Inflight: Practical 1

190018035

March 22, 2024

Contents

1 Introduction

This project aims to provide an interactive application, InFlight, that allows users to explore and analyze the *Airline Reporting Carrier On-Time Performance* dataset [1]. The dataset contains information on the on-time performance of domestic flights operated by large air carriers in the United States between 1987 and 2020. The application is divided into three parts, each with a set of features that allow users to explore different aspects of the dataset. The following sections will detail the objectives, accomplishments, and features implemented in each part of the project.

1.1 Project Objectives

The main objectives of the project is to achieve the following:

1. Develop a user-friendly interface that provides visualizations and insights into the dataset.
2. Gain familiarity with PySpark and the PySpark SQL API for processing large datasets.

3. Gain valuable insights into the on-time performance of domestic flights in the United States.

1.2 Accomplishments

The following accomplishments have been achieved in the project:

1. Implemented a web-based application to interact with the dataset. This application utilizes Flask and PySpark as the backend and ReactJS and Tailwind CSS as the frontend.
2. Gained sufficient knowledge of PySpark and the PySpark SQL API by successfully completing all three parts of the project.
3. Developed a set of features that allow users to explore and analyze different aspects of the dataset, such as flight performance metrics, regional performance, and airline performance.

1.2.1 Implemented Features

The project is divided into three parts, each with a set of features that are implemented. The features implemented in each part are shown in Table 1 below.

Part 1	**Part 2**	**Part 3**
Read in and store the dataset using PySpark	Determine a ”performance” metric for airports and display a radar chart to compare airport performance given a list of airports	Create a network graph displaying the flight history connections between US airports.
Display the total number of flights given a year	Chart region and state performance on on a US map	Create a timelapse visualization of flight count density over the years 1987 to 2020
Display the total number of flights given a range of years	Display a table of the best and worst performing states in each US region	
Display the total number of flights given comma-separated list of years		
Display the percentage of flights that departed on time, early and late		
Display the top reason for cancelled flights given a year		
Determine a ”performance” metric and display the top three worst performing airlines		

Table 1: Features implemented in each part of the project.

1.3 Organization of the Report

The remainder of this report is organized in the following manner:

- **Part 1: Core Features** - Details the core features implemented in the project, including reading in and storing the dataset, displaying flight statistics, and analyzing flight timeliness.
- **Part 2: Intermediate Features** - Details the intermediate features implemented in the project, including airport performance metrics, regional performance analysis, and state performance analysis.

- **Part 3: Advanced Features** - Details the advanced features implemented in the project, including airport connnection and flight density visualizations.
- **Evaluation** - Evaluates the project based on the objectives and accomplishments of the project, along with a reflection on lessons learned and experience gained from the project.
- **Conclusion** - Concludes the report with a summary of the project and future work that can be done to improve the application.

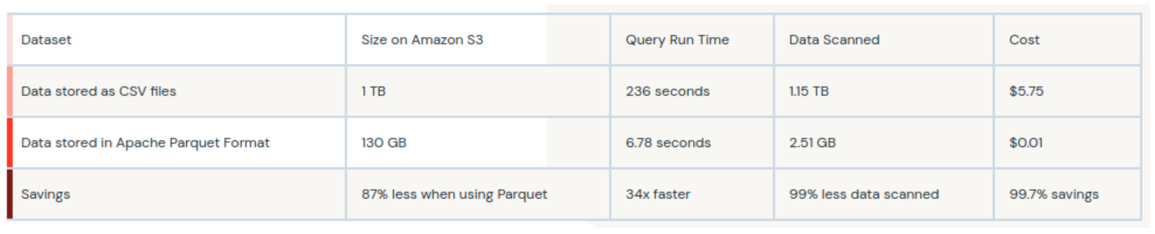
2 Part 1: Core Features

This section details the core features implemented in the project. This includes converting the dataset to a suitable file format and summarizing a few key statistics about the dataset.

2.1 Feature Description

The following section details each of the core features implemented in the project.

Task 1: Read in and store the dataset using PySpark. This task involves reading in the dataset and storing it using PySpark.The dataset is read in from *airline.csv*, which has a file size of over >80GB, using PySpark. The dataset is then stored in in the Parquet file format [2]. Parquet is a columnar storage format the provides efficient storage and encoding of data. Parquet utilizes the algorithm outlined in Dremel, an interactive query system for analysis of read-only nested data [3], to represent nested structures. This algorithm can be broken down into two individual algorithms: the column stripping algorithm, which transforms complex records into a columnar format, and the record assembly algorithm, which reconstructs the original records from these columns when needed. Column stripping allows for better compression than traditional row-oriented storage formats, such as CSV, by exploiting the redundancy in the data found within columns of data, skipping non-relevant data very quickly [4]. Meanwhile, he row assembly algorithm ensures that despite the high compression, the data can be reconstructed quickly when needed, which is essential for large datasets such as the one used in the project, where I/O operations can be a bottleneck. By storing the dataset in Parquet format, the amount of data read from disk is minimized, and the data is stored in a more efficient manner, allowing for faster query times and better performance overall. The benefits of the Parquet format as opposed to CSV are outlined in Figure 1 below.



Dataset	Size on Amazon S3	Query Run Time	Data Scanned	Cost
Data stored as CSV files	1 TB	236 seconds	1.15 TB	$5.75
Data stored in Apache Parquet Format	130 GB	6.78 seconds	2.51 GB	$0.01
Savings	87% less when using Parquet	34x faster	99% less data scanned	99.7% savings

Figure 1: Comparison of Parquet and CSV file formats in terms of file size, query run time, data scanned and cost [4].

Location: Code lines 13-27 in file `backend/scripts/convert.py`

Task 2: Display the total number of flights for a given year. This task allows the user to specify a year and view the total number of flights that occurred in that year. This is done by filtering the airline dataframe based on the specified year and counting the number of rows in the resulting dataframe.

Location: Code lines 26-36 in file `backend/part1/views.py`

Task 3: Display the total number of flights given a range of years. This task allows the user to specify a start year and end year and view the total number of flights that occurred within that range. This is done by filtering the airline dataframe based on the specified range of years and counting the number of rows in the resulting dataframe.

Location: Code lines 52-69 in file `backend/part1/views.py`

Task 4: Display the total number of flights given a list of years. This task allows the user to specify a list of years and view the total number of flights that occurred in each year. This is done by filtering the airline dataframe based on the specified list of years using the `isin()` function and counting the number of rows in the resulting dataframe for each year.

Location: Code lines 83-97 in file `backend/part1/views.py`

Task 4: Display the timeliness percentage of flights for a given year. This task allows the user to specify a year and view the percentage of flights that departed on time, early, and late. The 'DepDelay' column is used to determine the timeliness of flights, with flights that departed on time having a 'DepDelay' of 0, flights that departed early having a 'DepDelay' less than 0, and flights that departed late having a 'DepDelay' greater than 0. Flight with an unknown departure delay are determined by subtracting the number of flights that departed on time, early, and late from the total number of flights. The percentage of flights in each category is then calculated and displayed to the user in the form of a pie chart.

Location: Code lines 115-149 in file `backend/part1/views.py`

Task 5: Display the top reason for cancelled flights for a given year. This task allows the user to specify a year and view the top reason for cancelled flights in that year. This is done by filtering the airline dataframe based on the specified year and creating a new dataframe that groups the data by the 'CancellationCode' column and counts the number of occurrences of each cancellation code. The cancellation code with the highest count is then displayed to the user.

Location: Code lines 163-193 in file `backend/part1/views.py`

Task 6: Display the top 3 most punctual airports for a given year This task allows the user to specify a year and view the top 3 airports with the most punctual flights in that year. This is done by filtering the airline dataframe based on the specified year and creating a new dataframe aggregates this to data to calculate the median departure delay (using the 'DepDelay' column) for each airport. The median is used here as it is less affected by extreme values than the mean, making it more robust to outliers. The top 3 airports with the lowest median departure delay are then displayed to the user.

Location: Code lines 207-238 in file `backend/part1/views.py`

Task 7: Display the top 3 worst performing airlines This task allows the user to view the top 3 worst performing airlines of the twentieth century based on a performance metric. The performance metric takes into account three factors:

- The percentage of flights that were delayed when departing.
- The percentage of flights that were delayed when arriving.
- The percentage of flights that were cancelled.

The dataframe is first filtered to include only flights that occurred in the twentieth century. The performance metric is then calculated for each airline by aggregating the data based on the 'DOT_ID_Reporting_Airline' column. Finally, a composite percentage score is calculated for each airline based on the three factors mentioned above by taking the average of the three percentages. The top 3 airlines with the highest composite percentage score are then displayed to the user.

Location: Code lines 247-268 in file `backend/part1/views.py`

3 Part 2: Intermediate Features

This section details the intermediate features implemented in the project. These features include determining a "performance" metric for airports and displaying a radar chart to compare airport performance given a list of airports, charting region and state performance on a US map, and displaying a table of the best and worst performing states in each US region.

3.1 Feature Description

The following section details each of the intermediate features implemented in the project.

Task 1: Compare the performance of airports across the dataset This task allows the user to choose a list of airports and compare their performance based on a performance metric. The performance metric is calculated based on the following factors, with a short justification for each:

- **The percentage of flights that were delayed when departing**: Indicates the efficiency of the airport in managing departures.
- **The percentage of flights that were delayed when arriving**: Indicates the efficiency of the airport in managing arrivals.
- **The percentage of flights that were cancelled**: Indicates the reliability of the airport in terms of flight operations.
- **The percentage of flights that were diverted**: Indicates the ability of the airport to handle unexpected situations.
- **The mean taxi in time**: Indicates the efficiency of the airport in handling incoming flights.
- **The mean taxi out time**: Indicates the efficiency of the airport in handling outgoing flights.

These metrics were determined to have the most significant impact on the overall performance of an airport. To compare the performance of airports, a radar chart is displayed, with each axis representing one of the performance metrics. Given the seven metrics, the radar chart will have seven axes, each representing one of the metrics. This chart is especially useful for comparing the performance of multiple airports across different metrics. Each airport can be evaluated with respect to the others based on how far they extend along each axis. Additionally, a radar chart can provide a good estimate of the overall performance of the airport based on the distribution of the metrics. Smaller values (and therefore shorter axes) indicate better performance, allowing the user to quickly identify the best performing airports. To avoid cluttering the chart, only five airports can be selected at a time.

Location: Code lines 65-102 in file `backend/part2/views.py`

Task 2: Chart and analyze region and state performance This task allows the user to view the performance of regions and states on a US map. The performance is calculated based on the same performance metrics used in Task 1, with an additional composite percentage score. The performance of each region and state is displayed using a color gradient, with blue indicating better performance and yellow indicating worse performance. This visualization allows the user to quickly identify regions and states that are performing well or poorly based on the selected performance metric. The user can select a performance metric from a dropdown menu, along with a year to view the performance for that year. The user can also hover over a region to view the average performance score for that region.

Analysis: From the implementation described above, we can analyze and make a few key observations, answering the following questions:

1. *Are there any specific regions, as defined by the United States Census Bureau, of the continental United States that had the best/performance across the years?* This question can be answered by observing the consistency of the performance of regions across the years. If a region consistently performs well, it indicates that the region has a good infrastructure for air travel. When experimenting with different years using the composite score, it seems like the West and Midwest region perform the best, with the West performing slightly better after the 2000s. When determining the worst performing regions, the South and Northeast regions seem to be in contention, with the South performing slightly worse.
2. *Are there any specific states in the various regions that are having an effect on the overall performance of the region?* When investigating each region throughout the years, there are a few states that seem to consistently perform well or poorly. For example, North Dakota and South Dakota seem to perform well in the Midwest region, while New Jersey and Pennsylvania seem to perform poorly in the Northeast region. In the West region, Montana strikes as a good performer throughout the years, while Nevada seems to perform poorly. Finally, in the South region, Georgia interestingly stands out for performing poorly until around the mid-2010s, where it becomes one of the best performers in the region.
3. *Are there any trends of performance that you can identify as you move between all the years of the dataset?* To answer this question, a few key historical events can be considered. For example, the 9/11 attacks in 2001 had a significant impact on air travel, leading to increased security measures and delays [5]. This event can be observed in the performance metrics of airports and regions during that time. Additionally, the financial crisis of 2008 had a similar impact on air travel, leading to decreased demand and increased cancellations [6]. Figures 2

and 3 show the performance of regions in the year 2001, where the impact of the 9/11 attacks can be observed, and the year 2008, where the impact of the financial crisis can be observed, showing an accordance with the impact of travel from historical events.

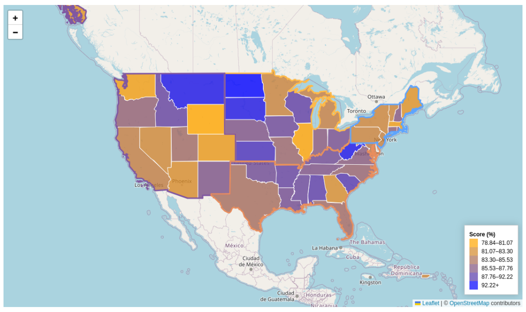


Figure 2: Airport Performance in 2001 during 9/11 attacks

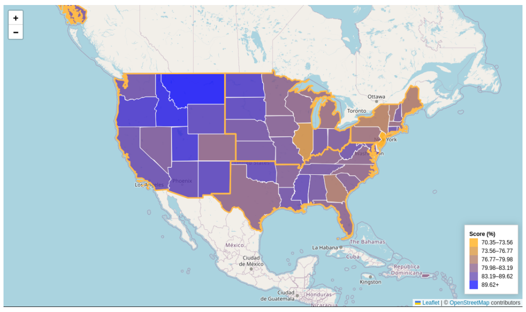


Figure 3: Airport Performance in 2008 during financial crisis

Location:

- Main logic: Code lines 118-145 in file `backend/part2/views.py`
- Single metric score logic: Code lines 16-57 in file `backend/part2/helper.py`
- Composite score logic: Code lines 72-116 in file `backend/part2/helper.py`
- Region score logic: Code lines 169-176 in file `backend/part2/helper.py`

4 Part 3: Advanced Features

This section details the advanced features implemented in the project. These features offer interesting visualizations that provide insights into the dataset.

4.1 Feature Description

The following section details each of the advanced features implemented in the project.

Extension 1: Create a network graph displaying the flight history connections between US airports One interesting visualization that can be created is a network graph that displays the flight history connections between US airports. This graph can help users understand the connectivity between airports and identify the most connected airports in the US. Additionally, the graph can provide information on how frequently flights occur between airports and the most common routes taken by airlines. Finally, the graph provides insight into the overall structure of the US air travel network, highlighting key airports that serve as hubs for domestic flights. To limit the complexity of the graph, only the top 500 most common flight connections are displayed. The graph is interative, displaying moving particles that represent the flow of flights between airports and allowing users to move around the graph move nodes around to explore the connections between airports.

From the graph, we can make a few key observations:

- **Identifying hubs:** The graph can help identify airports that serve as hubs for domestic flights. Hubs are airports that have a high volume of flights and serve as transfer points for passengers traveling to different destinations. Some prominent hubs include Chicago O'Hare International Airport (ORD), Los Angeles International Airport (LAX), Hartsfield-Jackson Atlanta International Airport (ATL), and Dallas/Fort Worth International Airport (DFW). It is also notable to see that some of these hubs, whilst not having a high volume of flight, are still integral to the network, such as Honolulu International Airport (HNL) and Baltimore/Washington International Thurgood Marshall Airport (BWI), which serve as the primary hubs for flights to Hawaii and the East Coast, respectively.
- **Identifying common routes:** The graph can help identify common routes taken by airlines. Figure 4 demonstrates one of the most common routes: the route from Los Angeles International Airport (LAX) to San Francisco International Airport (SFO). This route is one of the busiest routes in the US, with multiple airlines operating flights between the two airports. Other common routes included Chicago O'Hare International Airport (ORD) to New York's LaGuardia Airport (LGA) and Huoston's William P. Hobby Airport (HOU) to Dallas Love Field Airport (DAL).
- **Identifying state connectivity:** The graph can help identify the connectivity between different states in the US. For example, the graph can shows that the main contact between Hawaii and the mainland is through Los Angeles International Airport (LAX). Additionally, there seems to be a number of flights between Florida and New York.

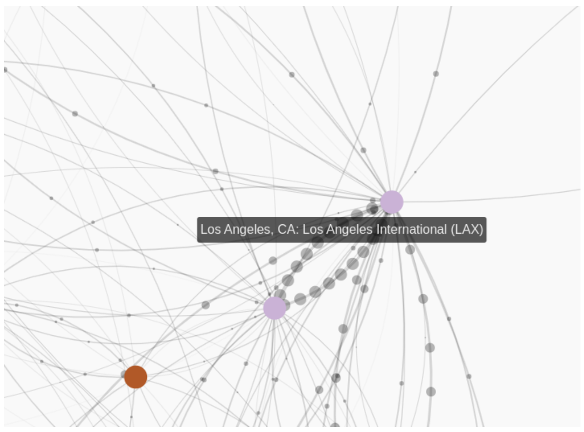


Figure 4: Network graph displaying flight connections between Los Angeles International Airport (LAX) and San Francisco International Airport (SFO).

Location: Code lines 21-67 in file `backend/part3/views.py`

Extension 2: Create a timelapse visualization of flight count density over the years 1987 to 2020 Another interesting visualization that can be created is a timelapse visualization of flight count density over the years 1987 to 2020. This visualization can help users understand how flight density has changed over time and identify trends in air travel within the US. The visualization is created using a heatmap that displays the density of flights across different regions in the US. The visualization provides a number of control for the user to experiment with, such as play, pause, speed controls and forward and backward buttons to navigate through the years.

From the visualization, we can make a few key observations:

- **Years with the highest flight density:** The visualization can help identify years with the highest flight density in the US. From the visualization, it seems that the years 1987, 1994, 1995, 1996, 2006, 2007, 2010, and 2015 had the highest flight density, with a large number of flights occurring across the US. These years may have seen an increase in air travel due to various factors, such as economic growth or increased globalization. For example, the mid-90s saw a significant increase in air travel due to the rise of low-cost carriers [7].
- **Years with the lowest flight density:** The visualization can help identify years with the lowest flight density in the US. From the visualization, it seems that the years 1988, 1991, 2003, 2004, 2008 and 2020 had the lowest flight density, with fewer flights occurring across the US. These can also be explained by various factors. For example, 2020 saw a significant decrease in air travel due to the COVID-19 pandemic, which led to travel restrictions and reduced demand for flights.

Location: Code lines 77-116 in file `backend/part3/views.py`

5 Evaluation

When evaluating the project, it is important to consider the objectives and accomplishments of the project. Aside from implementing the required tasks, one objective that stood out was the enhanced understanding of PySpark and the PySpark SQL API. There were many features in the API that were learned to either speed up query time, such as broadcasting smaller dataframes, which caches the dataframe in memory across all nodes in the cluster, or slow down query time, such as using the collect() function, which brings all the data from the distributed dataframe to the driver node. From these observations, queries were optimized to ensure that they were efficient and did not take too long to run. Additionally, learning more about the dataset and the insights that could be gained from it proved to be enriching. For example, the advanced visualizations in Part 3 deepened the understanding of the connectivity between US airports and the flight density over the years, which was fascinating to explore and even take into account for future travel plans within the US.

5.1 Difficulties Encountered

A few difficulties were encountered during the development of the project. These difficulties can be categorized into back-end and front-end challenges. One notable back-end challenge I faced when developing my queries was handling missing data, specifically when implementing the composite score logic for the region and state performance analysis. For example, the 'Taxi In' and 'Taxi Out' columns had a significant amount of missing data (across years in the dataset), which affected the calculation of the composite score. To address this issue, I decided to exclude these columns from the composite score calculation by imputing a value of 0 for that specific metric, as the missing data would skew the results. Another major backed issue I faced was deciding when to normalize or scale the data such that the radar chart and chloropleth map would be more informative. For example, when comparing the performance of airports in the radar chart, I had to normalize the data to ensure that the radar chart was not skewed by the different scales of the metrics.

On the front-end side, challenges were encountered in ensuring that the visualizations were responsive and interactive. For instance, in creating the timelapse , it was necessary to ensure that the user was able to have appropriate controls, such as 'play', and 'pause', to properly understand how flight density varies over the years. Moreover, in creating the chloropleth map, attention was paid to making the map interactive, allowing users to hover over a region to view the average performance score. Special attention was also given to the accessibility of the visualizations, ensuring that they were easy to understand and interact with.

6 Conclusion

The InFlight application was developed to provide users with an interactive platform to explore and analyze the *Airline Reporting Carrier On-Time Performance* dataset. It successfully achieved the main objectives of developing a user-friendly interface, gaining familiarity with PySpark, and providing valuable insights into the on-time performance of domestic flights in the United States. Future work can explore implementing a feature that predicts flight delays based on historical data using a regression model such as Linear Regression or Gradient-Boosted Tree Regression using the PySpark MLlib library. This feature would provide users with a predictive tool to estimate the likelihood of a flight being delayed based on various factors such as the airline, airport, and time of day.

Word Count: 3498

References

[1] United States Department of Transport. *Download Page.* 2023. URL: `https://www.transtats.bts.gov/DL_SelectFields.asp?gnoyr_VQ=FGJ&QO_fu146_anzr=b0-gvzr`.

[2] *Apache Parquet.* Apache Software Foundation. Accessed: 2024-03-23. 2024. URL: `https://parquet.apache.org/`.

[3] Sergey Melnik et al. "Dremel: Interactive Analysis of Web-Scale Datasets". In: *Proc. of the 36th Int'l Conf on Very Large Data Bases.* 2010, pp. 330–339. URL: `http://www.vldb2010.org/accept.htm`.

[4] *What is Parquet?* Databricks. URL: `https://www.databricks.com/glossary/what-is-parquet`.

[5] Garrick Blalock, Vrinda Kadiyali, and Daniel H. Simon. "The Impact of Post-9/11 Airport Security Measures on the Demand for Air Travel". In: *The Journal of Law and Economics* 50.4 (2007), pp. 731–755. DOI: `10.1086/519816`. eprint: `https://doi.org/10.1086/519816`. URL: `https://doi.org/10.1086/519816`.

[6] United States Department of Transportation. *A Review of the Aviation Industry in 2008.* 2009. URL: `https://www.oig.dot.gov/sites/default/files/Metrics_10_Final_Report_-_508_Compliant.pdf`.

[7] Elvis Picardo. *An economic analysis of the low-cost airline industry.* URL: `https://www.investopedia.com/articles/investing/022916/economic-analysis-lowcost-airline-industry-luvdal.asp`.