SEIS 736 Project

Flight Data Analysis with Apache Spark

Mansoo Cho

SEIS 736

1. Summary

1-a. overview:

This project will use Apache Spark to analyze flight dataset from Kaggle. Instead of using AWS EMR cluster, I've decided to manually set up the resources on AWS. I created a VPC to securely launch two EC2 instances for Data Storage & Analysis, and Backup Storage. The Data Storage and Analysis instance leverages Spark (specifically Pyspark) and Jupyter Notebook as an IDE to run Pyspark scripts.

1-b. Dataset description:

(Data source: Kaggle; Original data source: Bureau of Transportation Statistics)

Dataset name: Marketing Carrier On-Time Performance

The dataset contains US flight information from January 2018 to July 2022. It contains over 1000 different marketing airline carrier information. Each airline reports the scheduled/actual arrival and departure times, taxi-in and taxi-out times, airtime, flight distance and such. Each record in the dataset represents individual flight information.

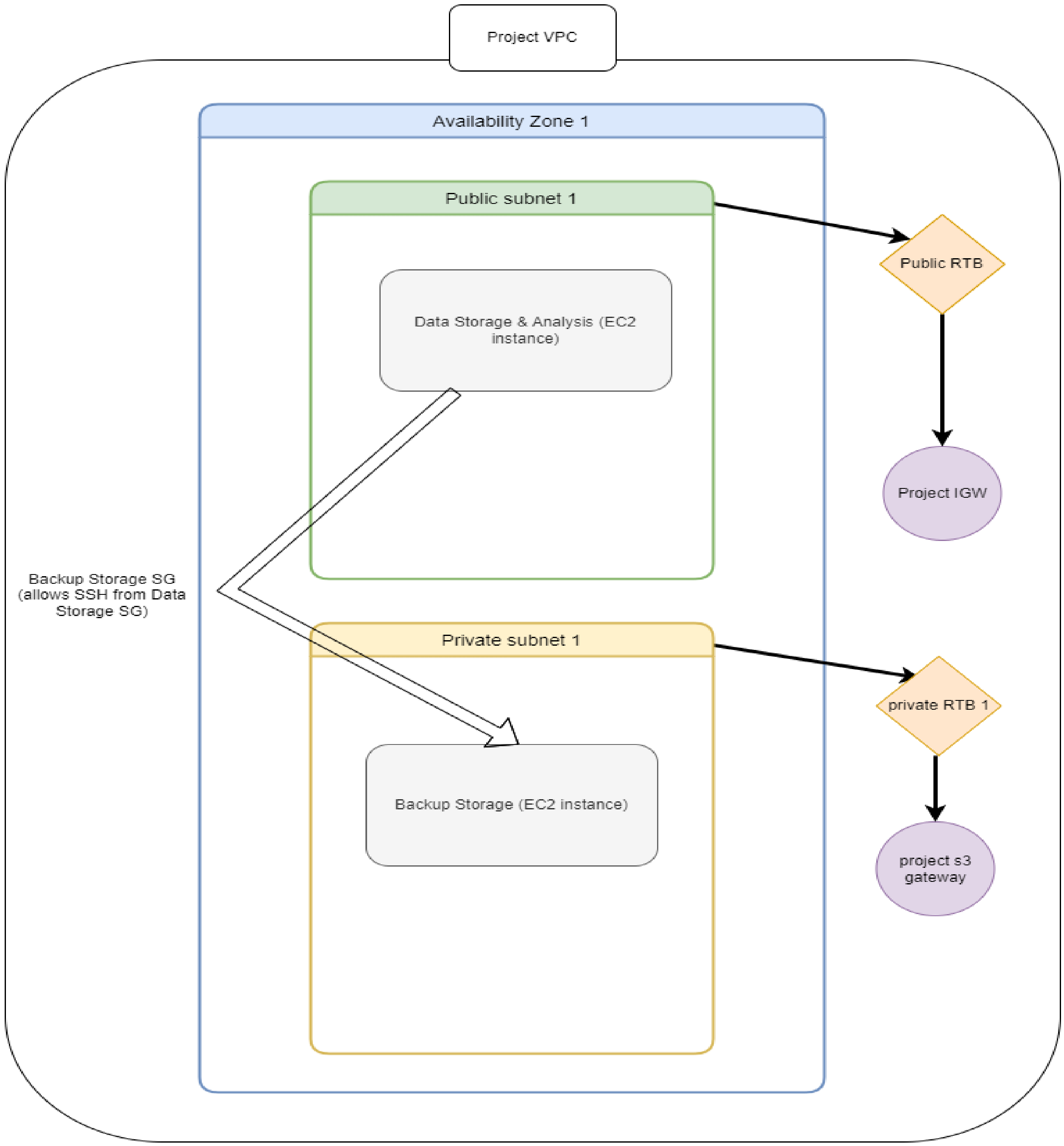
There are three main branches of files in this dataset: 1. Airlines.csv, 2. Raw csv files and 3. Combined flights csv and parquet files. Airlines.csv lists all the airlines present in the dataset along with the codes. The raw csv files are collected on a monthly basis. The combined data files are aggregated on an annual basis. I'll be working with the combined parquet files for this project. A detailed data dictionary is included in Appendix A.

1-c. reasons for choosing this data:

I chose this dataset because it was suitable for Big Data problems. The entire data set is larger than 20 GB and is not easily workable with common data tools (such as Excel). It also has great potential for multiple analyses. It contains a variety of data elements—flight distance, date and time, locations, delays, and cancellation information, etc. The distance element would allow for some geospatial analysis, the time element can be used for Time Series Analysis, and so on.

2. Setting up the environment

2-a. Create a VPC on AWS and launch EC2 instances:



(Figure 1. VPC design)

The diagram above describes how the network is set up. Data Storage and Analysis instance is stored in Public Subnet 1, and its security group (Data Storage SG) allows inbound traffic from TCP ports 22 (for SSH) and 8888 (Jupyter Notebook on browser). Backup Storage instance is set up in private subnet 1—its security group (Backup Storage SG) allows inbound SSH traffic from the Data Storage instance only.

Data storage instance will be assigned a public IP and is accessible through the internet (ip source specified as my machine's ip). However, the Backup Storage instance will not be accessible through a public ip. We can only SSH into the instance from Data Storage instance. Both Data storage and Backup Storage instances use the same keypair (project-keypair.ppk).

*** Sections 2-b through 2-c will describe detailed steps of how I set up the environment for storage and analysis.**

2-b. Login to Data Storage instance, and install Python3 and packages (Jupyter and Kaggle):

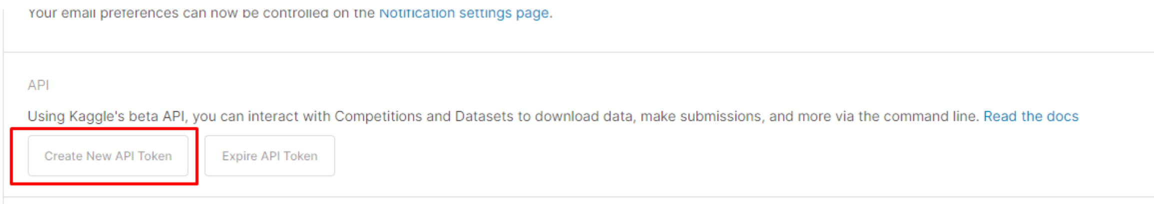
First SSH into your DSA. Then type these commands in terminal. This will install the necessary packages.

Commands:

```
sudo yum install python3
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py
pip install jupyter kaggle
```

2-c. Extract the flight dataset from Kaggle:

Download the dataset using Python Kaggle API. To download the dataset, a credential is required. It can be downloaded from your Kaggle account page. (Click the "Create New API Token" button to generate the credential.) Once created, upload the generated JSON file to the Data Storage instance in the directory `/home/ec2-user/.kaggle`. (I used WinSCP to transfer the file from my Windows laptop.)

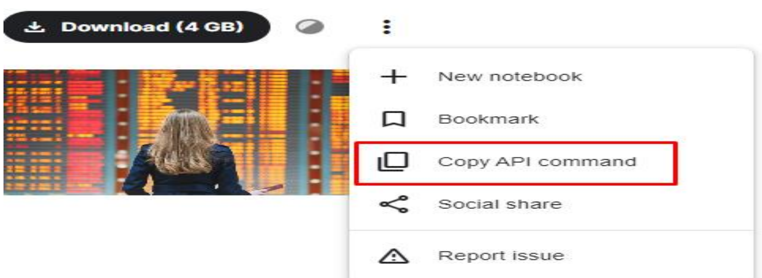


(Figure 2. Kaggle API Token)

```
[ec2-user@ip-10-0-9-65 ~]$ ls /home/ec2-user/.kaggle
kaggle.json
[ec2-user@ip-10-0-9-65 ~]$
```

(Figure 3. Kaggle.json)

Next step is to run the API command. The dataset page on kaggle can generate an API command to run in terminal. Click on "copy API command."



(Figure 4. Copy API command)

It should generate the following API command:

```
kaggle datasets download -d robikscube/flight-delay-dataset-20182022
```

Run this command. Then download and unzip the data.

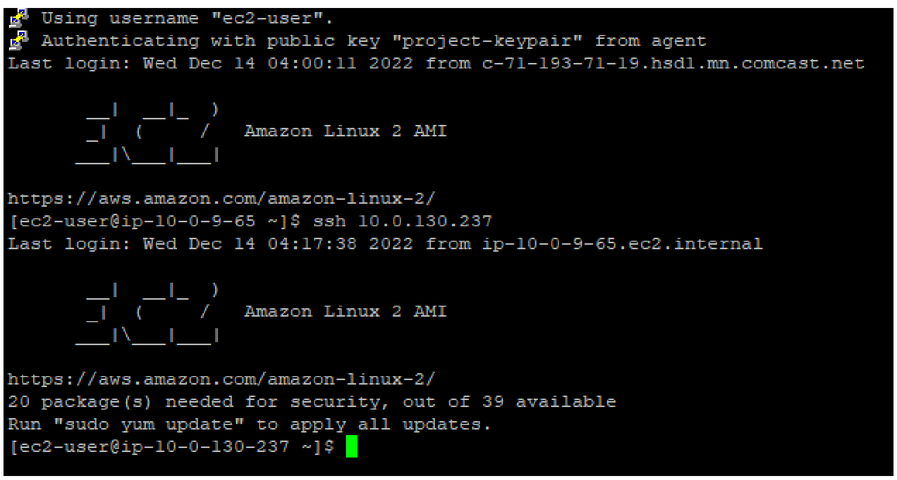
```
[ec2-user@ip-10-0-9-65 ~]$ ls data
Airlines.csv               Combined_Flights_2018.parquet  Combined_Flights_2019.parquet  Combined_Flights_2020.parquet  Combined_Flights_2021.parquet  Combined_Flights_2022.parquet      raw          readme.md
Combined_Flights_2018.csv  Combined_Flights_2019.csv      Combined_Flights_2020.csv      Combined_Flights_2021.csv      Combined_Flights_2022.csv      flight-delay-dataset-20182022.zip  readme.html
[ec2-user@ip-10-0-9-65 ~]$
```

(Figure 5. Downloaded data)

2-d. copy the dataset to Backup Storage instance:

The Backup Storage is in the private subnet and does not have a public ip or DNS Host Name assigned. To login to Backup Storage instance, use agent forwarding through pagent.exe. Add the keypair to pagent.exe and allow agent forwarding in putty.exe (SSH -> AUTH -> allow agent forwarding) to login. Once enabled, login to Data Storage instance first using putty.exe, then SSH into the Backup Storage unit.

```
ssh <backup_storage_private_ip>
```



(Figure 6. Login to Backup Storage)

2-e. use secure copy (scp) to backup data in Backup Storage instance:

Copy the entire dataset directory from Data Storage to Backup Storage.

```
scp -r /home/ec2-user/<dataset_dir> ec2-
user@<backup_storage_private_ip>:/home/ec2-user
```

(No need to specify the key file since already authenticated through agent forwarding and both instances use the same keypair)

A successful transfers should look like the following:

(Figure 7. Successfully transferred data files)

2-f: Install Spark and launch Pyspark on Jupyter Notebook:

First, install Java, scala and Spark and Hadoop binary.

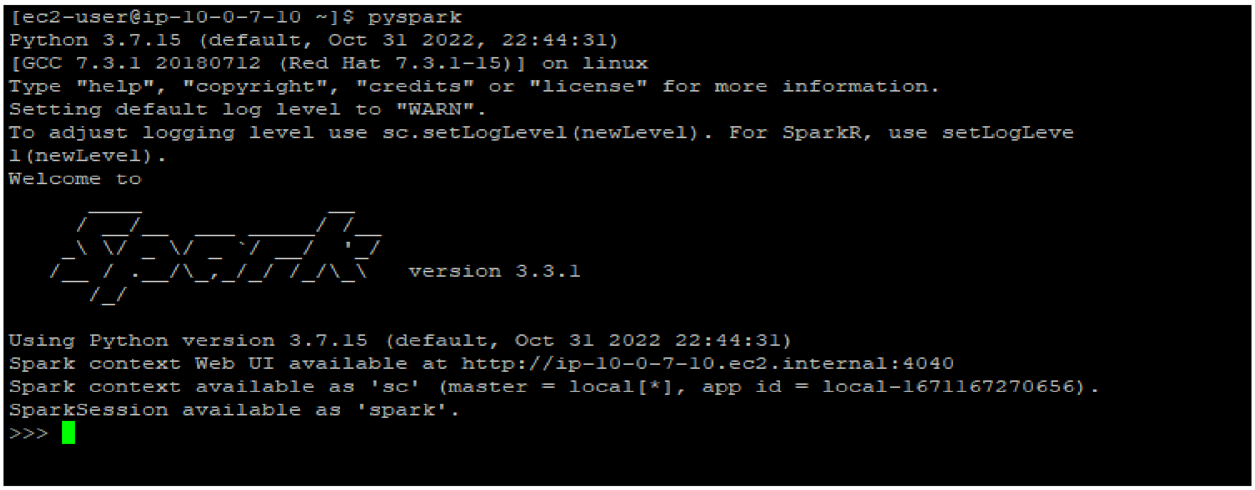
```
sudo amazon-linux-extras enable corretto8
sudo yum install java-1.8.0-amazon-corretto java-1.8.0-amazon-corretto-devel
wget https://downloads.lightbend.com/scala/2.12.2/scala-2.12.2.tgz
wget https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.3.1-bin-hadoop2.tgz
wget https://dlcdn.apache.org/hadoop/common/hadoop-2.10.2/hadoop-2.10.2.tar.gz
```

Unzip the Spark and Hadoop binary tar files. We will need to add some environment variables to `.bashrc` file. These variables will allow us to run Spark commands in terminal. `.bash` file is located in `/home/ec2-user` directory. Open the file and add these lines:

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-amazon-corretto.x86_64
export SPARK_HOME=/home/ec2-user/spark-3.3.1-bin-hadoop2
export HADOOP_HOME=/home/ec2-user/hadoop-2.10.2
```

```
export LD_LIBRARY_PATH=$HADOOP_HOME/lib/native
export SCALA_HOME=/home/ec2-user/scala-2.12.2
export
PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin:$JAVA_HOME/bin:$JAVA_HOME:$JAVA_HOME/jre/bin:$SPARK_HOME:$SCALA_HOME/bin
```

Now type `source .bashrc` on terminal. Then type `pyspark`. This confirms that Spark is installed and configured correctly.



(Figure 8. Pyspark on terminal)

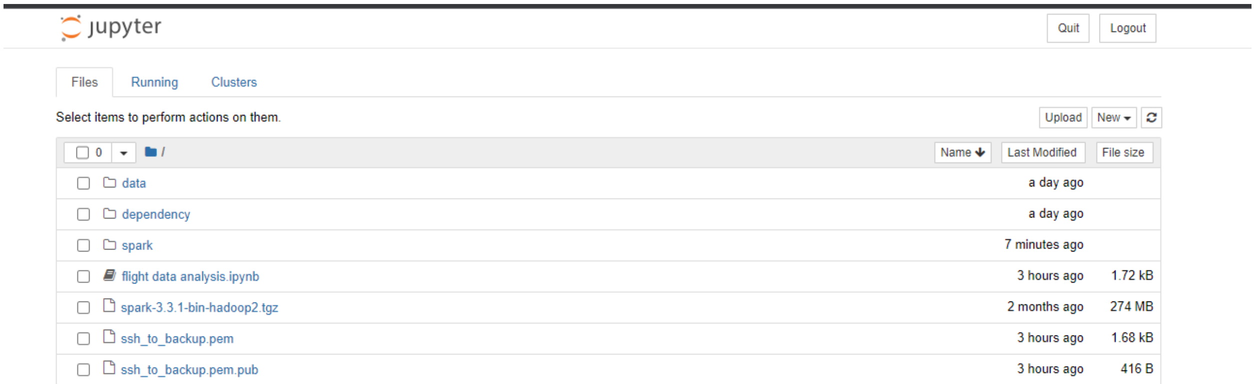
2-g: Open jupyter notebook in browser.

The last step is setting up Jupyter Notebook in Data Storage instance. Create a configuration file:

```
jupyter notebook --generate-config
```

The configuration file location is `/home/ec2-user/.jupyter/jupyter_notebook_config.py`.

Add this line to the file `c.NotebookApp.ip = '0.0.0.0'.` (the default is "localhost" but the EC2 instance does not have a browser to open the notebook. We will need our local machine to open it in browser.) Open your browser and visit `<EC2_instance_public_ip>:8888`. The login page will be prompted. You may create a password after initial login using the token generated on the terminal.



(Figure 9. Jupyter Notebook web UI)

3. Data preparation

For the analysis, I was interested in constructing a predictive linear regression model. I examined the dataset and there were more than 60 columns. I identified a subset of useful variables to include in the analysis.

Variable name	Data Type	Definition
FlightDate	timestamp	Date of flight (yyyy-mm-dd)
Airline	string	Name of the airline
Tail_Number	string	Unique Identifier for each plane flown
Flight_Number_Marketing_Airline	long	Unique identifier for each flight
Origin	string	Origin airport code
Dest	string	Destination airport code
Cancelled	boolean	True if flight is cancelled, False otherwise
Diverted	boolean	True if flight is diverted (change of destination), False otherwise
CRSDepTime	Long	Scheduled departure time (4 digits, hhmm)
Deptime	double	Actual departure time (4 digits, hhmm)
DepDelayMinutes	Double	Difference between scheduled and actual departure time. Early departures are set to 0
CRSArrTime	long	Scheduled arrival time (4 digits, hhmm)
ArrTime	Double	Actual arrival time (4 digits, hhmm)
ArrDelayMinutes	double	Difference between scheduled and actual arrival time. Early arrivals are set to 0
AirTime	Double	Flight time in minutes
Distance	double	Distance between origin and destination airports

In addition, I created two more variables:

- "on_time": a Boolean variable that describes if a flight is on time or not. If the departure and arrival delay times are 0, then the flight is on time. Otherwise, the value will be false.
- "delay_group": a string variable that categorizes the level of delay. Conditions are as follows—departure delay = 0 is considered "On time", departure delay up to 15 minutes is "Small Delay," departure delay between 15 to 45 minutes is considered "Medium Delay," and departure delay

greater than 45 minutes falls within the “Large Delay” category. Otherwise, the value will be “Cancelled.”

One last modification to the data was changing data types. Some variables were of type “long” and others were of type “double.” I wanted to match the variable types for consistency. The variables of long type (“CRSDepTime”, “CRSArrTime”, “Flight_Number_Marketing_Airline”) were cast to double.

```
In [87]: # cast column types long -> double
         cols_long = [
             "Flight_Number_Marketing_Airline",
             "CRSDepTime",
             "CRSArrTime"
         ]

         for c in cols_long:
             df = df.withColumn(c, col(c).cast("double"))

         df.printSchema()

         root
          |-- FlightDate: timestamp (nullable = true)
          |-- Airline: string (nullable = true)
          |-- Tail_Number: string (nullable = true)
          |-- Flight_Number_Marketing_Airline: double (nullable = true)
          |-- Origin: string (nullable = true)
          |-- Dest: string (nullable = true)
          |-- Cancelled: boolean (nullable = true)
          |-- Diverted: boolean (nullable = true)
          |-- DepDelayMinutes: double (nullable = true)
          |-- CRSDepTime: double (nullable = true)
          |-- DepTime: double (nullable = true)
          |-- CRSArrTime: double (nullable = true)
          |-- ArrTime: double (nullable = true)
          |-- ArrDelayMinutes: double (nullable = true)
          |-- AirTime: double (nullable = true)
          |-- Distance: double (nullable = true)
          |-- ArrivalDelayGroups: double (nullable = true)
```

(Figure 10. Column casting to double)

4. Data Analysis

4-a. Exploratory Data Analysis

Before diving into predictive analysis, I wanted to explore the data first. I wanted to see the top 10 performing airlines and the bottom 10 performing airlines. I also wanted to see how many unique planes were used for flight.

- Performance:

The criteria for deciding the flight performance were whether it arrived on time or not. My goal was to calculate the percentage of flights that were on time for each airline. I grouped the data by airline name and counted the airlines. Then I filtered the delay group variable with value “On Time.” The percentages were calculated in this way - number of flights that were on time divided by total number of flights per airline. The top 10 performers were the ones with the greatest number of flights that were on time. The bottom 10 performers had the least number of flights on time.

```
In [14]:
# top 10 performers
df_airline = df.groupBy("Airline").count().withColumnRenamed("count", "total")
df_on_time = df.filter(col("delay_group") == "On Time").groupBy("Airline").count().withColumnRenamed("count", "on_time_count"

df_airline = df_airline \
    .join(df_on_time, "Airline") \
    .withColumn("pct", (col("on_time_count") / col("total")))

df_airline.orderBy(col("pct").desc()).show(10, truncate = False)
```

```
+--------------------------+-------+-------------+------------------+
|Airline                   |total  |on_time_count|pct               |
+--------------------------+-------+-------------+------------------+
|Cape Air                  |1661   |1413         |0.8506923540036123|
|Endeavor Air Inc.         |998224 |830582       |0.8320597380948564|
|Delta Air Lines Inc.      |3294917|2655921      |0.8060661315596114|
|Republic Airlines         |1283704|1025293      |0.7986989212466425|
|SkyWest Airlines Inc.     |3159683|2492446      |0.7888278665929462|
|Capital Cargo International|392011 |307136       |0.7834882184428498|
|Compass Airlines          |154985 |120306       |0.7762428622124722|
|Alaska Airlines Inc.      |906259 |700379       |0.7728243250549788|
|Horizon Air               |471153 |364045       |0.7726683264247495|
|Air Wisconsin Airlines Corp|380202 |293447       |0.7718186648150194|
+--------------------------+-------+-------------+------------------+
only showing top 10 rows
```

(Figure 11. Top 10 Performing Airlines)

```
In [13]: # bottom 10 performers
df_airline.orderBy(col("pct").asc()).show(10, truncate = False)

+-------------------------------------+-------+-------------+------------------+
|Airline                              |total  |on_time_count|pct               |
+-------------------------------------+-------+-------------+------------------+
|Peninsula Airways Inc.               |2783   |1410         |0.5066475026949335|
|Allegiant Air                        |489400 |319988       |0.6538373518594197|
|JetBlue Airways                      |1106079|744112       |0.6727476066356924|
|Southwest Airlines Co.               |5474339|3729079      |0.6811925604168833|
|Commutair Aka Champlain Enterprises, Inc.|260048 |178203       |0.6852696425275334|
|Frontier Airlines Inc.               |570452 |393388       |0.6896075392846375|
|Empire Airlines Inc.                 |23122  |15978        |0.691030187700026 |
|Trans States Airlines                |161590 |112895       |0.6986509066155084|
|American Airlines Inc.               |3134117|2298008      |0.7332234246519833|
|ExpressJet Airlines Inc.             |353669 |261873       |0.7404465757530346|
+-------------------------------------+-------+-------------+------------------+
only showing top 10 rows
```

(Figure 12. Bottom 10 Performing Airlines)

- Unique Plane ID's:

I was also interested in checking how many different flights were in the data set. Every plane has a unique identifier assigned and the number is displayed on the tail of the plane. The number is called tail number. I aggregated the tail numbers into groups by tail numbers and counted the number of rows. There were 7089 unique planes used in this dataset.

```
In [27]: # unique plane ID
df_plane = df.filter(~isnull(col("Tail_Number"))).groupBy("Tail_Number").count()
df_plane.orderBy(col("count").desc()).show(10)
df_plane.distinct().count()

+-----------+-----+
|Tail_Number|count|
+-----------+-----+
|     N480HA|12470|
|     N491HA|12376|
|     N494HA|12248|
|     N492HA|11985|
|     N487HA|11945|
|     N483HA|11935|
|     N493HA|11837|
|     N486HA|11797|
|     N476HA|11771|
|     N488HA|11727|
+-----------+-----+
only showing top 10 rows

Out[27]: 7089
```

(Figure 13. Unique planes)

4-b. Linear regression

Pyspark has a machine learning package with linear regression models. I attempted to build regression models to predict flight delay time. One model was to predict departure delay and the second model was to predict arrival delay. I chose departure delay (DepDelayMinutes) and arrival delay (ArrDelayMinutes) to be the response variables. The predictor variables to feed to the models were scheduled departure/arrival times (CRSDepTime and CRSArrTime), actual departure/arrival times (DepTime and ArrTime), distance between airports (Distance), cancellation status (Cancelled), whether the destination changed or not (Diverted), and whether the flight was on time or delayed (on_time).

Model 1

$$DepDelayMinutes = \beta_0 + \beta_1 CRSDepTime + \beta_2 DepTime + \beta_3 Distance + \beta_4 Cancelled + \beta_5 Diverted + \beta_6 on\_time$$

Model 2

$$ArrDelayMinutes = \beta_0 + \beta_1 CRSArrTime + \beta_2 ArrTime + \beta_3 Distance + \beta_4 Cancelled + \beta_5 Diverted + \beta_6 on\_time$$

β_0 is the intercept and β_1 through β_5 are regression coefficients. The linear regression algorithm will estimate the intercept and the coefficients to build the models. (See Appendix B for the source code)

The data needed some transformation before building the models. I split the data set into training and test data set. Then I fitted each model on the training data and generated some predicted values for departure and arrival delay from the train data. I used the test data to compare the actual values and predictions. Below are the coefficients and results from the regression models.

```
In [180]:
# regression coefficients
d = [mod_d.intercept]
coeffs_d = mod_d.coefficients.toArray().tolist()

for elem in coeffs_d:
    d.append(elem)

vars_d = variables_d
vars_d[0] = "intercept"
vars_d

data_d = list(map(lambda x, y: (x, y), vars_d, d))
cols_d = ["variable", "coeffcient"]

coef_df_d = spark.createDataFrame(data_d, cols_d)
coef_df_d.show(truncate = False)

+----------+----------------------+
|variable  |coeffcient            |
+----------+----------------------+
|intercept |1.8615466531080166E-14 |
|CRSDepTime|0.9999999999999998    |
|DepTime   |1.3731461534976354E-16 |
|Distance  |-1.3500237264990898E-16|
|Cancelled |-2.5297577974580323E-18|
|Diverted  |3.042259788571149E-15  |
|on_time   |2.5038072472242403E-15 |
+----------+----------------------+
```

(Figure 14. Departure delay coefficients)

```
In [114]:
# generate predictions
pred_results_d = mod_d.transform(test_data_d)
pred_results_d.select("DepDelayMinutes", "PredDepDelay").show(truncate = False)

+---------------+----------------------+
|DepDelayMinutes|PredDepDelay          |
+---------------+----------------------+
|0.0            |-4.2285136305267026E-15|
|0.0            |-3.2422036208527774E-13|
|0.0            |-3.2462536920322753E-13|
|0.0            |-3.2476037157587736E-13|
|0.0            |-3.248953739485273E-13 |
|0.0            |-3.248953739485273E-13 |
|0.0            |-3.2396534828844737E-13|
|0.0            |-3.250303763211772E-13 |
|0.0            |-3.259486784016545E-13 |
|0.0            |-3.2330094719710055E-13|
|0.0            |-3.2261630804337405E-13|
|0.0            |-3.2423535303374713E-13|
|0.0            |-3.2621868314695425E-13|
|0.0            |-3.254353834391269E-13 |
|0.0            |-3.3016348798038405E-13|
|0.0            |-3.3188878279825044E-13|
|0.0            |-3.3188878279825044E-13|
|0.0            |-5.7923389795509756E-15|
|0.0            |-3.2443144635169016E-13|
|0.0            |-3.2372292657008663E-13|
+---------------+----------------------+
only showing top 20 rows
```

(Figure 15. Departure delay predictions)

```
In [182]:
# regression coefficients arrival
a = [mod_a.intercept]
coeffs_a = mod_a.coefficients.toArray().tolist()

for elem in coeffs_a:
    a.append(elem)

data_a = list(map(lambda x, y: (x, y), vars_d, a))

coef_df_a = spark.createDataFrame(data_a, cols_d)
coef_df_a.show(truncate = False)

+----------+---------------------+
|variable  |coeffcient           |
+----------+---------------------+
|intercept |7.259549593647617E-10 |
|CRSDepTime|0.9999999999759857   |
|DepTime   |-5.553436632540114E-11|
|Distance  |5.122693075695906E-11 |
|Cancelled |1.833084157464325E-12 |
|Diverted  |0.0                  |
|on_time   |0.0                  |
+----------+---------------------+
```

(Figure 16. Arrival delay coefficients)

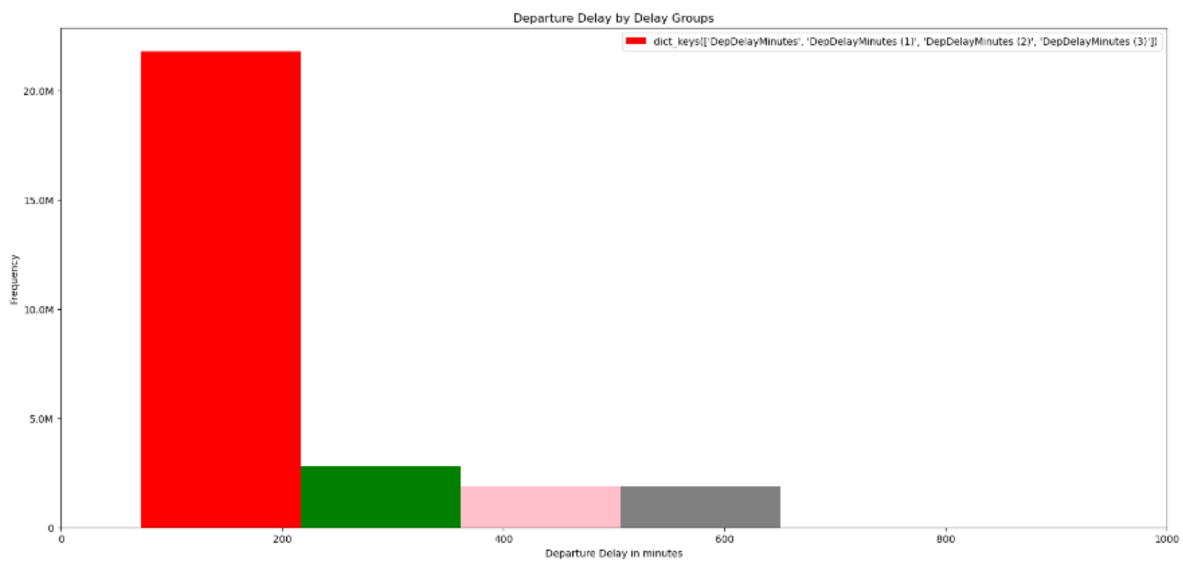
```
In [185]:  #print predicted results arrival
           pred_results_a = mod_a.transform(test_data_a)
           pred_results_a.select("ArrDelayMinutes", "predArrDelay").show(truncate = False)

+---------------+---------------------+
|ArrDelayMinutes|predArrDelay         |
+---------------+---------------------+
|0.0            |8.351832463592782E-9 |
|0.0            |8.560804057543716E-9 |
|0.0            |8.725781631715504E-9 |
|0.0            |8.725781631715504E-9 |
|0.0            |8.725781631715504E-9 |
|0.0            |8.881593785099972E-9 |
|0.0            |9.167554913664407E-9 |
|0.0            |9.229879775018195E-9 |
|0.0            |9.404022769977304E-9 |
|0.0            |9.724812497533562E-9 |
|0.0            |9.8256321261941E-9   |
|0.0            |9.8256321261941E-9   |
|0.0            |9.96128035384646E-9  |
|0.0            |9.96128035384646E-9  |
|0.0            |1.2974506226836798E-7|
|0.0            |1.3139097299046685E-7|
|0.0            |1.3139097299046685E-7|
|0.0            |1.3144219992122382E-7|
|0.0            |1.314934268519808E-7 |
|0.0            |1.293321212046783E-7 |
+---------------+---------------------+
only showing top 20 rows
```

(Figure 17. Arrival delay predictions)

5. Visualization

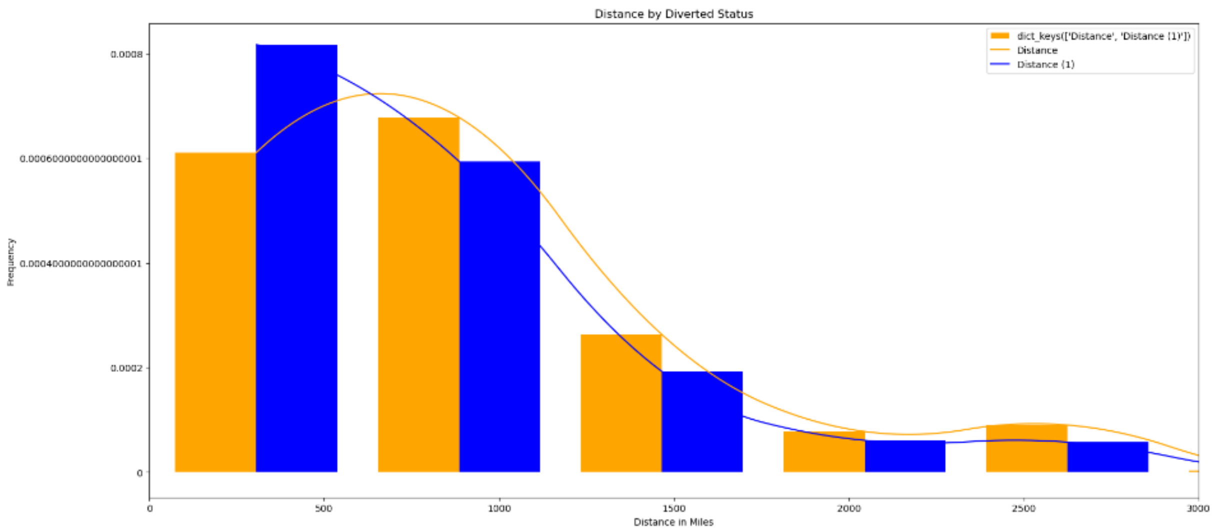
In the last part of the project, I attempted to create some visualization for the data. I was interested in examining the distributions of departure delay time and distance between airports based on different categories. I created a histogram based on delay groups and a distribution plot based on diverted status.

5-a. histogram of departure delay based on delay groups



The red represents on time flights, the green represents small delay group, the pink represents medium delay group, and the grey represents large delay group. Majority of the flights seems to have arrived on time and large delay group and medium delay group seems to be equal numbers.

5-b. distribution plot distance of base on diverted status



The orange represents diverted flights and the blue represents non-diverted flights. Overall diverted flights seems to have traveled longer distance, except for the first distance group (0 – 500 miles).

6. Reflection

Overall, the project was successful, and I learned a lot about Spark and its applications in data analysis. However, there were a few challenges I wanted to point out.

6.a The data collection process could have been automated using Hadoop Ecosystem.

I downloaded the dataset manually from Kaggle. Instead, if I utilized tools like NiFi or even Spark Streaming, I could have automated the data collection process.

6-b. The EC2 instances could have been separated.

I used one EC2 instance for backup data storage and another for both data analysis AND data storage. If I separated the data analysis instance into two instances (data analysis instance and data storage instance), I could have used the resources more efficiently. However, with my current setup, I was not sure how to use one instance to store data and import the data into another EC2 instance for analysis. For convenience, I could have used AWS EMR clusters instead as well.

6-c. EC2 instance ran out of memory.

The first time I noticed this incident was when I was trying to create some visualizations. Instead of using pyspark_dist_explore package, I originally intend to use matplotlib package since it had more visualization options. However, the data frames generated using Pyspark were not compatible with matplotlib. So I had to convert the data frames into Pandas data frame. I applied toPandas() method to the data frames but since the size of the data was too huge, it threw out-of-memory errors. My first EC2 instance was t2.micro class. After realizing the issue I had to upgrade the instance to t2.xLarge, but the issue persisted. My next approach was to import data directly into Pandas data frames. However, the import process did not even execute because the file was too huge for EC2 instance to process. I tried using my local machine for importing data and it now took about 10-15 minutes just to load the data. Eventually I had to revert to using Pyspark data frames to create visualizations. What I learned from this experience, though, was that Spark is a very powerful tool especially when working with Big Data.

(link to errors that I was having: https://stackoverflow.com/questions/26892389/org-apache-spark-sparkexception-job-aborted-due-to-stage-failure-task-from-app)

7. References

Flight Delay EDA Kaggle https://www.kaggle.com/code/ahmedeltom/flights-delay-analysis-eda

Data source: https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022?resource=download

Kaggle API: https://github.com/Kaggle/kaggle-api

AWS documentation: https://docs.aws.amazon.com/

SCP between EC2 instances https://blog.e-zest.com/how-to-do-scp-from-one-ec2-instance-to-another-ec2-instance/

Set up Jupyter Notebook https://medium.com/analytics-vidhya/set-up-jupyter-notebook-on-aws-ec2-instance-1a87d1707467

Spark By Examples Tutorial https://sparkbyexamples.com/pyspark-tutorial/

Setting environment variables on EC2 https://bhargavamin.com/how-to-do/setting-up-java-environment-variable-on-ec2/

Install Spark on EC2 https://sparkour.urizone.net/recipes/installing-ec2/

Pyspark documentation https://spark.apache.org/docs/3.1.1/api/python/reference/pyspark.sql.html

Linear Regression with Pyspark https://www.kaggle.com/code/fatmakursun/pyspark-ml-tutorial-for-beginners

Pyspark visualization documentation https://github.com/Bergvca/pyspark_dist_explore

Matplotlib documentation https://matplotlib.org/stable/index.html

8. Appendix

A. Data dictionary: https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGK

B. Source code: https://github.com/cmansoo/big-data-engineering-project