Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J COMPONENT REPORT

Programme : M.TECH(INTG) CSE & BA

Course Title : BIG DATA FRAMEWORKS

Course Code : CSE3120

Slot : F1

Title: FLIGHT DELAY PREDICTION

Team Members: Harini Gokulram Naidu | 19MIA1004

Shivani Gokulram Naidu | 19MIA1006

P Subhashri | 19MIA1008

Deekshitha L | 19MIA1030

Faculty: G. SUGANESHWARI

Sign:

Date:

DECLARATION

I hereby declare that the project entitled **"FLIGHT DELAY PREDICTION"** submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai 600127 in partial fulfilment of the requirements for the award of the degree of M.Tech (Integrated) Business Analytics – **Computer Science and Engineering** is a record of bonafide work carried out by me**.** I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature

TABLE OF CONTENTS

ABSTRACT

Nowadays, the aviation industry plays a crucial role in the world's transportation sector, and a lot of businesses rely on various airlines to connect them with other parts of the world. But extreme weather conditions may directly affect the airline services by means of flight delays.

To solve this issue, accurately predicting these flight delays allows passengers to be well prepared for the deterrent caused to their journey and enables airlines to respond to the potential causes of the flight delays in advance to diminish the negative impact.

The purpose of this project is to look at the approaches used to build models for predicting flight delays that occur due to bad weather conditions in pyspark.

We have also compared the execution times of the models in both pyspark and python

INTRODUCTION

In the present world, the major components of any transportation system include passenger airline, cargo airline, and air traffic control system. With the passage of time, nations around the world have tried to evolve numerous techniques of improving the airline transportation system.

This has brought drastic change in the airline operations. Flight delays occasionally cause inconvenience to the modern passengers. Every year approximately 20% of airline flights are cancelled or delayed, costing passengers more than 20 billion dollars in money and their time.

Average aircraft delay is regularly referred to as an indication of airport capacity. Flight delay is a prevailing problem in this world. It's very tough to explain the reason for a delay. A few factors responsible for the flight delays like runway construction to excessive traffic are rare, but bad weather seems to be a common cause.

Some flights are delayed because of the reactionary delays, due to the late arrival of the previous flight. It hurts airports, airlines, and affects a company's marketing strategies as companies rely on customer loyalty to support their frequent flying programs.

PROBLEM STATEMENT

Nowadays the phenomenon of flight delays and cancellations is becoming more and more serious. Flight delays and cancellations not only waste transportation resources, but also affect passengers travel plans, which cause increase in passenger discontent and complaint rates. The passengers’ dissatisfaction and distrust of airlines seriously damage the airlines' corporate reputation and then affect passengers’ loyalty

LITERATURE SURVEY

The main concern of the researchers and analysts is to predict the reasons for flight delays and for that they have put in their efforts on collecting data about flight and the weather. Mohamed et al. have studied the pattern of arrival delay for non-stop domestic flights at the Orlando International Airport. They focused primarily on the cyclic variations that happen in the air travel demand and the weather at that particular airport.

In Shervin et al.'s work , their motive of research is to propose an approach that improves the operational performance without hampering or effecting the planned cost.

Adrian et al. have created a data mining model which enables the flight delays by observing the weather conditions. They have used WEKA and R to build their models by selecting different classifiers and choosing the one with the best results. They have used different machine learning techniques like Naïve Bayes and Linear Discriminant Analysis classifier.

Choi et al. have focused on overcoming the effects of the data imbalancing caused during data training. They have used techniques like Decision Trees, AdaBoost, and K-Nearest Neighbors for predicting individual flight delays. A binary classification was performed by the model to predict the scheduled flight delay.

DATASET DESCRIPTION

The Dataset is downloaded from Kaggle. The dataset that we have used in this process basically has 3 csv files. They are airlines, airports and flights. The airlines csv file consists of 2 attributes, the Airline names and its corresponding ID. The airports csv file consists of 7 attributes, the Airport name, ID, City, State, Country, Latitude and Longitude. The flights csv file consists of the Year, Month, Day, Day of week, Airline, Flight Number, Tail number ,Origin airport, Destination airport, Scheduled departure.

PROPOSED METHODOLOGY

LOADING THE DATA:

We are loading all three csv files to our environment by using the pd.read_csv command.

CREATING A TEMP:

Usually a SQL query (using the .sql() method) that references the DataFrame will throw an error. To access the data in this way, we have to save it as a temporary table.

We can do this using the .createTempView() Spark DataFrame method, which takes as its only argument the name of the temporary table we'd like to register. This method registers the

DataFrame as a table in the catalog, but as this table is temporary, it can only be accessed from the specific SparkSession used to create the Spark DataFrame.

DROPPING THE MIDDLE MAN

Our SparkSession has a .read attribute which has several methods for reading different data sources into Spark DataFrames. Using these we can create a DataFrame from a .csv file just like with regular pandas DataFrames

The variable file_path is a string with the path to the file airports.csv. This file contains information about different airports all over the world.

CREATING DATAFRAME / COLUMNS

Now to perform column-wise operations we can use the .withColumn() method, which takes two arguments. First, a string with the name of our new column, and second the new column itself. The new column must be an object of class Column. Creating one of these is as easy as extracting a column from our DataFrame using df.colName.

Thus, all these methods return a new DataFrame. To overwrite the original DataFrame we must reassign the returned DataFrame using the method like so:

df = df.withColumn("newCol", df.oldCol + 1)

The above code creates a DataFrame with the same columns as df plus a new column, newCol, where every entry is equal to the corresponding entry from oldCol, plus one.

DATA TYPES

We can see that some of the columns in our DataFrame are strings containing numbers as opposed to actual numeric values.To remedy this, we can use the .cast() method in combination with the .withColumn() method. It's important to note that .cast() works on columns, while .withColumn() works on DataFrames.The only argument we need to pass to .cast() is the kind of value we want to create, in string form. For example, to create integers, you'll pass the argument "integer" and for decimal numbers you'll use "double".

AGGREGATING

All of the common aggregation methods, like .min(), .max(), and .count() are GroupedData methods and all of these are applied to our dataframe. These are created by calling the .groupBy() DataFrame method.

In addition to the GroupedData methods, there is also the .agg() method. This method lets us pass an aggregate column expression that uses any of the aggregate functions from the pyspark.sql.functions submodule.

JOINING

A join will combine two different tables along a column that they share. This column is called the key. Examples of keys here include the tailnum and airline columns from the flights table.

Supposedly we want to know more information about the plane that flew a flight than just the tail number. This information isn't in the flights table because the same plane flies many different flights over the course of two years, so including this information in every row would result in a lot of duplication.

To avoid this, we'd have a second table that has only one row for each plane and whose columns list all the information about the plane, including its tail number. We could call this table planes

When we join the flights table to this table of airplane information, we're adding all the columns from the planes table to the flights table. To fill these columns with information, we'll look at the tail number from the flights table and find the matching one in the planes table, and then use that row to fill out all the new columns.

MACHINE LEARNING

At the core of the pyspark.ml module are the Transformer and Estimator classes. Almost every other class in the module behaves similarly to these two basic classes.

Transformer classes have a .transform() method that takes a DataFrame and returns a new DataFrame; usually the original one with a new column appended.

Estimator classes all implement a .fit() method. These methods also take a DataFrame, but instead of returning another DataFrame they return a model object. This can be something like a StringIndexerModel for including categorical data saved as strings in our models.

STRINGS AND FACTORS

The first step to encoding our categorical feature is to create a StringIndexer. Members of this class are Estimators that take a DataFrame with a column of strings and map each unique string to a number. Then, the Estimator returns a Transformer that takes a DataFrame, attaches the mapping to it as metadata, and returns a new DataFrame with a numeric column corresponding to the string column.

The second step is to encode this numeric column as a one-hot vector using a OneHotEncoder. This works exactly the same way as the StringIndexer by creating an Estimator and then a Transformer. The end result is a column that encodes our categorical feature as a vector that's suitable for machine learning routines

ASSEMBLE A VECTOR

The last step in the Pipeline is to combine all of the columns containing our features into a single column. This has to be done before modeling can take place because every Spark modeling routine expects the data to be in this form. You can do this by storing each of the values from a column as an entry in a vector. Then, from the model's point of view, every observation is a vector that contains all of the information about it and a label that tells the modeler what value that observation corresponds to.

Because of this, the pyspark.ml.feature submodule contains a class called VectorAssembler. This Transformer takes all of the columns we specify and combines them into a new vector column.

CREATE THE PIPELINE

Pipeline is a class in the pyspark.ml module that combines all the Estimators and Transformers that we've already created. This lets us reuse the same modeling process over and over again by wrapping it up in one simple object

ML MODELS:

LOGISTIC REGRESSION

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable.

LINEAR SVC

The Linear Support Vector Classifier (SVC) method applies a linear kernel function to perform classification and it performs well with a large number of samples. If we compare it with the SVC model, the Linear SVC has additional parameters such as penalty normalization which applies 'L1' or 'L2' and loss function.

RANDOM FOREST CLASSIFIER

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

DECISION TREE CLASSIFIER

The decision tree classifier creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.

IMPLEMENTATION

We have carried out both in PySpark and Python.

1. PYSPARK :

Importing all the 3 csv files

In [ ]:
```
import numpy as np
import pandas as pd

airlines = pd.read_csv("/content/drive/MyDrive/BDF J COMP/airlines.csv")
airlines
```

Out[ ]:

	IATA_CODE	AIRLINE
0	UA	United Air Lines Inc.
1	AA	American Airlines Inc.
2	US	US Airways Inc.
3	F9	Frontier Airlines Inc.
4	B6	JetBlue Airways
5	OO	Skywest Airlines Inc.
6	AS	Alaska Airlines Inc.
7	NK	Spirit Air Lines
8	WN	Southwest Airlines Co.
9	DL	Delta Air Lines Inc.
10	EV	Atlantic Southeast Airlines
11	HA	Hawaiian Airlines Inc.
12	MQ	American Eagle Airlines Inc.
13	VX	Virgin America

In [ ]:
```
airports = pd.read_csv("/content/drive/MyDrive/BDF J COMP/airports.csv")
airports
```

Out[ ]:

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447
...	...	...	...	...	...	...	...
317	WRG	Wrangell Airport	Wrangell	AK	USA	56.48433	-132.36982
318	WYS	Westerly State Airport	West Yellowstone	MT	USA	44.68840	-111.11764
319	XNA	Northwest Arkansas Regional Airport	Fayetteville/Springdale/Rogers	AR	USA	36.28187	-94.30681
320	YAK	Yakutat Airport	Yakutat	AK	USA	59.50336	-139.66023
321	YUM	Yuma International Airport	Yuma	AZ	USA	32.65658	-114.60597

322 rows × 7 columns

In [ ]:
```
flights = pd.read_csv("/content/drive/MyDrive/BDF J COMP/flights.csv")
flights_data = flights[0:50000]
flights_data
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:2882: DtypeWarning: Columns (7,8) have mixed types.Spec
ify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[ ]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE
0	2015	1	1	4	AS	98	N407AS	ANC	SEA	5
1	2015	1	1	4	AA	2336	N3KUAA	LAX	PBI	10
2	2015	1	1	4	US	840	N171US	SFO	CLT	20
3	2015	1	1	4	AA	258	N3HYAA	LAX	MIA	20
4	2015	1	1	4	AS	135	N527AS	SEA	ANC	25
...	...	...	...	...	...	...	...	...	...	...
49995	2015	1	4	7	AA	1524	N499AA	DFW	LAS	915
49996	2015	1	4	7	AA	2316	N3FNAA	STX	MIA	915
49997	2015	1	4	7	AS	5	N566AS	DCA	LAX	915
49998	2015	1	4	7	DL	688	N893AT	ATL	ICT	915
49999	2015	1	4	7	DL	972	N130DL	MSP	LAX	915

50000 rows × 31 columns

Creating a Temp

In [ ]:
```
# Create pd_temp
pd_temp = pd.DataFrame(np.random.random(10))

# Create spark_temp from pd_temp
spark_temp = spark.createDataFrame(pd_temp)

# Examine the tables in the catalog
print(spark.catalog.listTables())

# Add spark_temp to the catalog
spark_temp.createOrReplaceTempView('temp')

# Examine the tables in the catalog again
print(spark.catalog.listTables())
```

```
[]
[Table(name='temp', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]
```

In [ ]:
```
# Read in the airports data
airports = spark.read.csv(['/content/drive/MyDrive/BDF J COMP/airports.csv'], header = True)

# Show the data
airports.show(10)
```

```
+---------+--------------------+------------+-----+-------+--------+----------+
|IATA_CODE|             AIRPORT|        CITY|STATE|COUNTRY|LATITUDE| LONGITUDE|
+---------+--------------------+------------+-----+-------+--------+----------+
|      ABE|Lehigh Valley Int...|   Allentown|   PA|    USA|40.65236| -75.44040|
|      ABI|Abilene Regional ...|     Abilene|   TX|    USA|32.41132| -99.68190|
|      ABQ|Albuquerque Inter...| Albuquerque|   NM|    USA|35.04022|-106.60919|
|      ABR|Aberdeen Regional...|    Aberdeen|   SD|    USA|45.44906| -98.42183|
|      ABY|Southwest Georgia...|      Albany|   GA|    USA|31.53552| -84.19447|
|      ACK|Nantucket Memoria...|   Nantucket|   MA|    USA|41.25305| -70.06018|
|      ACT|Waco Regional Air...|        Waco|   TX|    USA|31.61129| -97.23052|
|      ACV|      Arcata Airport|Arcata/Eureka|   CA|    USA|40.97812|-124.10862|
|      ACY|Atlantic City Int...|Atlantic City|   NJ|    USA|39.45758| -74.57717|
|      ADK|        Adak Airport|        Adak|   AK|    USA|51.87796|-176.64603|
+---------+--------------------+------------+-----+-------+--------+----------+
only showing top 10 rows
```

```
In [ ]: flight_data = spark.read.csv(['/content/drive/MyDrive/BDF J COMP/flights.csv'], header = True)
        flight_data = flight_data.take(50000)

        flights=spark.createDataFrame(flight_data)

        # print the tables in catalog
        print(spark.catalog.listTables())

        # adding data into spark view for sql querying
        flights.createOrReplaceTempView('flights')

        # print the tables in catalog
        print(spark.catalog.listTables())
```

```
[Table(name='temp', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]
[Table(name='flights', database=None, description=None, tableType='TEMPORARY', isTemporary=True), Table(name='temp', database=N
one, description=None, tableType='TEMPORARY', isTemporary=True)]
```

```
In [ ]: flights.show()
```

```
+----+-----+---+-----------+-------+-------------+-----------+--------------+-------------------+-------------------+----------
----+---------------+--------+----------+--------------+------------+--------+--------+---------+-------+-----------------+----
--------+-------------+--------+---------+-------------------+----------------+--------------+-------------+-------------------
+-------------+
|YEAR|MONTH|DAY|DAY_OF_WEEK|AIRLINE|FLIGHT_NUMBER|TAIL_NUMBER|ORIGIN_AIRPORT|DESTINATION_AIRPORT|SCHEDULED_DEPARTURE|DEPARTURE_
TIME|DEPARTURE_DELAY|TAXI_OUT|WHEELS_OFF|SCHEDULED_TIME|ELAPSED_TIME|AIR_TIME|DISTANCE|WHEELS_ON|TAXI_IN|SCHEDULED_ARRIVAL|ARRI
VAL_TIME|ARRIVAL_DELAY|DIVERTED|CANCELLED|CANCELLATION_REASON|AIR_SYSTEM_DELAY|SECURITY_DELAY|AIRLINE_DELAY|LATE_AIRCRAFT_DELAY
|WEATHER_DELAY|
+----+-----+---+-----------+-------+-------------+-----------+--------------+-------------------+-------------------+----------
----+---------------+--------+----------+--------------+------------+--------+--------+---------+-------+-----------------+----
--------+-------------+--------+---------+-------------------+----------------+--------------+-------------+-------------------
+-------------+
|2015|    1|  1|          4|     AS|           98|     N407AS|           ANC|                SEA|               0005|
2354|            -11|      21|      0015|           205|         194|     169|    1448|     0404|      4|             0430|
0408|           -22|       0|         0|               null|            null|          null|         null|               null|
null|
|2015|    1|  1|          4|     AA|         2336|     N3KUAA|           LAX|                PBI|               0010|
0002|             -8|      12|      0014|           280|         279|     263|    2330|     0737|      4|             0750|
0741|             -9|       0|         0|               null|            null|          null|         null|               null|
```

```
In [ ]: # Show the data shape
        print((flights.count(), len(flights.columns)))
```

```
(50000, 31)
```

```
In [ ]: # see all columns in the table
        print(flights.columns)
```

```
['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'AIRLINE', 'FLIGHT_NUMBER', 'TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', 'SC
HEDULED_DEPARTURE', 'DEPARTURE_TIME', 'DEPARTURE_DELAY', 'TAXI_OUT', 'WHEELS_OFF', 'SCHEDULED_TIME', 'ELAPSED_TIME', 'AIR_TIM
E', 'DISTANCE', 'WHEELS_ON', 'TAXI_IN', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME', 'ARRIVAL_DELAY', 'DIVERTED', 'CANCELLED', 'CANCELL
ATION_REASON', 'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY', 'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY']
```

Creating Queries in SparkSQL

```
In [ ]: query = "SELECT AIRLINE, FLIGHT_NUMBER, TAIL_NUMBER, ORIGIN_AIRPORT, DESTINATION_AIRPORT, SCHEDULED_DEPARTURE FROM flights LIMIT
        5"

        flights5 = spark.sql(query)
        flights5.show()
```

```
+-------+-------------+-----------+--------------+-------------------+-------------------+
|AIRLINE|FLIGHT_NUMBER|TAIL_NUMBER|ORIGIN_AIRPORT|DESTINATION_AIRPORT|SCHEDULED_DEPARTURE|
+-------+-------------+-----------+--------------+-------------------+-------------------+
|     AS|           98|     N407AS|           ANC|                SEA|               0005|
|     AA|         2336|     N3KUAA|           LAX|                PBI|               0010|
|     US|          840|     N171US|           SFO|                CLT|               0020|
|     AA|          258|     N3HYAA|           LAX|                MIA|               0020|
|     AS|          135|     N527AS|           SEA|                ANC|               0025|
+-------+-------------+-----------+--------------+-------------------+-------------------+
```

```
In [ ]: query = "SELECT ORIGIN_AIRPORT, DESTINATION_AIRPORT, COUNT(*) as N FROM flights GROUP BY ORIGIN_AIRPORT, DESTINATION_AIRPORT"


        flight_counts = spark.sql(query)
        pd_counts = flight_counts.toPandas()

        print(pd_counts.head())
```

```
  ORIGIN_AIRPORT DESTINATION_AIRPORT   N
0            BQN                 MCO   8
1            PHL                 MCO  46
2            MCI                 IAH  19
3            SPI                 ORD  10
4            SNA                 PHX  36
```

Creating dataframe

In [ ]:

```python
# Create the DataFrame flights
flights = spark.table("flights")

# Add duration_hrs
flights = flights.withColumn('duration_hrs', flights.AIR_TIME/60.)

# Show the head
flights.select('duration_hrs').show(10)
```

```
+------------------+
|      duration_hrs|
+------------------+
| 2.816666666666667|
| 4.383333333333334|
| 4.433333333333334|
|               4.3|
| 3.316666666666667|
| 3.433333333333333|
| 2.566666666666667|
|               3.8|
|2.8833333333333333|
|               3.1|
+------------------+
only showing top 10 rows
```

In [ ]:

```python
# Filter flights by passing a string
long_flights1 = flights.filter("DISTANCE > 1000")

# Filter flights by passing a column of boolean values
long_flights2 = flights.filter(flights.DISTANCE > 1000)
```

In [ ]:

```python
# Select the first set of columns
selected1 = flights.select('TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT',)

# Select the second set of columns
temp = flights.select(flights.ORIGIN_AIRPORT, flights.DESTINATION_AIRPORT, flights.AIRLINE)

temp.show()
```

```
+--------------+-------------------+-------+
|ORIGIN_AIRPORT|DESTINATION_AIRPORT|AIRLINE|
+--------------+-------------------+-------+
|           ANC|                SEA|     AS|
|           LAX|                PBI|     AA|
|           SFO|                CLT|     US|
|           LAX|                MIA|     AA|
|           SEA|                ANC|     AS|
|           SFO|                MSP|     DL|
|           LAS|                MSP|     NK|
|           LAX|                CLT|     US|
|           SFO|                DFW|     AA|
|           LAS|                ATL|     DL|
|           DEN|                ATL|     DL|
|           LAS|                MIA|     AA|
|           LAX|                MSP|     DL|
|           SLC|                ATL|     DL|
|           SEA|                MSP|     DL|
|           ANC|                SEA|     AS|
|           ANC|                SEA|     DL|
|           SFO|                IAH|     UA|
|           ANC|                PDX|     AS|
|           PDX|                MSP|     DL|
+--------------+-------------------+-------+
only showing top 20 rows
```

In [ ]:
```
# Define first filter
filterA = flights.ORIGIN_AIRPORT == "SEA"

# Define second filter
filterB = flights.DESTINATION_AIRPORT == "PDX"

# Filter the data, first by filterA then by filterB
selected2 = temp.filter(filterA).filter(filterB)
```

In [ ]:
```
# Define avg_speed
avg_speed = (flights.DISTANCE/(flights.AIR_TIME/60)).alias("avg_speed")

# Select the correct columns
speed1 = flights.select('TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', avg_speed)

# Create the same table using a SQL expression
speed2 = flights.selectExpr('TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT', "DISTANCE/(AIR_TIME/60) as avg_speed")
```

In [ ]:
```
#Cast the columns to integers
flights = flights.withColumn("MONTH", flights.MONTH.cast("integer"))
flights = flights.withColumn("DAY_OF_WEEK", flights.DAY_OF_WEEK.cast("integer"))
flights = flights.withColumn("AIR_TIME", flights.AIR_TIME.cast("integer"))
flights = flights.withColumn("DISTANCE", flights.DISTANCE.cast("double"))
flights = flights.withColumn("ARRIVAL_DELAY", flights.ARRIVAL_DELAY.cast("integer"))
```

In [ ]:
```
# Find the shortest flight from PDX in terms of distance
flights.filter(flights.ORIGIN_AIRPORT == 'PDX').groupBy().min('DISTANCE').show()

# Find the longest flight from SEA in terms of air time
flights.filter(flights.ORIGIN_AIRPORT == 'SEA').groupBy().max('AIR_TIME').show()
```

```
+-------------+
|min(DISTANCE)|
+-------------+
|        129.0|
+-------------+

+-------------+
|max(AIR_TIME)|
+-------------+
|          388|
+-------------+
```

In [ ]:
```
# Group by tailnum
by_plane = flights.groupBy("TAIL_NUMBER")

# Number of flights each plane made
by_plane.count().show(10)

# Group by origin
by_origin = flights.groupBy("ORIGIN_AIRPORT")

# Average duration of flights from PDX and SEA
by_origin.avg("AIR_TIME").show(10)
```

```
+-----------+-----+
|TAIL_NUMBER|count|
+-----------+-----+
|     N38451|    8|
|     N567AA|   16|
|     N623NK|   18|
|     N442AS|   12|
|     N902DE|   13|
|     N4YUAA|   14|
|     N466SW|   19|
|     N516UA|    9|
|     N866AS|   19|
|     N499AA|   15|
+-----------+-----+
only showing top 10 rows

+--------------+------------------+
|ORIGIN_AIRPORT|     avg(AIR_TIME)|
+--------------+------------------+
|           PSE|184.58333333333334|
|           INL|40.833333333333336|
|           MSY|104.45588235294117|
|           PPG|             299.0|
|           GEG| 87.19767441860465|
|           SNA|112.48580441640378|
|           BUR| 72.13939393939394|
|           GRB|              50.9|
|           GTF| 76.77777777777777|
|           IDA| 46.88461538461539|
+--------------+------------------+
only showing top 10 rows
```

In [ ]:

```python
import pyspark.sql.functions as F

# cast
flights = flights.withColumn("DEPARTURE_DELAY", flights.DEPARTURE_DELAY.cast("integer"))

# Group by month and dest
by_month_dest = flights.groupBy('MONTH', 'DESTINATION_AIRPORT')

# Average departure delay by month and destination
by_month_dest.avg('DEPARTURE_DELAY').show(10)

# Standard deviation of departure delay
by_month_dest.agg(F.stddev('DEPARTURE_DELAY')).show(10)
```

```
+-----+-------------------+--------------------+
|MONTH|DESTINATION_AIRPORT|avg(DEPARTURE_DELAY)|
+-----+-------------------+--------------------+
|    1|                ACY|                15.5|
|    1|                EYW|    4.235294117647059|
|    1|                OME|                -5.2|
|    1|                RDM|   10.666666666666666|
|    1|                TWF|    5.833333333333333|
|    1|                AEX|   18.703703703703702|
|    1|                GNV|   14.263157894736842|
|    1|                PIB|                53.2|
|    1|                YAK|                25.0|
|    1|                ABE|-0.23076923076923078|
+-----+-------------------+--------------------+
only showing top 10 rows

+-----+-------------------+----------------------------+
|MONTH|DESTINATION_AIRPORT|stddev_samp(DEPARTURE_DELAY)|
+-----+-------------------+----------------------------+
|    1|                ACY|          31.147985002195085|
|    1|                EYW|          24.356019508008895|
|    1|                OME|           5.215361924162119|
|    1|                RDM|          30.961076132688245|
|    1|                TWF|          17.451838489588045|
|    1|                AEX|           27.79016817677517|
|    1|                GNV|          31.869250427442083|
|    1|                PIB|           72.52378920050992|
|    1|                YAK|           42.91852746774987|
|    1|                ABE|           6.482718644646012|
+-----+-------------------+----------------------------+
only showing top 10 rows
```

In [ ]:

```python
print(airports.columns)

# Examine the data
print(airports.show(10))
```

```
['IATA_CODE', 'AIRPORT', 'CITY', 'STATE', 'COUNTRY', 'LATITUDE', 'LONGITUDE']
+---------+--------------------+-------------+-----+-------+--------+----------+
|IATA_CODE|             AIRPORT|         CITY|STATE|COUNTRY|LATITUDE| LONGITUDE|
+---------+--------------------+-------------+-----+-------+--------+----------+
|      ABE|Lehigh Valley Int...|    Allentown|   PA|    USA|40.65236| -75.44040|
|      ABI|Abilene Regional ...|      Abilene|   TX|    USA|32.41132| -99.68190|
|      ABQ|Albuquerque Inter...|  Albuquerque|   NM|    USA|35.04022|-106.60919|
|      ABR|Aberdeen Regional...|     Aberdeen|   SD|    USA|45.44906| -98.42183|
|      ABY|Southwest Georgia...|       Albany|   GA|    USA|31.53552| -84.19447|
|      ACK|Nantucket Memoria...|    Nantucket|   MA|    USA|41.25305| -70.06018|
|      ACT|Waco Regional Air...|         Waco|   TX|    USA|31.61129| -97.23052|
|      ACV|      Arcata Airport|Arcata/Eureka|   CA|    USA|40.97812|-124.10862|
|      ACY|Atlantic City Int...|Atlantic City|   NJ|    USA|39.45758| -74.57717|
|      ADK|        Adak Airport|         Adak|   AK|    USA|51.87796|-176.64603|
+---------+--------------------+-------------+-----+-------+--------+----------+
only showing top 10 rows

None
```

In [ ]:

```python
# Rename the faa column
airports = airports.withColumnRenamed("IATA_CODE", "DESTINATION_AIRPORT")

# Join the DataFrames
flights_with_airports = flights.join(airports , on = 'DESTINATION_AIRPORT', how = 'leftouter')

# Examine the new DataFrame
print(flights_with_airports.columns)
print(flights_with_airports.count())
```

```
['DESTINATION_AIRPORT', 'YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'AIRLINE', 'FLIGHT_NUMBER', 'TAIL_NUMBER', 'ORIGIN_AIRPORT', 'SC
HEDULED_DEPARTURE', 'DEPARTURE_TIME', 'DEPARTURE_DELAY', 'TAXI_OUT', 'WHEELS_OFF', 'SCHEDULED_TIME', 'ELAPSED_TIME', 'AIR_TIM
E', 'DISTANCE', 'WHEELS_ON', 'TAXI_IN', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME', 'ARRIVAL_DELAY', 'DIVERTED', 'CANCELLED', 'CANCELL
ATION_REASON', 'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY', 'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY', 'duration_hrs',
'AIRPORT', 'CITY', 'STATE', 'COUNTRY', 'LATITUDE', 'LONGITUDE']
50000
```

In [ ]:

```
flights_with_airports.select('FLIGHT_NUMBER', 'AIRPORT', 'CITY', 'STATE', 'COUNTRY', 'LATITUDE', 'LONGITUDE').show(10)
```

```
+-------------+--------------------+-----------------+-----+-------+--------+----------+
|FLIGHT_NUMBER|             AIRPORT|             CITY|STATE|COUNTRY|LATITUDE| LONGITUDE|
+-------------+--------------------+-----------------+-----+-------+--------+----------+
|           98|Seattle-Tacoma In...|          Seattle|   WA|    USA|47.44898|-122.30931|
|         2336|Palm Beach Intern...|  West Palm Beach|   FL|    USA|26.68316| -80.09559|
|          840|Charlotte Douglas...|        Charlotte|   NC|    USA|35.21401| -80.94313|
|          258|Miami Internation...|            Miami|   FL|    USA|25.79325| -80.29056|
|          135|Ted Stevens Ancho...|        Anchorage|   AK|    USA|61.17432|-149.99619|
|          806|Minneapolis-Saint...|      Minneapolis|   MN|    USA|44.88055| -93.21692|
|          612|Minneapolis-Saint...|      Minneapolis|   MN|    USA|44.88055| -93.21692|
|         2013|Charlotte Douglas...|        Charlotte|   NC|    USA|35.21401| -80.94313|
|         1112|Dallas/Fort Worth...|Dallas-Fort Worth|   TX|    USA|32.89595| -97.03720|
|         1173|Hartsfield-Jackso...|          Atlanta|   GA|    USA|33.64044| -84.42694|
+-------------+--------------------+-----------------+-----+-------+--------+----------+
only showing top 10 rows
```

In [ ]:

```
# Read in the airports data
airlines = spark.read.csv(['/content/drive/MyDrive/BDF J COMP/airlines.csv'], header = True)

# Show the data shape
print((airlines.count(), len(airlines.columns)))

airlines.show()
```

```
(14, 2)
+---------+--------------------+
|IATA_CODE|             AIRLINE|
+---------+--------------------+
|       UA|United Air Lines ...|
|       AA|American Airlines...|
|       US|     US Airways Inc.|
|       F9|Frontier Airlines...|
|       B6|     JetBlue Airways|
|       OO|Skywest Airlines ...|
|       AS|Alaska Airlines Inc.|
|       NK|    Spirit Air Lines|
|       WN|Southwest Airline...|
|       DL|Delta Air Lines Inc.|
|       EV|Atlantic Southeas...|
|       HA|Hawaiian Airlines...|
|       MQ|American Eagle Ai...|
```

MLLIB

Pre-processing:

In [ ]:

```
# filtering columns
model_data = flights.select('MONTH', 'DAY_OF_WEEK', 'AIRLINE', 'TAIL_NUMBER', 'DESTINATION_AIRPORT', 'AIR_TIME', 'DISTANCE', 'AR
RIVAL_DELAY',)

# Remove missing values
model_data = model_data.filter("ARRIVAL_DELAY is not NULL and AIRLINE is not NULL and AIR_TIME is not NULL and TAIL_NUMBER is no
t NULL")

# rows left
model_data.count()
```

Out[ ]: 48753

In [ ]:

```
# Create is_late (label)
model_data = model_data.withColumn("is_late", model_data.ARRIVAL_DELAY > 0)

# cast
model_data = model_data.withColumn("is_late", model_data.is_late.cast("integer"))

# rename column
model_data = model_data.withColumnRenamed("is_late", 'label')
```

In [ ]:

```
model_data.show(15)
```

```
+-----+-----------+-------+-----------+-------------------+--------+--------+-------------+-----+
|MONTH|DAY_OF_WEEK|AIRLINE|TAIL_NUMBER|DESTINATION_AIRPORT|AIR_TIME|DISTANCE|ARRIVAL_DELAY|label|
+-----+-----------+-------+-----------+-------------------+--------+--------+-------------+-----+
|    1|          4|     AS|     N407AS|                SEA|     169|  1448.0|          -22|    0|
|    1|          4|     AA|     N3KUAA|                PBI|     263|  2330.0|           -9|    0|
|    1|          4|     US|     N171US|                CLT|     266|  2296.0|            5|    1|
|    1|          4|     AA|     N3HYAA|                MIA|     258|  2342.0|           -9|    0|
|    1|          4|     AS|     N527AS|                ANC|     199|  1448.0|          -21|    0|
|    1|          4|     DL|     N3730B|                MSP|     206|  1589.0|            8|    1|
|    1|          4|     NK|     N635NK|                MSP|     154|  1299.0|          -17|    0|
|    1|          4|     US|     N584UW|                CLT|     228|  2125.0|          -10|    0|
|    1|          4|     AA|     N3LAAA|                DFW|     173|  1464.0|          -13|    0|
|    1|          4|     DL|     N826DN|                ATL|     186|  1747.0|          -15|    0|
|    1|          4|     DL|     N958DN|                ATL|     133|  1199.0|          -30|    0|
|    1|          4|     AA|     N853AA|                MIA|     238|  2174.0|          -10|    0|
|    1|          4|     DL|     N547US|                MSP|     188|  1535.0|           -4|    0|
|    1|          4|     DL|     N3751B|                ATL|     176|  1590.0|          -22|    0|
|    1|          4|     DL|     N651DL|                MSP|     166|  1399.0|            8|    1|
+-----+-----------+-------+-----------+-------------------+--------+--------+-------------+-----+
only showing top 15 rows
```

In [ ]:

```
print('Labels distrubution:')
model_data.groupBy('label').count().show()
```

```
Labels distrubution:
+-----+-----+
|label|count|
+-----+-----+
|    1|26053|
|    0|22700|
+-----+-----+
```

- Creating String Indexer, One hot Encoder, Vector Assembler, and Pipeline

```
In [ ]: from pyspark.ml.feature import OneHotEncoder, StringIndexer
        from pyspark.ml.feature import HashingTF, IDF, Tokenizer

        # Create a StringIndexer
        airline_indexer = StringIndexer(inputCol="AIRLINE", outputCol="airline_index")

        # Create a OneHotEncoder
        airline_encoder = OneHotEncoder(inputCol="airline_index", outputCol="airline_fact")
```

```
In [ ]: # Create a StringIndexer
        dest_indexer = StringIndexer(inputCol="DESTINATION_AIRPORT", outputCol="dest_index")

        # Create a OneHotEncoder
        dest_encoder = OneHotEncoder(inputCol="dest_index", outputCol="dest_fact")
```

```
In [ ]: # Create a StringIndexer
        tail_indexer = StringIndexer(inputCol="TAIL_NUMBER", outputCol="tail_index")

        # Create a OneHotEncoder
        tail_encoder = OneHotEncoder(inputCol="tail_index", outputCol="tail_fact")
```

```
In [ ]: from pyspark.ml.feature import VectorAssembler

        # Make a VectorAssembler of 'MONTH', 'DAY_OF_WEEK', 'AIR_TIME', 'DISTANCE', 'ARRIVAL_DELAY','AIRLINE', 'TAIL_NUMBER', 'DESTINATI
        ON_AIRPORT'
        vec_assembler = VectorAssembler(inputCols=["MONTH", "DAY_OF_WEEK", "AIR_TIME", "DISTANCE", "airline_fact", "dest_fact", "tail_fa
        ct"], outputCol="features")
```

```
In [ ]: # Import Pipeline
        from pyspark.ml import Pipeline

        # Make the pipeline
        flights_pipe = Pipeline(stages=[dest_indexer, dest_encoder, airline_indexer, airline_encoder, tail_indexer, tail_encoder, vec_as
        sembler])
```

```
In [ ]: piped_data = flights_pipe.fit(model_data).transform(model_data)
```

- Train Test Split

TRAIN - TEST SPLIT

```
In [ ]: train_data, test_data = piped_data.randomSplit([.7, .3])
```

```
In [ ]: print('data points(rows) in train data :',  train_data.count())
        print('data points(rows) in test data :',  test_data.count())

        data points(rows) in train data : 34135
        data points(rows) in test data : 14618
```

- Built 4 models : They are : Logistic regression, Linear SVC, Random Forest classifier, Decision Tree classifier.

```python
In [ ]: from pyspark.ml.classification import LogisticRegression
        from pyspark.ml.classification import LinearSVC
        from pyspark.ml.classification import DecisionTreeClassifier
        from pyspark.ml.classification import RandomForestClassifier

        import pyspark.ml.tuning as tune
        from time import time
        import pyspark.ml.evaluation as evals

        # Create a BinaryClassificationEvaluator
        evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")
```

```python
In [ ]: start_time = time()

        # Train a LogisticRegression model
        lr = LogisticRegression()

        model = lr.fit(train_data)

        end_time = time()
        elapsed_time = end_time - start_time
        print("Time to train LogisticRegression model: %.3f seconds" % elapsed_time)
```

```
Time to train model: 13.573 seconds
```

```python
In [ ]: start_time = time()

        # Train a Linear SVC model
        lsvc= LinearSVC()

        model = lsvc.fit(train_data)

        end_time = time()
        elapsed_time = end_time - start_time
        print("Time to train LinearSVC model: %.3f seconds" % elapsed_time)
```

```
Time to train model: 20.897 seconds
```

```python
In [ ]: start_time = time()

        # Train a DecisionTree model
        dt = DecisionTreeClassifier()


        model = dt.fit(train_data)

        end_time = time()
        elapsed_time = end_time - start_time
        print("Time to train DecisionTree model: %.3f seconds" % elapsed_time)
```

```
Time to train model: 16.446 seconds
```

```python
In [ ]: start_time = time()

        # Train a RandomForest model
        rf = RandomForestClassifier()

        model = rf.fit(train_data)

        end_time = time()
        elapsed_time = end_time - start_time
        print("Time to train RandomForest model: %.3f seconds" % elapsed_time)
```

```python
In [ ]: ########
        start_time = time()

        # Train a RandomForest model
        rf = RandomForestClassifier(labelCol="label",
             featuresCol="features",
             numTrees=500,
             maxDepth=3,
             seed = 1,
             featureSubsetStrategy="sqrt",
             impurity='gini')

        model = rf.fit(train_data)

        end_time = time()
        elapsed_time = end_time - start_time
        print("Time to train RandomForest model: %.3f seconds" % elapsed_time)
```

LOGISTIC REGRESSION

In [ ]:
```
# Import LogisticRegression
from pyspark.ml.classification import LogisticRegression

# Create a LogisticRegression Estimator
lr = LogisticRegression()
```

In [ ]:
```
# Import the evaluation submodule
import pyspark.ml.evaluation as evals

# Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")
```

In [ ]:
```
# Import the tuning submodule
import pyspark.ml.tuning as tune

# Create the parameter grid
grid1 = tune.ParamGridBuilder()

# Add the hyperparameter
grid1 = grid1.addGrid(lr.regParam, np.arange(0, .1, .01))
grid1 = grid1.addGrid(lr.elasticNetParam, [0, 1])

# Build the grid
grid1 = grid1.build()
```

In [ ]:
```
# Create the CrossValidator
cv = tune.CrossValidator(estimator=lr,
               estimatorParamMaps=grid1,
               evaluator=evaluator)
```

In [ ]:
```
# Call lr.fit()
best_lr = lr.fit(train_data)

# Print best_lr
print(best_lr)
```

```
LogisticRegressionModel: uid=LogisticRegression_7d2f24b40302, numClasses=2, numFeatures=5540
```

In [ ]:
```
# Use the model to predict the test set
test_results = best_lr.transform(test_data)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```

```
0.6264944692991006
```

LINEAR SVC

In [ ]:
```
from pyspark.ml.classification import LinearSVC

lsvc= LinearSVC()
```

In [ ]:
```
# Import the evaluation submodule
import pyspark.ml.evaluation as evals

# Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")
```

In [ ]:
```
# Import the tuning submodule
import pyspark.ml.tuning as tune

# Create the parameter grid
grid2 = tune.ParamGridBuilder()

# Add the hyperparameter
grid2 = grid2.addGrid(lsvc.regParam, np.arange(0, .1, .01))


# Build the grid
grid2 = grid2.build()
```

In [ ]:
```
# Create the CrossValidator
cv = tune.CrossValidator(estimator=lsvc,
               estimatorParamMaps=grid2,
               evaluator=evaluator)
```

In [ ]:
```
# Call lsvc.fit()
best_lsvc = lsvc.fit(train_data)

# Print best_lr
print(best_lsvc)
```

```
LinearSVCModel: uid=LinearSVC_00ce8cd6c87f, numClasses=2, numFeatures=5540
```

In [ ]:
```
# Use the model to predict the test set
test_results = best_lsvc.transform(test_data)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```

```
0.6138897685038908
```

DECISION TREE

In [ ]:
```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier()


# Import the evaluation submodule
import pyspark.ml.evaluation as evals

# Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")

# Import the tuning submodule
import pyspark.ml.tuning as tune

# Create the parameter grid
grid3 = tune.ParamGridBuilder()

# Build the grid
grid3 = grid3.build()

# Create the CrossValidator
cv = tune.CrossValidator(estimator=dt,
               estimatorParamMaps=grid3,
               evaluator=evaluator)


# Call dt.fit()
best_dt = dt.fit(train_data)

# Print best_dt
print(best_dt)


# Use the model to predict the test set
test_results = best_dt.transform(test_data)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```

```
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_4df89ef9808c, depth=5, numNodes=29, numClasses=2, numFeatures=5540
0.5219383090728698
```

RANDOM FOREST CLASSIFIER

In [ ]:
```python
from pyspark.ml.classification import RandomForestClassifier
import pyspark.ml.tuning as tune
from time import time
import pyspark.ml.evaluation as evals


# Create a BinaryClassificationEvaluator
evaluator = evals.BinaryClassificationEvaluator(metricName="areaUnderROC")


rf = RandomForestClassifier()


# Create the parameter grid
grid4 = tune.ParamGridBuilder()

# Build the grid
grid4 = grid4.build()

# Create the CrossValidator
cv = tune.CrossValidator(estimator=rf,
              estimatorParamMaps=grid4,
              evaluator=evaluator)


# Call rf.fit()
start_time = time()

best_rf = rf.fit(train_data)

end_time = time()
elapsed_time = end_time - start_time
print("Time to train model: %.3f seconds" % elapsed_time)

# Print best_rf
print(best_rf)
```

```
Time to train model: 26.316 seconds
RandomForestClassificationModel: uid=RandomForestClassifier_29a3e474cb3d, numTrees=20, numClasses=2, numFeatures=4406
```

In [ ]:
```python
# Use the model to predict the test set
s = time()
test_results = best_rf.transform(test_data)
e= time()
elapsed_time = e-s
print("Time to test model: %.3f seconds" % elapsed_time)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```

```
Time to test model: 0.060 seconds
0.6564176358340545
```

In [ ]:
```python
from pyspark.ml.classification import RandomForestClassifier
from time import time

start_time = time()

# Train a RandomForest model
rf = RandomForestClassifier(labelCol="label",
     featuresCol="features",
     numTrees=500,
     maxDepth=3,
     seed = 1,
     featureSubsetStrategy="sqrt",
     impurity='gini')

model = rf.fit(train_data)

end_time = time()
elapsed_time = end_time - start_time
print("Time to train model: %.3f seconds" % elapsed_time)
```

In [ ]:
```python
# Create the parameter grid
grid4 = tune.ParamGridBuilder()

# Build the grid
grid4 = grid4.build()

# Create the CrossValidator
cv = tune.CrossValidator(estimator=rf,
              estimatorParamMaps=grid4,
              evaluator=evaluator)


# Call rf.fit()
best_rf = rf.fit(train_data)

# Print best_rf
print(best_rf)

# Use the model to predict the test set
test_results = best_rf.transform(test_data)

# Evaluate the predictions
print(evaluator.evaluate(test_results))
```

In [ ]:

2. PYTHON :

Importing

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
low_memory=False
```

In [4]:
```
# Store the path in variables
airlines_path = "../input/flight-delays/airlines.csv"
airport_path = "../input/flight-delays/airports.csv"
flights_path = "../input/flight-delays/flights.csv"

# Load the data
airlines_data = pd.read_csv(airlines_path)
airport_data = pd.read_csv(airport_path)
flights_data = pd.read_csv(flights_path)
```

```
/opt/conda/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3524:
ify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [5]:
```
airlines_data.head()
```

Out[5]:

	IATA_CODE	AIRLINE
0	UA	United Air Lines Inc.
1	AA	American Airlines Inc.
2	US	US Airways Inc.
3	F9	Frontier Airlines Inc.
4	B6	JetBlue Airways

In [6]:
```
airport_data.head()
```

Out[6]:

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

In [7]:
```
flights_data['DEPARTURE_DELAY'].max()
```

Out[7]: 1988.0

In [8]:
```
flights_data.head()
```

Out[8]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...
0	2015	1	1	4	AS	98	N407AS	ANC	SEA	5	...
1	2015	1	1	4	AA	2336	N3KUAA	LAX	PBI	10	...
2	2015	1	1	4	US	840	N171US	SFO	CLT	20	...
3	2015	1	1	4	AA	258	N3HYAA	LAX	MIA	20	...
4	2015	1	1	4	AS	135	N527AS	SEA	ANC	25	...

5 rows × 31 columns

In [9]:
```
flights_data.shape
```

Out[9]: (5819079, 31)

In [9]:
```
flights_data.shape
```

Out[9]: (5819079, 31)

In [10]:
```
#lets take a segment of this data for now
flights_seg = flights_data[0:150000]
flights_seg
```

Out[10]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE
0	2015	1	1	4	AS	98	N407AS	ANC	SEA	5
1	2015	1	1	4	AA	2336	N3KUAA	LAX	PBI	10
2	2015	1	1	4	US	840	N171US	SFO	CLT	20
3	2015	1	1	4	AA	258	N3HYAA	LAX	MIA	20
4	2015	1	1	4	AS	135	N527AS	SEA	ANC	25
...	...	...	...	...	...	...	...	...	...	...
149995	2015	1	10	6	EV	4607	N15572	XNA	IAH	1504
149996	2015	1	10	6	WN	4388	N7723E	ATL	DAL	1505
149997	2015	1	10	6	WN	2903	N218WN	ATL	DCA	1505
149998	2015	1	10	6	WN	4519	N436WN	ATL	MCO	1505
149999	2015	1	10	6	WN	2492	N940WN	ATL	MSY	1505

150000 rows × 31 columns

In [11]:
```
flights_seg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 31 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   YEAR                 150000 non-null  int64  
 1   MONTH                150000 non-null  int64  
 2   DAY                  150000 non-null  int64  
 3   DAY_OF_WEEK          150000 non-null  int64  
 4   AIRLINE              150000 non-null  object 
 5   FLIGHT_NUMBER        150000 non-null  int64  
 6   TAIL_NUMBER          149693 non-null  object 
 7   ORIGIN_AIRPORT       150000 non-null  object 
 8   DESTINATION_AIRPORT  150000 non-null  object 
 9   SCHEDULED_DEPARTURE  150000 non-null  int64  
 10  DEPARTURE_TIME       146099 non-null  float64
 11  DEPARTURE_DELAY      146099 non-null  float64
 12  TAXI_OUT             145976 non-null  float64
 13  WHEELS_OFF           145976 non-null  float64
 14  SCHEDULED_TIME       150000 non-null  float64
 15  ELAPSED_TIME         145557 non-null  float64
 16  AIR_TIME             145557 non-null  float64
 17  DISTANCE             150000 non-null  int64  
 18  WHEELS_ON            145821 non-null  float64
 19  TAXI_IN              145821 non-null  float64
 20  SCHEDULED_ARRIVAL    150000 non-null  int64  
 21  ARRIVAL_TIME         145821 non-null  float64
 22  ARRIVAL_DELAY        145557 non-null  float64
 23  DIVERTED             150000 non-null  int64  
 24  CANCELLED            150000 non-null  int64  
 25  CANCELLATION_REASON  4061 non-null    object 
 26  AIR_SYSTEM_DELAY     46924 non-null   float64
 27  SECURITY_DELAY       46924 non-null   float64
 28  AIRLINE_DELAY        46924 non-null   float64
 29  LATE_AIRCRAFT_DELAY  46924 non-null   float64
 30  WEATHER_DELAY        46924 non-null   float64
dtypes: float64(16), int64(10), object(5)
memory usage: 35.5+ MB
```

In [12]:
```
#year column is unneccesary since the data is bounded to 2015 but day and month are important
delay =[]
for row in flights_seg['ARRIVAL_DELAY']:
    if row > 60:
        delay.append(3)
    elif row > 30:
        delay.append(2)
    elif row > 15:
        delay.append(1)
    else:
        delay.append(0)
flights_seg['delay'] = delay
```

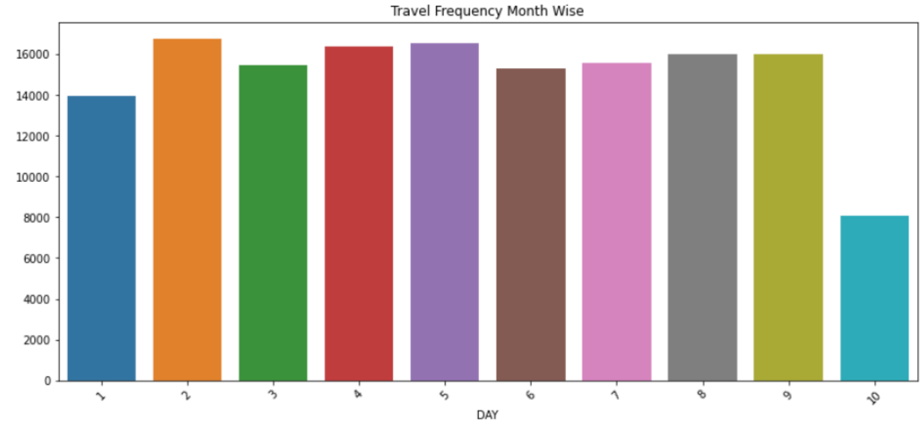
```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/ind
rsus-a-copy
  if sys.path[0] == '':
```

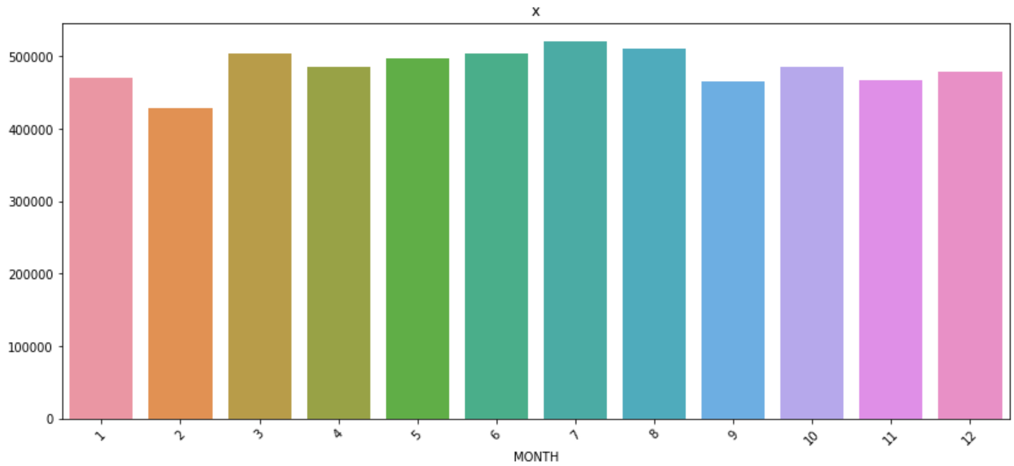
In [13]:
```
# 0 = On time/ before time/ not more than 15 mins of delay
# 1 = more than 15 mins and less than 30 mins of delay
# 2 = more than 30 mins and less than 1 hr of delay
# 3 = more than an hour of delay
flights_seg.value_counts('delay')
```

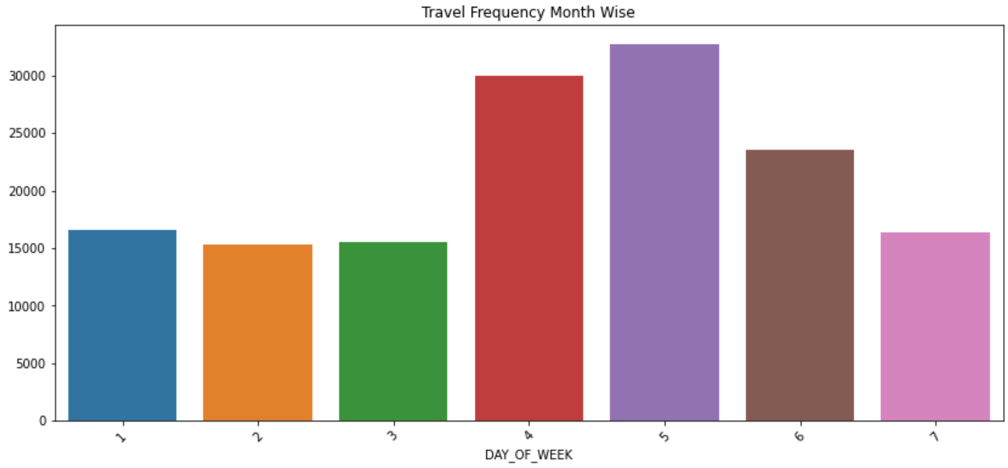
Out[13]:
```
delay
0    104480
1     15460
3     15397
2     14663
dtype: int64
```

Visualization

In [14]:
```
def plot_bar(group, title):
    plt.figure(figsize=(14,6))
    sns.barplot(x=group.index,y=group.values)
    plt.title(title)
    plt.xticks(rotation=45)
    plt.show()
plot_bar(flights_seg.value_counts('DAY'), 'Travel Frequency Month Wise')
plot_bar(flights_data.value_counts('MONTH'), 'x')
plot_bar(flights_seg.value_counts('DAY_OF_WEEK'), 'Travel Frequency Month Wise')
```

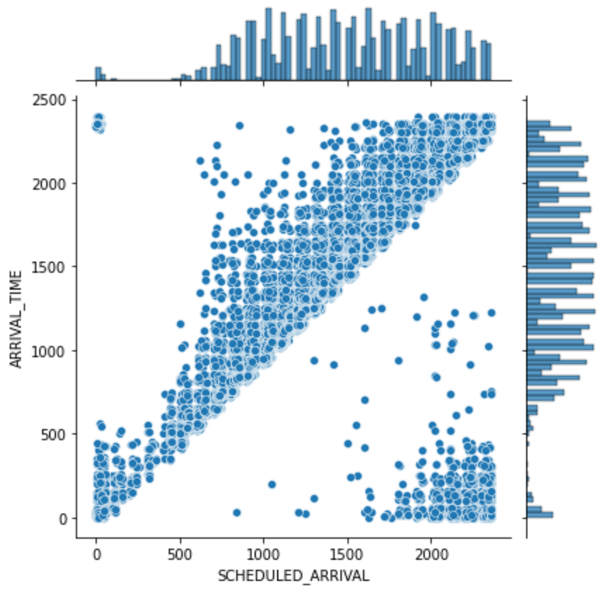






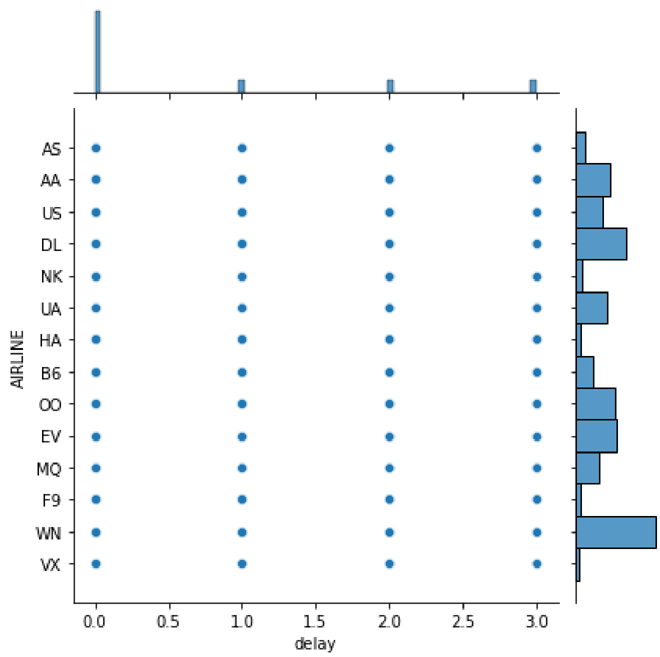
```
In [15]: sns.jointplot(data=flights_seg, x="SCHEDULED_ARRIVAL", y="ARRIVAL_TIME")
```

```
Out[15]: <seaborn.axisgrid.JointGrid at 0x7f4eef4a38d0>
```




```
In [16]: sns.jointplot(data=flights_seg, y="AIRLINE", x="delay")
```

```
Out[16]: <seaborn.axisgrid.JointGrid at 0x7f4ef061af10>
```

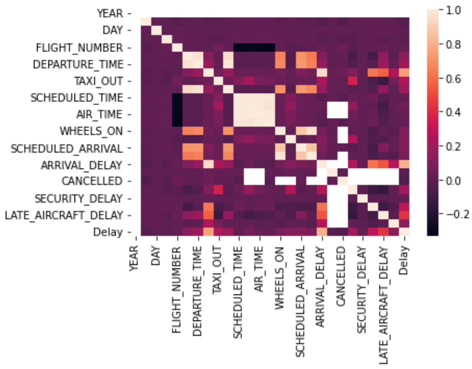


In [17]:
```python
Flight_data_delay =[]
for row in flights_data['ARRIVAL_DELAY']:
    if row > 60:
        Flight_data_delay.append(3)
    elif row > 30:
        Flight_data_delay.append(2)
    elif row > 15:
        Flight_data_delay.append(1)
    else:
        Flight_data_delay.append(0)
```

In [18]:
```python
flights_data['Delay'] = Flight_data_delay
```

In [19]:
```python
sns.heatmap(flights_data.corr())
```

Out[19]: <AxesSubplot:>



In [20]:
```python
flights_data=flights_data.drop(['YEAR','FLIGHT_NUMBER','AIRLINE','DISTANCE','TAIL_NUMBER','TAXI_OUT','SCHEDULED_TIME','DEPARTURE
_TIME','WHEELS_OFF','ELAPSED_TIME','AIR_TIME','WHEELS_ON','DAY_OF_WEEK','TAXI_IN','CANCELLATION_REASON','ORIGIN_AIRPORT', 'DESTI
NATION_AIRPORT', 'ARRIVAL_TIME', 'ARRIVAL_DELAY', "CANCELLED"],
                                axis=1)
```

In [21]:
```python
flights_data.describe()
```

Out[21]:

	MONTH	DAY	SCHEDULED_DEPARTURE	DEPARTURE_DELAY	SCHEDULED_ARRIVAL	DIVERTED	AIR_SYSTEM_DELAY	SECURITY_DEL.
count	5.819079e+06	5.819079e+06	5.819079e+06	5.732926e+06	5.819079e+06	5.819079e+06	1.063439e+06	1.063439e+
mean	6.524085e+00	1.570459e+01	1.329602e+03	9.370158e+00	1.493808e+03	2.609863e-03	1.348057e+01	7.615387e-
std	3.405137e+00	8.783425e+00	4.837518e+02	3.708094e+01	5.071647e+02	5.102012e-02	2.800368e+01	2.143460e+
min	1.000000e+00	1.000000e+00	1.000000e+00	-8.200000e+01	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
25%	4.000000e+00	8.000000e+00	9.170000e+02	-5.000000e+00	1.110000e+03	0.000000e+00	0.000000e+00	0.000000e+
50%	7.000000e+00	1.600000e+01	1.325000e+03	-2.000000e+00	1.520000e+03	0.000000e+00	2.000000e+00	0.000000e+
75%	9.000000e+00	2.300000e+01	1.730000e+03	7.000000e+00	1.918000e+03	0.000000e+00	1.800000e+01	0.000000e+
max	1.200000e+01	3.100000e+01	2.359000e+03	1.988000e+03	2.400000e+03	1.000000e+00	1.134000e+03	5.730000e+

In [22]:
```python
flights_data=flights_data.fillna(flights_data.mean())
```

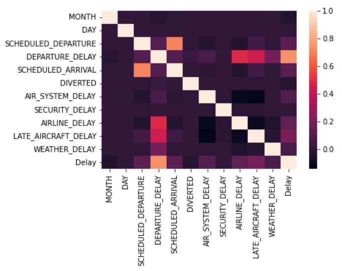
In [23]:
```python
flights_data.head(20)
```

Out[23]:

	MONTH	DAY	SCHEDULED_DEPARTURE	DEPARTURE_DELAY	SCHEDULED_ARRIVAL	DIVERTED	AIR_SYSTEM_DELAY	SECURITY_DELAY	AIRLINE_DELAY
0	1	1	5	-11.0	430	0	13.480568	0.076154	18.969547
1	1	1	10	-8.0	750	0	13.480568	0.076154	18.969547
2	1	1	20	-2.0	806	0	13.480568	0.076154	18.969547
3	1	1	20	-5.0	805	0	13.480568	0.076154	18.969547
4	1	1	25	-1.0	320	0	13.480568	0.076154	18.969547
5	1	1	25	-5.0	602	0	13.480568	0.076154	18.969547
6	1	1	25	-6.0	526	0	13.480568	0.076154	18.969547
7	1	1	30	14.0	803	0	13.480568	0.076154	18.969547

In [24]:
```python
sns.heatmap(flights_data.corr())
```

Out[24]: <AxesSubplot:>



Models

In [25]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
```

In [26]:
```
data = flights_data.values
X, y = data[:,:-1], data[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [27]:
```
clf = DecisionTreeClassifier()
clf = clf.fit(X_train,y_train)
```

In [28]:
```
pred_prob = clf.predict_proba(X_test)
auc_score = roc_auc_score(y_test, pred_prob, multi_class='ovr')
auc_score
```

Out[28]:
```
0.9983520792255044
```

In [29]:
```
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression()
lr = lr.fit(X_train,y_train)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (sta
tus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

In [30]:
```
pred_prob = lr.predict_proba(X_test)
lr.score(X_test, y_test)
```

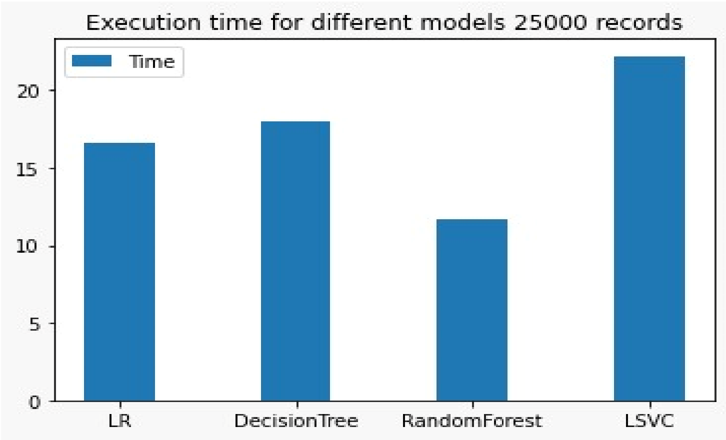
Out[30]:
```
0.9496123098496669
```

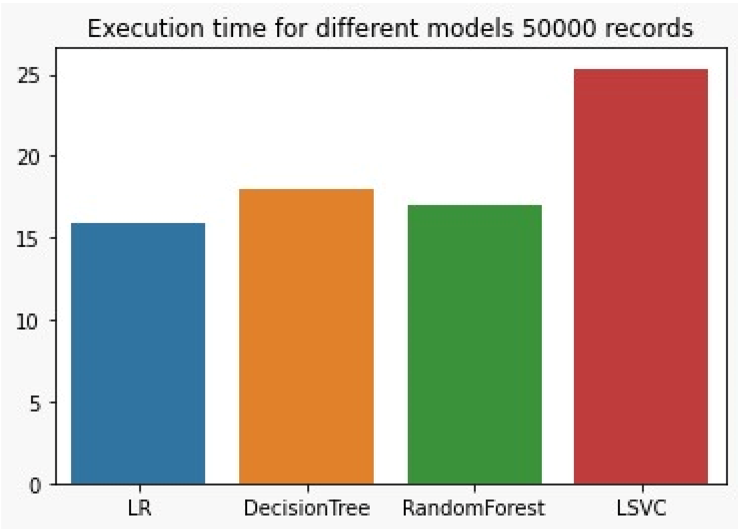
In [31]:
```
from sklearn.ensemble import RandomForestClassifier
rf= RandomForestClassifier()
rf= rf.fit(X_train,y_train)
```

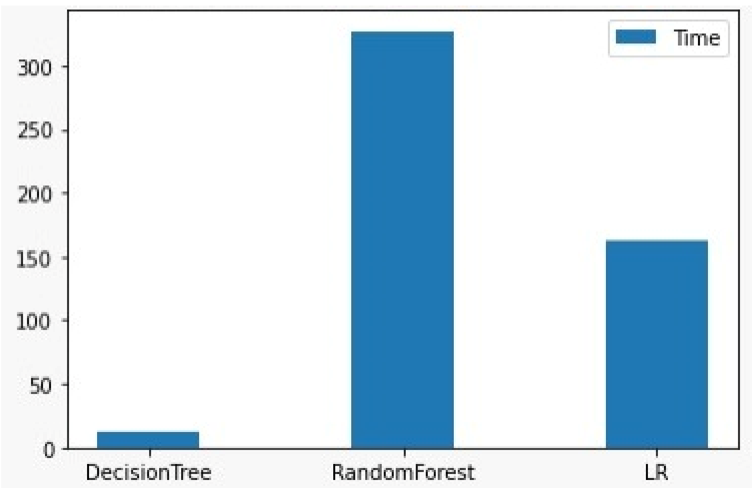
In [32]:
```
pred_prob = rf.predict_proba(X_test)
rf.score(X_test, y_test)
```

Out[32]:
```
0.9990462409865477
```

RESULTS







FOR 50000 RECORDS

	PYTHON	PYSPARK
LOGISTIC REGRESSION	161.811	15.924
DECISION TREE	11.998	18.024
RANDOM FOREST	327.073	17.026
LINEAR SVC		25.35

CONCLUSION

We have then compared the execution time for 4 Machine Learning models which are Logistic Regression, Decision Tree, Linear SVC, Random Forest in both Python and PySpark. Our results show that the time executed in PySpark is less than Python proving PySpark being better for processing large datasets.

REFERENCES

[1] A. B. Guy, "Flight delays cost $32.9 billion, passengers foot half the bill". [Online] Available : https://news.berkeley.edu/2010/10/18/flight_delays/3/. [Accessed on June 2017].

[2] M. Abdel-Aty, C. Lee, Y. Bai, X. Li and M. Michalak, "Detecting periodic patterns of arrival delay", Journal of Air Transport Management,, Volume 13(6), pp. 355– 361, November, 2007.

[3] S. AhmadBeygi, A. Cohn and M. Lapp, "Decreasing Airline Delay Propagation By Re-Allocating Scheduled Slack", Annual Conference, Boston, 2008.

[4] A. A. Simmons, "Flight Delay Forecast due to Weather Using Data Mining", M.S. Disseration, University of the Basque Country, Department of Computer Science, 2015.

[5] S. Choi, Y. J. Kim, S. Briceno and D. Mavris, "Prediction of weather-induced airline delays based on machine learning algorithms", Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th, Sacramento, CA, USA, 2016.

[6] L. Schaefer and D. Millner, "Flight Delay Propagation Analysis With The Detailed Policy Assessment Tool", Man and Cybernetics Conference, Tucson, AZ, 2001.

[7] B. Liu "Sentiment Analysis and Opinion Mining Synthesis", Morgan & Claypool Publishers, p. 167, 2012.