PREDICTIVE ANALYSIS OF FLIGHT PUNCTUALITY USING MACHINE LEARNING

A PROJECT REPORT

Submitted By

SANDHYA SHANKAR

submitted to the Faculty of

INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY

COLLEGE OF ENGINEERING, GUINDY

ANNA UNIVERSITY

CHENNAI 600 025

AUGUST 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report on "**Predictive Analysis On Flight Punctuality**" is the bonafide work of "**SANDHYA SHANKAR"** who carried out the project under my supervision.

SIGNATURE

Dr. T.J.VIJAY KUMAR
ASSISTANT PROFESSOR

Department of Information Science and Technology, CEG Campus, Anna University, Chennai - 25

ANNA UNIVERSITY: CHENNAI 600 025

COMPLETION CERTIFICATE

Certified that this project report on "**Predictive Analysis On Flight Punctuality**" is the bonafide work of "**SANDHYA SHANKAR"** who carried out the project under my supervision.

SIGNATURE

Dr. T.J.VIJAY KUMAR
ASSISTANT PROFESSOR

Department of Information Science and Technology, CEG Campus, Anna University, Chennai – 25

TABLE OF CONTENTS

ABSTRACT

In this project, we present a cutting-edge solution for predicting on-time flight performance using advanced machine learning techniques. Utilizing basic machine learning algorithms, we aim to develop a highly accurate model capable of forecasting flight punctuality. By analyzing historical flight data and employing powerful predictive algorithms, we intend to create a robust system that can effectively predict delays and improve overall flight scheduling.

The primary objective of this project is to explore various machine learning algorithms, evaluate their performance, and develop a reliable predictor to enhance the efficiency of flight operations. This research contributes significantly to the field of aviation by providing a data-driven approach to optimize flight schedules and enhance passenger experiences.

OBJECTIVE

The primary objective of the "**Predictive Analysis On Flight Punctuality**" project is to create a reliable system for predicting flight punctuality based on historical data. This involves:

1. **Model Development :** Select and refine machine Learning algorithms, including tested linear regression classifiers, to sort flights into on-time categories. We'll tweak different aspects to make sure our predictions are accurate.

2. **Data Preparation:** Gather and clean flight data, considering various factors like weather and airport conditions. We'll organize this data properly to make it useful for training our prediction models.

3. **Training and Evaluation:** Train the chosen algorithms with our prepared data and rigorously assess their accuracy. We'll compare different models to find the one that works best for predicting flight punctuality.

4. **Creating a User-Friendly Tool:** Build an easy-to-use interface that airlines, airports, and passengers can access. This tool will provide real-time predictions about flight punctuality, helping people plan their travels more effectively.

By completing these steps, the project aims to provide a practical solution for improving flight schedules, enhancing passenger experiences, and increasing overall efficiency in the aviation industry.

INTRODUCTION

In today's fast-paced world, accurate and timely flight information is crucial for travelers, airlines, and airport authorities. Flight punctuality, influenced various factors such as weather conditions, is a significant concern for the aviation industry. To address this challenge, our project delves into the realms of Machine Learning (ML) and Web Development to create a robust solution.

The primary objective of our project is to predict flight punctuality by harnessing historical data and real-time weather information. By employing machine learning algorithms, specifically tested linear regression classifiers, we categorize flights into groups based on their on-time performance. To enhance the accuracy of our predictions, we integrate weather details obtained through APIs into our dataset. Additionally, we utilize the Flask framework to seamlessly integrate our machine learning model with an intuitive user interface.

The significance of our On-Time Flight Performance Predictor lies in its ability to revolutionize the way stakeholders in the aviation industry make decisions. Airlines can optimize schedules, airports can streamline operations, and passengers can plan their travels more efficiently. By providing real-time insights into flight punctuality, our system enables proactive decision-making, enhancing the overall travel experience.

Project Components:

1. Data Integration: weather data and flight information retrieved through pre-existing data is seamlessly integrated into our dataset, enriching the information available for prediction.

2. Machine Learning Algorithms: Tested linear regression classifiers are employed to categorize flights into on-time performance groups, allowing for accurate predictions.

3. Web Development: Flask, a versatile web framework, is utilized to create a user-friendly interface. This interface allows stakeholders to access real-time flight punctuality predictions conveniently.

4. APIs: We enhance prediction accuracy by seamlessly integrating live weather data via APIs. This real-time data ensures our forecasts align with actual conditions, providing precise and relevant insights for specific days.

After the dataset (consisting of pre-existing data) is trained into a suitable Machine Learning Model, a user can now feed their flight information into the model, the API integration in this project will make sure to extract real-time weather information in that particular city feed it into the model at the same time. The model will then produce a realistic time frame delay to the user.

The steps to achieve this project follow the methodology of a Machine Learning pipeline which includes data collection, data preprocessing, feature selection, data splitting, model development, model training, error analysis and model comparison, which have been elaborated in detail in the following sections of this report.

SOFTWARE REQUIREMENTS

This project relies on a comprehensive set of software components, each serving a crucial role in the development and deployment of the On-Time Flight Performance Predictor. Here's a breakdown of the essential software requirements:

1. Python:

- Description: The project is implemented using Python, serving as the core programming language for the entire system.
- Purpose: Python facilitates data processing, model development, web integration, and overall system functionality.

2. Flask Framework:

- Description: Flask is a lightweight web framework in Python used for building the web application's backend.
- Purpose: Flask enables seamless integration of HTML, CSS, APIs, and machine learning models, creating an interactive user interface.

3. pandas:

- Description: pandas is a powerful data manipulation library in Python.
- Purpose: Utilized for data preprocessing tasks, including cleaning, organizing, and transforming the flight dataset, ensuring it

is model-ready.

4. NumPy:

- Description: NumPy is a fundamental package for scientific computing with Python.

- Purpose: Handles arrays and data manipulation, playing a key role in processing and organizing data for machine learning algorithms.

5. scikit-learn:

- Description: scikit-learn is a versatile machine learning library in Python.

- Purpose: Employed for splitting the dataset into training and testing sets, standardizing features, and implementing machine learning models for flight punctuality prediction.

6. requests:

- Description: The requests library enables HTTP requests to retrieve real-time weather data from APIs.

- Purpose: Used to fetch live weather information, enhancing the prediction model's accuracy by integrating current atmospheric conditions.

7. pickle:

- Description: pickle is a Python library for object serialization and deserialization.

- Purpose: Facilitates the saving and loading of trained machine

learning models, ensuring persistence between sessions and deployments.

These software components collectively form the backbone of the On-Time Flight Performance Predictor, enabling data processing, model training, real-time weather integration, and interactive user experience. Their seamless integration empowers the system to deliver accurate and actionable flight punctuality predictions.

MACHINE LEARNING ARCHITECTURE (PIPELINE)

Machine learning is the subset of Artificial Intelligence that focuses on the development of algorithms and statistical models that enable computer systems to improve their performance. This is done by training them to learn using historical data and reinforcement, without being explicitly programmed to do so. Essentially, the data is given to a machine learning system, and it makes predictions or decisions based on the learned information. There are various categories of machine learning, or rather various algorithms used to train a machine to do what the user requires it to do.

These are as follows:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

The Flight On-Time Prediction project harnesses the power of a supervised learning model, aligning with the fundamental principles of machine learning. In this supervised setting, the model is trained on meticulously labeled datasets, where inputs are intricately paired with the corresponding correct outputs. Within the realm of supervised learning, this project falls under the domain of regression, a sophisticated term denoting the predictive nature of the model. The

supervised learning approach allows the algorithm to learn from historical flight data, mapping intricate patterns and relationships between various parameters to predict flight punctuality accurately.

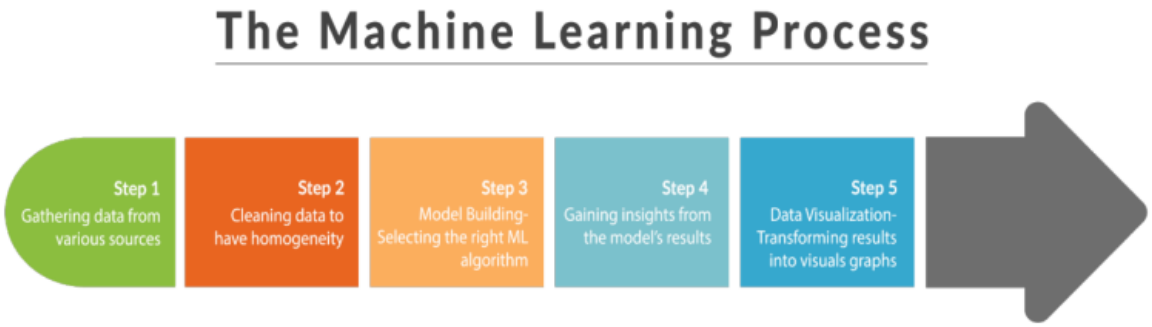

Figure 1 - Machine Learning pipeline

PROBLEM STATEMENT

To predict flight delays accurately by analyzing historical flight data, weather conditions, and relevant factors, enhancing operational efficiency and passenger experiences in the aviation industry.

DATA COLLECTION

For this project, we receive data in the form of two parts :

- Flight data
- Weather data

The data is sourced from an independent research academy – "Bright Academy – Chennai" which has generously helped in providing information collected for 15 different airports in the USA over 10 years, but for the sake of this project we have narrowed it down to the years 2016 and 2017 alone.

The flight data is split into two folders, these two folders have 12 sub-folders, each designated to every month of that year.

- 2016_original
    1. On_Time_On_Time_Performance_2016_1
    2. On_Time_On_Time_Performance_2016_2
    3. On_Time_On_Time_Performance_2016_3
    4. On_Time_On_Time_Performance_2016_4

5. On_Time_On_Time_Performance_2016_5
6. On_Time_On_Time_Performance_2016_6
7. On_Time_On_Time_Performance_2016_7
8. On_Time_On_Time_Performance_2016_8
9. On_Time_On_Time_Performance_2016_9
10. On_Time_On_Time_Performance_2016_10
11. On_Time_On_Time_Performance_2016_11
12. On_Time_On_Time_Performance_2016_12

- 2017_original
    1. On_Time_On_Time_Performance_2017_1
    2. On_Time_On_Time_Performance_2017_2
    3. On_Time_On_Time_Performance_2017_3
    4. On_Time_On_Time_Performance_2017_4
    5. On_Time_On_Time_Performance_2017_5
    6. On_Time_On_Time_Performance_2017_6
    7. On_Time_On_Time_Performance_2017_7
    8. On_Time_On_Time_Performance_2017_8
    9. On_Time_On_Time_Performance_2017_9
    10. On_Time_On_Time_Performance_2017_10
    11. On_Time_On_Time_Performance_2017_11
    12. On_Time_On_Time_Performance_2017_12

Each of these folders consist of a csv file as well as a “readme.html” Which contain various flight detail parameters as listed below in fig 1.1. 1.2 and 1.3

BACKGROUND

The data contained in the compressed file has been extracted from the On-Time Performance data table of the "On-Time" database from the TranStats data library. The time period is indicated in the name of the compressed file; for example, XXX_XXXXX_2001_1 contains data of the first month of the year 2001.

RECORD LAYOUT

Below are fields in the order that they appear on the records:

Year	Year
Quarter	Quarter (1-4)
Month	Month
DayofMonth	Day of Month
DayOfWeek	Day of Week
FlightDate	Flight Date (yyyymmdd)
UniqueCarrier	Unique Carrier Code. When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2). Use this field for analysis across a range of years.
AirlineID	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its Code, Name, or holding company/corporation.
Carrier	Code assigned by IATA and commonly used to identify a carrier. As the same code may have been assigned to different carriers over time, the code is not always unique. For analysis, use the Unique Carrier Code.
TailNum	Tail Number
FlightNum	Flight Number
OriginAirportID	Origin Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.
OriginAirportSeqID	Origin Airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.
OriginCityMarketID	Origin Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.
Origin	Origin Airport
OriginCityName	Origin Airport, City Name
OriginState	Origin Airport, State Code
OriginStateFips	Origin Airport, State Fips
OriginStateName	Origin Airport, State Name
OriginWac	Origin Airport, World Area Code
DestAirportID	Destination Airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.

Fig 1.1

DestCityMarketID	Destination Airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.
Dest	Destination Airport
DestCityName	Destination Airport, City Name
DestState	Destination Airport, State Code
DestStateFips	Destination Airport, State Fips
DestStateName	Destination Airport, State Name
DestWac	Destination Airport, World Area Code
CRSDepTime	CRS Departure Time (local time: hhmm)
DepTime	Actual Departure Time (local time: hhmm)
DepDelay	Difference in minutes between scheduled and actual departure time. Early departures show negative numbers.
DepDelayMinutes	Difference in minutes between scheduled and actual departure time. Early departures set to 0.
DepDel15	Departure Delay Indicator, 15 Minutes or More (1=Yes)
DepartureDelayGroups	Departure Delay intervals, every (15 minutes from <-15 to >180)
DepTimeBlk	CRS Departure Time Block, Hourly Intervals
TaxiOut	Taxi Out Time, in Minutes
WheelsOff	Wheels Off Time (local time: hhmm)
WheelsOn	Wheels On Time (local time: hhmm)
TaxiIn	Taxi In Time, in Minutes
CRSArrTime	CRS Arrival Time (local time: hhmm)
ArrTime	Actual Arrival Time (local time: hhmm)
ArrDelay	Difference in minutes between scheduled and actual arrival time. Early arrivals show negative numbers.
ArrDelayMinutes	Difference in minutes between scheduled and actual arrival time. Early arrivals set to 0.
ArrDel15	Arrival Delay Indicator, 15 Minutes or More (1=Yes)
ArrivalDelayGroups	Arrival Delay intervals, every (15-minutes from <-15 to >180)
ArrTimeBlk	CRS Arrival Time Block, Hourly Intervals
Cancelled	Cancelled Flight Indicator (1=Yes)
CancellationCode	Specifies The Reason For Cancellation
Diverted	Diverted Flight Indicator (1=Yes)
CRSElapsedTime	CRS Elapsed Time of Flight, in Minutes
ActualElapsedTime	Elapsed Time of Flight, in Minutes
AirTime	Flight Time, in Minutes
Flights	Number of Flights
Distance	Distance between airports (miles)

Fig 1.2

Div1TailNum	Aircraft Tail Number for Diverted Airport Code1
Div2Airport	Diverted Airport Code2
Div2AirportID	Airport ID of Diverted Airport 2. Airport ID is a Unique Key for an Airport
Div2AirportSeqID	Airport Sequence ID of Diverted Airport 2. Unique Key for Time Specific Information for an Airport
Div2WheelsOn	Wheels On Time (local time: hhmm) at Diverted Airport Code2
Div2TotalGTime	Total Ground Time Away from Gate at Diverted Airport Code2
Div2LongestGTime	Longest Ground Time Away from Gate at Diverted Airport Code2
Div2WheelsOff	Wheels Off Time (local time: hhmm) at Diverted Airport Code2
Div2TailNum	Aircraft Tail Number for Diverted Airport Code2
Div3Airport	Diverted Airport Code3
Div3AirportID	Airport ID of Diverted Airport 3. Airport ID is a Unique Key for an Airport
Div3AirportSeqID	Airport Sequence ID of Diverted Airport 3. Unique Key for Time Specific Information for an Airport
Div3WheelsOn	Wheels On Time (local time: hhmm) at Diverted Airport Code3
Div3TotalGTime	Total Ground Time Away from Gate at Diverted Airport Code3
Div3LongestGTime	Longest Ground Time Away from Gate at Diverted Airport Code3
Div3WheelsOff	Wheels Off Time (local time: hhmm) at Diverted Airport Code3
Div3TailNum	Aircraft Tail Number for Diverted Airport Code3
Div4Airport	Diverted Airport Code4
Div4AirportID	Airport ID of Diverted Airport 4. Airport ID is a Unique Key for an Airport
Div4AirportSeqID	Airport Sequence ID of Diverted Airport 4. Unique Key for Time Specific Information for an Airport
Div4WheelsOn	Wheels On Time (local time: hhmm) at Diverted Airport Code4
Div4TotalGTime	Total Ground Time Away from Gate at Diverted Airport Code4
Div4LongestGTime	Longest Ground Time Away from Gate at Diverted Airport Code4
Div4WheelsOff	Wheels Off Time (local time: hhmm) at Diverted Airport Code4
Div4TailNum	Aircraft Tail Number for Diverted Airport Code4
Div5Airport	Diverted Airport Code5
Div5AirportID	Airport ID of Diverted Airport 5. Airport ID is a Unique Key for an Airport
Div5AirportSeqID	Airport Sequence ID of Diverted Airport 5. Unique Key for Time Specific Information for an Airport
Div5WheelsOn	Wheels On Time (local time: hhmm) at Diverted Airport Code5
Div5TotalGTime	Total Ground Time Away from Gate at Diverted Airport Code5
Div5LongestGTime	Longest Ground Time Away from Gate at Diverted Airport Code5
Div5WheelsOff	Wheels Off Time (local time: hhmm) at Diverted Airport Code5
Div5TailNum	Aircraft Tail Number for Diverted Airport Code5

Fig 1.3

Likewise the weather data is stored in 15 different files for 15 different cities that we have considered for this project. The cities have the following airport codes:

ATL	CLT	DEN	DFW	EWR
IAH	JFK	LAS	LAX	MCO
MIA	ORD	PHX	SEA	SFO

Each city has a folder consisting of 24 json files for each month of the years 2016 and 2017.

For example:

- MIA

1. 2016-1.json
2. 2016-2.json
3. 2016-3.json
4. 2016-4.json
5. 2016-5.json
6. 2016-6.json
13. 2017-1.json
14. 2017-2.json
15. 2017-3.json
16. 2017-4.json
17. 2017-5.json
18. 2017-6.json

7. 2016-7.json
8. 2016-8.json
9. 2016-9.json
10. 2016-10.json
11. 2016-11.json
12. 2016-12.json
19. 2017-7.json
20. 2017-8.json
21. 2017-9.json
22. 2017-10.json
23. 2017-11.json
24. 2017-12.json

Each json file has various weather parameters, and after careful consideration, the following parameters have been shortlisted for maximum efficiency.

Recommended Weather Columns:

WindSpeedKmph	WindDirDegree	WeatherCode	precipMM
Visibilty	Pressure	Cloudcover	DewPointF
WindGustKmph	tempF	WindChillF	Humidity
date	time	airport	

DATA PRE-PROCESSING

Data preprocessing refers to the cleaning, transforming, and integrating of data in order to make it ready for analysis. The goal of data preprocessing is to improve the quality of the data and to make it more suitable for the specific task.

- Date Collection and Consolidation:

Raw flight data for 2016 and 2017 is gathered from separate CSV files, with each file representing a month. The file paths are stored in lists ‘m’ and ‘l’ for the respective years.

m = [...] # List of file paths for 2016 data

l = [...] # List of file paths for 2017 data

- Data Filtering And Selection:

Relevant columns are chosen, and data is filtered and stored as a pandas dataframe based on specific criteria, including airport codes (origin and destination) and non-null departure times. This step ensures that only necessary data is included for analysis. The relevant columns are listed as below:

'FlightDate', "Quarter", "Origin", "Year", "Month", "DayofMonth", "DepTime", "DepDel15", "CRSDepTime", "DepDelayMinutes", "OriginAirportID", "DestAirportID", "ArrTime", "CRSArrTime", "ArrDel15", "ArrDelayMinutes"

columns = [...] # List of relevant column names

df1 = pd.read_csv(l[0])

df1 = df1[...] # Select and filter relevant columns

- Data Concatenation:

Data from both years is combined into a single DataFrame (`df1`) using the `pd.concat()` function. This step consolidates the information for comprehensive analysis and model training.

for i in range(1, len(l)):

df2 = pd.read_csv(l[i])

df2 = df2[...] # Select and filter relevant columns

df1 = pd.concat([df1, df2], ignore_index=True)

 # Concatenate dataframes

- Weather Data Integration:

Real-time weather data for selected airports and months is collected from JSON files and organized into a dictionary (`all_weather_data`). This data is essential for understanding the weather conditions during flight times.

all_weather_data = {...} # Dictionary with airport codes as keys and weather data as values

In this particular part of the code, as it was incredibly hard to map

the values of parameters in such a cluttered json file, I used the help of an **online json viewer** [Online JSON Viewer and Formatter (stack.hu)]

This greatly helped in the formatting of the json file for easy mapping of values for data abstraction.

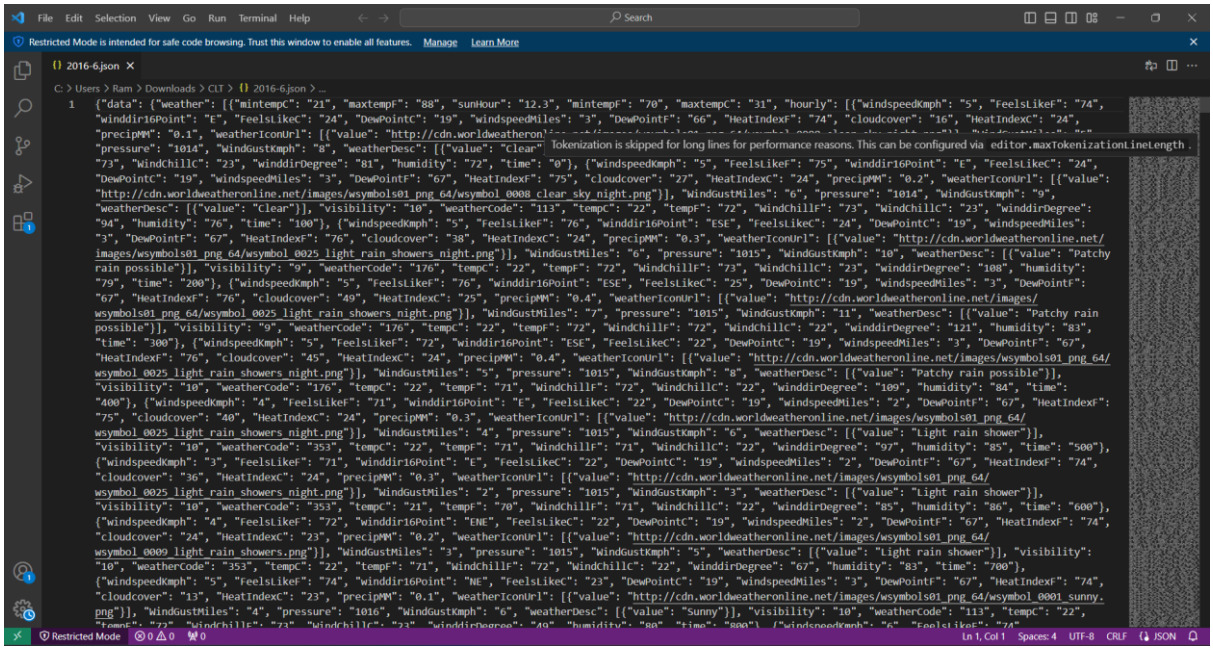

Fig 2.0 : Weather data before using json formatter

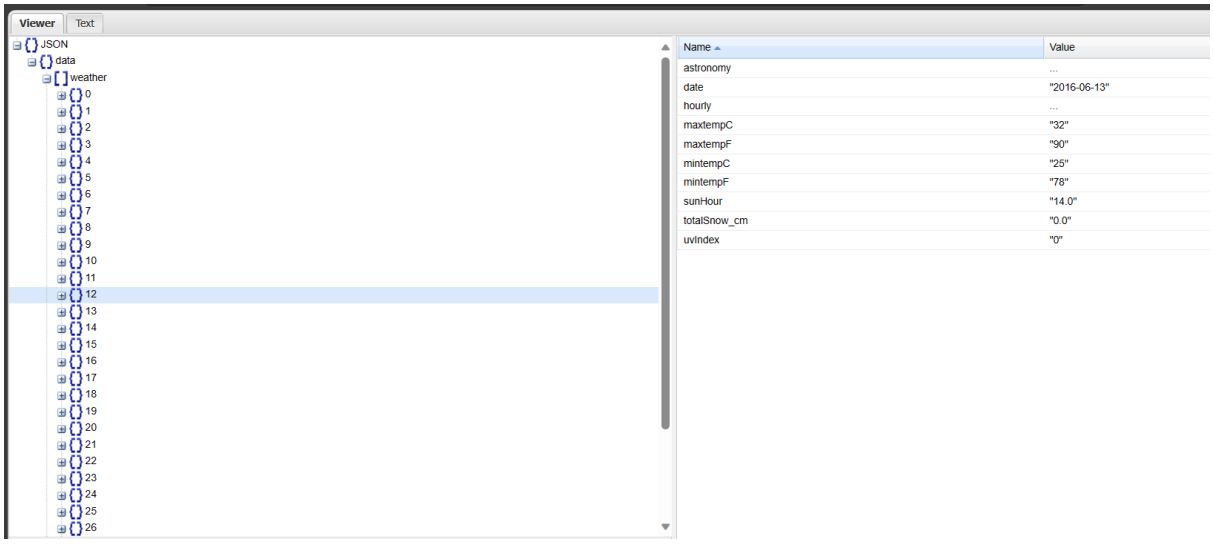

Fig 2.1: Weather data after using json formatter

- Data Alignment with Weather Information:

Flight data is aligned with the corresponding weather data based on airport code, year, month, day, and departure time. This alignment

ensures that each flight record is associated with accurate weather details.

```
for index, row in df1.iterrows():
# Extract necessary indices and align flight data with weather data
    origin_index = ...
    dep_month_index = ...
    dep_day_index = ...
    dep_time_index = ...
    flight_weather = all_weather_data[...][...][...]['hourly'][...]
    flight_weather_data.append(flight_weather)
```

- Feature Engineering:

Various weather parameters (such as windspeed, visibility, and temperature) are extracted from the aligned weather data. These parameters are added as new columns in the DataFrame to enrich the dataset for model training and analysis.

```
windspeedKmphList = [...]  # Extracted windspeed values
visibilityList = [...]  # Extracted visibility values
...
df1['windspeedKmph'] = windspeedKmphList
# Add windspeed as a new column
df1['visibility'] = visibilityList
# Add visibility as a new column
...
```

- Data Export:

The preprocessed and enriched data is saved to a CSV file (`preprocessed_data.csv`) for further exploratory data analysis and machine learning model development.

df1.to_csv('preprocessed_data.csv') # Save the preprocessed data to a CSV file

Note: The provided code snippets are illustrative and are merely a structure of the actual code.

MODEL TRAINING

In machine learning, datasets are typically divided into two subsets: the **training set** and the **testing set**.

Training Set:

- The training set is a subset of the dataset that is used to train the machine learning model.
- It consists of input features (independent variables) and corresponding target values (labels or dependent variables).
- The model learns patterns, relationships, and representations from this data.
- During the training process, the model adjusts its parameters to minimize the difference between its predictions and the actual target values in the training set.
- A larger and diverse training set helps the model learn a wide variety of patterns, improving its ability to generalize to new, unseen data.

Testing Set:

- The testing set is a separate subset of the dataset that the trained model has never seen before.
- It contains similar input features as the training set but does not include the corresponding target values.
- The testing set is used to evaluate the model's performance and assess how well it generalizes to new, unseen data.
- By making predictions on the testing set and comparing them

with the actual target values (which are known but kept hidden from the model), the model's accuracy, precision, recall, or other performance metrics can be calculated.

- The testing set provides an unbiased evaluation of the model's ability to make predictions on new, real-world data.

Why We Need Training and Testing Sets:

The primary goal of a machine learning model is to generalize well to new, unseen data. The testing set simulates real-world scenarios where the model encounters previously unseen examples.

Overfitting occurs when a model learns the training set too well, capturing noise and outliers instead of underlying patterns. Testing on an independent dataset helps identify overfitting. If a model performs well on the training set but poorly on the testing set, it's a sign of overfitting.

Machine learning models often have hyperparameters that need to be tuned for optimal performance. Testing sets are crucial for assessing different hyperparameter configurations to find the best settings that generalize well.

A machine learning model for predicting flight delays is trained and evaluated using logistic regression and linear regression techniques. Here's a breakdown of what's happening :

.

1. Loading the Dataset:

The code starts by loading a dataset from the file 'dataset.csv'

located at 'C:\\Users\\Ram\\Desktop\\Flight_project\\dataset.csv'. The columns of interest for features (X) and the target variable (y) are defined. Here the target variable is ‘ArrDelayMinutes’ (Difference in minutes between scheduled and actual arrival time. Early arrivals set to 0.)

2. Data Preprocessing:

The target variable 'y' is defined based on whether 'ArrDelayMinutes' is greater than 0 (1 for delays, 0 for no delays).

The features (X) and target variable (y) are split into training and testing sets using the `train_test_split` function from scikit-learn. The test set size is set to 20% of the data, and a random seed (random_state) is used for reproducibility.

3. Logistic Regression Model:

A logistic regression model is created using `LogisticRegression()` from scikit-learn. The model is trained using the training data (X_train, y_train) with the `fit` method. The model accuracy is calculated using the test data, and the accuracy score is printed out.

The trained logistic regression model is saved as a binary file named 'flight_model_logistic_regression.joblib' using `joblib.dump`.

5. Classifying Delay Classes:

The code then categorizes delays into different classes based on the 'ArrDelayMinutes' values.

- If the delay is less than or equal to 30 minutes, it's assigned class 0.
- If the delay is more than 30 minutes but less than or equal to 60 minutes, it's assigned class 1.

- If the delay is more than 60 minutes, it's assigned class 2.
- The 'delay_class' column is added to the DataFrame.

6. Linear Regression Model:

Another linear regression model is created using `LinearRegression()` from scikit-learn. The model is trained using the features (X_train) and target variable (y_train) with the `fit` method. Predictions are made on the test data using the trained model. Mean Squared Error (MSE) and R-squared (R2) are calculated to evaluate the performance of the linear regression model. The trained linear regression model is saved as a binary file named 'linear_regression_model.bin' using `joblib.dump`.

In summary, the code is training two different machine learning models (logistic regression and linear regression) to predict flight delays based on different criteria. The logistic regression model predicts binary delays (yes or no), while the linear regression model predicts delay classes based on delay duration. Both models are evaluated, and the trained models are saved for future use.

MODEL EVALUATION

Model evaluation is the process that uses some metrics whichhelp us to analyze the performance of the model on unseen data to understand how well it generalizes to new examples. Model development is a multi-step process and a check should be kept on how well the model generalizes future predictions. Therefore, evaluating a model plays a vital role so that performance of the model can be judged. The evaluation also helps to analyze a model's key weaknesses. There are many metrics like Accuracy, Precision, Area under Curve, Confusion Matrix, and Mean Square Error to evaluate a model.

ACCURACY

Accuracy is defined as the ratio of the number of correct predictions to the total number of predictions. This is the mostfundamental metric used to evaluate the model.

Using Logistic regression model,
Model Accuracy: 0.8104961543447262
Model saved as flight_model_logistic_regression.joblib

We also train the dataset on a linear regression classifier, since the logistic regression model's accuracy is not satisfactory when predicting a continuous value, dividing the output into sections (classes) can help improve accuracy. By converting the problem into a classification task, the model focuses on predicting which range or category the output falls into, rather than predicting the exact value. This simplifies the problem and often leads to better results, especially if the relationships between inputs and outputs are complex and nonlinear.

WEB SITE DEVELOPEMENT

After developing the sophisticated yet properly trained model, it is time to create a website where a customer can input his flight details and get an accurate prediction. To integrate html, APIs and the ML model, we use flask framework.

Here, we input the existing flight details from the user however the weather data for the particular day of his/her flight is obtained using a weather API.

Components:

1. app.py:

- **Functionality:** Primary application responsible for handling user interactions, weather data retrieval, and predictions.
- **Explanation:**
  - Utilizes the Flask framework for creating a user-friendly web interface.
  - Generates and validates CSRF tokens to secure user data during form submissions.

1. app = Flask(__name__): This line initializes a Flask web application. `Flask(__name__)` creates a Flask instance. The `__name__` variable is a special Python variable that is used to determine the root path of the application.

app = Flask(__name__)

2. app.secret_key = secrets.token_hex(16): This line sets a secret key for the Flask application. The secret key is used for securely signing session cookies and other security-related functionalities. It's important to keep this key secret to ensure the security of the application.

app.secret_key = secrets.token_hex(16)

3. @app.route('/', methods=['GET', 'POST']): This is a decorator in Flask that specifies the route for the following function. In this case, the route is the root URL `'/'`. The `methods=['GET', 'POST']` parameter indicates that this route can handle both `GET` and `POST` requests.

@app.route('/', methods=['GET', 'POST'])

4. def index():: This line defines a function named `index()`. This function will be called when a user accesses the root URL of the application. It handles both `GET` and `POST` requests made to the root URL.

5. current_date = str(date.today()): This line gets the current date using the `date.today()` function from the `date` module. It converts the date to a string format and stores it in the `current_date` variable.

current_date = str(date.today())

6. if 'csrf_token' not in session:: This line checks if the `'csrf_token'`

key is not already present in the session. The session is a Flask object that allows you to store data specific to a user across requests. In this case, it's being used to store a CSRF token.

session['csrf_token'] = secrets.token_hex(16): If the `'csrf_token'` key is not in the session, a new CSRF token is generated using the `secrets.token_hex(16)` function. This token is a unique string of 16 hexadecimal characters, which is then stored in the session.

```
if 'csrf_token' not in session:
    session['csrf_token'] = secrets.token_hex(16)
```

In summary, this code sets up a Flask web application, assigns a secret key for session security, defines a route for the root URL, and handles both `GET` and `POST` requests to the root URL by executing the `index()` function. Within the `index()` function, the current date is obtained, and a CSRF token is generated and stored in the session for security purposes.

The app.py also:

- Collects user input such as departure time, origin, destination, and expected delay, ensuring accurate predictions.
- Communicates with the Weather API to obtain current weather conditions for the specified origin city.
- Invokes machine learning models to predict flight delays based on user input and real-time weather data.
- Presents informative delay messages to users, aiding them in understanding potential disruptions.

 # Flask route for handling form submissions and displaying delay messages

@app.route('/', methods=['GET', 'POST'])

def index():

 # ... (Code for handling form submissions, obtaining weather data, and generating delay messages)

return render_template('index.html', current_date=current_date, result=result, csrf_token=session['csrf_token'])

2. API.py:

- **Functionality:** Interacts with the Weather API to fetch real-time weather data for a specific city.

- **Explanation:**

  - Utilizes the requests library to send API requests and retrieve JSON responses containing weather details.

`base_url`: This variable holds the base URL of the Weather API endpoint. It specifies the API version (`v1`) and the expected response format (`json`).

base_url = "http://api.weatherapi.com/v1/forecast.json"

`params`: This dictionary contains the parameters to be included in the API request. It includes the API key (used for authentication), the `city_name` variable representing the city for which weather data is requested, and the `days` parameter set to `1` to request a

one-day forecast.

```
  params = {
      "key": 'e68b08683c5f4e2a83f142528230611',  # API key for authentication
      "q": city_name,  # Name of the city for which weather data is requested
      "days": 1,  # Request a 1-day weather forecast
  }
```

API Request: The `requests.get()` function is used to send a GET request to the API endpoint with the specified parameters. The response from the API is stored in the `response` variable.

```
 response = requests.get(base_url, params=params)
```

Handling the Response: The code checks the HTTP status code of the response. If the status code is `200`, it indicates a successful API request. In such cases, the response content is parsed as JSON data, and the weather information is returned.

```
  if response.status_code == 200:
      data = response.json()  # Parse the response content as JSON data
      return data  # Return the weather data
  else:
      return None  # Return None if the API request was not
```

successful

- Extracts essential weather parameters, such as wind speed, visibility, temperature, etc., for analysis.

3. ml_model_final.py:

- **Functionality:** Houses pre-trained machine learning models for predicting flight delays.

- **Explanation:**

- Utilizes the pandas library to structure and preprocess data for prediction.

- Implements a Logistic Regression model for binary delay prediction (0 for not delayed, 1 for delayed).

- Incorporates a Linear Regression model to classify delays into severity categories (< 30 minutes, 30-60 minutes, > 1 hour).

```
# Code for loading pre-trained machine learning models (Logistic Regression and Linear Regression)
filename_logistic = 'flight_model_logistic_regression.joblib'
model_logistic = joblib.load(filename_logistic)

filename_linear = 'linear_regression_model.bin'
model_linear = joblib.load(filename_linear)
```

Workflow of the Application:

1. **User Input:**

- Travelers provide input including departure time, origin, destination, and expected delay through the web interface.
- CSRF token validation ensures secure data transmission, safeguarding against unauthorized access or tampering.

2. **Weather Data Integration:**

- The origin city information triggers a request to the Weather API, fetching real-time weather conditions.
- Key weather parameters are extracted, offering vital data for delay prediction and analysis.

3. **Machine Learning Predictions:**

- The Logistic Regression model assesses binary delay prediction, distinguishing between delayed and not delayed flights.
- The Linear Regression model categorizes delays into severity levels based on predicted delay minutes.

4. **Output Generation:**

- Predicted outcomes trigger the creation of user-friendly messages, explaining the likelihood and extent of flight delays.
- Messages are presented on the web interface, empowering travelers with actionable insights for informed decision-making.

Conclusion:

The Flight Delay Prediction System seamlessly integrates user

input, real-time weather data, and sophisticated machine learning techniques. By providing clear and understandable delay messages, the system equips travelers with actionable insights, enhancing their overall travel experience and enabling them to plan accordingly.

LEARNING OUTCOMES

The skills acquired through the completion of the Flight Delay Prediction project includes

- **Utilized Python Libraries:**
  - Pandas: Leveraged Pandas for data manipulation, cleaning, and integration.
  - NumPy:Utilized NumPy for numerical operations and handling arrays.
  - Flask: Employed Flask for web development, creating interactive user interfaces.
  - Requests: Utilized Requests for API integration, fetching real-time weather data.
  - Joblib: Utilized Joblib for saving and loading machine learning models efficiently.

- **New Functions in Machine Learning:**
  - Logistic Regression: Implemented Logistic Regression for binary classification, predicting flight delays.
  - Linear Regression: Applied Linear Regression for predicting delay classes based on continuous values.
  - Session Management: Implemented session management techniques to enhance user security.
  - CSRF Token Handling:Utilized CSRF tokens to prevent cross-site

request forgery attacks, ensuring secure user interaction.

These focused skills encompass essential libraries and functions used in machine learning, web development, and data integration, providing a foundation for future projects and endeavors in the realm of data science.

CONCLUSION

In wrapping up this project, I'm thrilled to say that we've accomplished something truly impactful. By diving deep into machine learning techniques and integrating them with real-time weather data, we've created a sophisticated Flight Delay Prediction system. Through the use of Logistic Regression and Linear Regression models, we've been able to provide users with accurate insights into potential flight delays, making travel planning more reliable and efficient.

Throughout this journey, we've harnessed the power of essential Python libraries such as Pandas, NumPy, Flask, and Requests. These tools enabled us to seamlessly process data, connect with external APIs, and craft an intuitive web interface for users. We've also paid close attention to security, implementing robust session management and CSRF token handling to ensure a safe user experience.

This project has not only sharpened our skills in machine learning, data

integration, and web development but has also demonstrated the practical applications of predictive analytics in the dynamic aviation industry. The hands-on experience gained here equips us with invaluable insights for future data-driven projects, emphasizing the pivotal role of accurate algorithms and real-time data in making predictions that truly matter.

REFERENCES

- Flask:
  - Official Flask Documentation: https://flask.palletsprojects.com/en/2.1.x/

- Requests:
  - Official Requests Documentation: https://docs.python-requests.org/en/latest/

- Pandas:
  - Official Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/index.html
  - Python for Data Analysis, 2nd Edition (2017); Authors: Wes McKinney

- Scikit-Learn:
  - Official Scikit-Learn Documentation: [https://scikit-learn.org/stable/documentation.html](https://scikit-learn.org/stable/documentation.html

- WeatherAPI:
  - WeatherAPI Documentation:
https://www.weatherapi.com/

- Machine Learning:
  - Python Machine Learning, 3rd Edition (2019); Author: Sebastian Raschka

- Data Preprocessing:
  - Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists (2018); Authors: Alice Zheng and Amanda Casar