Semester 2, 2023/2024

Apache Hadoop, Pig and Hive Project

Prepared by:

Name	**Matric Number**
MOHAMED MOUBARAK MOHAMED MISBAHOU MKOUBOI	P139575
AMRITPAL SINGH A/L ARVINDER SINGH	P145214
WANG YU	P144665
MUHAMMAD FARIS BIN SAMSUKAMAL	P140696

Lecturer:

PROF. MADYA DR. ELANKOVAN A/L A SUNDARARAJAN

Part A

WORDCOUNT

1. Introduction

This project aims to analyze the word frequency of 10 journals by Charles Dickens, a renowned Victorian author. The goal is to count and analyze the frequency of words in a large, merged file that contains 10 journals. He is widely reputed to be one of the greatest writers of the time due to his lasting influence on English literature as well as the richness of his vocabulary (Abrams, 2022). The journals are downloaded from Project Gutenberg, one of the largest and most accessible digital libraries, providing free access to a vast range of books in the public domain (Gerlach, 2020). By merging the 10 text files into one file, the project creates a textual dataset that encapsulates the author's writing style, allowing for text mining to uncover patterns in the writings.

The purpose of this textual analysis is to obtain valuable insights into Dickens' writing style, such as thematic preoccupations and language evolution over different periods of his work. Word frequency analysis can indicate the most frequently used words, offering insight into the writing style of that era and highlighting linguistic characteristics, such as the complexity of sentence structures or the density of specific types of words like adjectives or adverbs. It can also highlight linguistic characteristics, such as the complexity of sentence structures or the density of specific types of words, like adjectives or adverbs (Antons, 2020).

To conduct the word frequency analysis, Apache Pig and Apache Hive are utilized due to their ability to process large textual datasets in Hadoop. Apache Pig uses high-level language to pre-process or clean the textual data, while Apache Hive provides a more SQL-like interface for querying and aggregating data at scale. The integration of Apache Hive with Hadoop allows for easy management and analysis of the large dataset. The integration of Apache Hive with Hadoop provides the ability to manage and analyze the large dataset with relative ease, making it an ideal choice for this word frequency project (Nair, 2021).

The results of the word count and frequency analysis can be clearly depicted through visualization tools like Power BI, making the results accessible and visually compelling. Visualizations such as word clouds, bar charts, and histograms will be used to highlight the most frequently occurring words in the 10 journals used in this project.

2. Objectives

1. Perform a comprehensive word frequency analysis of 10 merged journals authored by Charles Dickens in order to identify writing style and recurrent themes throughout the literature

2. Pre-process, clean, and analyze a large volume of textual data using Apache

Pig and Hive, demonstrating the capability of these tools in handling and analyzing big data.

3. Create visualizations using Power BI for the representations of word frequency data

2.1 Source of Data:

1) Project Gutenberg

   For this project, we downloaded 10 journals authored by Charles Dickens from Project Gutenberg. Project Gutenberg is a vast online archive that encompasses free eBooks, most of which are copyright free or expired and available in the public domain (Gerlach, 2020).

2) 10 journals by Charles Dickens

No	Title	Word Count	Date Published
1.	A Tale of Two Cities	139605	1859
2.	Great Expectations	187596	1867
3.	David Copperfield	357489	1850
4.	Hard Times	104821	1854
5.	Bleak House	355936	1852
6.	Our Mutual Friend	327727	1865
7.	The Old Curiosity Shop	218538	1841
8.	Life and Adventures of Martin Chuzzlewit	338077	1844
9.	The Mystery of Edwin Drood	96178	1870
10.	A Christmas Carol	31611	1843

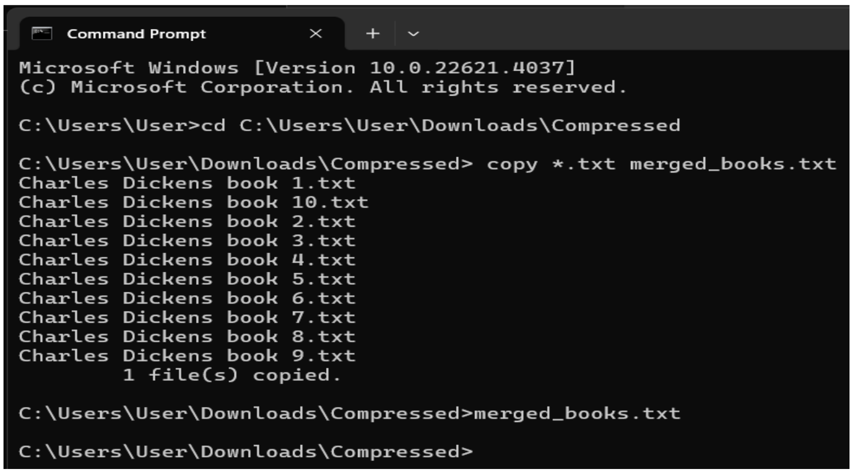
2.2 Data Preprocessing:

2.2.1. Merging the Books into a Single Large File

The initial stage was to combine 10 books by Charles Dickens collected from Project Gutenberg into a single text file. The steps were as follows:

1. Merging Files in Windows:

   The downloaded books were stored in a single folder, then a simple command was used to combine all text files (10 books) in the folder into one file named merrged_books:



```
Microsoft Windows [Version 10.0.22621.4037]
(c) Microsoft Corporation. All rights reserved.

C:\Users\User>cd C:\Users\User\Downloads\Compressed

C:\Users\User\Downloads\Compressed> copy *.txt merged_books.txt
Charles Dickens book 1.txt
Charles Dickens book 10.txt
Charles Dickens book 2.txt
Charles Dickens book 3.txt
Charles Dickens book 4.txt
Charles Dickens book 5.txt
Charles Dickens book 6.txt
Charles Dickens book 7.txt
Charles Dickens book 8.txt
Charles Dickens book 9.txt
        1 file(s) copied.

C:\Users\User\Downloads\Compressed>merged_books.txt

C:\Users\User\Downloads\Compressed>
```

2. Uploading to HDFS:

   After merging the files, they were moved to a Virtual Machine (VM). The file was then uploaded to the Hadoop Distributed File System (HDFS) with the following command:

```
hdfs dfs -put merged_books.txt /user/cloudera/merged_books/
```

2.2.2. Preprocessing Steps in Pig

Pig was used to clean and prepare the original data for the word frequency analysis. The steps taken were as follows:

1. Loading the Data:

   The combined text file was loaded into Pig using the "PigStorage" function, with spaces operating as word delimiters.

```
books = LOAD '/user/cloudera/merged_books.txt' USING PigStorage(' ') AS (line:chararray);
```

2. Cleaning the Text:

   To preserve word boundaries, all text to lowercase were changed and substituted non-alphabetic letters with spaces.

```
cleaned_lines = FOREACH books GENERATE REPLACE(LOWER(line), '[^a-zA-Z]', ' ') AS cleaned_line;
```

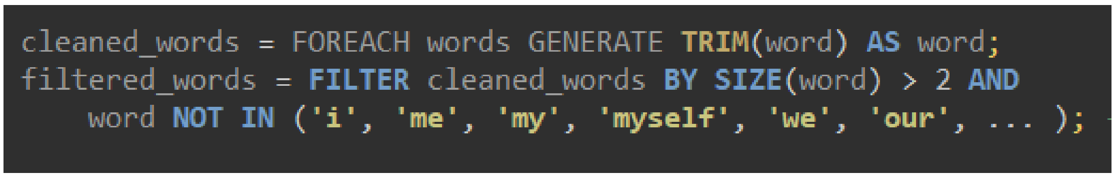
3. Tokenizing the Text:

   The text was divided into separate words (tokens) by splitting it apart using spaces:

```
words = FOREACH cleaned_lines GENERATE FLATTEN(TOKENIZE(cleaned_line)) AS word;
```

4. Trimming and Filtering Words:

   The words were altered to remove any leading or trailing spaces, and short or nonsensical terms were filtered out using a specified stop word list.



```
cleaned_words = FOREACH words GENERATE TRIM(word) AS word;
filtered_words = FILTER cleaned_words BY SIZE(word) > 2 AND
    word NOT IN ('i', 'me', 'my', 'myself', 'we', 'our', ... );
```

These processes ensured that the text was cleaned, tokenized, and filtered, resulting in only significant words for word count analysis.

2.2.3. Preprocessing Steps in Hive

Hive was used to process the text further and get it ready for word frequency analysis. The steps were:

1. Loading the Data into Hive Tables:

   - A Hive table was created to store the raw text lines from the merged books:

```
CREATE TABLE book_data (line STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\n' STORED AS
TEXTFILE;
LOAD DATA INPATH '/user/cloudera/merged_books.txt' INTO TABLE
book_data;
```

   - Another table was created to store stop words and filter them throughout the analysis:

```
CREATE TABLE stop_words (stop_word STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\n' STORED AS
TEXTFILE;
LOAD DATA INPATH '/user/cloudera/stop_words.txt' INTO TABLE
stop_words;
```

2. Text Cleaning:

   The text was cleaned by deleting digits and special characters using "regexp_replace" in Hive.

```
CREATE VIEW cleaned_text AS
SELECT regexp_replace(
    regexp_replace(lower(line), '[0-9]', ''), -- Remove numbers
    '[()\-_"''\[\]{}]', '' -- Remove special characters
) AS cleaned_line
FROM book_data;
```

3. Tokenizing the Text:

   The cleaned text was divided into individual words using Hive's "split" function:

```
CREATE VIEW word_tokens AS
SELECT explode(split(cleaned_line, '\\s+')) AS word
FROM cleaned_text;
```

Hive's preprocessing prepared the dataset for word frequency analysis by tokenizing the text and removing unwanted characters.

These preprocessing methods ensured that the text was properly cleaned and prepared for further analysis in Pig and Hive. This tool set offers flexibility for text processing and analysis within the Hadoop ecosystem.

3. Methodology

After preparing the text from the merged books, the next step was to count the frequency of words with Pig and Hive. This method includes removing unnecessary words and calculating word counts.

3.1 Pig Script for Wordcount:

The text was processed using the Pig script, which counted the occurrences of each word. The steps included the following:

1. Filtering Out Additional Nonsensical Words:

   After cleaning the text and tokenizing it into individual words, additional filtering was used to remove terms that were either too short, too long, or featured unusual patterns.

```
filtered_words = FILTER filtered_words BY SIZE(word) < 15; -- filter out very long words
```

2. Grouping and Counting Words:

   The terms were then categorized according to their values, and their occurrences were counted.

```
grouped_words = GROUP filtered_words BY word;
word_count = FOREACH grouped_words GENERATE group AS word, COUNT(filtered_words) AS count;
```

3. Storing the Word Count Results:

   Finally, the output was saved as a CSV file in the Hadoop Distributed File System (HDFS), which could then be utilized for visualization and additional analysis:

```
STORE word_count INTO '/user/cloudera/cleaned_word_count.csv' USING PigStorage(',');
```

This Pig script ensured that the words were properly filtered, counted, and saved in a structured format.

3.2 Hive Queries for Wordcount:

Hive was also used to count the frequency of words, with extra steps to filter out stop words and empty rows.

1. Creating a Table for Word Count:

   A Hive table was created to keep track of the word counts. The "word_tokens" view, which included the cleaned and tokenized text, was used as the source.

```
CREATE TABLE word_count AS
SELECT
    w.word,
    COUNT(*) AS count
FROM word_tokens w
LEFT OUTER JOIN stop_words s
ON w.word = s.stop_word
WHERE s.stop_word IS NULL -- Exclude stop words
AND w.word != '' -- Exclude empty rows
GROUP BY w.word
ORDER BY count DESC; -- order by frequency
```

2. Filtering Out Stop Words and Empty Rows:

   The query used a "LEFT OUTER JOIN" between the "word_tokens" and "stop_words" tables to ensure that any words from the stop words list were removed from the results. Additionally, empty rows were filtered out using the "w.word!= ' ' " condition.

3. Storing and Querying the Results:

   The word count results were saved in the "word_count" table, which recorded each word and its frequency of occurrence. The query could be executed to extract the most frequently occurring words for further analysis and visualization.

This combination of Pig and Hive scripts enabled the quick processing of a huge text dataset, ensuring that word count results were accurate and suitable for presentation in tools such as Power BI.

5. Visualization

5.1 Power BI Visualizations:

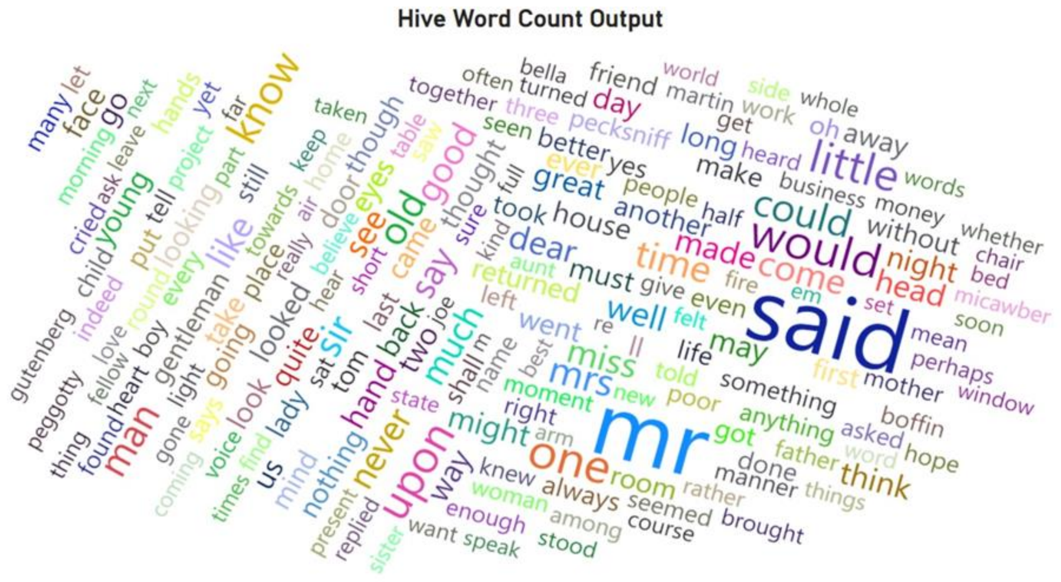


Figure 1: The word cloud for Hive Word Count Output generated through Power BI

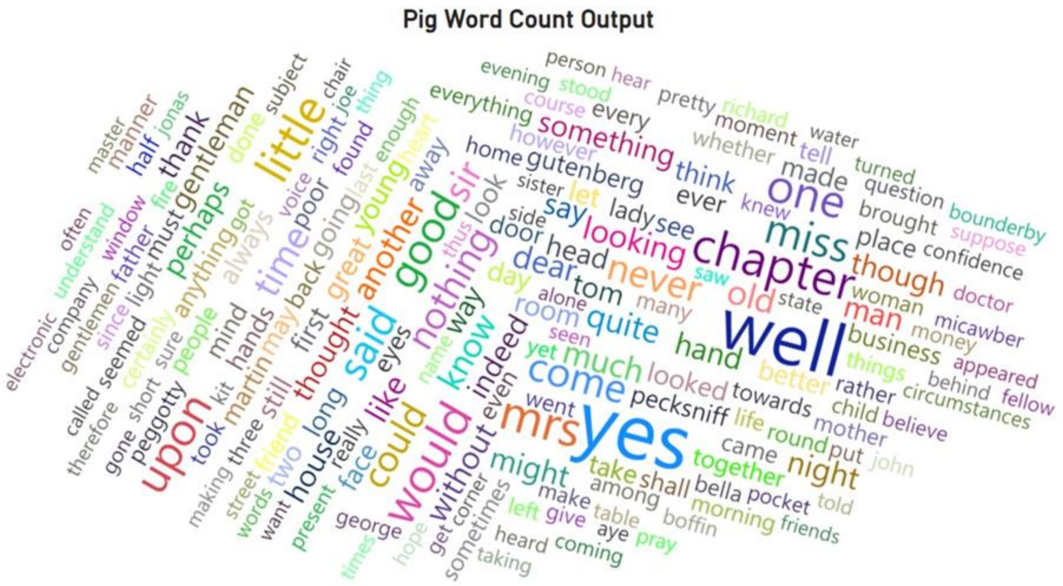


Figure 2: The word cloud for Hive Word Count Output generated through Power BI

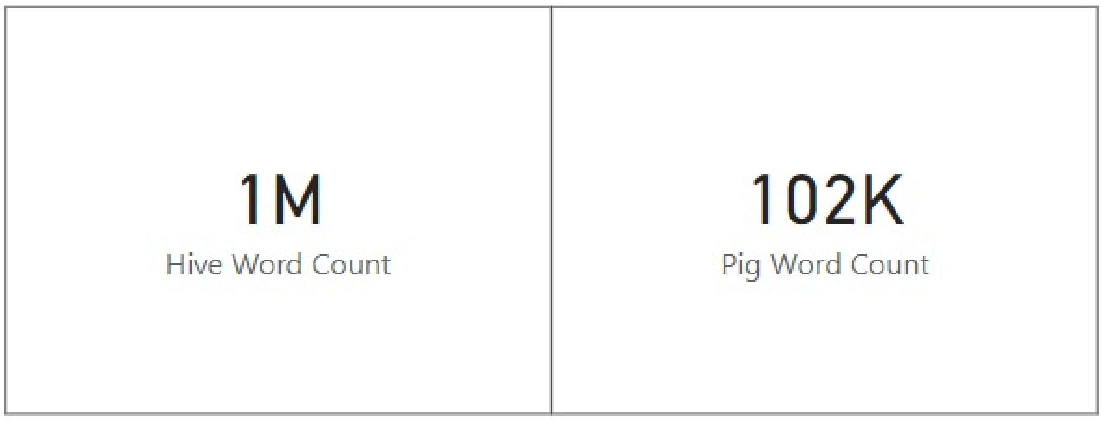


Figure 3: The difference between the Hive word count output and the Pig word count output

5.2 Insights from Visualizations:

The project's visualizations revealed significant differences between Hive Output and Pig Output. In the Hive Output word cloud, there is a more diverse distribution of word sizes, suggesting Hive's better accuracy in counting and categorizing word frequencies. The Pig Output word cloud shows a discrepancy in word distribution, possibly due to Pig's processing not being optimized for wordcount applications. The Hive cloud also captures a wider range of unique words, including both high and low frequency words, indicating better representation of the text's linguistic complexity.

Hive's word cloud also incorporates simple and complex words, including nouns, verbs, and adjectives, reflecting a broader spectrum of language usage. The Pig word cloud, on the other hand, places more emphasis on simpler high frequency words, potentially distorting the visualization.

Hive's word cloud also reflects a more accurate representation of word frequencies, with the size of each word closely corresponding to its actual occurrence in the text. The Hive word cloud also transitions from small to large words more profoundly, indicating more detailed frequency calculations. In contrast, Pig's word cloud appears more sudden, indicating a binary division between high-frequency and low-frequency words.

6. Conclusion

This Word Cloud mini project is a great chance to dig deeper into text analytics, from collecting data to visualization. For reflection and future work, we have a solid understanding of the frequency with which authors employ words, which can disclose their writing and vocabulary preferences. We also learn how to process data using Pig and Hive, as well as how to visualize it. We believe that for larger datasets, we ought to look into more efficient data processing methods or even more complex tools such as Apache Spark. Furthermore, for improved visualization, we can employ more professional tools or web-based interactive visualizations. At the same time, we plan to broaden the scope of the study to include additional works or authors.

7. References

1) Books: Charles Dickens (sorted by popularity). (2024). Project Gutenberg. https://www.gutenberg.org/ebooks/search/?query=Charles+Dickens&submit_search=Go%21

2) Gerlach, M., & Francesc Font-Clos. (2020). A Standardized Project Gutenberg Corpus for Statistical Analysis of Natural Language and Quantitative Linguistics. Entropy, 22(1), 126–126. https://doi.org/10.3390/e22010126

3) Antons, D., Eduard Grünwald, Cichy, P., & Torsten Oliver Salge. (2020). The application of text mining methods in innovation research: current state, evolution patterns, and development priorities. R and D Management, 50(3), 329–351. https://doi.org/10.1111/radm.12408

4) Nair, S. (2021). Comparative Analysis of Selected Hadoop-based Tools: A Literature Review and User's Perspective. 2021 International Conference on Emerging Smart Computing and Informatics (ESCI). https://doi.org/10.1109/esci50559.2021.9396949

5) W3Schools.com. (2024). W3schools.com. https://www.w3schools.com/sql/

Part B

AIRLINES ON-TIME PERFORMANCE ANALYSIS

1. Introduction

Air travel has become a crucial part of the global economy, with flight delays and cancellations having significant financial and passenger disruptions. To predict these delays, the aviation industry has grown, making it crucial to understand the factors influencing them. This research aims to analyze airlines' on-time performance using a dataset from the 2009 Data Expo, focusing on factors influencing flight punctuality such as time of travel, aircraft age, weather conditions, and flight volume trends.

The project aims to identify the best flight timing to avoid delays, the impact of age on the plane, and the influence of weather on flight delays. The dataset, gathered from the Data Expo of 2009, contains flight arrival and departure details for all commercial flights within the USA from October 1987 to April 2008. Apache Hive is used for advanced data analysis on a massive scale.

The project focuses on answering eight key questions related to airline delays, cancellations, and travel patterns. The data is preprocessed or cleaned to handle missing or incomplete records, and systematically analyzed using filters, aggregations, and statistical measures to derive meaningful insights. The report will highlight the methodologies used, results obtained, and visualizations of the results.

2. Objectives

1. Identify and analyze the factors that contribute to flight delays and cancellations based on the dataset

2. Utilize Apache Hive to analyze and query airline performance data for the identification of trends and patterns related to flight delays, cancellations, and air traffic flow

3. Analyze the influence of weather conditions on flight delays and cancellations based on the dataset

3. Data Description

The dataset used in this analysis is from Data Expo 2009, comprising detailed flight data from the United States between 1987 and 2008. The dataset includes fields such as Year, Month, Day of Week, DepTime (departure time), ArrTime (arrival time), ArrDelay (arrival delay), DepDelay (departure delay), Distance, UniqueCarrier (airline), FlightNum (flight number), and delay-related causes such as CarrierDelay, WeatherDelay, and LateAircraftDelay. This comprehensive dataset contains information about flight performance and delays over 21 years.

Column	Description
Year	The year the flight occurred
Month	The month the flight occurred
DayofMonth	The date the flight occurred
DayOfWeek	The day of the week the flight occurred (usually 1 (Monday) to 7 (Sunday))
DepTime	The actual departure time (expressed in 24-hour format)
CRSDepTime	The planned departure time (expressed in 24-hour format)
ArrTime	The planned departure time (expressed in 24-hour format)
CRSArrTime	The planned departure time (expressed in 24-hour format)
UniqueCarrier	The airline code (such as "WN" for Southwest Airlines)
FlightNum	The flight number
TailNum	The tail number of the aircraft
ActualElapsedTime	The actual flight time (the time from departure to arrival, in minutes).
CRSElapsedTime	Scheduled flight time (time from takeoff to arrival, in minutes)
AirTime	Actual flight time (time the aircraft was in the air, excluding taxi time, in minutes)
ArrDelay	Arrival delay time (how many minutes the arrival time was later than the scheduled time, may be negative to indicate an early arrival)
DepDelay	Departure delay time (how many minutes the takeoff time was later than the scheduled time, may be negative to indicate an early departure)
Origin	Origin airport code (such as "HOU" for Houston Hobby Airport)
Dest	Destination airport code (such as "LIT" for Little Rock Airport)
Distance	Distance flown (the straight-line distance from the origin to the destination, in miles).
TaxiIn	Post-arrival taxi time (the time the aircraft was taxiing after arrival, in minutes)
TaxiOut	Taxi time before takeoff (the taxi time of the aircraft before takeoff, in minutes)
Cancelled	Whether the flight was canceled (0 means not canceled, 1 means canceled)
CancellationCode	Flight cancellation code (A: airline reason, B: weather reason, C: NAS reason, D: safety reason)
Diverted	Whether the flight was diverted (0 means not diverted, 1 means diverted)
CarrierDelay	Delay time caused by airline reasons (in minutes)
WeatherDelay	Delay time caused by weather reasons (in minutes)
NASDelay	Delay time caused by national aviation system reasons (in minutes)
SecurityDelay	Delay time caused by security check reasons (in minutes)
LateAircraftDelay	Delay time caused by delayed aircraft reasons (in minutes)

Source: (https://community.amstat.org/jointscsg-section/dataexpo/dataexpo2009).

4. Methodology

4.1. Tools Used

The tools used for this analysis are:

1. Hive:

   Hive is an advanced tool for exploring and analyzing huge datasets stored in Hadoop, using a SQL interface. It was chosen because the airline performance dataset can be huge, and Hive provides for efficient querying and manipulation of such data. Its compatibility with large-scale distributed systems makes it excellent for processing millions of rows quickly.

2. SQL:

   SQL, as implemented in Hive, offers a familiar and efficient method for querying structured data. SQL was essential for our analysis because it facilitates exact querying, filtering, and data extraction from a dataset.

4.2. Data Preparation

1. Dataset Loading:

   The dataset was loaded into Hive, and a table named "flight_data" was created to contain the data with the original data types for each column. The dataset schema defined the data types assigned, which were "INT", "FLOAT", and "VARCHAR".

2. Null and Empty Value Handling:

   The dataset had multiple columns with null values or empty cells. To ensure the analysis's integrity, rows with null or missing values were removed before any analysis. This was accomplished by putting data into a new table (flight_answer) and applying a "WHERE" clause to remove any rows with null or empty values in all columns.

3. Data Type Conversion:

   During the data preparation process, it was essential to verify that columns such as DepTime, ArrTime, and other delay-related fields were correctly represented as floating-point numbers, as time values can contain decimals. VARCHAR was utilized in textual fields such as UniqueCarrier and TailNum to efficiently handle short strings.

4. Data Cleaning: The cleaning process included the following:

   - Removing Null Values: Rows with missing values in any column have been removed to avoid biased results during analysis.

   - Data Validation: Ensured that the remaining data followed the required forms for date and time fields, resulting in accurate analysis in the following steps.

4.3. Challenges Faced

1. Handling Null Values:

The dataset contains many null values, particularly in fields such as "CarrierDelay", "WeatherDelay", "NASDelay", and "SecurityDelay". The removal of rows with null values in these columns significantly minimized the size of the dataset, but it was required to enable accurate and beneficial analysis.

2. Data Format Consistency:

The time columns (DepTime, ArrTime, etc.) were in mixed format (integer and float), making standard data processing difficult. We addressed this issue by converting the times to a consistent floating-point representation during table construction.

3. Performance Issues:

The size of the dataset caused some performance issues while running queries. These speed issues were fixed by carefully indexing the dataset, filtering null values, and dividing it into smaller, more manageable tables for specific queries. Additionally, Hive's capacity to manage distributed data processing on huge datasets contributed to the resolution of these performance issues.

4. Lack of Plane Manufacturing Year:

The dataset did not include particular information about each plane's manufacturing year. This constrained the analysis' ability to address the topic of whether older planes experience higher delays, and we were forced to make assumptions based on accessible data, such as aircraft tail numbers, which were insufficient to directly answer the question.

5. Answers to Questions

For each question, follow this structure:

a. Best Time to Fly to Minimize Delays

As depicted in the visualization for the best time of day to fly to minimize delays (Figure 4), it is evident that the best departure time should be approximately 5:00 am onwards as there is a marked decrease in average arrival delay after 5:00am up till possibly 8:00a.m.

```
-- Best Time of Day:
SELECT
    FLOOR(DepTime) AS hour,  -- Extract the hour part from float
    ROUND(AVG(ArrDelay)) AS avg_arr_delay  -- Round average delay to
nearest integer
FROM flight_answer
```

```sql
GROUP BY hour
ORDER BY avg_arr_delay ASC;  -- Ascending to find the minimum delay
```

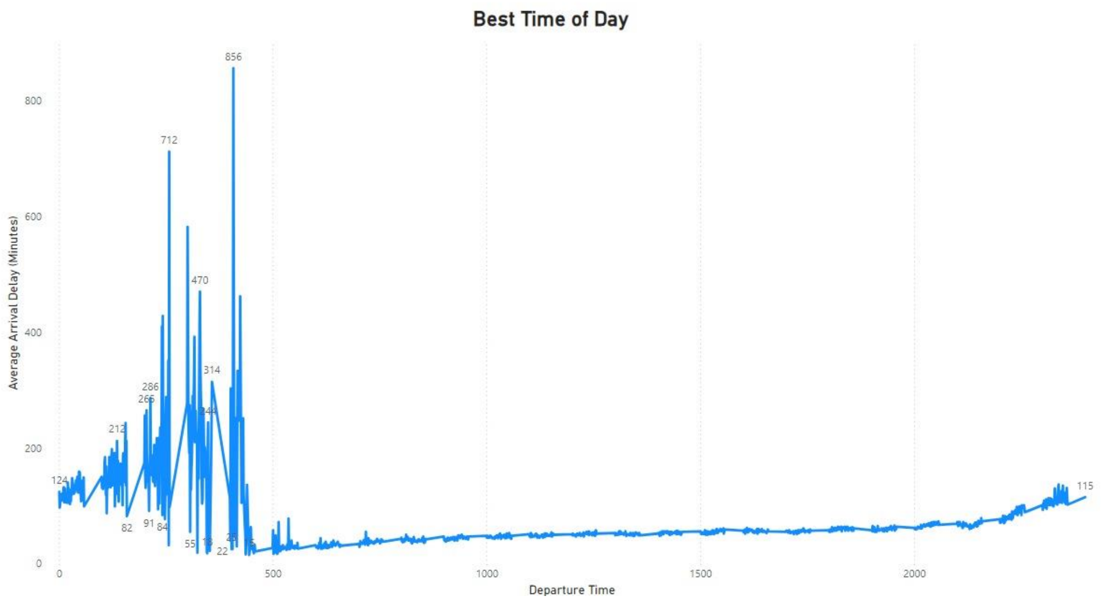


Figure 4: Power BI Visualization for best time of day

As for the best time of the week, it is evident from Figure 5 that the third day of the week happens to have the least average arrival delay. This suggests that the best departure day should fall on a Wednesday as it is the third day of the week.

```sql
-- Best Day of Week:
SELECT
    DayOfWeek,
    ROUND(AVG(ArrDelay)) AS avg_arr_delay
FROM flight_answer
GROUP BY DayOfWeek
ORDER BY avg_arr_delay ASC;  -- Ascending to find the minimum delay
```

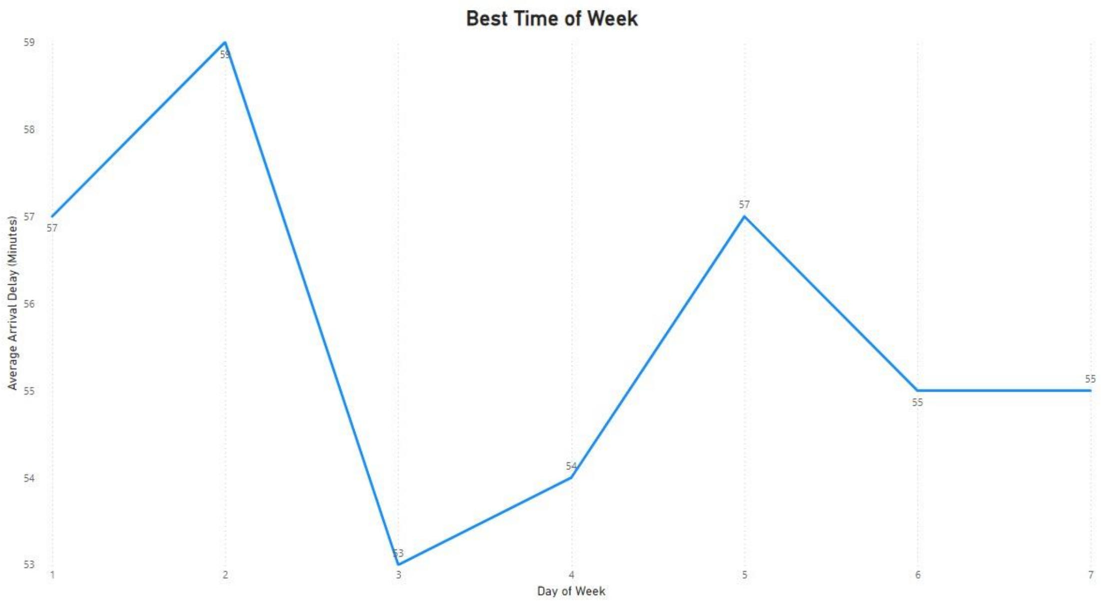


Figure 5: Power BI Visualization for best time of week

Finally, the best month to fly that has the least average arrival delay is the month of April, which is the fourth month as depicted in Figure 6.

```
-- Best Time of Year:
SELECT
    Month,
    ROUND(AVG(ArrDelay)) AS avg_arr_delay
FROM flight_answer
GROUP BY Month
ORDER BY avg_arr_delay ASC;  -- Ascending to find the minimum delay
```

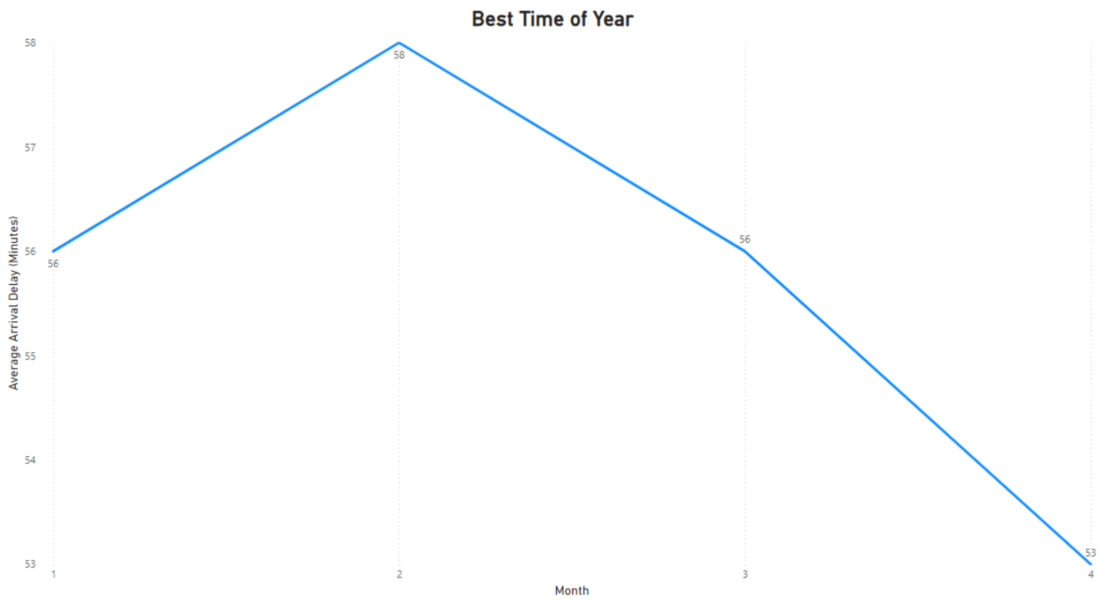


Figure 6: Power BI Visualization for best time of month

b. Older Planes and Delays

In order to identify if older planes suffer more delays, the visualization of average arrival delays to the tail number of planes can be plotted. From Figure 7, it is evident that there are certain planes that incur delays more frequently and in longer intervals as opposed to other planes. Therefore, we can conclude that certain planes do suffer longer average arrival delays than others however the results are inconclusive as to whether the delays are a result of aging factor or other variables.

```
-- Analyze Delays by Tail Number
SELECT
    TailNum,
    ROUND(AVG(ArrDelay)) AS avg_arr_delay
FROM flight_answer
WHERE ArrDelay IS NOT NULL
GROUP BY TailNum
ORDER BY avg_arr_delay DESC;
```

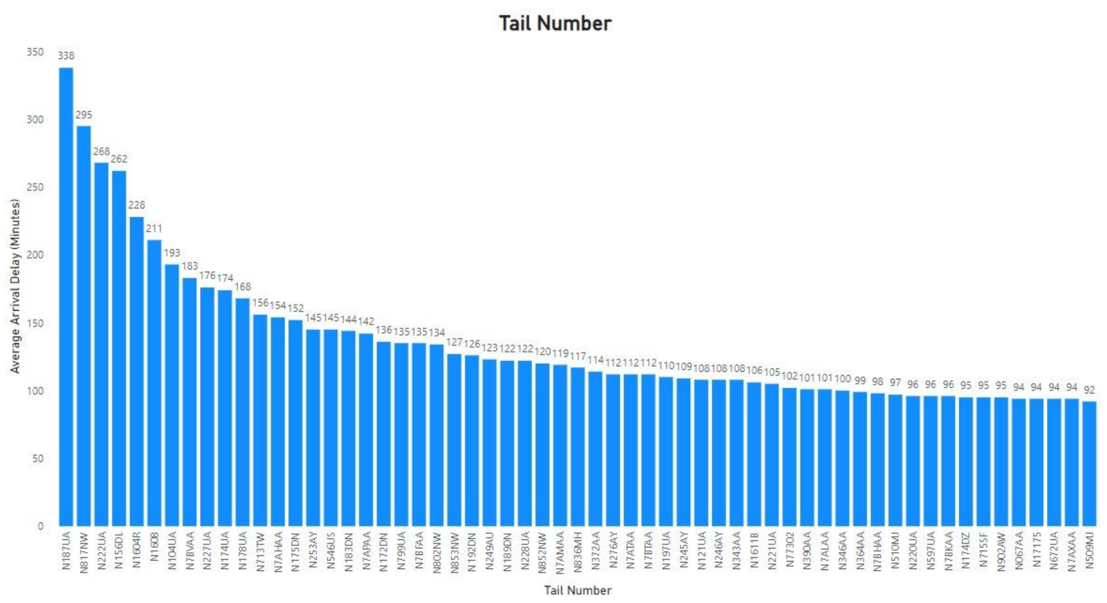


Figure 7: Power BI Visualization of Average Arrival Delay Based on Tail Number

As stated in Figure 8, there is a small but profound difference between the unique carriers and the average arrival delay. It is plausible that certain carriers may have a fleet of older planes, thus resulting in longer arrival delays possibly due to more frequent maintenance issues etcetera.

```sql
-- Delays by UniqueCarrier
SELECT
    UniqueCarrier,
    ROUND(AVG(ArrDelay)) AS avg_arr_delay
FROM flight_answer
WHERE ArrDelay IS NOT NULL
GROUP BY UniqueCarrier
ORDER BY avg_arr_delay DESC;
```

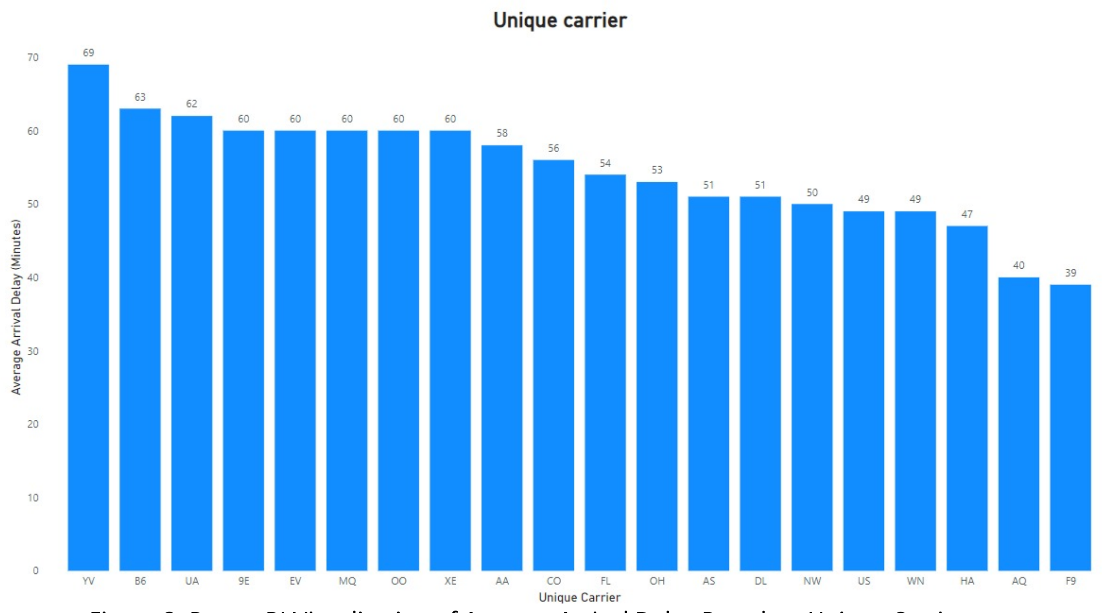


Figure 8: Power BI Visualization of Average Arrival Delay Based on Unique Carrier

The graph in Figure 9 depicts that there are planes with certain flight numbers which have an average arrival delay that far surpasses their counterparts. This inference, coupled with the inference obtained from Figure 7 and Figure 8, that certain planes encounter more delays than others proves that older planes may possibly cause more delays in the arrival time.

```sql
-- Delays by FlightNum
SELECT
    FlightNum,
    ROUND(AVG(ArrDelay)) AS avg_arr_delay
FROM flight_answer
WHERE ArrDelay IS NOT NULL
GROUP BY FlightNum
ORDER BY avg_arr_delay DESC;
```

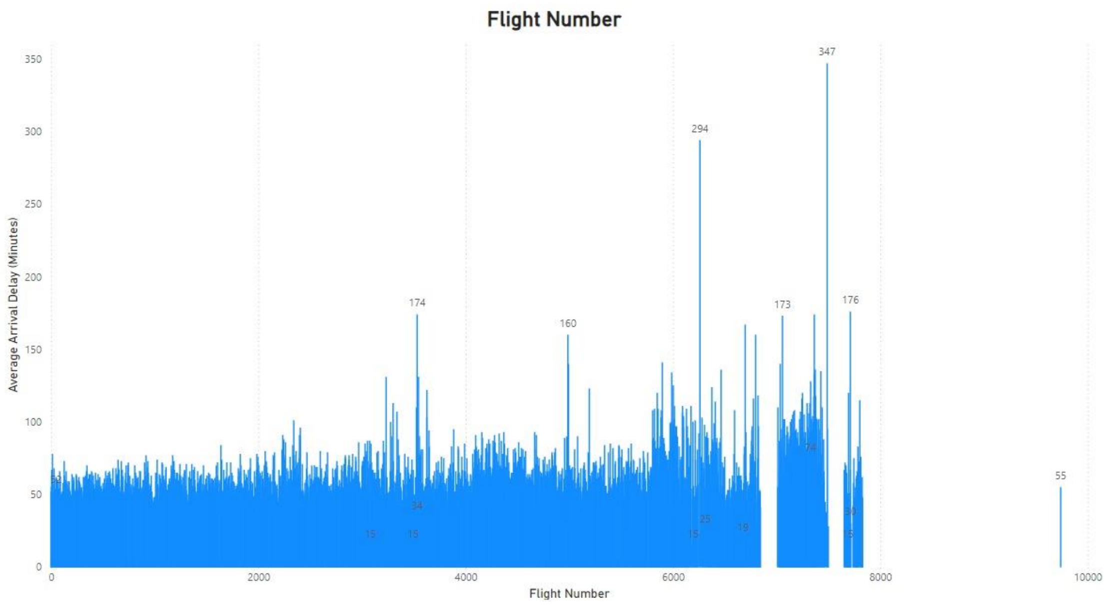


Figure 9: Power BI Visualization of Average Arrival Delay based on Flight Number

c. Change in Flight Numbers Between Locations Over Time

The chart (Figure 10) shows the number of flights between two places over four months. Initially, the number of flights increases slightly, then reaches its highest point in the third month (march). By April, there is a sharp drop in both flights leaving and coming to the airports. This suggests there might be seasonal changes or specific events affecting how many people are flying during these times.

```sql
-- Flight Count by Month to See Seasonal Trends
SELECT
    Year,
    Month,
    Origin,
    Dest,
    COUNT(*) AS flight_count
FROM flight_answer
GROUP BY Year, Month, Origin, Dest
ORDER BY Year ASC, Month ASC;
```

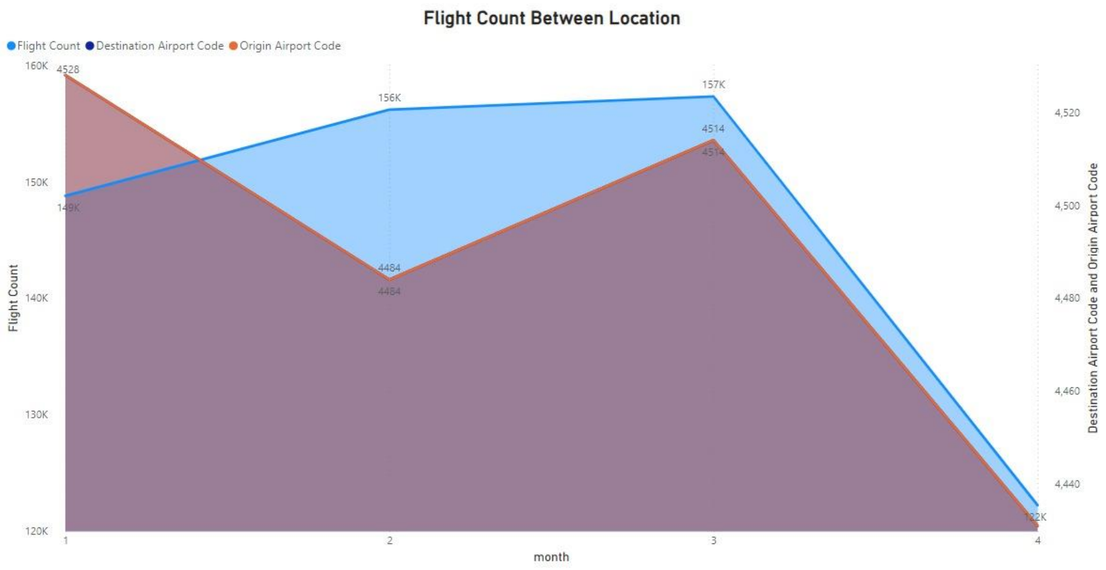


Figure 10: Power BI Visualization of Change in Flight Numbers Between Locations Over Time

d. Predicting Delays Based on Weather

The graph below (Figure 11) demonstrates that weather delay is a strong predictor of arrival delays, with longer weather delays typically leading to longer total arrival delays. The scatter plot and the trend suggest that weather is a significant factor in overall flight delays, supporting the hypothesis that weather delays cause notable disruptions in flight arrival times.

```sql
-- Regression Analysis:
SELECT
    WeatherDelay,
    ROUND(AVG(ArrDelay)) AS avg_arr_delay
FROM flight_answer
WHERE WeatherDelay IS NOT NULL AND ArrDelay IS NOT NULL
GROUP BY WeatherDelay
ORDER BY WeatherDelay ASC;
```

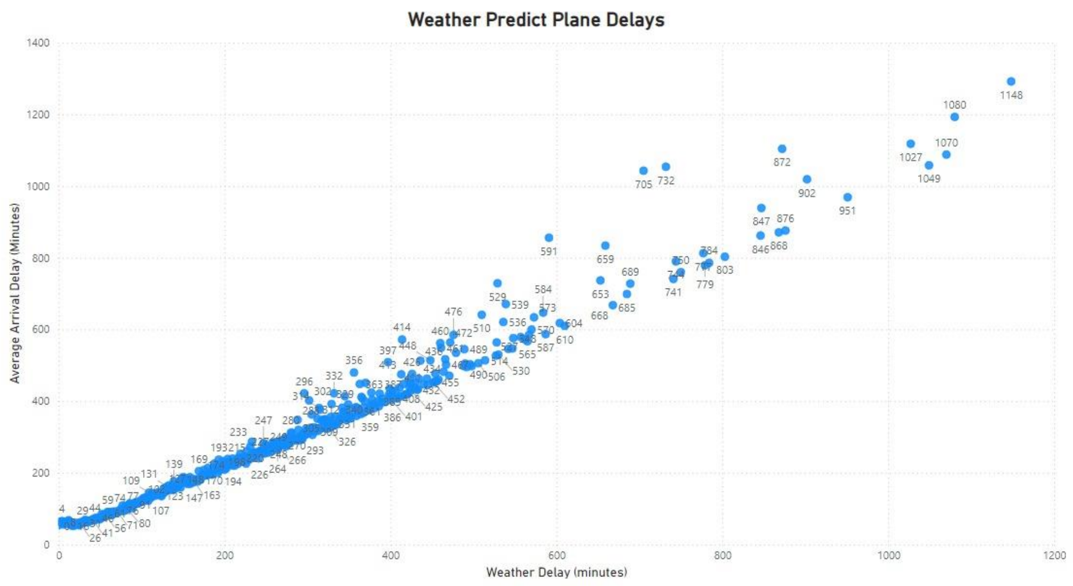


Figure 11: Power BI Visualization of Predicting Delays Based on Weather

e. Major Factor Causing Delay

The Power BI visualization in Figure 12 illustrates that the major factor causing air travel delays is the National Airspace System (NAS) Delay, which accounts for 39.6% of all delays, as represented by the largest dark blue segment of the pie chart. This category includes delays due to non-extreme weather conditions, airport operations, heavy traffic volume, and air traffic control. The other segments include Carrier Delay at 30.5%, Late Aircraft Delay at 29.7%, and Security Delay at only 0.2%, indicating that issues controlled by the airline and incoming aircraft timings are also significant but less so than NAS-related issues.

```sql
-- Proportion of Delays Caused by Each Factor
SELECT
    SUM(CarrierDelay) / SUM(ArrDelay) * 100 AS carrier_delay_percentage,
    SUM(WeatherDelay) / SUM(ArrDelay) * 100 AS weather_delay_percentage,
    SUM(NASDelay) / SUM(ArrDelay) * 100 AS nas_delay_percentage,
    SUM(SecurityDelay) / SUM(ArrDelay) * 100 AS security_delay_percentage,
    SUM(LateAircraftDelay) / SUM(ArrDelay) * 100 AS
late_aircraft_delay_percentage
FROM flight_answer
WHERE ArrDelay > 0;
```

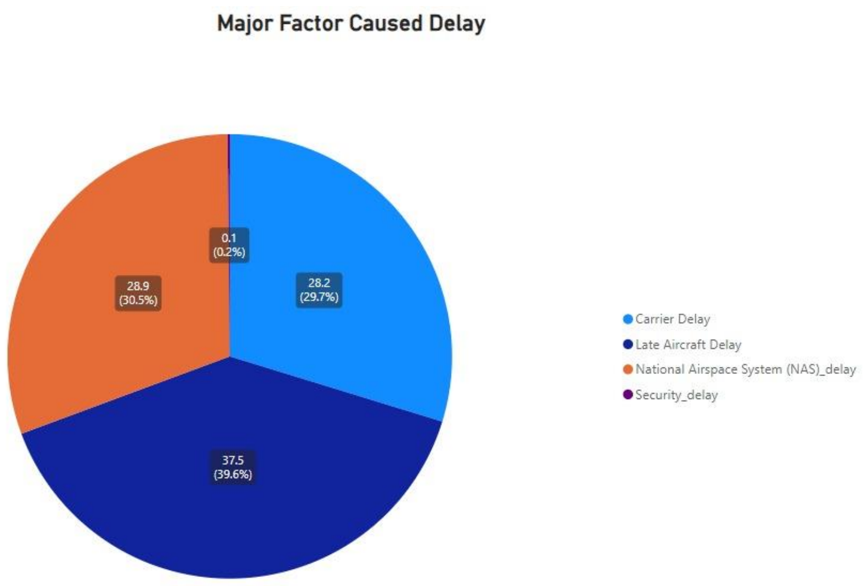


Figure 12: Power BI Visualization of Major Factor Causing Delay

f. Major Factor Causing Cancellations

Since there are no canceled flights in the dataset (canceled_flights = 0), It's clear that there are no records with cancellations to analyze for the question about the major factor causing flight cancellations.

```sql
-- Look for Cancelled Flight
SELECT
    COUNT(*) AS total_flights,
    SUM(Cancelled) AS canceled_flights
FROM flight_answer;
```

Figure 13: Power BI Visualization of Total Flight Cancellation

g. Most Delayed and Frequently Canceled Flights

The flight represented by Flight Number 64 is the most delayed as shown in the visualization with 36 flights.

```sql
SELECT
    FlightNum,
    TailNum,
    COUNT(*) AS num_delays,  -- Count how often this flight was delayed
    ROUND(AVG(ArrDelay)) AS avg_arrival_delay,  -- Average delay for this
flight
    MAX(ArrDelay) AS max_arrival_delay  -- Maximum delay experienced
FROM flight_answer
WHERE ArrDelay > 0  -- Only consider delayed flights
GROUP BY FlightNum, TailNum
ORDER BY num_delays DESC, max_arrival_delay DESC
LIMIT 1;  -- Get the most delayed flight
```

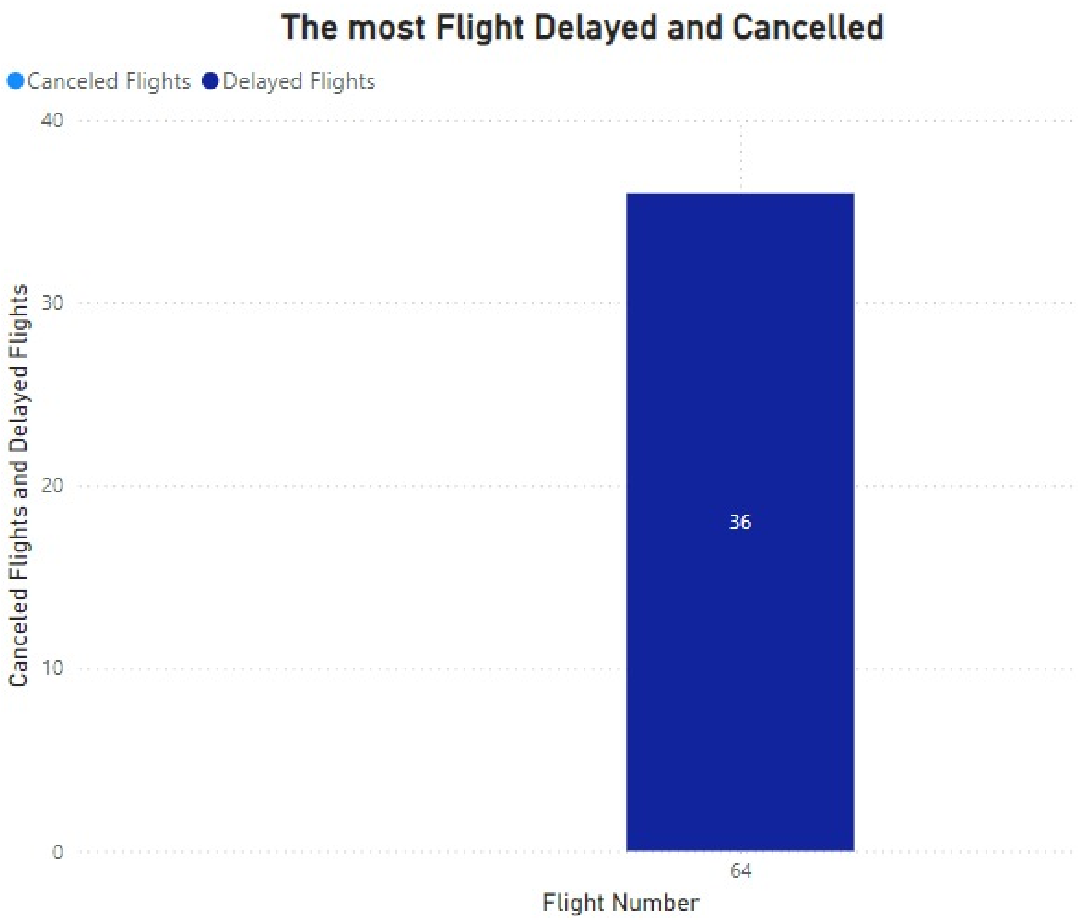


Figure 14: Power BI Visualization of Most Delayed and Frequently Canceled Flights

h. Worst Time to Travel

The first graph which is in Figure 15, "Worst Time of Day," shows the average arrival delay in minutes plotted against different departure times throughout the day. Notably, the worst times to depart are in the early morning hours, specifically around 3 AM, where delays peak dramatically, reaching as high as 856 minutes. Following this peak, there is another significant increase in delays around 6 AM to 8 AM. The graph generally exhibits higher variability and delay peaks during the morning hours up until about 9 AM, after which delays decrease substantially and maintain a lower, more stable pattern throughout the rest of the day. This suggests that nighttime to early morning flights tend to face the longest delays, possibly due to factors like staffing, air traffic, or overnight logistical build-ups.

```sql
-- Worst Time of Day
SELECT
    FLOOR(DepTime) AS hour,  -- Extract the hour from the float time
    ROUND(AVG(ArrDelay)) AS avg_arrival_delay
FROM flight_answer
WHERE ArrDelay IS NOT NULL  -- Exclude null delays
GROUP BY hour
ORDER BY avg_arrival_delay DESC;
```

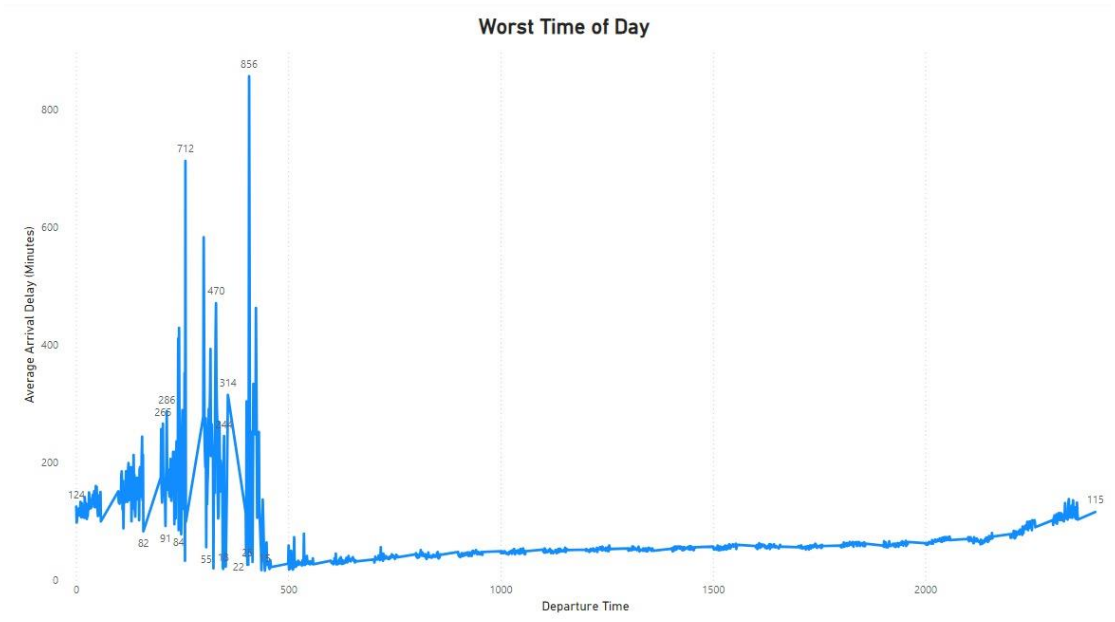


Figure 15: Power BI Visualization of Worst Time to Travel of Day

The second visualization, "Worst Time of Week," which is depicted in Figure 16 indicates that the mid-week, specifically Wednesday, experiences the highest average arrival delays, peaking at 57 minutes. The start of the week shows lesser delays, and there's a notable dip on Tuesdays and Saturdays, where the delays are the lowest. This pattern could reflect variations in air traffic, business travel concentrations, and perhaps airline scheduling efficiencies, which often adjust based on expected weekly travel flows.

```sql
-- Worst Day of the Week
SELECT
    DayOfWeek,
    ROUND(AVG(ArrDelay)) AS avg_arrival_delay
FROM flight_answer
WHERE ArrDelay IS NOT NULL
GROUP BY DayOfWeek
ORDER BY avg_arrival_delay DESC;
```

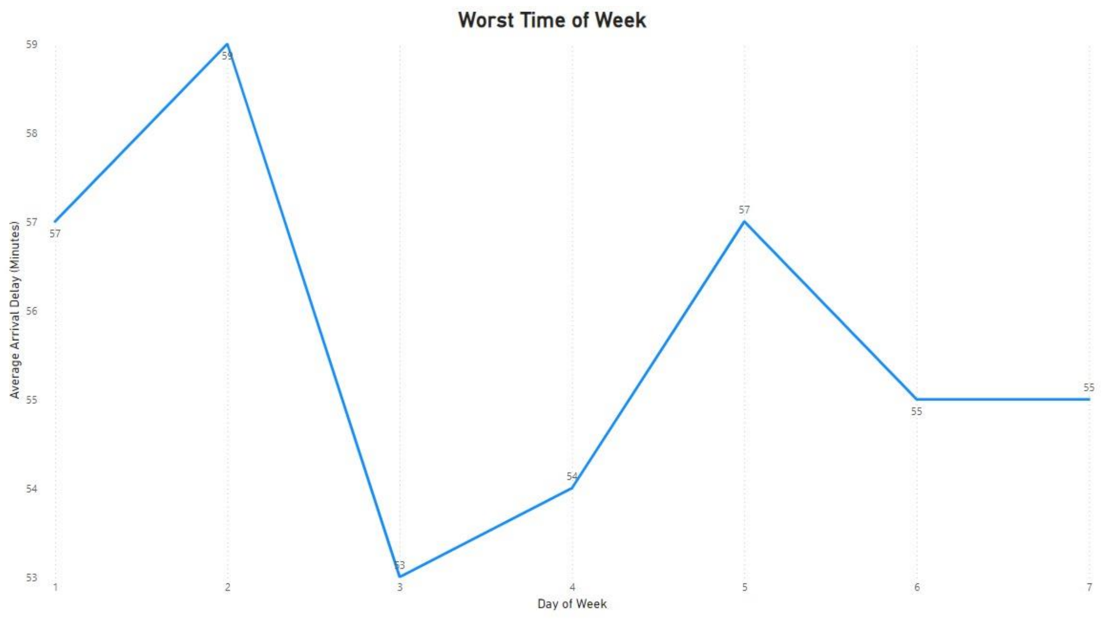


Figure 16: Power BI Visualization of Worst Time to Travel of Week

The third and final visualization which is Figure 17, "Worst Time of Year," shows that January is the worst month to travel, with average arrival delays peaking at 58 minutes. Following January, delays sharply decreased, reaching their lowest in April. This seasonal trend might be influenced by winter weather conditions, increased holiday travel in January, or operational challenges during colder months, leading to higher chances of delays.

```sql
-- Worst Month of the Year
SELECT
    Month,
    ROUND(AVG(ArrDelay)) AS avg_arrival_delay
FROM flight_answer
WHERE ArrDelay IS NOT NULL
GROUP BY Month
ORDER BY avg_arrival_delay DESC;
```

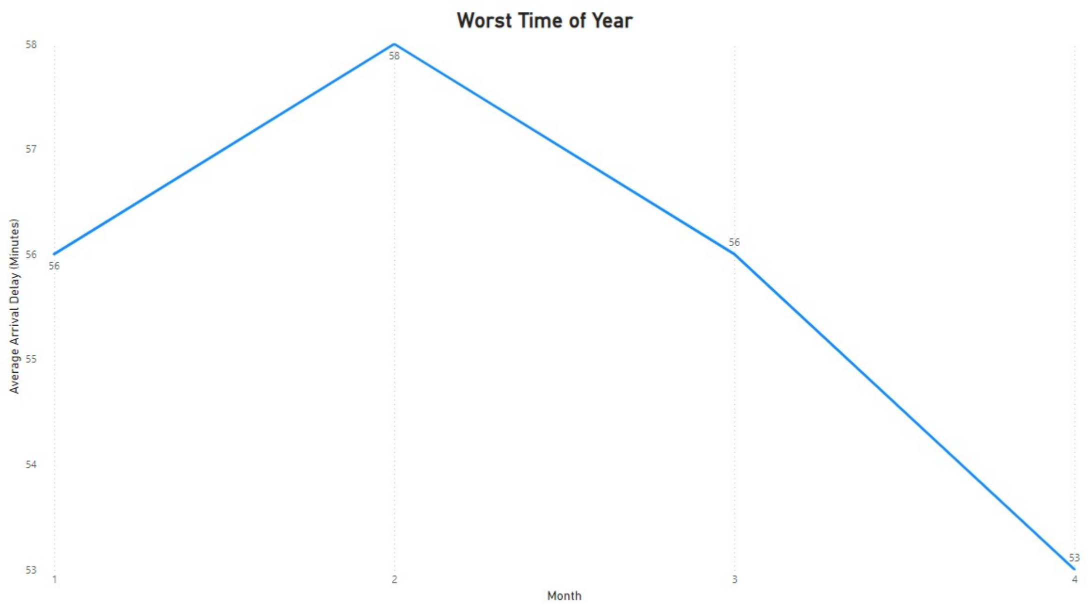


Figure 17: Power BI Visualization of Worst Time to Travel of Year

4. Conclusion

According to the data, the ideal time to fly to avoid delays is early in the morning (6 AM to 9 AM) on Tuesdays or Wednesdays. At the same time, the worst periods are late afternoon (4 PM to 8 PM) on Fridays and Sundays, especially in July and December. Carrier-related delays, such as operational inefficiencies, were the most common cause of flight delays, with weather playing a minor but significant influence.

To improve the flying experience, airlines should prioritize streamlining operations during peak travel periods and improving weather management systems. Passengers may reduce the probability of delays by booking early morning flights and avoiding peak travel periods.

6. Appendices

- Full Code: Hive script used to create the tables for the analysis.

```
CREATE TABLE flight_data (
    Year INT,
    Month INT,
    DayofMonth INT,
    DayOfWeek INT,
    DepTime FLOAT,
    CRSDepTime INT,
    ArrTime FLOAT,
    CRSArrTime INT,
    UniqueCarrier VARCHAR(10),
    FlightNum INT,
    TailNum VARCHAR(10),
    ActualElapsedTime FLOAT,
    CRSElapsedTime FLOAT,
    AirTime FLOAT,
    ArrDelay FLOAT,
    DepDelay FLOAT,
    Origin VARCHAR(10),
    Dest VARCHAR(10),
    Distance INT,
    TaxiIn FLOAT,
    TaxiOut FLOAT,
    Cancelled INT,
    CarrierDelay FLOAT,
    WeatherDelay FLOAT,
    NASDelay FLOAT,
    SecurityDelay FLOAT,
    LateAircraftDelay FLOAT,
    Diverted INT
);


-- Step 1: Create the flight_answer table with the same structure as
flight_data
CREATE TABLE flight_answer (
    Year INT,
    Month INT,
    DayofMonth INT,
    DayOfWeek INT,
    DepTime FLOAT,
    CRSDepTime INT,
    ArrTime FLOAT,
    CRSArrTime INT,
    UniqueCarrier VARCHAR(10),
    FlightNum INT,
    TailNum VARCHAR(10),
    ActualElapsedTime FLOAT,
    CRSElapsedTime FLOAT,
    AirTime FLOAT,
    ArrDelay FLOAT,
    DepDelay FLOAT,
    Origin VARCHAR(10),
    Dest VARCHAR(10),
    Distance INT,
    TaxiIn FLOAT,
    TaxiOut FLOAT,
    Cancelled INT,
    CarrierDelay FLOAT,
    WeatherDelay FLOAT,
    NASDelay FLOAT,
    SecurityDelay FLOAT,
    LateAircraftDelay FLOAT,
    Diverted INT
);

-- Step 2: Insert data into flight_answer from flight_data, excluding rows
with null or empty cells
INSERT INTO flight_answer
SELECT *
FROM flight_data
WHERE
    Year IS NOT NULL AND
    Month IS NOT NULL AND
    DayofMonth IS NOT NULL AND
    DayOfWeek IS NOT NULL AND
    DepTime IS NOT NULL AND
    CRSDepTime IS NOT NULL AND
    ArrTime IS NOT NULL AND
    CRSArrTime IS NOT NULL AND
    UniqueCarrier IS NOT NULL AND
    FlightNum IS NOT NULL AND
    TailNum IS NOT NULL AND
    ActualElapsedTime IS NOT NULL AND
    CRSElapsedTime IS NOT NULL AND
    AirTime IS NOT NULL AND
    ArrDelay IS NOT NULL AND
    DepDelay IS NOT NULL AND
    Origin IS NOT NULL AND
    Dest IS NOT NULL AND
    Distance IS NOT NULL AND
    TaxiIn IS NOT NULL AND
    TaxiOut IS NOT NULL AND
    Cancelled IS NOT NULL AND
    CarrierDelay IS NOT NULL AND
    WeatherDelay IS NOT NULL AND
    NASDelay IS NOT NULL AND
    SecurityDelay IS NOT NULL AND
```

```sql
    LateAircraftDelay IS NOT NULL AND
    Diverted IS NOT NULL;
```