Claude Bernard University Lyon 1

Data Processing and Analytics (DISS - DPA)

Analyzing Flight Interconnected Data

Students:
LE Thi Hoa - 12310380
TRAN Hai Linh - 12310487

Supervised by:
Prof. Angela Bonifati
Prof. Andrea Mauri

27, November 2023

Link Code: https://github.com/tranhailinh97/Graph-PageRank/tree/master

I. Introduction

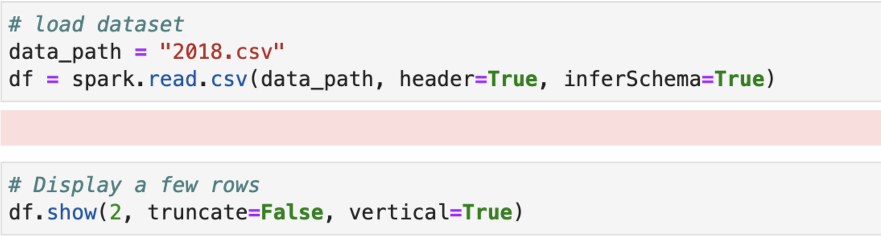
The objective of this project is to use Spark's APIs to analyze the flight interconnected data to find which are the most popular airports.

II. Dataset

The data has been collected and organized concerning flights throughout the year 2018. The dataset contains detailed information about each flight, including date, airline, origin and destination, estimated and actual times, as well as factors influencing the flights.

1. Load Data

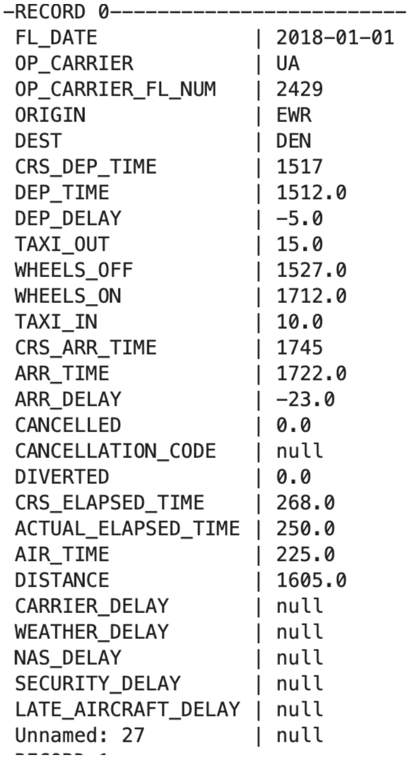
Use spark to read dataset



```python
# load dataset
data_path = "2018.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
```

```python
# Display a few rows
df.show(2, truncate=False, vertical=True)
```

Display several rows of dataset



```
-RECORD 0-------------------------
 FL_DATE             | 2018-01-01
 OP_CARRIER          | UA
 OP_CARRIER_FL_NUM   | 2429
 ORIGIN              | EWR
 DEST                | DEN
 CRS_DEP_TIME        | 1517
 DEP_TIME            | 1512.0
 DEP_DELAY           | -5.0
 TAXI_OUT            | 15.0
 WHEELS_OFF          | 1527.0
 WHEELS_ON           | 1712.0
 TAXI_IN             | 10.0
 CRS_ARR_TIME        | 1745
 ARR_TIME            | 1722.0
 ARR_DELAY           | -23.0
 CANCELLED           | 0.0
 CANCELLATION_CODE   | null
 DIVERTED            | 0.0
 CRS_ELAPSED_TIME    | 268.0
 ACTUAL_ELAPSED_TIME | 250.0
 AIR_TIME            | 225.0
 DISTANCE            | 1605.0
 CARRIER_DELAY       | null
 WEATHER_DELAY       | null
 NAS_DELAY           | null
 SECURITY_DELAY      | null
 LATE_AIRCRAFT_DELAY | null
 Unnamed: 27         | null
```

2. Data structure

Data set with **7213446** flights

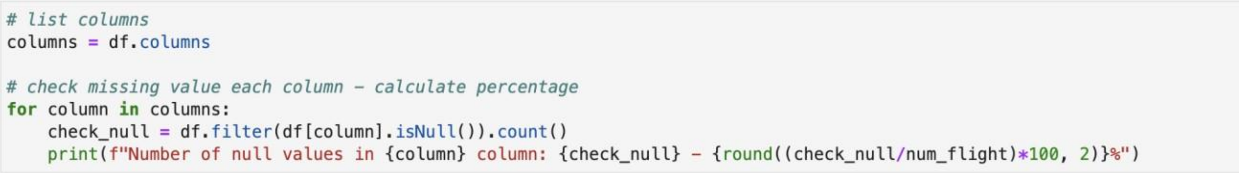
The dataset is structured with the following columns:

- **FL_DATE**: Date of the flight.
- **OP_CARRIER**: Operating Carrier code, indicating the airline that performed the flight.
- **OP_CARRIER_FL_NUM:** Flight number assigned by the operating carrier.
- **ORIGIN:** Three-letter code for the origin airport.
- **DEST:** Three-letter code for the destination airport.
- **CRS_DEP_TIME**: Scheduled departure time.
- **DEP_TIME:** Actual departure time.
- **DEP_DELAY:** Difference in minutes between actual and scheduled departure times.
- **TAXI_OUT:** Time in minutes from departure from the origin gate to wheels off.
- **WHEELS_OFF**: Actual departure time when the aircraft wheels leave the ground.
- **WHEELS_ON:** Actual arrival time when the aircraft wheels touch the ground.
- **TAXI_IN:** Time in minutes from wheels-on to arrival at the destination gate.
- **CRS_ARR_TIME:** Scheduled arrival time.
- **ARR_TIME:** Actual arrival time.
- **ARR_DELAY:** Difference in minutes between actual and scheduled arrival times.
- **CANCELLED:** Indicates whether the flight was canceled (1) or not (0).
- **CANCELLATION_CODE:** Code specifying the reason for cancellation.
- **DIVERTED:** Indicates whether the flight was diverted to another airport (1) or not (0).
- **CRS_ELAPSED_TIME:** Scheduled elapsed time of the flight.
- **ACTUAL_ELAPSED_TIME:** Actual elapsed time of the flight.
- **AIR_TIME:** Time the aircraft spends airborne.
- **DISTANCE:** Distance between airports.
- **CARRIER_DELAY:** Delay attributed to the carrier.
- **WEATHER_DELAY:** Delay attributed to weather conditions.
- **NAS_DELAY:** Delay attributed to the National Airspace System.
- **SECURITY_DELAY:** Delay attributed to security-related issues.
- **LATE_AIRCRAFT_DELAY**: Delay attributed to issues with the aircraft.
- **Unnamed: 27**: An unnamed column (seems to have no specific meaning).

III. Data Preprocessing

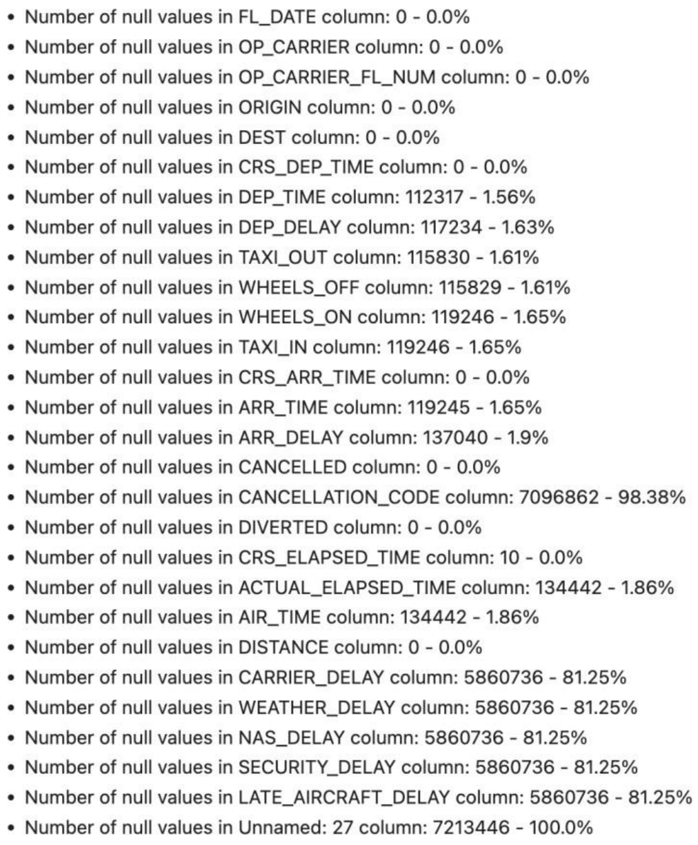
1. Check for missing values

First, we check to see if there are any missing values in the data set. We count the missing values in each column and then calculate the missing percentage



```
# list columns
columns = df.columns

# check missing value each column - calculate percentage
for column in columns:
    check_null = df.filter(df[column].isNull()).count()
    print(f"Number of null values in {column} column: {check_null} - {round((check_null/num_flight)*100, 2)}%")
```

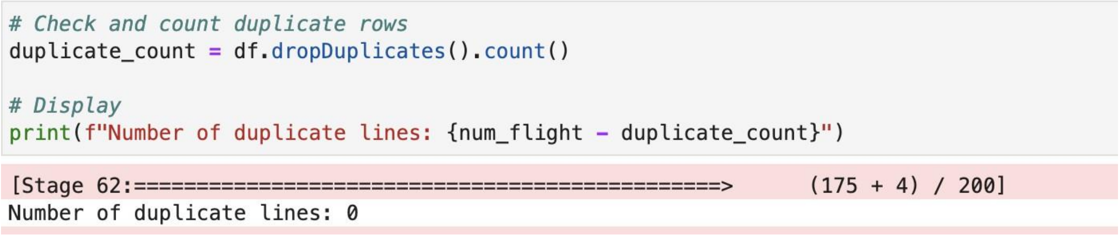
After checking for missing values, we can see that:



- Number of null values in FL_DATE column: 0 - 0.0%
- Number of null values in OP_CARRIER column: 0 - 0.0%
- Number of null values in OP_CARRIER_FL_NUM column: 0 - 0.0%
- Number of null values in ORIGIN column: 0 - 0.0%
- Number of null values in DEST column: 0 - 0.0%
- Number of null values in CRS_DEP_TIME column: 0 - 0.0%
- Number of null values in DEP_TIME column: 112317 - 1.56%
- Number of null values in DEP_DELAY column: 117234 - 1.63%
- Number of null values in TAXI_OUT column: 115830 - 1.61%
- Number of null values in WHEELS_OFF column: 115829 - 1.61%
- Number of null values in WHEELS_ON column: 119246 - 1.65%
- Number of null values in TAXI_IN column: 119246 - 1.65%
- Number of null values in CRS_ARR_TIME column: 0 - 0.0%
- Number of null values in ARR_TIME column: 119245 - 1.65%
- Number of null values in ARR_DELAY column: 137040 - 1.9%
- Number of null values in CANCELLED column: 0 - 0.0%
- Number of null values in CANCELLATION_CODE column: 7096862 - 98.38%
- Number of null values in DIVERTED column: 0 - 0.0%
- Number of null values in CRS_ELAPSED_TIME column: 10 - 0.0%
- Number of null values in ACTUAL_ELAPSED_TIME column: 134442 - 1.86%
- Number of null values in AIR_TIME column: 134442 - 1.86%
- Number of null values in DISTANCE column: 0 - 0.0%
- Number of null values in CARRIER_DELAY column: 5860736 - 81.25%
- Number of null values in WEATHER_DELAY column: 5860736 - 81.25%
- Number of null values in NAS_DELAY column: 5860736 - 81.25%
- Number of null values in SECURITY_DELAY column: 5860736 - 81.25%
- Number of null values in LATE_AIRCRAFT_DELAY column: 5860736 - 81.25%
- Number of null values in Unnamed: 27 column: 7213446 - 100.0%

2. Check for duplicate values

We count the duplicate values



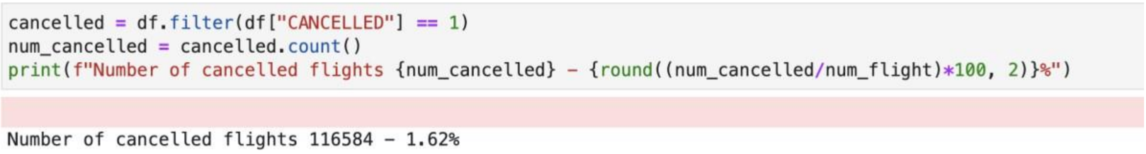
```
# Check and count duplicate rows
duplicate_count = df.dropDuplicates().count()

# Display
print(f"Number of duplicate lines: {num_flight - duplicate_count}")
```

```
[Stage 62:==============================================>       (175 + 4) / 200]
Number of duplicate lines: 0
```

We can see there is no duplicate value

3. Check for canceled flights

We count the canceled flights then calculate the missing percentage



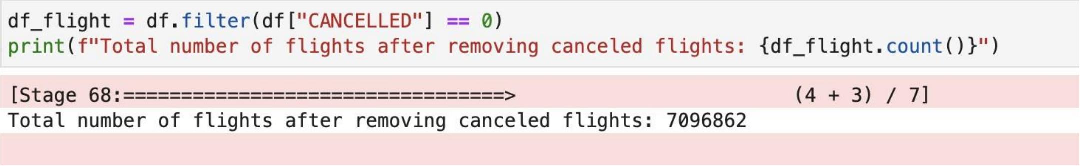
```
cancelled = df.filter(df["CANCELLED"] == 1)
num_cancelled = cancelled.count()
print(f"Number of cancelled flights {num_cancelled} - {round((num_cancelled/num_flight)*100, 2)}%")
```

```
Number of cancelled flights 116584 - 1.62%
```

There are 116584 canceled flights, accounts for 1.62% of the total data

4. Data Cleaning

We removed canceled flights



```
df_flight = df.filter(df["CANCELLED"] == 0)
print(f"Total number of flights after removing canceled flights: {df_flight.count()}")
```

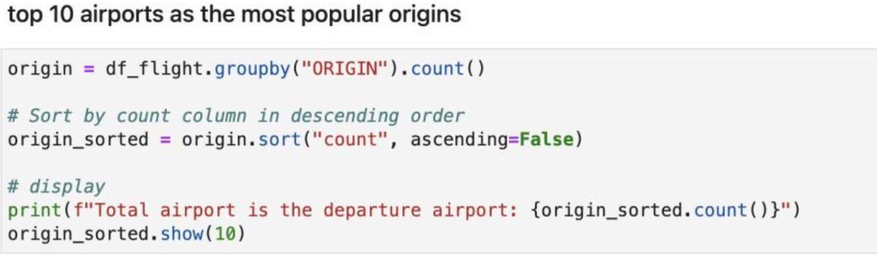
```
[Stage 68:================================>                          (4 + 3) / 7]
Total number of flights after removing canceled flights: 7096862
```

After removing canceled flights, there are 7096862 flights

5. Data Exploring

- ***Top 10 airports as the most popular origins***

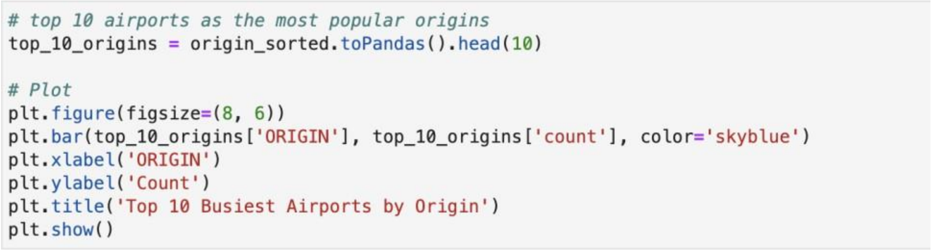
This analysis provides insights into the busiest departure airports, helping us understand the distribution of flights and identify key hubs in the dataset.



top 10 airports as the most popular origins

```
origin = df_flight.groupby("ORIGIN").count()

# Sort by count column in descending order
origin_sorted = origin.sort("count", ascending=False)

# display
print(f"Total airport is the departure airport: {origin_sorted.count()}")
origin_sorted.show(10)
```

There are 385 airports that is the departure airport
Visualize the top 10 most popular origins



```
# top 10 airports as the most popular origins
top_10_origins = origin_sorted.toPandas().head(10)

# Plot
plt.figure(figsize=(8, 6))
plt.bar(top_10_origins['ORIGIN'], top_10_origins['count'], color='skyblue')
plt.xlabel('ORIGIN')
plt.ylabel('Count')
plt.title('Top 10 Busiest Airports by Origin')
plt.show()
```

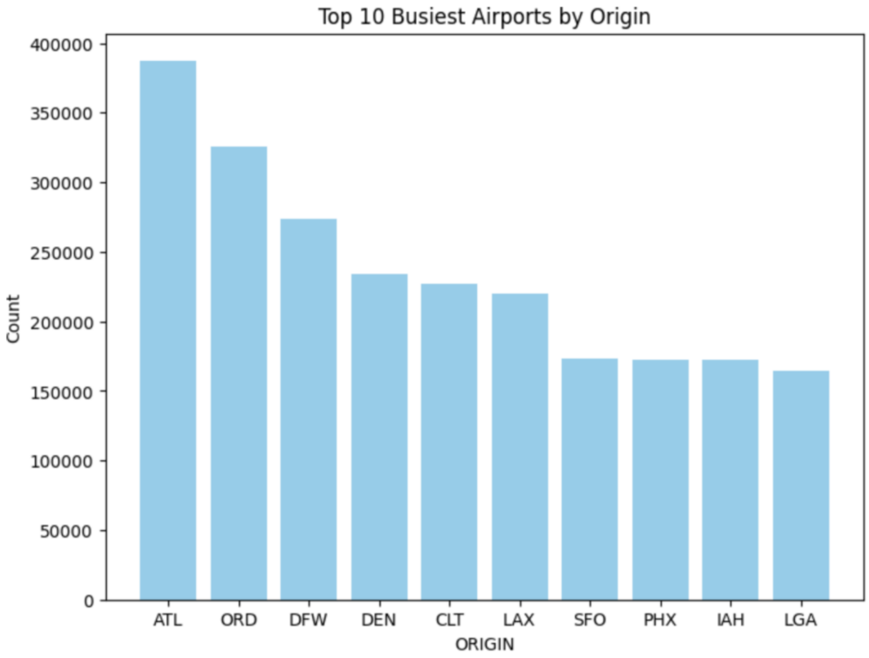


Figure 1: Top 10 most popular origins

- ***Top 10 airports as the most popular destinations***

  Similar to the top 10 most popular origins, we take the top 10 most popular destinations

  There are 385 airports that is the destination airport

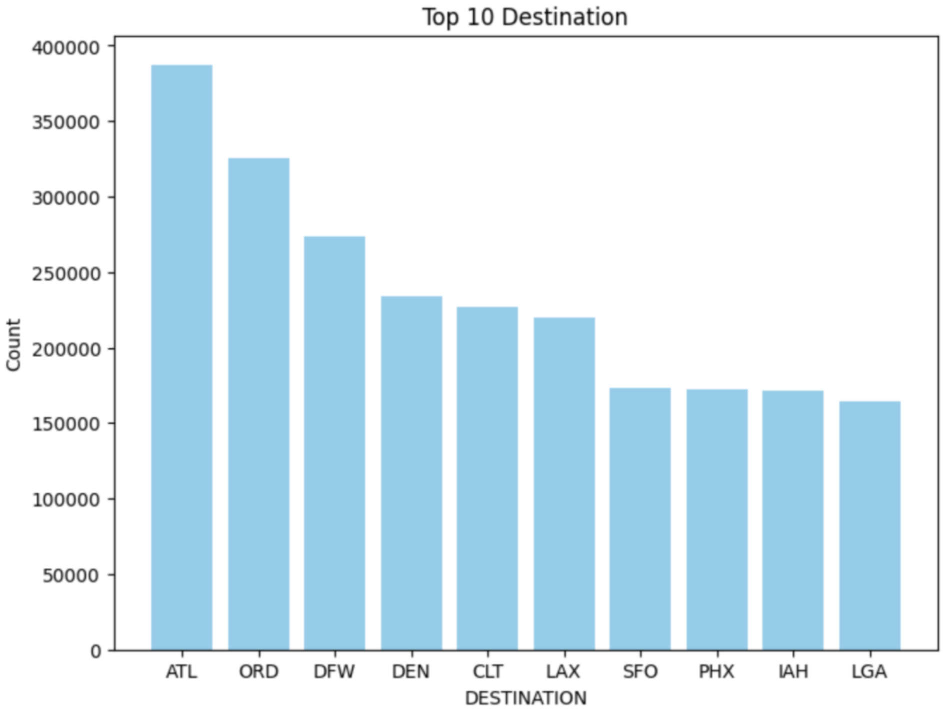


Figure 1: Top 10 the most popular destinations

IV. Build A Graph

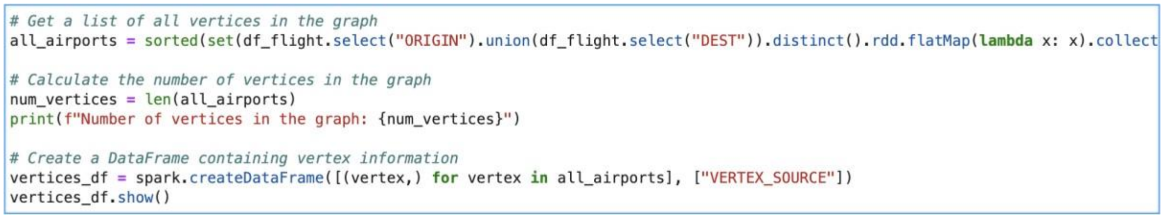
In the process of constructing a graph based on flight data, we define the vertices, edges, and computation of edge weights. This graph serves as a representation of the connections between airports and the frequency of flights between them.

1. **Vertices**

   The vertices in this graph are the airports. Each unique airport code from the dataset is considered a vertex. For instance, if there are N unique airports, the graph will have N vertices, each representing a specific airport.

   Solution:

   - Extracts unique airport codes from both the "ORIGIN" and "DEST" columns in the DataFrame (df_flight).
   - The airports are collected into a set to remove duplicates
   - Then sorted to create a list of all unique vertices (airports) in the graph.



```
# Get a list of all vertices in the graph
all_airports = sorted(set(df_flight.select("ORIGIN").union(df_flight.select("DEST")).distinct().rdd.flatMap(lambda x: x).collect

# Calculate the number of vertices in the graph
num_vertices = len(all_airports)
print(f"Number of vertices in the graph: {num_vertices}")

# Create a DataFrame containing vertex information
vertices_df = spark.createDataFrame([(vertex,) for vertex in all_airports], ["VERTEX_SOURCE"])
vertices_df.show()
```

There are 358 vertices in the graph

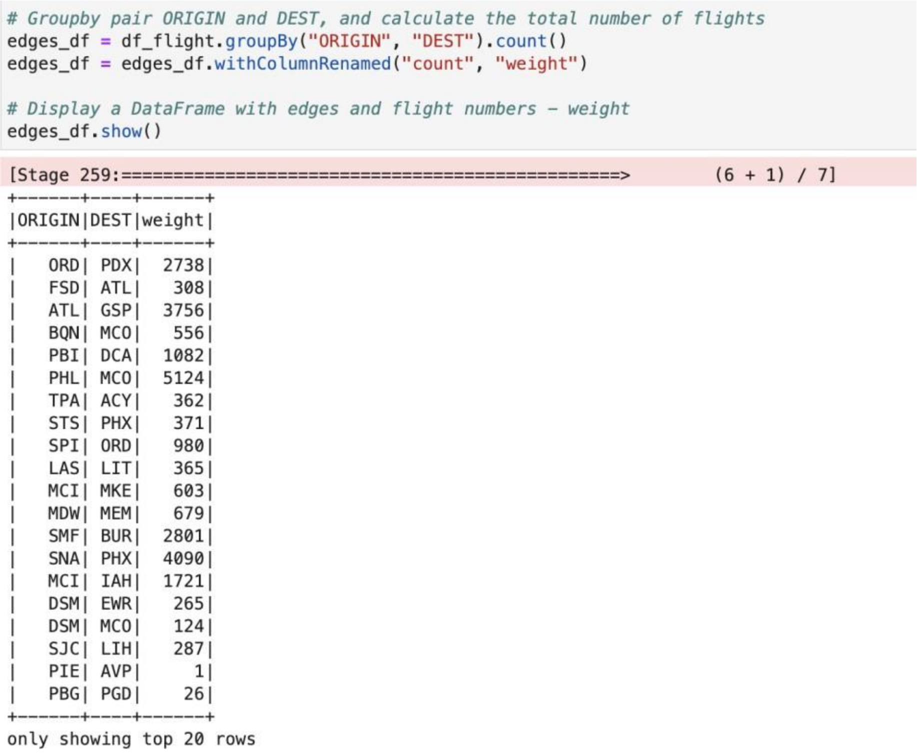
2. **Edges and weight**

   The edges in the graph represent the routes between airports. If there is a direct flight from airport A to airport B, then there is an edge connecting vertex A to vertex B in the graph. The presence of an edge indicates a direct connection or route between two airports.

   The weight of an edge, representing the connection between two airports, is computed as the total number of flights or the frequency of flights between those two airports. For example, if there were 100 flights from airport A to airport B in a given time period, the weight of the edge between A and B would be 100.

Solution:

- Group by pair ORIGIN and DEST, and calculate the total number of flights
- Renames the count column to "weight"



```
# Groupby pair ORIGIN and DEST, and calculate the total number of flights
edges_df = df_flight.groupBy("ORIGIN", "DEST").count()
edges_df = edges_df.withColumnRenamed("count", "weight")

# Display a DataFrame with edges and flight numbers - weight
edges_df.show()
```

```
[Stage 259:=============================================>          (6 + 1) / 7]
+------+----+------+
|ORIGIN|DEST|weight|
+------+----+------+
|   ORD| PDX|  2738|
|   FSD| ATL|   308|
|   ATL| GSP|  3756|
|   BQN| MCO|   556|
|   PBI| DCA|  1082|
|   PHL| MCO|  5124|
|   TPA| ACY|   362|
|   STS| PHX|   371|
|   SPI| ORD|   980|
|   LAS| LIT|   365|
|   MCI| MKE|   603|
|   MDW| MEM|   679|
|   SMF| BUR|  2801|
|   SNA| PHX|  4090|
|   MCI| IAH|  1721|
|   DSM| EWR|   265|
|   DSM| MCO|   124|
|   SJC| LIH|   287|
|   PIE| AVP|     1|
|   PBG| PGD|    26|
+------+----+------+
only showing top 20 rows
```

V. Implement The PageRank Algorithm

1. PageRank Formula

The PageRank algorithm calculates the importance of vertices (nodes) in a graph based on the structure of links between them. The formula for PageRank of a vertex A is expressed as follows:

$$PageRank(A) = \frac{1-d}{N} + d \times \left(\frac{PageRank(B)}{L(B)} + \frac{PageRank(C)}{L(C)} + \dots \right)$$

In there:

- PageRank(A) is the PageRank of page A.
- N is the total number of nodes in the graph.
- d is the damping factor.
- PageRank(B),PageRank(C),… are the PageRanks of pages that link to page A.

- L(B),L(C),… are the number of outbound links on pages B, C, etc.

2. Damping Factor Explanation

The damping factor is a parameter used in the PageRank algorithm to model the probability that a user will continue clicking on links rather than jumping to a new page. It introduces a level of randomness to the model, simulating the behavior of a web surfer who, with a certain probability, might decide to navigate to a random page instead of following links.

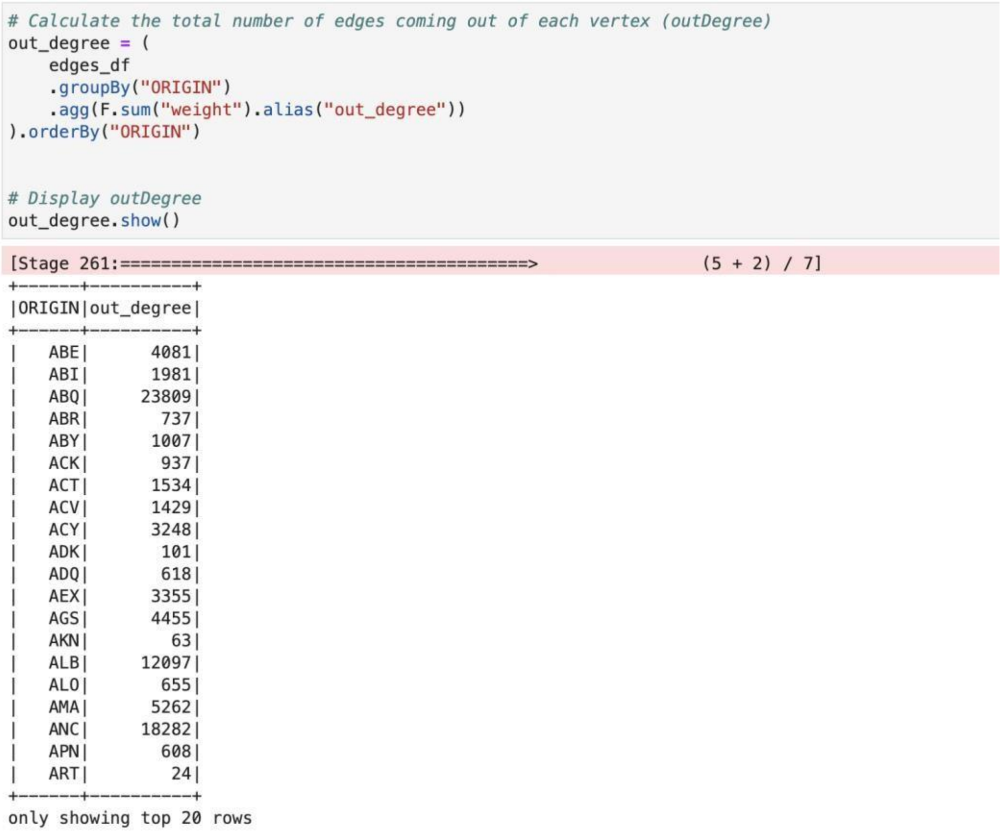
In the context of the PageRank algorithm, the damping factor is typically denoted by the symbol d. The standard value for d is often set to 0.85, but it can vary depending on the specific application. The remaining probability, 1−d, is distributed evenly among all the nodes in the graph, reflecting the surfer's chance of jumping to any page at random.

3. Steps in PageRank Algorithm

- **It starts by calculating the total number of edges coming out of each vertex (outDegree)**
    - ○ The outDegree is calculated for each airport, representing the total number of flights leaving each airport.
    - ○ This helps quantify the level of "power" of each vertex in transmitting its rank score to other vertices.

Solution:

- Calculates the outDegree by grouping the edges_df DataFrame by "ORIGIN”
- Aggregating the sum of weights (flights) for each origin airport.

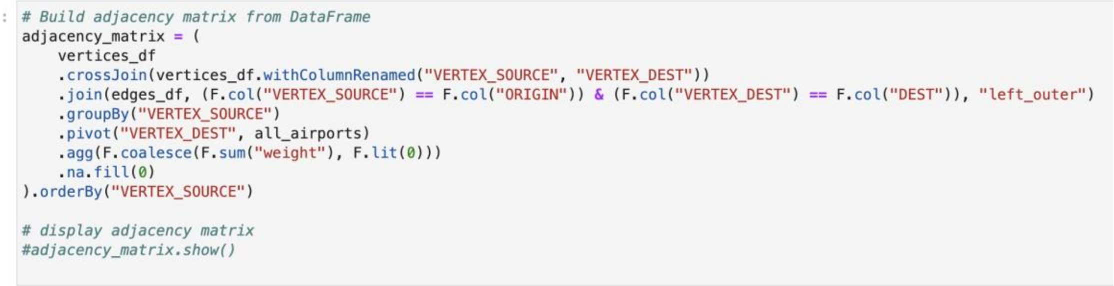


```
# Calculate the total number of edges coming out of each vertex (outDegree)
out_degree = (
    edges_df
    .groupBy("ORIGIN")
    .agg(F.sum("weight").alias("out_degree"))
).orderBy("ORIGIN")


# Display outDegree
out_degree.show()
```

```
[Stage 261:=======================================>                (5 + 2) / 7]
+------+----------+
|ORIGIN|out_degree|
+------+----------+
|   ABE|      4081|
|   ABI|      1981|
|   ABQ|     23809|
|   ABR|       737|
|   ABY|      1007|
|   ACK|       937|
|   ACT|      1534|
|   ACV|      1429|
|   ACY|      3248|
|   ADK|       101|
|   ADQ|       618|
|   AEX|      3355|
|   AGS|      4455|
|   AKN|        63|
|   ALB|     12097|
|   ALO|       655|
|   AMA|      5262|
|   ANC|     18282|
|   APN|       608|
|   ART|        24|
+------+----------+
only showing top 20 rows
```

- **Constructing an adjacency matrix**
  Adjacency matrix - representing the connections (flights) between each pair of airport

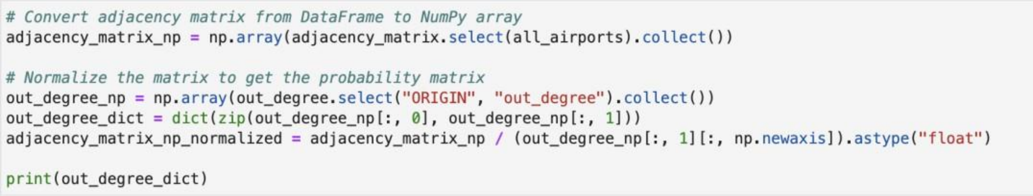
  Solution:
  - It does a cross join on the vertices_df to create all possible combinations of source and destination airports.
  - Then joins with the edges_df to get the weights (flight counts) for each airport pair
  - The result is a DataFrame (adjacency_matrix) with airports as rows, airports as columns, and flight counts as values.



```
# Build adjacency matrix from DataFrame
adjacency_matrix = (
    vertices_df
    .crossJoin(vertices_df.withColumnRenamed("VERTEX_SOURCE", "VERTEX_DEST"))
    .join(edges_df, (F.col("VERTEX_SOURCE") == F.col("ORIGIN")) & (F.col("VERTEX_DEST") == F.col("DEST")), "left_outer")
    .groupBy("VERTEX_SOURCE")
    .pivot("VERTEX_DEST", all_airports)
    .agg(F.coalesce(F.sum("weight"), F.lit(0)))
    .na.fill(0)
).orderBy("VERTEX_SOURCE")

# display adjacency matrix
#adjacency_matrix.show()
```

- **Normalizing it to a probability matrix**
  Solution:

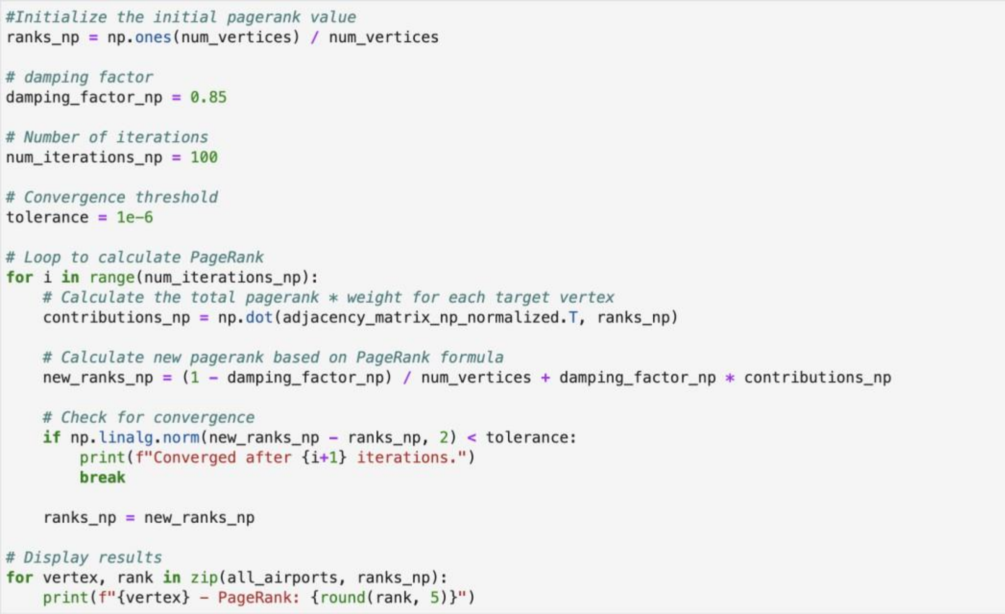
  - Convert adjacency matrix from DataFrame to NumPy array
  - The adjacency matrix is normalized to create a probability matrix.
  - Each element in the matrix is divided by the corresponding outDegree to convert the counts into probabilities.



```
# Convert adjacency matrix from DataFrame to NumPy array
adjacency_matrix_np = np.array(adjacency_matrix.select(all_airports).collect())

# Normalize the matrix to get the probability matrix
out_degree_np = np.array(out_degree.select("ORIGIN", "out_degree").collect())
out_degree_dict = dict(zip(out_degree_np[:, 0], out_degree_np[:, 1]))
adjacency_matrix_np_normalized = adjacency_matrix_np / (out_degree_np[:, 1][:, np.newaxis]).astype("float")

print(out_degree_dict)
```

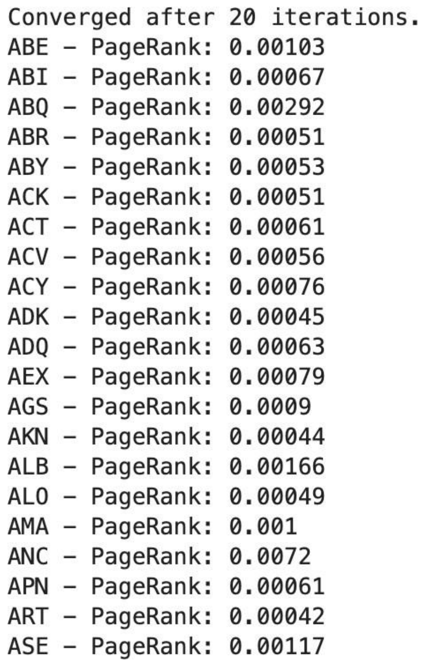
- **Then iteratively calculating PageRank values until convergence**
  Solution:

  - The initial PageRank values for all vertices are set. All vertices are given equal initial importance.
  - Iteratively calculates the PageRank values for each airport.
  - It uses the PageRank formula, considering the normalized adjacency matrix and damping factor.
  - The loop continues until convergence (change in PageRank values is below a tolerance level), or until the specified number of iterations is reached.



```
#Initialize the initial pagerank value
ranks_np = np.ones(num_vertices) / num_vertices

# damping factor
damping_factor_np = 0.85

# Number of iterations
num_iterations_np = 100

# Convergence threshold
tolerance = 1e-6

# Loop to calculate PageRank
for i in range(num_iterations_np):
    # Calculate the total pagerank * weight for each target vertex
    contributions_np = np.dot(adjacency_matrix_np_normalized.T, ranks_np)

    # Calculate new pagerank based on PageRank formula
    new_ranks_np = (1 - damping_factor_np) / num_vertices + damping_factor_np * contributions_np

    # Check for convergence
    if np.linalg.norm(new_ranks_np - ranks_np, 2) < tolerance:
        print(f"Converged after {i+1} iterations.")
        break

    ranks_np = new_ranks_np

# Display results
for vertex, rank in zip(all_airports, ranks_np):
    print(f"{vertex} - PageRank: {round(rank, 5)}")
```

With a convergence threshold of 1e-6, the algorithm converges after 20 iterations. We can see the PageRank values of some airports:



```
Converged after 20 iterations.
ABE – PageRank: 0.00103
ABI – PageRank: 0.00067
ABQ – PageRank: 0.00292
ABR – PageRank: 0.00051
ABY – PageRank: 0.00053
ACK – PageRank: 0.00051
ACT – PageRank: 0.00061
ACV – PageRank: 0.00056
ACY – PageRank: 0.00076
ADK – PageRank: 0.00045
ADQ – PageRank: 0.00063
AEX – PageRank: 0.00079
AGS – PageRank: 0.0009
AKN – PageRank: 0.00044
ALB – PageRank: 0.00166
ALO – PageRank: 0.00049
AMA – PageRank: 0.001
ANC – PageRank: 0.0072
APN – PageRank: 0.00061
ART – PageRank: 0.00042
ASE – PageRank: 0.00117
```

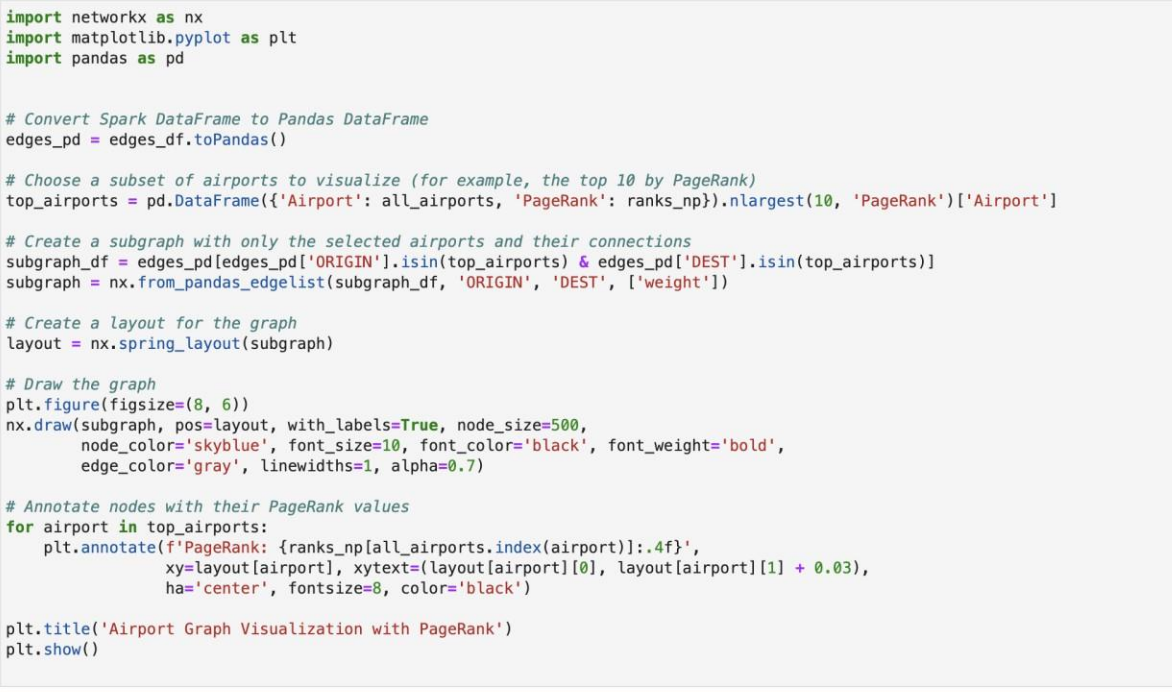
VI. Graph Visualization

To gain insights into the airport network based on the PageRank algorithm, we visualize a subgraph containing the top 10 airports ranked by PageRank.

Use the “networkx” library to create graphs and matplotlib to draw graphs.

Solution:

- Select a subset of airports based on their PageRank values, focusing on the top 10 airports.
- Generate a subgraph containing only the selected airports and their connections.
- Defines the layout, draws the graph with specified visual attributes
- Annotates nodes with PageRank values, and finally displays the graph with a title.



```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd


# Convert Spark DataFrame to Pandas DataFrame
edges_pd = edges_df.toPandas()

# Choose a subset of airports to visualize (for example, the top 10 by PageRank)
top_airports = pd.DataFrame({'Airport': all_airports, 'PageRank': ranks_np}).nlargest(10, 'PageRank')['Airport']

# Create a subgraph with only the selected airports and their connections
subgraph_df = edges_pd[edges_pd['ORIGIN'].isin(top_airports) & edges_pd['DEST'].isin(top_airports)]
subgraph = nx.from_pandas_edgelist(subgraph_df, 'ORIGIN', 'DEST', ['weight'])

# Create a layout for the graph
layout = nx.spring_layout(subgraph)

# Draw the graph
plt.figure(figsize=(8, 6))
nx.draw(subgraph, pos=layout, with_labels=True, node_size=500,
        node_color='skyblue', font_size=10, font_color='black', font_weight='bold',
        edge_color='gray', linewidths=1, alpha=0.7)

# Annotate nodes with their PageRank values
for airport in top_airports:
    plt.annotate(f'PageRank: {ranks_np[all_airports.index(airport)]:.4f}',
                 xy=layout[airport], xytext=(layout[airport][0], layout[airport][1] + 0.03),
                 ha='center', fontsize=8, color='black')

plt.title('Airport Graph Visualization with PageRank')
plt.show()
```

=> Result:

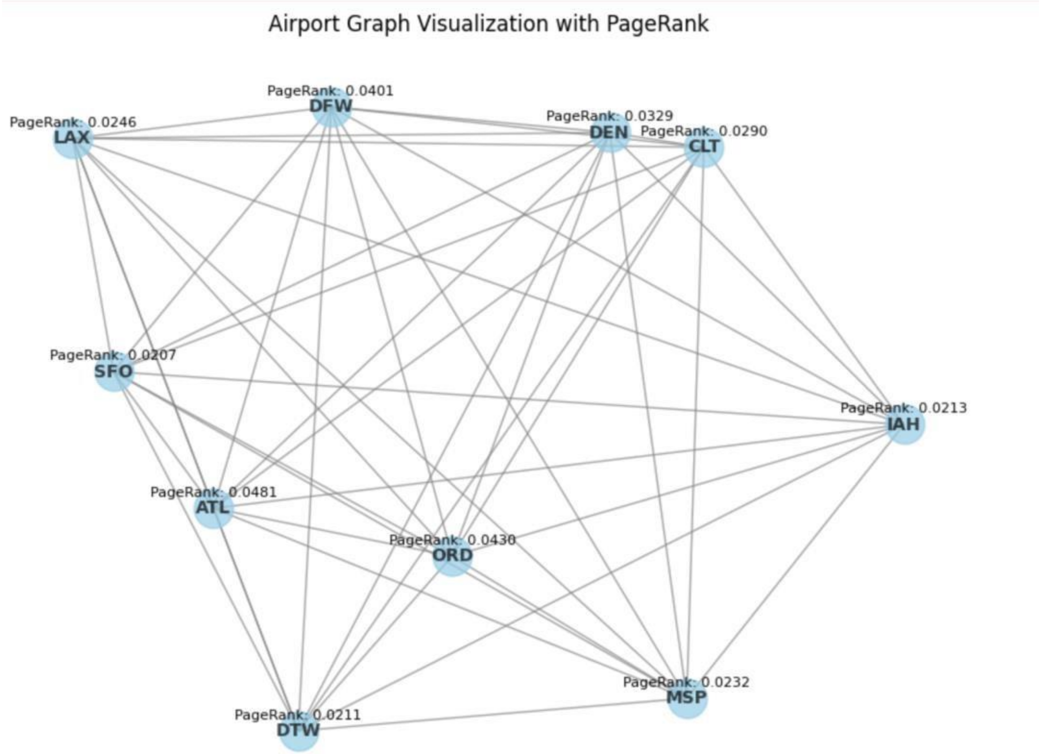


Figure 3: the top 10 airports ranked visualization by PageRank.