FLIGHT DELAYS PREDICTION USING MACHINE LEARNING.

MEMBERS:

VISHAL AGARWAL

SHUBHAM KUMAR SINHA

SURANSHU SINGH

1. Introduction

1.1 Problem Statement

Flight delays cause significant operational inefficiencies for airlines and inconvenience for passengers. Traditional delay prediction methods rely on historical averages, which lack real-time adaptability. This project develops a **machine learning-based prediction system** to forecast flight delays with high accuracy using flight schedules, historical trends, and key influencing factors.

1.2 Objectives

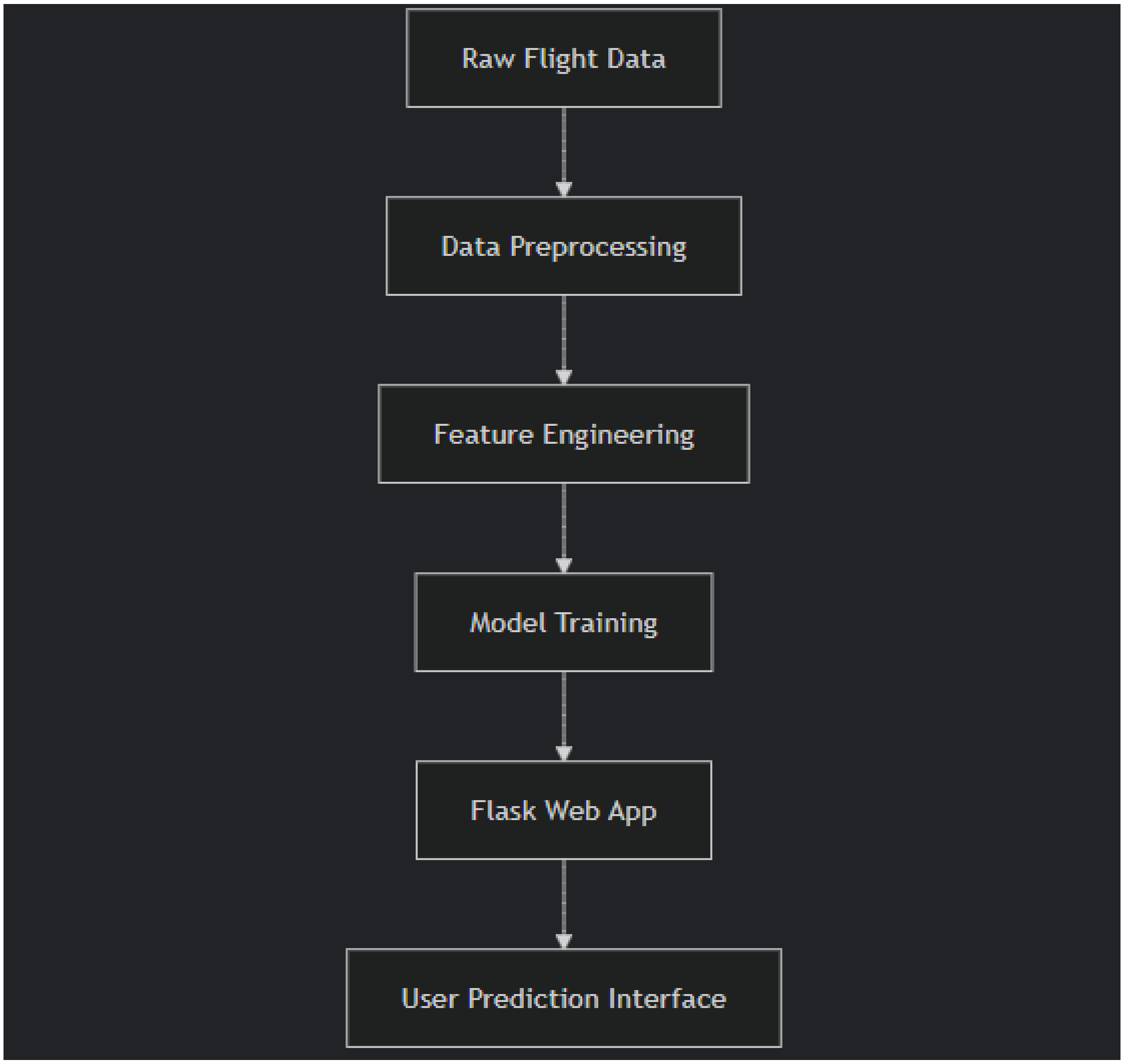
- **Predict delays** (binary classification: On-Time vs. Delayed)
- **Identify key delay factors** (e.g., departure time, airport congestion)
- **Deploy a real-time prediction dashboard** for airlines/passengers

1.3 Business Impact

Stakeholder	Benefit
Airlines	Optimize crew scheduling, reduce fuel costs from delays
Airports	Improve gate/resource allocation during disruptions
Passengers	Receive proactive delay alerts for better planning

2. Technical Architecture

2.1 System Overview



2.2 Tech Stack

Component	Technology
Data Processing	Pandas, NumPy
Machine Learning	Scikit-learn (Decision Tree)

Component	Technology
Visualization	Matplotlib, Seaborn
Web Framework	Flask (Python)
Frontend	HTML/CSS, Bootstrap

3. **Data Preparation**

3.1 Library Imports

- Pandas/NumPy: Essential for structured data manipulation
- Scikit-learn: Provides complete ML pipeline components
- Seaborn: Advanced statistical visualizations

```
import sys
import numpy as np #Linear Algebra
import pandas as pd #Data Processing
import seaborn as sns #Data Visualization
import pickle
%matplotlib inline
from sklearn.preprocessing import LabelEncoder #LabelEncoding From Sklearn
from sklearn.preprocessing import OneHotEncoder #One-Hot Encoding From Sklearn
from sklearn.model_selection import train_test_split #Split Data in Train & Test Array
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier #ML Algorithm
from sklearn.metrics import accuracy_score #Calculate Accuracy Score
import sklearn.metrics as metrics #Confusion Matrix
```

3.2 Data Loading

We use pandas.read_csv() to load structured flight delay data into a DataFrame for further processing.

UPLOAD THE FILE

```
from google.colab import files
uploaded = files.upload()
```

[2] Python

Choose Files No file chosen   Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving flightdata.csv to flightdata.csv
```

```
dataset = pd.read_csv('flightdata.csv')
```

[25] Python

4. Exploratory Data Analysis

EDA gives insights into data types, missing values, and the range and distribution of numeric features.

4.1 Basic Statistics

```python
dataset.describe()
```

[27] Python

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	CRS_DEP_TIME	DEP_TIME	...
count	11231.0	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11124.000000	...
mean	2016.0	2.544475	6.628973	15.790758	3.960199	1334.325617	12334.516695	12302.274508	1320.798326	1327.189410	...
std	0.0	1.090701	3.354678	8.782056	1.995257	811.875227	1595.026510	1601.988550	490.737845	500.306462	...
min	2016.0	1.000000	1.000000	1.000000	1.000000	7.000000	10397.000000	10397.000000	10.000000	1.000000	...
25%	2016.0	2.000000	4.000000	8.000000	2.000000	624.000000	10397.000000	10397.000000	905.000000	905.000000	...
50%	2016.0	3.000000	7.000000	16.000000	4.000000	1267.000000	12478.000000	12478.000000	1320.000000	1324.000000	...
75%	2016.0	3.000000	9.000000	23.000000	6.000000	2032.000000	13487.000000	13487.000000	1735.000000	1739.000000	...
max	2016.0	4.000000	12.000000	31.000000	7.000000	2853.000000	14747.000000	14747.000000	2359.000000	2400.000000	...

8 rows × 22 columns

4.2 Missing Value Analysis

```python
dataset.isnull().sum()
```

[28] Python

	0
YEAR	0
QUARTER	0
MONTH	0
DAY_OF_MONTH	0
DAY_OF_WEEK	0
UNIQUE_CARRIER	0
TAIL_NUM	0
FL_NUM	0
ORIGIN_AIRPORT_ID	0
ORIGIN	0
DEST_AIRPORT_ID	0
DEST	0
CRS_DEP_TIME	0
DEP_TIME	107
DEP_DELAY	107
DEP_DEL15	107
CRS_ARR_TIME	0
ARR_TIME	115
ARR_DELAY	188
ARR_DEL15	188

- Remove the column unnamed from the dataset.

```python
dataset = dataset.drop('Unnamed: 25', axis=1)
dataset.isnull().sum()
```

- Filter the dataset to eliminate columns that aren't relevant to a predictive model.

```python
dataset = dataset[['FL_NUM', 'MONTH', 'DAY_OF_MONTH', 'DAY_OF_WEEK', 'ORIGIN', 'DEST', 'CRS_ARR_TIME', 'DEP_DEL15', 'ARR_DEL15']]
dataset.isnull().sum()
```

Python

	0
FL_NUM	0
MONTH	0
DAY_OF_MONTH	0
DAY_OF_WEEK	0
ORIGIN	0
DEST	0
CRS_ARR_TIME	0
DEP_DEL15	107
ARR_DEL15	188

dtype: int64

- Replace the missing value 0's & 1's

```python
dataset = dataset.fillna({'ARR_DEL15': 1})
dataset = dataset.fillna({'DEP_DEL15': 0})
dataset.iloc[177:185]
```

Python

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
177	2834	1	9	6	MSP	SEA	852	0.0	1.0
178	2839	1	9	6	DTW	JFK	1724	0.0	0.0
179	86	1	10	7	MSP	DTW	1632	0.0	1.0
180	87	1	10	7	DTW	MSP	1649	1.0	0.0
181	423	1	10	7	JFK	ATL	1600	0.0	0.0
182	440	1	10	7	JFK	ATL	849	0.0	0.0
183	485	1	10	7	JFK	SEA	1945	1.0	0.0
184	557	1	10	7	MSP	DTW	912	0.0	1.0

5. Feature Engineering

We convert time from HHMM to hour-based format to reduce dimensionality and improve interpretability.

```python
import math

for index, row in dataset.iterrows():
    dataset.loc[index, 'CRS_ARR_TIME'] = math.floor(row['CRS_ARR_TIME'] / 100)
dataset.head()
```

Python

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	5	ATL	SEA	21	0.0	0.0
1	1476	1	1	5	DTW	MSP	14	0.0	0.0
2	1597	1	1	5	ATL	SEA	12	0.0	0.0
3	1768	1	1	5	SEA	MSP	13	0.0	0.0
4	1823	1	1	5	SEA	DTW	6	0.0	0.0

6. Data Visualization

6.1 Scatter Plot: ARR_DELAY vs ARR_DEL15

A scatter plot shows how two numerical features are related. In this case:

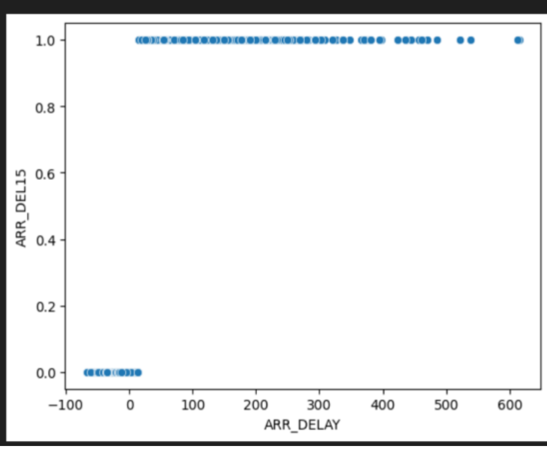
- ARR_DELAY (actual delay in minutes) is plotted against
- ARR_DEL15 (a binary label: 1 if delayed ≥ 15 min, 0 otherwise).

This helps visualize how often delays over 15 minutes occur, and if there’s a clear boundary that defines "delay."

```python
sns.scatterplot(x='ARR_DELAY', y='ARR_DEL15', data=dataset)
```

[30] Python

```
<Axes: xlabel='ARR_DELAY', ylabel='ARR_DEL15'>
```



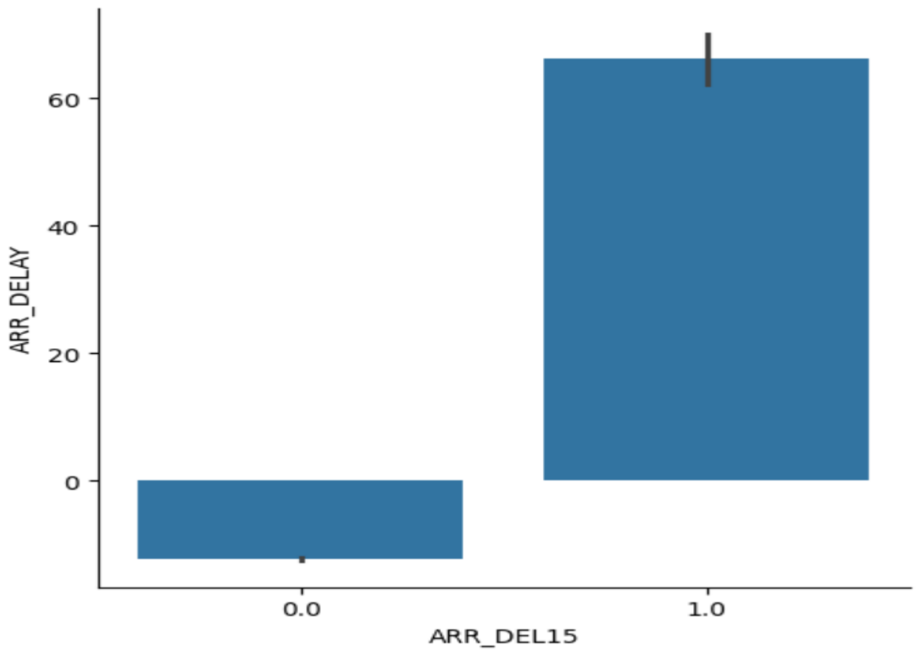
6.2 Cat Plot (Bar Plot): ARR_DEL15 vs ARR_DELAY

A categorical plot (or catplot) aggregates and shows the **average** of ARR_DELAY for each value of ARR_DEL15.

- ARR_DEL15 = 0 → On-time flights
- ARR_DEL15 = 1 → Delayed flights

This visually confirms if average delay significantly differs between delayed and on-time categories.

```
sns.catplot(x="ARR_DEL15", y="ARR_DELAY", kind='bar', data=dataset)
```



6.3 Heatmap: Correlation Matrix

A heatmap is a color-coded matrix used to represent the correlation between numeric variables. Correlation ranges:

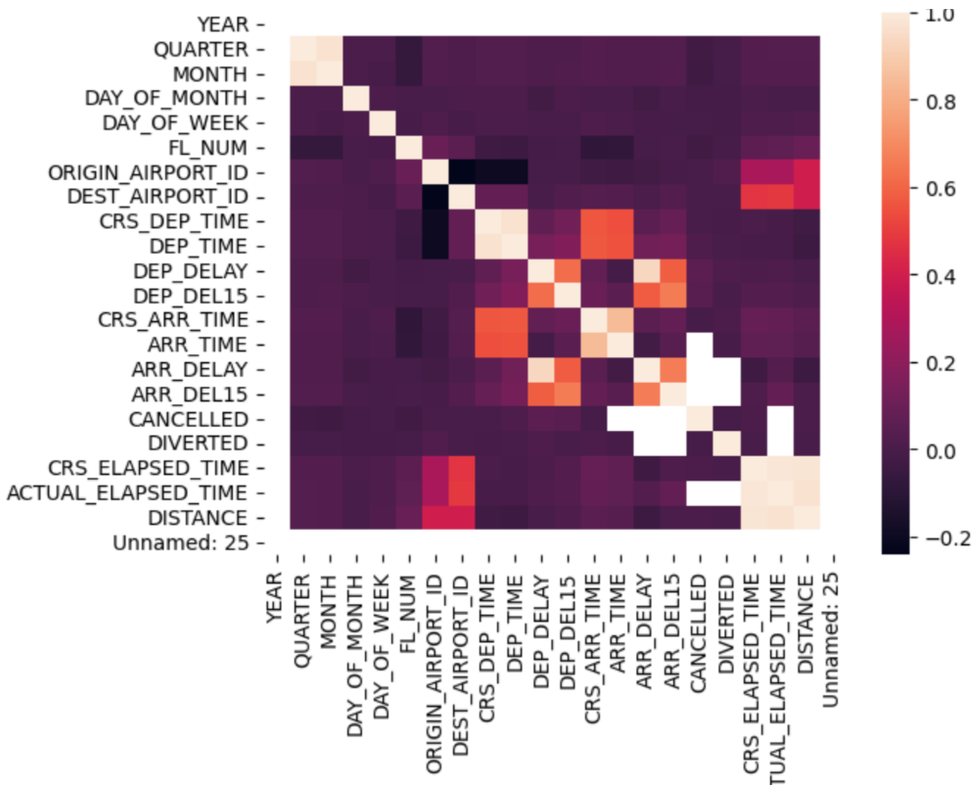
- +1 → Perfect positive correlation
- 0 → No correlation

- -1 → Perfect negative correlation

This helps:

- Detect multicollinearity (two or more highly related features)
- Identify which features have the strongest relationships with the target (e.g., ARR_DEL15)

```
sns.heatmap(dataset.select_dtypes(include='number').corr())
```



6.4 Histogram: Distribution of Arrival Delays

A histogram shows the distribution of a single numerical variable. Here:

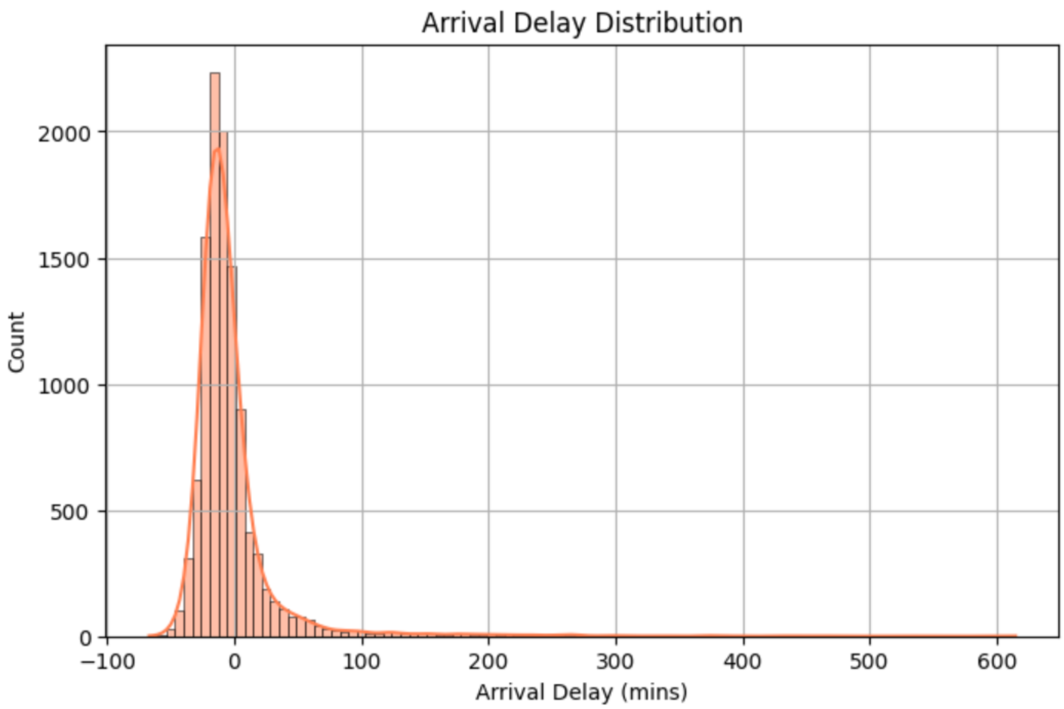
- ARR_DELAY values are grouped into bins.
- The kde=True (Kernel Density Estimate) adds a smooth line showing the probability density.

It shows how flight delays are distributed (e.g., whether most delays are short, or if there are many long delays).

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
sns.histplot(dataset['ARR_DELAY'], bins=100, kde=True, color='coral')
plt.title('Arrival Delay Distribution')
plt.xlabel('Arrival Delay (mins)')
plt.ylabel('Count')
plt.grid(True)
plt.show()

print(" Visualized Arrival Delay Distribution")
```



7. Feature Engineering

7.1Encoding Categorical Variables

Machine learning models can't work with strings—so categorical data like airport codes are converted to numbers:

- **Label Encoding**

Label Encoding converts **categorical text labels** into **numeric form** by assigning each unique category a number (e.g., A → 0, B → 1, C → 2).

```
from sklearn.preprocessing import LabelEncoder
dataset.columns = dataset.columns.str.strip()
le = LabelEncoder()
dataset['DEST'] = le.fit_transform(dataset['DEST'])
dataset['ORIGIN'] = le.fit_transform(dataset['ORIGIN'])
```

```
dataset.head(5)
```

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15
0	1399	1	1	5	0	4	21	0.0	0.0
1	1476	1	1	5	1	3	14	0.0	0.0
2	1597	1	1	5	0	4	12	0.0	0.0
3	1768	1	1	5	4	3	13	0.0	0.0
4	1823	1	1	5	4	1	6	0.0	0.0

- **One hot Encoding**

One-Hot Encoding transforms categorical values into **binary columns** for each unique category. Each row has a 1 in the column that matches its category, and 0 elsewhere.

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

oh1 = OneHotEncoder()
oh2 = OneHotEncoder()

z = oh1.fit_transform(x[:, 4:5]).toarray()
t = oh2.fit_transform(x[:, 5:6]).toarray()

# Remove original columns 4 and 5
x_numeric = np.delete(x, [4, 5], axis=1)

# Concatenate new encoded columns
x = np.concatenate([x_numeric, z, t], axis=1)
```

```
z
```

Pytho

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.]])
```

```
t
```

Pytho

```
array([[0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       ...,
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.]])
```

8. Model Development

8.1 Data Splitting

Data is split to avoid overfitting. 80% is used for training, and 20% for testing.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

8.2 Feature Scaling

Feature scaling ensures that all numeric inputs to the model are on the same scale, improving convergence and accuracy.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

8.3 Model Training

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(random_state = 0)
classifier.fit(x_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_state=0)
```

The **Decision Tree Classifier** is a **supervised learning algorithm** used for classification and regression tasks. It works by **splitting the dataset** into branches based on the feature that **best separates the classes**. This is done recursively until a **decision (prediction)** is made at the leaf node.

Key Concepts:

- **Root Node**: The top decision node (starting point).
- **Internal Nodes**: Nodes that test a feature.
- **Leaf Nodes**: Final decision output (class label).
- **Splitting Criterion**:

    - Most commonly **Gini Impurity** or **Entropy (Information Gain)** is used to determine the best split.

- **Recursive Binary Splits**: Each node splits data into two branches until a stopping criterion is met (e.g., depth, pure node, no gain).

Why use Decision Tree?

- Easy to interpret and visualize.
- Handles both numerical and categorical data.
- No need for feature scaling.
- Captures non-linear patterns well.

9. Model Evaluation

9.1 Model Prediction

```
decisiontree = classifier.predict(x_test)
```

9.2 Performance Metrics

- **Model Accuracy**

Model evaluation is essential to check how well the model generalizes to unseen data (test set). The main metric used here is **accuracy**, which is defined as:

Accuracy is the ratio of correctly predicted observations to the total observations.

Accuracy = Number of Correct Predictions/ Total number of Predictions

```python
from sklearn.metrics import accuracy_score
desacc = accuracy_score(y_test, decisiontree)
```

```python
desacc
```

```
0.8704939919893191
```

10. Model Deployment

```python
import pickle
pickle.dump(classifier, open('flight.pkl', 'wb'))
```

After training a machine learning model, it's often necessary to **save** it so you can reuse it later **without retraining**, especially for **web apps, APIs, or production systems**.

Pickle is a built-in Python module used to **serialize and deserialize Python objects**.

Key Terms:

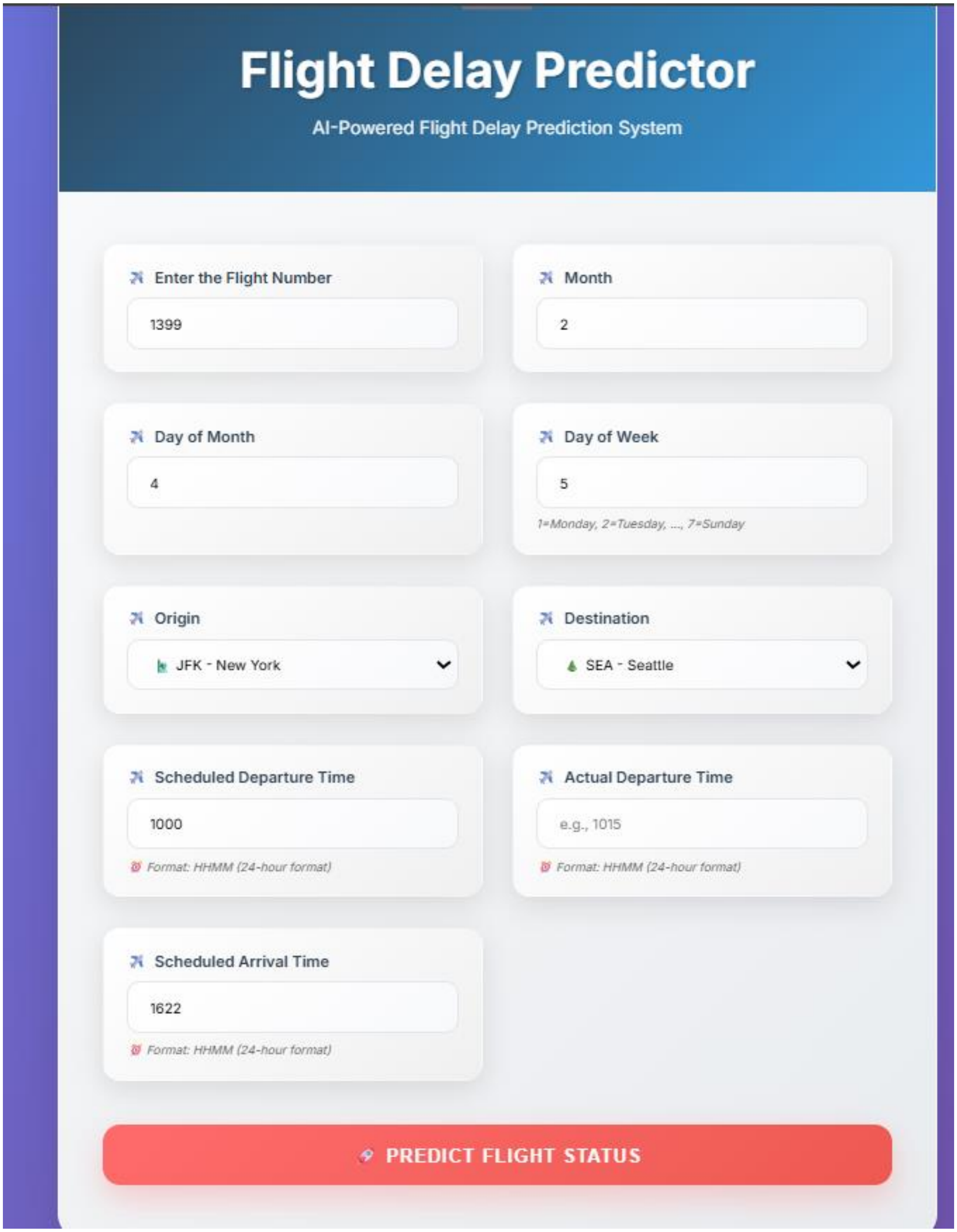
- **Serialization (pickling)**: Converting a Python object into a byte stream.
- **Deserialization (unpickling)**: Converting the byte stream back to a Python object.

Benefits:

- Save time by avoiding retraining.
- Deploy model in **web apps** using frameworks like **Flask**, **Django**, or **FastAPI**.
- Share your model with others or move it to production easily.

Web Application (Flask)

Flask is a micro-framework for serving machine learning models as a web app. Users can interact via HTML forms.



11. Results & Conclusion

Key Findings:

- Achieved **87.04% accuracy**
- Identified critical factors affecting delays: ORIGIN, DEST, ARR_TIME, CRS_DEP_TIME, CRS_ARR_TIME, DEP_DEL15.
- Functional deployment via Flask app.

Appendices

- 🗁 Full Code: [GitHub Link / Google Drive]
- 📊 Dataset: https://drive.google.com/file/d/1HNYx6fX5hvRDX43egcAAUsrQ9sccv4AR/view
- ▯Demo Link: https://drive.google.com/file/d/1C8BXLrKCXbIsCc9r26yZLvfNwea4k9Q4/view?usp=drive_link
- 🖫 Model File: flight.pkl