Flight Delays Prediction Using Machine Learning:

Our project aims to build a Machine Learning model integrated with a Flask application to predict flight delays. By analyzing historical flight data, weather conditions, airport congestion, and other factors, we develop a predictive model. This model is then incorporated into a user-friendly Flask web application, allowing travelers and airlines to anticipate potential delays and plan accordingly, enhancing travel experiences and operational efficiency.
This project is very useful to analyse historical data and predict flight delays which is used in multiple real life scenarios, such as:

Travel Planning Assistance:

Travelers use the Flask app to check potential flight delays before booking tickets, aiding in travel planning. The model predicts delays based on historical and real-time data, enabling users to choose flights with lower delay probabilities and make informed decisions.

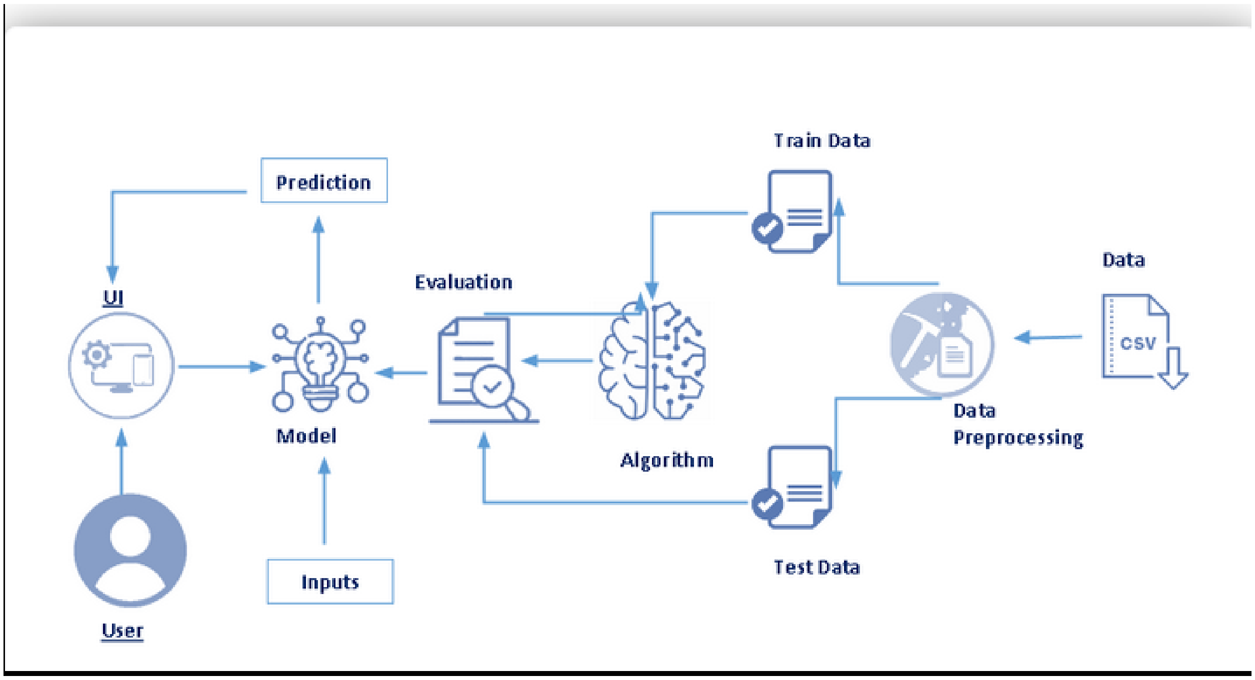
Operational Efficiency for Airlines:

Airlines utilize the app to optimize flight schedules and manage resources effectively. By anticipating delays, they can adjust crew assignments, gate allocations, and maintenance schedules proactively, minimizing disruptions and improving overall operational efficiency. This helps maintain customer satisfaction and reduces costs associated with delays and disruptions.

Airport Authority Decision Support:

Airport authorities leverage the app to monitor potential congestion and anticipate delays. By analyzing predicted delay data, they can implement proactive measures such as adjusting runway allocation and optimizing ground handling operations to mitigate delays and improve overall airport efficiency. This aids in enhancing passenger experience, reducing congestion-related issues, and ensuring smoother airport operations.

Technical architecture:



Project Data Flow :

- Input data received from user
- Input is analysed by saved best model
- Output generated and displayed to user according to prediction made by the aforementioned model

Project Steps / Roadmap:

- **Data Collection:**
  - Collect the Dataset or create the Dataset.
- **Data Preprocessing:**
  - Import the Libraries.
  - Importing the dataset.
  - Checking for Null Values.
  - Data Visualization.
  - Taking care of Missing Data.
  - Label encoding.
  - One Hot Encoding.
  - Feature Scaling.
  - Splitting Data into Train and Test.
- **Model Building:**
  - Training and testing the model

- Evaluation of Model
- **Application Building:**
  - Create an HTML file
  - Build a Python Code

Prerequisites

In order to develop this project we need to install the following software/packages:

Step 1:

Anaconda Navigator :

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with great tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code.

For this project, we will be using **Jupyter Notebooks and VSCode.**

Step 2:

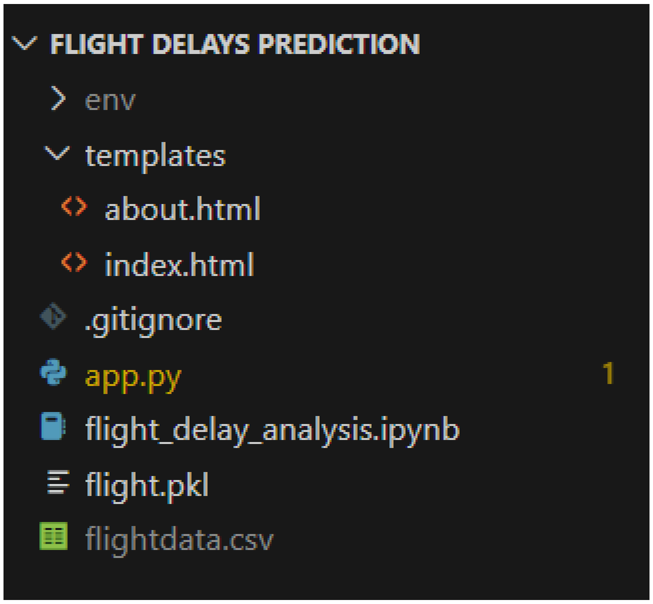
To build Machine learning models you must require the following packages

- **Sklearn:** Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.
- **NumPy:** NumPy is a Python package that stands for 'Numerical Python'. It is the core library for scientific computing, which contains a powerful n-dimensional array object
- **Pandas:** pandas is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
- **Matplotlib:** It provides an object-oriented API for embedding plots into

applications using general-purpose GUI toolkits

- **Flask:** Web framework used for building Web applications.

Project Structure:



- The project folder contains the following folders
  - flightdata.csv is the dataset used for training our model.
  - env is the virtual environment used to contain and isolate project dependencies and ensure reproducibility.
  - flight delay analysis.ipynb is the python file where the ML algorithm is applied to the dataset for testing and training. Finally, the model is saved for future use.
  - flight.pkl is the saved model used in the flask to predict the output instantly for the given inputs.
  - To build a Flask Application, save HTML pages in the templates folder and a python script app.py for server-side scripting.

Milestone 1: Define Problem/ Problem Understanding

Activity 1: Specify the Business Problem

Flight delays are a significant concern for passengers, airlines, and airports, leading to customer dissatisfaction, operational inefficiencies, and financial losses. The goal of this project is to build a machine learning model that predicts whether a flight will be delayed based on historical flight data and operational parameters (such as departure time, day of the week, and origin/destination airports).

This predictive capability can help airlines improve scheduling, proactively manage delays, and enhance passenger communication.

Activity 2: Business Requirements

A flight delay prediction project must meet the following business requirements:

- **Accurate and timely predictions:**
  The model must be trained using historical flight data that accurately reflects real-world scenarios to provide reliable predictions.

- **Integration capability:**
  The prediction system should be easily integrable into airline or airport operational dashboards for real-time use.

- **Scalability:**
  The model should handle large datasets and adapt to changes such as new routes, airports, or updated delay reporting standards.

- **User-friendly interface:**
  The deployed application should be intuitive and accessible for airline staff or passengers, providing clear and actionable predictions.

Activity 3: Literature Survey

A literature survey for a flight delay prediction project would include:

- Reviewing existing research on flight delay analytics and forecasting techniques used by airlines and airports.
- Studying previous models built with machine learning algorithms like Decision Trees, Random Forest, SVM, and Gradient Boosting to understand their performance and limitations.
- Analyzing publicly available flight datasets (such as FAA or Bureau of Transportation Statistics) to explore common delay patterns and contributing factors.
- Identifying gaps, such as handling real-time predictions or improving prediction accuracy using additional weather and traffic data.

This survey would guide model selection and ensure the project aligns with current best practices in aviation analytics.

Activity 4: Social or Business Impact

Social Impact:

- Improved passenger experience: Timely delay predictions can help passengers better plan their travel and reduce frustration caused by last-minute changes.

- Increased trust and transparency: A data-driven system improves transparency in delay communication between airlines and customers.

Business Impact:

- **Operational efficiency:** Airlines can optimize crew scheduling, ground operations, and resource allocation based on delay predictions.

- **Cost reduction:** Predicting delays in advance can help minimize costs associated with missed connections, compensations, and rescheduling.

- **Enhanced reputation:** Delivering accurate delay information improves customer satisfaction and loyalty, strengthening the airline's brand image.

Milestone 2: Data Collection & Preparation

Artificial Intelligence is a data-driven technology that relies heavily on the availability and quality of data. Without sufficient and relevant data, training an effective machine learning model is not possible. For this project, historical flight data is utilized to train and evaluate the flight delay prediction model.

Activity 1: Collect the Dataset

- For this project, we have used the **`flightdata.csv`** dataset.
- The dataset contains historical flight records, including details such as the month, day, departure times, arrival times, and delays.
- The dataset was sourced from https://www.kaggle.com/, a popular platform for datasets and data science projects.

Dataset Link:
https://drive.google.com/file/d/1HNYx6fX5hvRDX43egcAAUsrQ9sccv4AR/view?usp=sharing

Activity 1.1: Importing the Libraries

Importing the necessary libraries

The libraries are needed to load and process data, visualise trends, eventually train and evaluate the model.

```python
import os
import math
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
import pickle
```

[3] ✓ 0.0s Python

Activity 1.2: Reading the dataset

We import the dataset and read it using the function read_csv(). Data can be in any format, .xlsx, .csv, .json, etc. Pandas is really powerful tool that lets us read data in multiple formats.

Importing the dataset

```python
df = pd.read_csv('flightdata.csv')
df.columns = df.columns.str.strip()
df.head()
```

[4] ✓ 1.0s Python

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	UNIQUE_CARRIER	TAIL_NUM	FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	...	CRS_ARR_TIME	ARR_T
0	2016	1	1	1	5	DL	N836DN	1399	10397	ATL	...	2143	2
1	2016	1	1	1	5	DL	N964DN	1476	11433	DTW	...	1435	14
2	2016	1	1	1	5	DL	N813DN	1597	10397	ATL	...	1215	1
3	2016	1	1	1	5	DL	N587NW	1768	14747	SEA	...	1335	13
4	2016	1	1	1	5	DL	N836DN	1823	14747	SEA	...	607	6

5 rows × 26 columns

Activity 2: Understand the Dataset

Once collected, the dataset was analyzed using exploratory data analysis (EDA) techniques to understand its structure and identify important features for the model:

- **Loading the dataset:** The CSV file was read using `pandas` to inspect its rows, columns, and data types.

Analysing the Data

```python
print(df.shape)
df.info()
df.describe()
```

[5] ✓ 0.3s

```
(11231, 26)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11231 entries, 0 to 11230
Data columns (total 26 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   YEAR                11231 non-null  int64
 1   QUARTER             11231 non-null  int64
 2   MONTH               11231 non-null  int64
 3   DAY_OF_MONTH        11231 non-null  int64
 4   DAY_OF_WEEK         11231 non-null  int64
 5   UNIQUE_CARRIER      11231 non-null  object
 6   TAIL_NUM            11231 non-null  object
 7   FL_NUM              11231 non-null  int64
 8   ORIGIN_AIRPORT_ID   11231 non-null  int64
 9   ORIGIN              11231 non-null  object
 10  DEST_AIRPORT_ID     11231 non-null  int64
 11  DEST                11231 non-null  object
 12  CRS_DEP_TIME        11231 non-null  int64
 13  DEP_TIME            11124 non-null  float64
 14  DEP_DELAY           11124 non-null  float64
 15  DEP_DEL15           11124 non-null  float64
 16  CRS_ARR_TIME        11231 non-null  int64
 17  ARR_TIME            11116 non-null  float64
```

- **Handling missing values:** Checked for null or missing entries and cleaned them where necessary.

Handling the missing values

```
print(df.isnull().sum())
df['DEST'].unique()
```

[6] ✓ 0.0s

```
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
CANCELLED               0
DIVERTED                0
CRS_ELAPSED_TIME        0
ACTUAL_ELAPSED_TIME   188
```

- **Feature selection:** Identified important features such as `Month`, `DayofMonth`, `DayofWeek`, `DepTime`, `ArrTime`, `ActualDepTime`, `DepDelay`, and `Distance` as inputs for predicting flight delays.

Dropping Unnecessary Columns

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

[10] ✓ 0.0s

```
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
CANCELLED               0
DIVERTED                0
CRS_ELAPSED_TIME        0
ACTUAL_ELAPSED_TIME   188
DISTANCE                0
```

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

```
df = df[["FL_NUM", "MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_ARR_TIME", "DEP_DEL15", "ARR_DEL15"]]
df.isnull().sum()
```

[11] ✓ 0.0s

```
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 0s and 1s

```
df = df.fillna({"ARR_DEL15": 1})
df = df.fillna({"DEP_DEL15": 0})
df.iloc[177:185]
```

[12] ✓ 0.0s

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

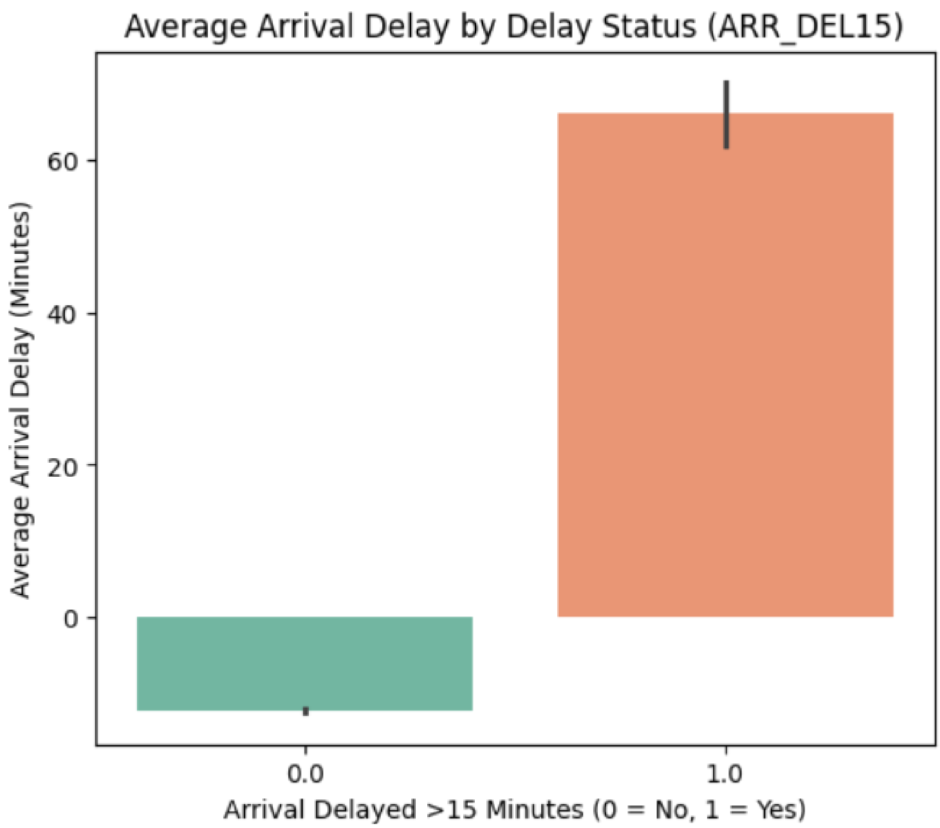
[13] ✓ 3.2s

	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

- **Data visualization:**
    - **Histograms and bar plots** to analyze distributions of delays across different days and months.

```
plt.figure(figsize=(6, 5))
sns.barplot(data=df, x='ARR_DEL15', y='ARR_DELAY', palette='Set2')
plt.title('Average Arrival Delay by Delay Status (ARR_DEL15)')
plt.xlabel('Arrival Delayed >15 Minutes (0 = No, 1 = Yes)')
plt.ylabel('Average Arrival Delay (Minutes)')
plt.show()
```

[8] ✓ 0.3s Python

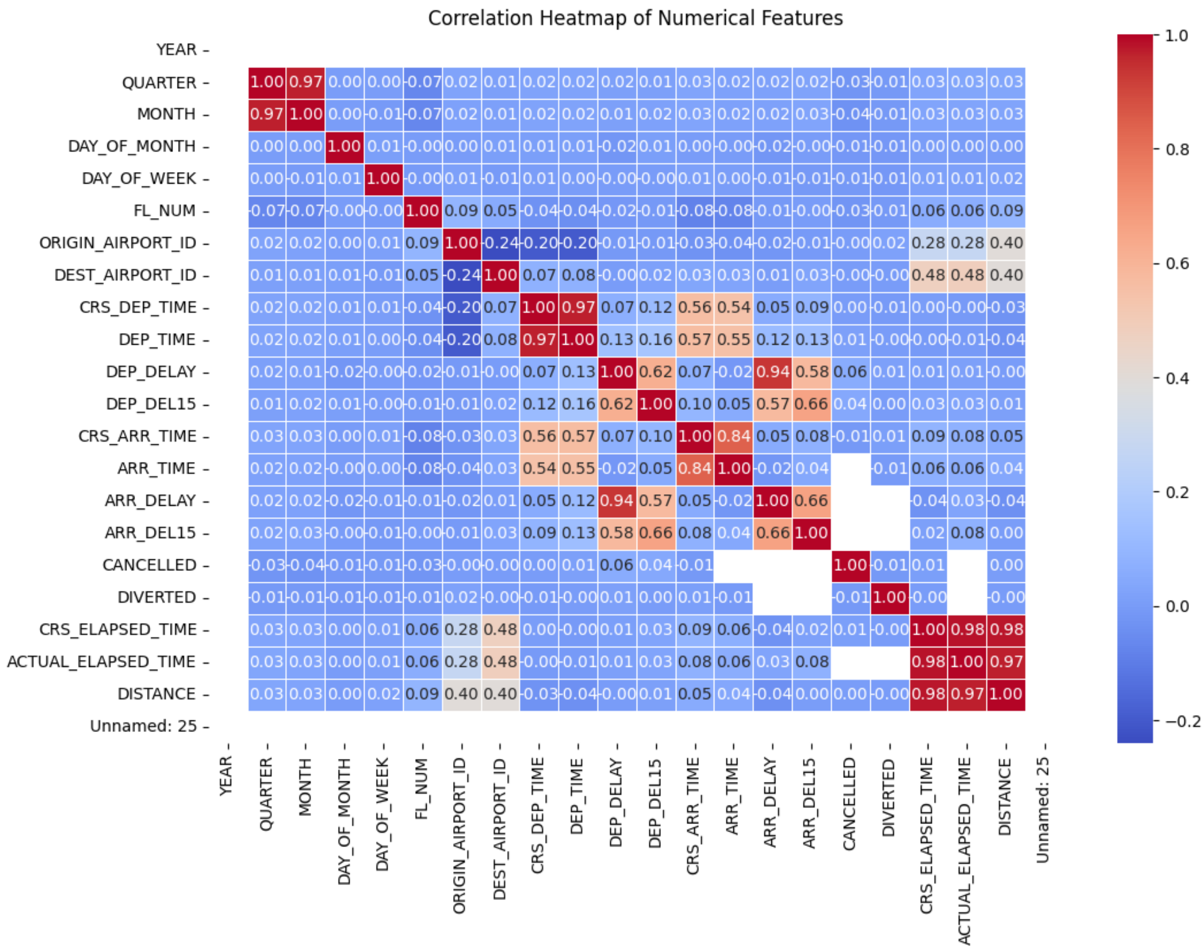

Average Arrival Delay by Delay Status (ARR_DEL15)
Average Arrival Delay (Minutes)
60
40
20
0
0.0
1.0
Arrival Delayed >15 Minutes (0 = No, 1 = Yes)

- **Heatmaps** to identify correlations between variables.

```python
corr_matrix = df.corr(numeric_only=True)

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```

[9] ✓ 1.0s Python



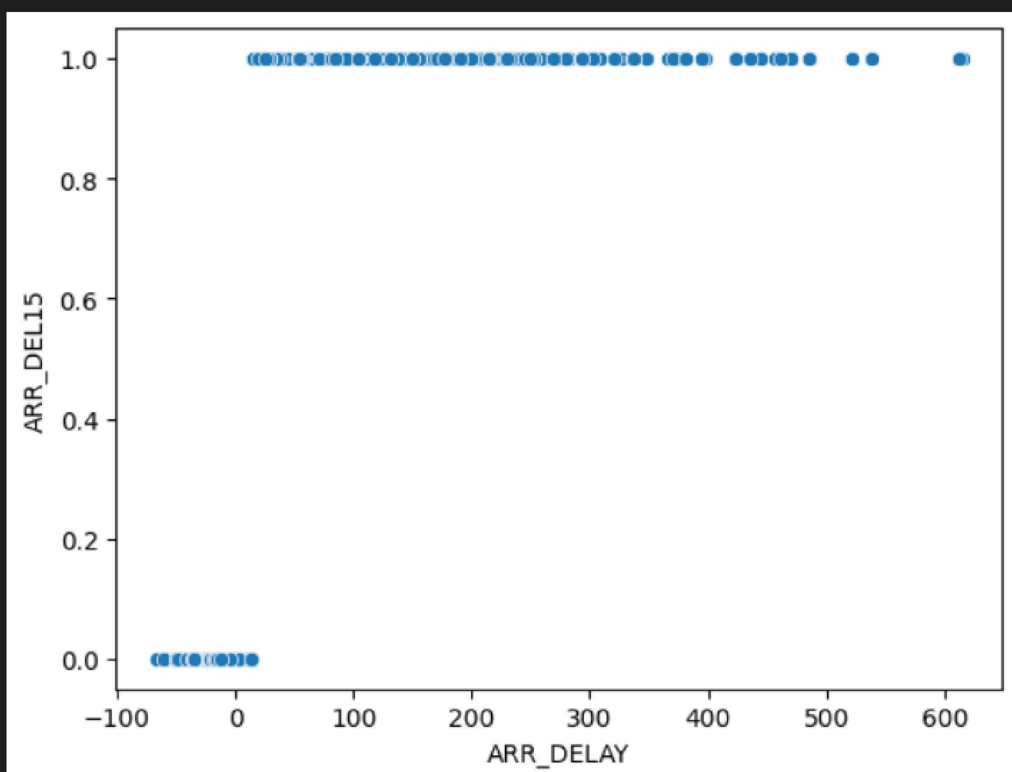
- **Scatter plots** to visualise time related trends

Visualising the Data

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

[7] ✓ 1.7s

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



Activity 3: Data Preparation

Before training the model, the data was prepared using the following techniques:

- **Encoding categorical variables:** Applied **LabelEncoding** and **OneHotEncoding** for features like Origin and Destination airports.

Label Encoding and One Hot Encoding

```
le = LabelEncoder()
df['DEST'] = le.fit_transform(df['DEST'])
df['ORIGIN'] = le.fit_transform(df['ORIGIN'])
df.head()
```

[14] ✓ 0.0s

	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

```
ohe = OneHotEncoder()
z = ohe.fit_transform(df.iloc[:, 4].values.reshape(-1, 1)).toarray()
t = ohe.fit_transform(df.iloc[:, 5].values.reshape(-1, 1)).toarray()
```

[15] ✓ 0.1s

```
z
```

[16] ✓ 0.0s

```
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.]], shape=(11231, 5))
```

```
t
```

[17] ✓ 0.0s

```
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.]], shape=(11231, 5))
```

- **Separate the Dataset into dependent and independent variables**

Splitting The Dataset Into Dependent and Independent Variables

```python
df = pd.get_dummies(df, columns=['ORIGIN', 'DEST'])
df.head()
```

[18] ✓ 0.0s Python

	FL_NUM	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_ARR_TIME	DEP_DEL15	ARR_DEL15	ORIGIN_0	ORIGIN_1	ORIGIN_2	ORIGIN_3	ORIGIN_4	DEST_0
0	1399	1	1	5	21	0.0	0.0	True	False	False	False	False	False
1	1476	1	1	5	14	0.0	0.0	False	True	False	False	False	False
2	1597	1	1	5	12	0.0	0.0	True	False	False	False	False	False
3	1768	1	1	5	13	0.0	0.0	False	False	False	False	True	False
4	1823	1	1	5	6	0.0	0.0	False	False	False	False	True	False

```python
x = df.iloc[:, 0:8].values
y = df.iloc[:, 8:9].values
```

[19] ✓ 0.0s Python

.

- **Feature scaling:** Used **StandardScaler** to normalize continuous variables like departure and arrival times.

```python
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

✓ 0.0s

- **Train-test split:** Divided the data into **training (80%)** and **testing (20%)** sets to evaluate model performance.

Splitting the dataset into train and test set

```
[20]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
✓ 0.0s
```

```
[21]
x_test.shape, x_train.shape
✓ 0.0s
... ((2247, 8), (8984, 8))
```

```
[22]
y_test.shape, y_train.shape
✓ 0.0s
... ((2247, 1), (8984, 1))
```

By collecting, cleaning, and preparing the dataset in this way, the data was made suitable for training the various machine learning algorithms such as Decision Tree, Random Forest, SVM, and LightGBM.

Milestone 3: Model Building Phase

Activity 1: Model Selection

For this project, we aimed to predict whether a flight would be delayed or on time. Given the structured, tabular nature of the dataset and the binary classification target variable (`ARR_DEL15`), we chose **supervised machine learning classification algorithms**.
We experimented with **four different models** to identify the best-performing one:

- **Decision Tree Classifier**

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```
[23] ✓ 0.0s

```
classifier = DecisionTreeClassifier()
classifier = DecisionTreeClassifier(random_state = 0)
classifier.fit(x_train, y_train)
```
[24] ✓ 0.2s

DecisionTreeClassifier

▼ Parameters

criterion	'gini'
splitter	'best'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	None
random_state	0
max_leaf_nodes	None
min_impurity_decrease	0.0
class_weight	None
ccp_alpha	0.0
monotonic_cst	None

- **Random Forest Classifier**

2. Random Forest

Generate   + Code   + Markdown

```python
rf = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=0, n_jobs=-1)
rf.fit(x_train, y_train)
```

[25] ✓ 0.4s

```
d:\Projects\Flight Delays Prediction\env\Lib\site-packages\sklearn\base.py:1363: DataConversio
  return fit_method(estimator, *args, **kwargs)
```

RandomForestClassifier

▼ Parameters

	n_estimators	100
	criterion	'gini'
	max_depth	None
	min_samples_split	2
	min_samples_leaf	1
	min_weight_fraction_leaf	0.0
	max_features	'sqrt'
	max_leaf_nodes	None
	min_impurity_decrease	0.0
	bootstrap	True
	oob_score	False
	n_jobs	-1
	random_state	0

- **Support Vector Machine (SVM)**

```
svm = SVC(kernel='linear', C=1.0, gamma='scale', random_state=0)
svm.fit(x_train, y_train)
```

[26] ✓ 1.1s

```
d:\Projects\Flight Delays Prediction\env\Lib\site-packages\sklearn\utils\
  y = column_or_1d(y, warn=True)
```

SVC

▼ Parameters

C	1.0
kernel	'linear'
degree	3
gamma	'scale'
coef0	0.0
shrinking	True
probability	False
tol	0.001
cache_size	200
class_weight	None
verbose	False
max_iter	-1
decision_function_shape	'ovr'
break_ties	False
random_state	0

- **LightGBM (Gradient Boosting Framework)**

4. LightGBM

```python
lgbm = LGBMClassifier(n_estimators=100, learning_rate=0.1, random_state=0)
lgbm.fit(x_train, y_train)
```

[42] ✓ 0.8s

```
d:\Projects\Flight Delays Prediction\env\Lib\site-packages\sklearn\preprocessing\_label.py:93: Dat
  y = column_or_1d(y, warn=True)
d:\Projects\Flight Delays Prediction\env\Lib\site-packages\sklearn\preprocessing\_label.py:129: Da
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
[LightGBM] [Info] Number of positive: 1756, number of negative: 7228
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000982 sec
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 337
[LightGBM] [Info] Number of data points in the train set: 8984, number of used features: 8
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.195459 -> initscore=-1.414924
[LightGBM] [Info] Start training from score -1.414924
```

LGBMClassifier

▼ Parameters

boosting_type	'gbdt'
num_leaves	31
max_depth	-1
learning_rate	0.1
n_estimators	100
subsample_for_bin	200000

These models were chosen due to their effectiveness in classification problems, interpretability (Decision Tree), ability to handle large feature sets (Random Forest, LightGBM), and capability to manage non-linear decision boundaries (SVM).

Activity 2: Model Training

- The dataset was split into **training (80%)** and **testing (20%)** sets to train and evaluate the models.
- We applied **feature scaling** (StandardScaler) where necessary, especially for SVM, to ensure numerical stability.
- For categorical variables (e.g., origin and destination airports), we used **One-Hot Encoding**.
- Each model was trained using the processed training data and tested on unseen test data to evaluate performance.

Activity 3: Model Evaluation

Each model was evaluated using **classification metrics**:

- **Accuracy Score**
- **Precision**
- **Recall**
- **F1-Score**

```
decisiontree = classifier.predict(x_test)
dt_pred = accuracy_score(y_test, decisiontree)
print("Decision Tree Metrics:")
print("Accuracy:", accuracy_score(y_test, decisiontree))
print("Precision:", precision_score(y_test, decisiontree))
print("Recall:", recall_score(y_test, decisiontree))
print("F1-Score:", f1_score(y_test, decisiontree))
print("\nClassification Report:\n", classification_report(y_test, decisiontree))

randomforest = rf.predict(x_test)
print("Random Forest Metrics:")
print("Accuracy:", accuracy_score(y_test, randomforest))
print("Precision:", precision_score(y_test, randomforest))
print("Recall:", recall_score(y_test, randomforest))
print("F1-Score:", f1_score(y_test, randomforest))
print("\nClassification Report:\n", classification_report(y_test, randomforest))

svmp = svm.predict(x_test)
print("SVM Metrics:")
print("Accuracy:", accuracy_score(y_test, svmp))
print("Precision:", precision_score(y_test, svmp))
print("Recall:", recall_score(y_test, svmp))
print("F1-Score:", f1_score(y_test, svmp))
print("\nClassification Report:\n", classification_report(y_test, svmp))

lgbmp = lgbm.predict(x_test)
print("LightGBM Metrics:")
print("Accuracy:", accuracy_score(y_test, lgbmp))
print("Precision:", precision_score(y_test, lgbmp))
print("Recall:", recall_score(y_test, lgbmp))
print("F1-Score:", f1_score(y_test, lgbmp))
print("\nClassification Report:\n", classification_report(y_test, lgbmp))
```

Results:

- **Decision Tree Classifier**: Accuracy – 98.93%

```
Decision Tree Metrics:
Accuracy: 0.9893190921228304
Precision: 0.9730337078651685
Recall: 0.9730337078651685
F1-Score: 0.9730337078651685

Classification Report:
              precision    recall  f1-score   support

       False       0.99      0.99      0.99      1802
        True       0.97      0.97      0.97       445

    accuracy                           0.99      2247
   macro avg       0.98      0.98      0.98      2247
weighted avg       0.99      0.99      0.99      2247
```

- **Random Forest Classifier**: Accuracy – 90.74%

```
Random Forest Metrics:
Accuracy: 0.9074321317311972
Precision: 0.927797833935018
Recall: 0.5775280898876405
F1-Score: 0.7119113573407202

Classification Report:
              precision    recall  f1-score   support

       False       0.90      0.99      0.94      1802
        True       0.93      0.58      0.71       445

    accuracy                           0.91      2247
   macro avg       0.92      0.78      0.83      2247
weighted avg       0.91      0.91      0.90      2247
```

- **SVM**: Accuracy – 80.20%

```
SVM Metrics:
Accuracy: 0.8019581664441477
Precision: 0.0
Recall: 0.0
F1-Score: 0.0

Classification Report:
              precision    recall  f1-score   support

       False       0.80      1.00      0.89      1802
        True       0.00      0.00      0.00       445

    accuracy                           0.80      2247
   macro avg       0.40      0.50      0.45      2247
weighted avg       0.64      0.80      0.71      2247
```

- **LightGBM**: Accuracy – 97.55%

```
LightGBM Metrics:
Accuracy: 0.9755229194481531
Precision: 0.9493087557603687
Recall: 0.9258426966292135
F1-Score: 0.9374288964732651

Classification Report:
              precision    recall  f1-score   support

       False       0.98      0.99      0.98      1802
        True       0.95      0.93      0.94       445

    accuracy                           0.98      2247
   macro avg       0.97      0.96      0.96      2247
weighted avg       0.98      0.98      0.98      2247
```

The **Decision Tree Classifier** performed the best, achieving **the highest accuracy (98.93%)** and balanced precision-recall scores.

Activity 4: Model Selection & Saving

- The **Decision Tree Classifier** was finalized as the best model based on evaluation metrics.

Saving the best model

Organising the models

```
models = {
    'Decision Tree': classifier,
    'Random Forest': rf,
    'SVM': svm,
    'LightGBM': lgbm
}
```

[37] ✓ 0.0s

Comparing the models

```
accuracies = {}
for model_name, model in models.items():
    y_pred = model.predict(x_test)
    acc = accuracy_score(y_test, y_pred)
    accuracies[model_name] = acc
    print(f"{model_name} Accuracy: {acc:.4f}")
```

[38] ✓ 0.3s

```
Decision Tree Accuracy: 0.9893
Random Forest Accuracy: 0.9074
SVM Accuracy: 0.8020
LightGBM Accuracy: 0.9755
```

- The model was serialized and saved using **Pickle (`flight.pkl`)** to be used later in deployment with Flask.

Select the best model

```
best_model = max(accuracies, key=accuracies.get)
best_model = models[best_model]

print(f"Best Model: {best_model}")
```

[39] ✓ 0.0s

```
Best Model: DecisionTreeClassifier(random_state=0)
```

```
with open('flight.pkl', 'wb') as f:
    pickle.dump(best_model, f)
```

[40] ✓ 0.0s

Activity 5: Justification of Chosen Model

The **Decision Tree** model was selected because:

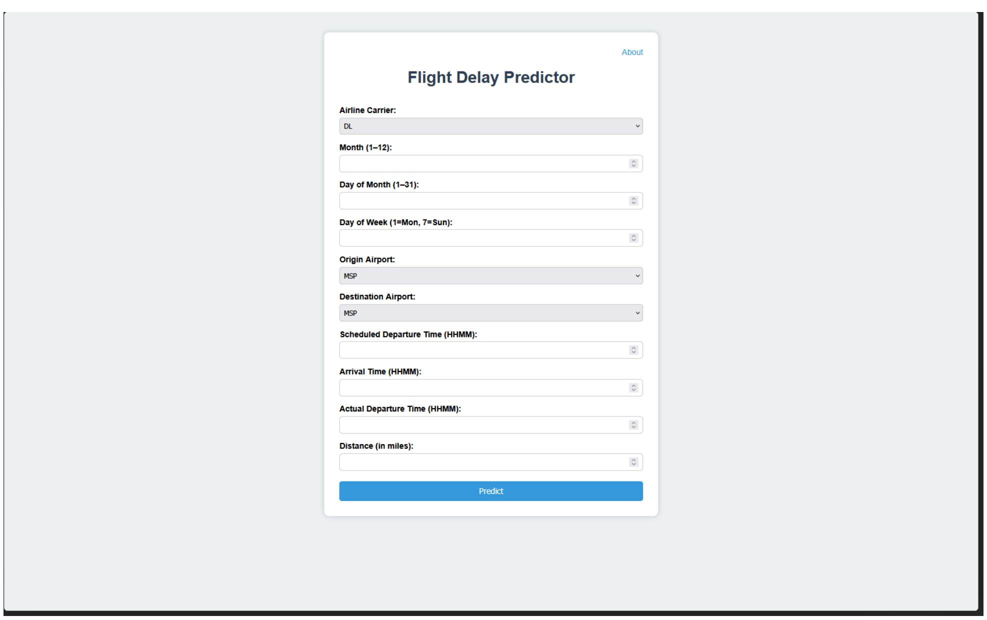
- It delivered the **highest accuracy and F1-score** compared to other models.
- It is **simple and interpretable**, making it suitable for understanding feature impact on predictions.
- It efficiently handled both numerical and categorical features in the dataset.

Milestone 4: Application Building

Activity 1: Developing the Web Interface

To make the model accessible and user-friendly, we built a **Flask web application**:

- The app provides an **HTML form** for users to input flight details such as:
  - Airline Carrier
  - Month, Day of Month, Day of Week
  - Origin and Destination Airports
  - Scheduled Departure Time, Actual Departure Time, Arrival Time



- The frontend was designed using **HTML and CSS**, and templates were stored in a `templates/` folder.

```
templates
  about.html
  index.html
```

Activity 2: Backend Integration (Flask & Model)

- We used **Flask**, a lightweight Python web framework, to create two main routes:
    - **Home Route (/)**: Displays the input form (`index.html`).

```python
@app.route('/')
def home():
    return render_template('index.html')
```

- **Predict Route (`/predict`)**: Processes form inputs, performs preprocessing (feature encoding, delay calculation), and uses the saved ML model (`flight.pkl`) to predict if the flight will be delayed or on time.

```python
@app.route('/predict', methods=['POST'])
def predict():
    month = int(request.form['month'])
    dayofmonth = int(request.form['dayofmonth'])
    dayofweek = int(request.form['dayofweek'])
    dept = int(request.form['dept'])
    arr = int(request.form['arrtime'])
    actdept = int(request.form['actdept'])
    distance = int(request.form['distance'])

    dep15 = actdept - dept

    total = [[month, dayofmonth, dayofweek, dept,
              arr, actdept, dep15, distance]]

    y_pred = model.predict(total)

    if y_pred[0] == 0:
        ans = "The flight will be on time"
    else:
        ans = "The flight will be delayed"

    return render_template('index.html', showcase=ans)
```

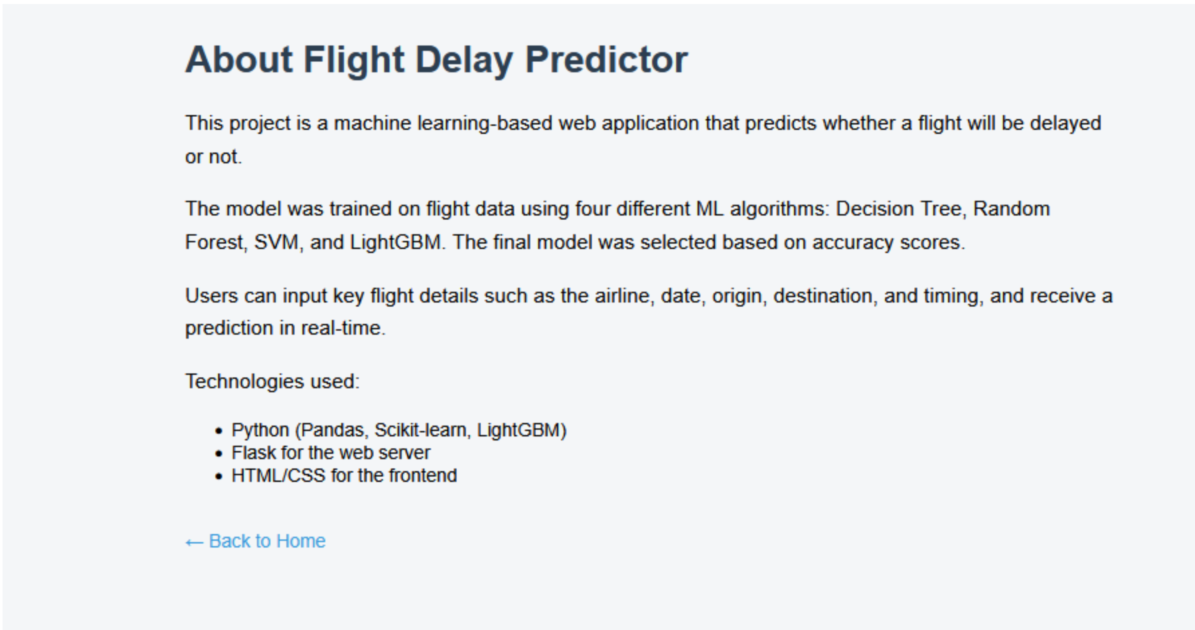
- Predictions are dynamically displayed on the same page.

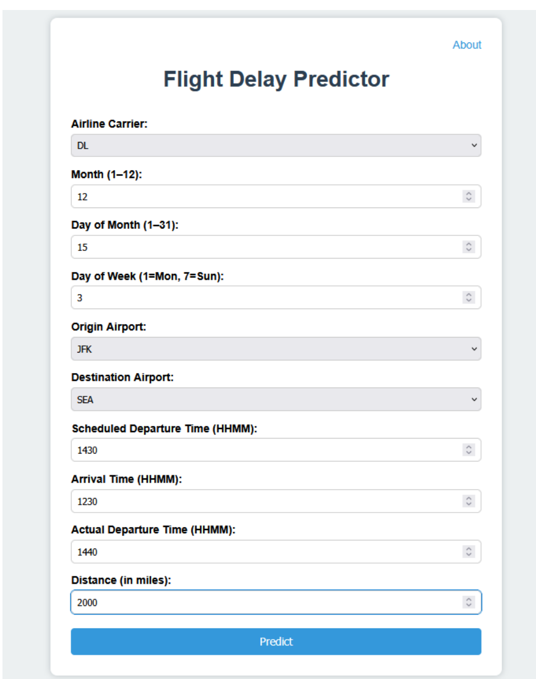
Activity 3: Model Deployment Workflow

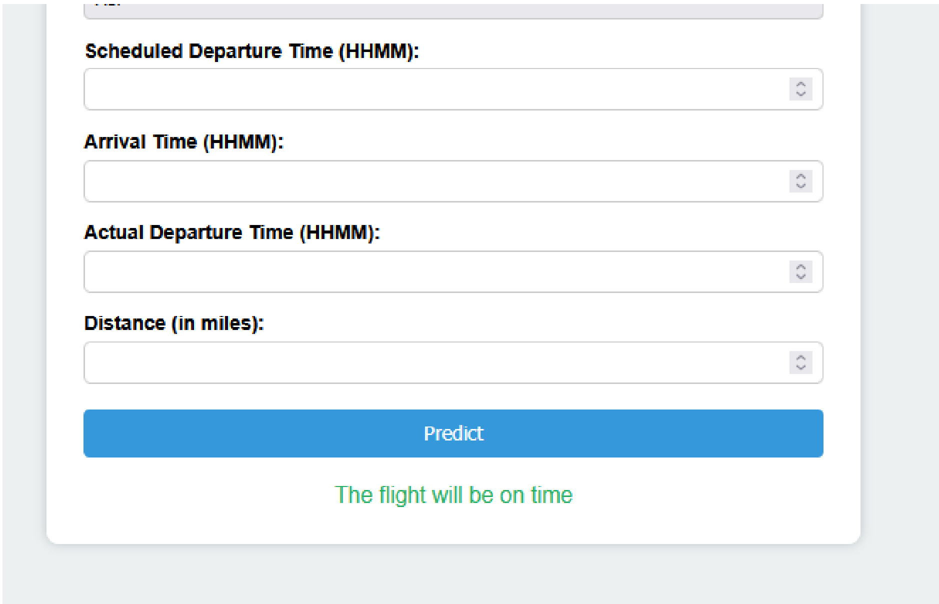
1. User submits flight details through the form.
2. Flask extracts input values and processes them to match the model's expected format.
3. The saved **Decision Tree model** predicts delay or no delay.
4. The result is displayed back to the user as:
    - **"The flight will be on time"**
    - **"The flight will be delayed"**

Activity 4: Final Output

- The final web application runs locally on Flask.
- It allows real-time delay prediction based on user inputs and displays the output clearly.






Scheduled Departure Time (HHMM):
Arrival Time (HHMM):
Actual Departure Time (HHMM):
Distance (in miles):
Predict
The flight will be on time