# Flight Delay Prediction Pipeline V2 - Project Document

## Project Title

Flight Delay Prediction Pipeline with Real-Time Streaming and SQLite Integration (Version 2)

## Project Scope

This project develops a real-time flight delay prediction system using PySpark, Kafka, and SQLite on a Windows 11 environment. It generates synthetic flight data, trains an offline ML model, streams data via Kafka, applies the model for predictions, visualizes results with Plotly, and stores results in a SQLite database. Version 2 enhances the pipeline with improved visualizations (temperature-based sizing, airport hover data), model accuracy tracking, delayed flight alerts, and robust error handling.

### Learning Outcomes

* \*\*Data Engineering\*\*: Master streaming pipelines with Kafka and PySpark.
* \*\*Machine Learning\*\*: Train and apply ML models for real-time predictions.
* \*\*Data Generation\*\*: Create synthetic datasets for testing.
* \*\*Visualization\*\*: Build interactive Plotly plots for insights.
* \*\*Database Management\*\*: Integrate SQLite for persistent storage.
* \*\*Troubleshooting\*\*: Resolve schema, streaming, and environment issues.
* \*\*Project Management\*\*: Plan, execute, and document a data science project.

## Index

1. Project Title

2. Project Scope

3. Planning

* Objectives
* Resources Required
* Key Stakeholders

4. Execution

* Startup Steps
* Progress Tracking
* Timeline
* Deviations from the Plan

5. Code Explanations

6. Troubleshooting Steps

7. Desired Outputs

8. Dependencies and Installations

9. Appendices

* Appendix 1: Dependency Setup
* Appendix 2: Cleanup Steps
* Appendix 3: Performance Enhancements

## Planning

### Objectives

* Generate synthetic flight data for testing and ML training.
* Train an offline ML model for delay predictions using `train\_flight\_model.ipynb`.
* Build a Kafka-based streaming pipeline.
* Predict flight delays using a PySpark model.
* Store predictions in SQLite (`D:\flight\_db\flight\_predictions.db`).
* Visualize delays with Plotly (temperature size, airport hover).
* Compute model accuracy and delay statistics.
* Implement alerts for delayed flights.
* Document setup, execution, and troubleshooting.

### Resources Required

* \*\*Hardware\*\*: Windows 11 PC, 16GB RAM, 500GB SSD.
* \*\*Software\*\*:
* Python 3.9.13 (`python39venv`).
* PySpark 3.5.5, Spark 3.5.5.
* Kafka 2.12-3.7.0.
* SQLite (built-in), SQLite Browser.
* Jupyter Notebook.
* \*\*Libraries\*\*:
* `pandas`
* `numpy`
* `plotly`
* `kafka-python`
* `jupyter`
* \*\*Scripts\*\*:
* `kafka\_control.py`: Manages Kafka/Zookeeper.
* `generate\_flight\_data.py`: Produces synthetic flight data.
* `train\_flight\_model.ipynb`: Trains ML model.
* `flight\_pipeline.ipynb`: Main pipeline.
* \*\*Files\*\*:
* `D:/flight\_delay\_model`: Pre-trained ML model.
* `D:\flight\_db\flight\_predictions.db`: SQLite database.
* `D:\flight\_data\_sample.json`: Sample data for ML training.
* \*\*Storage\*\*: `D:\`, `D:\kafka\_2.12-3.7.0`, `D:\spark-3.5.5-bin-hadoop3`.

### Key Stakeholders

* \*\*Developer\*\*: Builds scripts, trains model, tests pipeline, drives upskilling.
* \*\*Mentor\*\*: Guides coding, debugging, and documentation.
* \*\*End User\*\*: Analyst using predictions and visuals.

## Execution

### Startup Steps

1. \*\*Activate Virtual Environment\*\*:

D:\>D:\python39venv\Scripts\activate  
 (python39venv) D:\>

2. \*\*Start Kafka\*\*:

(python39venv) D:\>python kafka\_control.py

* Opens Zookeeper (`2181`) and Kafka (`9092`) cmd windows.

3. \*\*Generate Synthetic Data\*\*:

(python39venv) D:\>python generate\_flight\_data.py

* Generates `D:/flight\_data\_sample.json` (1000 records).
* Stop with `Ctrl+C` after seeing `Sent: FL100` to proceed.

4. \*\*Train ML Model\*\*:

* Start Jupyter:

(python39venv) D:\>jupyter notebook

* Open `D:\train\_flight\_model.ipynb` and run all cells.
* Creates `D:/flight\_delay\_model`.

5. \*\*Verify Scripts\*\*:

* Ensure `kafka\_control.py`, `generate\_flight\_data.py`, `train\_flight\_model.ipynb` in `D:\`.
* Check `flight\_pipeline.ipynb` in Jupyter.

6. \*\*Run Pipeline\*\*:

* In Jupyter, open `flight\_pipeline.ipynb`.
* Run cells: table creation, streaming, stats, accuracy.

### Progress Tracking

* \*\*Daily Logs\*\*: Record hours, cells run, errors (e.g., `no column named temperature`).
* \*\*Metrics\*\*:
* Batches processed (`Batch X: Saved Y rows`).
* Model accuracy (`X% correct`).
* SQLite row count via CLI/Browser.
* \*\*Reporting\*\*:
* Jupyter console: Batch outputs, alerts.
* SQLite Browser: Table data.
* Plotly: Visual updates every 10 seconds.
* \*\*Checkpoints\*\*:
* Weekly: Pipeline stability, data volume.
* Milestones: Model training, data generation, SQLite writes, visualizations.

### Timeline

* \*\*Week 1 (Setup, V1)\*\*:
* Day 1: Install dependencies, write `kafka\_control.py`.
* Day 2: Build `generate\_flight\_data.py`, test Kafka.
* Day 3-4: Run `train\_flight\_model.ipynb`, develop V1 pipeline, model predictions.
* Day 5: Plotly visuals, console output.
* \*\*Week 2 (V2 Enhancements)\*\*:
* Day 8: SQLite integration, fix schema errors.
* Day 9: Enhanced visuals, alerts, accuracy.
* Day 10: Document V2, test pipeline.
* \*\*Week 3 (Future)\*\*:
* Add features (e.g., `wind\_speed` in model).
* Optimize performance.
* Finalize documentation.

### Deviations from the Plan

* \*\*Schema Mismatches\*\*:
* Issue: `predictions` table lacked `temperature`, `departure\_airport`.
* Fix: Recreated table with all columns.
* Reason: Early test scripts used partial schema.
* \*\*Empty SQLite Database\*\*:
* Issue: No data saved due to write failures.
* Fix: Debugged `write\_to\_sqlite`, ensured Kafka data flow.
* Reason: Producer timing, schema errors.
* \*\*Windows Setup\*\*:
* Issue: Hadoop/Spark conflicts in early setup.
* Fix: Configured `HADOOP\_HOME`, `RawLocalFileSystem`.
* Reason: Windows-specific DLL issues.

## Code Explanations

### `kafka\_control.py`

import subprocess  
import time  
import sys  
  
def start\_kafka():  
 # Clear old processes  
 subprocess.run("taskkill /F /IM java.exe /T", shell=True, capture\_output=True)  
 time.sleep(2)  
 # Start Zookeeper  
 zookeeper\_cmd = "D:\\kafka\_2.12-3.7.0\\bin\\windows\\zookeeper-server-start.bat D:\\kafka\_2.12-3.7.0\\config\\zookeeper.properties"  
 subprocess.Popen(["cmd", "/c", "start", "cmd", "/k", zookeeper\_cmd], shell=True)  
 time.sleep(10) # Wait longer for Zookeeper  
 # Start Kafka  
 kafka\_cmd = "D:\\kafka\_2.12-3.7.0\\bin\\windows\\kafka-server-start.bat D:\\kafka\_2.12-3.7.0\\config\\server.properties"  
 subprocess.Popen(["cmd", "/c", "start", "cmd", "/k", kafka\_cmd], shell=True)  
 time.sleep(5)  
 print("Kafka started")  
  
def stop\_kafka():  
 subprocess.run("taskkill /F /IM java.exe /T", shell=True, capture\_output=True)  
 print("Kafka and Zookeeper stopped")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) > 1 and sys.argv[1] == "stop":  
 stop\_kafka()  
 else:  
 start\_kafka()

* \*\*Purpose\*\*: Manages Kafka and Zookeeper servers by starting or stopping them via Windows batch files, enabling data streaming for the pipeline.
* \*\*Key Lines\*\*:
* `subprocess.run("taskkill /F /IM java.exe /T", ...)`: Terminates existing Java processes to avoid conflicts.
* `subprocess.Popen(["cmd", "/c", "start", ...])`: Launches Zookeeper (`2181`) and Kafka (`9092`) in separate cmd windows.
* `time.sleep(10)`: Ensures Zookeeper is fully initialized before Kafka starts.

### `generate\_flight\_data.py`

import numpy as np  
import pandas as pd  
from datetime import datetime, timedelta  
import json  
import time  
from kafka import KafkaProducer  
  
# Global counter for flight numbers  
flight\_counter = 100  
  
def generate\_flight\_batch(size=1000, start\_index=None):  
 global flight\_counter  
 airports = ["AYD", "VNS", "IXD", "HYD"]  
 now = datetime.now()  
 # Use provided start\_index or global counter  
 start = start\_index if start\_index is not None else flight\_counter  
 flights = {  
 "flight\_number": [f"FL{i:03d}" for i in range(start, start + size)],  
 "departure\_airport": np.random.choice(airports, size),  
 "arrival\_airport": np.random.choice(airports, size),  
 "scheduled\_departure": [(now + timedelta(minutes=int(m))).strftime("%Y-%m-%d %H:%M:%S")  
 for m in np.random.randint(0, 120, size)],  
 "temperature": np.random.uniform(0, 40, size),  
 "wind\_speed": np.random.uniform(0, 20, size),  
 "precipitation": np.random.uniform(0, 10, size),  
 "delay\_minutes": np.where(np.random.random(size) > 0.7, np.random.randint(0, 60, size), 0)  
 }  
 df = pd.DataFrame(flights)  
 # Ensure arrival != departure  
 df["arrival\_airport"] = df.apply(lambda row: np.random.choice([a for a in airports if a != row["departure\_airport"]]), axis=1)  
 # Update counter  
 if start\_index is None:  
 flight\_counter = start + size  
 return df  
  
# Save sample for ML (use fixed range)  
sample\_df = generate\_flight\_batch(1000, start\_index=100)  
sample\_df.to\_json("D:/flight\_data\_sample.json", orient="records", lines=True)  
  
# Real-time producer  
producer = KafkaProducer(bootstrap\_servers="localhost:9092", value\_serializer=lambda v: json.dumps(v).encode("utf-8"))  
while True:  
 flight = generate\_flight\_batch(1).iloc[0].to\_dict()  
 producer.send("flight-data", flight)  
 print(f"Sent: {flight['flight\_number']}")  
 time.sleep(1)

* \*\*Purpose\*\*: Generates 1000 synthetic flight records for ML training, saves them to `D:/flight\_data\_sample.json`, and streams real-time data to Kafka’s `flight-data` topic.
* \*\*Key Lines\*\*:
* `generate\_flight\_batch(size=1000, ...)`: Creates data with `flight\_number`, `temperature`, `delay\_minutes`, etc.
* `sample\_df.to\_json("D:/flight\_data\_sample.json", ...)`: Saves 1000 records for ML training.
* `producer.send("flight-data", flight)`: Streams one flight at a time to Kafka.

### `train\_flight\_model.ipynb`

import os  
os.environ["PYSPARK\_PYTHON"] = "D:\\python39venv\\Scripts\\python.exe"  
os.environ["PYSPARK\_DRIVER\_PYTHON"] = "D:\\python39venv\\Scripts\\python.exe"  
os.environ["HADOOP\_HOME"] = "D:\\hadoop-3.3.6"  
  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, when  
from pyspark.ml.feature import VectorAssembler  
from pyspark.ml.classification import RandomForestClassifier  
from pyspark.ml import Pipeline  
import shutil  
  
# Initialize SparkSession  
spark = SparkSession.builder \  
 .appName("FlightDelayModelTraining") \  
 .master("local[\*]") \  
 .config("spark.hadoop.fs.defaultFS", "file:///") \  
 .config("spark.hadoop.fs.file.impl", "org.apache.hadoop.fs.RawLocalFileSystem") \  
 .config("spark.driver.memory", "4g") \  
 .config("spark.executor.memory", "4g") \  
 .getOrCreate()  
  
print("Spark version:", spark.version)  
print("Hadoop home:", os.environ.get("HADOOP\_HOME"))  
  
# Load synthetic flight data  
try:  
 df = spark.read.json("D:/flight\_data\_sample.json")  
 print("Data loaded successfully, rows:", df.count())  
except Exception as e:  
 print("Error loading data:", str(e))  
 spark.stop()  
 raise e  
  
# Preprocess data  
df = df.withColumn("is\_delayed", when(col("delay\_minutes") > 15, 1).otherwise(0))  
  
# Define features  
feature\_cols = ["temperature", "wind\_speed", "precipitation", "delay\_minutes"]  
assembler = VectorAssembler(inputCols=feature\_cols, outputCol="features")  
  
# Define classifier  
rf = RandomForestClassifier(  
 labelCol="is\_delayed",  
 featuresCol="features",  
 numTrees=100,  
 maxDepth=10,  
 seed=42  
)  
  
# Create pipeline  
pipeline = Pipeline(stages=[assembler, rf])  
  
# Split data  
train\_df, test\_df = df.randomSplit([0.8, 0.2], seed=42)  
  
# Train model  
try:  
 model = pipeline.fit(train\_df)  
 print("Model trained successfully")  
except Exception as e:  
 print("Error training model:", str(e))  
 spark.stop()  
 raise e  
  
# Evaluate on test set  
predictions = model.transform(test\_df)  
correct = predictions.filter(col("is\_delayed") == col("prediction")).count()  
total = predictions.count()  
accuracy = correct / total if total > 0 else 0  
print(f"Test accuracy: {accuracy:.2%} ({correct}/{total} correct)")  
  
# Save model  
model\_path = "D:/flight\_delay\_model"  
try:  
 # Remove existing model directory if it exists  
 if os.path.exists(model\_path):  
 shutil.rmtree(model\_path)  
 model.save(model\_path)  
 print(f"Model saved to {model\_path}")  
except Exception as e:  
 print("Error saving model:", str(e))  
 spark.stop()  
 raise e  
  
# Clean up  
spark.stop()  
print("Spark session stopped")

* \*\*Purpose\*\*: Trains a RandomForestClassifier to predict `is\_delayed` using `D:/flight\_data\_sample.json` and saves the model to `D:/flight\_delay\_model` for use in the pipeline.
* \*\*Key Lines\*\*:
* `df = spark.read.json("D:/flight\_data\_sample.json")`: Loads 1000 synthetic records.
* `df.withColumn("is\_delayed", ...)`: Creates target column (1 if `delay\_minutes` > 15).
* `VectorAssembler(inputCols=["temperature", ...])`: Combines features into a vector.
* `RandomForestClassifier(numTrees=100, maxDepth=10)`: Defines the classifier.
* `model.save(model\_path)`: Saves the trained model, overwriting if necessary.

### `flight\_pipeline.ipynb` - Table Creation (Cell 1)

import sqlite3  
import os  
  
db\_path = "D:/flight\_db/flight\_predictions.db"  
if os.path.exists(db\_path):  
 print(f"Database exists at {db\_path}")  
else:  
 print(f"Database not found at {db\_path}, creating new")  
  
conn = sqlite3.connect(db\_path)  
cursor = conn.cursor()  
  
cursor.execute("DROP TABLE IF EXISTS predictions")  
cursor.execute("CREATE TABLE predictions (flight\_number TEXT, scheduled\_departure TEXT, delay\_minutes INTEGER, is\_delayed INTEGER, predicted\_delayed REAL, temperature REAL, departure\_airport TEXT)")  
conn.commit()  
  
cursor.execute("PRAGMA table\_info(predictions)")  
schema = cursor.fetchall()  
print("Table schema:")  
for col in schema:  
 print(col)  
  
conn.close()  
print("Table recreated")

* \*\*Purpose\*\*: Creates or recreates the `predictions` table in SQLite to store pipeline outputs, ensuring the database is ready before streaming predictions.
* \*\*Key Lines\*\*:
* `DROP TABLE IF EXISTS predictions`: Avoids schema conflicts by clearing old table.
* `CREATE TABLE predictions (...)`: Defines columns for storing flight data and predictions.
* `PRAGMA table\_info(predictions)`: Verifies the table schema.

### `flight\_pipeline.ipynb` - Streaming Cell (Cell 2)

import os  
os.environ["PYSPARK\_PYTHON"] = "D:\\python39venv\\Scripts\\python.exe"  
os.environ["PYSPARK\_DRIVER\_PYTHON"] = "D:\\python39venv\\Scripts\\python.exe"  
os.environ["HADOOP\_HOME"] = "D:\\hadoop-3.3.6"  
  
from pyspark.sql import SparkSession  
from pyspark.sql.functions import col, when, from\_json  
from pyspark.sql.types import StructType, StructField, StringType, DoubleType, IntegerType  
from pyspark.ml import PipelineModel  
import plotly.express as px  
import pandas as pd  
import sqlite3  
  
# SparkSession for model loading  
spark = SparkSession.builder \  
 .appName("FlightDelayStreaming") \  
 .master("local[\*]") \  
 .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.5") \  
 .config("spark.hadoop.fs.defaultFS", "file:///") \  
 .config("spark.hadoop.fs.file.impl", "org.apache.hadoop.fs.LocalFileSystem") \  
 .config("spark.driver.memory", "4g") \  
 .config("spark.executor.memory", "4g") \  
 .getOrCreate()  
  
print("Spark version:", spark.version)  
print("Hadoop home:", os.environ.get("HADOOP\_HOME"))  
  
try:  
 model = PipelineModel.load("D:/flight\_delay\_model")  
 print("Model loaded successfully")  
except Exception as e:  
 print("Error loading model:", str(e))  
 spark.stop()  
 raise e  
  
spark.stop()  
spark = SparkSession.builder \  
 .appName("FlightDelayStreaming") \  
 .master("local[\*]") \  
 .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.5") \  
 .config("spark.hadoop.fs.defaultFS", "file:///") \  
 .config("spark.hadoop.fs.file.impl", "org.apache.hadoop.fs.RawLocalFileSystem") \  
 .config("spark.hadoop.fs.file.impl.disable.cache", "true") \  
 .config("spark.hadoop.hadoop.io.native.lib.available", "false") \  
 .config("spark.driver.memory", "4g") \  
 .config("spark.executor.memory", "4g") \  
 .getOrCreate()  
  
try:  
 kafka\_df = spark.readStream \  
 .format("kafka") \  
 .option("kafka.bootstrap.servers", "localhost:9092") \  
 .option("subscribe", "flight-data") \  
 .option("startingOffsets", "latest") \  
 .load()  
 print("Connected to Kafka")  
except Exception as e:  
 print("Kafka connection error:", str(e))  
 spark.stop()  
 raise e  
  
schema = StructType([  
 StructField("flight\_number", StringType()),  
 StructField("departure\_airport", StringType()),  
 StructField("arrival\_airport", StringType()),  
 StructField("scheduled\_departure", StringType()),  
 StructField("temperature", DoubleType()),  
 StructField("wind\_speed", DoubleType()),  
 StructField("precipitation", DoubleType()),  
 StructField("delay\_minutes", IntegerType())  
])  
flight\_df = kafka\_df.selectExpr("CAST(value AS STRING)") \  
 .select(from\_json(col("value"), schema).alias("data")) \  
 .select("data.\*")  
  
flight\_df = flight\_df.withColumn("is\_delayed", when(col("delay\_minutes") > 15, 1).otherwise(0))  
  
pred\_df = model.transform(flight\_df)  
  
output\_df = pred\_df.select(  
 col("flight\_number"),  
 col("scheduled\_departure"),  
 col("delay\_minutes"),  
 col("is\_delayed"),  
 col("prediction").alias("predicted\_delayed"),  
 col("temperature"),  
 col("departure\_airport")  
)  
  
output\_df.printSchema()  
  
console\_query = output\_df.writeStream \  
 .outputMode("append") \  
 .format("console") \  
 .start()  
  
def write\_to\_sqlite(batch\_df, batch\_id):  
 try:  
 pandas\_df = batch\_df.toPandas()  
 print(f"Batch {batch\_id}: {len(pandas\_df)} rows to save")  
 if not pandas\_df.empty:  
 conn = sqlite3.connect("D:/flight\_db/flight\_predictions.db")  
 pandas\_df.to\_sql("predictions", conn, if\_exists="append", index=False)  
 conn.commit()  
 print(f"Batch {batch\_id}: Saved {len(pandas\_df)} rows to SQLite")  
 conn.close()  
 else:  
 print(f"Batch {batch\_id}: No data to save")  
 except Exception as e:  
 print(f"Batch {batch\_id}: SQLite error: {str(e)}")  
  
sqlite\_query = output\_df.writeStream \  
 .outputMode("append") \  
 .trigger(processingTime="10 seconds") \  
 .foreachBatch(write\_to\_sqlite) \  
 .start()  
  
def plot\_batch(batch\_df, batch\_id):  
 pandas\_df = batch\_df.toPandas()  
 if not pandas\_df.empty:  
 fig = px.scatter(  
 pandas\_df,  
 x="scheduled\_departure",  
 y="delay\_minutes",  
 color="predicted\_delayed",  
 size="temperature",  
 title="Flight Delays: Predicted vs Actual",  
 labels={  
 "scheduled\_departure": "Time",  
 "delay\_minutes": "Delay (min)",  
 "predicted\_delayed": "Predicted Delayed",  
 "temperature": "Temp (°C)"  
 },  
 hover\_data=["flight\_number", "departure\_airport"]  
 )  
 fig.update\_traces(marker=dict(sizemode='area', sizemin=5))  
 fig.show()  
  
plot\_query = output\_df.writeStream \  
 .outputMode("append") \  
 .format("memory") \  
 .queryName("flight\_predictions") \  
 .trigger(processingTime="10 seconds") \  
 .foreachBatch(plot\_batch) \  
 .start()  
  
def alert\_delayed(batch\_df, batch\_id):  
 delayed = batch\_df.filter(col("predicted\_delayed") == 1.0).toPandas()  
 if not delayed.empty:  
 print("Delayed flights:\n", delayed[["flight\_number", "delay\_minutes", "scheduled\_departure"]].to\_string(index=False))  
  
alert\_query = output\_df.writeStream \  
 .outputMode("append") \  
 .trigger(processingTime="10 seconds") \  
 .foreachBatch(alert\_delayed) \  
 .start()  
  
try:  
 alert\_query.awaitTermination(timeout=180)  
except KeyboardInterrupt:  
 console\_query.stop()  
 sqlite\_query.stop()  
 plot\_query.stop()  
 alert\_query.stop()  
 spark.stop()  
 print("Streaming stopped")

* \*\*Purpose\*\*: Streams Kafka data, applies the pre-trained ML model, saves predictions to the `predictions` table, visualizes delays with Plotly, and alerts for delayed flights.
* \*\*Key Lines\*\*:
* `model = PipelineModel.load("D:/flight\_delay\_model")`: Loads the model from `train\_flight\_model.ipynb`.
* `pred\_df = model.transform(flight\_df)`: Applies predictions to streaming data.
* `write\_to\_sqlite(batch\_df, batch\_id)`: Saves predictions to `D:/flight\_db/flight\_predictions.db`.
* `plot\_batch(batch\_df, batch\_id)`: Generates scatter plots every 10 seconds.
* `alert\_delayed(batch\_df, batch\_id)`: Prints alerts for delayed flights.

### `flight\_pipeline.ipynb` - Stats Cell (Cell 3)

#avg flight delay  
import sqlite3  
  
conn = sqlite3.connect("D:/flight\_db/flight\_predictions.db")  
cursor = conn.cursor()  
  
cursor.execute("SELECT departure\_airport, COUNT(\*) as total, SUM(CASE WHEN is\_delayed = 1 THEN 1 ELSE 0 END) as delayed FROM predictions GROUP BY departure\_airport")  
print("Delays by Airport:")  
for row in cursor.fetchall():  
 airport, total, delayed = row  
 print(f"{airport}: {delayed}/{total} ({delayed/total:.2%})")  
  
cursor.execute("SELECT AVG(delay\_minutes) FROM predictions WHERE is\_delayed = 1")  
avg\_delay = cursor.fetchone()[0] or 0  
print(f"Average delay (delayed flights): {avg\_delay:.2f} minutes")  
  
conn.close()

* \*\*Purpose\*\*: Analyzes delay trends by airport and computes the average delay for delayed flights.
* \*\*Key Lines\*\*:
* `SELECT departure\_airport, COUNT(\*) ...`: Aggregates total and delayed flights per airport.
* `SELECT AVG(delay\_minutes)`: Calculates average delay for flights with `is\_delayed = 1`.

### `flight\_pipeline.ipynb` - Accuracy Cell (Cell 4)

#accuracy  
import sqlite3  
  
conn = sqlite3.connect("D:/flight\_db/flight\_predictions.db")  
cursor = conn.cursor()  
  
cursor.execute("SELECT COUNT(\*) FROM predictions WHERE is\_delayed = predicted\_delayed")  
correct = cursor.fetchone()[0]  
cursor.execute("SELECT COUNT(\*) FROM predictions")  
total = cursor.fetchone()[0]  
  
accuracy = correct / total if total > 0 else 0  
print(f"Accuracy: {accuracy:.2%} ({correct}/{total} correct)")  
  
conn.close()

* \*\*Purpose\*\*: Calculates the model’s prediction accuracy by comparing `is\_delayed` and `predicted\_delayed`.
* \*\*Key Lines\*\*:
* `WHERE is\_delayed = predicted\_delayed`: Counts correct predictions.
* `accuracy = correct / total ...`: Computes accuracy percentage.

## Troubleshooting Steps

### Error: `table predictions has no column named temperature`

* \*\*Cause\*\*: `predictions` table lacks `temperature` or `departure\_airport`.
* \*\*Fix\*\*:
* Run Cell 2 in `flight\_pipeline.ipynb` \*before\* Cell 1.
* Or CLI:

(python39venv) D:\>sqlite3 D:\flight\_db\flight\_predictions.db  
 sqlite> DROP TABLE IF EXISTS predictions;  
 sqlite> CREATE TABLE predictions (flight\_number TEXT, scheduled\_departure TEXT, delay\_minutes INTEGER, is\_delayed INTEGER, predicted\_delayed REAL, temperature REAL, departure\_airport TEXT);  
 sqlite> .exit

* \*\*Verify\*\*:

(python39venv) D:\>sqlite3 D:\flight\_db\flight\_predictions.db ".schema predictions"

### Error: `Kafka connection error`

* \*\*Cause\*\*: Kafka/Zookeeper down or port `9092` blocked.
* \*\*Fix\*\*:

(python39venv) D:\>python kafka\_control.py  
 (python39venv) D:\>netstat -aon | findstr :9092

* If blocked: `taskkill /F /PID <pid>`.
* \*\*Verify\*\*:

(python39venv) D:\>D:\kafka\_2.12-3.7.0\bin\windows\kafka-console-consumer.bat --topic flight-data --bootstrap-server localhost:9092

### Error: `No data in SQLite`

* \*\*Cause\*\*: Producer stopped, write failed, or table not created.
* \*\*Fix\*\*:
* Ensure Cell 2 runs before Cell 1.
* Restart:

(python39venv) D:\>python generate\_flight\_data.py

* Check memory table:

spark.sql("SELECT \* FROM flight\_predictions").show()

* \*\*Verify\*\*:

(python39venv) D:\>sqlite3 D:\flight\_db\flight\_predictions.db "SELECT \* FROM predictions"

### Error: `Model not found at D:/flight\_delay\_model`

* \*\*Cause\*\*: `train\_flight\_model.ipynb` failed to save model.
* \*\*Fix\*\*:
* Ensure `train\_flight\_model.ipynb` runs successfully (check `model.save(model\_path)`).
* Verify `D:/flight\_data\_sample.json` exists.
* \*\*Verify\*\*:

(python39venv) D:\>dir D:\flight\_delay\_model

### Error: `write() takes 1 positional argument but 2 were given`

* \*\*Cause\*\*: Incorrect `model.write("D:/flight\_delay\_model")` in `train\_flight\_model.ipynb`.
* \*\*Fix\*\*:
* Updated to `model.save("D:/flight\_delay\_model")` with `shutil.rmtree()` to overwrite.
* \*\*Verify\*\*:
* Run `train\_flight\_model.ipynb` and check for `Model saved to D:/flight\_delay\_model`.

## Desired Outputs

* \*\*SQLite Database\*\*:
* Table: `predictions`.
* Columns: `flight\_number`, `scheduled\_departure`, `delay\_minutes`, `is\_delayed`, `predicted\_delayed`, `temperature`, `departure\_airport`.
* Rows: >100 after 3-minute stream.
* Sample: `FL100|2025-04-14 16:00:00|0|0|0.0|22.5|AYD`.
* \*\*Plotly Visuals\*\*:
* Scatter plot every 10 seconds.
* X: `scheduled\_departure`, Y: `delay\_minutes`.
* Color: `predicted\_delayed` (blue/orange).
* Size: `temperature`.
* Hover: `flight\_number`, `departure\_airport`.
* \*\*Console\*\*:
* Batches: `FL100`, `0`, `0.0`, etc.
* Alerts: `Delayed flights: FL101 20 2025-04-14 16:05:00`.
* \*\*Metrics\*\*:
* Accuracy: 74.70% (378/506 correct).
* Stats: `AYD: 31/129 (24.03%)`, `HYD: 32/116 (27.59%)`, avg delay `38.11 minutes`.
* \*\*ML Model\*\*:
* Folder: `D:/flight\_delay\_model`.
* Test accuracy: ~70-80% (varies with data; note: 100% may indicate overfitting).

## Dependencies and Installations

### Dependencies

* \*\*Python\*\*: 3.9.13.
* \*\*PySpark\*\*: 3.5.5.
* \*\*Spark\*\*: 3.5.5.
* \*\*Kafka\*\*: 2.12-3.7.0.
* \*\*Hadoop\*\*: 3.3.6 (`winutils.exe`).
* \*\*Libraries\*\*:
* `pandas`
* `numpy`
* `plotly`
* `kafka-python`
* `jupyter`
* \*\*Scripts\*\*:
* `kafka\_control.py`: Controls Kafka/Zookeeper.
* `generate\_flight\_data.py`: Generates flight data.
* `train\_flight\_model.ipynb`: Trains ML model.
* \*\*Tools\*\*: SQLite Browser.

### Installation Commands

D:\>python -m venv D:\python39venv  
D:\>D:\python39venv\Scripts\activate  
(python39venv) D:\>pip install pyspark==3.5.5 pandas numpy plotly kafka-python jupyter

### Script Setup

* \*\*Save Scripts\*\*:
* Create `kafka\_control.py`, `generate\_flight\_data.py`, `train\_flight\_model.ipynb` in `D:\`.
* Verify:

(python39venv) D:\>dir kafka\_control.py generate\_flight\_data.py train\_flight\_model.ipynb

* \*\*Test Scripts\*\*:
* Kafka:

(python39venv) D:\>python kafka\_control.py

* Producer:

(python39venv) D:\>python generate\_flight\_data.py

* ML Training:

(python39venv) D:\>jupyter notebook

* Open and run `train\_flight\_model.ipynb`.

### Environment Variables

(python39venv) D:\>set HADOOP\_HOME=D:\hadoop-3.3.6  
(python39venv) D:\>set PATH=%PATH%;D:\hadoop-3.3.6\bin  
(python39venv) D:\>set PYSPARK\_PYTHON=D:\python39venv\Scripts\python.exe  
(python39venv) D:\>set PYSPARK\_DRIVER\_PYTHON=D:\python39venv\Scripts\python.exe

## Appendices

### Appendix 1: Dependency Setup

* \*\*Virtual Environment\*\*:

D:\>python -m venv D:\python39venv  
 D:\>D:\python39venv\Scripts\activate

* \*\*Libraries\*\*:

(python39venv) D:\>pip install pyspark==3.5.5 pandas numpy plotly kafka-python jupyter

* \*\*Kafka\*\*:
* Download: `kafka\_2.12-3.7.0.tgz` from kafka.apache.org.
* Extract to `D:\kafka\_2.12-3.7.0`.
* \*\*Hadoop\*\*:
* Download: `winutils.exe` for Hadoop 3.3.6.
* Place in `D:\hadoop-3.3.6\bin`.
* \*\*Spark\*\*:
* Download: `spark-3.5.5-bin-hadoop3.tgz` from spark.apache.org.
* Extract to `D:\spark-3.5.5-bin-hadoop3`.
* \*\*Scripts\*\*:
* Save `kafka\_control.py`, `generate\_flight\_data.py`, `train\_flight\_model.ipynb` to `D:\`.

### Appendix 2: Cleanup Steps

* \*\*Stop Streaming\*\*:
* Jupyter: `Ctrl+C` in Cell 2 of `flight\_pipeline.ipynb` or interrupt the kernel.
* \*\*Stop Kafka\*\*:

(python39venv) D:\>python kafka\_control.py stop

* \*\*Stop Producer\*\*:
* Cmd: `Ctrl+C` in `generate\_flight\_data.py`.
* \*\*Stop Jupyter\*\*:
* File > Shut Down.
* \*\*Clear SQLite\*\*:

(python39venv) D:\>del D:\flight\_db\flight\_predictions.db

* \*\*Verify\*\*:

(python39venv) D:\>netstat -aon | findstr "2181 9092 4040"  
 (python39venv) D:\>tasklist | findstr "java python"

### Appendix 3: Performance Enhancements

* \*\*Kafka\*\*:
* Add partitions:

(python39venv) D:\>D:\kafka\_2.12-3.7.0\bin\windows\kafka-topics.bat --alter --topic flight-data --partitions 4 --bootstrap-server localhost:9092

* In `generate\_flight\_data.py`, set `linger.ms`: controls how long a producer waits before sending a batch of records, even if the batch hasn't reached its maximum size

producer = KafkaProducer(bootstrap\_servers="localhost:9092", value\_serializer=lambda v: json.dumps(v).encode("utf-8"), linger\_ms=10)

* \*\*Spark\*\*:
* Increase memory: `spark.driver.memory=6g`.
* Cache data:

flight\_df.cache()

* \*\*SQLite\*\*:
* Index `flight\_number`:

CREATE INDEX idx\_flight ON predictions(flight\_number);

* Enable WAL : ensures data integrity and durability by logging all changes to the database before they are actually applied to the database files

conn.execute("PRAGMA journal\_mode=WAL")

* \*\*Best Practices\*\*:
* Monitor Spark UI: `http://localhost:4040`.
* Log errors:

import logging  
 logging.basicConfig(filename="pipeline.log")

* Backup SQLite:

(python39venv) D:\>copy D:\flight\_db\flight\_predictions.db D:\flight\_db\backup.db