

# *Airline Delay Prediction Under Extreme Weather Conditions*

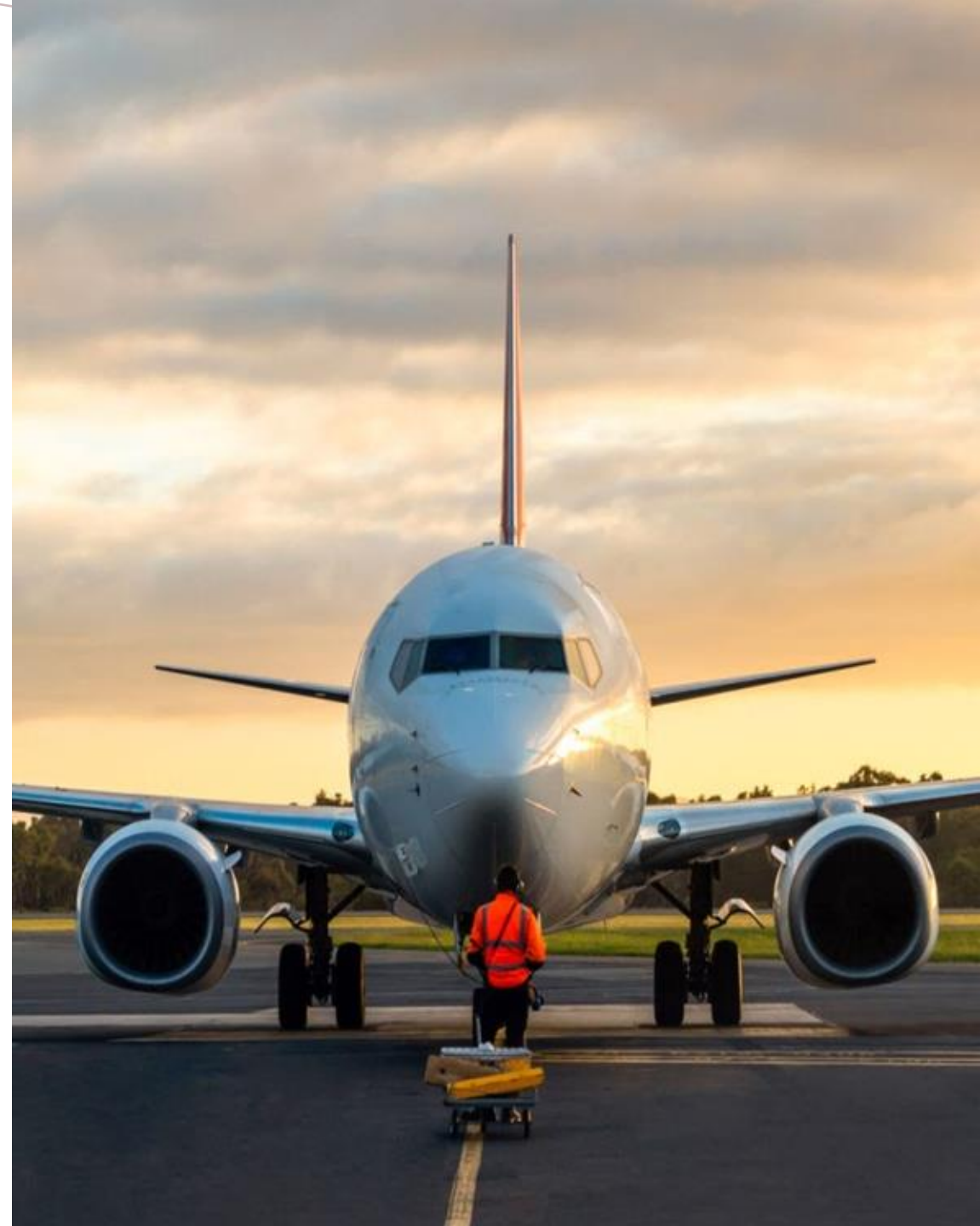
APAN5400

Term Project Proposal

Team 1: Ana Zegarra,

Arshia Mahant, Poorva Desai,

Samatha Krishna & Shirley Zhang



# *Background & Definition of the Business Use Case*

## ***Introduction:***

*Flight delays and cancellations caused by severe weather cost U.S. airlines millions of dollars annually, disrupt schedules, and frustrate passengers. This project aims to develop a predictive analytics system that assesses the impact of extreme weather( storms, hurricanes, heavy rain, snow) on flight punctuality across major U.S. airports.*

## ***Problem Statement:***

*Airlines and airports struggle to anticipate cascading delays from weather events. By integrating flight and meteorological data, our ETL pipeline will enable more accurate forecasting and resource optimization.*

## ***Objectives:***

- *Focused on Top 15 airports in the US, and analyzed 10 years worth of historical data.*
- *Quantified the correlation between weather severity and flight delay probabilities.*
- *Created a machine learning pipeline to predict delay likelihood under upcoming weather patterns.*
- *Provided insights for scheduling, passenger alerts, and operational planning.*

## ***Business Value & Impact:***

- *For Airlines: Predictive insights improve schedule resilience and fuel planning.*
- *For Airports: Enable proactive runway and gate allocation.*
- *For Passengers: Increased reliability and advanced delay notifications.*
- *For Regulators: Data driven understanding of how climate trends affect aviation efficiency*

# *Data Source Specification and Procurement*

Data was procured through a hybrid sourcing strategy utilizing both direct **public APIs** and **aggregated research repositories**. While NOAA's Climate Data Online and Hurricane Data APIs were accessed directly to supply granular hourly weather observations and historical storm tracks respectively, flight performance records were obtained via the Airline On-Time Statistics and Delay Causes dataset on Kaggle.

**\*\* The actual size of the full data is much larger than the one we used now since we only fetched data for 15 airports from 2015 to 2025**

## **NOAA Historical Weather Data**

### **Source:**

NOAA Climate Data Online (CDO) API

### **Format:**

JSON / CSV / API

### **Key Variables:**

Flightmonth, Station ID, timestamp, temperature, precipitation, wind, visibility, weather code

### **Description:**

Hourly weather observations for 15 mapped airport weather stations.

### **Data Size:**

10 MB

### **Last Update:**

Ongoing

## **NOAA Hurricane Data**

### **Source:**

NOAA Hurricane Data API

### **Format:**

JSON / API

### **Key Variables:**

Flightmonth, Storm ID, name, time, location, wind speed, pressure, category

### **Description:**

Historical tracks and intensity of tropical storms and hurricanes affecting U.S. airspace.

### **Data Size:**

55 MB

### **Last Update:**

Seasonal

## **Airline Delay Data**

### **Source:**

Airline On-Time Statistics and Delay Causes (Kaggle)

### **Format:**

CSV/API

### **Key Variables:**

Flightmonth, weather delay, carrier delay

### **Description:**

This raw dataset was pulled from the Airline On-Time Statistics and Delay Causes dashboard on the BTS website for all carriers and all airports.

### **Data Size:**

2 GB

### **Last Update:**

Ongoing

# Extract Transform Load Pipeline



## API Connection

- Created API connection for Airline delay dataset using Kaggle Hub
- Created API endpoint for Hurricane dataset
- Generated API connection for NOAA weather data set using NOAA token
- Used **Spark** to parallelize tens of thousands of Weather API calls

## MongoDB storage

- Connected all three datasets to MongoDB for flexible schema storage
- Weather data df: noaa\_weather\_raw
- Hurricane data df: hurricane\_ibtracs\_na
- Airline delay data df: airline\_delay\_cause

## MongoDB → PostgreSQL

- Cleaned and transformed three datasets
- Created a standard monthly date for all datasets for later merging
- Cleaned datasets are pushed to PostgreSQL, creating relational staging tables

## PostgreSQL → Tableau

- Conducted SQL query to merge the three staged datasets into one fact table using left join
- Connected PostgreSQL to Tableau for dashboard and visualization

## Tableau → Flask

- Embedded the Tableau Dashboards on a **Flask** interface



# Extract Transform Load (Code Display)



## 1. Created API connections using Spark

```
[8]: from dotenv import load_dotenv
import os

load_dotenv()
NOAA_TOKEN = os.getenv("NOAA_TOKEN")

if NOAA_TOKEN:
    print("NOAA token loaded successfully!")
else:
    print("NOAA token not found.")

NOAA token loaded successfully!

[24]: airport_to_station = {
    "ATL": "GHND:USW00013874",
    "DFW": "GHND:USW00003927",
    "DEN": "GHND:USW00003817",
    "ORD": "GHND:USW00094846",
    "LAX": "GHND:USW00093134",
    "CLT": "GHND:USW00013881",
    "LAS": "GHND:USW00023169",
    "MCO": "GHND:USW00012815",
    "PHX": "GHND:USW00023183",
    "MIA": "GHND:USW00012839",
    "SEA": "GHND:USW00042333",
    "EWR": "GHND:USW00014734",
    "JFK": "GHND:USW00094789",
    "SFO": "GHND:USW00023234",
    "BOS": "GHND:USW00014739"
}
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("FAA NOAA Integration") \
    .master("local[*]") \
    .config("spark.driver.memory", "2G") \
    .config("spark.executor.memory", "2G") \
    .config("spark.driver.extraJavaOptions", "-Duser.timezone=UTC") \
    .config("spark.executor.extraJavaOptions", "-Duser.timezone=UTC") \
    .getOrCreate()

spark

2025-01-01 10:00:00
Spark version: 3.4.1, 2025-01-01 10:00:00
OpenJDK Runtime Environment: 2025-01-01 10:00:00 (build 17.0.14_10-b10)
OpenJDK 64-Bit Server VM: 2025-01-01 10:00:00 (build 17.0.14_10-b10, mixed mode, sharing)
WARNING: Using incubator modules: jdk.incubator.vector
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
2025-01-01 10:00:00 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

SparkSession - in-memory

SparkContext

Spark UI

Version: v4.0.1
Master: local[4]
AppName: FAA_NOAA_Integration
```

## 2. Connected all three datasets to MongoDB

```
from pymongo import MongoClient

MONGO_URI = "mongodb://localhost:27017"
client = MongoClient(MONGO_URI)

db = client["flight_weather"]
weather_coll = db["noaa_weather_raw"]

weather_docs = df_weather_all.to_dict("records")
result = weather_coll.insert_many(weather_docs)
print(f"✅ Inserted {len(result.inserted_ids)} documents into 'noaa_weather_raw'")

✅ Inserted 54601 documents into 'noaa_weather_raw'
```

## 3. Data cleansing process in MongoDB

```
[13]: import pandas as pd
import numpy as np

# 1) make sure year & month are numeric
delay_df["year"] = pd.to_numeric(delay_df["year"], errors="coerce")
delay_df["month"] = pd.to_numeric(delay_df["month"], errors="coerce")

# drop rows where year or month is missing
delay_df = delay_df.dropna(subset=["year", "month"])

delay_df["year"] = delay_df["year"].astype(int)
delay_df["month"] = delay_df["month"].astype(int)

# 2) create a date for the first day of each month
delay_df["flight_month"] = pd.to_datetime(
    delay_df["year"].astype(str) + "-" +
    delay_df["month"].astype(str) + "-01",
    errors="coerce"
)

# 3) keep only your 15 airports
target_airports = [
    'ATL', 'DFW', 'DEN', 'ORD', 'LAX',
    'CLT', 'LAS', 'MCO', 'PHX', 'MIA',
    'SEA', 'EWR', 'JFK', 'SFO', 'BOS'
]

delay_df = delay_df[delay_df["airport"].isin(target_airports)]
```

## 4. Pushed cleaned datasets to PostgreSQL

```
[8]: from sqlalchemy import create_engine

PG_USER = "postgres" # or whatever user you use in pgAdmin
PG_PASSWORD = "123" # <<< your REAL password, no quotes inside
PG_HOST = "localhost"
PG_PORT = "5432"
PG_DB = "flight_project"

engine = create_engine(
    f"postgresql+psycopg2://{PG_USER}:{PG_PASSWORD}@{PG_HOST}:{PG_PORT}/{PG_DB}"
)

delay_df.to_sql("stg_airline_delay", engine, if_exists="replace", index=False)
```

## 5. Merged datasets using left join

(For both the weather and airline delay datasets, we have the airport name and the flight month where we used left join to merge the datasets together)

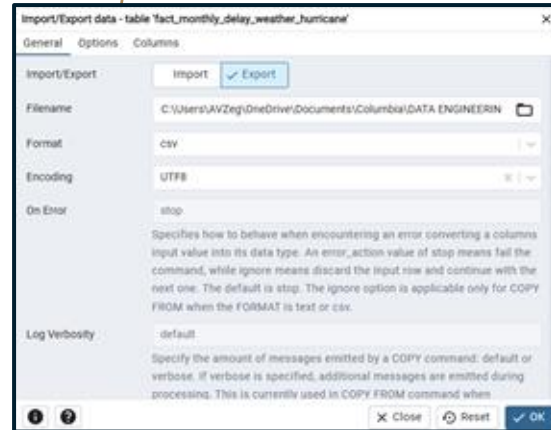
	airport text	flight_month timestamp without time zone
1	EWR	2025-01-01 00:00:00
2	LAS	2025-01-01 00:00:00
3	LAX	2025-01-01 00:00:00

```
FROM stg_airline_delay d
LEFT JOIN stg_weather w
ON d.airport = w.airport
AND d.flight_month = w.flight_month
LEFT JOIN stg_hurricane h
ON d.flight_month = h.flight_month;
```

# Extract Transform Load

## (Code Display)

## 6. Exported data into CSV



## 7. Loaded data into Python

```
# =====
# STEP 1 - Imports
# =====

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score, RocCurveDisplay
from sklearn.impute import SimpleImputer # <--- NEW

# =====
# STEP 2 - Load data
# =====

df = pd.read_csv("fact_monthly_delay_weather_hurricane.csv")

print("Shape:", df.shape)
display(df.head())
print(df.columns.tolist())

# If you have 'flight_month' as text and want it as date:
if "flight_month" in df.columns:
    df["flight_month"] = pd.to_datetime(df["flight_month"])
```

## 8. Python- Correlation between Weather & Flight delay

```
# STEP 4 - Pick weather columns & correlation
#
candidate_weather_cols = [c for c in df.columns
                           if any(k in c.lower() for k in
                                   ["temp", "humid", "precip", "wind", "storm"])]
print("Guessed weather columns:", candidate_weather_cols)

# EDIT these names if needed based on the printout above
weather_cols = [c for c in [
    "avg_temp", "avg_humidity", "precip_in", "wind_kt", "n_storms"
] if c in df.columns]

print("Using weather columns:", weather_cols)

corr_cols = weather_cols + ["delay_rate"]
corr_df = df[corr_cols].dropna()

plt.figure(figsize=(8, 6))
sns.heatmap(corr_df.corr(), annot=True, fmt=".2f", square=True)
plt.title("Correlation: Weather vs Delay Rate")
plt.tight_layout()
plt.show()
```

## 9. Python- Predicted Delays for 2025-2031

```
[10]: RandomForestRegressor
RandomForestRegressor(estimator=100, random_state=42)

[11]: # Define airports and months from your data
airports = df['airport'].unique()
months = range(1, 13)
years_future = range(2003, 2005) # 2003-2005 inclusive

future_rows = []
for airport in airports:
    for year in years_future:
        for month in months:
            future_rows.append([airport, airport, 'year', year, 'month', month])

future_df = pd.DataFrame(future_rows)

[12]: # Compute historical average weather by airport + month
weather_means = [
    df_w[
        groupby(["airport", "month"])[feature_cols]
        .mean()
        .reset_index()
    ]

# Join averages into the future calendar
future_df = future_df.merge(
    weather_means,
    on=["airport", "month"],
    how="left"
)

# If data_category was labeled earlier, it's already numeric in weather_means

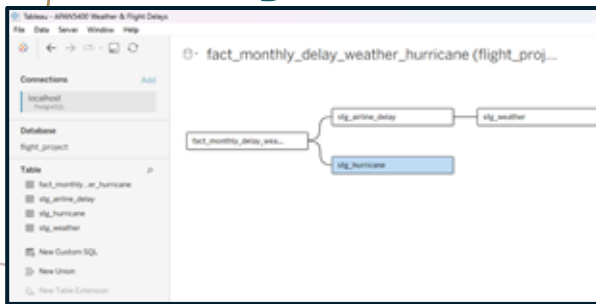
[13]: future_df = future_df.dropna(subset=feature_cols)

[14]: # Predict with the trained model
future_df['predicted_delay_rate'] = model.predict(future_df.feature_cols)
```

**10. Import Dataset into Tableau (From PostgreSQL)**

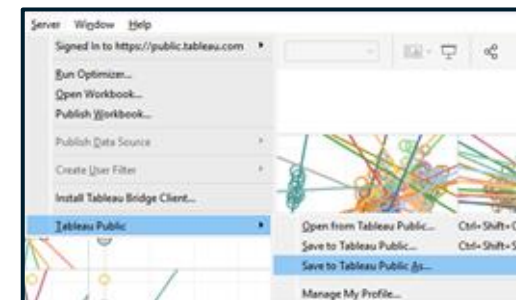


**11. Import Dataset into Tableau (Predictive Analysis completed in Python)**



The screenshot shows the Google Cloud Data Studio interface. At the top, the dataset name is 'delay\_predictions\_2025\_2035\_expanded\_trend (2)'. Below the name, there's a 'Table Editor' tab. The table has columns: 'Year', 'Month', and 'Avg Time'. The data is organized into rows for each year and month combination. The 'Avg Time' column shows values that are mostly 0.000000, with some non-zero values in the 'Year' column (e.g., 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035). The interface also includes a sidebar with navigation options like 'Data Explorer', 'Table Editor', and 'Table Schema'.

## 12. Published Dashboards into Tableau Public



### 13. Flask interface with tableau dashboard embedded

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def dashboard():
    return '''
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Flight Delay and Cancellation Dashboard</title>
<style>
    body {
        font-family: Arial, sans-serif;
        background-color: #f5f7fa;
        margin: 0;
        padding: 20px;
    }
    h1 {
        text-align: center;
        color: #333;
    }
    .description {
        max-width: 1100px;
        margin: 30px auto 0 auto;
        background: #fff;
    }
'''
```

# Nationwide Scalability Plan

Scaling from a pilot of **15 airports** to **all U.S. airports** across 10 years. Includes: flight operations, weather, enriched delay features, and metadata.

## Data Pipeline & Processing (Spark / EMR)

- Run large-scale joins across airports, weather, and historical delays.
- Use **Spark on AWS EMR** for:
  - feature engineering
  - monthly model retraining
  - building Tableau-ready aggregates
- Medium EMR cluster (2–4 nodes) triggered monthly or weekly.

## Storage & Databases

- **Amazon S3:** Central data lake for raw, cleaned, and versioned data (~30 GB).
- **PostgreSQL (RDS):**
  - Stores analytical tables
  - Powers Tableau dashboards
  - Handles structured, relational metrics
- **MongoDB Atlas:**
  - Stores semi-structured weather snapshots
  - Airport-level metadata
  - JSON-based feeds used by Flask API

## Visualization & Application Layer

- **Tableau Cloud:** Airport-level dashboards, delay patterns, KPIs, and geographic comparisons.
- **Flask App (EC2/Fargate):**
  - Provides prediction APIs
  - Serves model outputs to dashboards
  - Supports multi-user access



# Cost Considerations

## Core Components & Costs

### 1. Spark (AWS EMR) → \$350-\$500/month

- Medium cluster (2-4 nodes)
- Monthly processing + model updates

### 2. PostgreSQL (RDS) → \$150-\$220/month

- db.m5.large or similar
- 60-100 GB storage

### 3. MongoDB Atlas → \$70-\$120/month

- M20 / M30 tier (no sharding required)

### 4. Tableau Cloud → \$250-\$350/month

- 5 Creator licenses + viewer seats + scheduled refreshes

### 5. Flask Hosting + Monitoring → \$40-\$70/month

- EC2 or Fargate
- CloudWatch logs

### 6. S3 Storage (30 GB Data Lake) → \$10-\$15/month

**Total Estimated Monthly Cost ≈ \$870 - \$1,275 per month**

## Future Possibilities for Exploration

Add real-time streaming (FAA + NOAA) and multi-region dashboards → **~20-30% cost increase.**

Dataset size: **~30 GB**  
Refresh frequency: **monthly or biweekly** Spark processing  
Users: **Small data team + dashboard viewers**

# Data Governance & Compliance

Data	Data Type/Compliance	Security	Auditability
NOAA Historical weather data	Public and read only. Follows guidelines implemented by NOAA.	Encrypted, final datasets are open	Secure HTTPS API requests, traceable from source to publication
NOAA Hurricane data	Public and read only. Follows guidelines implemented by NOAA.	Encrypted, SOC monitoring	Secure HTTPS API requests
Airline On-Time Statistics and Delay Causes	Public, Kaggle Data	Open data, publicly downloadable	Open data with traceable ETL and loading steps

## *Team Roles*

Name	Responsibilities
Ana Zegarra	Weather delay correlation modeling, Machine learning to predict delay likelihood under upcoming weather patterns, Tableau Dashboard with historical data.
Arshia Mahant	Nationwide scalability plan, cost considerations, data governance and compliance, Code and process documentation
Poorva Desai	Set initial structure for API connection. Tableau Dashboard with predictive data, utilized random forest modeling to predict future yearly delay rates by Airport.
Samatha Krishna	Created the Flask interface with description and how to interpret instructions
Shirley Zhang	Datasets API connections, Data cleansing and loading in MongoDB and PostgreSQL

# Demo: Visualization placeholder

## Historical Dashboard



[APAN5400 Weather & Flight Delays | Tableau Public](#)

## Predictive Dashboard



[Predictive Dashboard | Tableau Public](#)

## Flask Interface

