VIT®

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 Integrated (CSE)

Course Title: Big Data Frameworks

Course Code: CSE3120

Slot: F1

Title: Airport data analysis using Big Data Frameworks

Team Members: Arya Dadhich | 19MIA1025

Udbhav Nemmani | 19MIA1041

Gaurav Trivedi | 19MIA1077

Faculty: Suganeshwari G.

DECLARATION

We hereby declare that the project entitled "**Airport data analysis using Big Data Frameworks**" submitted by Arya(19MIA1025), Udbhav(19MIA1041), Gaurav(19MIA1077) 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 us. We 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

Arya Dadhich | 19MIA1025

Udbhav Nemmani | 19MIA1041

Gaurav Trivedi | 19MIA1077

INDEX

Ch. No	Chapter
1.	Abstract
2.	Introduction
3.	Review of Literature
4.	Problem Statement
5.	Proposed System, architectures, modules
6.	Implementation and Results
7.	Conclusion
	Reference

ABSTRACT:

Precise prediction of passenger flow is very important for any company to create their business policies. The passenger analysis uses key technologies that is transmission of data dynamically, huge amount of data storage, fusing of data through multiple sources, data-mining and other analysis. With the use of visualisation, data prediction and decision making, the complete set of data (authorities, passengers) can create their own goals and perspectives. Therefore, the research provides both, accurate information about the transport services to common citizens and at the same time specify business models for lower tier and higher tier companies alike.

INTRODUCTION:

Urban traffic includes a variety of elements such as cars, trucks, buses, taxis, public administration, transport interchange, Infrastructure of the traffic and air travel. A huge volume of Big Data is in crude form that is structureless information, large volume of a single type of data, administration in flight analysis is information obtaining, large amount of data, managing the data, testing the data, and data representation. Notwithstanding the traits of enormous information specified above, it is fundamental that instruments exist for representation and comprehension of the data and relations between the information present in the datasets, which is called, Business Insight (BI). This requires information stockpiling and administration, equipment and programming assets, proper space learning, and new techniques and advances. Joining enormous information with examination can give a significant preferred standpoint to settle on auspicious and efficient choices identified with A) Cost B) Time C) Item Improvement D) Enhancement A great amount of data is captured and used in different configurations (organized, semi-organized and unstructured), from several sources (sensors, machines, applications, web, IoT) and checked by the associations. The information is captured, kept aside or is taken care by constant or clusters with the help of mechanical procedures or calculations. Implementation of these plans change for different areas along with time, from avionics to automobile industries, taking care of all the account and capital conjectures, correlation, utilities and mining, government, hospitality industry, protection, retail, innovation, and so on. It is important for these professions to make most out of these little signs from a few important data sources both organized and unstructured. It conveys an ongoing effect for a simple, efficient and powerful basic leadership. The paper deals with

Commercial Aviation data for the improvements of service, and makes use of Latin Pig Scripts which are far more efficient than normal SQL Queries, and can work on unstructured as well as structured data. The data from the previous years is visualized and according to the trends the future plans and services are made according to the same. The proposed system is available for both business as well as individual users. Data Mining & Prediction through Visualization is taken care of more than the storage and transportation which makes it more efficient. The proposed system is far more efficient than the traditional methods of data mining and processing. Structured as well as Unstructured data is taken in to account.

REVIEW OF LITERATURE:

- ***Prediction for Air Route Passenger Flow Based on a Grey Prediction Model***
  The method used by this paper is regression analysis and grey prediction. It predicts the passenger flow of an air route which guides the airline company to estimate the passenger flow and thus helping in making better sales policy. Its major drawback is it doesn’t work efficiently with big data sets.

- ***A Kind of Novel ITS Based-on Space-Air-Ground Big-Data***
  The method used by this paper is dynamic data transmission, multi-source data fusion. It provides accurate transportation information services for the citizens. The major drawback is that It uses complex map reduction algorithms.

- ***Application of Big Data Visualization in Passenger Flow Analysis of Shanghai Metro Network***
  The method used here is Cluster analysis. It provides new means for passenger flow analysis and operation aid decision making (ADM). The major drawback is it doesn’t work efficiently with big data sets.

PROBLEM STATEMENT:

The Air Traffic in recent years have drastically increased and hence requires more assistance and directory. As there is increase in the number of passengers, there is a need for a better data storage and data analysis. With the increase in the number of passengers, the time taken for travel is also taken into considerable account. As the passenger doesn't want to waste any time by booking a flight, it is evident that he wants to avoid the waiting time at the airports too. To add to this, the waiting time in queue also eats up a significant amount of time. Due to this, not only the passengers but also the airport agencies are affected by this. Therefore, many agencies and companies have done analysis and created methods to overcome these problems and help the customer. As the data is increasing drastically day by day, the need for a better system to handle and analyze the data is required. The existing systems collect data from helicopters, satellites, planes etc. and the analysis is done using traditional means such as SQL. The collected data is used for data mining and visualization purpose. The data collected has a lot of unwanted data which increases complexity. These methods are focused more on the cloud storage & transportation, rather than on mining and other applications. The performance and efficiency of the system is compromised due to the large data sets. The point of view of the paper is purely from business perspective.

PROPOSED SYSTEM, ARCHITECTURE AND MODULES:

Applied descriptive statistics visualization and predictive modelling to identify the delay in flights and do passenger analysis. While descriptive analysis enabled us to compare the Average Cancellation by each passenger at average conditions, hard clustering and various visualization techniques permitted us to identify and predict the cancellation and delays in flights. Using regression, we determined the strength and relationship b/w the dependent and series of independent variables. The accuracy of 8 each model i.e; logistic regression, decision tree and SVM was hence calculated. Random Forest Classifier gave the highest accuracy so it was used for prediction.

IMPLEMENTATION AND RESULTS:

1. *SQL and Mapreduce*

- Installing packages and libraries

```
[1] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
    !wget -q https://downloads.apache.org/spark/spark-3.0.3/spark-3.0.3-bin-hadoop3.2.tgz
    !tar -xvf spark-3.0.3-bin-hadoop3.2.tgz  #extract the file using the tar command
    !pip install -q findspark     #install python package

    spark-3.0.3-bin-hadoop3.2/
    spark-3.0.3-bin-hadoop3.2/NOTICE
    spark-3.0.3-bin-hadoop3.2/kubernetes/
    spark-3.0.3-bin-hadoop3.2/kubernetes/tests/
    spark-3.0.3-bin-hadoop3.2/kubernetes/tests/worker_memory_check.py
    spark-3.0.3-bin-hadoop3.2/kubernetes/tests/py_container_checks.py
    spark-3.0.3-bin-hadoop3.2/kubernetes/tests/pyfiles.py
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/entrypoint.sh
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/R/
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/R/Dockerfile
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/python/
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/bindings/python/Dockerfile
    spark-3.0.3-bin-hadoop3.2/kubernetes/dockerfiles/spark/Dockerfile
    spark-3.0.3-bin-hadoop3.2/jars/
    spark-3.0.3-bin-hadoop3.2/jars/hive-vector-code-gen-2.3.7.jar
    spark-3.0.3-bin-hadoop3.2/jars/guice-servlet-4.0.jar
    spark-3.0.3-bin-hadoop3.2/jars/kerb-crypto-1.0.1.jar
```

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop3.2"
```

```
[3] import findspark
    findspark.init()
    from pyspark.sql import SparkSession  #Connect spark code on top of spark engine
    spark = SparkSession.builder.master("local[4]").getOrCreate()
```

```
[4] import pyspark
    from pyspark.context import SparkContext

    from pyspark import SparkConf
    sc = SparkContext.getOrCreate(SparkConf().setMaster("local[13]"))
```

- Collecting data

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop3.2"
```

```
[3] import findspark
    findspark.init()
    from pyspark.sql import SparkSession  #Connect spark code on top of spark engine
    spark = SparkSession.builder.master("local[4]").getOrCreate()
```

```
[4] import pyspark
    from pyspark.context import SparkContext

    from pyspark import SparkConf
    sc = SparkContext.getOrCreate(SparkConf().setMaster("local[13]"))
```

- Reduce by key

```
origin_airport_passenger = data.map(lambda orig:(orig[0],int(orig[6]))).reduceByKey(lambda a,b:a+b)
origin_airport_passenger.collect()
```

```
[('SEA', 17966),
 ('PDX', 354295),
 ('LMT', 13235),
 ('SFO', 47513),
 ('LAX', 3588),
 ('EAT', 53),
 ('EKO', 703),
 ('LWS', 26),
 ('ACV', 4),
 ('MHK', 21),
 ('EUG', 1259),
 ('MFR', 151),
 ('YKM', 25),
 ('SLE', 5),
 ('GEG', 33),
 ('RDD', 545),
 ('AST', 6),
 ('CLM', 25),
 ('PDT', 2),
 ('SJC', 5993),
 ('PUW', 9)]
```

- Date of passenger boarding

```
[8] date_passenger = data.map(lambda date:(date[10],int(date[6]))).reduceByKey(lambda a,b:a+b)
    date_passenger.collect()
```

```
[('01-10-2008', 21),
 ('01-12-1990', 4762),
 ('01-03-1990', 6471),
 ('01-01-1990', 4127),
 ('01-09-1990', 2696),
 ('01-06-1990', 4254),
 ('01-08-1990', 5024),
 ('01-05-1990', 3660),
 ('01-07-1990', 4699),
 ('01-12-1991', 6017),
 ('01-09-1991', 5181),
 ('01-05-1991', 5549),
 ('01-08-1991', 6954),
 ('01-07-1991', 6818),
```

- Summary description

```
[12] df.describe().show()

+-------+--------------+-------------------+------------+----------------+-----------------+-----------------+-----------------+-----------------+----------+-----------------+----------------------
|summary|Origin_airport|Destination_airport| Origin_city|Destination_city|       Passengers|            Seats|          Flights|         Distance|  Fly_date|Origin_population|Destination_populati
+-------+--------------+-------------------+------------+----------------+-----------------+-----------------+-----------------+-----------------+----------+-----------------+----------------------
|  count|         74217|              74217|       74217|           74217|            74217|            74217|            74217|            74217|     74217|            74217|                 742
|   mean|          null|               null|        null|            null|2258.6351375021895|3580.2349057493566|32.32231159976825|678.1930959214196|      null|6368695.889540132|     3481464.5136020
| stddev|          null|               null|        null|            null| 3778.589652499597| 5608.353583669643|46.22818655994732|546.3547071652132|      null|8834578.317738028|     4214724.7497633
|    min|           ABE|                ACT|Aberdeen, SD|      Akron, OH|                0|                0|                0|                0|1990-01-01|            13150|                 155
|    max|           YUM|                YUM|    Yuma, AZ|        Yuma, AZ|            73505|            89790|              730|             4862|2009-12-01|         38139592|              102405
+-------+--------------+-------------------+------------+----------------+-----------------+-----------------+-----------------+-----------------+----------+-----------------+----------------------
```

```
[13] df.describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
Origin_airport	74217	None	None	ABE	YUM
Destination_airport	74217	None	None	ACT	YUM
Origin_city	74217	None	None	Aberdeen, SD	Yuma, AZ
Destination_city	74217	None	None	Akron, OH	Yuma, AZ
Passengers	74217	2258.6351375021895	3778.589652499597	0	73505
Seats	74217	3580.2349057493566	5608.353583669643	0	89790
Flights	74217	32.32231159976825	46.22818655994732	0	730
Distance	74217	678.1930959214196	546.3547071652132	0	4862
Fly_date	74217	None	None	1990-01-01	2009-12-01
Origin_population	74217	6368695.889540132	8834578.317738028	13150	38139592
Destination_population	74217	3481464.513602005	4214724.749763398	15501	10240512
Org_airport_lat	74217	37.6069108368573	6.121091111834443	20.8985996246338	NA
Org_airport_long	74217	-96.5018484470655	19.5090601237591	-100.2860031	NA
Dest_airport_lat	74217	34.63539489725733	7.936275533346021	19.721399307251	NA
Dest_airport_long	74217	-97.72908151310604	20.061416283940822	-103.2170029	NA

- Total passengers

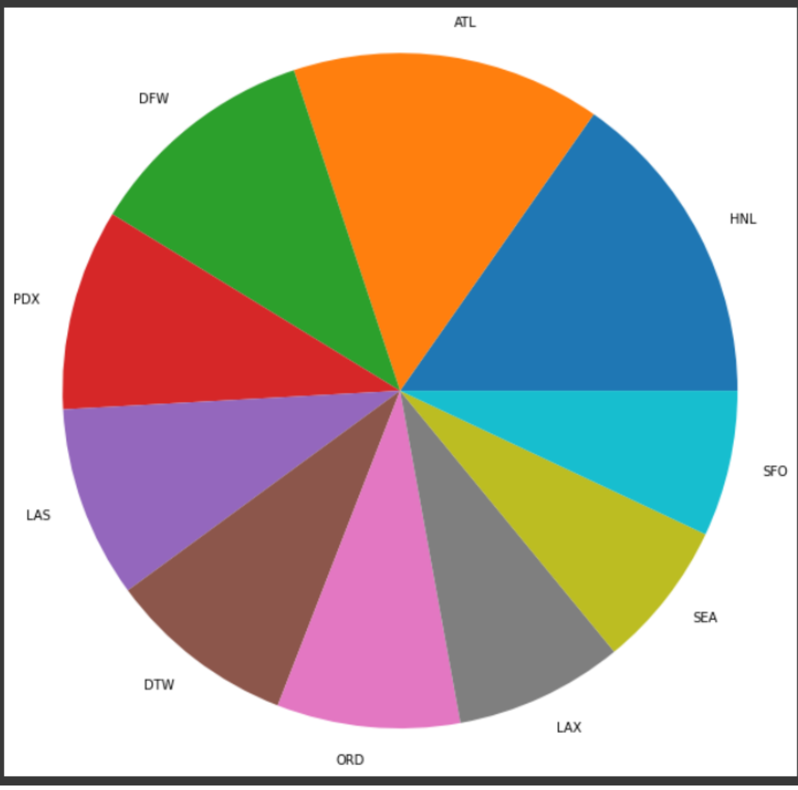
```
from pyspark.sql import SQLContext
from pyspark.sql import functions as F
from pyspark.sql.functions import col
from pyspark.sql.functions import desc

sqlContext=SQLContext(sc)
```

```
[17] airportAgg_DF = df.groupBy("Origin_airport").agg(F.sum("Passengers"))
     airportAgg_DF.show(10)
```

```
+--------------+--------------+
|Origin_airport|sum(Passengers)|
+--------------+--------------+
|           BGM|           483|
|           MOR|             0|
|           MSY|       1270467|
|           RDG|          2367|
|           GEG|        110034|
|           DRT|             0|
|           SNA|        584609|
|           GTF|         16623|
|           GRB|         15250|
|           FOD|          3465|
+--------------+--------------+
only showing top 10 rows
```

- Flights and airport plot

```
plt.pie(org_Airport['Flights'], labels =org_Airport['Origin_Airport'], radius=3)
plt.show()
```



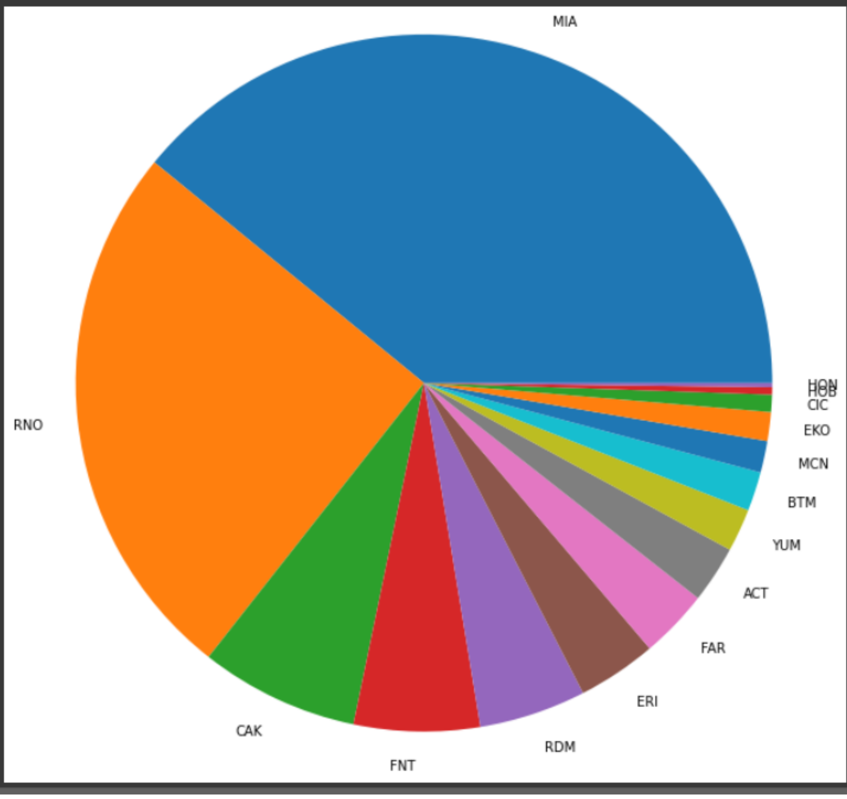
- Total flights from each airport

```
MostFlightsByAirports = sqlContext.sql("""with destination as (select Destination_airport as Airport, sum(Flights) as Out_Flights
                                        from df group by Destination_airport),
                                        origin as (select Origin_airport as Airport, sum(Flights) as In_Flights
                                        from df group by Origin_airport)
                                        select origin.Airport, (destination.Out_Flights+origin.In_Flights) as Total_Flights
                                        from origin, destination
                                        where origin.Airport = destination.Airport
                                        order by (origin.In_Flights + destination.Out_Flights) DESC
                                        limit 15""")
MostFlightsByAirports.show()
```

```
+-------+-------------+
|Airport|Total_Flights|
+-------+-------------+
|    MIA|       856236|
|    RNO|       554210|
|    CAK|       161283|
|    FNT|       127994|
|    RDM|       108638|
|    ERI|        80363|
|    FAR|        69904|
|    ACT|        58052|
|    YUM|        43595|
|    BTM|        39758|
|    MCN|        32077|
|    EKO|        29286|
|    CIC|        17499|
|    HOB|         7419|
|    HON|         4225|
+-------+-------------+
```

- Frequency of flights from each airport

```
MostFlightsbyAirports=MostFlightsByAirports.toPandas()
plt.pie(MostFlightsbyAirports['Total_Flights'], labels =MostFlightsbyAirports['Airport'], radius=3)
plt.show()
```



- Most passengers from an airport

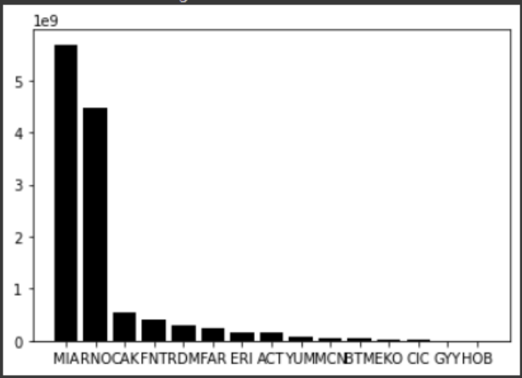
```
MostPassengersByAirports = sqlContext.sql("""with destination as (select Destination_airport as Airport, sum(Passengers*Flights) as Out_Passengers
                                        from df group by Destination_airport),
                                        origin as (select Origin_airport as Airport, sum(Passengers) as In_Passengers
                                        from df group by Origin_airport)
                                        select origin.Airport, (destination.Out_Passengers+origin.In_Passengers) as Total_Passengers
                                        from origin, destination
                                        where origin.Airport = destination.Airport
                                        order by (origin.In_Passengers + destination.Out_Passengers) DESC
                                        limit 15""")
MostPassengersByAirports.show()
```

```
+-------+----------------+
|Airport|Total_Passengers|
+-------+----------------+
|    MIA|      5688348832|
|    RNO|      4476354160|
|    CAK|       554628683|
|    FNT|       395171158|
|    RDM|       291413882|
|    FAR|       248798054|
|    ERI|       163961609|
|    ACT|       156731548|
|    YUM|        82148810|
|    MCN|        45773130|
|    BTM|        42962984|
|    EKO|        31973581|
|    CIC|        21865312|
|    GYY|          997510|
|    HOB|          694844|
+-------+----------------+
```

- Bar graph for above situation

```
most_pass=MostPassengersByAirports.toPandas()
plt.bar(most_pass['Airport'],most_pass['Total_Passengers'], color ='black')
```

<BarContainer object of 15 artists>



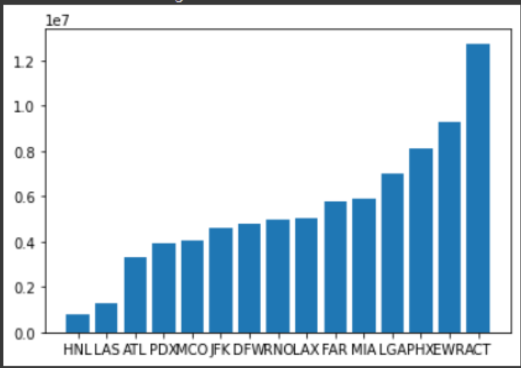
- Distance query

```
distanceQuery = sqlContext.sql("""with table1 as
                                (select least(Origin_airport, Destination_airport) as Airport1,
                                greatest(Destination_airport, Origin_airport) as Airport2,
                                sum(Flights) as Flights,
                                sum(Passengers) as Passengers,
                                sum(Seats) as Seats
                                from df
                                group by least(Origin_airport, Destination_airport), greatest(Destination_airport, Origin_airpor
                                order by 1,2)
                                select t.*, (Passengers*100/Seats) as Occupancy_Rate
                                from table1 t
                                order by Flights DESC, Seats DESC, Passengers DESC, Occupancy_Rate DESC
                                limit 15;""")
distanceQuery = distanceQuery.filter((col("Occupancy_Rate").isNotNull()) & (col("Occupancy_Rate")<=100.0))
distanceQuery.show(15)
```

```
+--------+--------+-------+----------+--------+-----------------+
|Airport1|Airport2|Flights|Passengers|   Seats|   Occupancy_Rate|
+--------+--------+-------+----------+--------+-----------------+
|     HNL|     ITO| 170678|  12753744|19303636| 66.06912811658901|
|     LAS|     RNO|  97167|   9272647|13212128| 70.18284261248454|
|     ATL|     MIA|  70316|   8113561|12725929| 63.75613913923298|
|     PDX|     RDM|  70150|   1268215| 2081640| 60.92383889625488|
|     MCO|     MIA|  67725|   4601137| 9482059| 48.52466115218224|
|     JFK|     MIA|  53557|   6990116|11431809| 61.14619304783696|
|     DFW|     MIA|  53538|   5783852| 9481801| 60.99950842672189|
|     RNO|     SEA|  53126|   4974938| 7117472| 69.89754227343641|
|     LAX|     RNO|  51782|   3950703| 6192810|  63.7949977473877|
|     FAR|     MSP|  51605|   3325836| 5509051|60.370397732749254|
|     MIA|     ORD|  48152|   5910919| 8767493| 67.41857678129882|
|     LGA|     MIA|  47927|   5064249| 7655239| 66.15402863320139|
|     PHX|     RNO|  45217|   4072080| 6041460| 67.40225044939469|
|     EWR|     MIA|  42867|   4799983| 7027564| 68.30223104336012|
|     ACT|     DFW|  41921|    762544| 1352914|56.363079988824126|
+--------+--------+-------+----------+--------+-----------------+
```

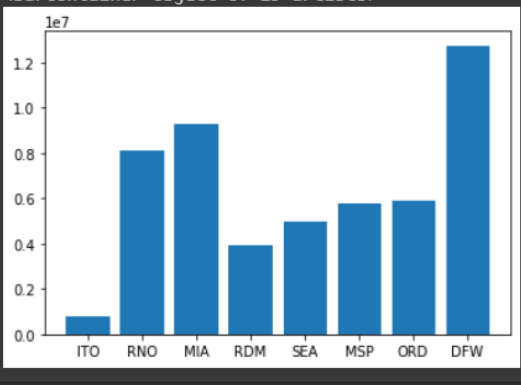
```
[27] plt.bar(distQuery['Airport1'],sorted(distQuery['Passengers']))
```

<BarContainer object of 15 artists>



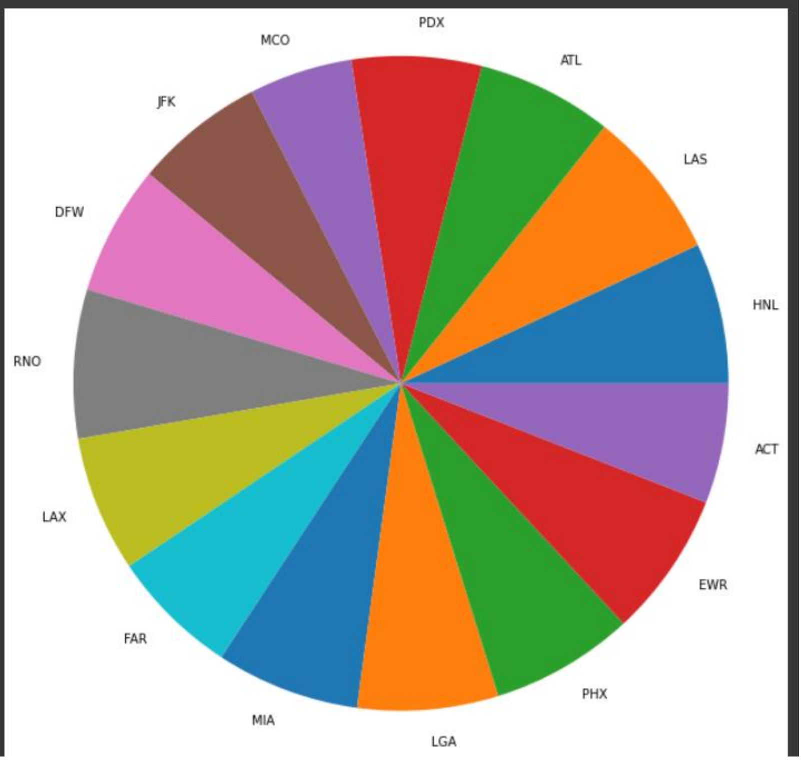

```
plt.bar(distQuery['Airport2'],sorted(distQuery['Passengers']))
```

<BarContainer object of 15 artists>



- Occupancy rate

```
plt.pie(distQuery['Occupancy_Rate'], labels =distQuery['Airport1'], radius=3)
plt.show()
```



```
distanceQuery = sqlContext.sql("""with table1 as
                                    (select least(Origin_airport, Destination_airport) as Airport1,
                                    greatest(Destination_airport, Origin_airport) as Airport2,
                                    mean(Distance) as Distance,
                                    sum(Flights) as Flights
                                    from df
                                    group by least(Origin_airport, Destination_airport), greatest(Destination_airport, Origin_airport)
                                    order by 1,2)
                                    select t.*
                                    from table1 t
                                    where Flights>0
                                    order by Distance DESC
                                    limit 15;""")
# distanceQuery = distanceQuery.filter((col("Occupancy_Rate").isNotNull()) & (col("Occupancy_Rate")<=100.0))
distanceQuery.show(15)
```

```
+--------+--------+--------+-------+
|Airport1|Airport2|Distance|Flights|
+--------+--------+--------+-------+
|     HIK|     MIA|  4862.0|      1|
|     HNL|     MIA|  4862.0|      1|
|     ITO|     STL|  4036.0|      1|
|     ANC|     MIA|  4004.0|    280|
|     FAR|     HNL|  3807.0|      6|
|     ACT|     HIK|  3780.0|      1|
|     EDF|     WRB|  3495.0|      2|
|     ANC|     DOV|  3412.0|      1|
|     DOV|     EIL|  3293.0|      1|
|     ANC|     CNW|  3114.0|      1|
|     AZA|     ITO|  2823.0|      1|
|     DQF|     ITO|  2823.0|      1|
|     ITO|     PHX|  2804.0|      6|
|     HIK|     YUM|  2762.0|      1|
|     HNL|     YUM|  2762.0|      2|
+--------+--------+--------+-------+
```

2. *Linear regression to predict number of people:*

- Installing necessary libraries

```
[1] !apt-get install openjdk-8-jdk-headless -qq > /dev/null
    !wget -q https://downloads.apache.org/spark/spark-3.0.3/spark-3.0.3-bin-hadoop3.2.tgz
    !tar -xvf spark-3.0.3-bin-hadoop3.2.tgz
    !pip install -q findspark
```

```
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-respond.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-sbt-launch-lib.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-antlr.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-dagre-d3.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-pyrolite.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-sorttable.js.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-janino.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-protobuf.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-jquery.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-scopt.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-netlib.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-d3.min.js.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-graphlib-dot.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-AnchorJS.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-datatables.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-pmml-model.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-paranamer.txt
spark-3.0.3-bin-hadoop3.2/licenses/LICENSE-jakarta-ws-rs-api
```

- Importing necessary packages

```
[2] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

    import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop3.2"

[4] import findspark
    findspark.init()
    from pyspark.sql import SparkSession  #Connect spark code on top of spark engine
    spark = SparkSession.builder.master("local[4]").getOrCreate()

[5] import pyspark
    from pyspark.context import SparkContext

    from pyspark import SparkConf
    sc = SparkContext.getOrCreate(SparkConf().setMaster("local[4]"))
```

- Reading file

```
[6] df = spark.read.csv("/content/airports_reduced.csv", header=True, inferSchema=True)
    df.registerTempTable('df')

    df.printSchema()

    root
     |-- Origin_airport: string (nullable = true)
     |-- Destination_airport: string (nullable = true)
     |-- Origin_city: string (nullable = true)
     |-- Destination_city: string (nullable = true)
     |-- Passengers: integer (nullable = true)
     |-- Seats: integer (nullable = true)
     |-- Flights: integer (nullable = true)
     |-- Distance: integer (nullable = true)
     |-- Fly_date: string (nullable = true)
     |-- Origin_population: integer (nullable = true)
     |-- Destination_population: integer (nullable = true)
     |-- Org_airport_lat: string (nullable = true)
     |-- Org_airport_long: string (nullable = true)
     |-- Dest_airport_lat: string (nullable = true)
     |-- Dest_airport_long: string (nullable = true)
```

- Show

```
[8] df.show(10)
```

```
----+----------+-----+-------+--------+----------+-----------------+----------------------+----------------+-----------------+---------------+--
_city|Passengers|Seats|Flights|Distance|  Fly_date|Origin_population|Destination_population| Org_airport_lat| Org_airport_long|Dest_airport_lat|De
----+----------+-----+-------+--------+----------+-----------------+----------------------+----------------+-----------------+---------------+--
s, IA|        21|   30|      1|     254|2008-10-01|           122049|                 86219| 39.140998840332|-96.6707992553711|             NA|
d, OR|        41|  396|     22|     103|1990-11-01|           284093|                 76034|44.1245994567871| -123.21199798584|     44.2541008|
d, OR|        88|  342|     19|     103|1990-12-01|           284093|                 76034|44.1245994567871| -123.21199798584|     44.2541008|
d, OR|        11|   72|      4|     103|1990-10-01|           284093|                 76034|44.1245994567871| -123.21199798584|     44.2541008|
d, OR|         0|   18|      1|     156|1990-02-01|           147300|                 76034|42.3741989135742|-122.873001098633|     44.2541008|
d, OR|        11|   18|      1|     156|1990-03-01|           147300|                 76034|42.3741989135742|-122.873001098633|     44.2541008|
d, OR|         2|   72|      4|     156|1990-01-01|           147300|                 76034|42.3741989135742|-122.873001098633|     44.2541008|
d, OR|         7|   18|      1|     156|1990-09-01|           147300|                 76034|42.3741989135742|-122.873001098633|     44.2541008|
d, OR|         7|   36|      2|     156|1990-11-01|           147300|                 76034|42.3741989135742|-122.873001098633|     44.2541008|
d, OR|         8|   18|      1|     228|1990-02-01|          5154164|                 76034|47.4490013122559| -122.30899810791|     44.2541008|
----+----------+-----+-------+--------+----------+-----------------+----------------------+----------------+-----------------+---------------+--
```

- Vector Assembler

```
[9] from pyspark.ml.linalg import Vectors
    from pyspark.ml.feature import VectorAssembler

    assembler=VectorAssembler(inputCols=['Seats',
     'Flights',
     'Distance',
     'Origin_population',
     'Destination_population'],outputCol='features')
    output=assembler.transform(df)
    output.select('features','Passengers').show(5)
```

```
+--------------------+----------+
|            features|Passengers|
+--------------------+----------+
|[30.0,1.0,254.0,1...|        21|
|[396.0,22.0,103.0...|        41|
|[342.0,19.0,103.0...|        88|
|[72.0,4.0,103.0,2...|        11|
|[18.0,1.0,156.0,1...|         0|
+--------------------+----------+
only showing top 5 rows
```

- Passenger summary

```
[10] final_data=output.select('features','Passengers')
     train_data,test_data=final_data.randomSplit([0.7,0.3])
     train_data.describe().show()
```

```
+-------+-----------------+
|summary|       Passengers|
+-------+-----------------+
|  count|            73191|
|   mean|2371.5234250112717|
| stddev| 4143.377649477439|
|    min|                0|
|    max|            73505|
+-------+-----------------+
```

```
[11] test_data.describe().show()
```

```
+-------+-----------------+
|summary|       Passengers|
+-------+-----------------+
|  count|            31665|
|   mean|  2308.10159482078|
| stddev|3974.6511497245097|
|    min|                0|
|    max|            51594|
+-------+-----------------+
```

- Calculating R-square error and testing model on unlabelled data

```
[12] from pyspark.ml.regression import LinearRegression

     plane_lr=LinearRegression(featuresCol='features',labelCol='Passengers')

     trained_plane_model=plane_lr.fit(train_data)

     plane_results=trained_plane_model.evaluate(train_data)

     print('Rsquared Error :',plane_results.r2)

     Rsquared Error : 0.9475630686936911

[13] #testing Model on unlabeled data
     unlabeled_data=test_data.select('features')
     unlabeled_data.show(5)

     +--------------------+
     |            features|
     +--------------------+
     |[0.0,0.0,70.0,240...|
     |[0.0,0.0,95.0,113...|
     |[0.0,0.0,109.0,23...|
     |[0.0,0.0,113.0,15...|
     |[0.0,0.0,113.0,17...|
     +--------------------+
     only showing top 5 rows
```

CONCLUSION:

The results demonstrated that big data presents a plenty of promising opportunities for the aviation industry. Big data provides airlines with modern insights that can invent new business models. Through big data analytics, airlines can improve services and product development, support customer experience and loyalty, obtain predictive maintenance, provide safe flights, personalize customer experience, provide differential price for each customer and groups, anticipate future supply and demand, cut costs, support decision makers, evaluate current routes and open new ones, develop creativity in customer services and technology, and affective management for cabin crew and staff. Big data also assists airline companies to optimize the process of booking, ordering and luggage tracking. Furthermore, big data can help airlines to have a better understanding of their customers. They can identify each customer's behaviour, keep track of their preferences, and predict future demands. The airlines can also promote their businesses and marketing campaigns, enabling them to achieve success and superiority in a strong competitive market. With a huge stock of data at their disposal, big data technology can change the way airlines do work. By giving a priority to data collection and analysis, they can react to customer needs, desires, and market trends with accurate and fast.

REFERENCES:

1. Mac Con Iomaire, M., Afifi, M., and Healy, J. (2020). Chefs' perspectives of failures in foodservice kitchens, Part 1**: A phenomenological exploration of the concepts, types, and causes of food production failure**. Journal of Foodservice Business Research, 24(2), pp. 177–214.
2. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., and Byers, A. H. (2011)**. Big data: The next frontier for innovation, competition, and productivity**.
3. Mariani, M., Baggio, R., Fuchs, M. and Höpken, W. (2018). **Business intelligence and big data in hospitality and tourism: A systematic literature review.** International Journal of Contemporary Hospitality Management, 30 (12). pp. 3514 - 3554.
4. Maroufkhani, P., Wagner, R. Ismail, W. Baroto, M. and Nourani, M. (2019). **Big Data analytics and firm performance: A systematic review.** Information,10, pp.1- 21.
5. Marr, B. (2018). How much data do we create every day? **The Mind-Blowing Stats Everyone Should Read**. Available at: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-createevery-day-the-mind-blowing-stats-everyone-should-read/?sh=3435068160ba, Accessed on: 24 September 2021.
6. Mayer-Schönberger, V., and Cukier, K. (2013). **Big data: A revolution that will transform how we live, work, and think**. New York: Houghton Mifflin Harcourt.
7. McAfee, A., and Brynjolfsson, E. (2012). **Big Data: The management revolution.** Harvard Business Review, pp. 61- 68.
8. Mikalef, P., Pappas, I. O., Krogstie, J., and Giannakos, M. (2017). Big data analytics capabilities: **A systematic literature review and research agenda. Information Systems and e-Business Management**, 1–32.
9. Mikalef, P., Pappas, I., Krogstie, J. and Giannakos, M. (2018). Big data analytics capabilities: **A systematic literature review and research agenda. Information System Electronic Business Management**. 16, pp. 547-578.