Final Project Report

Flight Delay Analysis and Prediction

CPSC 531: Advanced Database Management Systems

Instructor: Tseng Ching James Shen, PhD

Team Members

Bhuvan Tenkayala - 837408988

Anish Ummenthala - 866515844

Introduction:

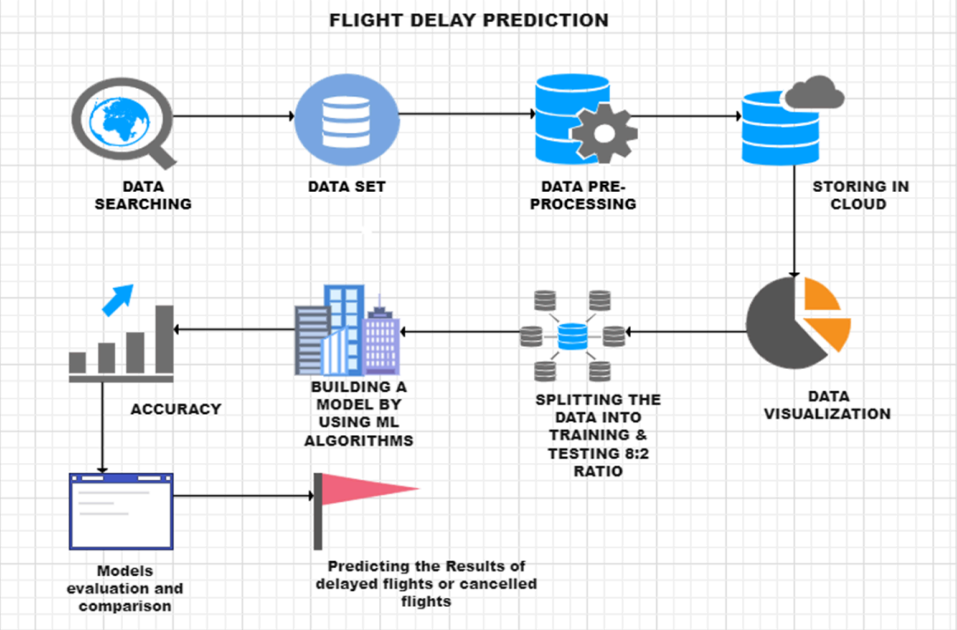
In today's fast-paced world, air travel plays a crucial role in connecting people and businesses across the globe. However, one persistent challenge that plagues the aviation industry is flight delays. Whether caused by weather conditions, air traffic congestion, mechanical issues, or other factors, flight delays can disrupt travel plans, incur additional costs, and inconvenience passengers and airlines alike. Therefore, there is a growing need for effective solutions to analyze and predict flight delays accurately.

Objective:

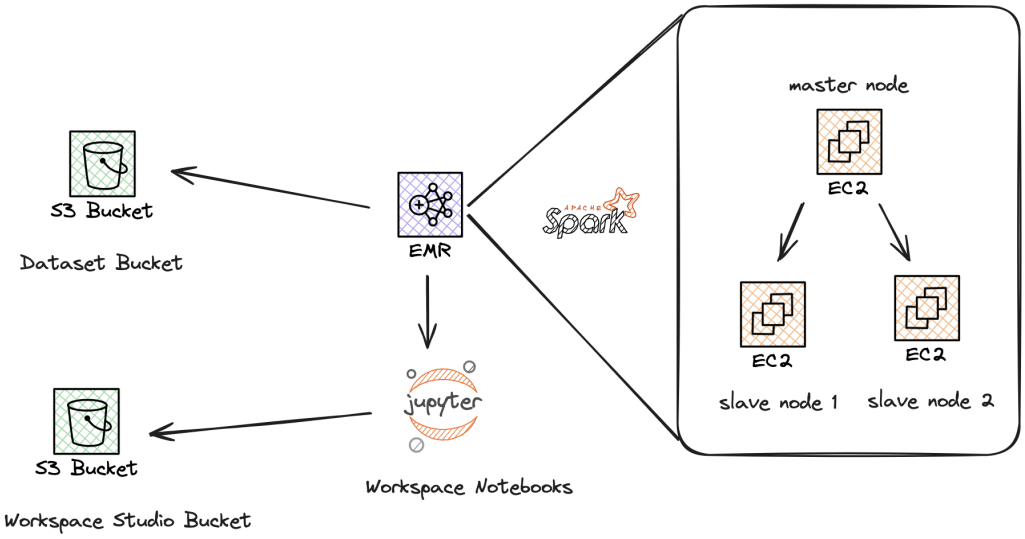
The aim of this project is to analyze the dataset and develop a machine learning model that can predict flight delays based on various features such as operating airline, origin airport, destination airport, scheduled departure time, and historical delay information. Flight delays can cause inconvenience to passengers and airlines, leading to increased costs and decreased customer satisfaction. By accurately predicting flight delays, airlines can take proactive measures to minimize disruptions and improve overall operational efficiency.

This project also aims to develop a predictive analysis that can accurately forecast flight delays based on historical data from 2023. By leveraging various features such as departure time, airline, origin and destination airports, and historical flight performance, we aim to predict the likelihood of flight delays. Using the distributed computing capabilities of Hadoop and the PySpark API, the analysis aims to handle large volumes of data efficiently and produce accurate delay predictions.

Workflow:



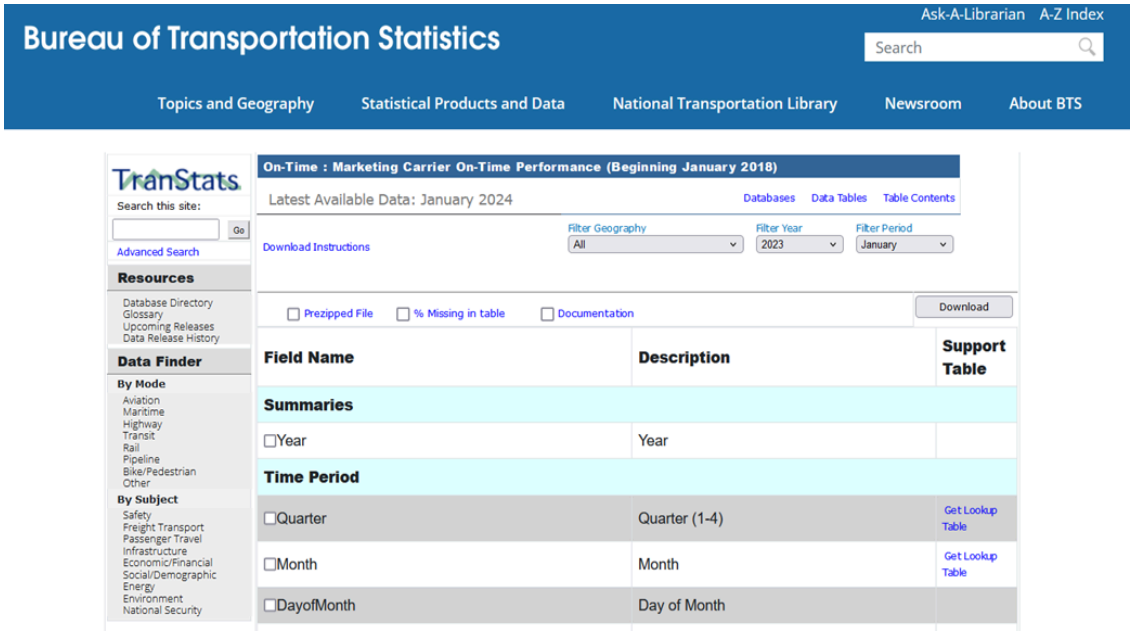
Architecture:



Overview of Dataset:

Dataset is obtained from the Bureau of Transportation Statistics, United States Transportation Department. The website contains all flight information including delays by airline for dates back from January 2018 to January 2024. There are 61 parameters in the dataset and the number of rows in the dataset are 1936758.

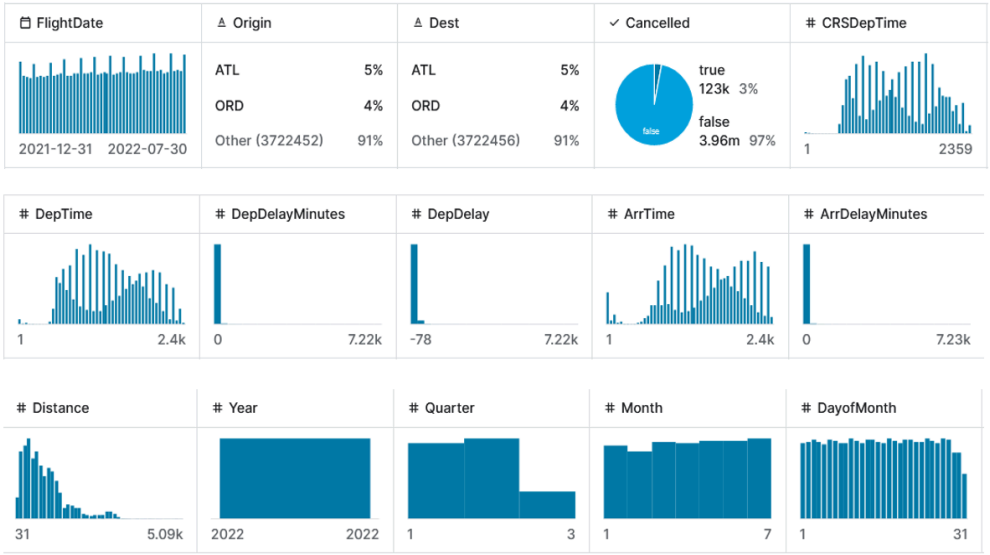
Dataset Link: link



Below are key parameters commonly utilized in flight delay analysis and prediction:

1. Scheduled Departure Time: The designated time for a flight's departure provides insights into potential congestion and operational challenges at airports during different times of the day.
2. Scheduled Arrival Time: Similar to scheduled departure time, the planned arrival time aids in identifying patterns related to arrival delays.
3. Flight Number: Unique identifiers for individual flights, allowing for tracking and analysis of specific routes and carrier operations.

4. Origin Airport: The airport from which the flight departs, providing information on airport-specific factors affecting departure delays.
5. Destination Airport: The airport to which the flight is headed, influencing factors related to arrival delays and airport operations.
6. Scheduled Elapsed Time: The anticipated duration of the flight, with longer flights potentially facing more challenges and delays.
7. Departure Delay: The time difference between the actual and scheduled departure times, a common cause of overall flight delays.
8. Arrival Delay: The time difference between the actual and scheduled arrival times, affected by departure delays and other factors.
9. Delay Categories: Various delay categories, such as carrier delays, weather-related delays, air traffic control issues, and security delays, provide insights into the specific causes of delays.
10. Day of the Week: The day on which the flight operates, influencing travel patterns and potential congestion.
11. Month: The month of the year, revealing seasonal variations, holiday-related travel, and peak travel times.
12. Year: Analysis of trends over time helps in understanding long-term patterns and changes in delay rates.

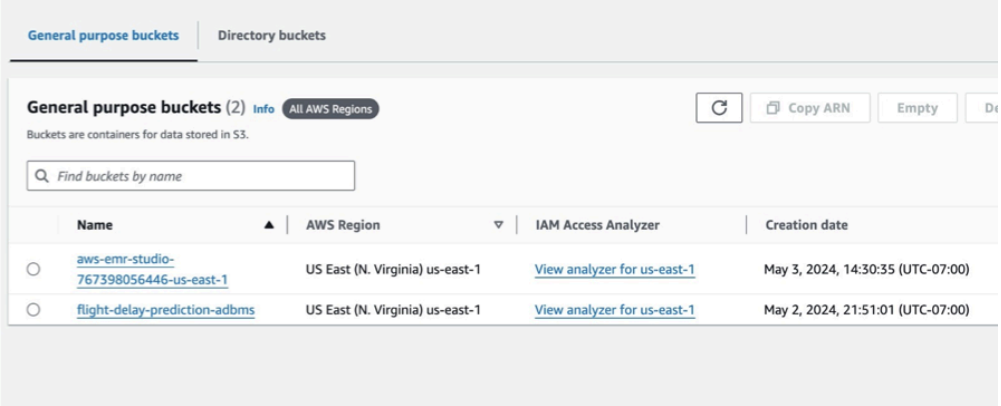


Approach:

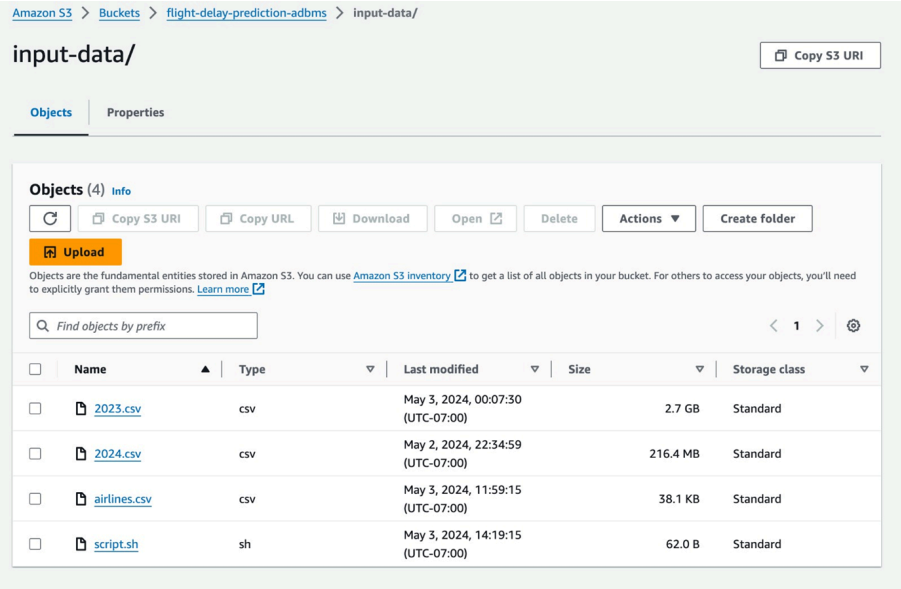
We have implemented the flight delay analysis and prediction system on AWS services. We have utilized AWS S3 as the primary data storage solution, where we uploaded the historical flight dataset securely into buckets. We have configured AWS EMR (Elastic MapReduce) to enable distributed data processing using Hadoop and Spark. EMR clusters have been provisioned to handle processing tasks efficiently, with customized configurations optimized for performance and scalability. EMR workspaces have been established for code development and deployment, facilitating seamless integration with EMR clusters. We have deployed the codebase using PySpark and other relevant libraries, to the EMR workspaces.

Infrastructure Setup:

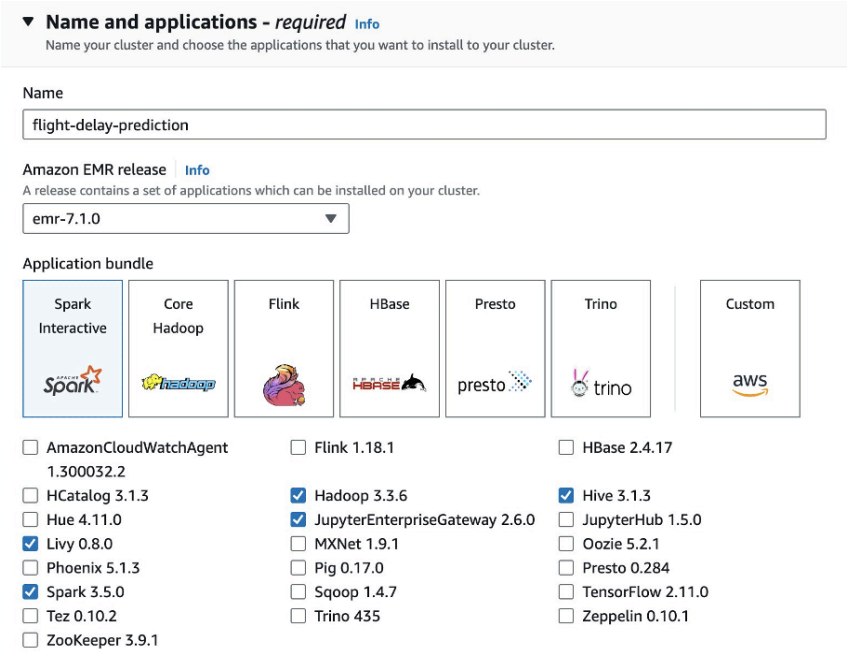
- AWS account creation: We have created an AWS account to gain access to AWS services and resources. Necessary information and payment details were provided to set up the account.
- AWS S3 Setup: Simple Storage Service (S3), a scalable object storage service, was configured to store the flight and airlines dataset securely. S3 buckets were created to organize and manage the dataset effectively.

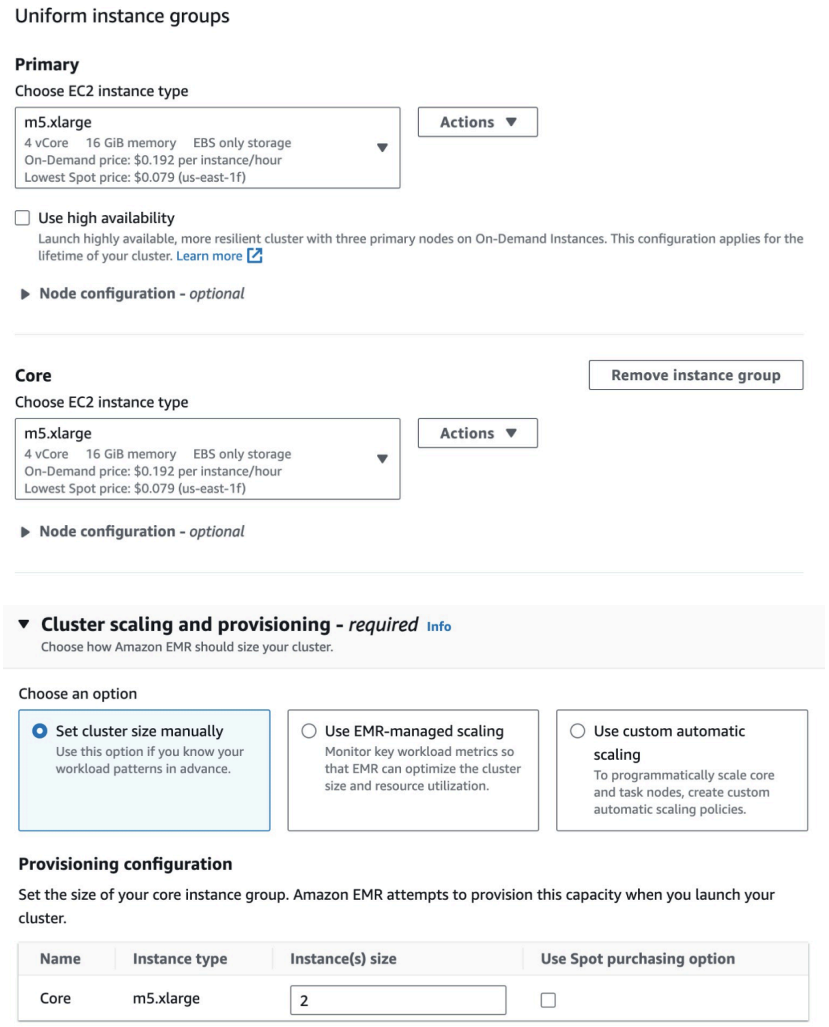


- Storing Dataset into Created Buckets: Uploaded the historical flight dataset and airline dataset to the S3 buckets created in the previous step.

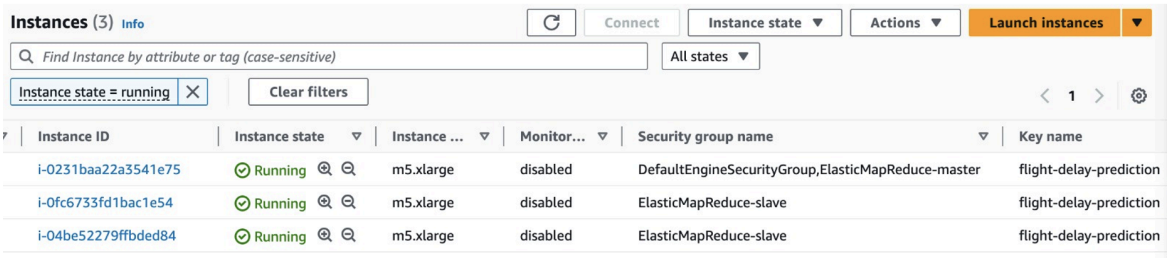


- EMR Setup: Elastic MapReduce was configured to enable distributed data processing using Hadoop and Spark. EMR settings were customized to meet the specific requirements of the flight delay analysis tasks.

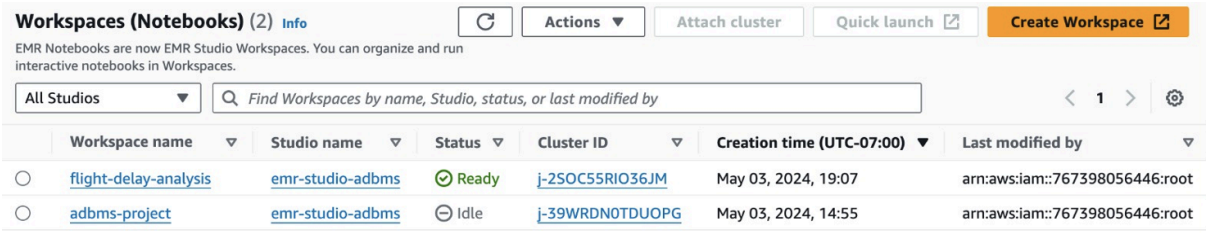


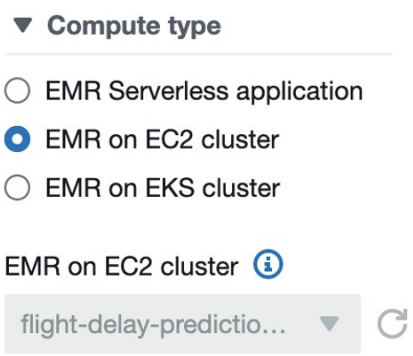


Behind the scenes, AWS EMR dynamically provisions EC2 instances as compute nodes within the clusters. These instances are automatically scaled up or down based on workload requirements, optimizing resource utilization and cost efficiency.



- EMR Cluster and Workspace Setup: EMR workspaces were set up for development and analysis tasks, providing a collaborative environment for writing, testing, and running code on EMR clusters using the jupyter notebook, which is enabled using the Jupyter Enterprise Gateway. We link clusters to the jupyter notebooks.





- Loading Code into EMR Workspaces: The code was loaded into EMR workspaces to facilitate development and execution within the EMR environment. The paths to the dataset were updated, which varies from local setup.
- Executing the code: After successfully loading the code into the EMR workspaces, we can execute the code blocks similar to the local jupyter notebook, but now on the cloud.

Implementation Approach:

- We imported the required packages such as pyspark-sql, pyspark-ml and matplot library.
- After that we have created a spark session by using spark session builder.

Importing the required packages

In [1]:
```
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
import matplotlib.pyplot as plt
```

Creating Spark Session

In [2]:
```
spark = SparkSession.builder\
                    .appName("FlightDelayAnalysis")\
                    .getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/05/03 16:09:40 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

- We loaded the flight and airline dataset using spark read command and explored the dataset to understand its structure, features, and any data preprocessing that may be required.

Loading the dataset

```
df = spark.read.csv("data/flights/", header=True, inferSchema=True)
df.printSchema()

airlines_df = spark.read.csv("data/airlines.csv", header=True, inferSchema=True)
airlines_df.printSchema()
```

```
[Stage 1:=============================================>           (17 + 5) / 22]
root
 |-- Year: integer (nullable = true)
 |-- Quarter: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- DayofMonth: integer (nullable = true)
 |-- DayOfWeek: integer (nullable = true)
 |-- FlightDate: date (nullable = true)
 |-- Marketing_Airline_Network: string (nullable = true)
 |-- Operated_or_Branded_Code_Share_Partners: string (nullable = true)
 |-- DOT_ID_Marketing_Airline: integer (nullable = true)
 |-- IATA_Code_Marketing_Airline: string (nullable = true)
 |-- Flight_Number_Marketing_Airline: integer (nullable = true)
 |-- Originally_Scheduled_Code_Share_Airline: string (nullable = true)
```

- We performed data preprocessing by handling the missing values in the dataset by either imputing them or dropping rows/columns with missing values, depending on the extent of missing data.
- Duplicate rows are removed from the dataset to ensure data integrity and prevent redundancy in subsequent analysis.
- Columns are renamed for clarity and consistency, improving the readability and understanding of the dataset structure.

Data Preprocessing

```
# Required columns for the analysis
requiredColumns = [
    "FlightDate", "Year", "Quarter", "DayofMonth", "Month", "DayOfWeek", # Flight date details
    "Operating_Airline ", "Marketing_Airline_Network", # Airline details
    "Origin", "OriginState", "Dest", "DestState", # Airport details
    "CRSArrTime", "ArrDelayMinutes", "ArrDel15", "CRSDepTime", "DepDelayMinutes", "DepDel15", # Time details
    "CarrierDelay", "WeatherDelay", "NASDelay", "SecurityDelay", "LateAircraftDelay", # Delay reason details
    "Cancelled", "CancellationCode", # Cancellation details
    'AirTime', 'Distance', 'TaxiIn', 'TaxiOut' # Additional details
]

# Dropping duplicate rows
df = df.dropDuplicates()

# Keeping necessary columns and dropping others
df = df[requiredColumns]

# Renaming columns
df = df.withColumnRenamed("Operating_Airline ","OperatingAirline")
df = df.withColumnRenamed("Marketing_Airline_Network","MarketingAirline")
df = df.withColumnRenamed("CRSDepTime","ScheduledDepTime")
df = df.withColumnRenamed("CRSArrTime","ScheduledArrTime")

df.printSchema()
```

- We performed Exploratory Data Analysis (EDA) to analyze the distribution of flight delays and cancellations over different time periods (e.g., day of week, month, year) and a lot more.
- We identified relevant features for predicting flight delays and cancellations. We selected specific columns relevant to flight delay analysis, such as flight date, airline details, airport information, time-related features, delay reasons, and additional flight details.

Data Visualization:

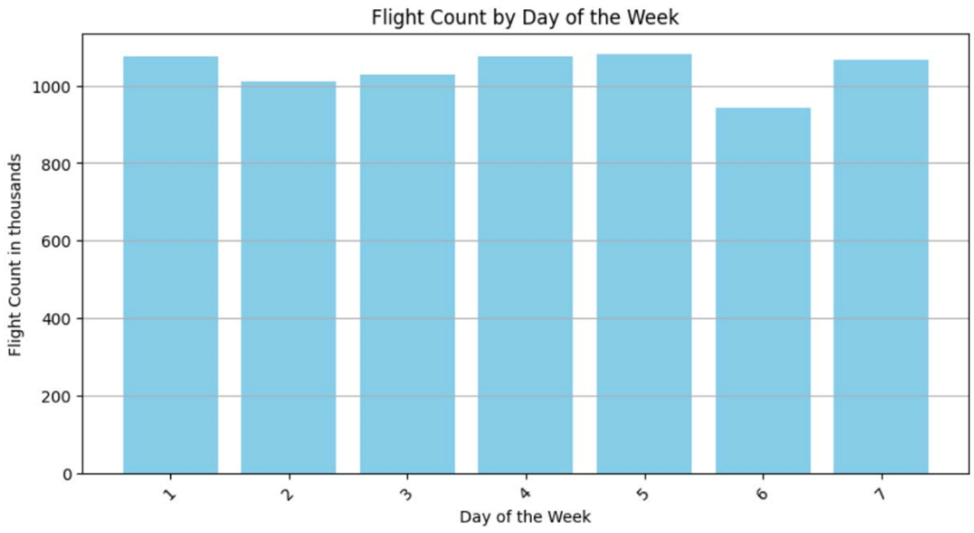
The below chart aggregates the flight counts based on the day of the week, providing insights into the distribution of flights over different days. We used Matplotlib to generate a bar plot to visualize the flight count variation across days of the week.

```
# Analyzing the distribution of flights over different days of the week

flightCountByDayOfWeek = df.groupBy("DayOfWeek").count().orderBy("DayOfWeek").collect()

daysOfWeek = [row["DayOfWeek"] for row in flightCountByDayOfWeek]
flightCountDay = [row["count"] / 1000 for row in flightCountByDayOfWeek]

plt.figure(figsize=(10, 5))
plt.bar(daysOfWeek, flightCountDay, color='skyblue')
plt.title('Flight Count by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Flight Count in thousands')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.ticklabel_format(style='plain', axis='y')
plt.show()
```



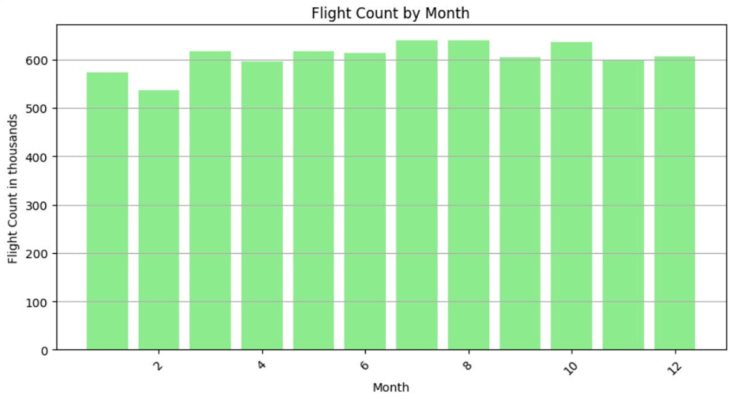
Bar plot below is to visualize the flight count variation across months of the year.

[11]:
```
# Analyzing the distribution of flights over different months

flightCountByMonth = df.groupBy("Month").count().orderBy("Month").collect()

months = [row["Month"] for row in flightCountByMonth]
flightCountMonth = [row["count"] / 1000 for row in flightCountByMonth]

plt.figure(figsize=(10, 5))
plt.bar(months, flightCountMonth, color='lightgreen')
plt.title('Flight Count by Month')
plt.xlabel('Month')
plt.ylabel('Flight Count in thousands')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.ticklabel_format(style='plain', axis='y')
plt.show()
```



Below is the code for displaying the top 10 airlines with the highest flight counts, including their operating airline codes, names, and corresponding flight counts. By analyzing the airline distribution of flights, it helps in understanding the market share and dominance of different airlines in the dataset.

[22]:
```
# Count the occurrences of flights for each airline network
airline_counts = df.groupBy("OperatingAirline").agg(F.count("*").alias("FlightCount"))

# Join with airline_info_df to get airline description
airline_counts_with_description = airline_counts.join(airlines_df, airline_counts["OperatingAirline"] == airlines_df["Code"], "left_outer") \
    .select(airline_counts["OperatingAirline"], airlines_df["Description"].alias("Airline Name"), airline_counts["FlightCount"])

# Displaying results
print("Airlines with the most flights:")
airline_counts_with_description.show(10, truncate=False)
```

```
Airlines with the most flights:
[Stage 127:=============================================================>   (33 + 2) / 35]
+---------------+-------------------------------+-----------+
|OperatingAirline|Airline Name                   |FlightCount|
+---------------+-------------------------------+-----------+
|UA             |United Air Lines Inc.          |732212     |
|NK             |Spirit Air Lines               |263871     |
|AA             |American Airlines Inc.         |940531     |
|B6             |JetBlue Airways                |274852     |
|PT             |Capital Cargo International    |99047      |
|DL             |Delta Air Lines Inc.           |984986     |
|OO             |SkyWest Airlines Inc.          |675285     |
|F9             |Frontier Airlines Inc.         |177542     |
|YV             |Mesa Airlines Inc.             |88679      |
|MQ             |Envoy Air                      |227505     |
+---------------+-------------------------------+-----------+
only showing top 10 rows
```

Below code aggregates the delay times (arrival and departure) for each airline, giving an overview of delay trends across different carriers. It ranks airlines based on their total delay time, enabling the identification of the top 10 airlines with the highest delay times. Through this analysis, insights are derived regarding the airlines that encounter the most delays, offering valuable information for operational optimization and customer service enhancement without copying from other sources.

[24]:
```
# Calculate total delay time (arrival delay + departure delay) for each airline
airline_delays = df.groupBy("OperatingAirline")\
    .agg((F.sum("ArrDelayMinutes") + F.sum("DepDelayMinutes")).alias("TotalDelayMinutes"))

# Order by total delay time to find airlines with the most delays and limit to top 10
top_10_airlines = airline_delays.orderBy("TotalDelayMinutes", ascending=False).limit(10)

# Join with airline_info_df to get airline names
top_10_airlines_with_description = top_10_airlines.join(airlines_df, top_10_airlines["OperatingAirline"] == airlines_df["Code"], "left_outer")
    .select(top_10_airlines["OperatingAirline"], airlines_df["Description"].alias("Airline Name"))

# Displaying results
print("Top 10 Airlines with the most delays:")
top_10_airlines_with_description.show(truncate=False)
```

```
Top 10 Airlines with the most delays:
[Stage 141:=====================================================> (34 + 1) / 35]
+---------------+--------------------+
|OperatingAirline|Airline Name        |
+---------------+--------------------+
|AA              |American Airlines Inc.|
|WN              |Southwest Airlines Co.|
|DL              |Delta Air Lines Inc.  |
|UA              |United Air Lines Inc. |
|OO              |SkyWest Airlines Inc. |
|B6              |JetBlue Airways       |
|NK              |Spirit Air Lines      |
|F9              |Frontier Airlines Inc.|
|AS              |Alaska Airlines Inc.  |
|YX              |Republic Airlines     |
+---------------+--------------------+
```

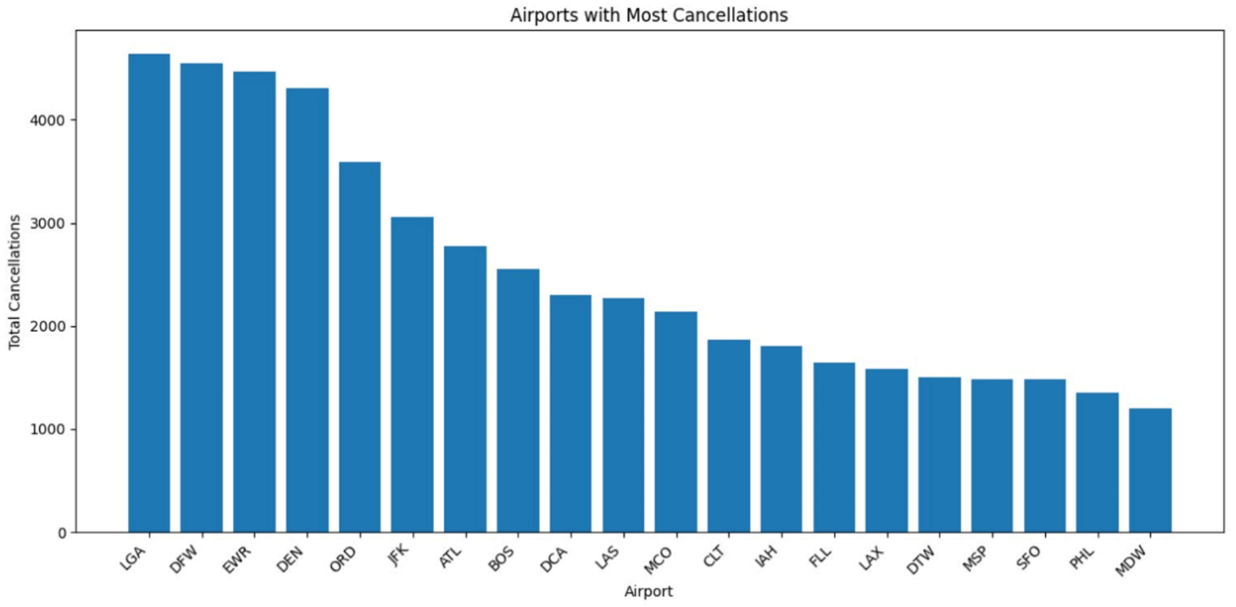
Below code groups flight data by the origin airport, allowing for the calculation of the number of flights departing from each airport. The busiest airports are identified by sorting the count of flights in descending order and limiting the results to the top 20 airports. A bar chart is generated to visually represent the busiest airports, aiding in the identification of major hubs and traffic distribution and also airports with the most cancellations.

[33]:
```
# Group by Origin airport and sum cancellations
most_cancellations_airports = df.groupBy("Origin").agg(F.sum("Cancelled").alias("TotalCancellations")).orderBy("TotalCancellations", ascending

top_10_airports_with_cancellations = most_cancellations_airports.limit(20)

# Convert most_cancellations_airports dataframe to Pandas for plotting
most_cancellations_airports_pd = top_10_airports_with_cancellations.toPandas()

# Plot
plt.figure(figsize=(12, 6))
plt.bar(most_cancellations_airports_pd['Origin'], most_cancellations_airports_pd['TotalCancellations'])
plt.xlabel('Airport')
plt.ylabel('Total Cancellations')
plt.title('Airports with Most Cancellations')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

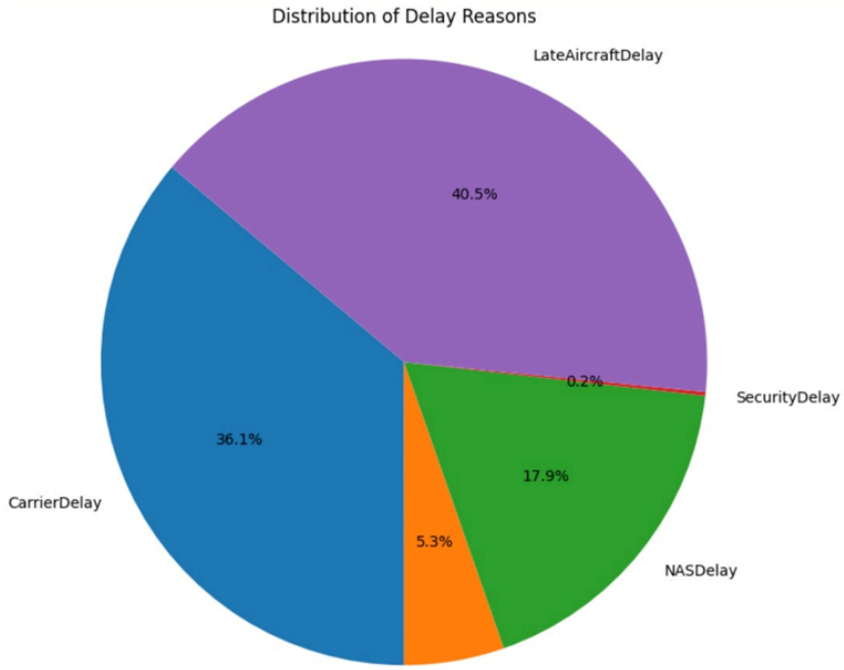


Below code calculates the total delay time for each delay reason (CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay) by aggregating the corresponding columns. A pie chart is generated to visualize the distribution of delay reasons, showcasing the proportion of each delay reason to the total delay time without copying from other sources.

[37]:
```
# Calculate the total delay time for each delay reason
delay_reasons = df.groupBy().agg(
    F.sum("CarrierDelay").alias("CarrierDelay"),
    F.sum("WeatherDelay").alias("WeatherDelay"),
    F.sum("NASDelay").alias("NASDelay"),
    F.sum("SecurityDelay").alias("SecurityDelay"),
    F.sum("LateAircraftDelay").alias("LateAircraftDelay")
)

# Convert the delay reasons DataFrame to Pandas for plotting
delay_reasons_pd = delay_reasons.toPandas()

# Plot the pie chart
labels = delay_reasons_pd.columns.tolist()
sizes = delay_reasons_pd.values.flatten()
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Distribution of Delay Reasons')
plt.show()
```



Delay Prediction using Machine Learning Algorithms:

Gradient Boosting Algorithm

We split our dataframe (model_df) into training and testing data frames using a 70-30 ratio for training and testing respectively. We used Gradient Boosted Tree (GBT) classifiers to predict flight delays. We initialized the classifiers with specific parameters, including the maximum number of iterations and bins. Then we fit the model to the training dataframe to learn from the data and evaluate its accuracy on the testing dataframe. Finally, we print out the accuracy of the model's predictions.

In [22]:
```
# Split the dataset into training and testing sets
training_df, testing_df = model_df.randomSplit([0.7, 0.3])

print(training_df.count())
print(testing_df.count())
```

```
5017410
2149634
```

In [23]:
```
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

In [27]:
```
# Initialize GBTClassifier with increased maxBins
gbt = GBTClassifier(labelCol="isDelayed", featuresCol="features", maxIter=10, maxBins=700)
gbt_model = gbt.fit(training_df)


# Fit the model
train_model = gbt_model.transform(training_df)
test_model = gbt_model.transform(testing_df)

# Make predictions
predictions = gbt_model.transform(testing_df)

# Evaluate model
evaluator = MulticlassClassificationEvaluator(labelCol="isDelayed", predictionCol="prediction", metricName="accuracy")
test_accuracy = evaluator.evaluate(predictions)
print("Accuracy:", test_accuracy)
```

```
Accuracy: 0.7550294608291458
```

Linear Regression

We initialized the regression model with specific parameters, including the features and label columns. Then, we fit the model to the training data and evaluated its performance using the R^2 metric, which measures the proportion of the variance in the dependent variable that is predictable from the independent variables. The R^2 scores for both the training and testing datasets are printed out. Additionally, we calculated mean absolute error, mean squared error, and R^2 score using the RegressionEvaluator. It indicates how well the linear regression model fits the data.

In [12]:
```
# Split the dataset into training and testing sets
training_df, testing_df = model_df.randomSplit([0.7, 0.3])

print(training_df.count())
print(testing_df.count())
```

```
5017743
2149301
```

In [14]:
```
# Train the linear regression model
lr = LinearRegression(featuresCol='features2', labelCol='isDelayed')
lr_model = lr.fit(training_df)

# Evaluate the model
train_predictions = lr_model.transform(training_df)
test_predictions = lr_model.transform(testing_df)

evaluator = RegressionEvaluator(
    labelCol="isDelayed", predictionCol="prediction", metricName="r2")
train_r2 = evaluator.evaluate(train_predictions)
test_r2 = evaluator.evaluate(test_predictions)

print("Training R^2 Score:", train_r2)
print("Testing R^2 Score:", test_r2)
```

```
Training R^2 Score: 0.00293397229804071
Testing R^2 Score: 0.002859896629911929
```

In [17]:
```
pred_results = regressor.evaluate(testing_df)
pred_results.predictions.show(30)
```

```
+-------------+---------+------------------+
|    features2|isDelayed|        prediction|
+-------------+---------+------------------+
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
|[1.0,0.0,1.0]|      0.0|0.2691172897964861|
+-------------+---------+------------------+
only showing top 30 rows
```

Conclusion

The analysis of flight delay prediction using machine learning models offers valuable insights into the factors influencing flight delays. Through the utilization of gradient boosting and linear regression techniques, we have gained a deeper understanding of delay reasons, airport operations, and predictive capabilities. These findings underscore the importance of proactive delay management, operational efficiency, and continuous improvement in enhancing overall performance and customer satisfaction within the aviation sector. By leveraging predictive analytics and embracing data-driven decision-making, stakeholders can optimize resources, minimize delays, and deliver a seamless travel experience for passengers.

Github location of code: https://github.com/Anish-U/flight-delay-analysis