Flight Delays Prediction Using Machine Learning

Project Ideation :

Background and Motivation :

In today's fast-paced world, flight delays remain as a persistent issue in the aviation industry, causing inconvenience to passengers and financial losses to airlines. With the help of our website, it is possible to predict flight delays with high accuracy and such predictions can empower both passengers and airlines to make informed decisions.Our project aims to develop a machine learning-based system that predicts whether a flight is likely to be delayed, using real-time and historical flight parameters.

Primary Goal :

The main objective is to build a Flight Delay Prediction web application that allows users to input flight details and receive a delay prediction. This includes:

1. Predicting on-time or delayed status based on given inputs.
2. Providing an easy-to-use web interface built with Flask.
3. Ensuring reliable predictions using a trained machine learning model.

These requirements guided our ideation and development process in the initial phase.

Initial Approach and Challenges :

We began our project by taking given datasets related to flight records.After preprocessing and feature engineering,we used Logistic Regression,Random Forest Classifier and Decision Tree Classifier for training.

Key challenges :

1. Data Imbalance: Delayed flights were fewer than on-time flights.
2. Handling Categorical Inputs: Features like Origin and Destination needed one-hot encoding.

Target Users :

1. Frequent flyers and passengers.
2. Airline operation managers.
3. Travel agents.

Tools and Technologies :

1. Python basic libraries such as numpy,pandas,sklearn,matplotlib,seaborn.
2. Flask (for the web interface).
3. Render (for deployment).
4. HTML (for UI).
5. Pickle (for saving ML model).

Problem Statement :

Flight delays disrupt travel plans and cause passenger dissatisfaction.Current systems offer limited real-time predictive insights and there is a need for an intelligent system that can analyze flight details and predict delays effectively.Our project resolves this issue using machine learning to provide accurate flight delay predictions via a user-friendly web interface.

Requirement Analysis :

Functional Requirements :

1. The system must allow users to input flight details (like Flight number,origin, destination,date,day of week,scheduled/actual times).
2. The backend must process this input and return a prediction (On-Time or Delayed).
3. The machine learning model must be pre-trained and integrated using flight.pkl.
4. The prediction output must be shown clearly on the frontend.
5. The system must handle edge cases (e.g., same origin & destination) gracefully.

Non-Functional Requirements :

1. The application should respond within 2–3 seconds.
2. The interface should work on both desktop and mobile browsers.
3. The model and backend should handle unexpected inputs without crashing.
4. The model file (flight.pkl) should be securely stored on the server.

Tools Used :

1. Git : Version control.
2. Render : For Flask app deployment.

3. Numpy,Pandas,Scikit-learn... : For training and inference.
4. HTML : Frontend interface.
5. Jupyter Notebook : For exploratory data analysis and model training.

Code Overview :

Frontend (HTML/JS) :

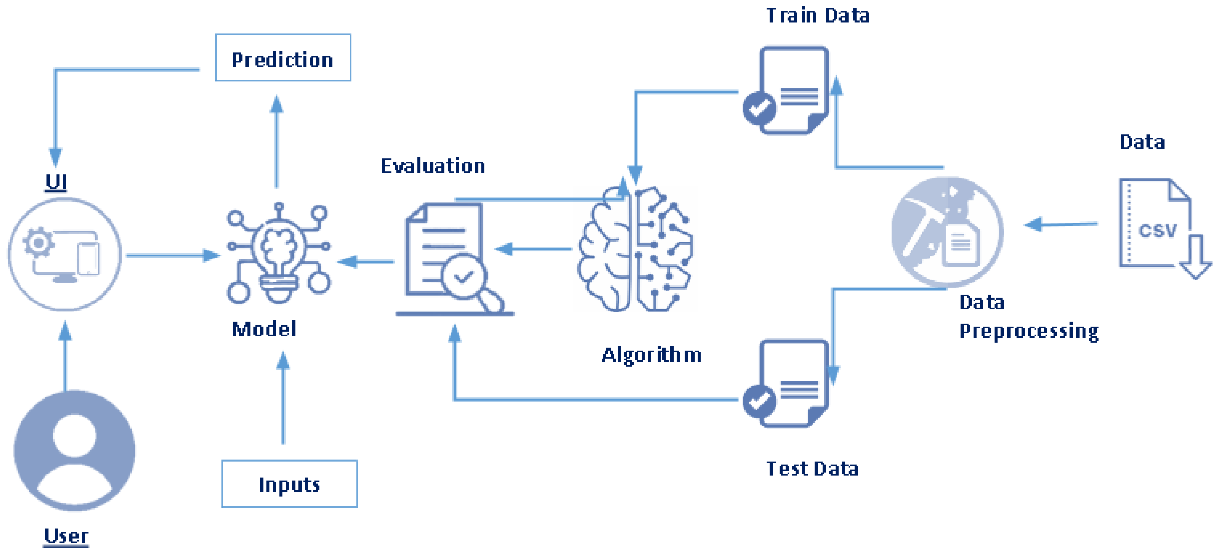
A form captures inputs like flight date,airports,times and weekday. JavaScript is used to send POST requests to the Flask API via fetch().

Backend (Python Flask) :

It receives input at predict-delay endpoint.It loads and applies the trained model (flight.pkl) and then preprocesses data using same encoding logic as during training.Finally,it returns prediction in form of "Flight is likely to be delayed" .

Project Design :

Technical Architecture :



System Architecture :

1. Frontend: We built our webpages using HTML pages as it provides a clean user interface for entering flight details such as date,origin,destination and times.
2. Input Handler(JS Logic): JavaScript validates and formats the input and then sends it to the backend via a POST request.
3. Backend(Flask App): The Python-based Flask server (app.py) receives user input, loads the trained ML model (flight.pkl), preprocesses input and predicts the delay.

4. ML Model: A trained machine learning classifier (e.g., Random Forest or Logistic Regression) predicts whether the flight will be delayed or on time.
5. Deployment: The entire application is deployed using Render(for backend) and can optionally be paired with GitHub Pages for the frontend.

We identified two use cases:

1. Passenger Use Case: A frequent flyer wants to check if a flight is likely to be delayed based on date, time, and airport.
2. Operational Use Case: Airline staff input multiple flight parameters to flag risky schedules.

UI/UX Considerations :

1. Simple form layout for entering flight details.
2. Clear labels and dropdowns for categorical inputs(e.g., Origin,Destination).
3. Prediction result shown as a card: "On Time" or "Delayed".
4. Responsive layout for both desktop and mobile viewing.
5. Real-time alert for missing or invalid inputs.

Technologies used :

1. Frontend: HTML, JavaScript
2. Backend: Python, Flask
3. Model: Scikit-learn trained classifier (e.g., Random Forest)

4. Deployment: Render(backend), GitHub Pages(frontend)
5. Storage: .pkl file for trained ML model

Project Planning :

Development Phases :

1. Requirement Gathering:
   Identified necessary inputs(e.g.,origin,destination,date,times) based on available flight datasets and user needs.

2. Data Preprocessing and Model Training:
   Cleaned and transformed data using techniques like one-hot encoding.Trained a classification model (flight.pkl) using scikit-learn.

3. Frontend UI Design:
   Developed a responsive form using HTML for users to input flight details easily.

4. Backend Development:
   Implemented a Flask server to handle form submissions,process input data,load the ML model and return predictions.

5. Testing & Validation:
   Ran multiple test cases to validate accuracy,check model response time and ensure proper error handling.

6. Deployment:
   Hosted the backend on Render.The frontend is served at GitHub Pages.

Project Timelines :

Phase	Duration	Description
Planning and Ideation	1-2 days	Defining scope and dataset

Data cleaning and modeling	2-3 days	Train and validate ML model
Frontend development	1-2 days	Build input UI using HTML
Backend Integration	1-2 days	Set up Flask and load the model
Testing	1-2 days	Test inputs and to calculate accuracy
Deployment	1 day	Hosting app on render

Team Member roles :

1. ML Engineer (V.V.Ramanji) : Handles loading a dataset,preprocessing,model training and optimization.
2. Backend Developer (T. Hema Sri) : Builds and maintains flask integration and make sure that input form is taking respnses from html page.
3. Frontend Developer(V. Yasaswi Venkat) : Designs UI for form inputs and displays prediction results.
4. Project Lead (T. Sandeep) : Coordinates all the phases,help team members and manages deployment.

References :

1. ML Basic Concepts :
    a. Supervised Learning : https://www.javatpoint.com/supervised-machine-learning
    b. Unsupervised Learning : https://www.javatpoint.com/unsupervised-machine-learning

2. ML Algorithms used :
    a. Decision trees : https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
    b. Random Forest : https://www.javatpoint.com/machine-learning-random-forest-algorithm

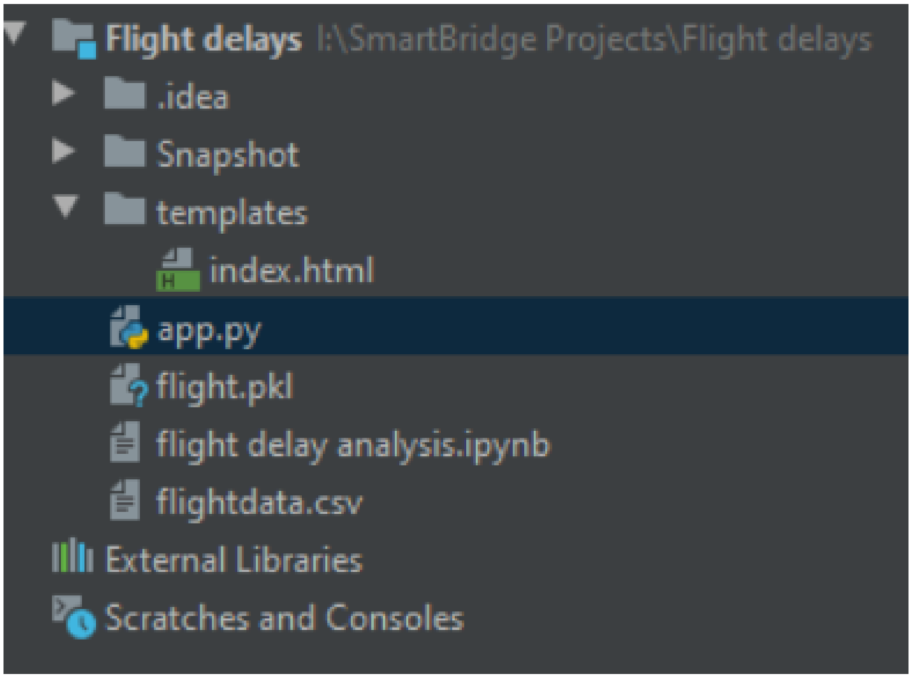
3. Model Evaluation and Metrics : https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

4. Data Preprocessing :
    a. Handling Missing data,Label Encoding,One-Hot Encoding : https://www.geeksforgeeks.org/data-preprocessing-machine-learning-python/
    b. Feature engineering : https://www.analyticsvidhya.com/blog/2020/04/feature-engineering-for-machine-learning-models/

5. Model deployment :

Render : https://render.com/docs/web-services

Project Structure :



Project Flow :

1. Defining the problem
2. Data Collection and preprocessing
3. Exploratory Data Analysis
4. Model Building
5. Performance metrics
6. Model Deployment

Data Collection and preprocessing :

1. Importing the libraries :

```
import sys
import numpy as np
import pandas as pd
import math
import pickle
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler,LabelEncoder,OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
[1]
```

2. Importing the dataset :

```
dataset=pd.read_csv("flightdata.csv")
[2]
```

3. Extracting information about datset :

```
dataset.describe()
✓ 0.0s
Python
```

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_NUM	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	CRS_DEP_TIME	DEP_TIME	...	CRS_ARR_TIME	ARR_TIME
count	11231.0	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11231.000000	11124.000000	...	11231.000000	11116.000000
mean	2016.0	2.544475	6.628973	15.790758	3.960199	1334.325617	12334.516695	12302.274508	1320.798326	1327.189410	...	1537.312795	1523.978499
std	0.0	1.090701	3.354678	8.782056	1.995257	811.875227	1595.026510	1601.988550	490.737845	500.306462	...	502.512494	512.536041
min	2016.0	1.000000	1.000000	1.000000	1.000000	7.000000	10397.000000	10397.000000	10.000000	1.000000	...	2.000000	1.000000
25%	2016.0	2.000000	4.000000	8.000000	2.000000	624.000000	10397.000000	10397.000000	905.000000	905.000000	...	1130.000000	1135.000000
50%	2016.0	3.000000	7.000000	16.000000	4.000000	1267.000000	12478.000000	12478.000000	1320.000000	1324.000000	...	1559.000000	1547.000000
75%	2016.0	3.000000	9.000000	23.000000	6.000000	2032.000000	13487.000000	13487.000000	1735.000000	1739.000000	...	1952.000000	1945.000000
max	2016.0	4.000000	12.000000	31.000000	7.000000	2853.000000	14747.000000	14747.000000	2359.000000	2400.000000	...	2359.000000	2400.000000

8 rows × 22 columns

```
dataset.info()
```

[3] ✓ 0.0s

```
<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
 18  ARR_DELAY          11043 non-null  float64
 19  ARR_DEL15          11043 non-null  float64
...
 24  DISTANCE           11231 non-null  float64
 25  Unnamed: 25        0 non-null      float64
dtypes: float64(12), int64(10), object(4)
memory usage: 2.2+ MB
```

4 . Handling missing values :

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

[10] ✓ 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
```

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

[11] ✓ 0.0s

	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

Exploratory Data Analysis :

```
dataset['DEST'].unique()
```

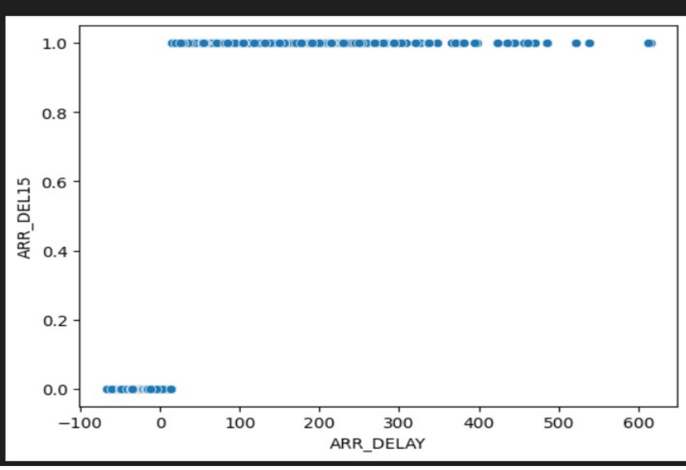
[5] ✓ 0.0s

```
array(['SEA', 'MSP', 'DTW', 'ATL', 'JFK'], dtype=object)
```

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

[6] ✓ 0.3s

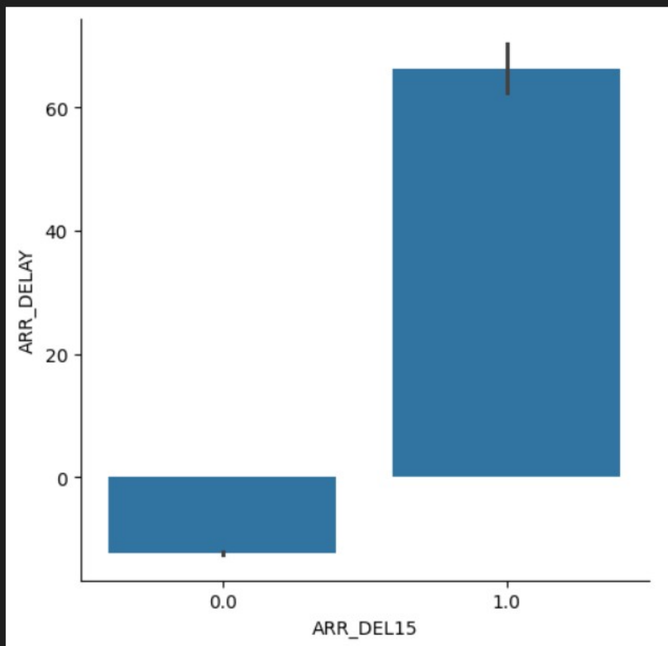
```
<Axes: xlabel='ARR_DELAY', ylabel='ARR_DEL15'>
```

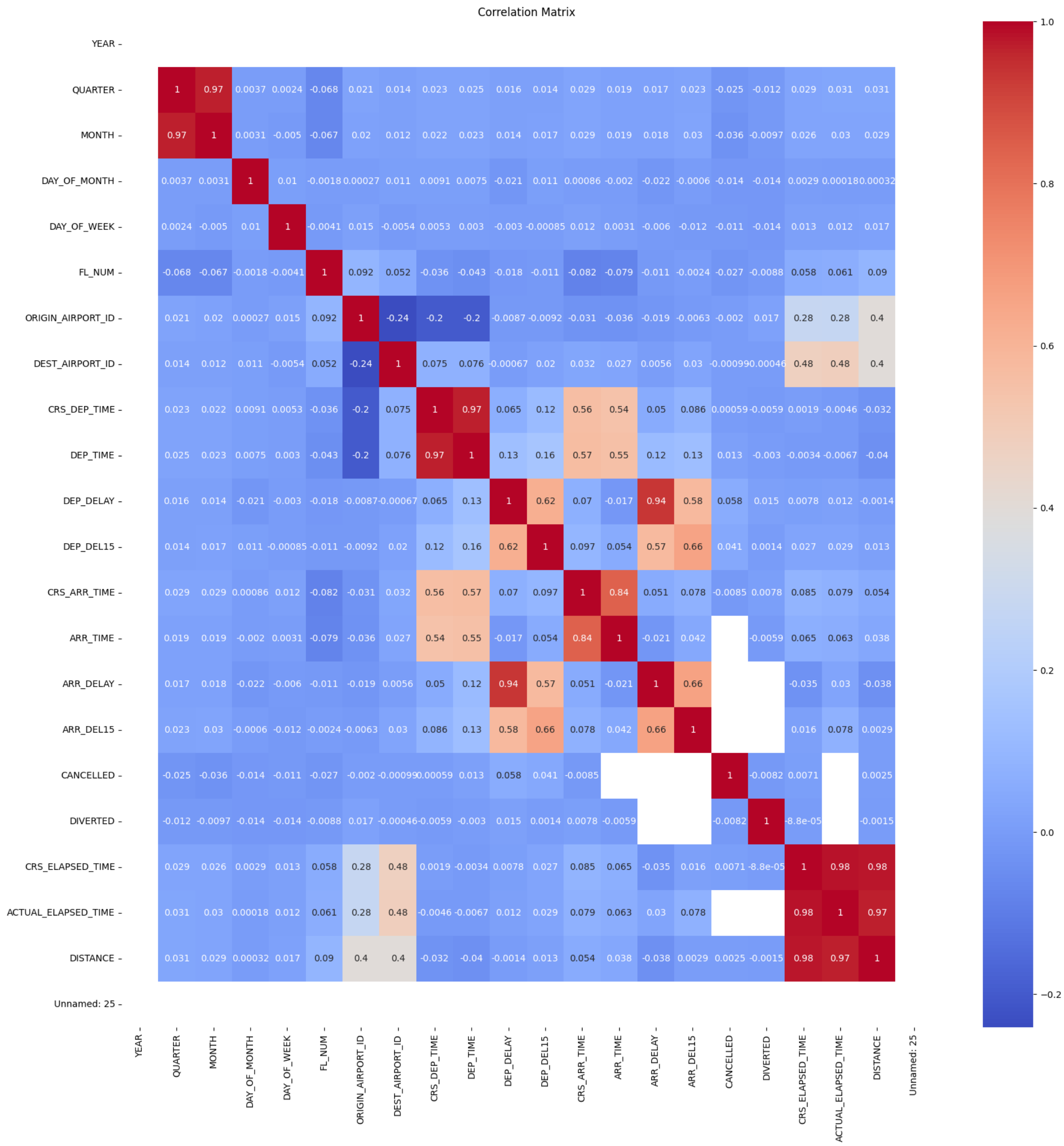


```
sns.catplot(x='ARR_DEL15',y='ARR_DELAY',kind='bar',data=dataset)
```

[7] ✓ 0.3s

```
<seaborn.axisgrid.FacetGrid at 0x1b838b9e3c0>
```




Correlation Matrix
YEAR
QUARTER
MONTH
DAY_OF_MONTH
DAY_OF_WEEK
FL_NUM
ORIGIN_AIRPORT_ID
DEST_AIRPORT_ID
CRS_DEP_TIME
DEP_TIME
DEP_DELAY
DEP_DEL15
CRS_ARR_TIME
ARR_TIME
ARR_DELAY
ARR_DEL15
CANCELLED
DIVERTED
CRS_ELAPSED_TIME
ACTUAL_ELAPSED_TIME
DISTANCE
Unnamed: 25
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Training and Testing the model :

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

[19]  x_test.shape
      ✓ 0.0s
...   (2247, 8)

[20]  x_train.shape
      ✓ 0.0s
...   (8984, 8)

[21]  y_test.shape
      ✓ 0.0s
...   (2247, 1)

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

Model Deployment :

1. Saving the ML model by using pickle :

```
...
```

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

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

[25] ✓ 0.0s

```
y_pred=classifier.predict(x_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
```

[26] ✓ 0.0s

```
... Accuracy: 0.8593680462839342
```

2. Integrate with Flask :

```python
from flask import Flask,request,render_template
import numpy as np
import pandas as pd
import pickle
import os

model=pickle.load(open('flight.pkl','rb'))
app=Flask(__name__)

@app.route('/')
def home():
    return render_template("index.html")
@app.route('/prediction',methods=['POST'])
```

3. For retrieving values from UI :

```python
def predict():
    try:
        name=int(request.form['name'])
        month=int(request.form['month'])
        dayofmonth=int(request.form['dayofmonth'])
        dayofweek=int(request.form['dayofweek'])
        origin=request.form['origin'].upper()
        destination=request.form['destination'].upper()
        dept=int(request.form['dept'])
        arrtime=int(request.form['arrtime'])
        actdept=int(request.form['actdept'])
        if (dept-actdept)>15:
            dept15=1
        else:
            dept15=0
        origin_map={'ATL':0,'DTW':1,'JFK':2,'MSP':3,'SEA': 4}
        dest_map={'ATL':0,'DTW':1,'JFK':2,'MSP':3,'SEA': 4}
        origin_encoded=origin_map.get(origin, 0)
        dest_encoded=dest_map.get(destination, 0)
        features=[name,month,dayofmonth,dayofweek,origin_encoded,dest_encoded,arrtime,dept15]

        input_df=pd.DataFrame([features],columns=['FL_NUM','MONTH','DAY_OF_MONTH','DAY_OF_WEEK','ORIGIN','DEST','CRS_ARR_TIME','DEP_DEL15'])
        y_pred=model.predict(input_df)
        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)
    except Exception as e:
        return f"Error in prediction:{e}"
```

4. Main Function :

```
if __name__=='__main__':
    port=int(os.environ.get('PORT',5000))
    app.run(host='0.0.0.0',port=port)
```