AeroVista

OPTIMIZING AIRLINE OPERATIONS

A Data-Driven Approach to Enhancing On-Time Performance of Ultra Low-Cost Carriers

AeroVista

Team

Jasvinder Singh

Tejaswini Kandula

Prachi Patankar

Chingyu Ting

Yujie Zhang

Jie Luo

AeroVista

Project Aim

Robust Data Pipeline

- Develop a scalable, flexible data ingestion and ETL pipeline. Connect it with a visualization tool.

ULCC Performance Analytics

- Analyze key drivers of ULCC departure delays, benchmark performance, and recommend best practice

AeroVista

Overview

Dataset Description
& Feature Engineering

Data Ingestion & ETL

Findings and Insights

Data Limitations, Future Scope, Appendix

AeroVista

DATASET

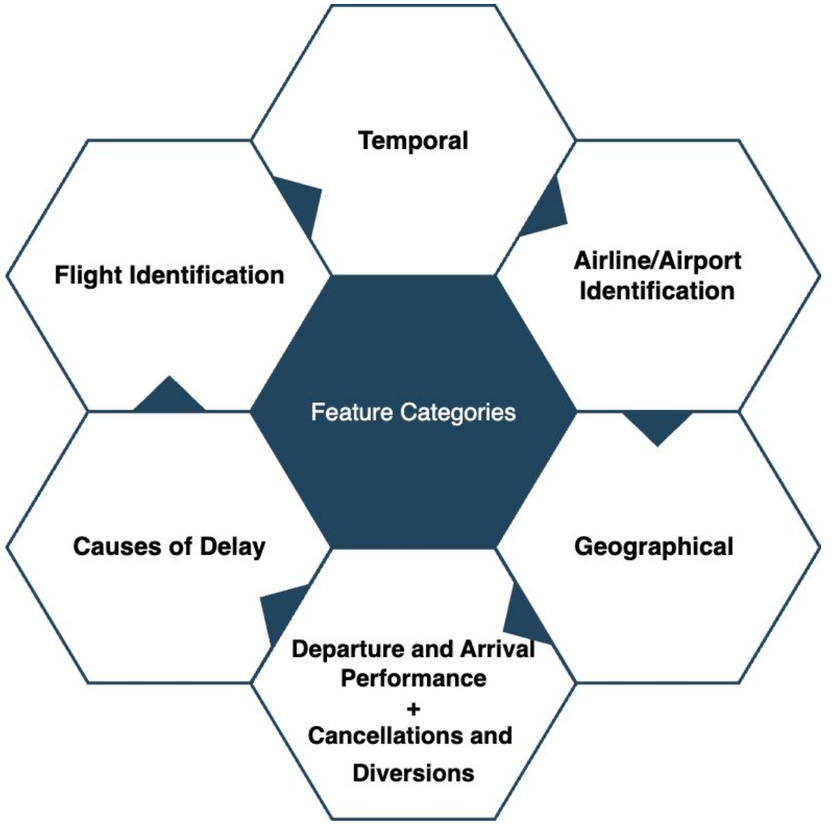
DESCRIPTION & FEATURE ENGINEERING

- **Data Source**: Obtained from the Bureau of Transportation Statistics (BTS), a division of the U.S. Department of Transportation, which provides extensive data on national transportation systems.

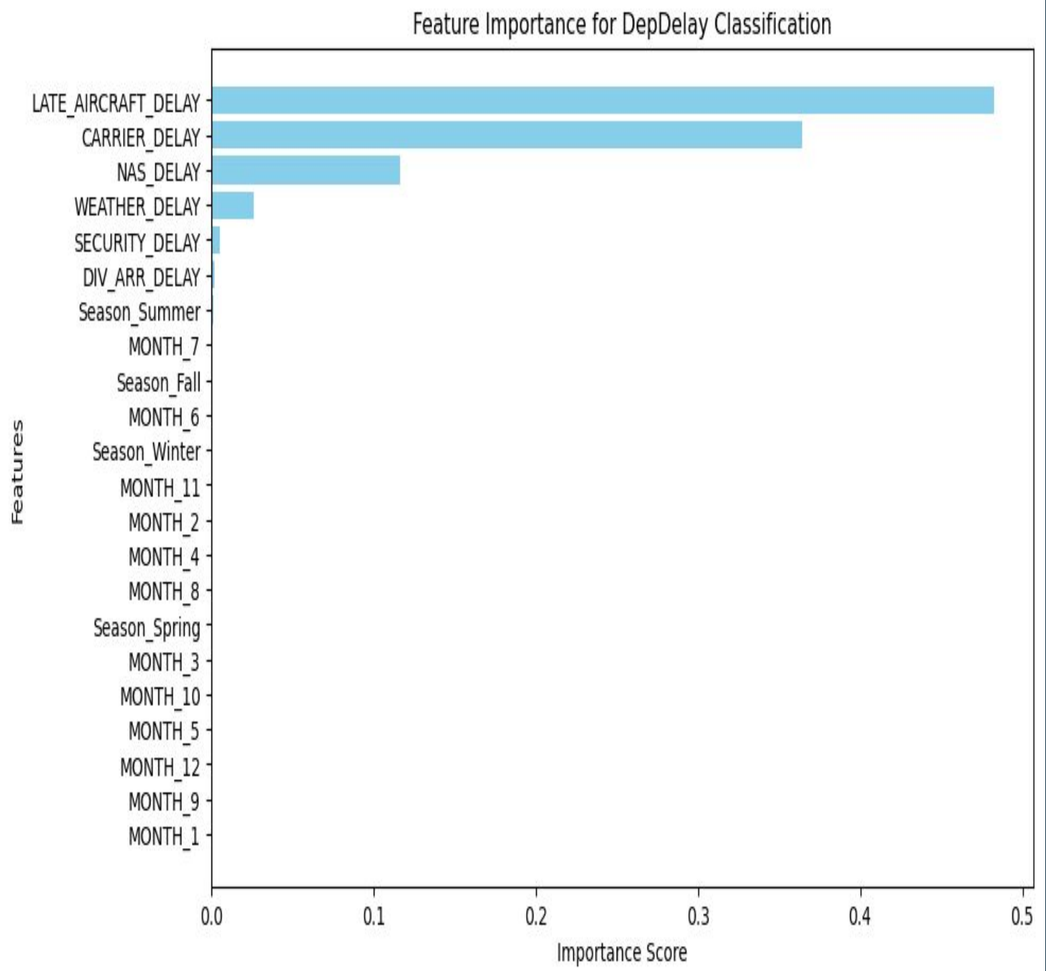
- **Data Collection**: Manually downloaded monthly on-time performance data for 2023, focusing on six key U.S. states: Texas, New York, Illinois, California, Florida, and Georgia.

- **Supplemental Data**: Included additional datasets detailing specific causes of flight delays.

- **Final Dataset**: After merging, the comprehensive dataset consists of 5.9 million records and 79 columns, encompassing a broad spectrum of airline operational metrics.



- **Data Standardization**: Corrected formats for date and time; imputed null delay values with zero.
- **Type Casting**: Ensured all data types were correct for accurate analysis.
- **New Features**: Created variables for Month, State, Season, carrier cost category, holiday season indicator, and a binary delay status based on a 15-minute threshold.
- **Feature Importance Analysis**: Employed Random Forest from Scikit-Learn to identify key predictors of departure delays.



AeroVista

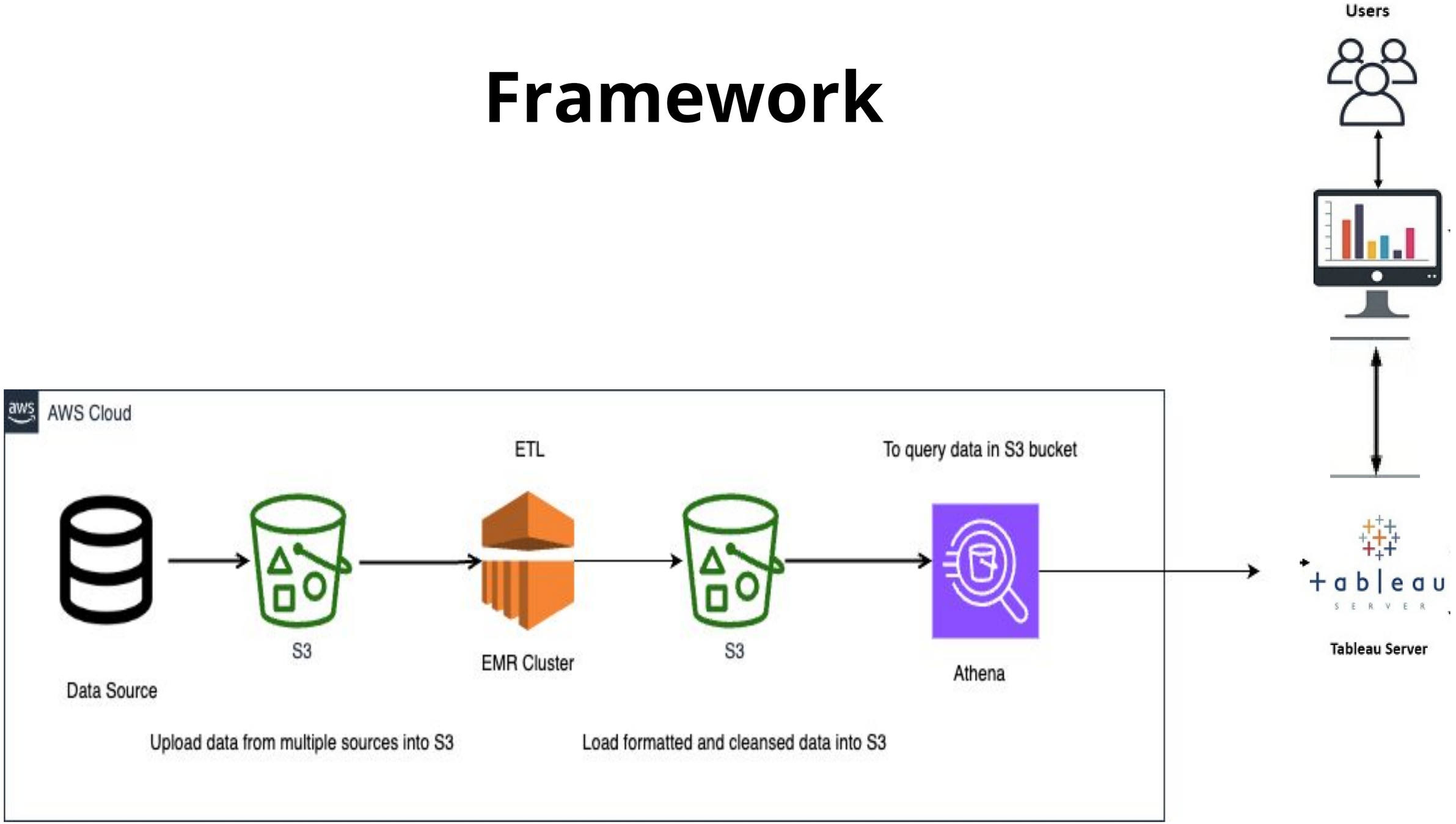
DATA

INGESTION

&

ETL

Framework


Users
AWS Cloud
ETL
To query data in S3 bucket
S3
EMR Cluster
S3
Athena
Tableau Server
Data Source
Upload data from multiple sources into S3
Load formatted and cleansed data into S3

Unveiling the Critical

FINDINGS

Ultra Low cost carriers (ULCCs)

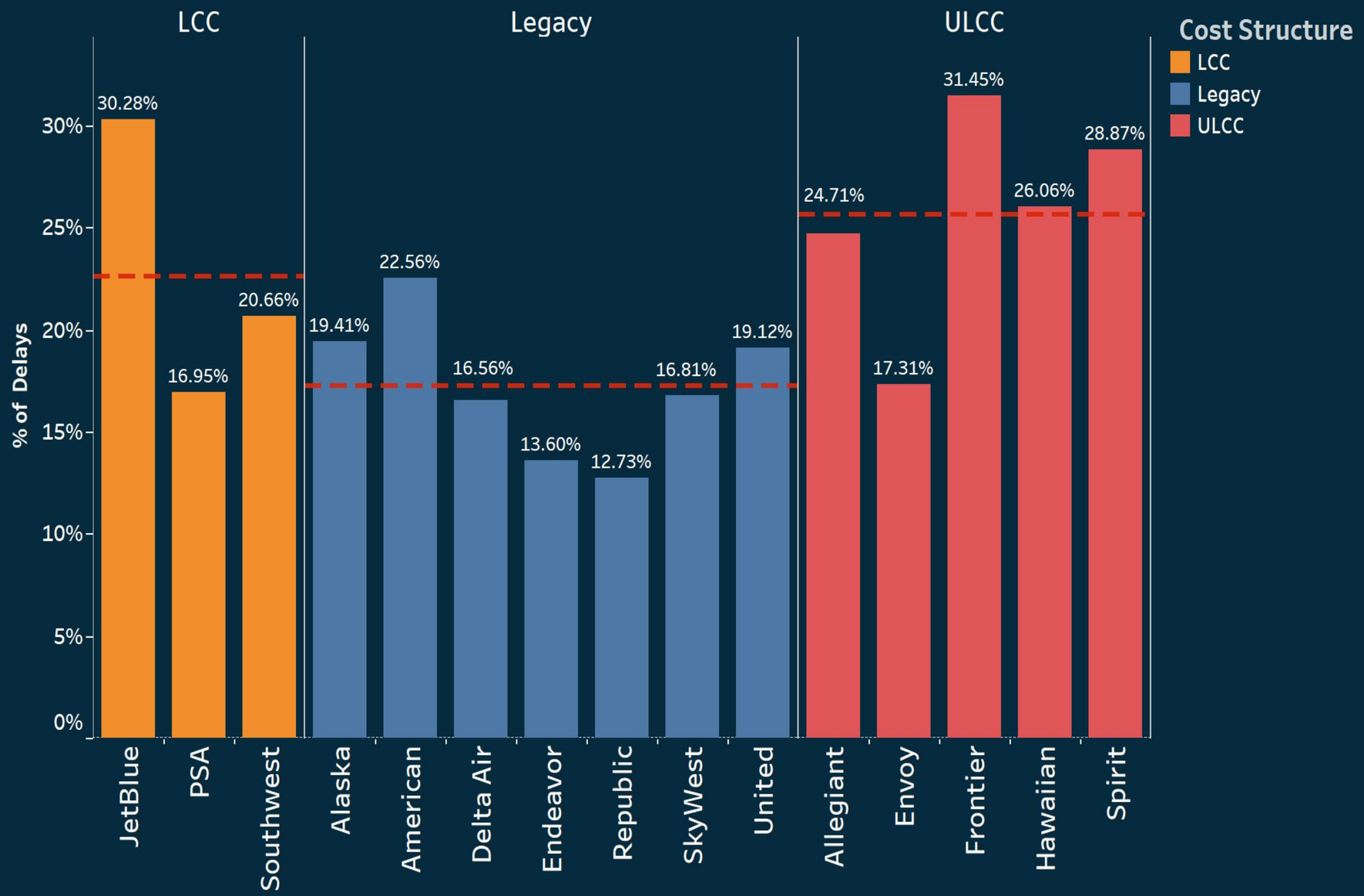
- Market Impact
- Operational Insights
- Industry-Wide Benefits

Departure Delay

- Impact Significance
- Efficiency in Resolution

AeroVista

Low and Lower: Do Flights delay more if they cost less?



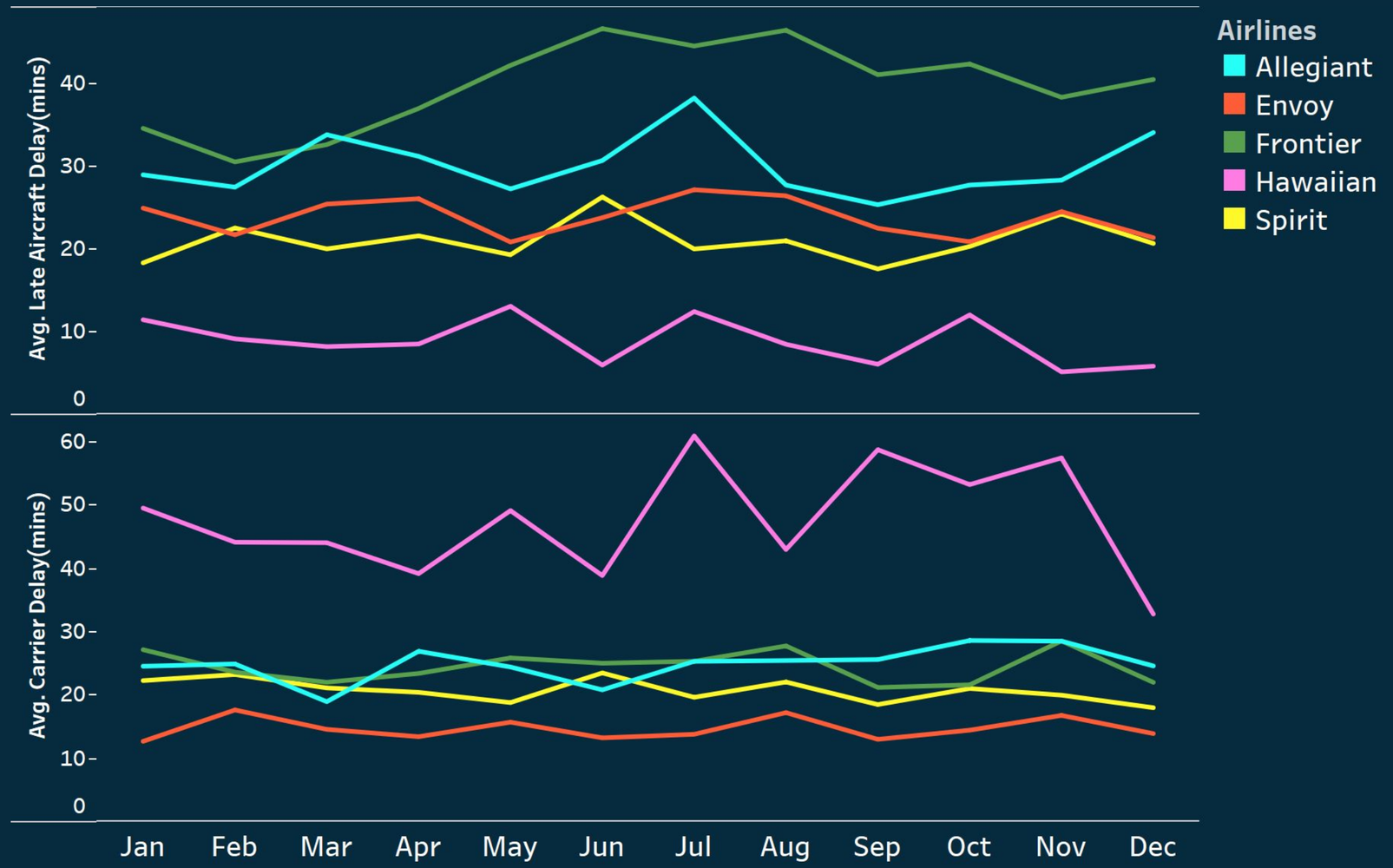
Operational Delays

A Comparative Study of Airline Performance and Operational Efficiency

AeroVista

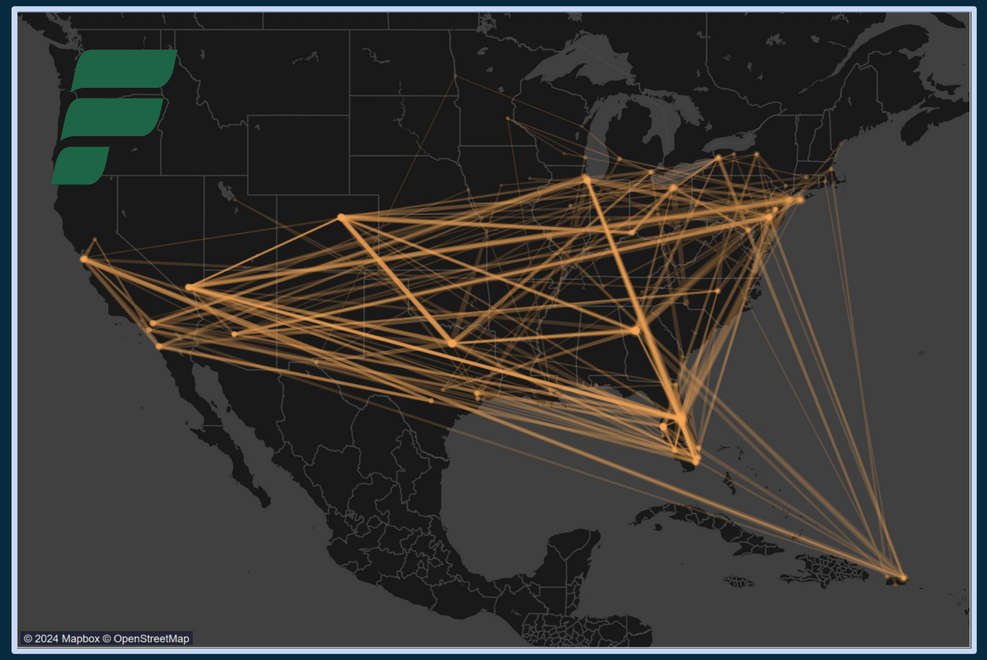
A Comparative Look at ULCCs

- **Envoy's** Unique Advantage
- Learning from **Spirit**

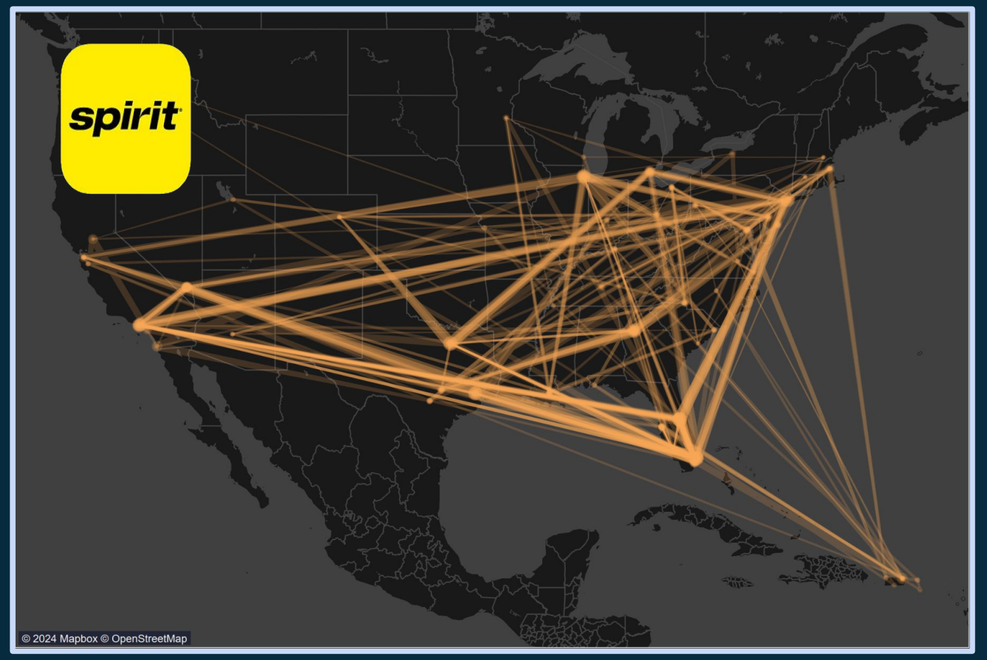


AeroVista

Operational Network Comparison: Frontier vs. Spirit Airlines



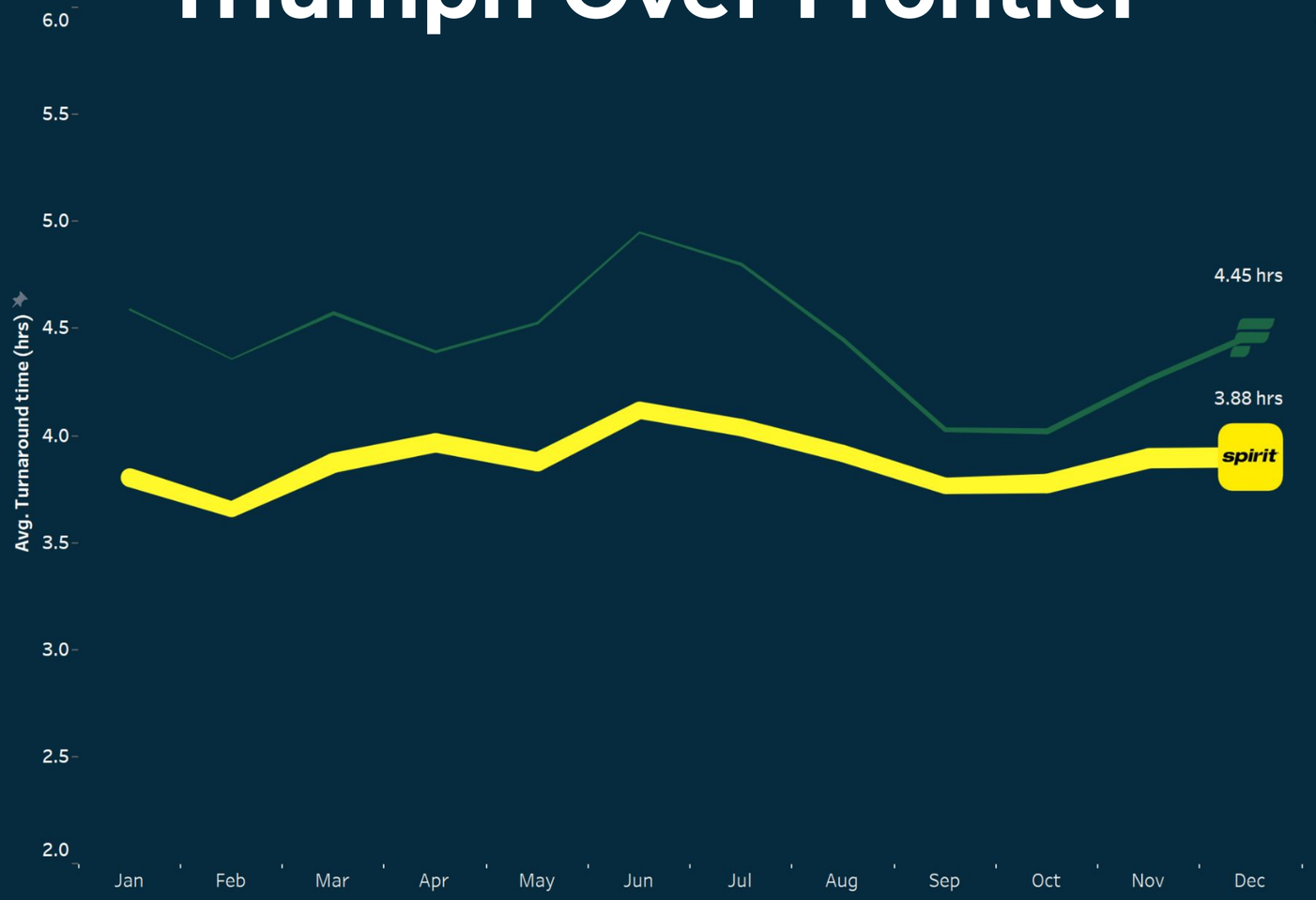
Frontier: Fewer and less densely connected routes



Spirit: Intricate and dense flight network

AeroVista

Spirit Speeds Ahead: Turnaround Time Triumph Over Frontier



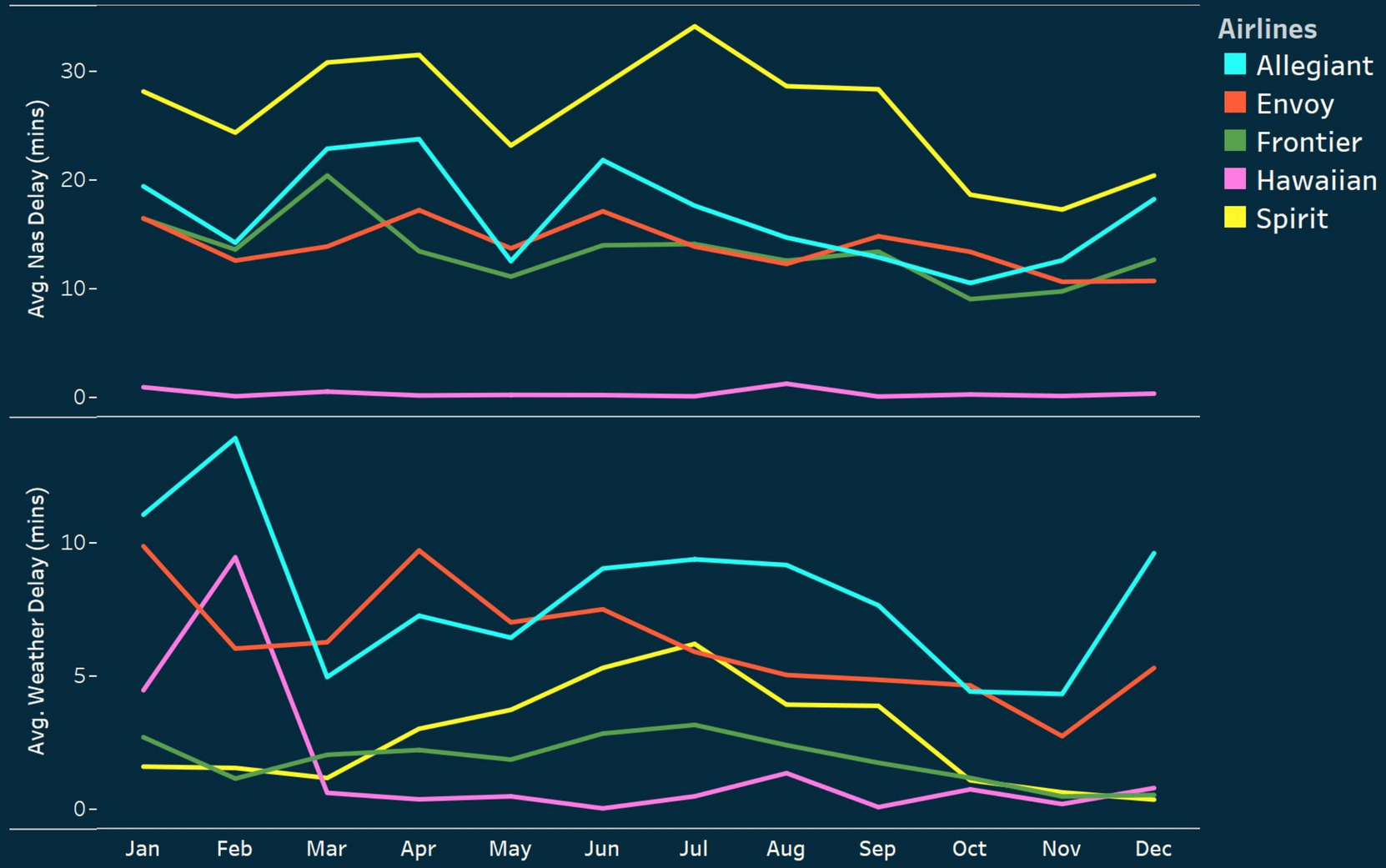
Frontier averages 4.45 hours, indicating slower operations.

Spirit averages 3.88 hours, suggesting quicker ground handling.

AeroVista

Navigating NAS and Weather Delays in Airline Operations

- **NAS Delays:** Beyond Our Control, Not Beyond Our Strategy.
- **Weather Delays:** Forecasting Challenges, Planning Opportunities.

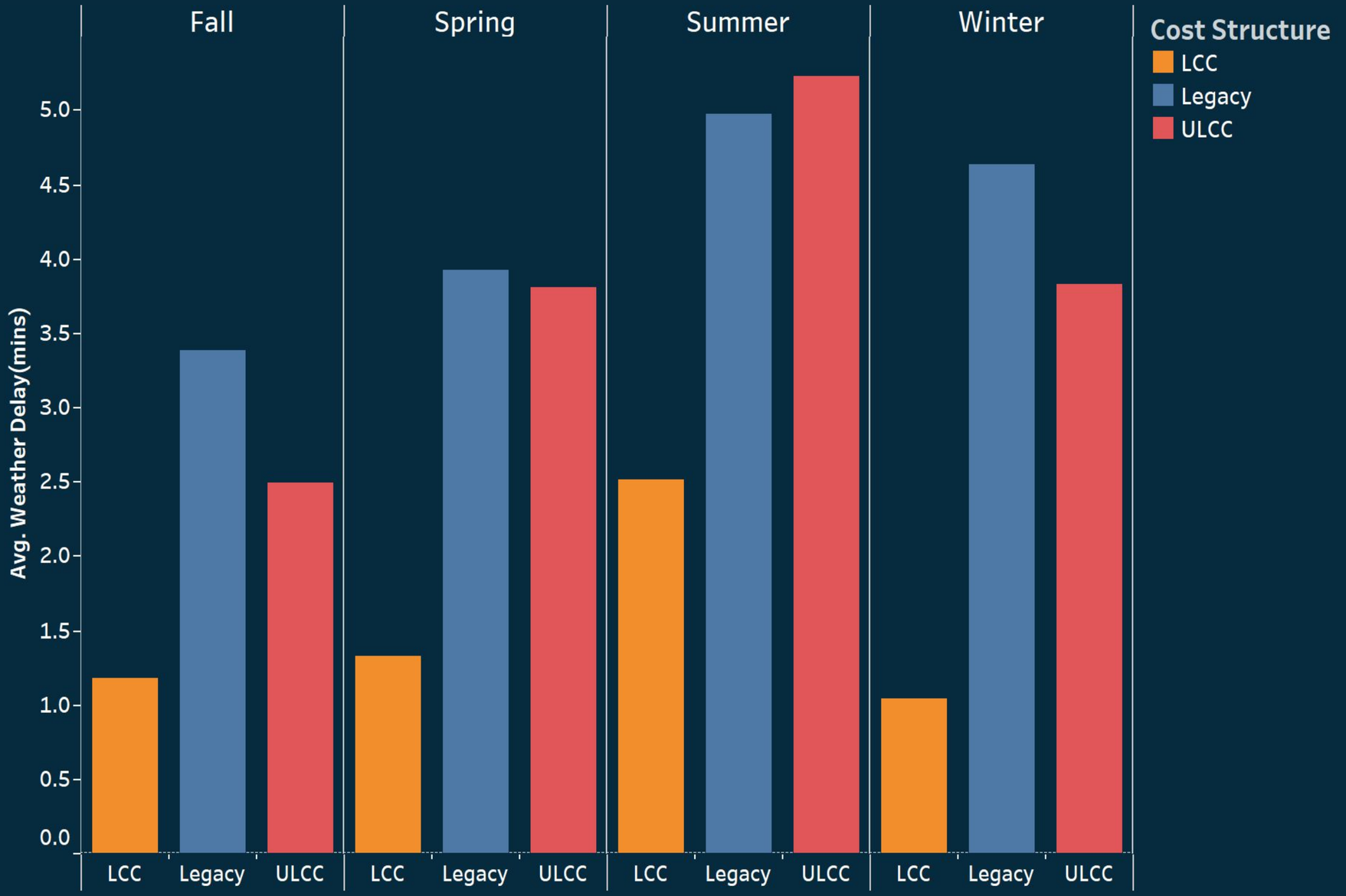


AeroVista

Impact of Weather & Seasonal Variations

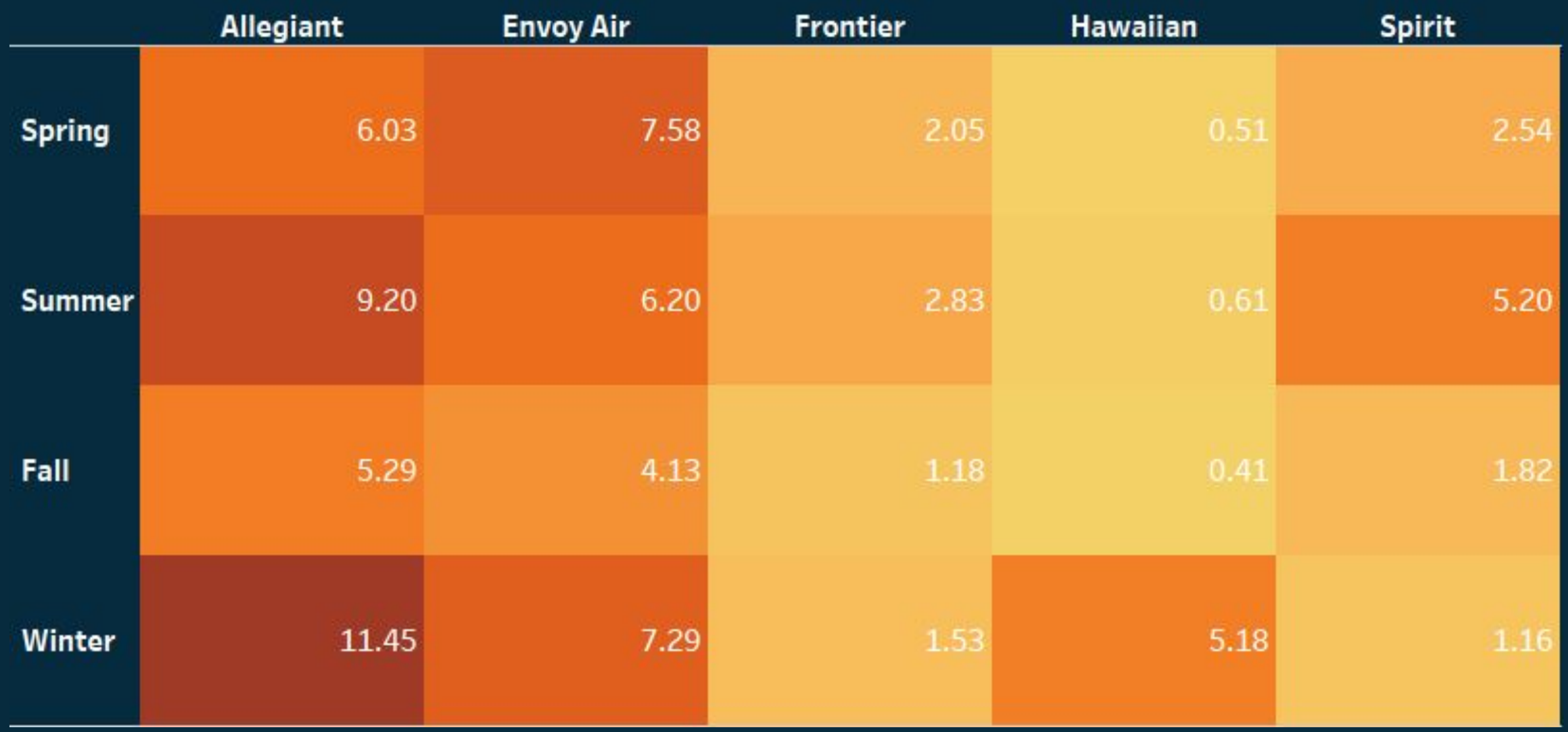
AeroVista

Seasonal Weather Delays by Airline Type



AeroVista

Seasonal Performance of ULCCs



	Allegiant	Envoy Air	Frontier	Hawaiian	Spirit
Spring	6.03	7.58	2.05	0.51	2.54
Summer	9.20	6.20	2.83	0.61	5.20
Fall	5.29	4.13	1.18	0.41	1.82
Winter	11.45	7.29	1.53	5.18	1.16

Allegiant faced highest weather delays amongst all carriers ULCC's

Envoy Air faced high weather delays and highest in Spring

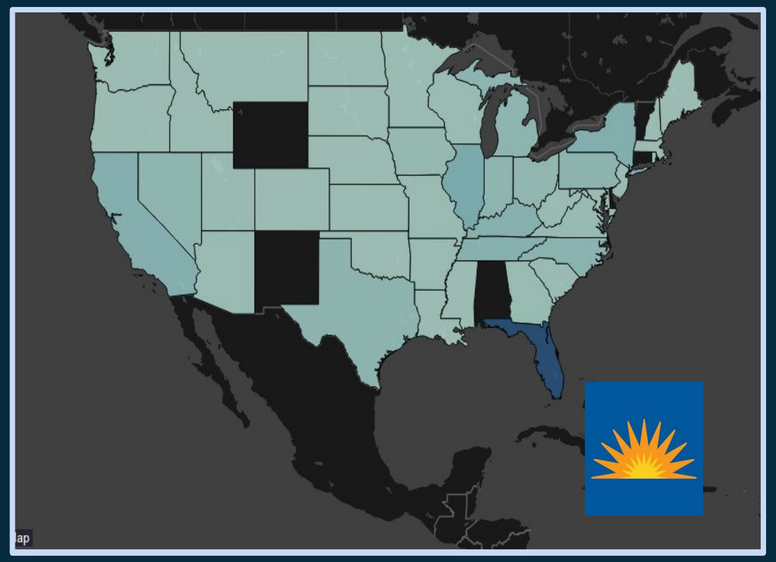
Frontier experienced consistently lower delays

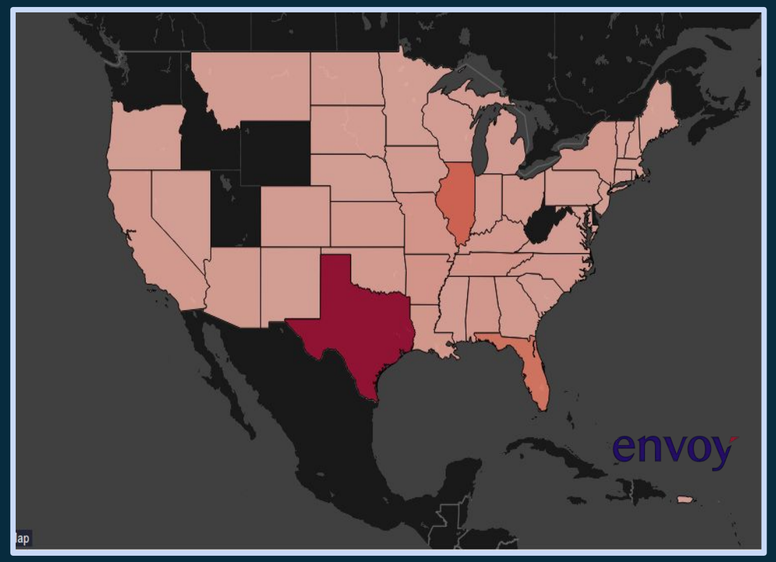
Spirit showed a spike in delays in Summer

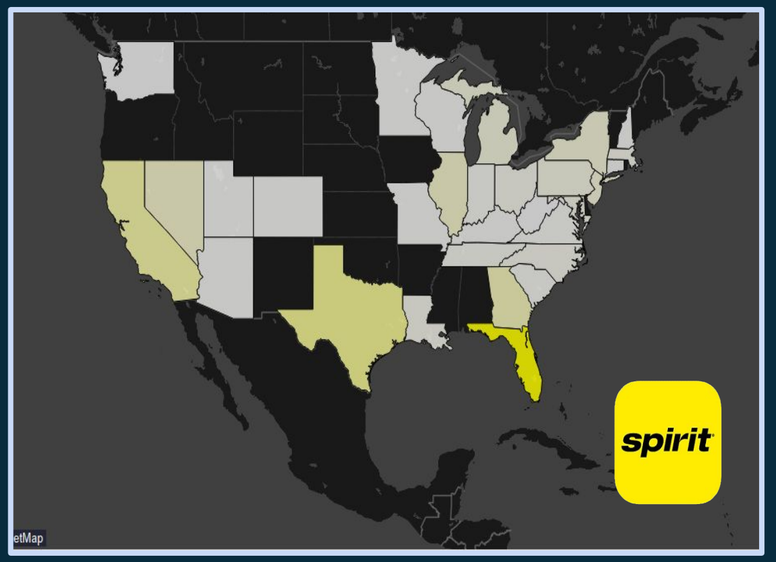
Hawaiian showed a rise in delays in Winters

AeroVista

Focusing on the regions through which the flight operates

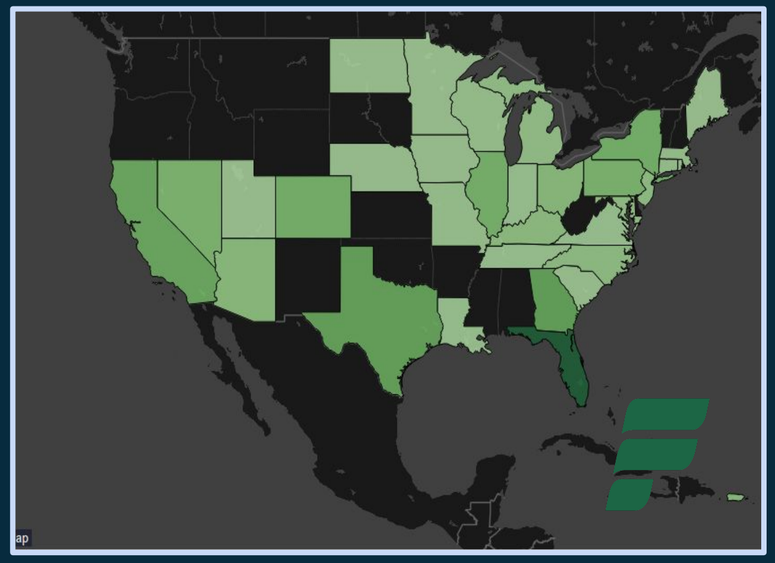


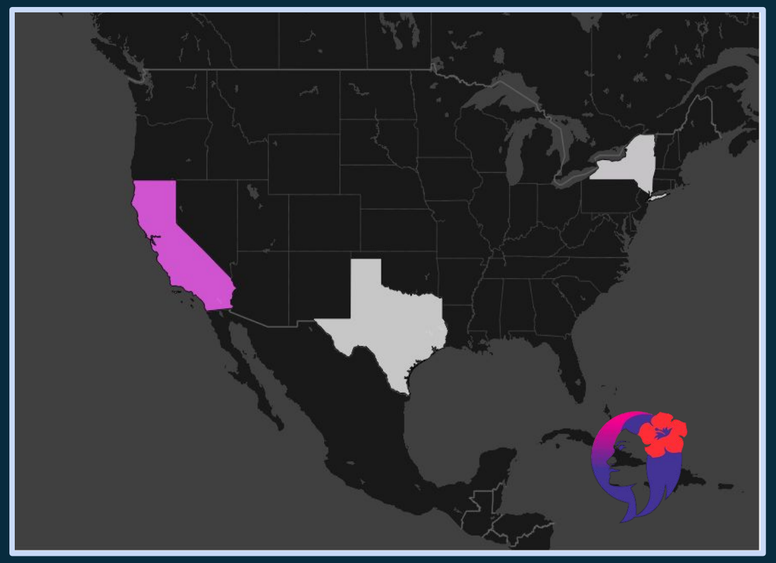




AeroVista

Focusing on the regions through which the flight operates





AeroVista

DATA

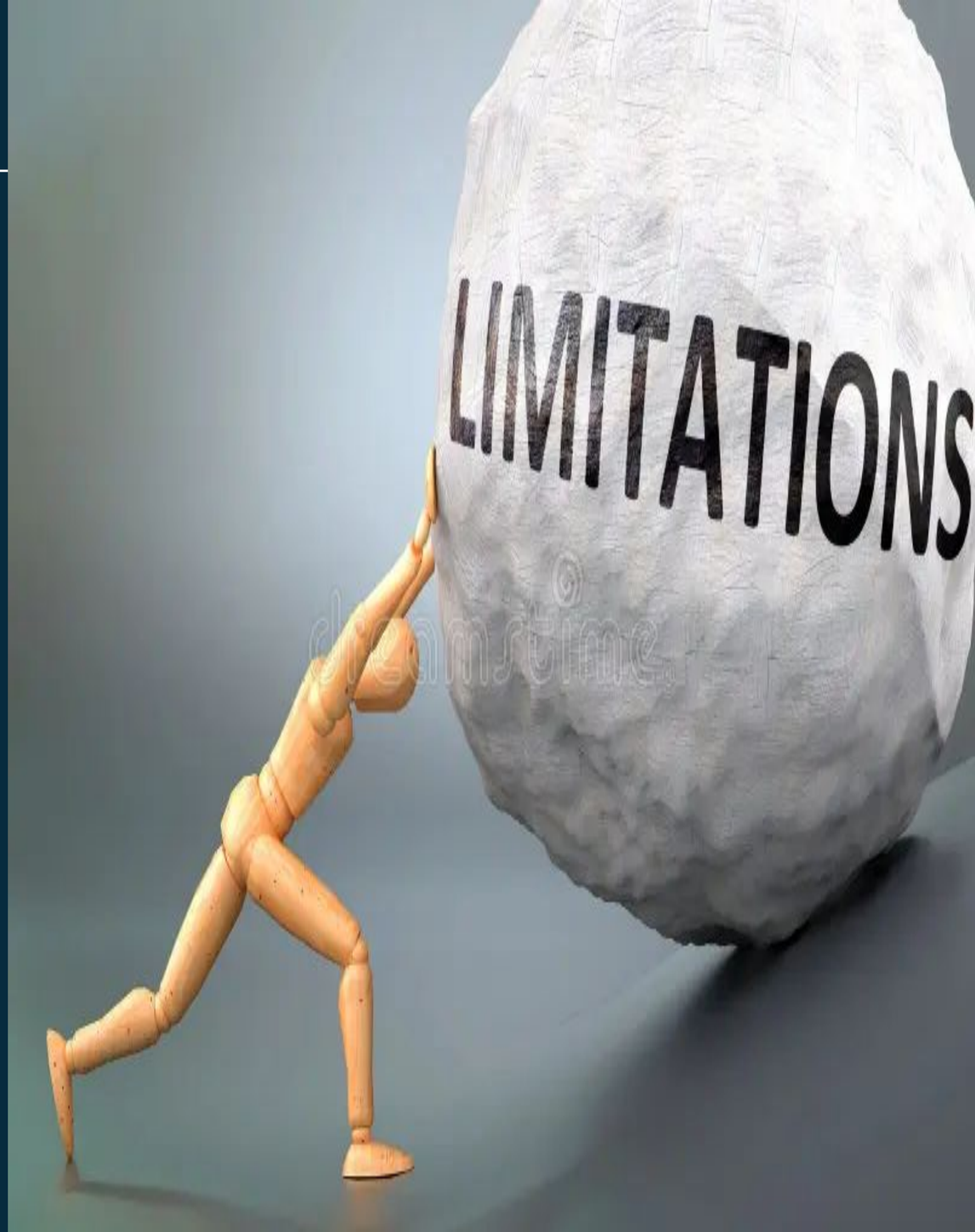
Limitations, Suggestions & Appendix

AeroVista

- **Limited Geographic Coverage**:
  - Data encompasses only six states within the US.
  - May not reflect national trends or regional variations effectively.

- **Granular Data Constraints**:
  - Includes only the duration of major delay causes (e.g., carrier, late aircraft).
  - Lacks detailed explanations for delays, hindering root cause analysis.

- **Historical Data Range**:
  - Data confined to the year 2023.
  - Limits ability to analyze long-term trends or the impact of past strategic changes.



AeroVista

Suggestions

- **Optimized Scheduling:** Design efficient schedules with minimal buffers to allow for quick turnarounds
- **Streamlined Operations:** Simplify services and processes to reduce gate and tarmac time.
- **Advanced Crew Management:** Employ flexible crew scheduling and reserve systems for rapid response to changes.
- **Maintenance Efficiency:** Implement proactive and predictive maintenance to avoid mechanical delays.
- **Leveraging Technology :** Utilize AI and real-time analytics to predict and manage delays effectively.
- **Partnerships and Coordination:** Foster strong partnerships for better coordination and priority services at airports.
- **Fleet Management:** Increasing the fleet size can help in better recovery from delays.
- **Customer Communication:** Keep passengers well-informed about delays to manage expectations and reduce operational disruptions.

AeroVista

Appendix

- Data Quality check to ensure all monthly data has the same features and schema

```
features = pd.read_csv('features.csv')
features = features['0'].tolist()

top_folder_path = "Data_aligned"

reference_columns = tuple(features)

# Empty list to store any inconsistencies
inconsistent_files = []

# Function to check if the columns of a file match the reference tuple
def check_columns(file_path):
    try:
        df = pd.read_csv(file_path, nrows=1)

        file_columns = tuple(df.columns)

        if file_columns != reference_columns:
            inconsistent_files.append((file_path, file_columns))
    except Exception as e:
        print(f"Error reading {file_path}: {e}")


for root, dirs, files in os.walk(top_folder_path):
    for filename in files:
        if filename.endswith(".csv"):
            file_path = os.path.join(root, filename)
            check_columns(file_path)

# Report the results
if inconsistent_files:
    print("\nInconsistent schemas detected")
else:
    print("\nAll files have the correct schema and are aligned with the reference columns.")
```

AeroVista

Appendix

- Aligning columns of all monthly data files before uploading to S3, so that there is no problem in schema definition

```
input_folder = 'Data'
output_folder = 'Data_aligned'

if not os.path.exists(output_folder):
    os.makedirs(output_folder)

master_columns = features

# Use os.walk to recursively go through directories
for root, dirs, files in os.walk(input_folder):
    for filename in files:
        if filename.endswith('.csv'):
            filepath = os.path.join(root, filename)

            df = pd.read_csv(filepath)

            # Add any missing columns with NaN values
            for col in master_columns:
                if col not in df.columns:
                    df[col] = pd.NA

            # Ensure columns are in the correct order
            df = df[master_columns]

            relative_path = os.path.relpath(root, input_folder)
            output_subfolder = os.path.join(output_folder, relative_path)
            if not os.path.exists(output_subfolder):
                os.makedirs(output_subfolder)

            output_filepath = os.path.join(output_subfolder, filename)
            df.to_csv(output_filepath, index=False)

            print(f"Processed and saved: {output_filepath}")
```

AeroVista

Appendix

AWS EMR

- Loading and merging monthly data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5872394 entries, 0 to 5872393
Data columns (total 81 columns):
 #   Column                  Dtype
---  ------                  -----
 0   YEAR                    int64
 1   FL_DATE                 object
 2   OP_UNIQUE_CARRIER       object
 3   OP_CARRIER_AIRLINE_ID   int64
 4   OP_CARRIER              object
 5   OP_CARRIER_FL_NUM       int64
 6   ORIGIN_AIRPORT_ID       int64
 7   ORIGIN_AIRPORT_SEQ_ID   int64
 8   ORIGIN                  object
 9   ORIGIN_CITY_NAME        object
 10  ORIGIN_STATE_FIPS       int64
 11  ORIGIN_STATE_NM         object
 12  DEST_AIRPORT_ID         int64
```

```python
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
import os

# Initialize the Spark session
spark = SparkSession.builder.master("local").appName("Airline Data Transformation").getOrCreate()

# Specify the S3 path to your data
base_path =  "s3://your-bucket-name/path-to-your-data/"


# Load all CSV files from each state folder
df_list = []
for state_folder in os.listdir(base_path):
    state_path = os.path.join(base_path, state_folder)
    if os.path.isdir(state_path):
        state_df = spark.read.option("header", "true").csv(os.path.join(state_path, "*.csv"))
        state_df = state_df.withColumn("STATE", F.lit(state_folder))  # Add state column for reference
        df_list.append(state_df)

# Merge all DataFrames into one
df = df_list[0]
for state_df in df_list[1:]:
    df = df.union(state_df)
```

- **Formatting and Type casting**

```
# Apply transformations

# 1. Convert to Boolean
boolean_columns = ['CANCELLED', 'DIVERTED', 'DIV_REACHED_DEST']
for col in boolean_columns:
    df = df.withColumn(col, F.when(df[col] == '1.0', True).otherwise(False))

# 2. Convert to Integer
int_columns = ['YEAR', 'DEP_DELAY', 'TAXI_OUT', 'TAXI_IN', 'ARR_DELAY', 'CRS_ELAPSED_TIME',
               'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'FLIGHTS', 'DISTANCE', 'CARRIER_DELAY',
               'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY',
               'TOTAL_ADD_GTIME', 'LONGEST_ADD_GTIME', 'DIV_ACTUAL_ELAPSED_TIME', 'DIV_ARR_DELAY',
               'DIV_DISTANCE', 'DIV1_TOTAL_GTIME', 'DIV1_LONGEST_GTIME', 'DIV2_TOTAL_GTIME',
               'DIV2_LONGEST_GTIME', 'DIV3_TOTAL_GTIME', 'DIV3_LONGEST_GTIME', 'DIV4_TOTAL_GTIME',
               'DIV4_LONGEST_GTIME', 'DIV5_TOTAL_GTIME', 'DIV5_LONGEST_GTIME']

for col in int_columns:
    df = df.withColumn(col, df[col].cast("int"))


# 3. Convert to hh:mm format
time_columns = ['CRS_DEP_TIME', 'DEP_TIME', 'WHEELS_OFF', 'WHEELS_ON', 'CRS_ARR_TIME',
                'ARR_TIME', 'FIRST_DEP_TIME', 'DIV1_WHEELS_ON', 'DIV1_WHEELS_OFF', 'DIV2_WHEELS_ON',
                'DIV2_WHEELS_OFF', 'DIV3_WHEELS_ON', 'DIV3_WHEELS_OFF', 'DIV4_WHEELS_ON',
                'DIV4_WHEELS_OFF', 'DIV5_WHEELS_ON', 'DIV5_WHEELS_OFF']


for col in time_columns:
    df = df.withColumn(col, F.lpad(df[col].cast("int").cast("string"), 4, '0'))  # Pad to 4 characters
    df = df.withColumn(col, F.date_format(F.to_timestamp(df[col], 'HHmm'), 'HH:mm'))  # Convert to HH:mm format
```

- **Imputing missing values**

```
from sklearn.impute import SimpleImputer
import pandas as pd

# List of delay columns where NaN should be replaced with 0
delay_columns = ['DEP_DELAY', 'ARR_DELAY', 'DISTANCE', 'CARRIER_DELAY',
                 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY',
                 'LATE_AIRCRAFT_DELAY', 'DIV_ARR_DELAY']

imputer = SimpleImputer(strategy='constant', fill_value=0)

# Apply the imputer to the delay columns
df_selected[delay_columns] = imputer.fit_transform(df_selected[delay_columns])
```

Data before cleaning

CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	WHEELS_ON	TAXI_IN	CRS_ARR_TIME	ARR_TIME	ARR_DELAY
900	856.0	-4.0	26.0	922.0	1019.0	4.0	1023	1023.0	0.0
1103	1101.0	-2.0	28.0	1129.0	1235.0	7.0	1244	1242.0	-2.0
1340	1333.0	-7.0	13.0	1346.0	1412.0	4.0	1432	1416.0	-16.0
1552	1549.0	-3.0	10.0	1559.0	1623.0	3.0	1652	1626.0	-26.0

Cleaned Data

CRS_DEP_TIME	DEP_TIME	DEP_DELAY	TAXI_OUT	WHEELS_OFF	WHEELS_ON	TAXI_IN	CRS_ARR_TIME	ARR_TIME	ARR_DELAY
06:00	05:50	-10.0	15.0	06:05	08:24	8.0	09:03	08:32	-31.0
21:27	21:23	-4.0	23.0	21:46	22:18	5.0	22:28	22:23	-5.0
15:17	15:13	-4.0	26.0	15:39	16:13	3.0	16:19	16:16	-3.0
17:04	16:59	-5.0	10.0	17:09	19:33	10.0	20:05	19:43	-22.0

Feature Creation

```python
# Define the function to categorize carriers
def categorize_carrier(carrier):
    if carrier in ['9E', 'AA', 'AS', 'DL', 'OO', 'UA', 'YX']:
        return 'Legacy'
    elif carrier in ['B6', 'OH', 'WN']:
        return 'LCC'
    else:
        return 'ULCC'

df['Carrier_Type_Cost'] = df['OP_UNIQUE_CARRIER'].apply(categorize_carrier)
```

```python
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Fall'

filtered_data['Season'] = filtered_data['MONTH'].apply(get_season)
```

```python
# List of holiday months
holiday_months = [12, 1, 11, 7]

filtered_data['IsHolidaySeason'] = np.where(filtered_data['MONTH'].isin(holiday_months), 1, 0)
```

```python
import numpy as np

df_selected['dep_delay_indicator'] = np.where(df_selected['DEP_DELAY'] > 15, True, False)
```

Feature Importance Analysis

```
# Get the feature importance from the classifier
feature_importances = rf_classifier.feature_importances_

importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)


print(importance_df.head(10))

plt.figure(figsize=(10,6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='skyblue')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.title('Feature Importance for DepDelay Class')
plt.gca().invert_yaxis()
plt.show()
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
import matplotlib.pyplot as plt

# Convert columns to categorical using one-hot encoding
data = pd.get_dummies(df_selected, columns=['MONTH', 'Season', 'IsHolidaySeason'],\
                      drop_first=False, prefix=['MONTH', 'Season', 'IsHolidaySeason'])

# Now, prepare the feature matrix and target variable
X = data[['CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY',\
          'DIV_ARR_DELAY'] + [col for col in data.columns if col.startswith('MONTH')]\
         + [col for col in data1.columns if col.startswith('Season')]]

y = data['dep_delay_indicator']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100,max_depth=10, random_state=42)
rf_classifier.fit(X_train, y_train)


y_pred = rf_classifier.predict(X_test)

# Evaluate the classifier
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```