization

# Model



# Work Flow Diagram



# Description of AWS Components

## **Amazon S3**

S3 is an object storage service managed by AWS that offers industry level scalability, data availability, security and performance.

In our project we have used S3 to store the raw CSV airlines.csv dataset which is the main input for our data analysis.

S3 is also used to store the output file parquet which contains the pre-processed data by Spark.



## AWS Glue

Glue is a serverless data integration service by AWS which is used to easily store prepared and combined data for analytics, machine learning, and application development.

The AWS Glue Data Catalog is the central repository which gets the data from the S3 bucket and store the schema of the data in a table that allows us to perform ETL (Extract, Load, Transform) tasks on the data for analytics.

Using Glue wasn't necessary in our project, but it is a good practise in Big data and data science projects since it can let us query the data and perform analytics using tools like AWS Athena.

## AWS EMR

EMR is a cloud-native big data platform, which allows businesses to process massive amounts of data in a quick and cost effective manner across resizable clusters of Amazon EC2 instances.

It supports various big data frameworks such as Apache Hadoop, Spark, HBase, and others.

Reasons we used AWS EMR over EC2:

- AWS EMR can handle vast amounts of data, performing complex processing tasks like batch processing, streaming, and machine learning prediction at scale.
- EMR manages cluster provisioning, configuration, and tuning. It automatically scales up or down to handle the workload efficiently, ensuring that we only pay for the resources you use.
- It integrates with other AWS services, such as S3 for storage, and Glue which allows us to easily send and retrieve data from the different resources for processing.

- ▶ For our project we created an EMR Cluster with resources like Hue, Spark, Hadoop, JupyterHub, etc.
- ▶ We used Spark for the writing the script on a Jupyter Notebook which reads the data from Glue and processes the data and makes it ready for training the ML models.
- ▶ SparkMlib in Spark was used to train and test the ML models.

# Dataset

- ▶ Airlines dataset has 539383 instances and 8 different features. The task is to predict whether a given flight will be delayed, given the information of the scheduled departure.
- ▶ Features
  1. **Airline:** The airline operating the flight.
  2. **Flight:** The flight number.
  3. **Airport From:** The departure airport.
  4. **Airport To:** The arrival airport.
  5. **DayOfWeek:** The day of the week on which the flight is scheduled.
  6. **Time:** The scheduled departure time.
  7. **Length:** The duration of the flight.
  8. **Delay:** The target variable, indicating if the flight was delayed.

id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay
1	CO	269	SFO	IAH	3	15	205	1
2	US	1558	PHX	CLT	3	15	222	1
3	AA	2400	LAX	DFW	3	20	165	1
4	AA	2466	SFO	DFW	3	20	195	1
5	AS	108	ANC	SEA	3	30	202	0
6	CO	1094	LAX	IAH	3	30	181	1
7	DL	1768	LAX	MSP	3	30	220	0
8	DL	2722	PHX	DTW	3	30	228	0
9	DL	2606	SFO	MSP	3	35	216	1
10	AA	2538	LAS	ORD	3	40	200	1
11	CO	223	ANC	SEA	3	49	201	1
12	DL	1646	PHX	ATL	3	50	212	1
13	DL	2055	SLC	ATL	3	50	210	0
14	AA	2408	LAX	DFW	3	55	170	0
15	AS	132	ANC	PDX	3	55	215	0
16	US	498	DEN	CLT	3	55	179	0
17	B6	98	DEN	JFK	3	59	213	0
18	CO	1496	LAS	IAH	3	60	162	0

Screenshot of Dataset

# Glue Data Catalog

Schema (9)

View and manage the table schema.

Q Filter schemas

< 1 > ⚙

Edit schema as JSON

Edit schema

#	Column name	Data type	Partition key	Comment
1	id	bigint	-	-
2	airline	string	-	-
3	flight	bigint	-	-
4	airportfrom	string	-	-
5	airportto	string	-	-
6	dayofweek	bigint	-	-
7	time	bigint	-	-
8	length	bigint	-	-
9	delay	bigint	-	-

# Data Preprocessing Techniques

The data is pre-processed using Spark on AWS EMR

## 1. Data Loading

- ▶ The data is loaded from AWS Glue

## 2. Data Cleaning

- ▶ Handled missing data and null values in the feature columns

## 3. Feature Transformation

- ▶ 'time' is converted to cyclic features ('time\_sin', 'time\_cos') to capture the time continuity.

## 4. Encoding

- ▶ Used index and one-hot encoding for categorical variables (airlines, airports, flight) to convert into numerical data

## 5. Feature Assembling

We combine all the numeric and transformed features into one feature vector using 'VectorAssembler'

## 6. Feature Scaling

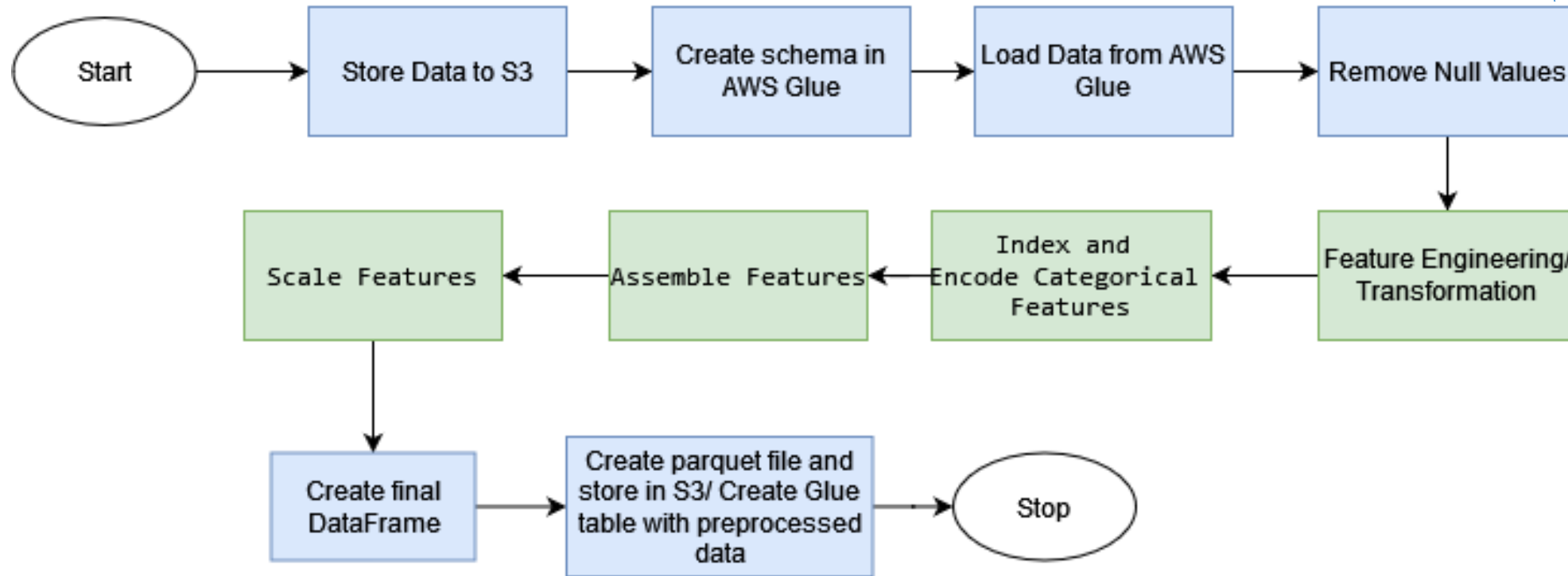
We normalize the features using 'StandardScaler' to ensure model fairness across different scales

## 7. Feature and Label Selection

Finally, we prepare the final data frame with features for machine learning and the 'delay' label which indicates the flight delays and store it in a parquet file in S3 as well as in a glue table



# Flowchart of Preprocessing



# Implementation

Big Data refers to very large and massive datasets that are analyzed to study the trends, reveal patterns and associations with human interaction.

Data Science is working with these large volumes of data to develop analytical and predictive models.

Our project uses Big Data tools such as Amazon S3 and Spark on Amazon EMR to store and preprocess the massive Airlines dataset to study flight delays and we use Data Science by building Machine Learning models using algorithms like Logistic Regression, Decision Trees, SVM to predict the flight delays.

[Amazon S3](#) > [Buckets](#) > [flight-delay-prediction-bigdata](#)

# flight-delay-prediction-bigdata [Info](#)

[Objects](#)

[Properties](#)

[Permissions](#)

[Metrics](#)

[Management](#)

[Access Points](#)

## Objects (1) [Info](#)



Copy S3 URI

Copy URL

Download

Open

Delete

Actions ▼

Create folder

Upload

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

Find objects by prefix

< 1 >

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	Airlines.csv	csv	March 31, 2024, 10:22:50 (UTC-05:00)	17.7 MB	Standard

We use Amazon S3 to store the Airlines dataset which is a csv file

AWS Glue

Getting started  
ETL jobs  
Visual ETL  
Notebooks  
Job run monitoring  
Data Catalog tables  
Data connections  
Workflows (orchestration)  
Data Catalog  
Databases  
Tables  
Stream schema registries  
Schemas  
Connections  
Crawlers  
Classifiers  
Catalog settings  
Data Integration and ETL  
Legacy pages  
What's New  
Documentation  
AWS Marketplace  
Enable compact mode  
Enable new navigation

AWS Glue > Tables > flight\_delay\_prediction\_bigdata

flight\_delay\_prediction\_bigdata

Last updated (UTC)  
April 29, 2024 at 03:00:36

Version 0 (Current version)

Actions

Table overviewData quality New

Table detailsAdvanced properties

Name  
flight\_delay\_prediction\_bigdata

Description  
-

Database  
flight-data-glue-crawler-output

Classification  
CSV

Location  
s3://flight-delay-prediction-bigdata/

Connection  
-

Deprecated  
-

Last updated  
March 31, 2024 at 15:26:05

Input format  
org.apache.hadoop.mapred.TextInputFormat

Output format  
org.apache.hadoop.hive.q1.io.HiveIgnoreKeyTextOutputFormat

Serde serialization lib  
org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

SchemaPartitionsIndexesColumn statistics - new

Schema (9)  
View and manage the table schema.

Filter schemas

#	Column name	Data type	Partition key	Comment
1	id	bigint	-	-
2	airline	string	-	-
3	flight	bigint	-	-
4	airportfrom	string	-	-
5	airportto	string	-	-
6	dayofweek	bigint	-	-
7	time	bigint	-	-
8	length	bigint	-	-
9	delay	bigint	-	-

We first create a role with permissions to access Glue and S3. Then we created a Glue Crawler and assign it this role and set the data store to S3 and give the location where the dataset is located. Then we create a database in Glue where the schema will be stored and run the Glue crawler. The crawler reads the files in the specified S3 path, infers the schema, and creates metadata tables in the Glue Data Catalog.

Review the Schema: After the crawler finishes, we can navigate to the Tables section in AWS Glue to see the schema it inferred from the data.

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

- User groups
- Users
- Roles
- Policies
- Identity providers
- Account settings

Access reports

- Access Analyzer
  - External access
  - Unused access
  - Analyzer settings
- Credential report
- Organization activity
- Service control policies

Related consoles

[IAM Identity Center](#)[AWS Organizations](#)

IAM > Roles > EMR\_Flight\_Delay\_Role

EMR\_Flight\_Delay\_Role

Info

Allows Elastic MapReduce to call AWS services such as EC2 on your behalf.

Summary

Creation date

April 06, 2024, 16:23 (UTC-05:00)

Last activity

46 minutes ago

ARN

arn:aws:iam::988926753451:role/EMR\_Flight\_Delay\_Role

Maximum session duration

1 hour

Permissions

Trust relationships

Tags

Access Advisor

Revoke sessions

Permissions policies (5)

Info

You can attach up to 10 managed policies.

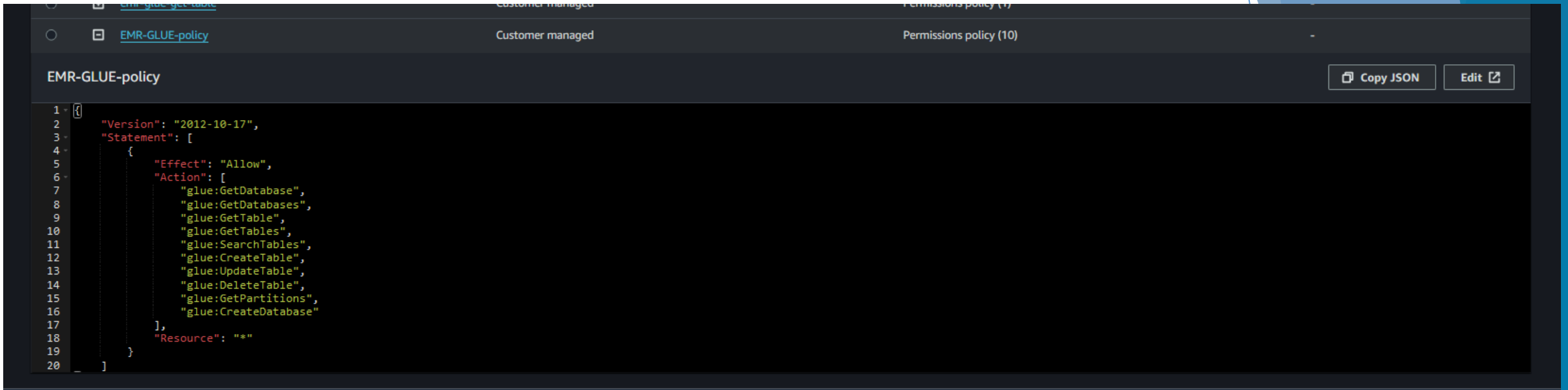
Search

Filter by Type

All types

	Policy name	Type	Attached entities
<input type="checkbox"/>	<div><div></div><div>AmazonEC2FullAccess</div></div>	AWS managed	2
<input type="checkbox"/>	<div><div></div><div>AmazonElasticMapReduceRole</div></div>	AWS managed	2
<input type="checkbox"/>	<div><div></div><div>AmazonEMRFullAccessPolicy_v2</div></div>	AWS managed	2
<input type="checkbox"/>	<div><div></div><div>AmazonS3FullAccess</div></div>	AWS managed	13
<input type="checkbox"/>	<div><div></div><div>EMR-GLUE-policy</div></div>	Customer managed	10

Creating IAM Roles and Permissions is very important for AWS services. We first create an IAM role “EMR\_Flight\_Delay\_Role” and assign it permissions. Only for the sake of testing purposes we gave S3 and Glue full access but this isn’t recommended in production. This role will be given to the EMR cluster while creating it.



The screenshot displays the AWS IAM console interface for a policy named "EMR-GLUE-policy". The policy is categorized as "Customer managed" and is a "Permissions policy (10)". In the top right corner, there are buttons for "Copy JSON" and "Edit". The main area shows the JSON definition of the policy, which includes a version of "2012-10-17" and a single statement that allows a list of Glue actions on all resources ("\*").

```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "glue:GetDatabase",
8         "glue:GetDatabases",
9         "glue:GetTable",
10        "glue:GetTables",
11        "glue:SearchTables",
12        "glue:CreateTable",
13        "glue:UpdateTable",
14        "glue>DeleteTable",
15        "glue:GetPartitions",
16        "glue:CreateDatabase"
17      ],
18      "Resource": "*"
19    }
20  ]
}
```

We also need to create a permission with the necessary actions as can be seen above and attach to the role. This is important so that EMR can perform actions on Glue like retrieving data, create tables, databases, etc.

## Clone "flight-delay-prediction" [Info](#)

### ▼ Name and applications - required [Info](#)

Name your cluster and choose the applications that you want to install on your cluster.

Name


flight-delay-prediction


Amazon EMR release [Info](#)


A release contains a set of applications which can be installed on your cluster.


emr-7.1.0


Application bundle


Spark  
Interactive  



Core  
Hadoop  


Flink  


HBase  


Presto  


Trino  


Custom  


- |   |  |  |
|---|--|--|
| <input type="checkbox"/> AmazonCloudWatchAgent 1.300032.2 | <input type="checkbox"/> Flink 1.18.1                              | <input type="checkbox"/> HBase 2.4.17                |
| <input type="checkbox"/> HCatalog 3.1.3                   | <input checked="" type="checkbox"/> Hadoop 3.3.6                   | <input checked="" type="checkbox"/> Hive 3.1.3       |
| <input checked="" type="checkbox"/> Hue 4.11.0            | <input checked="" type="checkbox"/> JupyterEnterpriseGateway 2.6.0 | <input checked="" type="checkbox"/> JupyterHub 1.5.0 |
| <input checked="" type="checkbox"/> Livy 0.8.0            | <input type="checkbox"/> MXNet 1.9.1                               | <input type="checkbox"/> Oozie 5.2.1                 |
| <input type="checkbox"/> Phoenix 5.1.3                    | <input type="checkbox"/> Pig 0.17.0                                | <input type="checkbox"/> Presto 0.284                |
| <input checked="" type="checkbox"/> Spark 3.5.0           | <input type="checkbox"/> Sqoop 1.4.7                               | <input type="checkbox"/> TensorFlow 2.11.0           |
| <input type="checkbox"/> Tez 0.10.2                       | <input type="checkbox"/> Trino 435                                 | <input type="checkbox"/> Zeppelin 0.10.1             |
| <input type="checkbox"/> ZooKeeper 3.9.1                  |  |  |

AWS Glue Data Catalog settings

Use the AWS Glue Data Catalog to provide an external metastore for your application.

- ☒ Use for Hive table metadata
- ☒ Use for Spark table metadata

Operating system options [Info](#)

- ☒ Amazon Linux release
- ☐ Custom Amazon Machine Image (AMI)

### Summary [Info](#)

#### Name and applications - required

Name

flight-delay-prediction

Amazon EMR release

emr-7.1.0

Application bundle

Custom (Hadoop 3.3.6, Hive 3.1.3, Hue 4.11.0, JupyterEnterpriseGateway 2.6.0, JupyterHub...)

#### Cluster configuration - required

Uniform instance groups

Primary (m5.xlarge), Core (m5.xlarge), Task (m5.xlarge)

#### Cluster scaling and provisioning - required

Provisioning configuration

Core size: 1 instance

Task size: 1 instance

Cancel

Clone cluster

Then, we created the EMR cluster. We choose the version and select the services we need such as Spark, Hue, Hadoop, JupyterHub, etc. We also choose the necessary options such as the AWS Glue Data Catalog settings for using Hive and Spark table metadata.

Service role

EMR\_Flight\_Delay\_Role

### EC2 instance profile for Amazon EMR

The instance profile assigns a role to every EC2 instance in a cluster. The instance profile must specify a role that can access the resources for your steps and bootstrap actions.

☐ Choose an existing instance profile

Select a default role or a custom instance profile with IAM policies attached so that your cluster can interact with your resources in Amazon S3.

☒ Create an instance profile

Let Amazon EMR create a new instance profile so that you can specify a custom set of resources for it to access in Amazon S3.

### S3 bucket access

[Info](#)

☒ Specific S3 buckets or prefixes in your account [Info](#)

Choose the buckets or prefixes that you want this instance profile to access.

☐ All S3 buckets in this account with read and write access

Grant the instance profile access to all buckets that have read and write access enabled in your account.

### S3 buckets

We've already added the resources that you configured in the [Cluster logs](#) section. Choose the S3 buckets and bucket prefixes where you store logs and data for your cluster, bootstrap actions, and steps.

### S3 URI

s3://flight-delay-bucket-output



[View](#)

[Browse S3](#)

[Add](#)

### S3 bucket

### Prefix

### Permission

aws-logs-98892675...

Inherited from Cluster logs

elasticmapreduce

Read and write

[Edit](#)

### Custom automatic scaling role - optional

When a custom automatic scaling rule triggers, Amazon EMR assumes this role to add and terminate EC2 instances. [Learn more](#)

### Custom automatic scaling role

Choose IAM role



[Create IAM role](#)

Name

flight-delay-prediction

Amazon EMR release

emr-7.1.0

Application bundle

Custom (Hadoop 3.3.6, Hive 3.1.3, Hue 4.11.0, JupyterEnterpriseGateway 2.6.0, JupyterHub...)

### Cluster configuration - required

Uniform instance groups

Primary (m5.xlarge), Core (m5.xlarge), Task (m5.xlarge)

### Cluster scaling and provisioning - required

Provisioning configuration

Core size: 1 instance

Task size: 1 instance

[Cancel](#)

[Clone cluster](#)

After choosing the keypair file, we select the service role we created and choose an instance profile and select an S3 bucket. This is where our preprocessed outputs will be stored. Finally, we create the cluster.



After creating the EMR cluster, the cluster takes some time to get all the components ready and turns to the waiting state and it ready for use.

aws

Services

Search

[Alt+S]

Amazon EMR

EMR Serverless

EMR on EC2

Clusters

Notebooks and Git repos

Events

Amazon EMR

EMR on EC2: Clusters

Clusters (24) Info

Filter clusters by status

Find clusters

Filter clusters by creation date-time

	Cluster ID	Cluster name	Status	Creation time (UTC-05:00)	Elapsed time	Normalized instance hours
	j-3S6KFFP542L51	flight-delay-prediction	Waiting Ready to run steps	April 29, 2024, 08:54	1 hour	0

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

- User groups
- Users**
- Roles
- Policies
- Identity providers
- Account settings

Access reports

- Access Analyzer
  - External access
  - Unused access
  - Analyzer settings
- Credential report
- Organization activity
- Service control policies

IAM > Users > big-data-project

big-data-project

Delete

Summary

ARN  
am:aws:iam::988926753451:user/big-data-project

Console access  
Enabled without MFA

Access key 1  
Create access key

Created  
April 18, 2024, 23:47 (UTC-05:00)

Last console sign-in  
Today

Permissions

Groups

Tags

Security credentials

Access Advisor

Permissions policies (1)

Permissions are defined by policies attached to the user directly or through groups.

Search

Filter by Type  
All types

< 1 >

	Policy name	Type	Attached via
<input type="checkbox"/>	AdministratorAccess	AWS managed - job function	Directly

Permissions boundary (not set)

We need to create an IAM user since we need to create a Studio and a Notebook in that studio and attach the created EMR cluster to it. If we try to do this in the root user it gives an error. So, it is necessary to create an IAM user with administrator access.

**Workspaces (Notebooks) (1/1)** [Info](#)

EMR Notebooks are now EMR Studio Workspaces. You can organize and run interactive notebooks in Workspaces.

All Studios

	Workspace name	Studio name	Status	Cluster ID	Creation time (UTC-05:00)	Last modified by	Last modified (UTC-05:00)
	<a href="#">flight-delay-prediction</a>	<a href="#">big-data-project</a>	Idle	<a href="#">j-WBH85IBAQRXF</a>	April 23, 2024, 09:17	big-data-project	April 28, 2024, 21:30

Workspace which we used to open a Jupyter Notebook for writing the PySpark script

Working with EMR Studio Workspaces

**Discover Workspaces**

Your EMR Notebooks are called EMR Studio Workspaces in the new console.

- Create and organize your Workspaces in secure Studios.
- Collaborate with other members of your team to write and run notebook code.
- Run Jupyter and JupyterLab notebooks in a single Workspace.
- Connect to Git repos from the Git tab in the left sidebar of JupyterLab.
- Browse data with SQL Explorer.
- Provision EMR clusters with Service Catalog.

[Learn more](#)

**Find and launch Workspaces**

**Attach cluster**

☒ **Launch in JupyterLab**  
Use the latest web-based interactive development environment for notebooks.

☐ **Launch in Jupyter**  
Use the classic development environment for notebooks.

**EMR cluster**  
Choose an EMR cluster to attach to your Workspace.  
[j-3S6KFFP542L51 \(flight-delay-prediction\)](#)

**Cluster security group**  
Choose the security group that will communicate between the Workspace and the attached Amazon EMR cluster running on Amazon EC2.  
[sg-033ac5b57b59f53a3 \(DefaultEngineSecurityGroup\)](#)

**Workspace security group**  
Choose the security group that will allow the Workspace to route traffic to the internet and enable linking Git repositories to the Workspace.  
[sg-06272f374a1cd61e0 \(DefaultWorkspaceSecurityGroupGit\)](#)

[Cancel](#) [Attach cluster and launch](#)

**Attach clusters to Workspaces**

Attach a Workspace to an Amazon EMR compute cluster in the console or in JupyterLab.

- Attach in the Amazon EMR console:
  - Select your Workspace, then select Launch Workspace > Launch with cluster.
  - Choose an EMR cluster to attach to your Workspace.
- Attach in JupyterLab:
  - Select your Workspace, then select Launch Workspace > Quick launch.
  - Inside JupyterLab, open the Cluster tab in the left sidebar.
  - Choose either an EMR on EC2 or EMR on EKS cluster type, find the cluster in the dropdown, then attach it to your Workspace.

[Learn more](#)

We attach the EMR cluster to the notebook and launch it

# Description of Preprocessing Code

## ► Data Cleaning

The process begins by dropping rows with null values in certain crucial columns. This ensures that the dataset used for modeling does not include missing data which could impact model accuracy.

```
# We start by Dropping rows with any null values in specific columns
columns_to_check = [
    "airline", "airportfrom", "airportto", "flight", "dayofweek", "time",
    "length", "delay"
]
df = df.dropna(subset=columns_to_check)
```

Since time represents time in minutes (assuming out of 1440, which are the number of minutes in a day), it is transformed into two cyclic components using trigonometric functions. This transformation helps in preserving the cyclical nature of time (e.g., minute 0 and minute 1440 are the same time, midnight).

```
# Convert 'time' from minutes to cyclic representation
df = df.withColumn("time_rad", radians(col("time") / 1440 * 360))\
    .withColumn("time_sin", sin(col("time_rad")))\
    .withColumn("time_cos", cos(col("time_rad")))
```

## ► Indexing and Encoding of Categorical Variables

Categorical variables (airline, airportfrom, airportto, flight) are indexed using StringIndexer, which assigns a numeric value to each unique category level. This transformation is necessary because most machine learning algorithms require numerical input.

```
# Indexing and encoding categorical columns
categorical_columns = ["airline", "airportfrom", "airportto", "flight"]
indexers = [
    StringIndexer(inputCol=c, outputCol=c + "_index", handleInvalid="keep")
    for c in categorical_columns
]
for indexer in indexers:
    df = indexer.fit(df).transform(df)
```

After indexing, these numeric indices are converted into one-hot encoded vectors using OneHotEncoder. This encoding converts categorical integer-based features into a vector of binary features, with each unique category in a column represented by a distinct binary feature.

```
encoder = OneHotEncoder(inputCols=[c + "_index" for c in categorical_columns],
                        outputCols=[c + "_vec" for c in categorical_columns])
df = encoder.fit(df).transform(df)
```

## ► Feature Vector Assembly

All the features, including newly created one-hot encoded vectors and numeric features (dayofweek, length, time\_sin, time\_cos), are assembled into a single feature vector using VectorAssembler. This vector is what will be fed into the machine learning algorithms.

```
# Assemble all features, including numeric features
numeric_cols = ["dayofweek", "length", "time_sin", "time_cos"]
feature_cols = [c + "_vec" for c in categorical_columns] + numeric_cols
assembler = VectorAssembler(inputCols=feature_cols,
                             outputCol="assembled_features")
df_assembled = assembler.transform(df)
```

## ► Feature Scaling

The assembled feature vector is then scaled using StandardScaler. This is important because it prevents features with larger ranges from dominating the distance calculations.

```
# Scaling the assembled features
scaler = StandardScaler(inputCol="assembled_features",
                        outputCol="features",
                        withStd=True,
                        withMean=False)
scalerModel = scaler.fit(df_assembled)
df_scaled = scalerModel.transform(df_assembled)
```

## ► Preparing the Final Dataset

Finally, the scaled feature vectors along with the labels are selected into a new DataFrame. This DataFrame (final\_df) now holds the processed data ready for machine learning training and testing.

```
# Select features and label for machine learning
final_df = df_scaled.select(col("features"),
                             col("delay").cast(DoubleType()).alias("label"))
```

## Storing the pre-processed data as a Parquet file in S3

```
# This is the path to the S3 bucket that will store the parquet file
s3_path = "s3a://flight-delay-bucket-output/spark-output/"

# Saving the the DataFrame as a Parquet file to S3
final_df.write.mode("overwrite").parquet(s3_path)
```

## Storing the pre-processed data as a table in Glue

```
: # database and table name that will store preprocessed data in Glue
database_name = "flight_delay_prediction"
table_name = "processed_flight_data"

# Create the database if it doesn't exist
spark.sql(f"CREATE DATABASE IF NOT EXISTS `{database_name}`")

# Drop the table if it exists
spark.sql(f"DROP TABLE IF EXISTS `{database_name}`.`{table_name}`")

# Creating a new table
spark.sql(f"""
    CREATE TABLE `{database_name}`.`{table_name}`
    USING parquet
    OPTIONS (path = '{s3_path}')
""")
```

# ML Algorithms used

1. **Logistic Regression:** A statistical model that estimates the probability of a binary outcome based on one or more predictor variables.
2. **Cross Validation and Grid Search:** Techniques used to evaluate model performance with different hyperparameter settings to find the optimal set of parameters for a model.
3. **Decision Trees:** A non-linear predictive model that divides the feature space into regions by making sequential, hierarchical decisions about the data based on feature values.
4. **Random Forest:** An ensemble learning method that builds multiple decision trees and merges them together to get a more accurate and stable prediction.
5. **SVM (Support Vector Machine):** A powerful classification technique that finds the hyperplane which best separates different classes by maximizing the margin between closest data points of the classes.



# Detailed steps on how the integration of Big Data and Data Science was done

## ► Use of Spark and PySpark

We used PySpark which is python's api for Spark to use Spark's capabilities in distributed data processing of the large Airlines dataset

In this code snippet, we initialized the Spark session and then load the data from the distributed storage system.

```
# Initialize Spark Session
spark = SparkSession.builder.appName(
    "FlightDelayPreprocessingNew").enableHiveSupport().getOrCreate()

# Load the dataset
df = spark.sql(
    "SELECT * FROM `flight-data-glue-crawler-output`.flight_delay_prediction_bigdata"
)
```

```
df.show()
```

```
VBox()
```

```
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),...
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
| id|airline|flight|airportfrom|airportto|dayofweek|time|length|delay|
+-----+-----+-----+-----+-----+-----+-----+-----+
|NULL|Airline|  NULL|AirportFrom|AirportTo|        NULL|NULL|  NULL|  NULL|
|  1|   CO|  269|      SFO|      IAH|        3|  15|  205|    1|
|  2|   US| 1558|      PHX|      CLT|        3|  15|  222|    1|
|  3|   AA| 2400|      LAX|      DFW|        3|  20|  165|    1|
|  4|   AA| 2466|      SFO|      DFW|        3|  20|  195|    1|
|  5|   AS|  108|      ANC|      SEA|        3|  30|  202|    0|
|  6|   CO| 1094|      LAX|      IAH|        3|  30|  181|    1|
|  7|   DL| 1768|      LAX|      MSP|        3|  30|  220|    0|
|  8|   DL| 2722|      PHX|      DTW|        3|  30|  228|    0|
|  9|   DL| 2606|      SFO|      MSP|        3|  35|  216|    1|
| 10|   AA| 2538|      LAS|      ORD|        3|  40|  200|    1|
| 11|   CO|  223|      ANC|      SEA|        3|  49|  201|    1|
| 12|   DL| 1646|      PHX|      ATL|        3|  50|  212|    1|
| 13|   DL| 2055|      SLC|      ATL|        3|  50|  210|    0|
| 14|   AA| 2408|      LAX|      DFW|        3|  55|  170|    0|
| 15|   AS|  132|      ANC|      PDX|        3|  55|  215|    0|
| 16|   US|  498|      DEN|      CLT|        3|  55|  179|    0|
| 17|   B6|   98|      DEN|      JFK|        3|  59|  213|    0|
| 18|   CO| 1496|      LAS|      IAH|        3|  60|  162|    0|
| 19|   DL| 1450|      LAS|      MSP|        3|  60|  181|    0|
+-----+-----+-----+-----+-----+-----+-----+-----+
```

only showing top 20 rows

After the Preprocessing stage, the data was stored in S3 as a parquet file as mentioned in the previous slides. This data is loaded for the training, test data split.

```
# Define the path to your S3 bucket
s3_path = "s3a://flight-delay-bucket-output/spark-output/"

# Read the parquet file
df = spark.read.parquet(s3_path)
```

This is the preprocessed data that was used to build the ML models.

```
[6]: df.show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:', layout=Layout(height='25px', width='50%'),...
+-----+-----+
|          features|label|
+-----+-----+
|(7193,[9,27,316,1...| 1.0|
|(7193,[5,24,321,1...| 1.0|
|(7193,[3,22,313,2...| 1.0|
|(7193,[3,27,313,1...| 1.0|
|(7193,[15,90,330,...| 0.0|
|(7193,[9,22,316,1...| 1.0|
|(7193,[1,22,324,2...| 0.0|
|(7193,[1,24,318,2...| 0.0|
|(7193,[1,27,324,3...| 1.0|
|(7193,[3,26,312,2...| 1.0|
|(7193,[9,90,330,1...| 1.0|
|(7193,[1,24,311,1...| 1.0|
|(7193,[1,30,311,4...| 0.0|
|(7193,[3,22,313,2...| 0.0|
|(7193,[15,90,346,...| 0.0|
|(7193,[5,21,321,7...| 0.0|
|(7193,[12,21,326,...| 0.0|
|(7193,[9,26,316,2...| 0.0|
|(7193,[1,26,324,3...| 0.0|
|(7193,[9,75,316,1...| 0.0|
+-----+-----+
only showing top 20 rows
```

In this step, we loaded the pre-processed data from the Glue Table and used it for building the Decision Tree model.

```
[17]: df_loaded = spark.sql(f"SELECT * FROM `{database_name}`.`{table_name}`")
df_loaded.show()
```

► Spark Job Progress

```
+-----+
|          features|label|
+-----+
|(7193,[9,27,316,1...| 1.0|
|(7193,[5,24,321,1...| 1.0|
|(7193,[3,22,313,2...| 1.0|
|(7193,[3,27,313,1...| 1.0|
|(7193,[15,90,330,...| 0.0|
|(7193,[9,22,316,1...| 1.0|
|(7193,[1,22,324,2...| 0.0|
|(7193,[1,24,318,2...| 0.0|
|(7193,[1,27,324,3...| 1.0|
|(7193,[3,26,312,2...| 1.0|
|(7193,[9,90,330,1...| 1.0|
|(7193,[1,24,311,1...| 1.0|
|(7193,[1,30,311,4...| 0.0|
|(7193,[3,22,313,2...| 0.0|
|(7193,[15,90,346,...| 0.0|
|(7193,[5,21,321,7...| 0.0|
|(7193,[12,21,326,...| 0.0|
|(7193,[9,26,316,2...| 0.0|
|(7193,[1,26,324,3...| 0.0|
|(7193,[9,75,316,1...| 0.0|
+-----+
only showing top 20 rows
```

```
[18]: # Assuming 'final_df' is your DataFrame loaded with features and labels
train_data, test_data = df_loaded.randomSplit([0.7, 0.3], seed=42)
```

```
[*]: from pyspark.ml.classification import DecisionTreeClassifier

# Initialize the Decision Tree Classifier
dt = DecisionTreeClassifier(featuresCol='features', labelCol='label')

# Fit the model
dt_model = dt.fit(train_data)

# Make predictions
dt_predictions = dt_model.transform(test_data)
```

Using PySpark's MLlib for training a Logistic Regression Model, Decision Tree Model, Gradient Boosting Model, etc. and tuning it with CrossValidator and ParamGridBuilder demonstrates applying machine learning at scale.

## Distributed model training and hyperparameter tuning:

only showing top 20 rows

```
[6]: # Split the data into training and test sets
train_data, test_data = df.randomSplit([0.8, 0.2], seed=42)

print(test_data)
```

DataFrame[features: vector, label: double]

```
[8]: from pyspark.ml.classification import LogisticRegression

# Initialize the classifier
lr = LogisticRegression(featuresCol='features', labelCol='label')

# Train the model
lr_model = lr.fit(train_data)

# Make predictions
predictions = lr_model.transform(test_data)

predictions.show()
```

features	label	rawPrediction	probability	prediction
(7193,[0,21,315,8...]	1.0	[-1.3844923500020...]	[0.20028847765277...]	1.0
(7193,[0,21,315,1...]	1.0	[-0.9865878700456...]	[0.27158656378124...]	1.0
(7193,[0,21,315,1...]	1.0	[-0.9191408578134...]	[0.28513298277851...]	1.0
(7193,[0,21,315,1...]	0.0	[-0.9248175703464...]	[0.28397729890871...]	1.0
(7193,[0,21,315,1...]	0.0	[-1.0586024986565...]	[0.25757660958457...]	1.0
(7193,[0,21,315,2...]	1.0	[-1.7788122061849...]	[0.14444986453704...]	1.0
(7193,[0,21,317,7...]	1.0	[-1.8538168247012...]	[0.13542538058137...]	1.0
(7193,[0,21,317,8...]	1.0	[-1.2828379106776...]	[0.21706753562499...]	1.0
(7193,[0,21,317,1...]	1.0	[-1.1304508034455...]	[0.24407791632155...]	1.0
(7193,[0,21,317,1...]	1.0	[-1.0967272973294...]	[0.25035360240566...]	1.0
(7193,[0,21,317,1...]	0.0	[-1.0630037912133...]	[0.25673584488195...]	1.0
(7193,[0,21,317,2...]	1.0	[-2.2384973879144...]	[0.09634628532123...]	1.0
(7193,[0,21,317,2...]	1.0	[-2.1710503756823...]	[0.10238046495581...]	1.0
(7193,[0,21,317,3...]	0.0	[-1.3015819309218...]	[0.21389890095924...]	1.0
(7193,[0,21,318,2...]	1.0	[-0.3699064159661...]	[0.40856363496267...]	1.0
(7193,[0,21,319,9...]	1.0	[-1.5262702464488...]	[0.17854004879107...]	1.0
(7193,[0,21,319,1...]	0.0	[-1.4708395641821...]	[0.18681503788738...]	1.0
(7193,[0,21,320,8...]	1.0	[-1.2415370096564...]	[0.22416856027272...]	1.0
(7193,[0,21,320,3...]	1.0	[-1.7887202390152...]	[0.14322969759425...]	1.0
(7193,[0,21,322,1...]	0.0	[-1.4442272091581...]	[0.19089159441491...]	1.0

only showing top 20 rows

## Hyperparameter Tuning using Cross Validators and Grid Search

```
[11]: from pyspark.ml.classification import LogisticRegression
      from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
      from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

      # Initialize the classifier
      lr = LogisticRegression(featuresCol='features', labelCol='label')

      # Create the parameter grid
      paramGrid = ParamGridBuilder() \
        .addGrid(lr.regParam, [0.01, 0.1, 0.5]) \
        .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
        .build()

      # Evaluator for binary classification (area under ROC)
      binaryEvaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

      # Set up 3-fold cross-validation
      crossval = CrossValidator(estimator=lr,
                               estimatorParamMaps=paramGrid,
                               evaluator=binaryEvaluator,
                               numFolds=3)

      # Fit the model
      cvModel = crossval.fit(train_data)

      # Use the best model to make predictions on the test data
      predictions = cvModel.transform(test_data)
```

Cross Validation utilizes a 3-fold technique to train and validate the model on different subsets which ensures the model becomes robust and there is less overfitting by evaluating the performance of the model on all data splits.

Grid Search performs exhaustive search over specific parameter combinations to find the optimal model settings that maximises the area under the ROC curve which improves the model accuracy and generalization.

This combination efficiently optimizes and validates the logistic regression model for reliable predictions on new data.

Evaluating the Logistic Regression Model using Metrics like Accuracy, f1 score, recall, precision are computed efficiently across potentially large testing datasets.

```
# Calculate and print precision, recall, F1 score, and accuracy
precision = evaluatorMulti.evaluate(
    predictions, {evaluatorMulti.metricName: "precisionByLabel"})
recall = evaluatorMulti.evaluate(predictions,
    {evaluatorMulti.metricName: "recallByLabel"})
f1 = evaluatorMulti.evaluate(predictions, {evaluatorMulti.metricName: "f1"})
accuracy = evaluatorMulti.evaluate(
    predictions, {evaluatorMulti.metricName: "accuracy"}) # Added for accuracy

# Evaluator for binary classification (area under ROC)
binaryEvaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

auc = binaryEvaluator.evaluate(predictions)
print("Area under ROC = " + str(auc))
print("Precision = %g" % precision)
print("Recall = %g" % recall)
print("F1 Score = %g" % f1)
print("Accuracy = %g" % accuracy)
```

Overall, these elements of the code demonstrate how data science techniques (preprocessing, feature engineering, machine learning, model evaluation) are integrated with big data technologies (Spark, Hive) to handle large datasets and complex computations efficiently and scalably.

# SparkSQL Queries

## Query to load the data from Glue

```
df = spark.sql(  
    "SELECT * FROM `flight-data-glue-crawler-output`.flight_delay_prediction_bigdata"  
)
```

## Query to count the number of delays per airport

```
# Query to count the number of delays per airport  
delays_per_airport = spark.sql("""  
    SELECT airportfrom, COUNT(*) AS number_of_delays  
    FROM flights  
    WHERE delay > 0  
    GROUP BY airportfrom  
    ORDER BY number_of_delays DESC  
    """)  
  
delays_per_airport.show()
```

airportfrom	number_of_delays
ATL	14601
ORD	11906
DEN	9433
DFW	8809
LAX	8270
IAH	7665
PHX	6844
LAS	6674
DTW	6300
SFO	6281
SLC	5532
MDW	5222
MSP	4961
BWI	4910
EWR	4740
MCO	4598
CLT	4582
JFK	3935
BOS	3722
SAN	3402

only showing top 20 rows



## Delay Frequency by Day of Week

*#Delay Frequency by Day of Week*

```
delay_frequency_by_dayofweek = spark.sql("""SELECT dayofweek, COUNT(*) AS delay_count
FROM flights
WHERE delay > 0
GROUP BY dayofweek
ORDER BY dayofweek""")
```

```
delay_frequency_by_dayofweek.show()
```

dayofweek	delay_count
1	34030
2	31913
3	42254
4	41244
5	35515
6	23615
7	31693

## Query to Find the Airline with the Maximum Delay

*: #Query to Find the Airline with the Maximum Delay*

```
most_delays_airline_query = """
SELECT airline, COUNT(*) AS num_delays
FROM flights
WHERE delay = 1
GROUP BY airline
ORDER BY num_delays DESC
LIMIT 1;
"""
```

```
most_delays_airline = spark.sql(most_delays_airline_query)
most_delays_airline.show()
```

airline	num_delays
WN	65657

## Query to find the number of delays per airline

```
# Query to find the number of delays per airline
delays_per_airline_query = """
SELECT airline, COUNT(*) AS num_delays
FROM flights
WHERE delay = 1
GROUP BY airline
ORDER BY num_delays DESC;
"""

# Execute the query using Spark
delays_per_airline = spark.sql(delays_per_airline_query)
delays_per_airline.show()
```

airline	num_delays
WN	65657
DL	27452
OO	22760
AA	17736
MQ	12742
CO	11957
XE	11795
US	11591
EV	11255
UA	8946
B6	8459
9E	8226
FL	6275
AS	3892
OH	3502
YV	3334
F9	2899
HA	1786

# Data Visualization for Analysis Trends in Data

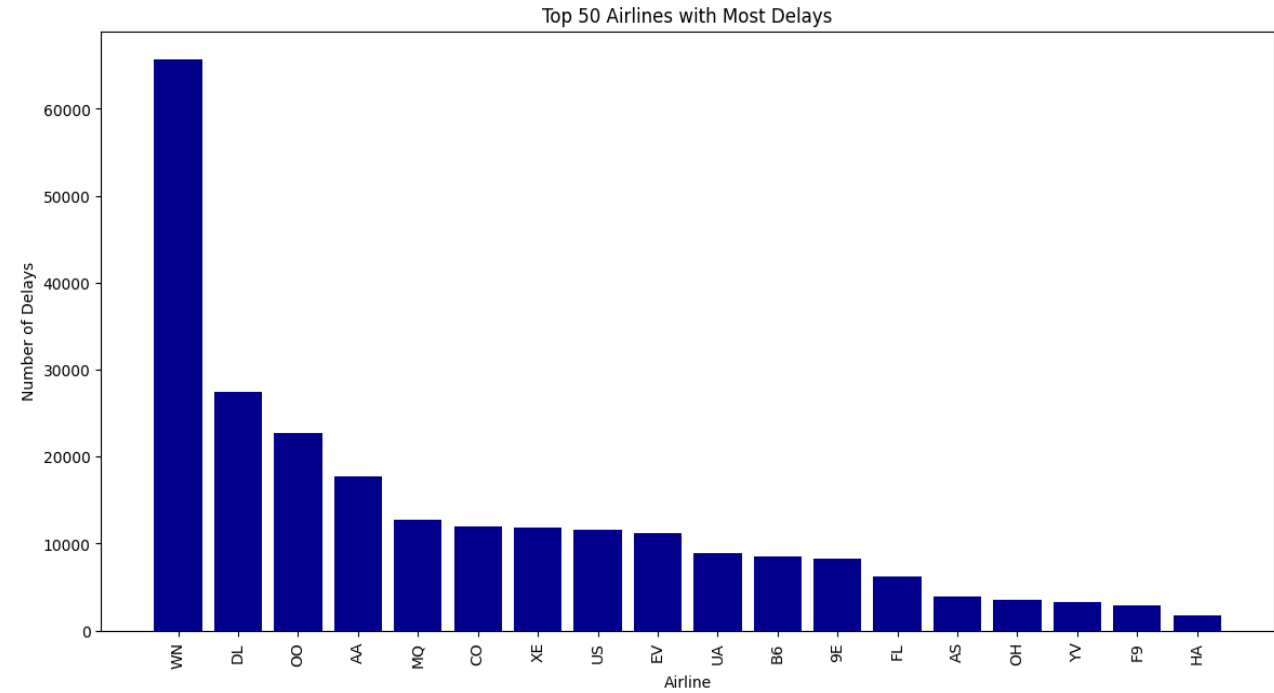
```
import pandas as pd

pd_df = most_delays_airline.toPandas()

import matplotlib.pyplot as plt

# Creating a bar plot
plt.figure(figsize=(14, 7))
plt.bar(pd_df['airline'], pd_df['num_delays'], color='darkblue')
plt.xlabel('Airline')
plt.ylabel('Number of Delays')
plt.title('Top 50 Airlines with Most Delays') # Title of the plot
plt.xticks(rotation=90)
plt.show() # Displaying the plot
```

Code for plotting the graph



Plot showing 'Top 50 Airlines with Most Delays'

```

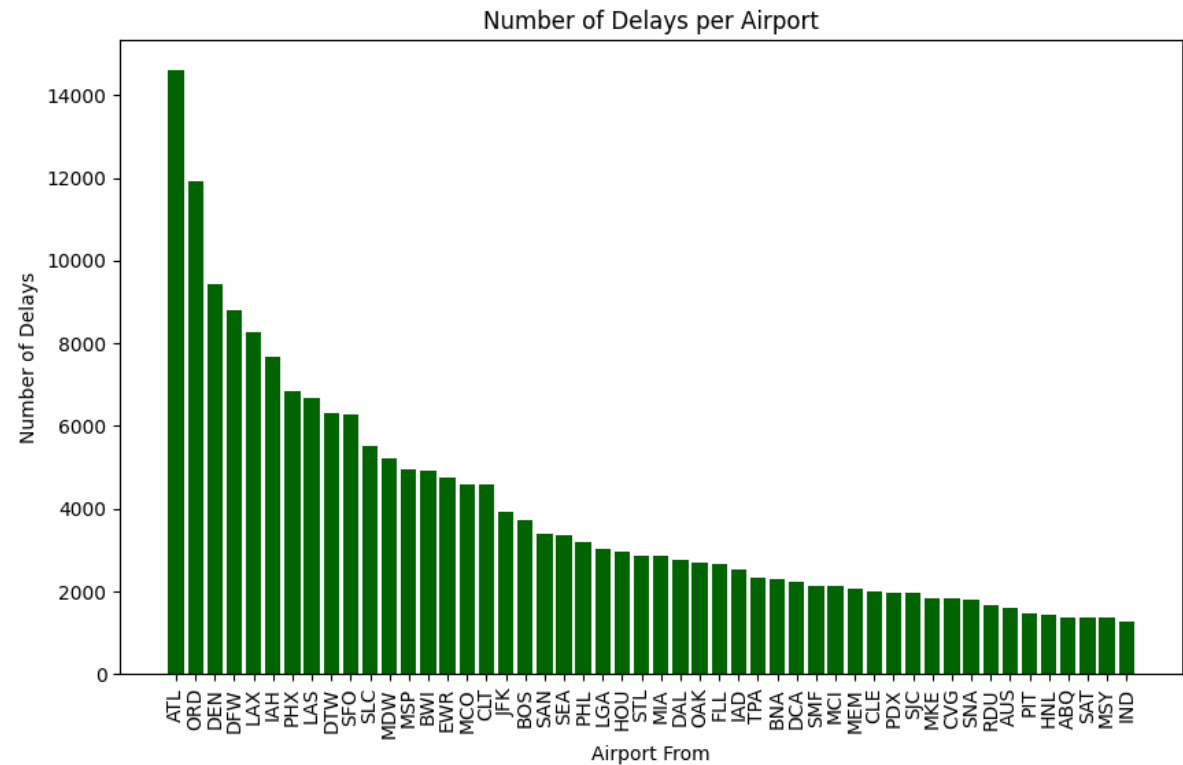
df.createOrReplaceTempView("flights")

delays_per_airport = spark.sql("""
SELECT airportfrom, COUNT(*) AS number_of_delays
FROM flights
WHERE delay > 0
GROUP BY airportfrom
ORDER BY number_of_delays DESC
LIMIT 50
""")
delays_per_airport_pd = delays_per_airport.toPandas()

import matplotlib.pyplot as plt
import pandas as pd
# Creating a bar plot
plt.figure(figsize=(10, 6)) # Setting the figure size
plt.bar(delays_per_airport_pd['airportfrom'], delays_per_airport_pd['number_of_delays'], color='darkgreen')
plt.xlabel('Airport From') # X-axis label
plt.ylabel('Number of Delays') # Y-axis label
plt.title('Number of Delays per Airport') # Title of the plot
plt.xticks(rotation=90)
plt.show() # Displaying the plot

```

Code for plotting the graph



Plot showing 'Number of Delays per Airport'

# Evaluation Metrics of ML Models

Model	Area under ROC	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.6997	0.6533	0.6659	0.7565	0.6482
Logistic Regression (Cross-Validation)	0.6995	0.6539	0.6514	0.6539	0.6482
SVM	0.6853	0.6458	0.6467	0.6458	0.6315
Decision Tree	0.4504	0.6372	0.6434	0.6372	0.6139
Random Forest	0.6059	0.5638	0.6204	0.5638	0.4249
Gradient-Boosted Trees	0.6883	0.6455	0.6495	0.6455	0.6273

# Summary of Evaluation Metrics

- ▶ Logistic Regression and Logistic Regression with Cross Validation and Grid Search both showed relatively high accuracy with scores around 65.3% and 65.4% respectively. This shows that these models perform good in classifying the data correctly.
- ▶ Support vector machine also performed well with an accuracy of about 64.6% making it a close contender for logistic regression.
- ▶ Decision Trees had a slightly low accuracy of 63.7%. Even though it is a very simple model, it performed decently although it lagged a little behind.
- ▶ Random Forest showed the lowest accuracy from all the models with just 56.4% which could suggest it might be overfitting.
- ▶ Gradient-Boosted Trees gave an accuracy of 64.5% placing it in the middle range of the models tested which shows that it has a good balance in classifying instances correctly.

Overall, logistic regression, either with or without the hyperparameter tuning, seems to be the most accurate. However, Gradient-Boosted Trees and SVM also provide competitive options depending on the nature and complexity of the data.

# Conclusion

- ▶ Utilized Big Data technologies like Apache Spark, AWS EMR, S3 and Glue to efficiently handle and process the large Airlines dataset
- ▶ Applied various Machine Learning models such as logistic regression, SVM, decision trees, random forests, gradient-boosted trees to predict flight delays and evaluated their effectiveness through metrics such as accuracy, precision, recall, F1 score, and ROC area.
- ▶ Highlighted the best performing model and its metrics to show its effectiveness of the data analytics approach.
- ▶ Insights can aid airlines in reducing delays and enhancing customer satisfaction by understanding and mitigating delay factors.

# References

Sternberg, A., Soares, J., Carvalho, D., & Ogasawara, E. (2017). A review on flight delay prediction. arXiv preprint arXiv:1703.06118.

Belcastro, L., Marozzo, F., Talia, D., & Trunfio, P. (2016). Using scalable data mining for predicting flight delays. ACM Transactions on Intelligent Systems and Technology (TIST), 8(1), 1-20.

Jiang, Y., Liu, Y., Liu, D., & Song, H. (2020, August). Applying machine learning to aviation big data for flight delay prediction. In 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCCom/CyberSciTech) (pp. 665-672). IEEE.

<https://www.kaggle.com/datasets/jimschacko/airlines-dataset-to-predict-a-delay>

<https://docs.aws.amazon.com>