Big Data Management

Project nr 3: Flight Interconnected Data Analysis

Team members: Eidi Paas, Pirjo Vainjärv

GitHub link: https://github.com/pirjo2/bdm-projects

Provided dataset

The data used for the analysis was provided in CSV format and includes details on flights, specifically:

- ORIGIN: The airport of departure.
- DEST: The airport of arrival.
- FL_DATE: The flight date.
- DISTANCE: The flight distance in miles.

The dataset was loaded into Spark for analysis.

Data Transformation

We performed data transformation and cleaning using Spark DataFrames, rows with null values were dropped.

We selected the columns ORIGIN, DEST, FL_DATE, and DISTANCE to focus on the necessary flight details for the graph construction.

Graph construction:

- Vertices (airports): Each unique airport is treated as a vertex.
- Edges (flights): Each flight between two airports is treated as an edge.
- The graph is created using GraphFrames with airports as vertices and flights as edges.

After the transformation, there were a total of 296 airports and a total of 6429338 flights in the data.

Queries

Query 1 - Compute different statistics : in-degree, out-degree, total degree and triangle count

- In-degree: The number of flights arriving at an airport.
- Out-degree: The number of flights departing from an airport.
- Total Degree: The sum of in-degree and out-degree for each airport.
- Triangle Count: The number of triangles involving each airport.

First, we computed the in-degree by counting how many flights arrive at each airport (dst). Then, we calculated the out-degree by counting how many flights depart from each airport (src). Finally, we combined both to compute the total degree for each airport by summing in-degree and out-degree. Missing values were handled using coalesce.

Top 10 rows:

id	inDegree	outDegree	totalDegree
ABE	4037	4034	8071
ABI	2490	2490	4980
ABQ	35577	35582	71159
ABY	997	995	1992
ACK	343	342	685
ACT	1052	1053	2105
ACV	3364	3370	6734
ACY	522	522	1044
ADK	103	103	206
ADQ	631	631	1262

Calculating the number of unique triangles in the graph by joining edges to form two-step paths (A → B → C) and then checking if a closing edge (C → A) exists. To avoid duplicate counting, we sorted and filtered nodes (A < B < C). Top 3 Airports by Triangle Count were ATL, ORD and DFW. These airports have a high number of direct connections to many other major airports.

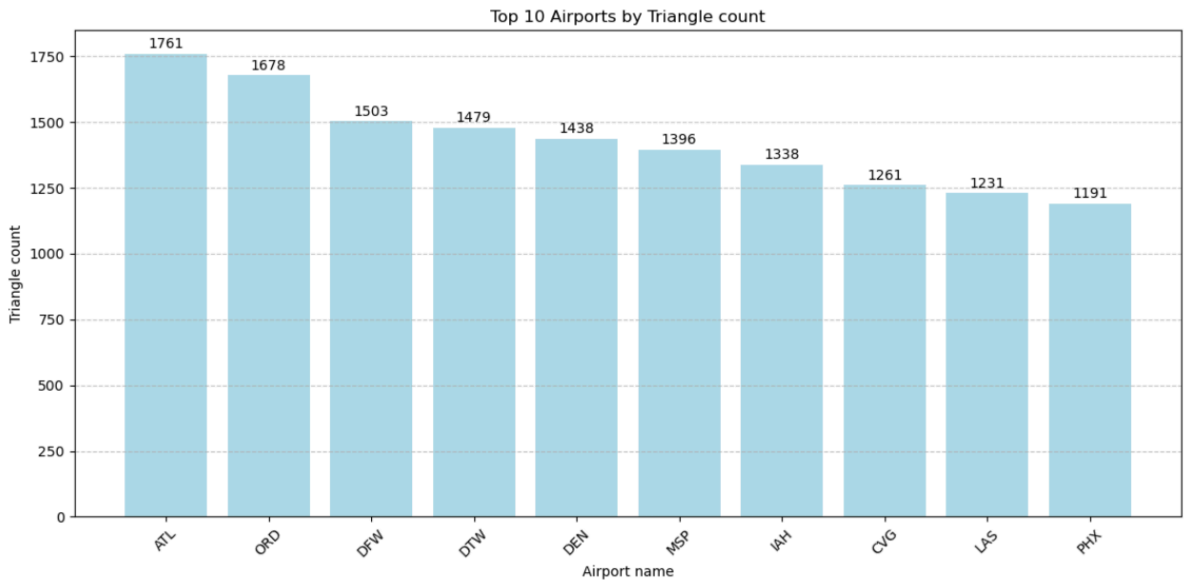
Top 10 rows:

airport	triangle_count
ATL	1761
ORD	1678
DFW	1503
DTW	1479
DEN	1438
MSP	1396
IAH	1338
CVG	1261
LAS	1231
PHX	1191

Validation:

```
+-------+--------------+
|airport|triangle_count|
+-------+--------------+
|ATL    |1761          |
|ORD    |1678          |
|DFW    |1503          |
|DTW    |1479          |
|DEN    |1438          |
|MSP    |1396          |
|IAH    |1338          |
|CVG    |1261          |
|LAS    |1231          |
|PHX    |1191          |
+-------+--------------+
only showing top 10 rows
```

Graph for Query 1 - triangle count:



Query 2 - Compute the total number of triangles in the graph

The code calculates and displays the number of triangles each airport is part of in the graph. Each triangle is a set of three airports that are all interconnected by flights. The result was validated against the triangleCount() function result. The result of Q2 was the total of triangles in the graph is 16015.

Top 5 rows:

```
Total triangles in the graph: 16015
+---+---+---+
|A  |B  |C  |
+---+---+---+
|EWR|GSO|MSP|
|EWR|GSO|IAH|
|EWR|GSO|MEM|
|EWR|GSO|MIA|
|BNA|IAH|SAT|
+---+---+---+
only showing top 5 rows
```

Validation:

```
Total triangles: 16015
```

Query 3 - Compute a centrality measure of your choice natively on Spark using Graphframes

Centrality measures help to identify the most important airports based on their position in the network. For this project, we selected degree centrality, which is computed by the total degree of each airport. We normalized the total degree by dividing it by the total number of airports in the data. The result of degree Centrality was validated, by using the built-in GraphFrames .degrees method. It calculated the total degree for each airport. Then we normalized the degree values (just like in our manual implementation), by dividing it with the total number of airports. This ensures the values are comparable. As of the result the top 1st airport is ATL airport.

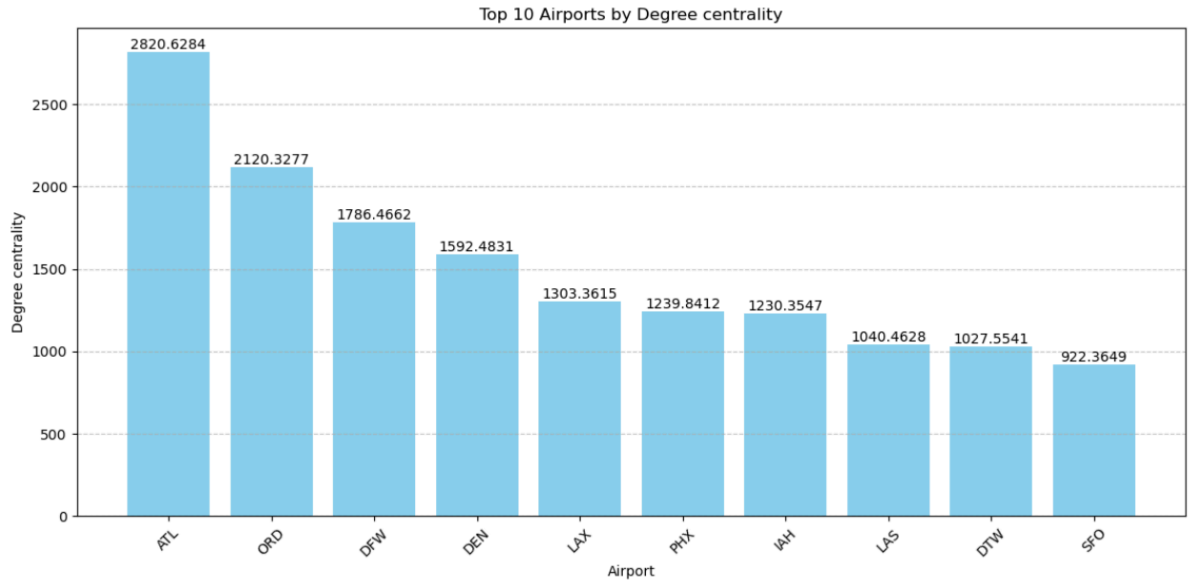
Top 10 rows:

id	inDegree	outDegree	totalDegree	degreeCentrality
ATL	417457	417449	834906	2820.6283783783783
ORD	313769	313848	627617	2120.3277027027025
DFW	264398	264396	528794	1786.4662162162163
DEN	235700	235675	471375	1592.4831081081081
LAX	192916	192879	385795	1303.3614864864865
PHX	183491	183502	366993	1239.8412162162163
IAH	182088	182097	364185	1230.3547297297298
LAS	153984	153993	307977	1040.462837837838
DTW	152075	152081	304156	1027.554054054054
SFO	136532	136488	273020	922.3648648648649

Validation:

```
+---+------+------------------+
|id |degree|degreeCentrality  |
+---+------+------------------+
|ATL|834906|2820.6283783783783|
|ORD|627617|2120.3277027027025|
|DFW|528794|1786.4662162162163|
|DEN|471375|1592.4831081081081|
|LAX|385795|1303.3614864864865|
|PHX|366993|1239.8412162162163|
|IAH|364185|1230.3547297297298|
|LAS|307977|1040.462837837838 |
|DTW|304156|1027.554054054054 |
|SFO|273020|922.3648648648649 |
+---+------+------------------+
only showing top 10 rows
```

Graph for Query 3 - Degree centrality:



Query 4 - Implement the PageRank algorithm natively on Spark using Graphframes

The code first implements the PageRank algorithm manually. It starts by assigning an initial rank of 1.0 to every airport in the graph. Then, it calculates how many outgoing connections each airport has. It simulates how each airport passes part of its rank to connected airports over 10 iterations. If an airport has no outgoing flights, its rank is evenly spread across all airports. The new rank of each airport is calculated using a damping factor of 0.85 to include both the received contributions and a small random teleportation factor. After the iterations, the ranks are scaled and normalized so that the total rank equals the number of airports.

After this, the same PageRank calculation is done using the built-in GraphFrames method. The damping factor and iteration count are kept the same. The results are also scaled and normalized to be comparable with the manual version. Finally, the airports are ranked by importance, and the top ten are displayed. As a result, the most important airport is ATL.

Top 10 rows manual: Validation:

```
Top 10 most important airports:
+---+------------------+
|id |rank              |
+---+------------------+
|ATL|18.91245735574146 |
|ORD|14.282048517055609|
|DFW|12.053169719686384|
|DEN|10.84097874666418 |
|LAX|8.91239097271667  |
|PHX|8.459008616180949 |
|IAH|8.283921450039648 |
|LAS|7.089101469537648 |
|DTW|6.9149802643466725|
|SFO|6.327373831559019 |
+---+------------------+
only showing top 10 rows
```

```
+---+------------------+
|id |PageRank_norm     |
+---+------------------+
|ATL|18.905010413236425|
|ORD|12.9271104227023  |
|DFW|11.735613068886781|
|DEN|9.998483451764704 |
|LAX|7.726101869875706 |
|IAH|7.15930503812503  |
|PHX|7.065371562599523 |
|SLC|7.038754134167847 |
|DTW|7.019968536748603 |
|SFO|5.904248175403023 |
+---+------------------+
only showing top 10 rows
```

Query 5 - Find the group of the most connected airports

The group of the most connected airports are connected via edges between each other. The algorithm logic, Breadth-First Search (BFS), was looked up from this source: [https://cp-algorithms.com/graph /search-for-connected-components.html](https://cp-algorithms.com/graph/search-for-connected-components.html) .

The dataset is cleaned by removing rows with missing origin or destination values. A list of flight connections is created, and reversed edges are added to make the graph undirected. A manual search starting from one airport is used to find all airports connected to it. This helps find the largest group of connected airports. This resulted in 1 big component, marked with ID 0, thus every airport is connected with some path to it.

The same process is repeated using GraphFrames. The built-in connectedComponents() function finds all groups of connected airports, and the largest group is selected based on size. This method approved of our finding of 1 big component, the picture shows only 10 of the airports in that component (eg. when you print it out, you will get all of the airports names).

Manual result: Validation:

```
Number of airports in largest connected component, manual BFS: 296
+---+
|id |
+---+
|ATL|
|CLT|
|BNA|
|HNL|
|SHV|
|CVG|
|VPS|
|SJC|
|MSY|
|TLH|
+---+
only showing top 10 rows
```

```
Largest component ID: 0
Number of airports in the largest component: 296
+---+---------+
|id |component|
+---+---------+
|BGM|0        |
|BNA|0        |
|CLL|0        |
|CLT|0        |
|HNL|0        |
|LCH|0        |
|TVC|0        |
|BLI|0        |
|DLG|0        |
|ERI|0        |
+---+---------+
only showing top 10 rows
```