Lister Kom

U19379791

MIT805 Assignment 2

October 30, 2022

Google Drive:

https://drive.google.com/drive/folders/12dRtdZTtXgNeLxAZQHhlUqTS0PMtqxwl?usp=share_link

GitHub Link:

https://github.com/listerkom/MIT.git

1. Dataset

The dataset was sourced from Bureau of Transportation Statistics website [1]. The information includes arrival and departure times and flight details for every commercial flight operating within the United States, from October 1995 to 2022. We limited our analysis to the last 5 years due to the large quantity (**volume**) of the dataset i.e. Jan 2018 – June 2022, with the total size of the csv files adding up to 3.46G. The dataset was delivered in the form of 54 zipped csv files organized by month and year. It took about 1 hour and 45 minutes to download the zipped files individually for each month, as there is no date range selector facility on the website. The files were unzipped and placed in the local drive and Google drive. I used Python to extract and merge the files in Jupyterlab. After merging the csv files, the resulting dataset contained about 28 million records, and takes up to 12.4G+ memory in Python. Every year, approximately 7 million flights are recorded from 28 carriers and 388 departure airports. The dataset also came with carrier names csv file, which we joined to the main dataset, using the carrier code. Every flight that occurred during that time period is represented by a row in the dataset, and each column has detailed information on every flight, including the airline, flight date, departure delay, and arrival delay, etc. The time it took to load the dataset into Python on a machine with 16GB of RAM was 5 seconds. It took another 2 minutes to concatenate the data. The concatenating process was done in steps, by using a for loop, rather than merging all files at once, which would have obliterated the memory.

Data like this is crucial in the aviation business, especially when it comes to understanding the causes of flight delays. Flight delays are now universally understood to result in monetary losses for the aviation industry. Over 20% of US flights were delayed in 2018, according to data from the United States Bureau of Transportation Statistics (BTS), resulting in a significant economic effect of around 41 billion US$. Airlines and their passengers are both inconvenienced by these delays. This can ruin airlines' reputations and reduce passenger demand [2]. Such data can be useful, not only to passengers but also airlines, for planning purposes. Consequently, losses can be minimised or mitigated.

2. Main feature(s) of focus

In this project I will focus on arrival delays, their frequency of occurrence and duration. I'm planning to look into the following relationships (1) airlines vs delays and (2) departure airports vs delays. An indication whether a delay has occurred or not will be based on values (minutes) being greater than 0 or not. The most interesting information the company might want to extract is the **delay data** per airline and per airport. This will allow them to be able to compare themselves

against other airlines and ensure their delay stats are better than competitors. Likewise, by knowing which routes usually lead to delays, and during which times/days/months these are most often, they can plan accordingly or take proactive actions to counter possible delays.

3. Why and how PySpark was used?

PySpark from Apache Spark was used for this assignment, through the use of PySpark dataframes, previously known as schemaRDDs. Query execution in Python might be much slower in previous Spark APIs (i.e. RDDs) owing to communication complexity between the Java JVM and Py4J. The utilisation of distributed, in-memory data structures in this virtual arrangement considerably improves performance for diverse data processing tasks. It is a centralised analytics engine that allows for the distribution and parallelization of large data processing operations. Spark divides data to allow for parallel execution. I used PySpark DataFrames since they do not require manual partition modification. All DataFrame operations are parallelized and dispersed among clusters automatically. One of the finest aspects of the new DataFrames API is the cutting-edge optimisations and code creation provided by the Spark SQL Catalyst optimiser.

I used '*spark.read'* interface to directly load data sources into Spark data frames, as shown in Fig 1. SparkSession is used to import data into DataFrames. The transformations (parallelization, mapping, and partitioning) that turn the RDD into a dataframe are contained within the DataFrame operation, spark.read.csv (what I used). Similar to RDD transformations, DataFrame transformations and actions are less lazy than RDD operations, largely due to the Catalyst Optimizer. A list of key/value pairs is first sent to the row class in order to create row instances. Then, using sampling of the data, Spark SQL transforms this RDD of row objects into a DataFrame, where the keys are the columns. [3] DataFrame functionality is efficient, and hence I decided to use it instead of Hadoop MapReduce, as the machine used wasn't capable enough.

Fig 1

```
fd = spark.read.format("csv")\
.option("inferSchema", "true")\
.option("header", "true")\
.load("C:/Users/User/OneDrive/Desktop/Toshiba/Studies/UP/2022 S2/MIT805/Assignment/Assignment 2/Cleaned_Airline_Data.csv")
fd.cache() #we cache the flight dataset so subsequent queries will be faster => store in memory
fd.createOrReplaceTempView("fdTable")
```

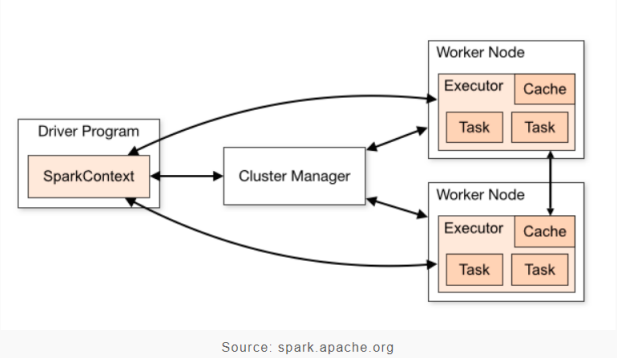
Secondly, Spark's key contribution was the introduction of an in-memory caching layer. Users can tell Spark to cache input data sets in memory so that they do not have to be read from disk for each action. This was also implemented as can be seen in the code in Fig 1 above. If one wants

to utilise PySpark for distributed computation, one must use Spark data frames rather than traditional Python data types.

3.1. SparkContext

SparkContext is used to load the settings from system properties. It is used to generate Spark RDD, accumulators, and broadcast variables programmatically on the cluster. This is illustrated in Fig 2. The Spark driver application establishes and utilises SparkContext to establish a connection with the cluster manager in order to submit Spark tasks and to determine which resource manager (YARN) to talk with. It is the central component of the Spark application. [4]

Figure 2



Spark Distribution Process consists of the following:

(1) Master or Driver: The application's heart, which runs the *main()* function and figures out what code and executors are needed to finish the job. It then gives the work to the slaves (executors) in terms of what they can do.

(2) Task Cluster Manager: A cluster, or group of machines, brings together the resources of several machines so that they can be used as if they were one. It gives the Master or Driver the executor resources they need.

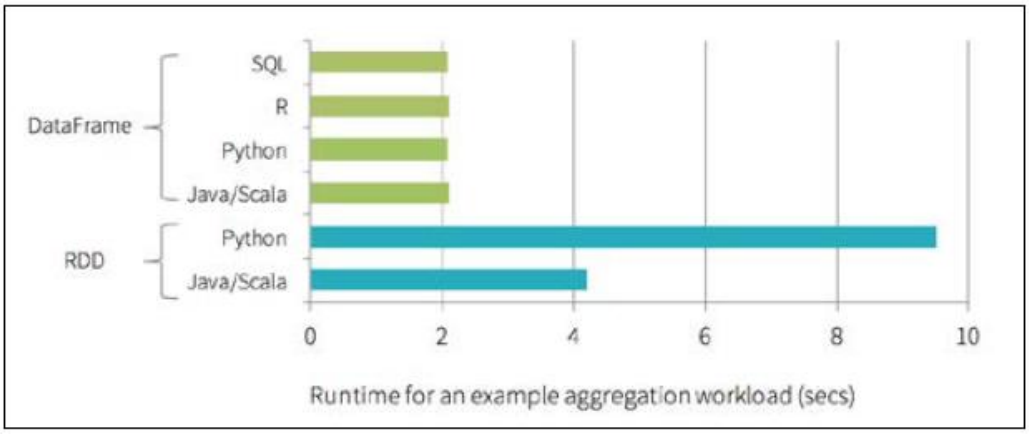
(3) Executor or Slave: The job of the executors is to do the work that the driver gives them and tell the driver how the computation is going on that executor.

3.2. Catalyst Optimiser

The Catalyst Optimizer (CO) is a big reason why the Spark SQL engine can run so quickly. This is important because the Spark engine's CO doesn't immediately process the query. Instead, it

builds and optimises a logical plan to find the most efficient physical plan. DataFrames and the Catalyst Optimizer are important because they make PySpark queries run faster than RDD queries that aren't optimised. Fig 3 [4] compares how long it takes for group-by-aggregation to run on 10 million integer pairs on a single machine. Clearly, using DataFrame than RDD is better, as illustrated.

Figure 3



3.3. Example Outputs using DataFrame

To demonstrate the usage of PySpark DataFrame, using SQL, as possibly being the most interesting feature, I selected the 'DELAY' variable from the dataset:

(1) To obtain the overall percentage of delayed flights, the query and results are shown in Fig 4. Here, we observe that overall, 33% of all flights are delayed. 'Delay' is defined as the arrival delay time > 0 (minutes).

Fig 4

```
spark.sql(
    """
    SELECT DELAY,
           COUNT(*) AS count,
           COUNT(*) / (SELECT COUNT(*) FROM fdTable WHERE DELAY <> '') AS ratio

    FROM fdTable
    GROUP BY DELAY HAVING DELAY <> ''
    """
).show()
```

```
+-----+--------+-------------------+
|DELAY|   count|              ratio|
+-----+--------+-------------------+
|    0|18296443| 0.6732472390576975|
|    1| 8879967|0.32675276094230254|
+-----+--------+-------------------+
```

(2) To obtain the distinct number of departure airports and airlines, the query and results are shown in Fig 5. Here, we observe that overall, we have 388 and 28 airports and airlines respectively.

Fig 5

```
spark.sql(
    """
    SELECT COUNT(distinct(ORIGIN_AIRPORT_ID)) AIRPORTS,
           COUNT(distinct(OP_UNIQUE_CARRIER)) AS AIRLINES
    FROM fdTable
    """
).show()
```

```
+--------+--------+
|AIRPORTS|AIRLINES|
+--------+--------+
|     388|      28|
+--------+--------+
```

3.4. Advantages of Spark over Hadoop

In terms of data processing speed, Spark is 10 to 100 times quicker than Hadoop MapReduce. It is also more efficient since it has several tools for doing sophisticated analytical processes. Finally, it is simple to connect with current Hadoop infrastructure. It is therefore why I chose to implement Spark.

4. Visualisations

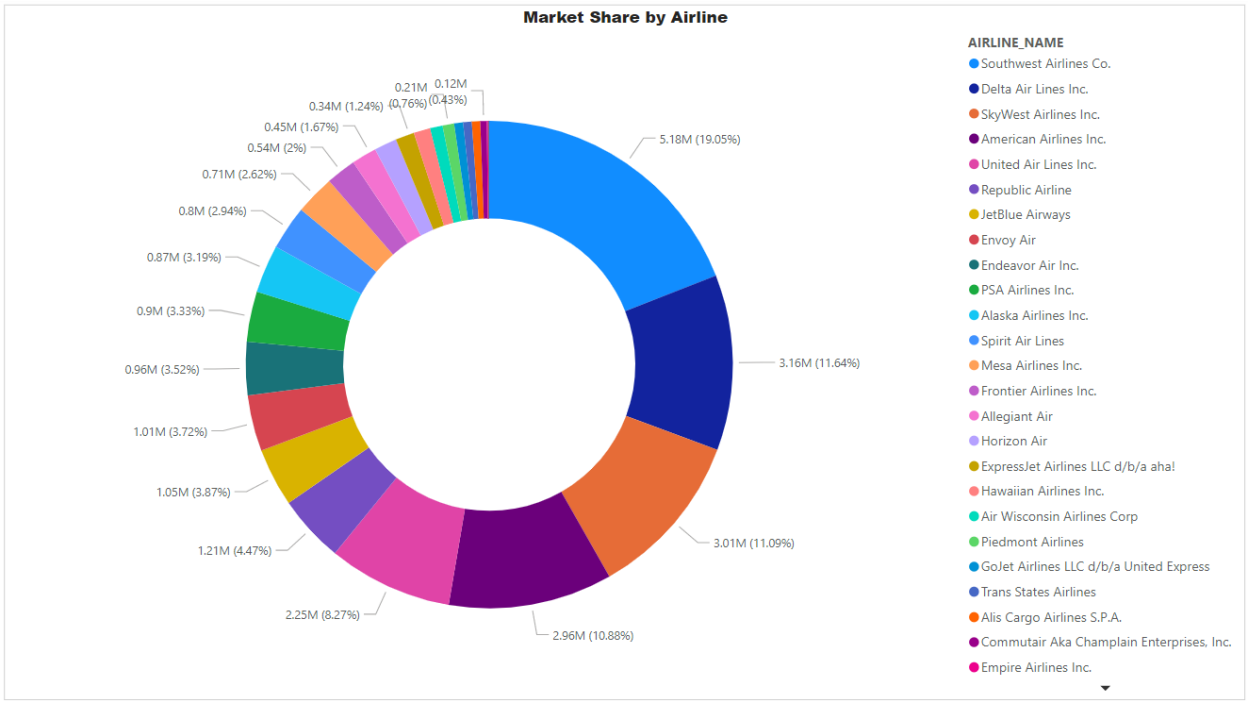
Visualisations were done using PowerBI and the following graphs were performed:

- Market share by airline
- Overall delays and number of flights over time
- Delays per airline
- Delay percentage per airline
- Delay percentage per month and day of the week
- Average departure delay time per airline
- Average arrival delay time per airline
- Market share by airline
- Flight distribution by departure airport
- Delay percentage per departure airport

(1) Market share by airline

Fig 6 depicts the number of flights and the market share (%) per airline respectively. Market share is defined in terms of number of flights as a percentage of total flights. We observe that SouthWest Airlines has the biggest market share (19.1%) and 5.2m flights over the period. It is then followed by Delta Airlines at 3.2m flights and 11.6% market share. Cape Air has the lowest number of flights (0.12m) and market share (0.4%). The top 4 airlines constitute more than 50% of the market share.

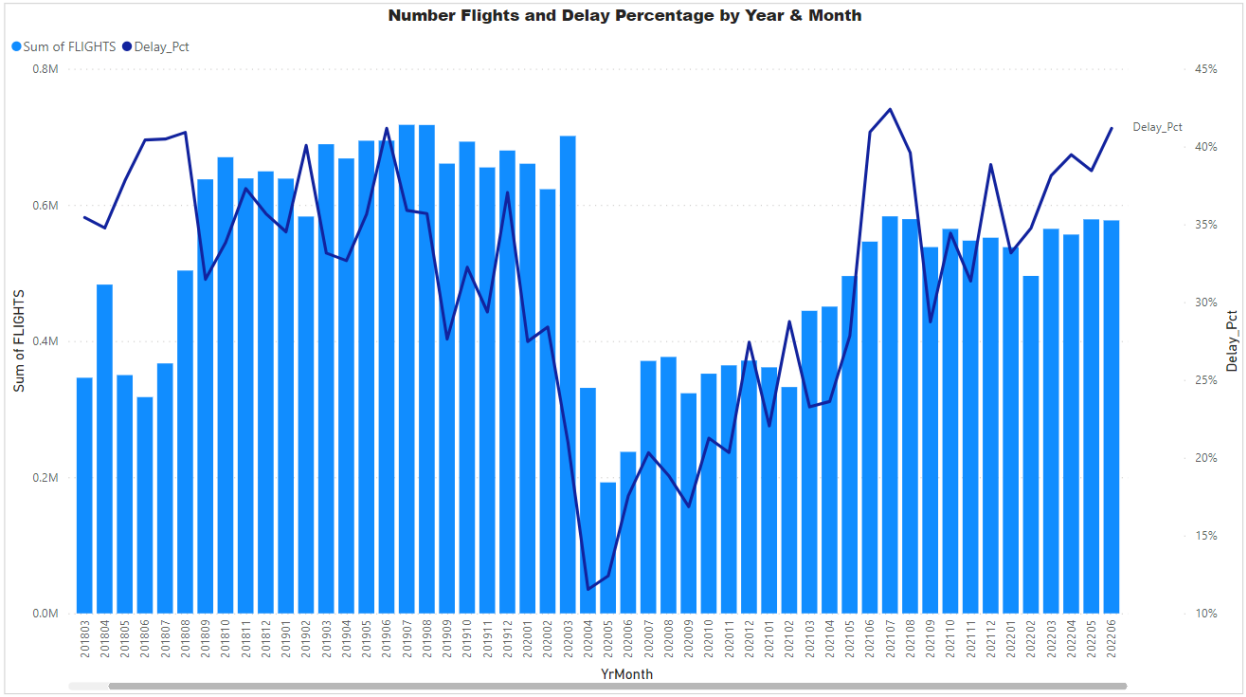
Fig 6



(2) Overall delays and number of flights over time

Fig 7 depicts the number of flights and percentage of delayed flights over time, from 2018. This data would also be quite interesting for airline management to look at. Pre-covid (before April 2020), the average number of flights per month was about 700,000. Post-covid, although they have been rising, they haven't reached the pre-covid levels yet. With regards to the percentage of delayed flights, despite the number of flights being lower post-covid, the highest delay percentage was reached in that period, in July 2021.

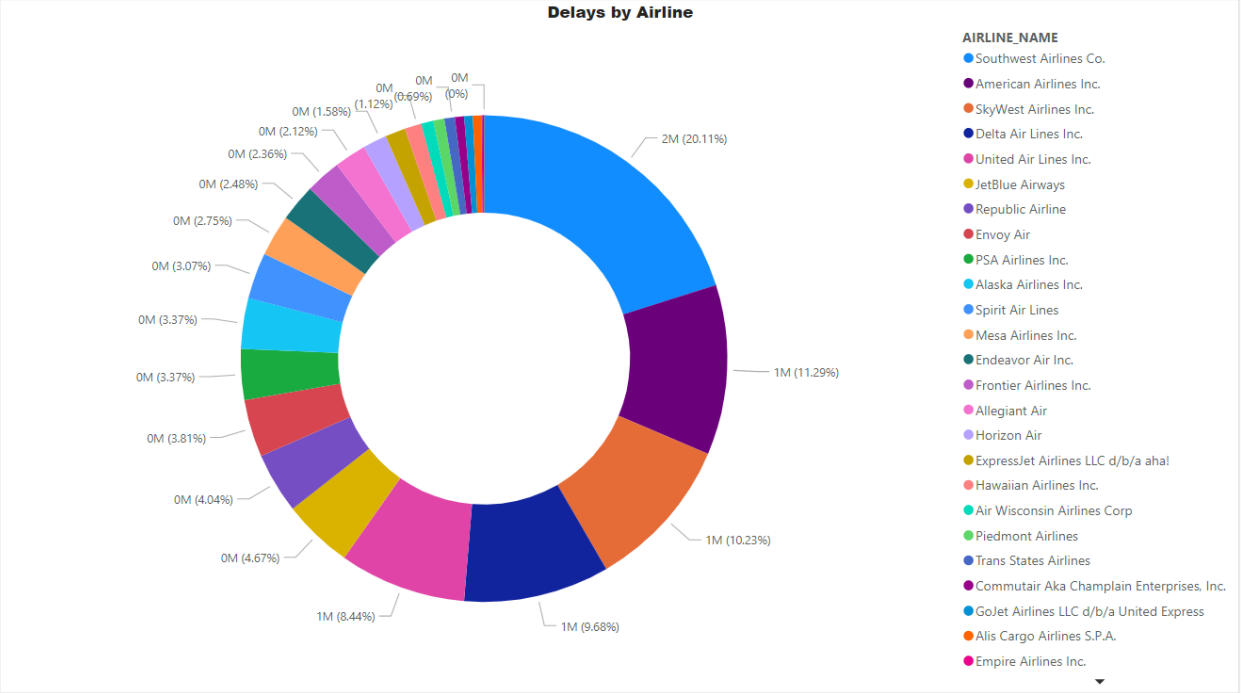
Fig 7



(3) Delays per airline

Fig 8 depicts the number of arrival flights that were delayed per airline over the period. SouthWest airlines has the highest number of delays (2m), which constitutes 20.1% of all the delays. Naturally, it would be expected that being the airline with the highest number of flights, the delays would also follow suit. Delays have been defined as arrival delay minutes which are greater than 0 minutes. This is followed by American Airlines with 1m delayed arrival flights, constituting 11.3% of all delays. Once again, the top 4 airlines constitute more than 50% of the total delays.

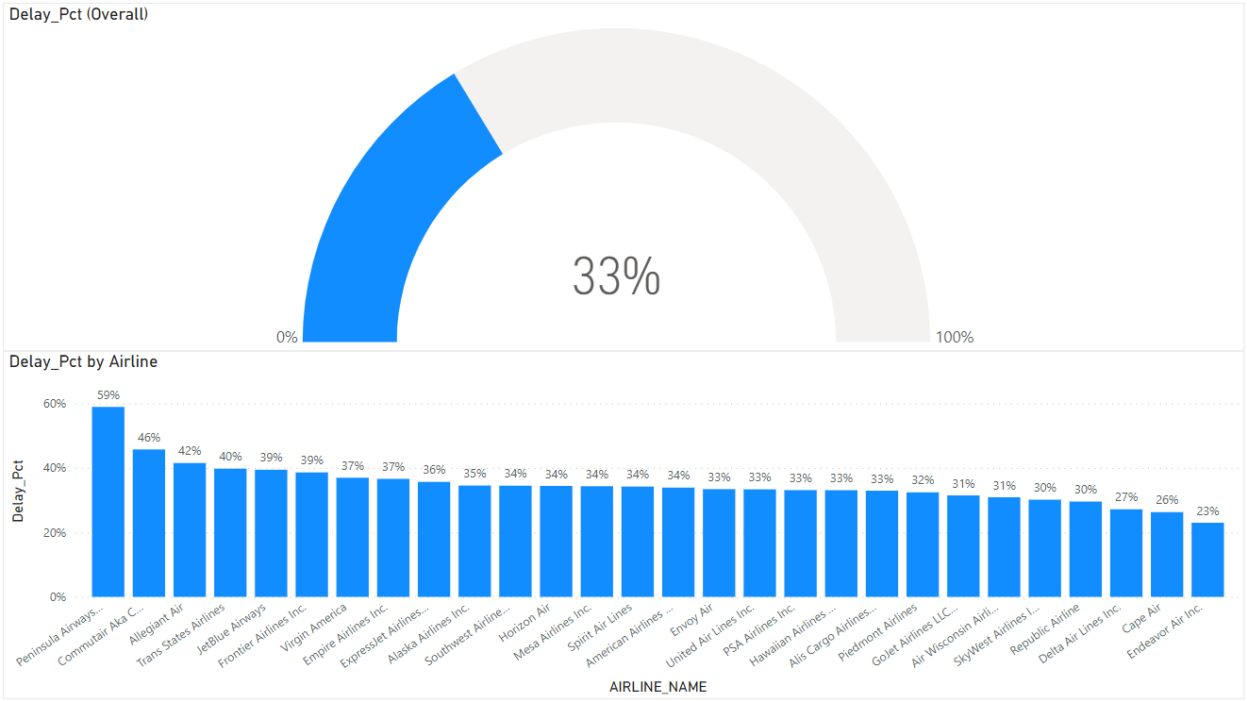
Fig 8



(4) Delay percentage per airline

Fig 9 depicts delay percentage per airline, which is number of delayed arrival flights over the number of flights. It's not enough to just look at the number of delays without relating them to the number of flights. Overall, 33% of all flights are delayed. Per airline, Peninsula Airways has the highest number of delayed flights at 59%. 15 of the 28 (53.5%) of the airlines are above the overall average of 33%.

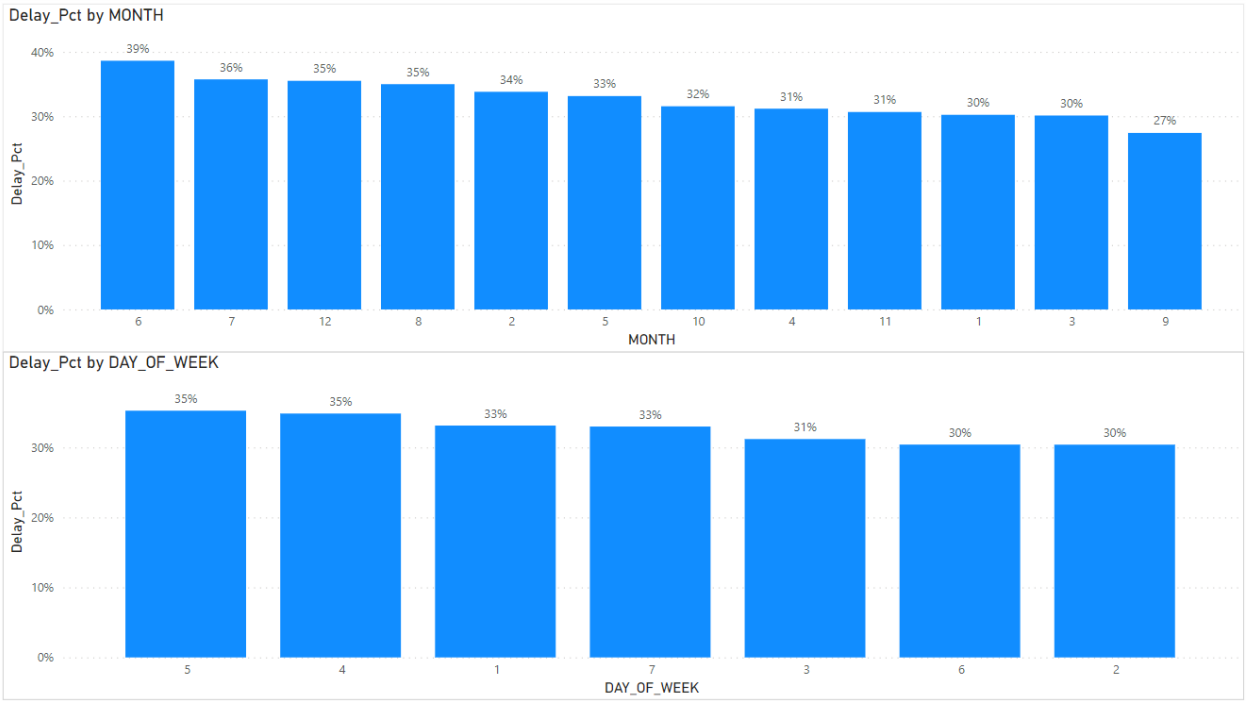
Figure 9



(5) Delay percentage per month and day of the week

Fig 10 depicts delay percentage per month and per day of the week. We observe that the percentage of delayed flights does vary by the month of the year, the range is 27% to 39%. June month has the highest delay percentage and September has the lowest. For June, it is definitely expected, as it's a holiday month. December month is also not far off at 35% delay percentage. With regards to the days of the week, there is some variability, however it is not as not pronounced as by months of the year. Friday (day 5) has the highest percentage (35%) of delays.

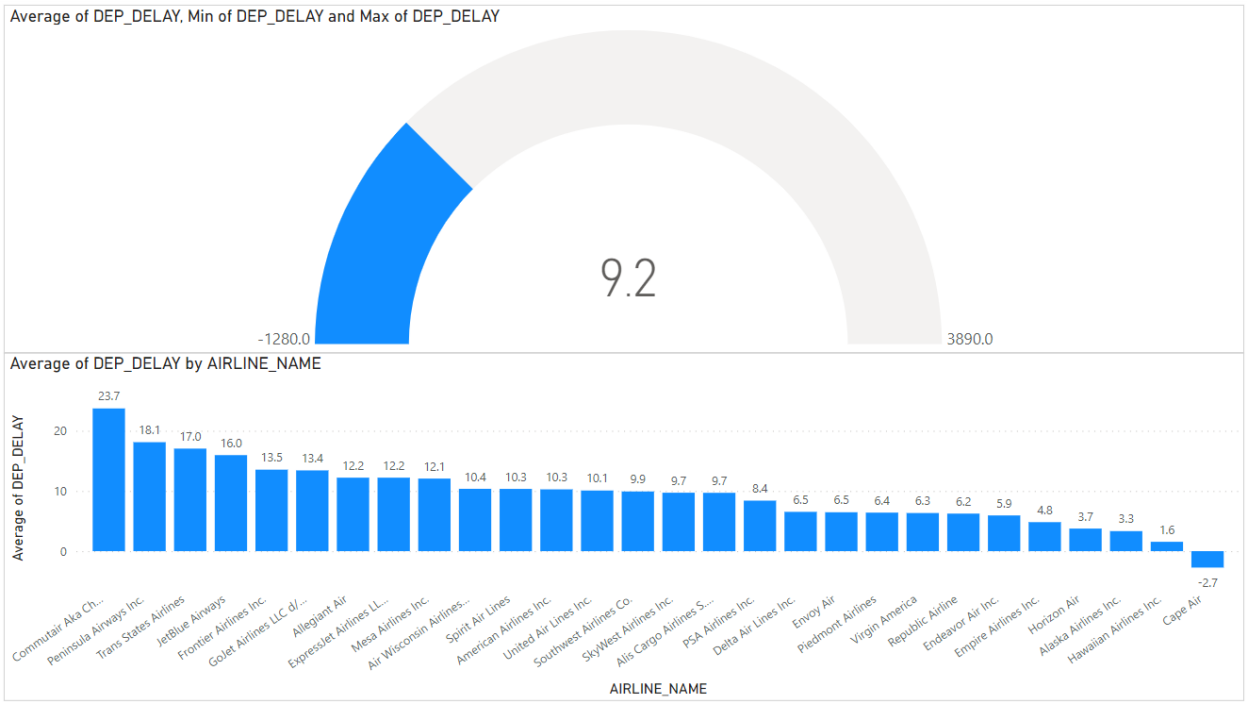
Fig 10



(6) Average departure delay time per airline

Figure 11 depicts average delay (in minutes) per airline for departure delays. The overall average departure delay is 9.2 minutes. Commutair has the highest average departure delay. The two largest airline, Southwest Airlines and Delta Airlines have an average departure delay time of 9.9 and 6.6 minutes respectively. The departure delay times have a large variation, with minimum and maximum being -1280 and 3890 minutes respectively. These are possibly outliers. As airline management, this is an important metric to keep close tabs on, in order to understand the drivers thereof.

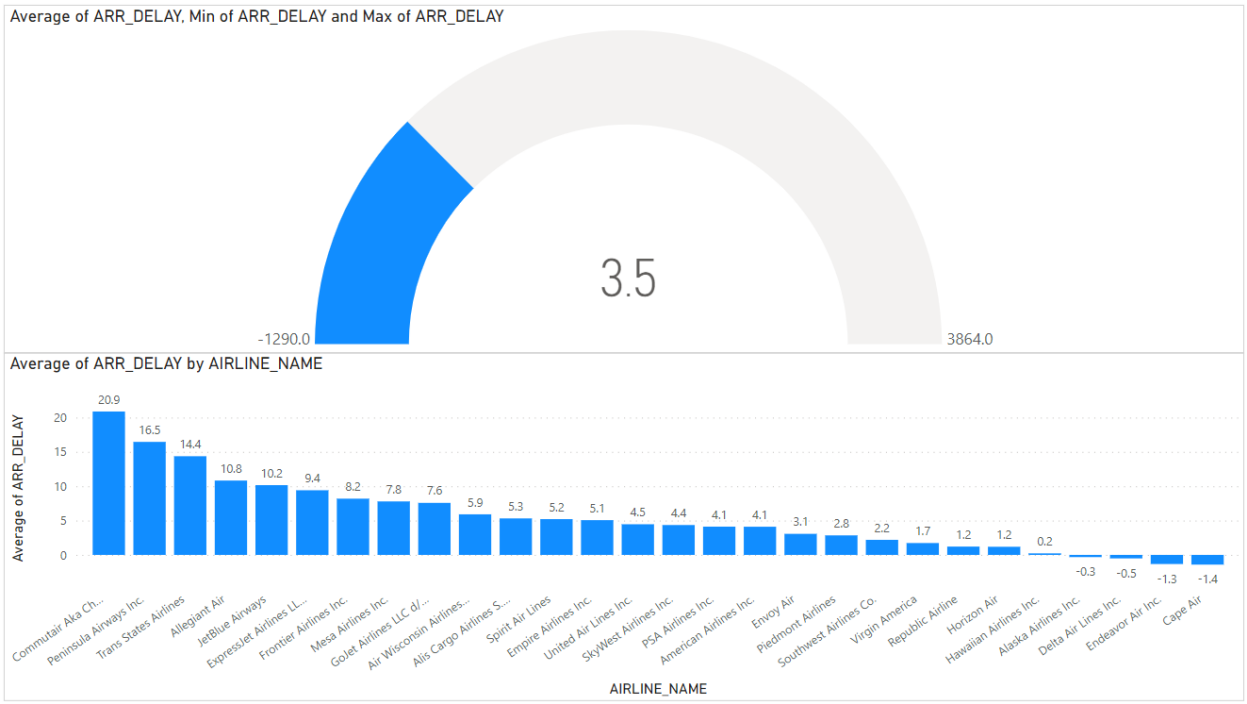
Fig 11



(7) Average arrival delay time per airline

Figure 12 depicts average delay (in minutes) per airline for arrival delays. The overall average arrival delay is 3.5 minutes, which is shorter than the departure delay time. The top 3 airlines were also the top 3 for departure delays.

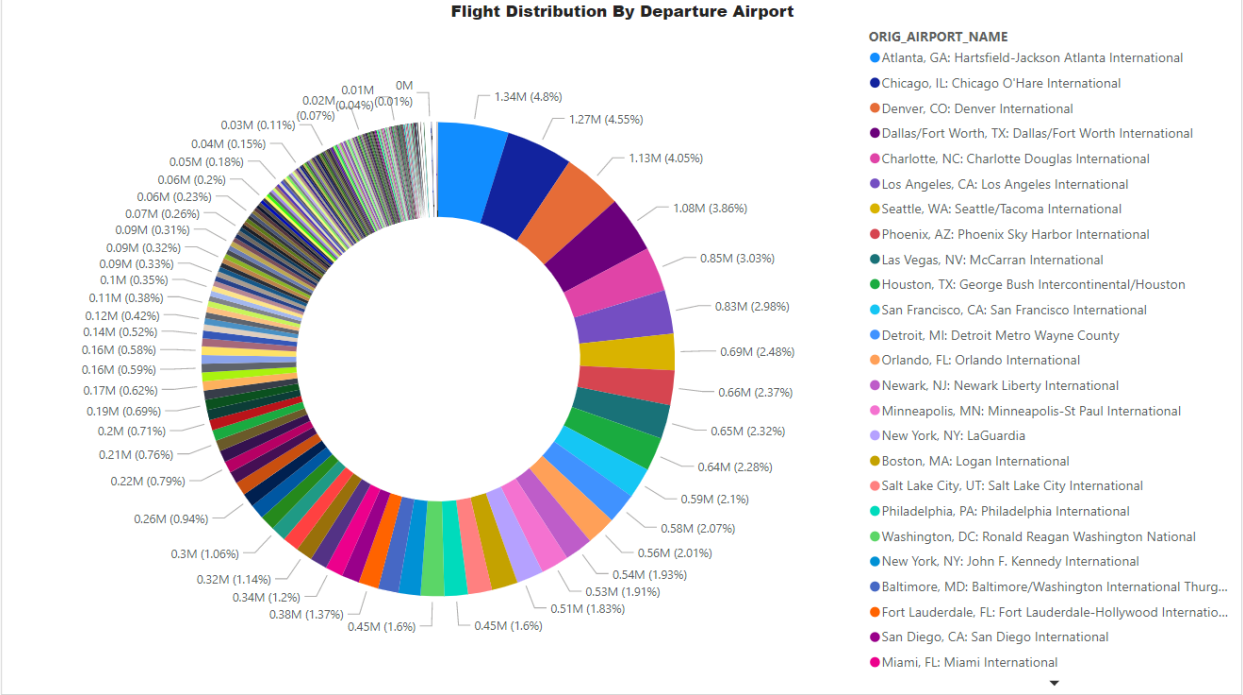
Fig 12



(8) Market share by airline

Fig 13 depicts the distribution of flights by departure airports. The highest number of flights departed from Atlanta GA (1.34m), followed closely by Chicago IL (1.27m). These two departure airports constitute close to 10% of all departure flights.

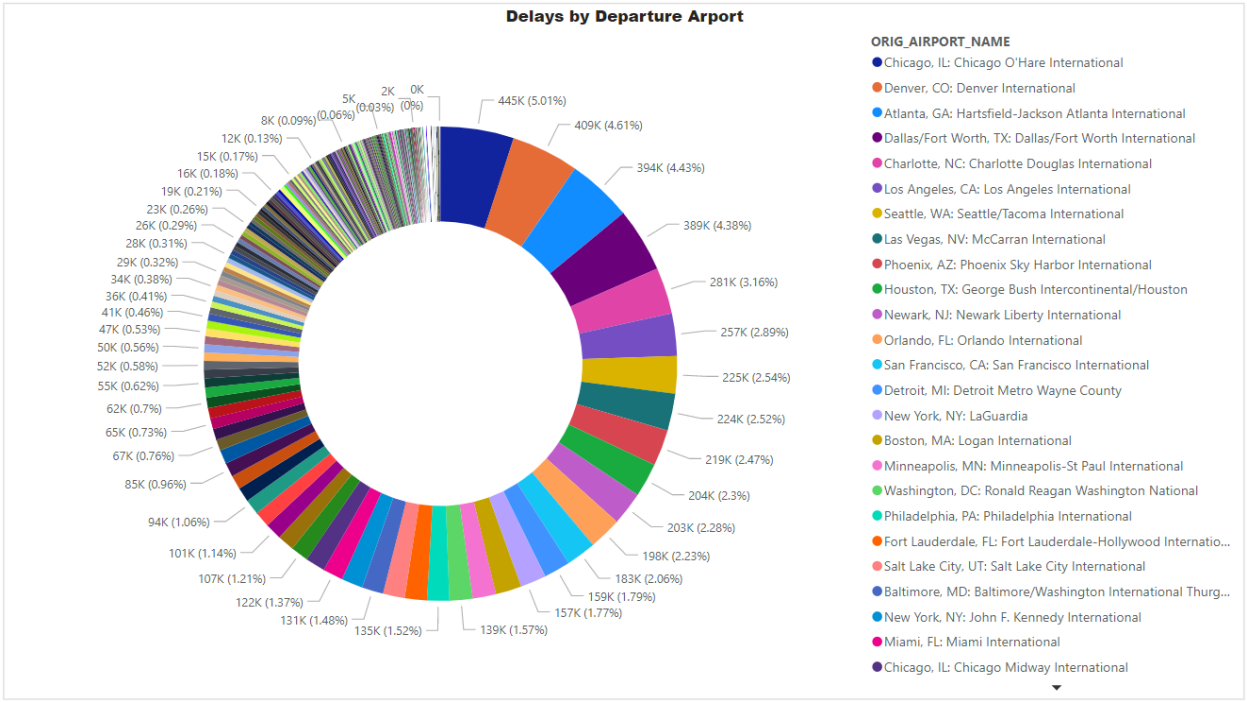
Fig 13



(9) Flight distribution by departure airport

Fig 14 depicts the distribution of delays across the departure airports. Chicago IL has the highest number of delays (0.5m), which constitutes 5.0% of all the delays. Naturally, it would be expected that being the airport with the highest number of flights, the delays would also follow suit. This is followed by Denver airport with 0.4m delayed flights, constituting 4.6% of all delays.

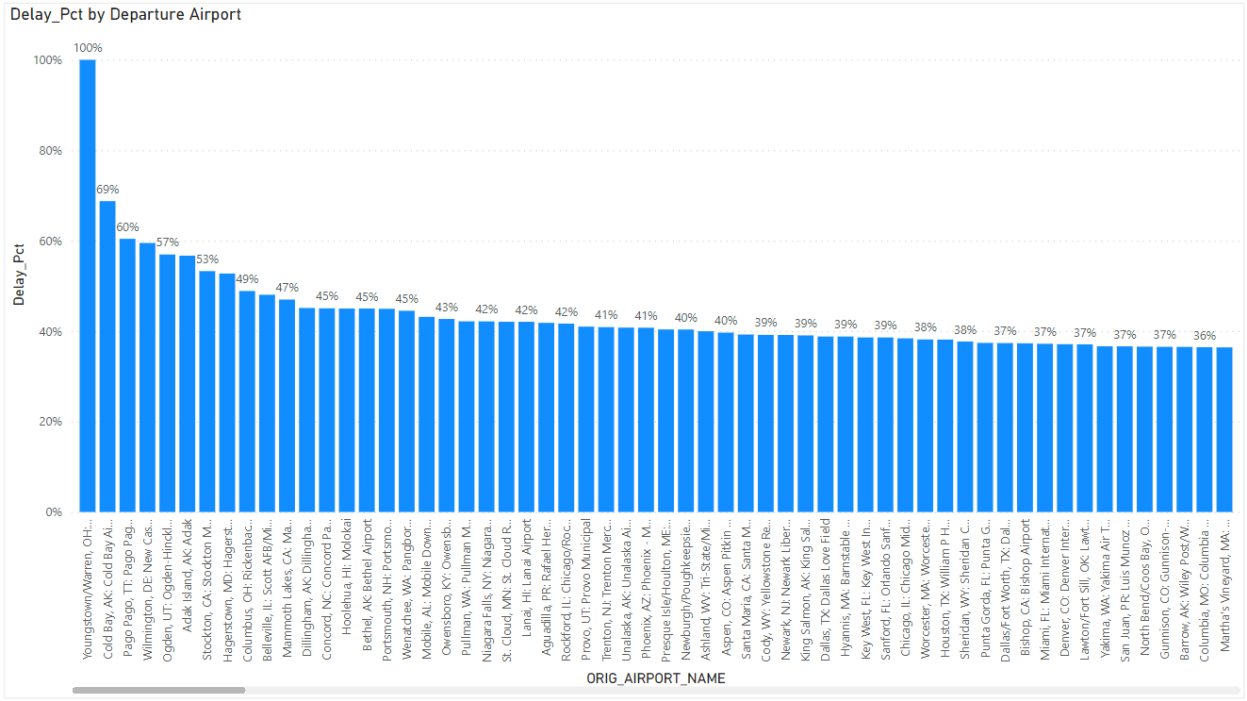
Fig 14



(10) Delay percentage per departure airport

Fig 15 depicts delay percentage per departure airport, which is number of delayed flights over the number of flights. Per airport, Youngstown Airport has the highest number of delayed flights at 100%, followed by Cold bay at 69%.

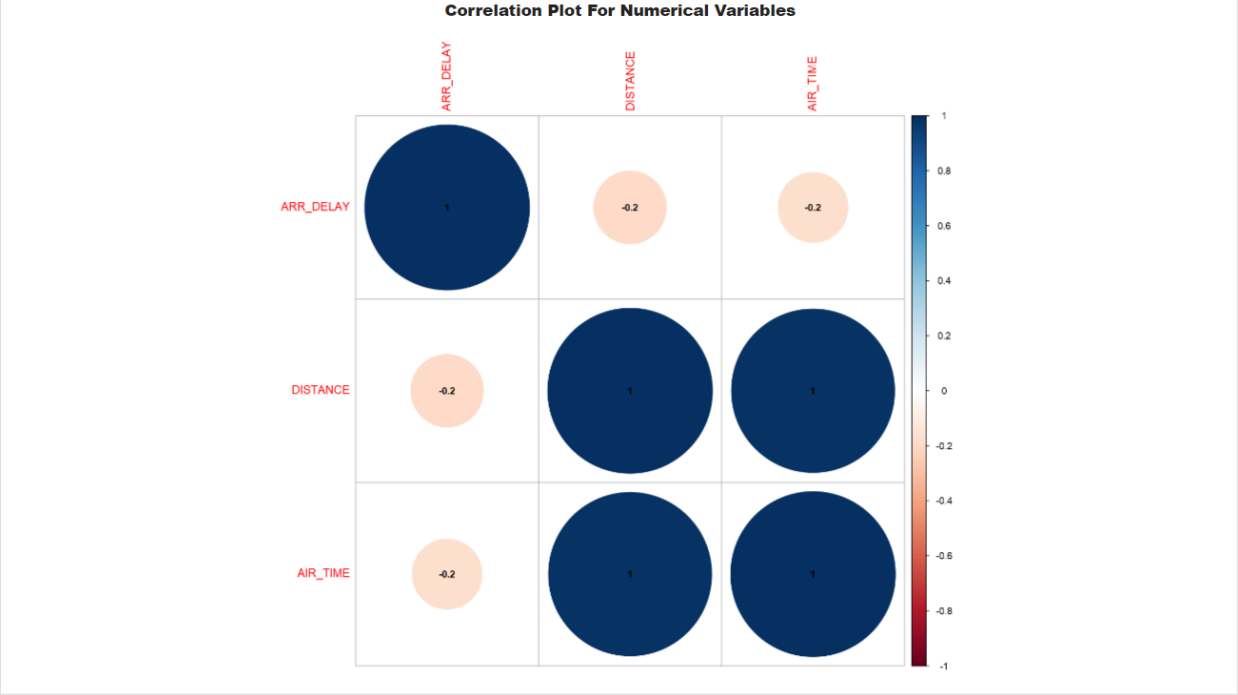
Figure 15



(11) Correlation plot

Fig 16 depicts correlations between the numerical variables. The plot shows, as expected, a high correlation between flight time and flight distance i.e. the longer the flight time, the longer the distance. The more interesting correlation is the one between delay time vs flight distance and flight time. It depicts a low negative correlation (-0.2), which can be interpreted as the longer the flight distance or flight time, the lower the delay time. This would make sense, as the longer flights would have enough time to catch up any delayed time, during the longer flight time.

Fig 16



5. Conclusion

We have implemented the project on PySpark, utilising the DataFrame API. This enabled me to efficiently implement parallelism and partitioning. Subsequently, we have done visualisations of the data using PowerBi. From these visualisations, it is evident that the delay feature is influenced by the airline, airport, month of the year and day of the week. If machine learning prediction was required for the project, these are the variables that would be explored closely and modelled as predictors for delay.

References

[1] "Bureau of Statistics," Bureau of Statistics, [Online]. Available: https://www.transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr. [Accessed 1 September 2022].

[2] JaHerbas, "GitHub: JaHerbas/Predicting Flight Delays," [Online]. Available: https://github.com/JaHerbas/Predicting_Flight_Delays. [Accessed 28 August 2022].

[3] R. Xin, M. Armbrust and D. Liu, "Databricks," [Online]. Available: https://www.databricks.com/blog/2015/02/17/introducing-dataframes-in-spark-for-large-scale-data-science.html. [Accessed 25 October 2022].

[4] NNK, "SparkByExamples," [Online]. Available: https://sparkbyexamples.com/spark/spark-sparkcontext/. [Accessed 25 October 2022].