

# PREDICTING DELAYS IN FLIGHTS

George Mason University  
ECE – 552 | Prof. Erton Boci

Indu Priya Sharma  
George Mason University  
Fairfax, Virginia  
isharma3@gmu.edu

## *Abstract*

Tens of thousands of individuals face inconvenience and maybe even skipping on events such as weddings, funerals, sporting or maybe even vacations due to the flights getting delayed. A flight is deemed to be delayed by the Federal Aviation Administration (FAA) when it departs 15 mins after scheduled time [1]. The idea for this topic as my final presentation occurred to me when I was traveling to Florida for a conference and the airlines which I booked got delayed by 1 hour 30 mins. Which was the case with other fellow mates who had come from California and Seattle.

The initial idea was to find out the airline who had enormous flight delays from 2009 to 2018. As per the feedback received, the dataset was changed to flight delays in the US from January 2022 to August 2022. This study demonstrates the possible reasons for flights getting delayed by channelizing it using Azure VM, Spark technologies, Jupyter notebook. Exploratory data analysis was conducted in pyspark, python and sql following that the machine learning was conducted in pyspark.

## I. INTRODUCTION

The airline industry or the aviation sector in the USA is worth more than \$140 billion with a growth of nearly 24% in market size [2]. An industry worth billions is mainly dependent on the airlines being on time and offering quality services. Services being offered is usually subjective, however the on time performance of the airline carrier can significantly impact not only the business but also the customers as they would rather prefer to be on time rather than being delayed. Almost for the first couple of quarters, nearly 1 of every 5 domestic flights were getting delayed. The year 2022 has observed the most number of cancellations till date since the year 2014. The fare for the flight seats have roughly increased by an average of 17%. With these statistics, it became crucial to gain insights about the aviation sector [3].

## II. LITERATURE REVIEW

According to [3], one of the primary reasons for the flight delays is the shortage of the pilots. During the covid period, when the number of operating flights and the resultant air traffic had decreased (due to the varying covid-19 policies of different

countries), there were a lot of pilots that had been let go. Now with the restrictions lifting the aviation sector going back to normal, there is a shortage of pilots to cope up with the growing demand. So there are a lot of delays in allocating pilots and sometimes it even leads to cancellations.

As the project would deal with machine learning, it was essential to review how the models would be evaluated. The model evaluation starts by analyzing the Confusion Matrix, Accuracy, Precision, Recall [4] for the models and comparing them to assess their capabilities:

**Confusion Matrix:** It is a matrix of data instances which are as shown in the below images that informs about True Positive, False Positive, True Negative, False Negative

**Accuracy:** Accuracy is simply the percentage of the correctly identified instances over all the instances. The formula is  $(\text{True Positive} + \text{True Negative}) / (\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative})$

**Precision:** The formula is:  $(\text{True Positive}) / (\text{True Positive} + \text{False Positive})$ . Precision refers to the proportion of true positives over all the positive predicted.

**Recall:** The formula is:  $(\text{True Positive}) / (\text{True Positive} + \text{False Negative})$ . Recall refers to the proportion of true positives over the predicted positives sometimes also referred as True Positive Rate.

## III. METHODOLOGY

### III.I. Acquiring the dataset

For this project, the dataset was extracted from the Bureau of Transport Statistics [5] for the year of 2022. At the time of acquiring data, the data was available from January till the month of August. The data was acquired by clicking on the desired parameter and selecting the month. So, for each individual month, a .csv was downloaded and then using spark all the files were read together by using the \*.csv option in read file. The parameters that had been selected are available in the next section.

### III.II. Dataset Overview

The combined dataset contained 4,495,839 rows spanned across 22 columns. Here is an overview of the dataset:

## Predicting Delays inFlight

Attribute	Overview
MONTH	Month of Flight
DAY_OF_MONTH	Day of the month, flight originates
DAY_OF_WEEK	Day of the week, flight originates
OP_CARRIER	Code of airline operating the flight
ORIGIN	Airport code for the flight's origin
ORIGIN_STATE_ABR	State code for the flight's origin
DEST	Airport code for the flight's destination
DEST_STATE_ABR	State code for the flight's destination
CRS_DEP_TIME	Official departure time of the flight
DEP_TIME	Actual departure time of the flight
DEP_DELAY_MIN	Delay in departure of flights in minutes
CRS_ARR_TIME	Official arrival time of the flight
ARR_TIME	Actual arrival time of the flight
ARR_DELAY_MIN	Delay in arrival of flights in minutes
CANCELLED	Flights canceled or not, 1 = canceled, 0 = not canceled
CANCELLATION_CODE	Code of cancellation if flight canceled
DISTANCE	Distance of the flight
CARRIER_DELAY	Delay in minutes because of the airline carrier
WEATHER_DELAY	Delay in minutes because of the weather
NAS_DELAY	Delay in minutes because of the National Air System
SECURITY_DELAY	Delay in minutes because of the security
LATE_AIRCRAFT_DELAY	Delay in minutes because of the late aircraft

Table 1: Dataset Overview

### III.III. Spark Scala

There were 8 sets of .csv files that had the data ranging from January 2022 till August 2022. It was essential to merge the files, so scala was utilized in the software tool called zeppelin. Here's the code for the same:

```
val newdf = spark.read.options(Map("header" -> "true")).csv("C:/BigData/~notebookJupyter/AirlinesData/2022_*.csv")
newdf.count()

newdf: org.apache.spark.sql.DataFrame = [MONTH: string, DAY_OF_MONTH: string ... 20 more fields]
res16: Long = 4495839
```

Image1: Zeppelin Scala loading the file

We can see that there are 4,495,839 records in total spanned across 8 months of flight data in 2022. A key requirement was to choose the option of header as true, otherwise it was treating each set of headers as one column while merging. Next was to export the dataset to postgresql for exploratory data analysis and storage of data. Here's the code for it:

```
spark.sql("CREATE DATABASE learn_spark_new_db")
spark.sql("USE learn_spark_new_db")
```

```
newdf.write
  .format("jdbc")
  .mode("overwrite")
  .option("driver", "org.postgresql.Driver")
  .option("url", "jdbc:postgresql://localhost:5432/sampleData")
  .option("dbtable", "airlinedata")
  .option("user", "postgres")
  .option("password", "admin")
  .save()
```

Image2: Exporting and storing file in Postgresql

Since we imported the csv file to zeppelin, the next step is to check its connectivity with psql. So, the first 2 lines of code create a new db connection from zeppelin to psql. Next step is to write the csv data to psql which is done using jdbc driver. Following that, the merged dataframe was exported to a .csv file using the coalesce(1) function for further analysis in pyspark.

```
newdf.coalesce(1).write.option("header", "true").option("emptyValue", "").
  csv("BigData/~notebookJupyter/AirlinesData/NewDF")
```

Image3: Exporting to new .csv file

### III.IV. Pyspark and Python

The tool utilized for working in pyspark and python was jupyter notebook as a part of Anaconda suite of software. The libraries utilized were Pandas for creating, processing and analyzing dataframe. Numpy library was also utilized for statistical analysis. Additionally, the seaborn library along with the matplotlib library were utilized for visualization purposes. Further sklearn, pyspark.ml libraries were utilized for testing out machine learning algorithms.

The first step was to utilize spark.read.csv() to read the .csv file, the file in consideration would be the newly exported merged .csv file from image 3.

```
airline22DF = spark.read.csv('NewDF.csv', sep=',', \
  header=True, inferSchema=True, nullValue='NA')
```

Image4: Pyspark reading the file

As the first step of exploratory data analysis, it was essential to check the data type for each attribute which was done using the printSchema():

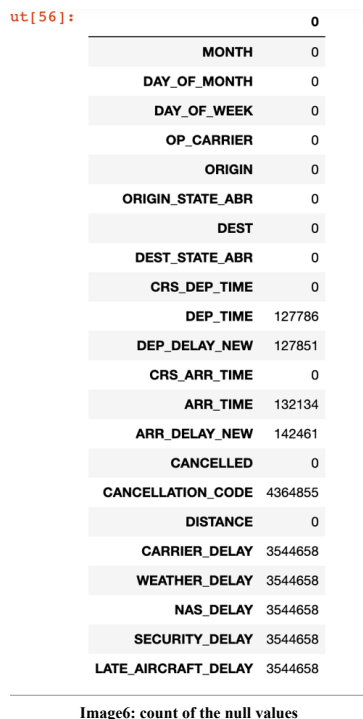
```
1 airline22DF.printSchema()

root
|-- MONTH: integer (nullable = true)
|-- DAY_OF_MONTH: integer (nullable = true)
|-- DAY_OF_WEEK: integer (nullable = true)
|-- OP_CARRIER: string (nullable = true)
|-- ORIGIN: string (nullable = true)
|-- ORIGIN_STATE_ABR: string (nullable = true)
|-- DEST: string (nullable = true)
|-- DEST_STATE_ABR: string (nullable = true)
|-- CRS_DEP_TIME: integer (nullable = true)
|-- DEP_TIME: integer (nullable = true)
|-- DEP_DELAY_MIN: double (nullable = true)
|-- CRS_ARR_TIME: integer (nullable = true)
|-- ARR_TIME: integer (nullable = true)
|-- ARR_DELAY_MIN: double (nullable = true)
|-- CANCELLED: double (nullable = true)
|-- CANCELLATION_CODE: string (nullable = true)
|-- DISTANCE: double (nullable = true)
|-- CARRIER_DELAY: double (nullable = true)
|-- WEATHER_DELAY: double (nullable = true)
|-- NAS_DELAY: double (nullable = true)
|-- SECURITY_DELAY: double (nullable = true)
|-- LATE_AIRCRAFT_DELAY: double (nullable = true)
```

Image5: Schema of the dataset

## Predicting Delays inFlight

Next, the aim was to check the number of missing values. Here is the count of the null values in the dataset.



ut[56]:	0
MONTH	0
DAY_OF_MONTH	0
DAY_OF_WEEK	0
OP_CARRIER	0
ORIGIN	0
ORIGIN_STATE_ABR	0
DEST	0
DEST_STATE_ABR	0
CRS_DEP_TIME	0
DEP_TIME	127786
DEP_DELAY_NEW	127851
CRS_ARR_TIME	0
ARR_TIME	132134
ARR_DELAY_NEW	142461
CANCELLED	0
CANCELLATION_CODE	4364855
DISTANCE	0
CARRIER_DELAY	3544658
WEATHER_DELAY	3544658
NAS_DELAY	3544658
SECURITY_DELAY	3544658
LATE_AIRCRAFT_DELAY	3544658

Image6: count of the null values

Following this, the next step was to convert the data into parquet format for further exploratory data analysis. Here is the code for the parquet conversion:

### Parquet

```
airline22DFP = airline22DF
airline22DFP.write.parquet("airline22DFP.parquet")

airline22DFParq = spark.read.parquet("airline22DFP.parquet")

airline22DFParq.createOrReplaceTempView("airline22DFParq")

airline22DFParq.count()

4495839
```

Image7: Creating Parquet

The parquet would be used for analysis done in SQL in a jupyter notebook. However, a point to note is that since the visualizations cannot be performed in jupyter using parquet, pandas were used. Libraries Seaborn and Matplot were utilized for generating bar plots and heat maps.

### III.V. SQL in Postgres

The primary SQL that has been used is Postgresql. The GUI that has been used is PgAdmin 4. The data had already been imported using spark scala with the source code available in Image 2. First couple of steps were to check the count of the

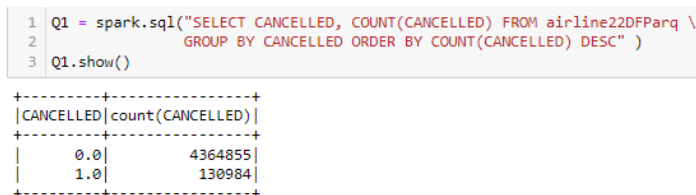
records and first few records, following which exploratory data analysis had been conducted.

By utilizing SQL, the aim was to figure out crucial information such as:

- Count of Records
- Top 20 airports and the count of flights to and fro from them
- Carriers and the count of flights canceled by each
- Airports with canceled flights.
- Count of Delayed flights, per carrier as well

### III.VI. SQL in Pyspark

Another SQL that has been used is the inbuilt SQL library of pyspark in jupyter. A couple of queries were run to ensure that the dataset that was exported was accurate and the results are in sync. First one amongst those was the count for which the query and the output is below:



```
1 Q1 = spark.sql("SELECT CANCELLED, COUNT(CANCELLED) FROM airline22DFParq \
2 GROUP BY CANCELLED ORDER BY COUNT(CANCELLED) DESC")
3 Q1.show()
```

CANCELLED	count(CANCELLED)
0.0	4364855
1.0	130984

Image8: Query 1

The first query is about the count of flights that have been canceled. 1 denotes that the flights have been canceled and the count is 130,984 while 0 denotes the flight has not been canceled and the count is 4,364,855. Another query was for checking the count of flights from each origin airport.

### III.VII. Machine Learning in Pyspark

Prior to starting machine learning, it was essential to clean up the dataset. So, here are the steps that were taken:

1. First step was to drop the column 'CANCELLATION\_CODE' as it would not be relevant to the machine learning algorithm.
2. Secondly, the missing values were dropped using the code `df.na.drop(subset = 'columnname')`
3. Next step was to extract useful information from the depart time and convert delay into a new target column that would be binary (0 and 1):
  - a. It was done by using the UDF which stands for User Defined Function. It is an option available in the pyspark for creating a function similar to the way a lambda function would operate in the python where it would call upon a defined function.

## Predicting Delays inFlight

- b. In this case, the functions that had been earlier designed in python language for dataset manipulation were used. Using the lambda function along with udf(), these functions were converted to UDF functions in pyspark language. Below is the code for the same.

```
newTUDF1 = udf(lambda z: convertTimeDay(z))
newTUDF2 = udf(lambda x: delayCheck(x))
```

Image9: pyspark udf function Output

- c. Then with the function convertTimeDay(), the time available in the CRS\_DEP\_TIME was segmented into four time periods under the new column name 'DEPPARTOFTDAY':
- 00:00 am - 06:00 am to Red Eye
  - 06:00 am - 12:00 pm to Morning
  - 12:00 pm - 06:00 pm to Afternoon
  - 06:00 pm - 00:00 am to Evening

```
1 def convertTimeDay(s):
2     res = ""
3     if s >= 0 and s < 600:
4         res = "Red Eye"
5     elif s >= 600 and s < 1200:
6         res = "Morning"
7     elif s >= 1200 and s < 1800:
8         res = "Afternoon"
9     elif s >= 1800 and s < 2400:
10        res = "Evening"
11    return res
```

Image10: convertTimeDay() function code

- d. Next function delaycheck() was used for converting the column into the target column '. The condition used was that any flight delayed more than 15 minutes [1] would be considered delayed and under would be considered as non delayed. They would be given 0 and 1 values respectively.

```
def delayCheck(s):
    res=0
    if s > 15:
        res= 1
    elif s <=15:
        res =0
    elif s is None:
        res = 123
    return res
```

Image11: delaycheck() function code

- e. Here is an overview of the results:

CRS_DEP_TIME	DEP_DELAY_NEW	DEPPARTOFTDAY	DELAYED
2225	45.0	Evening	1
1735	1.0	Afternoon	0
1950	13.0	Evening	0
540	0.0	Red Eye	0
2020	12.0	Evening	0
1145	0.0	Morning	0
1555	30.0	Afternoon	1
710	0.0	Morning	0
2000	22.0	Evening	1
1630	4.0	Afternoon	0
1030	0.0	Morning	0
1955	120.0	Evening	1
1035	0.0	Morning	0
1015	0.0	Morning	0
1945	55.0	Evening	1
530	0.0	Red Eye	0
605	0.0	Morning	0
1845	6.0	Evening	0
1155	14.0	Morning	0
1740	0.0	Afternoon	0

only showing top 20 rows

Image12: overview of the results

We can see from the above image that the CRS\_DEP\_TIME 2225 (essentially 22:25) was converted to Evening as it lies between 06:00 pm - 00:00 am. Since the delay was 45.0 minutes (greater than 15 minutes) the DELAYED shows 1. While the entry below had a delay of only one minute, so it shows DELAYED as 0.

- f. Then with the function convertTimeDay(), the time available in the CRS\_DEP\_TIME was segmented into four time periods under the new column name 'DEPPARTOFTDAY':
- 00:00 am - 06:00 am to Red Eye
  - 06:00 am - 12:00 pm to Morning
  - 12:00 pm - 06:00 pm to Afternoon
  - 06:00 pm - 00:00 am to Evening
4. Then the next step was to check the count of delayed (value = 1) vs the non delayed entries (value = 0):

```
1 airline22DFMDN1.groupby("DELAYED").count().show()
```

DELAYED	count
0	3430182
1	937806

Image13: count of delayed flights

5. Following that next step was to check the schema for the updated dataset that was done using the printSchema():

```
1 airline22DFMDN1.printSchema()

root
|-- MONTH: integer (nullable = true)
|-- DAY_OF_MONTH: integer (nullable = true)
|-- DAY_OF_WEEK: integer (nullable = true)
|-- OP_CARRIER: string (nullable = true)
|-- ORIGIN: string (nullable = true)
|-- ORIGIN_STATE_ABR: string (nullable = true)
|-- DEST: string (nullable = true)
|-- DEST_STATE_ABR: string (nullable = true)
|-- CRS_DEP_TIME: integer (nullable = true)
|-- DEP_TIME: integer (nullable = true)
|-- DEP_DELAY_NEW: double (nullable = true)
|-- CRS_ARR_TIME: integer (nullable = true)
|-- ARR_TIME: integer (nullable = true)
|-- ARR_DELAY_NEW: double (nullable = true)
|-- CANCELLED: double (nullable = true)
|-- DISTANCE: double (nullable = true)
|-- CARRIER_DELAY: double (nullable = true)
|-- WEATHER_DELAY: double (nullable = true)
|-- NAS_DELAY: double (nullable = true)
|-- SECURITY_DELAY: double (nullable = true)
|-- LATE_AIRCRAFT_DELAY: double (nullable = true)
|-- DEPARTOFTDAY: string (nullable = true)
|-- DELAYED: string (nullable = true)
```

Image14: Schema of dataset

6. DELAYED column was in string format, so it was converted into integer using the cast(IntegerType()) function.
7. Following this, the next step was to eliminate the columns that wouldn't be required for further machine learning process which were: 'CRS\_DEP\_TIME', 'DEP\_TIME', 'CRS\_ARR\_TIME', 'ARR\_TIME', 'DEP\_DELAY\_NEW', 'ARR\_DELAY\_NEW', 'CARRIER\_DELAY', 'WEATHER\_DELAY', 'NAS\_DELAY', 'SECURITY\_DELAY', 'LATE\_AIRCRAFT\_DELAY', 'DEST', 'DEST\_STATE\_ABR'.
8. Next, the DISTANCE was converted into float from double type using the cast(FloatType()) function.
9. Schema for the remaining dataset:

```
1 airlineModelN.printSchema()

root
|-- MONTH: integer (nullable = true)
|-- DAY_OF_MONTH: integer (nullable = true)
|-- DAY_OF_WEEK: integer (nullable = true)
|-- OP_CARRIER: string (nullable = true)
|-- ORIGIN: string (nullable = true)
|-- ORIGIN_STATE_ABR: string (nullable = true)
|-- CANCELLED: double (nullable = true)
|-- DISTANCE: float (nullable = true)
|-- DEPARTOFTDAY: string (nullable = true)
|-- DELAYED: integer (nullable = true)
```

Image15: Schema of dataset

10. The remaining dataset now has the following columns: 'MONTH', 'DAY\_OF\_MONTH', 'DAY\_OF\_WEEK', 'OP\_CARRIER', 'ORIGIN', 'ORIGIN\_STATE\_ABR', 'CANCELLED', 'DISTANCE', 'DEPARTOFTDAY', 'DELAYED'. 'DELAYED' is the target or the dependent variable.

For our machine learning there are steps taken in pyspark [9] [10]:

1. String Indexer
2. One Hot Encoder
3. Vector Assembler
4. Standard Scalar
5. Splitting dataset
6. Deploying Model
7. Model Evaluation

## III.VIII.I. String Indexer:

The dataset has four categorical variables: 'ORIGIN', 'OP\_CARRIER', 'ORIGIN\_STATE\_ABR', 'DEPARTOFTDAY'. For the categorical variables, in pyspark, the string indexer can be utilized for the purposes of label encoder.

The function would convert the categories into numbers starting from 0 till the count of unique categories in that column. 0 would be for the category (or value) with the highest frequency, 1 for the second highest frequency and 2 for the third highest frequency and so on. So for all the four categorical columns the string indexer was used. So for the above mentioned four categorical columns, their respective string indexed columns were generated. The code along with the resulting columns:

```
airlineModelInd = StringIndexer().setInputCol ("OP_CARRIER").
setOutputCol ("OP_CARRIER_INDEX").fit(airlineModelInd).transform(airlineModelInd)

airlineModelInd = StringIndexer().setInputCol ("ORIGIN").
setOutputCol ("ORIGIN_INDEX").fit(airlineModelInd).transform(airlineModelInd)

airlineModelInd = StringIndexer().setInputCol ("DEPARTOFTDAY").
setOutputCol ("DEPARTOFTDAY_INDEX").fit(airlineModelInd).transform(airlineModelInd)

airlineModelInd = StringIndexer().setInputCol ("ORIGIN_STATE_ABR").
setOutputCol ("ORIGIN_STATE_ABR_INDEX").fit(airlineModelInd).transform(airlineModelInd)
```

[DELAYED]	[OP_CARRIER_INDEX]	[ORIGIN_INDEX]	[DEPARTOFTDAY_INDEX]	[ORIGIN_STATE_ABR_INDEX]
1	0.0	23.0	2.0	2.0
0	0.0	23.0	1.0	2.0
0	0.0	23.0	2.0	2.0
0	0.0	23.0	3.0	2.0
0	0.0	23.0	2.0	2.0
0	0.0	23.0	0.0	2.0
1	0.0	23.0	1.0	2.0
0	0.0	23.0	0.0	2.0
1	0.0	23.0	2.0	2.0
0	0.0	23.0	1.0	2.0
0	0.0	23.0	0.0	2.0
1	0.0	23.0	2.0	2.0
0	0.0	23.0	0.0	2.0
0	0.0	23.0	0.0	2.0
1	0.0	23.0	2.0	2.0
0	0.0	23.0	3.0	2.0
0	0.0	23.0	0.0	2.0

Image16: String Indexer

## III.VII.II. One Hot Encoder:

For one hot encoding in pyspark (or spark), it was a requirement to first do the string indexing as categorical variables cannot be encoded directly. So, from the numerical value (acquired after string indexing) one hot encoder can be used with the function OneHotEncoderEstimator(). So the four indexed column 'OP\_CARRIER\_INDEX', 'ORIGIN\_INDEX', 'DEPARTOFTDAY\_INDEX' and 'ORIGIN\_STATE\_ABR\_INDEX' were encoded, the code along with results:



## Predicting Delays inFlight

```
airlineModelIndEnc = OneHotEncoderEstimator().setInputCols(["OP_CARRIER_INDEX"]).  
setOutputCols(["OP_CARRIER_INDEX_ENC"]).fit(airlineModelInd).transform(airlineModelInd)
```

```
airlineModelIndEnc = OneHotEncoderEstimator().setInputCols (["ORIGIN_INDEX"]).  
setOutputCols (["ORIGIN_INDEX_ENC"]).fit(airlineModelIndEnc).transform(airlineModelIndEnc)
```

```
airlineModelIndEnc = OneHotEncoderEstimator().setInputCols (["DEPPARTOFDAY_INDEX"]).  
setOutputCols (["DEPPARTOFDAY_INDEX_ENC"]).fit(airlineModelIndEnc).transform(airlineModelIndEnc)
```

```
airlineModelIndEnc = OneHotEncoderEstimator().setInputCols(["ORIGIN_STATE_ABR_INDEX"]).  
setOutputCols(["ORIGIN_STATE_ABR_INDEX_ENC"]).fit(airlineModelIndEnc).transform(airlineModelIndEnc)
```

```

1 airlineModelIndEnc.show()
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|MONTH|DAY_OF_MONTH|DAY_OF_WEEK|OP_CARRIER|ORIGIN|ORIGIN_STATE_ABR|CANCELLED|DISTANCE|DEPARTDAY|DEPARTDAY_INDEX|ORIGIN_INDEX|ORIGIN_STATE_ABR_INDEX|OP_CARRIER_INDEX_ENC|ORIGIN_INDEX_ENC|DEPARTDAY_INDEX_ENC|ORIGIN_STATE_ABR_INDEX_ENC|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|1|7|13|3|WN|FULL|2.0|2.0|0.0|925.0|Evening|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[2],[1.0])|(52,[2],[1.0])|
|1|7|13|3|WN|FULL|2.0|2.0|0.0|973.0|Afternoon|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[1],[1.0])|(52,[2],[1.0])|
|0|7|13|3|WN|FULL|2.0|2.0|0.0|1108.0|Evening|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[2],[1.0])|(52,[2],[1.0])|
|0|7|13|3|WN|FULL|2.0|2.0|0.0|1108.0|Red Eye|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[1],[1.0])|(52,[2],[1.0])|
|1|7|13|3|WN|FULL|2.0|2.0|0.0|899.0|Evening|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[2],[1.0])|(52,[2],[1.0])|
|0|7|13|3|WN|FULL|2.0|2.0|0.0|899.0|Morning|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[0],[1.0])|(52,[2],[1.0])|
|1|7|13|3|WN|FULL|2.0|2.0|0.0|957.0|Afternoon|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[1],[1.0])|(52,[2],[1.0])|
|0|7|13|3|WN|FULL|2.0|2.0|0.0|957.0|Morning|(16,[0],[1.0])|369.0|
|[23],[1.0]|(3,[0],[1.0])|(52,[2],[1.0])|

```

### Image17: One Hot Encoder

After the indexing the columns data type becomes double and after the encoding is done, the output is in vectors. Here is the schema after the indexing and the one hot encoding:

```
1 airlineModelIndEnc.printSchema()
```

```
root
|-- MONTH: integer (nullable = true)
|-- DAY_OF_MONTH: integer (nullable = true)
|-- DAY_OF_WEEK: integer (nullable = true)
|-- OP_CARRIER: string (nullable = true)
|-- ORIGIN: string (nullable = true)
|-- ORIGIN_STATE_ABR: string (nullable = true)
|-- CANCELLED: double (nullable = true)
|-- DISTANCE: float (nullable = true)
|-- DEPARTDAY: string (nullable = true)
|-- DELAYED: integer (nullable = true)
|-- OP_CARRIER_INDEX: double (nullable = false)
|-- ORIGIN_INDEX: double (nullable = false)
|-- DEPARTDAY_INDEX: double (nullable = false)
|-- ORIGIN_STATE_ABR_INDEX: double (nullable = false)
|-- OP_CARRIER_INDEX_ENC: vector (nullable = true)
|-- ORIGIN_INDEX_ENC: vector (nullable = true)
|-- DEPARTDAY_INDEX_ENC: vector (nullable = true)
|-- ORIGIN_STATE_ABR_INDEX_ENC: vector (nullable = true)
```

**Image18: Schema after indexing and encoding**

### III.VII.III. Vector Assembler:

For the purpose of training in machine learning model, the vector assembler converts the independent columns as input into one column as output that can be used as:

**3.1. First Set:** Input Columns: "MONTH", "DAY\_OF\_MONTH", "DAY\_OF\_WEEK", "DISTANCE", "OP\_CARRIER\_INDEX\_ENC", "ORIGIN\_INDEX\_ENC", "DEPARTOFTODAY INDEX\_ENC"

; Output Column: "VAM1" [This subset contains the origin airport]

**3.2. Second Set:** Input Columns: "MONTH", "DAY\_OF\_MONTH", "DAY\_OF\_WEEK", "DISTANCE", "OP\_CARRIER\_INDEX\_ENC", "ORIGIN\_STATE\_ABR\_INDEX", "DEPART\_OF\_DAY\_INDEX\_ENC"; Output Column: "VAM2"  
[This subset contains the origin airport's state instead]

Here is the code for the first one with the airport:

```
1 VectorAssemblerairlineModel1 = VectorAssembler()\n2     .setInputCols (["MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "DISTANCE", \n3                     "OP_CARRIER_INDEX_ENC", "ORIGIN_INDEX_ENC", "DEPARTOFTDAY_INDEX_ENC"])\n4     .setOutputCol  ("VAMI")
```

```
1 airlineVectAssem = vectAssemblerairlineModel1.transform(airlineModelIndEnc)
```

```
1 airlineVectAssem.select('VAM1').show()
```

[illegible]

**Image19: Vector Assembler**

### III.VII.IV. Standard Scaler:

This is used for standardizing or normalizing the features. This was used for scaling the output columns VAM1 and VAM2 that was the output column from the above step of Vector Assembler using the function StandardScaler; the code:

```
airlineVectAssemScal = StandardScaler().setInputCol ("VAM1").  
setOutputCol ("FEATURES1").fit(airlineVectAssem).transform(airlineVectAssem)
```

**Image20: Standard Scaler**

### III.VII.V. Splitting dataset:

For machine learning, the dataset needs to be split into two sets of training and testing. The percentage chosen for this purpose was 80% and 20% in both the situations (airport and state); the code:

```
trainA1, testA1 = airlineVectAssemScal.randomSplit([0.8,0.2],seed=123)
```

### Image21: Standard Scaler

### III.VII.VI. Deploying Model:

For both the set of features (VAM1 airport and VAM2 state), the following models were deployed:

**a. Logistic Regression:**

The method of modeling the likelihood of a discrete result given an input variable is known as

## Predicting Delays InFlight

logistic regression. The most popular types of logistic regression models a binary result, such as true or false, yes or no, and so on [6].

The function used was LogisticRegression() from the library pyspark.ml.classification with max iterations as 5. Here is the code:

```
log1 = LogisticRegression(featuresCol = 'FEATURES1', labelCol = 'DELAYED', maxIter=5)
logModel1 = log1.fit(trainA1)
predModel1 = logModel1.transform(testA1)
predModel1.select('DELAYED', 'FEATURES1', 'rawPrediction', 'prediction', 'probability').show(10)
```

DELAYED	FEATURES1	rawPrediction	prediction	probability
0	(392,[0,1,2,3,12,...])	[2.45305984844594...	0.0	[0.92078492386378...
0	(392,[0,1,2,3,12,...])	[1.62241695435993...	0.0	[0.83512818768593...
0	(392,[0,1,2,3,12,...])	[1.62241695435993...	0.0	[0.83512818768593...
1	(392,[0,1,2,3,12,...])	[1.55757588277701...	0.0	[0.82600523270891...
1	(392,[0,1,2,3,12,...])	[1.23317202322143...	0.0	[0.77437327027047...
0	(392,[0,1,2,3,12,...])	[1.07918587352375...	0.0	[0.74633988654502...
0	(392,[0,1,2,3,12,...])	[0.94619670794710...	0.0	[0.72034966241832...
1	(392,[0,1,2,3,12,...])	[1.27002729548199...	0.0	[0.78074742041272...
0	(392,[0,1,2,3,12,...])	[1.26921148529871...	0.0	[0.78060773731800...
0	(392,[0,1,2,3,12,...])	[0.94487377251475...	0.0	[0.72008308466169...

Image22: Logistic Regression

### b. Decision Tree:

A decision tree is a straightforward algorithm that resembles a flowchart, making it simple to understand. The root node, numerous tree nodes, and leaves make up a tree. Practically every node divides the data set into subsets, including the root node. Each division resembles a question about whether a specific condition is present or not for an essential attribute. Using the training set, a decision tree is constructed from top to bottom, with each level choosing the feature that best divides the training data according to the target variable [7].

The function used was DecisionTreeClassifier() from the library pyspark.ml.classification with max iterations as 5. Here is the code:

```
dt1 = DecisionTreeClassifier(featuresCol = 'FEATURES1', labelCol = 'DELAYED')
dtModel1 = dt1.fit(trainA1)
predModel1 = dtModel1.transform(testA1)
predModel1.select('DELAYED', 'FEATURES1', 'rawPrediction', 'prediction', 'probability').show(10)
```

DELAYED	FEATURES1	rawPrediction	prediction	probability
0	(392,[0,1,2,3,12,...])	[1137044.0,172946.0]	0.0	[0.86797914487896...
0	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
0	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
1	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
1	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
0	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
0	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
1	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
0	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...
0	(392,[0,1,2,3,12,...])	[1303665.0,408683.0]	0.0	[0.76133180872112...

Image23: Decision Tree

### c. Random Forest:

Random forest creates many decision trees at once so it is a form of bagging. The final prediction is then produced by combining the predictions of the many growth trees. Regression and classification issues can both be solved with a random forest [7].

The function used was RandomForestClassifier() from the library pyspark.ml.classification. Here is the code:

```
rand1 = RandomForestClassifier(featuresCol = 'FEATURES1', labelCol = 'DELAYED')
randModel1 = rand1.fit(trainA1)
predModel1 = randModel1.transform(testA1)
predModel1.select('DELAYED', 'FEATURES1', 'rawPrediction', 'prediction', 'probability').show(10)
```

DELAYED	FEATURES1	rawPrediction	prediction	probability
0	(392,[0,1,2,3,12,...])	[16.0528051187827...	0.0	[0.80264025593913...
0	(392,[0,1,2,3,12,...])	[15.6725337395053...	0.0	[0.78362668697526...
0	(392,[0,1,2,3,12,...])	[15.6725337395053...	0.0	[0.78362668697526...
1	(392,[0,1,2,3,12,...])	[15.6725337395053...	0.0	[0.78362668697526...
1	(392,[0,1,2,3,12,...])	[15.5528759054689...	0.0	[0.77764379527344...
0	(392,[0,1,2,3,12,...])	[15.5528759054689...	0.0	[0.77764379527344...
0	(392,[0,1,2,3,12,...])	[15.5528759054689...	0.0	[0.77764379527344...
1	(392,[0,1,2,3,12,...])	[15.6725337395053...	0.0	[0.78362668697526...
0	(392,[0,1,2,3,12,...])	[15.6725337395053...	0.0	[0.78362668697526...
0	(392,[0,1,2,3,12,...])	[15.5528759054689...	0.0	[0.77764379527344...

Image24: Random Forest

### d. Gradient Boosted Tree:

Gradient Boosting is a boosting model as opposed to the bagging model Random Forest. Using information from previously created predictors to build stronger predictors later is the primary concept of boosting, which aims to improve model performance. These predictors are decision trees for gradient boosting. Gradient boosting frequently employs decision trees with a much shallower level of detail than Random forest does [7].

The function used was GBTClassifier() from the library pyspark.ml.classification with max iterations as 5. Here is the code:

```
gbt1 = GBTClassifier(featuresCol = 'FEATURES1', labelCol = 'DELAYED', maxIter=10)
gbtModel1 = gbt1.fit(trainA1)
predModel1 = gbtModel1.transform(testA1)
predModel1.select('DELAYED', 'FEATURES1', 'rawPrediction', 'prediction', 'probability').show(10)
```

DELAYED	FEATURES1	rawPrediction	prediction	probability
0	(392,[0,1,2,3,12,...])	[0.83306900778770...	0.0	[0.84106023888535...
0	(392,[0,1,2,3,12,...])	[0.59281763763131...	0.0	[0.76599592801690...
0	(392,[0,1,2,3,12,...])	[0.59281763763131...	0.0	[0.76599592801690...
1	(392,[0,1,2,3,12,...])	[0.61483317878247...	0.0	[0.77376017370517...
1	(392,[0,1,2,3,12,...])	[0.49934828746824...	0.0	[0.73808223254160...
0	(392,[0,1,2,3,12,...])	[0.49934828746824...	0.0	[0.73808223254160...
0	(392,[0,1,2,3,12,...])	[0.49934828746824...	0.0	[0.73808223254160...
1	(392,[0,1,2,3,12,...])	[0.61483317878247...	0.0	[0.77376017370517...
0	(392,[0,1,2,3,12,...])	[0.61483317878247...	0.0	[0.77376017370517...
0	(392,[0,1,2,3,12,...])	[0.49934828746824...	0.0	[0.73808223254160...

Image25: Random Forest

### e. Naive Bayes:

Naive Bayes classifier is utilized for classification tasks. The Bayes theorem serves as the foundation of the classifier [8].

The function used was NaiveBayes() from the library pyspark.ml.classification with max iterations as 5. Here is the code:

```
nb1 = NaiveBayes(featuresCol = 'FEATURES1', labelCol = 'DELAYED', modelType="multinomial")
nbModel1 = nb1.fit(trainA1)
predModel1 = nbModel1.transform(testA1)
predModel1.select('DELAYED', 'FEATURES1', 'rawPrediction', 'prediction', 'probability').show(10)
```

DELAYED	FEATURES1	rawPrediction	prediction	probability
0	(392,[0,1,2,3,12,...])	[-314.24233942296...	0.0	[0.99999999910667...
0	(392,[0,1,2,3,12,...])	[-316.14728658384...	0.0	[0.99999999972564...
0	(392,[0,1,2,3,12,...])	[-316.14728658384...	0.0	[0.99999999972564...
1	(392,[0,1,2,3,12,...])	[-488.90710129907...	0.0	[0.99965635591079...
1	(392,[0,1,2,3,12,...])	[-491.08125120440...	0.0	[0.99914914579118...
0	(392,[0,1,2,3,12,...])	[-336.11496242141...	0.0	[0.999999995461140...
0	(392,[0,1,2,3,12,...])	[-73.323477372858...	0.0	[0.87927480911781...
1	(392,[0,1,2,3,12,...])	[-71.358129682142...	0.0	[0.94729580251279...
0	(392,[0,1,2,3,12,...])	[-71.541425141389...	0.0	[0.94729580251279...
0	(392,[0,1,2,3,12,...])	[-73.620713252718...	0.0	[0.87833920564880...

Image26: Naive Bayes

#### IV. DATA ARCHITECTURE

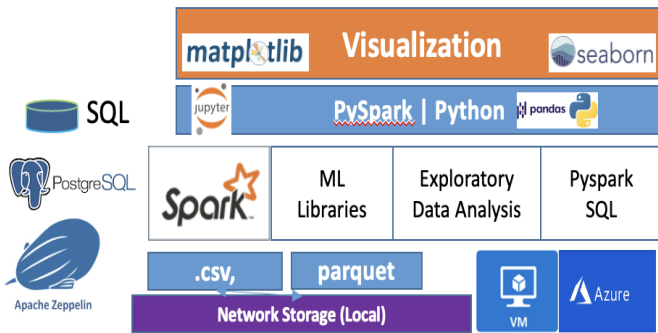


Image27: Data Architecture

The workflow has been the following:

1. Reading data using spark scala in Zeppelin
2. Merging data and exporting into postgresql for storage and exploratory data analysis
3. Importing exported data into pyspark
4. Exploratory data analysis and validation in SQL in Postgresql
5. Further analysis in python language using pandas, seaborn, matplotlib in the form of visualizations.
6. Using Pyspark machine learning libraries, deploying machine learning models

#### V. RESULTS

##### V.I. Python

The below are the findings from the exploratory data analysis through visualizations generated in python:

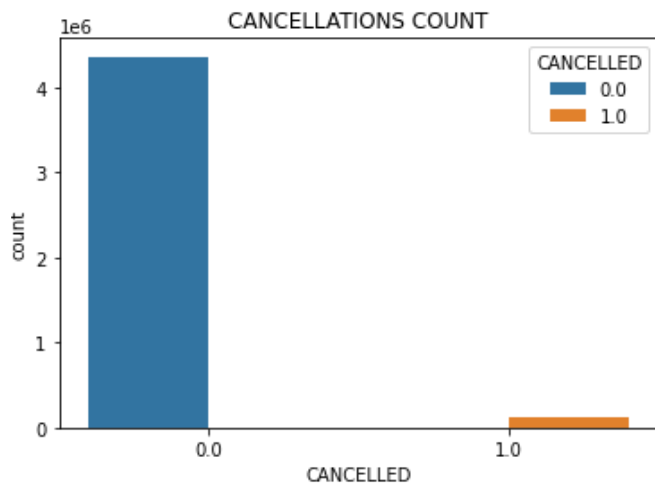


Image28: Count of flights that got canceled

Now we are going to understand the count of canceled flights. 0 illustrates good, 1 means axed. Whilst this seems to be

a huge difference between both. Now looking at the count of the delayed carriers.

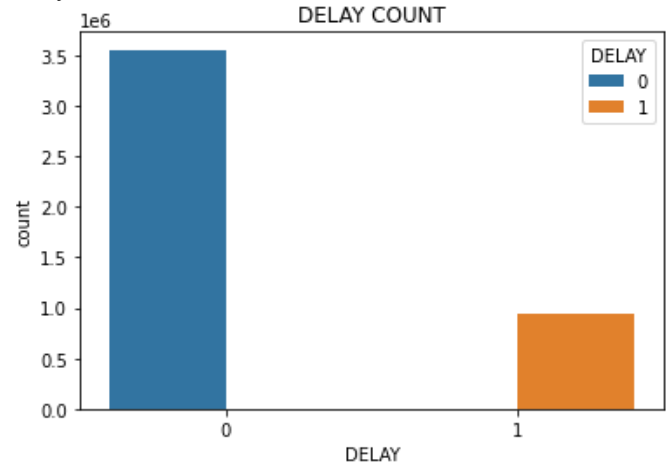


Image29: count of delayed and non-delayed flights

We have looked at the number of delays, cancellations, now we shall look at the time of the day and number of departures. Count of flights departing in Mornings are the most, Afternoon trails with Evenings. Red Eye seems to be less than 25 as per the below graph.

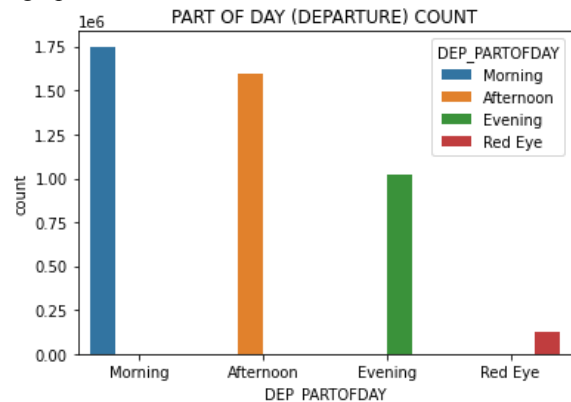


Image30: Count of departures per time of the day

Arrivals change the story of depiction; Afternoon sees most flights originating. Evening and mornings also are lofty when compared to dep\_partofday as described below.

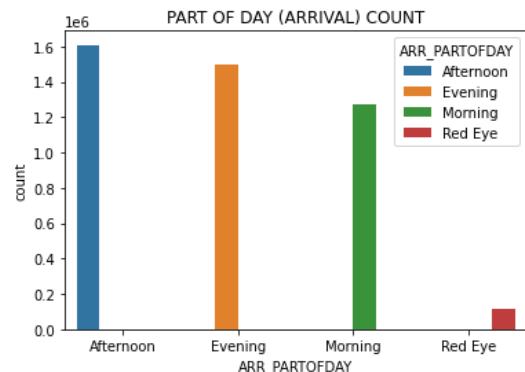


Image31: Count of arrivals per time of the day



## Predicting Delays inFlight

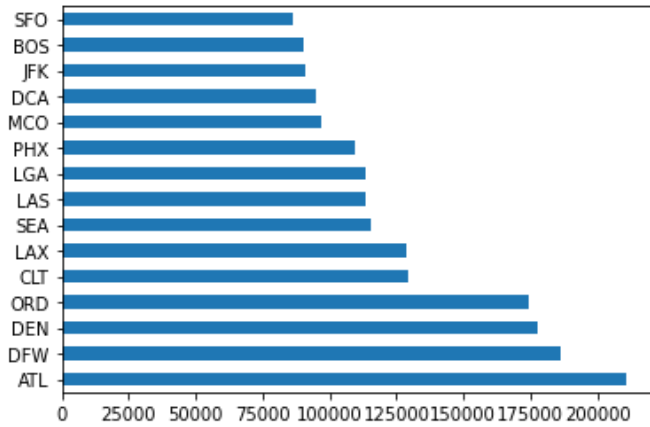


Image32: Average flights from origin airport

We look at the origin airports. ATL (Hartsfield-Jackson Atlanta) beat others by crossing over 200,000. O'Hare International, Denver International, Dallas/Fort Worth International being in the top 5. While on the other hand San Francisco international being the lowest of all.

To evaluate the destination places below the figures almost seems like the arrival. Which means that there were almost the same number of landings as takeoffs.

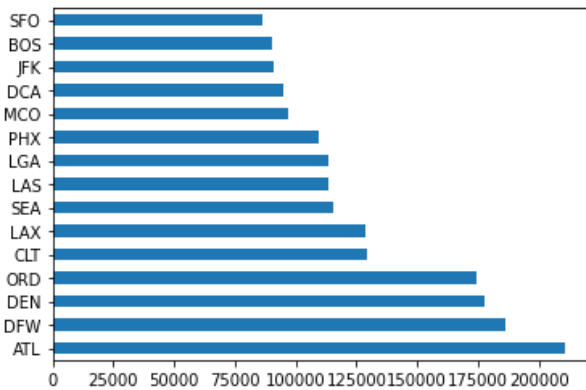


Image33: Average flights to destination airport

Now we shall look at the flight carriers who have helped people reach their destination. Southwest was overwhelming as it passed 800,000. The second highest being Delta followed by American. SkyWest and United kept their spot in the top 5.

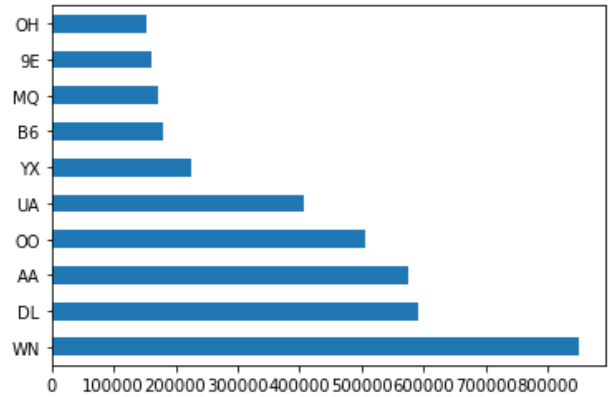


Image34: Average flights carriers to and from origin/destination airport

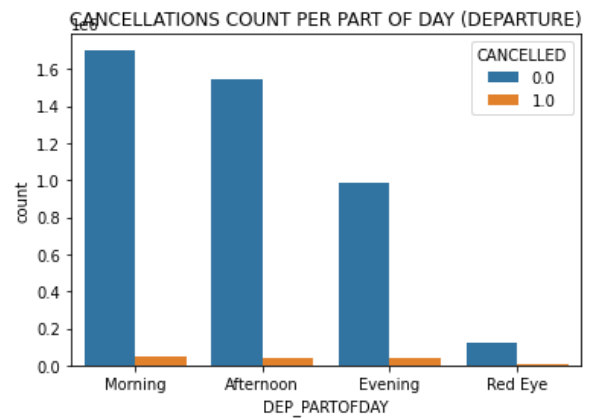


Image35: Average departure flights canceled per part of the day

The next thing that I was curious about was the time when the flights were getting canceled the most. Well, to my curiosity it shows that morning had topped, and afternoon and evenings gave constant results per departure.

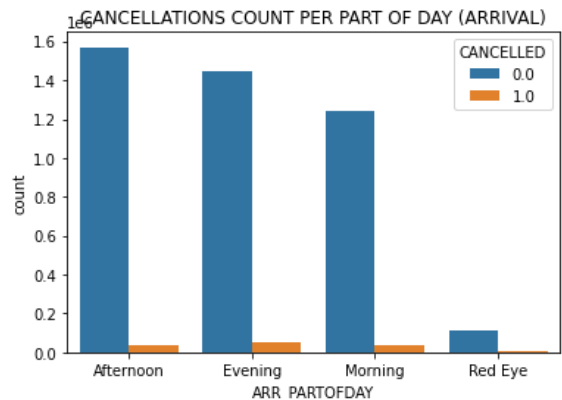


Image36: Average arrival flights canceled per part of the day

Per arrivals it says that the evenings get canceled the most. Afternoon, morning, red eye remains similar to departure. The

## Predicting Delays inFlight

same way let us take a look at the delays.

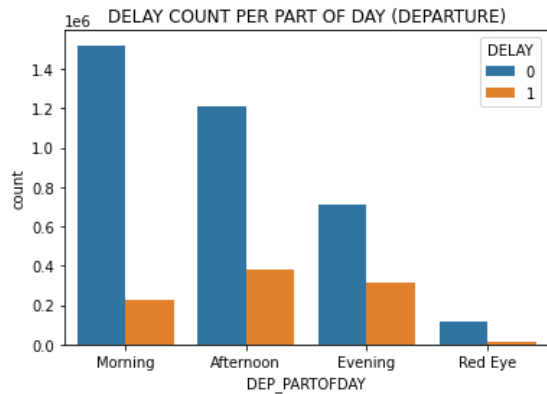


Image37: Average departure flights delayed per part of the day

On the other hand, delays seem to be more than cancellation. Afternoon hit 0.4 marking a highest record of delays, followed by Evening and mornings. As compared to on time delays are still low per departure.

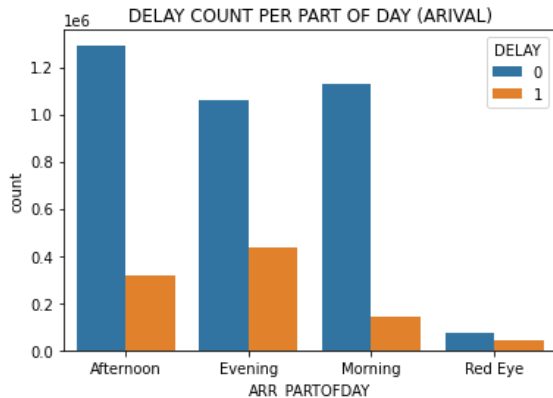


Image38: Average arrival flights delayed per part of the day

Evenings score more than afternoons in departure delays. Afternoons begin second and morning at third. Surprisingly there are almost 50% delays during red eye time.

Below is the list of origin airports with the count of delays.

Analyzing Flight Delays

```
In [92]: airline22DFPandasD1=airline22DFPandas.groupby(["ORIGIN", "DELAY"])
["DELAY"].count().reset_index(name="COUNT")
# Delay or Not Delay count from each origin
airline22DFPandasD1 = airline22DFPandasD1
sort_values(by = ['DELAY', 'COUNT'], ascending = [False, False])
airline22DFPandasD1
```

```
Out[92]:
```

ORIGIN	DELAY	COUNT
192	DEN	1 50795
45	ATL	1 42137
194	DFW	1 39371
506	ORD	1 34870
384	LAS	1 30403
...	...	...
343	ILG	0 26
550	PPG	0 19

Image39: Analyzing the delays in flights

```
In [94]: airline22DFPandasD10 = airline22DFPandasD1.
loc[airline22DFPandasD1['DELAY'] == 0]
# Origin with Most number of flights without delay
airline22DFPandasD10
```

```
Out[94]:
```

	ORIGIN	DELAY	COUNT
44	ATL	0	168989
193	DFW	0	146782
505	ORD	0	139762
191	DEN	0	127051
387	LAX	0	106081
...	...	...	...
343	ILG	0	26
550	PPG	0	19
491	OGD	0	14
197	DIK	0	8
543	PIR	0	3

369 rows x 3 columns

Image40: Pictorial presentation of non-delayed flights

The list of top airports who had more on time flights. Atlanta, Dallas/Fort Worth International, O'Hare International, Denver International, Los Angeles International are the top 5 airports from above.

```
In [95]: airline22DFPandasD11Top10=airline22DFPandasD11.head(10)
airline22DFPandasD10Top10= airline22DFPandasD10.head(10)
```

```
In [96]: airline22DFPandasD2=airline22DFPandas.groupby(
["OP_CARRIER", "DELAY"])[["DELAY"].count().
reset_index(name="COUNT")
# Delay or Not Delay count from each Flight Carrier
airline22DFPandasD2 = airline22DFPandasD2.sort_values
(by = ['DELAY', 'COUNT'], ascending = [False, False])
airline22DFPandasD2
```

```
Out[96]:
```

	OP_CARRIER	DELAY	COUNT
29	WN	1	241495
3	AA	1	120292
9	DL	1	100504
27	UA	1	82610
23	OO	1	77018
7	B6	1	57492
33	YX	1	38180
19	NK	1	34241
11	F9	1	29499
21	OH	1	28644
17	MQ	1	25268
13	G4	1	24455
5	AS	1	23908
1	9E	1	22418
31	YV	1	15247
25	QX	1	8762
15	HA	1	7773
28	WN	0	609319

Image41: graph depicting carrier being delayed or not

## Predicting Delays inFlight

Southwest is the carrier that had the highest number of delays this year. American, Delta, United, and SkyWest secured the top 5 amongst all.

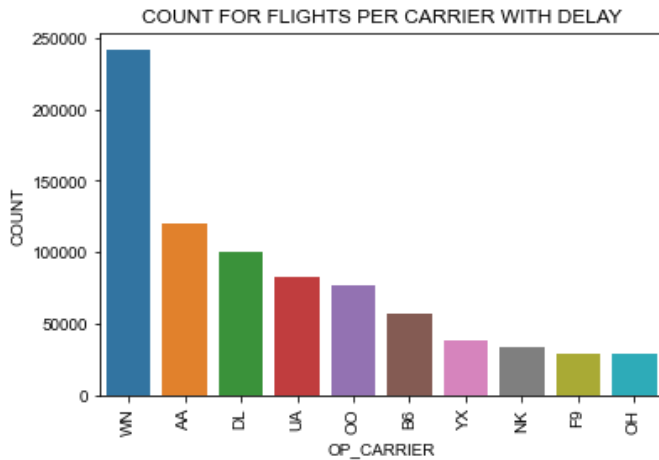


Image42: Count of flights per carrier per delay

Count of delays per carrier with Southwest is approximately 245,000. American more than 100,000, Delta at 100,000. Frontier, Blue Streak being the least in the 10.

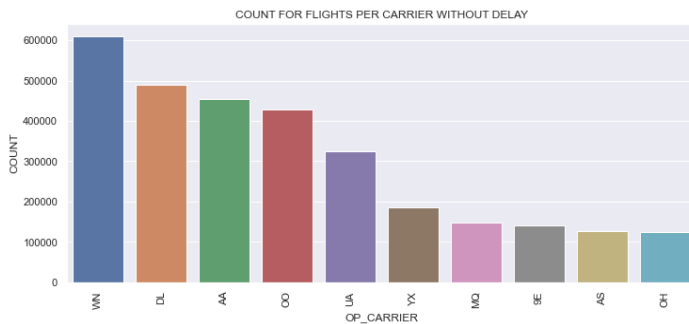


Image43: Count of flights per carrier without delay

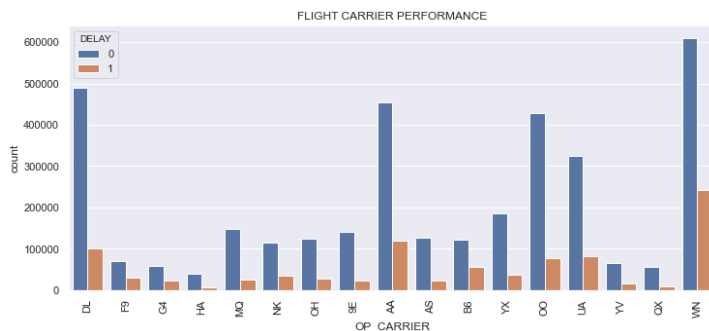


Image44: Performance of flights carrier

The above gives us a presentation about each carrier with their delays and non-delayed flights. Overall Southwest rules the market followed by Delta, American, SkyWest, and United.

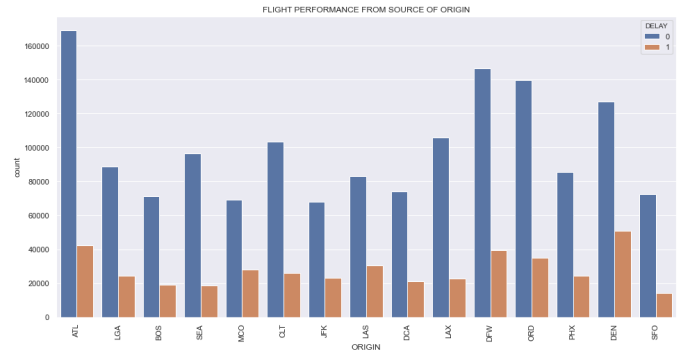


Image45: Flights performance from source to destination

Now we see the performance of resources from Origin airport. San Francisco has the lowest number of delayed flights. Seattle-Tacoma, Boston has the lowest record of delays.

## V.II. SQL Postgresql Pgadmin:

Here are the results from analysis done in pgadmin:

Query1:

```
30 SELECT COUNT(*) FROM AirlineData;
31 -- Count of total number of records
```

	count	bigint
1	4495839	

Image46: Flights performance from source to destination

We can see the count of records as 4,495,839, which is correct as per the count of records checked in pyspark earlier.

Query 2:

```
SELECT DISTINCT "MONTH" FROM AirlineData;
-- Show distinct months
```

	month	integer
1	8	
2	7	
3	1	
4	5	
5	2	
6	4	
7	6	
8	3	

Image47: Distinct months

The above query tells about the unique months available in the dataset, as the period ranges from January till August, the numerical value ranges from 1 to 8.

Query 3:

```
SELECT "CANCELLED", COUNT("CANCELLED") FROM AirlineData
GROUP BY "CANCELLED" ORDER BY COUNT("CANCELLED") DESC;
-- Count of cancelations
```

## Predicting Delays inFlight

	Data Output	Explain	Messages	Notifications
	cancelled numeric	count bigint		
1	0.0	4364855		
2	1.0	130984		

Image48: Count of cancellations

The above query tells about the total count of cancellations that is 130,984 from January 2022 till August 2022.

Query 4:

```
SELECT "ORIGIN", COUNT("ORIGIN") FROM AirlineData
GROUP BY "ORIGIN" ORDER BY COUNT("ORIGIN") DESC
LIMIT 20;
-- Count of flights from each origin airport
```

	Data Output	Explain	Messages
	origin character varying	count bigint	
1	ATL	211126	
2	DFW	186153	
3	DEN	177846	
4	ORD	174632	
5	CLT	129370	
6	LAX	128733	
7	SEA	115227	
8	LAS	113490	
9	LGA	113219	
10	PHX	109608	
11	MCO	97079	
12	DCA	94913	
13	JFK	90959	
14	BOS	90122	
15	SFO	86589	
16	EWB	86395	
17	DTW	86160	
18	IAH	83758	
19	MSP	81448	
20	MIA	74568	

Image49: Count of Flights from top 20 airports with highest count

The above query gives insights on the top 20 busiest airports in terms of the frequency of the flight. ATL, DFW and DEN are the top 3 airports.

Query 5:

```
SELECT "OP_CARRIER", "CANCELLED", COUNT("CANCELLED") FROM AirlineData
GROUP BY "OP_CARRIER", "CANCELLED" ORDER BY "CANCELLED" DESC,
COUNT("CANCELLED") DESC
LIMIT 40;
-- Count of canceled and departed flights per each carrier
```

	op_carrier character varying	cancelled numeric	count bigint
AA		1.0	22038
WN		1.0	21321
YX		1.0	13311
DL		1.0	10913
OO		1.0	10255
UA		1.0	9264
B6		1.0	8182
9E		1.0	6838
OH		1.0	5761
MQ		1.0	4493

NK	1.0	4374
AS	1.0	3872
YV	1.0	3383
G4	1.0	2994
F9	1.0	2627
QX	1.0	998
HA	1.0	360

	op_carrier character varying	cancelled numeric	count bigint
WN		0.0	829493
DL		0.0	579874
AA		0.0	552242
OO		0.0	494275
UA		0.0	397663
YX		0.0	211249
B6		0.0	171711
MQ		0.0	168158
9E		0.0	155601
OH		0.0	147516
AS		0.0	146152
NK		0.0	144415
F9		0.0	97267
G4		0.0	79792

Image50: Count of canceled Flights from each carrier

It is surprising to observe that the highest number of flights canceled are by American, Southwest, which fall under the top 4 in the world. This is then followed by the Republic with 13311 flights getting canceled. The next two are close enough to each other with a minute difference of 700 roughly. It was not surprising to see F9 (Frontier), NK (Spirit) in the list.

Query 6:

```
SELECT "ORIGIN", "CANCELLED", COUNT("CANCELLED") FROM AirlineData
GROUP BY "ORIGIN", "CANCELLED" ORDER BY "CANCELLED" DESC, COUNT("CANCELLED") DESC
LIMIT 100;
-- Count of canceled and departed flights from the origin airport
```

	origin character varying	cancelled numeric	count bigint
LGA		1.0	7367
DFW		1.0	6662
ORD		1.0	5795
EWB		1.0	5410
DCA		1.0	4722
DEN		1.0	4575
CLT		1.0	4242
JFK		1.0	4216
BOS		1.0	3803
ATL		1.0	3661
MCO		1.0	2612
MIA		1.0	2401
LAS		1.0	2310
LAX		1.0	2098

Image51: Count of canceled Flights from each airport

After finding out the airline that had called off their flights, I also wanted to find the airport of origin along with the count.

## Predicting Delays inFlight

The above query helps us find the same with the below output. LaGuardia from NY has the highest number of carriers getting canceled which is followed by Dallas Fort Worth international. O'Hare, Newark Liberty followed closely. Whereas Ronald Reagan Washington, Denver international, Charlotte Douglas, John F. Kennedy was in the top 10. As per [11] it is stated that LGA being the smallest airport and ranging in radius of 1500 miles to its destinations has become the most used airport. Due to the urge of its utility there was a new terminal opened in 2021, there seems to be continuing flight delays. There was another facility Terminal C replacing the old ones.

### Query 7:

```
SELECT
  COUNT(*) filter (where CAST("DEP_DELAY_NEW" AS NUMERIC) < 15) as NOT_DELAYED,
  COUNT(*) filter (where CAST("DEP_DELAY_NEW" AS NUMERIC) >= 15) as DELAYED
FROM AirlineData;
-- Count of Delayed instances
```

Data Output	Explain	Messages	Notifications
not_delayed bigint	delayed bigint		
1	3398547	969441	

Image52: Count of delayed flights

We can confirm that the number of delayed flights (delay greater than 15 minutes) are 969,441 (almost a million flights). That means approximately 1 out of 4 flights were delayed.

### Query 8:

```
SELECT OP_CARRIER,
  COUNT(*) filter (where CAST("DEP_DELAY_NEW" AS NUMERIC) < 15) as NOT_DELAYED,
  COUNT(*) filter (where CAST("DEP_DELAY_NEW" AS NUMERIC) >= 15) as DELAYED
FROM AirlineData GROUP BY OP_CARRIER ORDER BY DELAYED DESC; |
-- Count of Delayed instances for each carrier
```

Data Output	Explain	Messages	Notifications
op_carrier character varying	not_delayed bigint	delayed bigint	
1 WN	578624	251045	
2 AA	429051	123817	
3 DL	475863	104242	
4 UA	312323	85561	
5 OO	415113	79353	
6 B6	113087	58842	
7 YX	172531	39160	
8 NK	109231	35297	
9 F9	67088	30267	
10 OH	118341	29385	
11 MQ	142158	26166	
12 G4	54609	25208	
13 AS	121356	24917	
14 9E	132720	23064	
15 YV	61907	15661	
16 QX	54501	9132	
17 HA	40044	8324	

Image53: Count of delayed flights per carrier

We can see what flights have been delayed more. Again in terms of most count it is Southwest and American that lead the way.

## V.III. SQL Pyspark:

Another SQL utilized, that was with the builtin libraries of pyspark ml. The below are findings that we could utilize to validate that the .csv file was exported appropriately.

### Query 1:

```
Q1 = spark.sql("SELECT CANCELLED, COUNT(CANCELLED) FROM airline22DFPar
Q1.show()
```

CANCELLED	count(CANCELLED)
0.0	4364855
1.0	130984

Image54: EDA using SQL

Using spark.sql the above query was executed, the number of records in the dataset displaying the total number of canceled flights is 130984.

### Query 2:

```
Q2 = spark.sql("SELECT ORIGIN, COUNT(ORIGIN) FROM airline22DFPar
Q2.show()
```

ORIGIN	count(ORIGIN)
ATL	211126
DFW	186153
DEN	177846
ORD	174632
CLT	129370
LAX	128733
SEA	115227
LAS	113490
LGA	113219
PHX	109608
MCO	97079
DCA	94913
JFK	90959
BOS	90122
SFO	86589
EWK	86395
DTW	86160
IAH	83758
MSP	81448
MLA	74568

only showing top 20 rows

Image55: Query to retrieve top origin airports

To get the list of top 20 origin airports with the count of the flights taken off from them and to validate the results with the SQL results from the previous subsection.

From the results available in these couple of queries along with the results available the queries ran in Postgresql we could confirm that the export into a single .csv file went smoothly.

## V.IV. Pyspark:

Performing actions using Pyspark, we figure out the count of delayed flights. As per this there are 969441 rows which have delayed flights.

```
In [106]: airline22DFNew = airline22DF.
withColumn('TargetDel', (airline22DF.DEP_DELAY_NEW >= 15).
cast('integer'))
```

```
In [107]: airline22DFNew.groupby("TargetDel").count().show()
```

TargetDel	count
null	127851
1	969441
0	3398547

```
In [108]: airline22DFNew1 = airline22DFNew
airline22DFNew10 = airline22DFNew1.filter("CANCELLED == 0")
airline22DFNew11 = airline22DFNew1.filter("CANCELLED == 1")
```

```
In [109]: airline22DFNew10.groupby("TargetDel").count().show()
```

TargetDel	count
1	967677
0	3397178

Image56: finding delays using pyspark



## Predicting Delays inFlight

```
In [110]: airline22DFNew11.groupby("TargetDel").count().show()
```

TargetDel	count
null	127851
1	1764
0	1369

Image57: understanding the 0,1,null in dataset

## V.V. Machine Learning:

The model analysis has been divided into two segments, the first one contains results of models that were built on a feature set containing airport code, the second one contains results of models that were built on a feature set containing state.

### 1. For Features including Airport:

#### 1.1. Logistic Regression:

From the above confusion matrix, we can find the following:

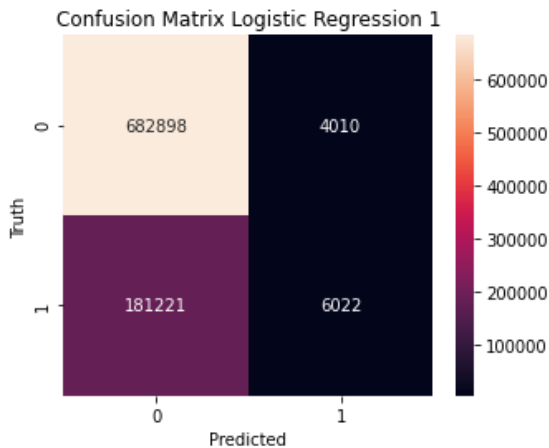


Image58: Logistic Regression 1

Precision: 0.6003

Recall: 0.0322

The model was able to predict the non delay instances, however it wasn't able to predict the delay instances that well.

#### 1.2. Random Forest:

From the above confusion matrix, we can find the following:

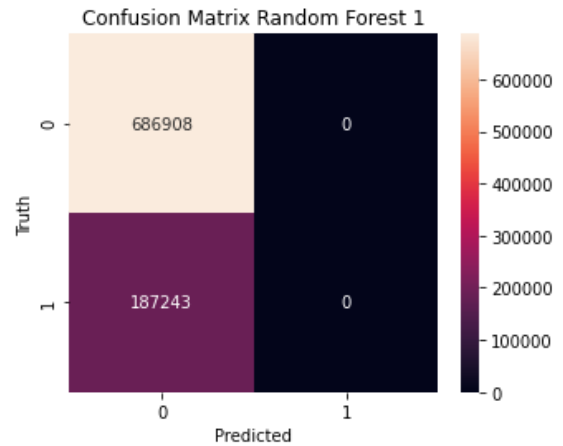


Image59: Random Forest 1

Precision: 0.00

Recall: 0.0000

The model was able to predict the non delay instances, however it wasn't able to predict the instances of delay at all. All the instances that were predicted came out as non delayed instances. Hence the Random Forest model has failed to predict delay in flights when the feature set contains airports.

#### 1.3. Gradient Boosted Tree:

From the above confusion matrix, we can find the following:

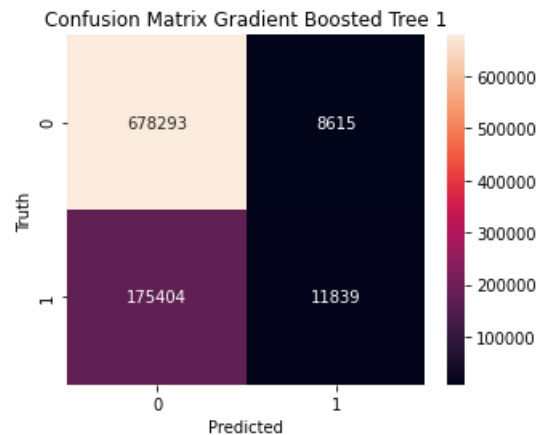


Image60: Gradient Boosted Tree 1

Precision: 0.5788

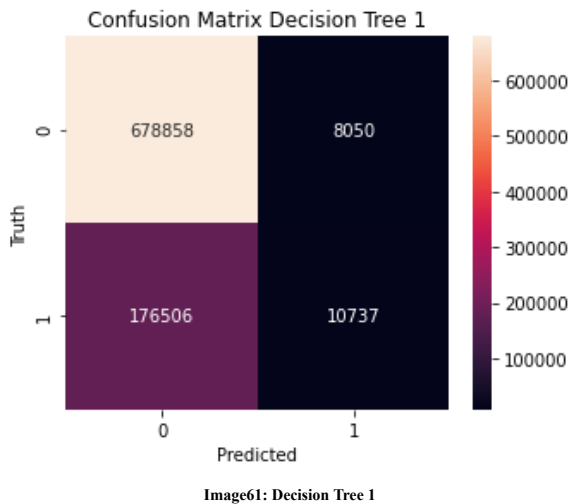
Recall: 0.0632

The gradient boosted tree did perform better than the previous 2 with the highest accuracy. Also, in comparison it was able to predict the instances of delay better than the logistic model. However, the percentage of correctly identified delayed instances is still low.

#### 1.4. Decision Tree:

## Predicting Delays inFlight

From the above confusion matrix, we can find the following:



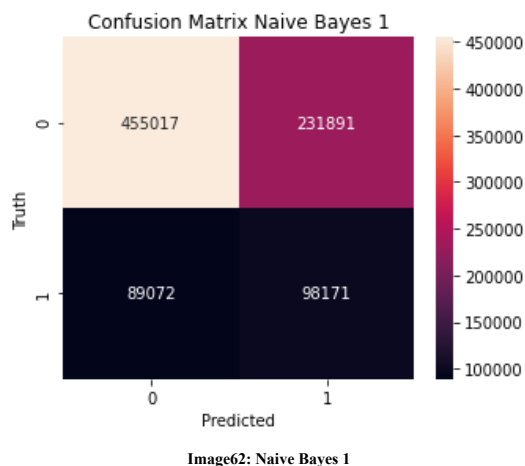
Precision: 0.5715

Recall: 0.0573

Decision tree had a comparable performance to the gradient boosted tree. It predicted slightly less number of delayed instances than the gradient boosted tree, however still performed better than the logistic and random forest.

### 1.5. Naive Bayes:

From the above confusion matrix, we can find the following:



Precision: 0.2974

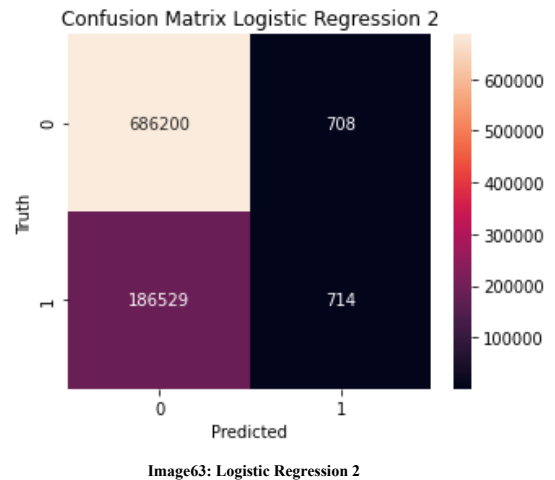
Recall: 0.5243

The Naive Bayes in comparison predicted more delays than any other model, however the accuracy was lower in comparison. The model also categorized more non delayed instances as delayed instances. So in terms of predicting delays, this model performed better.

## 2. For Features including State:

### 2.1. Logistic Regression:

From the above confusion matrix, we can find the following:



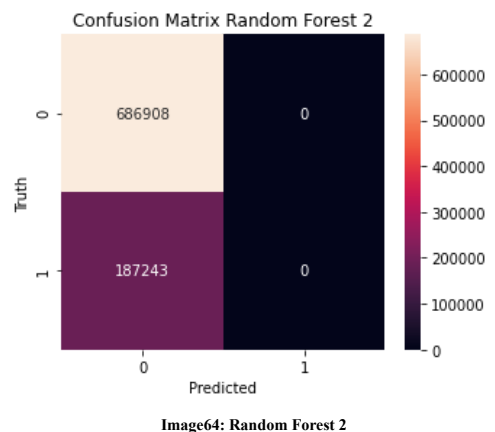
Precision: 0.5021

Recall: 0.0038

The model was able to predict the non delay instances quite well, however it wasn't able to predict the instances of delay that well. The performance of the model worsened once the state was part of the feature list instead of the airport.

### 2.2. Random Forest:

From the above confusion matrix, we can find the following:



Precision: 0

Recall: 0

The model was able to predict the non delay instances, however it wasn't able to predict the instances of delay at all. All the instances that were predicted came out as non delayed instances. Hence the Random Forest model has failed to predict

## Predicting Delays inFlight

delay in flights even when the feature set contains state instead of airport.

### 2.3. Gradient Boosted Tree:

From the above confusion matrix, we can find the following:

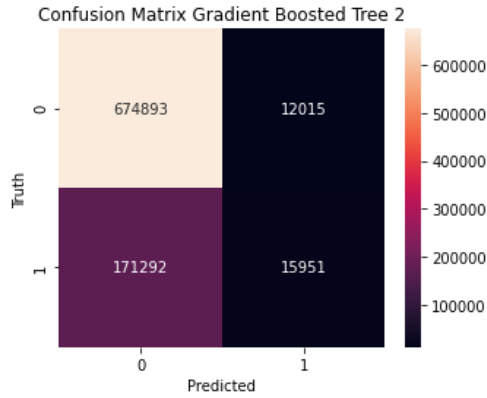


Image65: Gradient Boosted Tree 2

Precision: 0.5704

Recall: 0.0852

The gradient boosted tree did perform better than the previous 2 with the highest accuracy. Also, in comparison it was able to predict the instances of delay better than the logistic model. However, the percentage of correctly identified delayed instances is still low. The model improved when the feature set contained state instead of airport in the feature list.

### 2.4. Decision Tree:

From the above confusion matrix, we can find the following:

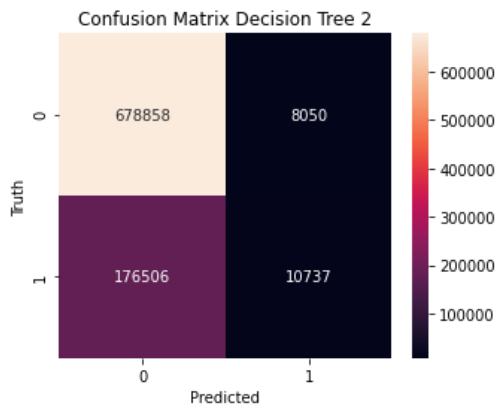


Image66: Decision Tree 2

Precision: 0.5715

Recall: 0.0573

Decision tree had a comparable performance to the gradient boosted tree. It predicted slightly less number of delayed instances than the gradient boosted tree, however still performed better than the logistic and random forest.

### 2.5. Naive Bayes:

From the above confusion matrix, we can find the following:

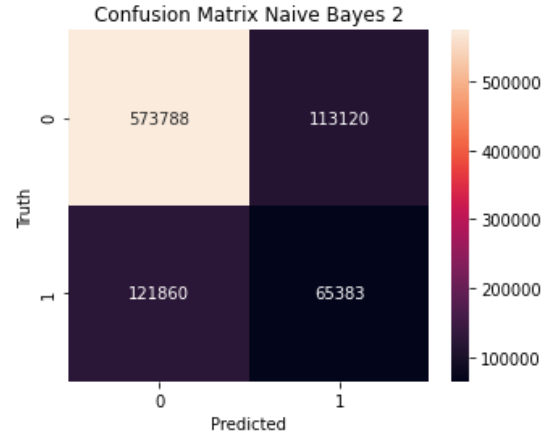


Image67: Naive Bayes 2

Precision: 0.3663

Recall: 0.3492

The Naive Bayes in comparison predicted more delays than any other model, but the accuracy was lower in comparison. The model also categorized more non delayed instances as delayed instances.

So in terms of predicting delays, this model performed better. Another point to note is that though the accuracy of the model improved with the presence of state in the feature list (instead of the airport) the correctly predicted instances of the delay lessened.

### Tabular Statistics:

VAM1 is for the feature set containing airport codes and VAM2 is for the feature set containing state codes. Here is the tabular statistics:

Feature	Parameter	Logistic	Random Forest	Gradient Boosted Tree	Decision Tree	Naive Bayes
VAM1	Accuracy	0.7881	0.7858	<b>0.7895</b>	0.7889	0.6328
	Precision	0.6003	0	0.5788	0.5715	0.2974
	Recall	0.0322	0	0.0632	0.0573	<b>0.5243</b>
VAM2	Accuracy	0.7858	0.7858	<b>0.7903</b>	0.7890	0.7312
	Precision	0.5021	0	0.5704	0.5715	0.3663
	Recall	0.0038	0	0.0852	0.0573	<b>0.3492</b>

Table2: Tabular Statistics

## VI. DISCUSSION

From the exploratory data analysis, we can say that there are a few airports and carriers which have a higher tendency for delays. The highest number of flights canceled are by American

and Southwest. LaGuardia from NY has the highest number of carriers getting canceled which is followed by Dallas Fort Worth international. O'Hare, Newark Liberty followed closely. Flights originating in the afternoon see more delay. All the models had accuracy ranging in the 70s except for the Naive Bayes in one case where the accuracy had dipped to 63.28% with the feature set that included individual airports instead of the state. If we consider the model to accurately classify each instance, then we would consider the Gradient Boosted Tree as the best performing model especially with the feature set containing state. However, another key aspect of the project was to classify the delayed instances better so as to inform potential customers. In that case, we would consider the Recall, that would mean the Naive Bayes performed better as it was better able to predict the instances of the delay.

### VII. LIMITATION & FUTURE WORK

One of the major limitations for the project was the inability of the Random Forest to predict any delayed instances and also the other models inability to predict significant instances of the delays. Due to the limitations of the virtual machine being utilized (free tier) working on bigger datasets for better machine learning would have drained out the resources. The error out of heap memory would have persisted with bigger datasets. So for future work, the dataset in consideration should be expanded to around 4 years as it would then contain data starting from the year 2019, which was the golden age of aviation as there were a lot of travelers. Additionally with that dataset, one can also find interesting insights from the Pandemic period and how the aviation sector suffered in terms of both delays and cancellations.

### VIII. REFERENCES

- [1] *Flight delay information - air traffic control system command center*. [Online]. Available: <https://testfly.faa.gov/flyfaa/usmap.jsp>. [Accessed: 20-Nov-2022].
- [2] "Industry market research, reports, and Statistics," *IBISWorld*. [Online]. Available: <https://www.ibisworld.com/industry-statistics/market-size/domestic-airlines-united-states/#>. [Accessed: 01-Dec-2022].
- [3] K. Adams, "2022 has brought more air travel delays and cancellations - and nearly double the risk of having a bag mishandled," *ValuePenguin*, 22-Aug-2022. [Online]. Available: <https://www.valuepenguin.com/travel/delays-cancellations-bags-study>. [Accessed: 01-Dec-2022].
- [4] "Confusion matrix online calculator," *Confusion Matrix - Online Calculator*. [Online]. Available: <https://onlineconfusionmatrix.com/>. [Accessed: 5-Dec-2022].
- [5] "Download Page - Bureau of Transportation Statistics," *Bureau of Transportation Statistics*. [Online]. Available: [https://transtats.bts.gov/DL\\_SelectFields.aspx?gnoyr\\_VQ=F&GJ&QO\\_fu146\\_anzr=b0-gvzr](https://transtats.bts.gov/DL_SelectFields.aspx?gnoyr_VQ=F&GJ&QO_fu146_anzr=b0-gvzr). [Accessed: 20-Nov-2022].
- [6] "Logistic regression," *Logistic Regression - an overview | ScienceDirect Topics*. [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/logistic-regression>. [Accessed: 10-Dec-2022].
- [7] R. Kraan, "Demystifying decision trees, Random Forests & Gradient Boosting," *Medium*, 05-May-2020. [Online]. Available: <https://towardsdatascience.com/demystifying-decision-trees-random-forests-gradient-boosting-20415b0a406f>. [Accessed: 25-Nov-2022].
- [8] R. Gandhi, "Naive Bayes classifier," *Medium*, 17-May-2018. [Online]. Available: <https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>. [Accessed: 25-Nov-2022].
- [9] L. Arora, "Building Machine Learning Pipelines using Pyspark," *Analytics Vidhya*, 22-Apr-2020. [Online]. Available: <https://www.analyticsvidhya.com/blog/2019/11/build-machine-learning-pipelines-pyspark/>. [Accessed: 25-Nov-2022].
- [10] D. Rai, "Feature engineering in pyspark-part I," *Medium*, 30-Oct-2018. [Online]. Available: <https://dhiraj-p-rai.medium.com/essentials-of-feature-engineering-in-pyspark-part-i-76a57680a85>. [Accessed: 25-Nov-2022].
- [11] C. to W. projects, "Laguardia Airport – travel guide at Wikivoyage," *Wikivoyage*, 16-Nov-2022. [Online]. Available: [https://en.wikivoyage.org/wiki/LaGuardia\\_Airport#](https://en.wikivoyage.org/wiki/LaGuardia_Airport#). [Accessed: 07-Dec-2022].