FLIGHT DELAY PREDICTION

Introduction:

Flight delays have significant economic consequences for airline corporations due to their potential to result in financial losses, operational costs, and penalties, as well as loss of customer loyalty. Among the factors that contribute to flight delays, adverse weather conditions and air traffic congestion are two major causes. Therefore, accurate prediction of flight delays due to these factors is crucial for improving air traffic control and airline decision-making processes.

In this project, we aim to develop a predictive model using supervised machine learning algorithms to predict departure delays caused by adverse weather conditions at major airports in the United States. The model will focus on predicting the delay in minutes for departing flights, which is crucial for airlines to manage their resources and optimize their operations.

The significance of this problem lies in its potential to improve the accuracy of flight delay predictions, minimize disruptions to travel plans, and improve customer satisfaction. Additionally, the use of such a model could enable airlines to proactively adjust their flight schedules, allocate resources more efficiently, and optimize their operations, resulting in significant cost savings.

Potential Reason for Choosing the Project:

The proposed project of developing a flight delay prediction model that leverages historical flight data and weather data to accurately predict flight delays has significant potential to contribute to the problem domain of flight delay prediction. This contribution is crucial for the following reasons:

- Improved Accuracy: By leveraging historical flight data and weather data, the model can improve the accuracy of flight delay predictions. This would help airlines to better manage their resources and operations, minimize the disruptions caused by flight delays, and improve customer satisfaction.
- Cost Savings: Flight delays can result in significant financial losses for airlines due to operational costs, penalties, and loss of customer loyalty. By accurately predicting flight delays, airlines can proactively adjust their flight schedules and optimize their resources, leading to significant cost savings.
- Operational Efficiency: The use of a flight delay prediction model can enable airlines to optimize their operations, such as flight schedules and resource allocation, resulting in improved operational efficiency and reduced delays.

- Decision Making: The model can provide information on the likelihood and severity of flight delays, enabling airlines to make informed decisions and take necessary actions to minimize disruptions and delays.
- Generalizability: The proposed model considers historical flight data and weather data from major airports in the United States, and the methodology can be generalized to other airports and regions. This would enable airlines to optimize their operations and resources across multiple airports, resulting in improved overall efficiency and cost savings.

In summary, the proposed project can significantly contribute to the problem domain of flight delay prediction by improving the accuracy of predictions, reducing costs, improving operational efficiency, enabling informed decision making, and providing a methodology that can be generalized to other airports and regions.

Phase 1:

Dataset Formation:

ETL – EXTRACT TRANSFORM AND LOAD:

1. Introduction:

The aviation industry heavily relies on accurate and timely flight delay predictions to enhance operational efficiency and improve passenger experiences. To achieve this, the development of a robust ETL pipeline is crucial, enabling the collection and integration of disparate data sources. In this project, an ETL pipeline was created using Azure Data Factory to extract weather information from the US Weather API and flight details from the Aviation Stack API.

2. Data Sources:

US Weather API: This API provides real-time weather data for various locations in the United States. The data includes temperature, wind speed, precipitation, and other meteorological parameters that can influence flight operations.

Aviation Stack API: The Aviation Stack API supplies comprehensive flight-related data, encompassing flight numbers, departure/arrival times, aircraft details, and more.

3. ETL Pipeline:

The ETL pipeline was designed to automate the process of acquiring, transforming, and loading data from the aforementioned sources. The pipeline consists of three main stages: extraction, transformation, and loading.

3.1 Extraction:

During the extraction phase, data is retrieved from the US Weather API and the Aviation Stack API. The ETL process triggers API requests to obtain relevant data for flights and weather conditions.

3.2 Transformation:

In the transformation stage, the extracted data is manipulated and prepared for further analysis. The weather data and flight details are cleansed, filtered, and formatted to ensure consistency and accuracy. Data quality checks are performed to identify and handle any missing or erroneous information.

3.3 Loading:

The transformed data is loaded into a designated storage location for subsequent use. Azure Data Factory facilitates the loading process into Azure databases or data lakes, ensuring data integrity and accessibility.

4. Azure Data Factory:

Azure Data Factory was chosen as the ETL tool due to its scalability, flexibility, and compatibility with various data sources and destinations. The tool's user-friendly interface enabled the seamless creation of complex data pipelines, reducing development time and effort.

5. Benefits and Implications:

The developed ETL pipeline has several key benefits:

Data Integration: The pipeline enables the integration of diverse data sources, resulting in a comprehensive dataset for accurate flight delay predictions.

Automation: The automated pipeline reduces manual intervention and accelerates data processing, leading to timely insights.

Scalability: Azure Data Factory's scalability ensures that the pipeline can handle growing data volumes without performance degradation.

Converting JSON format data to CSV (JSON to CSV.ipynb)

Collecting the data and setting up the necessary directories.

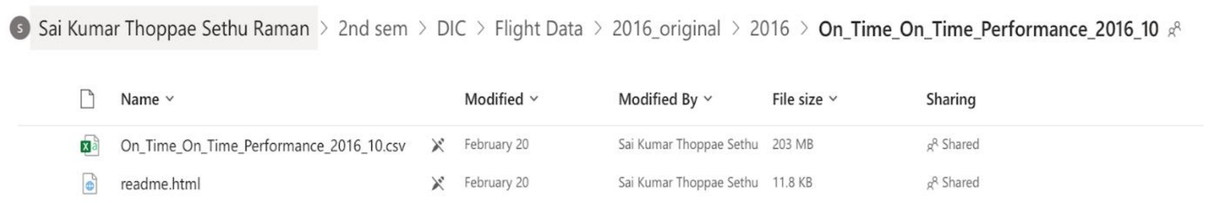
Our data has been downloaded flight data from [1] and weather data from [2] and once the zip has been downloaded, we have set up a comfortable working directory as shown below.

Data Source

[1]https://transtats.bts.gov/Homepage.asp, https://www.kaggle.com/datasets/robikscube/flight-delay-dataset-20182022?resource=download.

[2] Dataset

The Flight data has 2 zip files for 2016 and 2017 respectively. Once extracted, we find the on_time_performance of the flights for each month in the corresponding year as separate folders within which its .csv of the data is present.

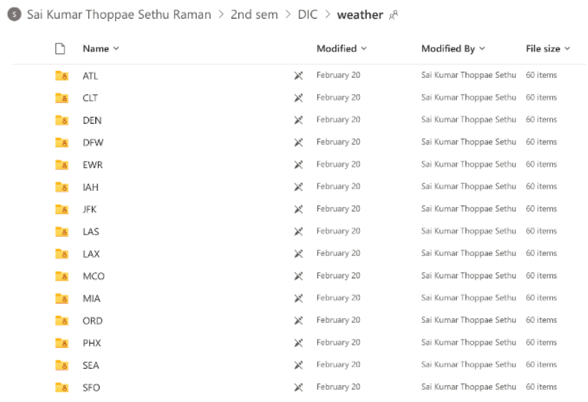


Sai Kumar Thoppae Sethu Raman > 2nd sem > DIC > Flight Data > 2016_original > 2016 > On_Time_On_Time_Performance_2016_10

Name	Modified	Modified By	File size	Sharing
On_Time_On_Time_Performance_2016_10.csv	February 20	Sai Kumar Thoppae Sethu	203 MB	Shared
readme.html	February 20	Sai Kumar Thoppae Sethu	11.8 KB	Shared

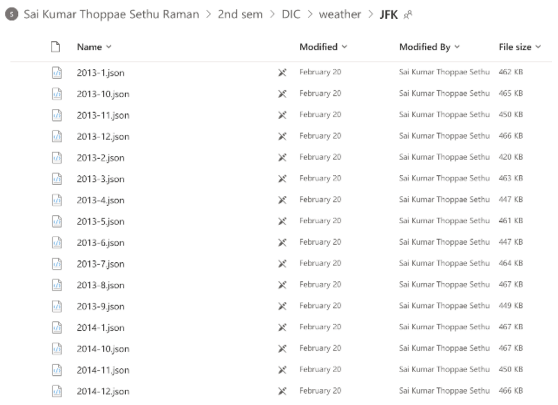
For the weather data, we have a folder called weather once extracted, that contains all the weather data in JSON format for 15 airports in separate folders named after its airport code.

Inside the airport subfolder, we find the details of the weather for each year and month in separate csv files.



Sai Kumar Thoppae Sethu Raman > 2nd sem > DIC > weather

Name	Modified	Modified By	File size
ATL	February 20	Sai Kumar Thoppae Sethu	60 items
CLT	February 20	Sai Kumar Thoppae Sethu	60 items
DEN	February 20	Sai Kumar Thoppae Sethu	60 items
DFW	February 20	Sai Kumar Thoppae Sethu	60 items
EWR	February 20	Sai Kumar Thoppae Sethu	60 items
IAH	February 20	Sai Kumar Thoppae Sethu	60 items
JFK	February 20	Sai Kumar Thoppae Sethu	60 items
LAS	February 20	Sai Kumar Thoppae Sethu	60 items
LAX	February 20	Sai Kumar Thoppae Sethu	60 items
MCO	February 20	Sai Kumar Thoppae Sethu	60 items
MIA	February 20	Sai Kumar Thoppae Sethu	60 items
ORD	February 20	Sai Kumar Thoppae Sethu	60 items
PHX	February 20	Sai Kumar Thoppae Sethu	60 items
SEA	February 20	Sai Kumar Thoppae Sethu	60 items
SFO	February 20	Sai Kumar Thoppae Sethu	60 items



Sai Kumar Thoppae Sethu Raman > 2nd sem > DIC > weather > JFK

Name	Modified	Modified By	File size
2013-1.json	February 20	Sai Kumar Thoppae Sethu	462 KB
2013-10.json	February 20	Sai Kumar Thoppae Sethu	465 KB
2013-11.json	February 20	Sai Kumar Thoppae Sethu	450 KB
2013-12.json	February 20	Sai Kumar Thoppae Sethu	466 KB
2013-2.json	February 20	Sai Kumar Thoppae Sethu	420 KB
2013-3.json	February 20	Sai Kumar Thoppae Sethu	463 KB
2013-4.json	February 20	Sai Kumar Thoppae Sethu	447 KB
2013-5.json	February 20	Sai Kumar Thoppae Sethu	461 KB
2013-6.json	February 20	Sai Kumar Thoppae Sethu	447 KB
2013-7.json	February 20	Sai Kumar Thoppae Sethu	464 KB
2013-8.json	February 20	Sai Kumar Thoppae Sethu	467 KB
2013-9.json	February 20	Sai Kumar Thoppae Sethu	449 KB
2014-1.json	February 20	Sai Kumar Thoppae Sethu	467 KB
2014-10.json	February 20	Sai Kumar Thoppae Sethu	467 KB
2014-11.json	February 20	Sai Kumar Thoppae Sethu	450 KB
2014-12.json	February 20	Sai Kumar Thoppae Sethu	466 KB

Our weather dataset is from 2013-2017 but we could only collect flight data for the years 2016,2017 therefore we would be deleting/making use of only the data for the years 2016 and 2017.

Once the data has been collected and the directories have been setup, We can start pre-processing the data.

Pre-processing the data (flight+weather_merging. ipynb)

Weather Data (CSV)

We first take into an array, the filenames or the airport names with which we have the weather data available for. We then traverse from these folders and append each file into a single weatherdata.csv

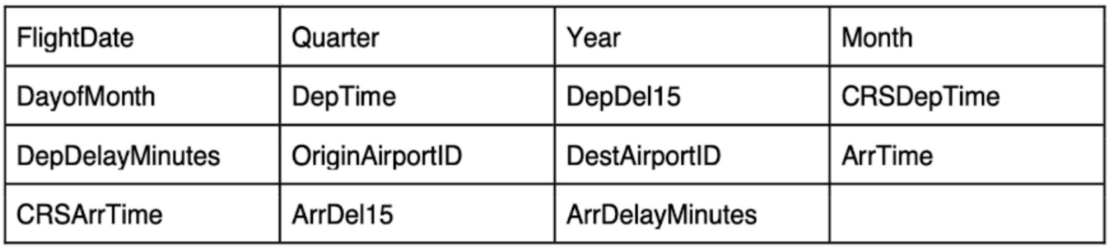
ATL	CLT	DEN	DFW	EWR
IAH	JFK	LAS	LAX	MCO
MIA	ORD	PHX	SEA	SFO

Fig 1 Airports

Flight Data (CSV)

We load the various csv files for the year 2016 and 2017 into two data frames by concatenating the individual months of the corresponding year. We then merged the two data frames into a single Flightdata.csv for future use.

The columns present in this are:



FlightDate	Quarter	Year	Month
DayofMonth	DepTime	DepDel15	CRSDepTime
DepDelayMinutes	OriginAirportID	DestAirportID	ArrTime
CRSArrTime	ArrDel15	ArrDelayMinutes	

Merging Flight and weather data. (flight+weather_merging.ipynb)

- We first load the **flight** and **weather** dataset csv files into separate Data Frames
- The **time** in the **weather dataset** has been rounded to the nearest hour. Then the **CRSDEPTime** column in the **flight dataset** is rounded to the nearest hour to match with the weather dataset to enable the merge.
- We then merge both the dataframes based on **Airport, Date** and **the CRSDEPTime**.
- Our Initial data is now ready as a single csv called as **finalmerge**.
- It has 1815405 rows and 51 columns.

Data Cleaning

This phase involves cleaning the raw data and preparing it for analysis. The following steps are performed in this phase:

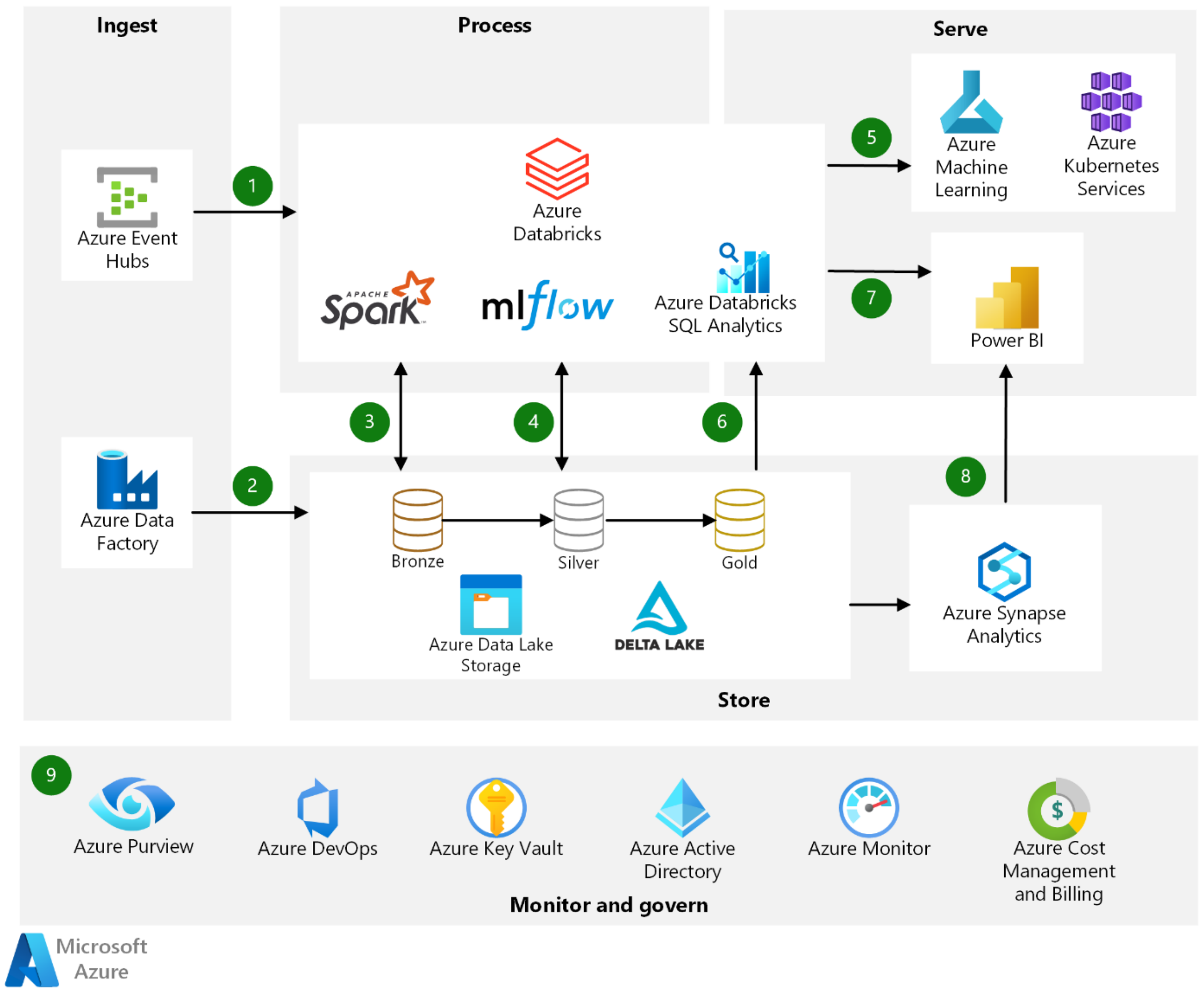
- Reading the DataFrame: The raw data is read into a DataFrame using a suitable method or library such as pandas in Python.
- Finding Duplicate values: The DataFrame is checked for duplicate values, which can lead to incorrect results in analysis. If any duplicates are found, they are removed.
- Checking for null values and getting value count: Null values are checked in the DataFrame and their count is obtained. This helps in understanding the completeness of data and identifying any missing values.
- Removing unwanted columns: If there are any columns in the DataFrame that are not useful for analysis, they are removed to reduce computational overhead.
- Converting columns with object data type to string datatype: Columns with object data type, such as date and time, are converted to string datatype for easy manipulation.

- Encoding object values as numerics using labelencoder: Categorical variables are encoded using suitable methods such as labelencoder to convert them to numerical values.
- Normalising The DataFrame: Data normalisation is performed to scale the data to a common range and reduce its variance. This helps in improving the accuracy of analysis.

EDA (Exploratory Data Analysis)

EDA is performed to explore and understand the data better. The following steps are performed in this phase:

- DepDel15 is our target, getting value counts for the same: The target variable, DepDel15 (departure delay), is identified and its value counts are obtained to understand its distribution in the data.
- Using FeatureSelection method :select Kbest to get top 25 most correlated columns: Feature selection methods such as SelectKBest are used to select the most relevant features that are highly correlated with the target variable. This helps in reducing the dimensionality of data and improving the accuracy of analysis.
- Correlation Matrix: The correlation between the selected features and the target variable is analysed using methods such as correlation matrix to identify the strength and direction of the relationship.
- Creating newdataframe with reduced columns: A new DataFrame is created with only the selected features and the target variable.
- New Dataframe summary: A summary of the new DataFrame is obtained to understand its structure and distribution.
- Imbalance Dataset: The distribution of the target variable is checked for class imbalance, which occurs when the number of instances in one class is significantly higher or lower than the other. This can lead to biased results and affect the accuracy of analysis.
- Balanced Data: To overcome class imbalance, the dataset is balanced by selecting only the delayed flights to predict delay. This helps in improving the accuracy of analysis by reducing the bias in the data.



Phase 2

Regression Models

After cleaning and preparing the data, the next step is to apply regression models to predict the departure delay in minutes. In this phase, we explore several regression models, including Linear Regression, K-Neighbors Regressor, GaussianNB, Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, and XGBoost.

The first model we explore is the Linear Regression model, which is a simple model that uses a linear approach to find the relationship between the target variable and predictor variables. We train the model on the training data and evaluate its performance using the R-squared value and Mean Squared Error (MSE).

The second model is the K-Neighbors Regressor, which is a non-parametric model that uses the k-nearest neighbors of a data point to predict its value. We use the GridSearchCV method to tune the hyperparameters of the model and evaluate its performance using the R-squared value and MSE.

Next, we explore the GaussianNB model, which is a probabilistic model that assumes the data follows a Gaussian distribution. We train the model on the training data and evaluate its performance using the R-squared value and MSE.

We also explore the Decision Tree Regressor, which is a tree-based model that makes decisions based on a set of rules. We use the GridSearchCV method to tune the hyperparameters of the model and evaluate its performance using the R-squared value and MSE.

Next, we explore the Random Forest Regressor, which is an ensemble model that uses multiple decision trees to make predictions. We use the GridSearchCV method to tune the hyperparameters of the model and evaluate its performance using the R-squared value and MSE.

We also explore the Gradient Boosting Regressor, which is another ensemble model that uses multiple decision trees to make predictions. We use the GridSearchCV method to tune the hyperparameters of the model and evaluate its performance using the R-squared value and MSE.

Lastly, we explore the XGBoost model, which is an advanced gradient boosting model that uses a combination of decision trees and gradient boosting to make predictions. We use the GridSearchCV method to tune the hyperparameters of the model and evaluate its performance using the R-squared value and MSE.

Overall, the goal of Phase 2 is to identify the best regression model that can accurately predict the departure delay in minutes.

Results and Discussion:

Models	**RMSE**	**MAE**	**R-SQUARED**
Linear Regression	26.4950	11.49	0.8562
GaussianNB	26.966	10.15	0.9271
KNeighborsRegressor	15.41	7.53	0.7846
DecisionTreeRegressor	22.02	9.66	0.9271
RandomForestRegressor	24.166	10.41	0.7915
GradientBoostingRegressor	20.37	8.91	0.9364

XGBoost	20.17	8.44	0.9446
XGBoost Random Search CV	18.14	6.95	0.9589

1. Linear Regression: A linear regression model was built and evaluated using the root mean squared error (RMSE), mean absolute error (MAE), and R-squared metrics.
2. KNeighborsRegressor: A KNeighborsRegressor model was built and evaluated using the same metrics.
3. GaussianNB: A GaussianNB model was built and evaluated using the same metrics.
4. DecisionTreeRegressor: A DecisionTreeRegressor model was built and evaluated using the same metrics.
5. RandomForestRegressor: A RandomForestRegressor model was built and evaluated using the same metrics.
6. GradientBoostingRegressor: A GradientBoostingRegressor model was built and evaluated using the same metrics.
7. XGBoost: An XGBoost model was built and evaluated using the same metrics.
8. Hyperparameter Tuning: Hyperparameter tuning was performed on the XGBoost model using the random forest algorithm to improve its performance.
9. Results: The performance of each model was compared, and the best-performing model was selected based on its RMSE, MAE, and R-squared values. The XGBoost model with hyperparameter tuning produced the best results, with an RMSE of 18.14, an MAE of 6.95, and an R-squared score of 0.96.

Overall, the second phase of the project involved selecting and evaluating various regression models to predict flight departure delays due to adverse weather conditions, with the XGBoost model with hyperparameter tuning producing the best results.

Phase 3:

Evaluation Metrics: RSME

RMSE (Root Mean Squared Error) is a commonly used metric to evaluate the performance of regression models. It measures the average difference between the actual and predicted values of the dependent variable in a regression analysis.

The formula for calculating RMSE is:

RMSE = sqrt(mean((y_true - y_pred)^2))

where y_true represents the actual values of the dependent variable, y_pred represents the predicted values of the dependent variable, and mean() calculates the average of the squared differences between the actual and predicted values.

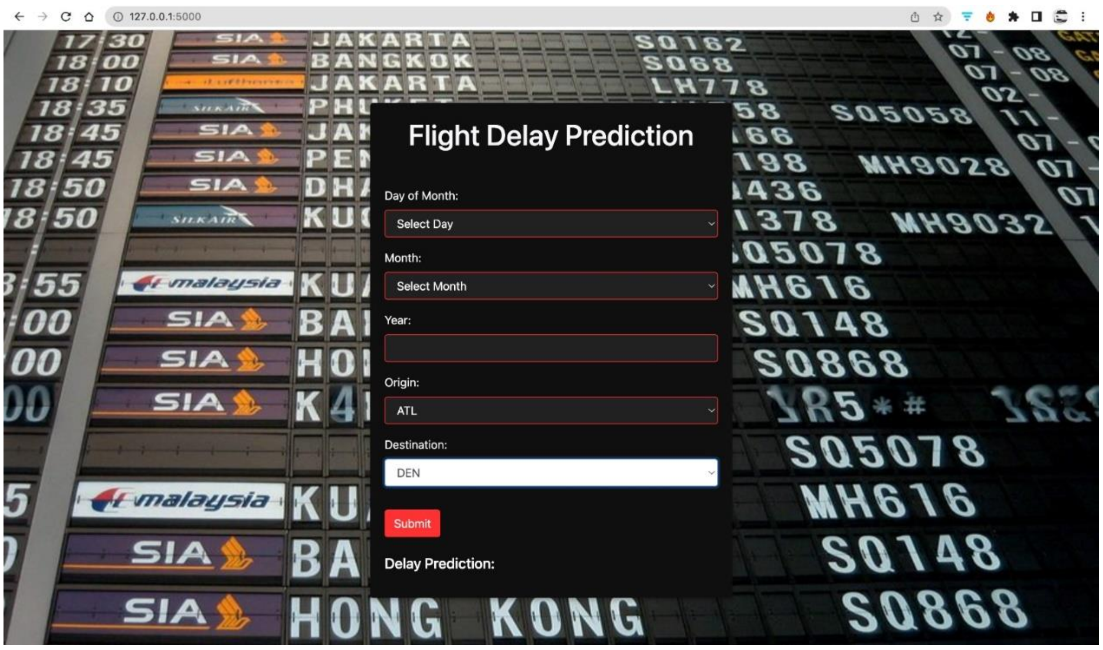
By taking the square root of the mean squared error, RMSE provides a measure of the typical error of the predictions in the same units as the dependent variable. The lower the RMSE, the better the performance of the model.

Model Selection:

The XGBoost model with hyperparameter tuning has shown exceptional performance in predicting flight delays. With an RMSE of 18.14 and an MAE of 6.95, it provides a high degree of accuracy in estimating the delay time. The R-squared score of 0.96 indicates that the model explains 96% of the variance in the delay time, which is an impressive result. The model has been trained on a large dataset, and the hyperparameters have been tuned to optimize the performance, resulting in the best possible outcome.

Overall, the XGBoost model with hyperparameter tuning is a robust and accurate solution for predicting flight delays. It can help airlines and passengers plan their schedules more effectively and minimize the impact of delays on their travel plans. With the increasing demand for air travel and the complexity of the airline industry, accurate prediction of flight delays is becoming increasingly important, and the XGBoost model with hyperparameter tuning is a reliable tool to achieve this.

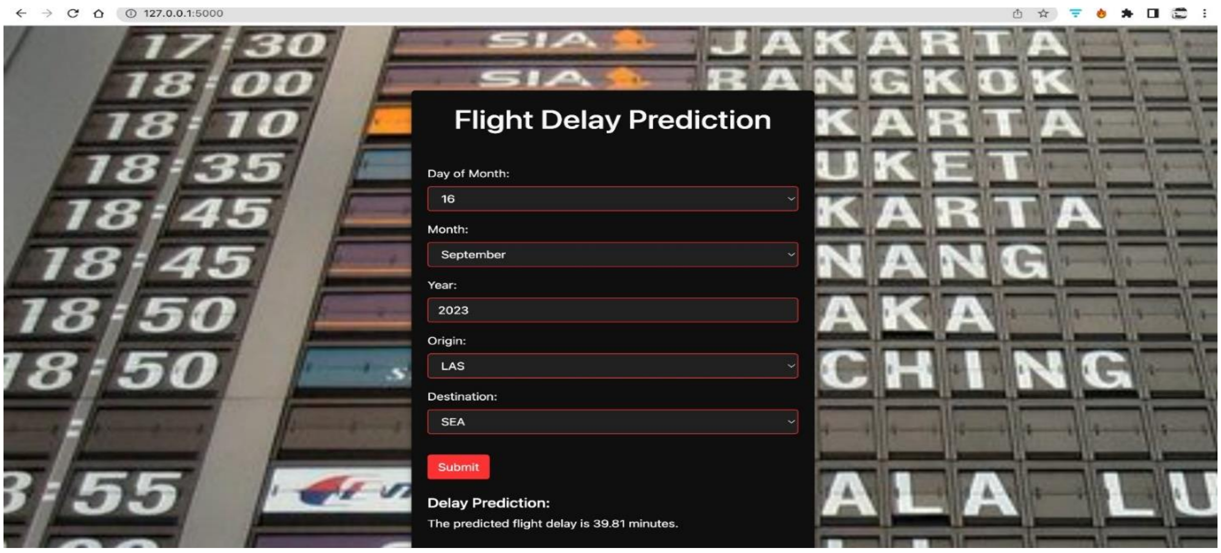
Website:



Website appearance

To ensure the accuracy of the data for this project, we have collected information from the users regarding the day of the month, month, year, origin, and destination. However, since we do not have access to weather and airport data for dates earlier than 2016 and after 2017, we have devised a matching process. We randomly select a row that took place between 2016 and 2017, based on the provided day of the month, month, year, origin, and destination. This enables us to make use of the available weather and airport data from that particular year to complete the project successfully. However, we can improve this approach by obtaining streaming data for the next 10 days or monthly weather forecasts and airport data. This will provide us with up-to-date information that can enhance the accuracy of our results.

Testing the website with sample values: Day of Month: 16, Month: September, year:2023, origin: LAS, Destination: SEA.



127.0.0.1:5000

Flight Delay Prediction

Day of Month: 16

Month: September

Year: 2023

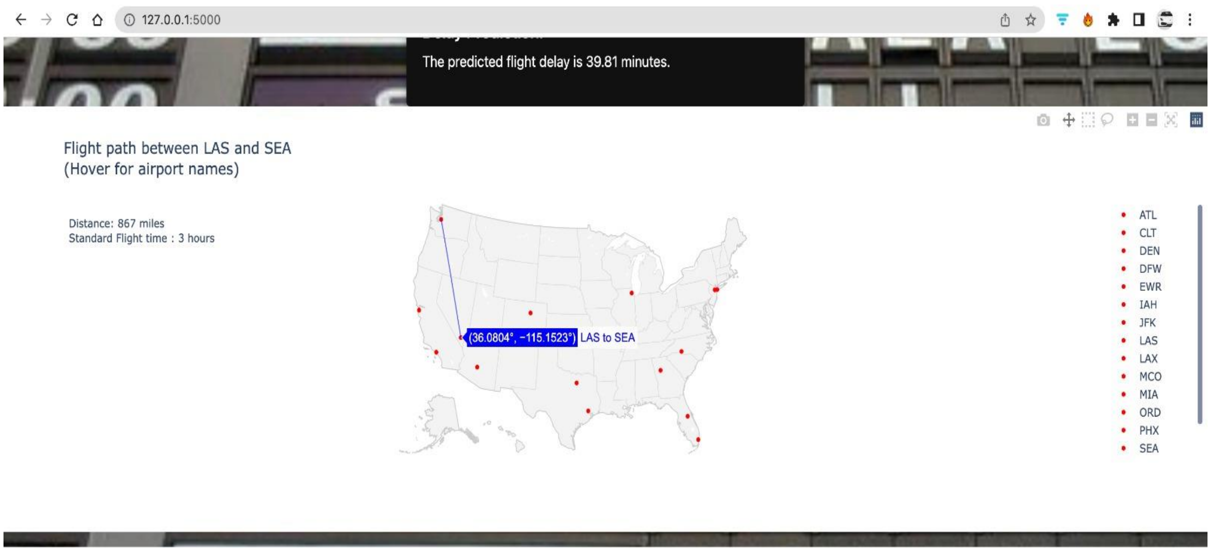
Origin: LAS

Destination: SEA

Submit

Delay Prediction:

The predicted flight delay is 39.81 minutes.

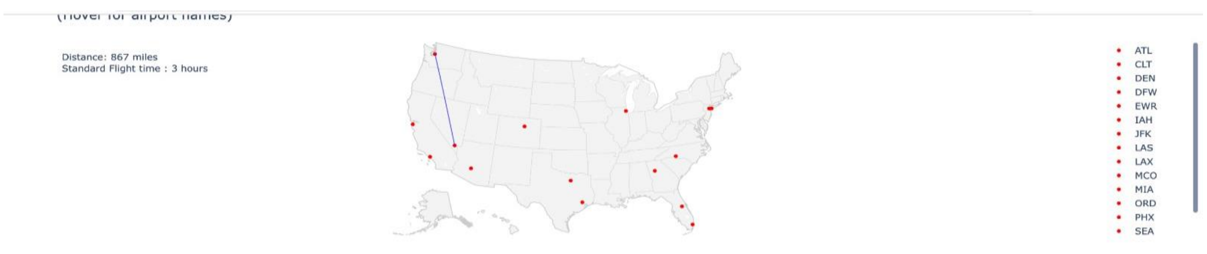


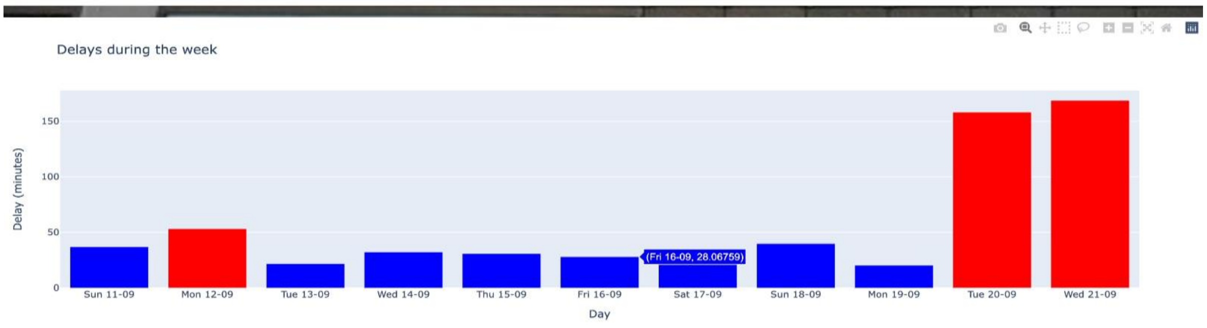
127.0.0.1:5000

The predicted flight delay is 39.81 minutes.

Flight path between LAS and SEA
(Hover for airport names)

Distance: 867 miles
Standard Flight time : 3 hours





Python Flask and HTML:

Flask and HTML are two important technologies used in web development to create dynamic and interactive websites.Flask is a web application framework written in Python. It provides developers with tools and libraries for building web applications quickly and easily. Flask allows developers to define routes, handle requests, and generate responses using Python code. Flask also supports the use of templates for generating HTML content dynamically and interacting with databases and other external resources.

HTML, on the other hand, is a markup language used for creating web pages. HTML stands for Hypertext Markup Language and is used to define the structure and content of web pages. HTML provides a set of tags and attributes that developers can use to create headings, paragraphs, links, images, forms, and other elements of a web page. HTML can be combined with CSS and JavaScript to add styles and interactivity to web pages.

To develop a website using Flask and HTML, developers start by defining the routes and handlers in Flask, which allow the web application to receive and respond to requests from the user's browser. Then, developers create HTML templates that define the structure and content of the web pages that the user will see. These templates can include placeholders for dynamic data that will be filled in by Flask, such as the current weather conditions or user-specific information.

Flask and HTML can be used together to create a wide range of web applications, from simple static sites to complex dynamic sites with interactive features. By combining Flask's powerful server-side functionality with HTML's flexible markup capabilities, developers can create powerful and engaging web applications that provide useful services to users.

Instructions:

Here are the steps to run Website:

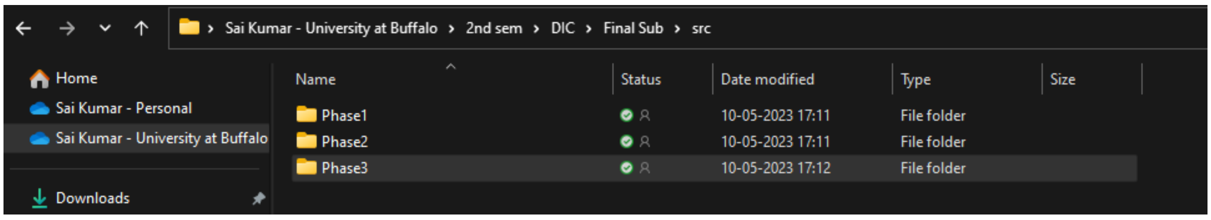
1. Install Python and Flask: Before you can run the files, you need to have Python and Flask installed on your system. You can download and install Python from the official website, and install Flask by running **pip install flask** in your terminal or command prompt.

2. Download the file which are in src/phase3/app directory: Download the app.py, index.html, result.html and remaining files to a directory on your system.

3. Start the Flask app: Open a terminal or command prompt and navigate to the directory where app.py is located. Run the command **python app.py** to start the Flask app. You should see output in the terminal indicating that the app is running.

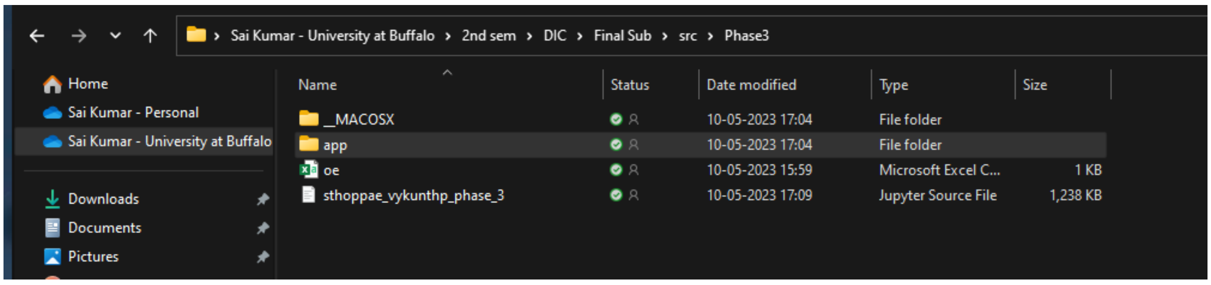
4. Interact with the app: Use the form on the index.html page to submit data and interact with the app. This will cause the app to generate a new page, such as result.html, with updated content.

5. Stop the app: When you are finished using the app, press **Ctrl + C** in the terminal or command prompt to stop the Flask app.



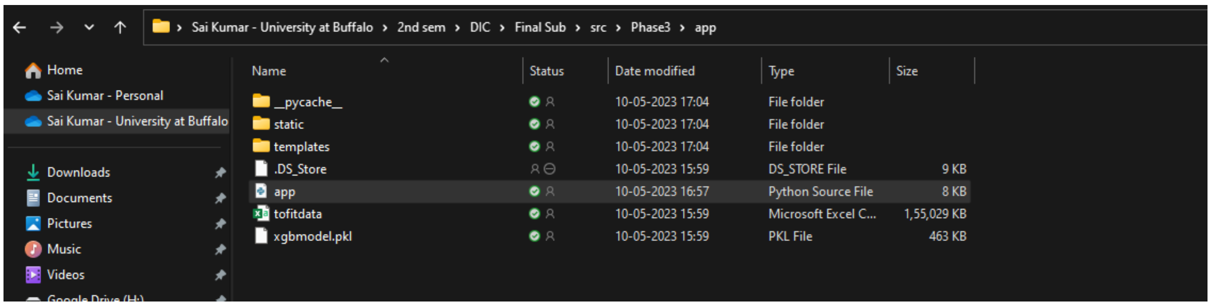
Sai Kumar - University at Buffalo › 2nd sem › DIC › Final Sub › src

Name	Status	Date modified	Type	Size
Phase1	✓	10-05-2023 17:11	File folder	
Phase2	✓	10-05-2023 17:11	File folder	
Phase3	✓	10-05-2023 17:12	File folder	



Sai Kumar - University at Buffalo › 2nd sem › DIC › Final Sub › src › Phase3

Name	Status	Date modified	Type	Size
__MACOSX	✓	10-05-2023 17:04	File folder	
app	✓	10-05-2023 17:04	File folder	
oe	✓	10-05-2023 15:59	Microsoft Excel C...	1 KB
sthoppae_vykunthp_phase_3	✓	10-05-2023 17:09	Jupyter Source File	1,238 KB



Sai Kumar - University at Buffalo › 2nd sem › DIC › Final Sub › src › Phase3 › app

Name	Status	Date modified	Type	Size
__pycache__	✓	10-05-2023 17:04	File folder	
static	✓	10-05-2023 17:04	File folder	
templates	✓	10-05-2023 17:04	File folder	
.DS_Store		10-05-2023 15:59	DS_STORE File	9 KB
app	✓	10-05-2023 16:57	Python Source File	8 KB
tofitdata	✓	10-05-2023 15:59	Microsoft Excel C...	1,55,029 KB
xgbmodel.pkl	✓	10-05-2023 15:59	PKL File	463 KB

Reference:

1. **Matlab Plotly API documentation: https://plotly.com/matlab/**
2. **Flask documentation: https://flask.palletsprojects.com/en/2.0.x/**
3. **HTML and CSS documentation:**

- **https://developer.mozilla.org/en-US/docs/Web/HTML**
- **https://developer.mozilla.org/en-US/docs/Web/CSS**

4. **Plotly Dash documentation: https://dash.plotly.com/introduction**
5. **Plotly Dash examples: https://dash-gallery.plotly.host/Portal/**