ARCHITECTURE

# Flight fare prediction using machine learning

*Submitted by*
**Laizin V**
**laizin2107@gmail.com**

# Document version control

| date issued | version | description | author |
| --- | --- | --- | --- |
| 09/05/2023 | 1.0 | Architecture | Laizin v |

# ARCHITECTURE

# Abstract

This project aims to predict flight prices using machine learning techniques. The proposed solution involves collecting and analyzing flight data from our dataset which is over 450000 flight records, including historical flight prices, weather conditions, flight routes, and other relevant factors.

The project will use machine learning algorithm , that is Random forest regressor, to predict flight prices accurately. The developed system will help travelers plan their trips efficiently and save money by predicting the best time to book their flights. The project's success will be evaluated based on the accuracy and efficiency of the developed model, and its ability to provide valuable insights and predictions for travelers.

users have to input a few parameters in order to predict the flight fare

# 1. Introduction

## 1.1 why Architecture?

The architecture of the flight price prediction app is of paramount importance as it lays the foundation for the entire system. A well-designed architecture provides a clear understanding of how different components of the application work together, ensuring efficient and effective development. It serves as a blueprint for the development team, enabling them to make informed decisions about the system's structure, technologies, and integration points. The architecture documentation acts as a communication tool, facilitating collaboration and understanding among developers, designers, testers, and stakeholders. It also plays a crucial role in the maintenance and scalability of the application, allowing for easy identification of areas for improvement or optimization. Additionally, the architecture documentation aids in troubleshooting and debugging, providing insights into the system's design principles and component interactions. It assists in onboarding new team members and ensuring knowledge transfer, enabling the seamless continuation of development efforts

## 1.2 Scope

The scope of the architecture for the flight price prediction app encompasses the design and organization of the system's components and their interactions. It defines the high-level structure, technologies, and architectural patterns employed to achieve the desired functionality

# Architecture

## 2.1 Data collection

Data collection is a critical step in the development of any machine learning application. In order to build accurate and effective models, it is essential to collect high-quality data that is relevant to the problem at hand. Here are some important considerations for data collection in a machine learning app

- Before collecting data, it is important to clearly define the problem you are trying to solve with your machine learning application. This will help ensure that the data you collect is relevant and useful for training your models
- It is important to collect data that is diverse in terms of the range of values and types of inputs. This can help improve the accuracy and robustness of your models by exposing them to a wider range of data

***https://www.kaggle.com/datasets/yashdharme36/airfare-ml-predicting-flight-fares .***

Dataset is collected from the above link (kaggle). this data is in the CSV format . consists of 450000+ previous flight fare records between the certain indian cities , which is a limited source of dataset . we will try to get the maximum out of this dataset

## 2.2 EDA and FE

EDA stands for Exploratory Data Analysis. It is a crucial step in the machine learning workflow that involves performing an initial analysis of the data to gain insights and better

understand the relationships between the variables. EDA helps to identify patterns, anomalies, and relationships in the data, which can help guide the development of machine learning models and feature engineering

We will use matplotlib and seaborn to visualize and study the data to understand the pattern in the features of our dataset

- we have used ***pandas*** for the data related operations throughout the development

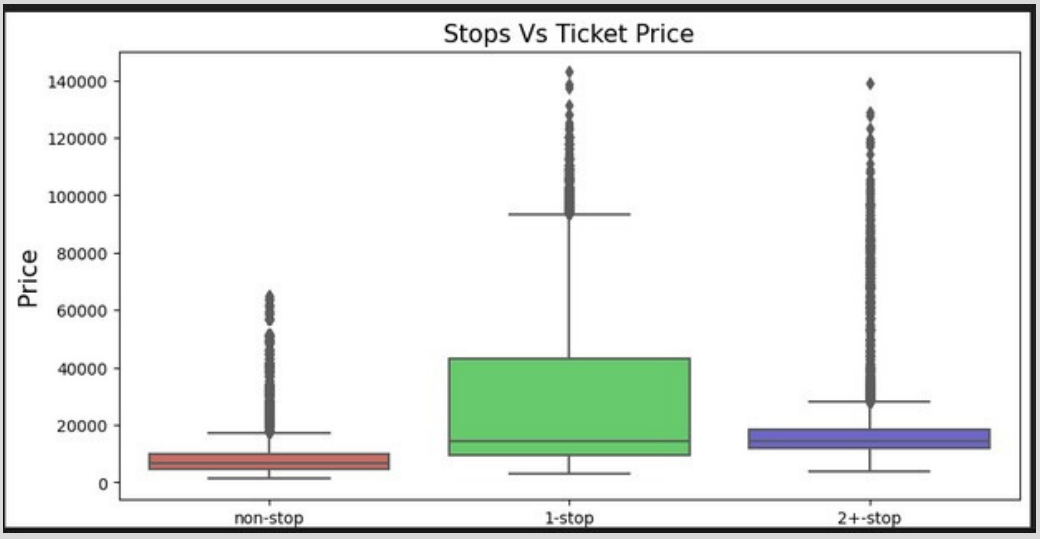
We had ***452087*** rows of data initially

| | Duration_in_hours | Days_left | Fare |
|---|---|---|---|
| count | 452088.000000 | 452088.000000 | 452088.000000 |
| mean | 12.349222 | 25.627902 | 22840.100890 |
| std | 7.431478 | 14.300846 | 20307.963002 |
| min | 0.750000 | 1.000000 | 1307.000000 |
| 25% | 6.583300 | 13.000000 | 8762.750000 |
| 50% | 11.333300 | 26.000000 | 13407.000000 |
| 75% | 16.500000 | 38.000000 | 35587.000000 |
| max | 43.583300 | 50.000000 | 143019.000000 |

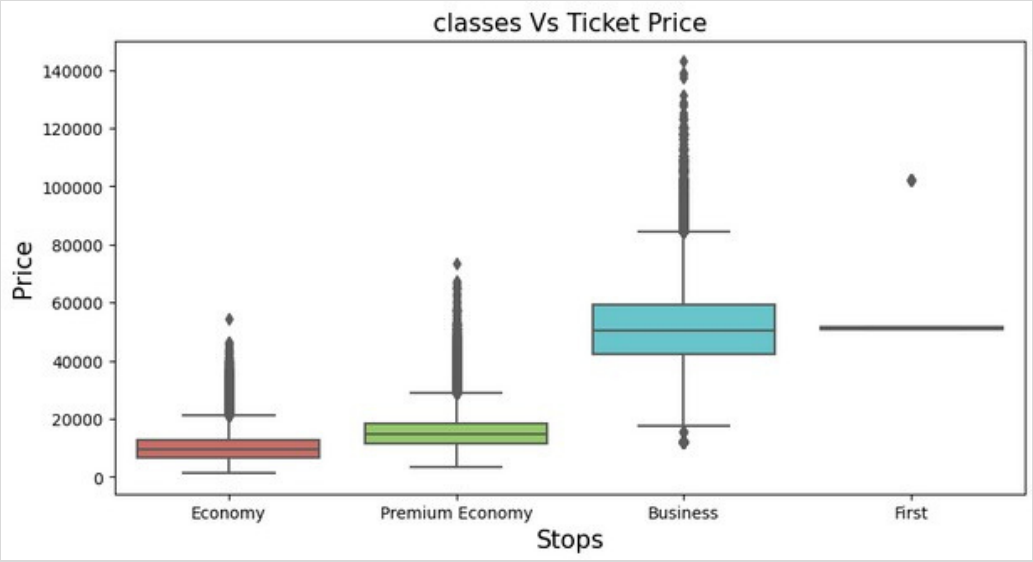
We then checked if there are any null values in the dataset. fortunately we do not have any null values in our dataset. if there were some, we will delete the row which contains the null values as we a ahve a huge datset

# ARCHITECTURE

we then checked the variation of the fare compared to some other features using the matplot lib library.

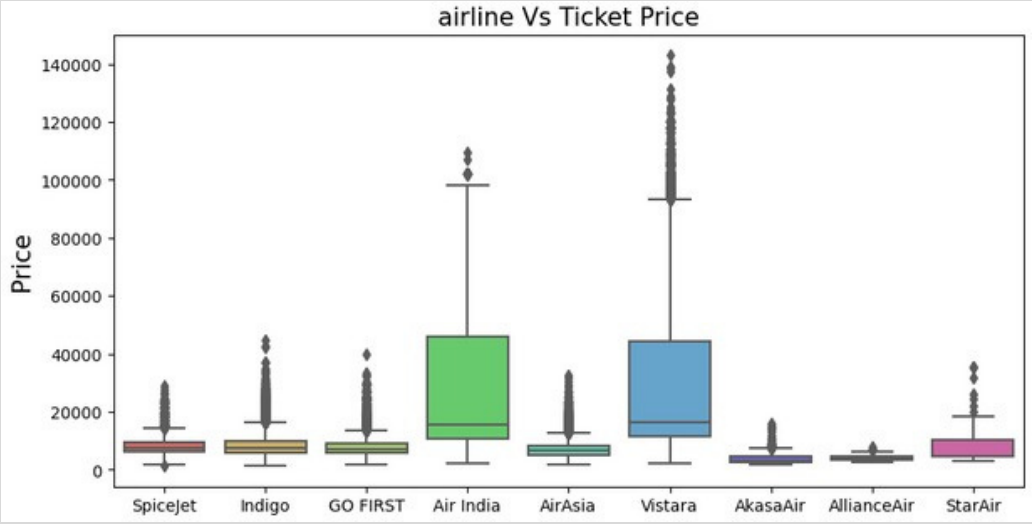



next we have checked the variation of fare with the classes of flights




after this we have also checked the variation of fare compared to the airlines. because the fare have a big dependency on the airline we choose . some airlines may charge high travel fares and some airlines may charge very low fares. as we know airlines like vistara costs much higher than airlines like go first

# ARCHITECTURE




we have also checked the fares variations with arrival city, departure city , arrival time and departure time. also duration and stops . all these features are much important to consider. all these variations are shown in the jupyter notebook using matplotlib.

***Date and time conversion***

The date column we have converted into month, year, day, and day of the week. day of the week is important because the weekend days , the charges might be much higher compartively

```
df['Date_of_journey'] = pd.to_datetime(df['Date_of_journey'])
#converting the date column to datetime object

# Extract features from "Date_of_journey" column
df['Year'] = df['Date_of_journey'].dt.year
df['Month'] = df['Date_of_journey'].dt.month
df['Day'] = df['Date_of_journey'].dt.day
df['Day_of_week'] = df['Date_of_journey'].dt.dayofweek

# Drop the original "Date_of_journey" column because we dont need it
df.drop('Date_of_journey', axis=1, inplace=True)
```

Above code snippet shows how the conversion is implemented in the code with the datetime module in python

# ARCHITECTURE

After that, we dropped some columns. date journey column we have deleted because we have already converted it into different unique columns. and later we have also deleted the flight code column
flight code column has 1400+ unique values and which are related to the airline and class columns. the wide range of unique values can cause complexity while encoding

***Performed Shapiro-Wilk normality test to find out if the flight code really matters or not***

The Shapiro-Wilk test is a statistical test that is used to determine whether a set of data follows a normal distribution. It is often used to test the assumption of normality in a dataset before performing parametric statistical tests, such as t-tests and ANOVA.

```
import scipy.stats as stats

# Assume flightcode_mean_fare is the pandas Series with the mean fare f

# Perform Shapiro-Wilk normality test
shapiro_results = stats.shapiro(flightcode_mean_fare)

# Print the test statistic and p-value
print(f"Shapiro-Wilk test statistic: {shapiro_results.statistic:.3f}")
print(f"Shapiro-Wilk p-value: {shapiro_results.pvalue:.3f}")
```

***Shapiro-Wilk test statistic: 0.777***
***Shapiro-Wilk p-value: 0.000***

## *Feature extraction (FE)*

After dropping the specific columns , we have came to the conclusion that , remaining columns should be the features for the prediction

```
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Airline           440087 non-null  object
 1   Classes           440087 non-null  object
 2   Source            440087 non-null  object
 3   Departure         440087 non-null  object
 4   Total_stops       440087 non-null  object
 5   Arrival           440087 non-null  object
 6   Destination       440087 non-null  object
 7   Duration_in_hours 440087 non-null  float64
 8   Days_left         440087 non-null  int64
 9   Year              440087 non-null  int64
 10  Month             440087 non-null  int64
 11  Day               440087 non-null  int64
 12  Day_of_week       440087 non-null  int64
dtypes: float64(1), int64(5), object(7)
```

Above snippet shows the features for our machine learning model. this dataframe we have names as "X" . and we have named our target variable dataframe "y"

***X - dataframe of features***

***y - dataframe of target variable ("fare")***

## 2.3 Preprocessor pickling

We have to preprocess the data , means we have to do some encoding to normalize the data . we have created the pipeline for encoding and training . later that , we have to save the preprocessor to preprocess the new dataframe which we will get from input through our flask app

# ARCHITECTURE

To achieve this we have used three modules from scikit-learn

***ColumnTransformer***
***Pipeline***
***Onehotencoder***

Onehot encoder is usewd to encode allcolumns which are having categorical values. code snippet is shown below

```
# create the transformer for categorical features
cat_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])


# create the column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', cat_transformer, categorical_features)
```

OneHotEncoder is a class in the scikit-learn library that is used to transform categorical data into numerical data. It is a part of the preprocessing module of scikit-learn and is commonly used in machine learning pipelines to preprocess data before modeling.

We have transformed the train and test datasets using the preprocessor we have created , we then saved the preprocessor using pickle module . after saving as pickle object , we can load the object anytime to preprocess the new dataframe

```
# transform the training data
x_train_processed = preprocessor.fit_transform(x_train)

# transform the test data
x_test_processed = preprocessor.transform(x_test)

# save the preprocessor as a pickle object
import pickle
with open('preprocessor.pkl', 'wb') as f:
    pickle.dump(preprocessor, f)
```

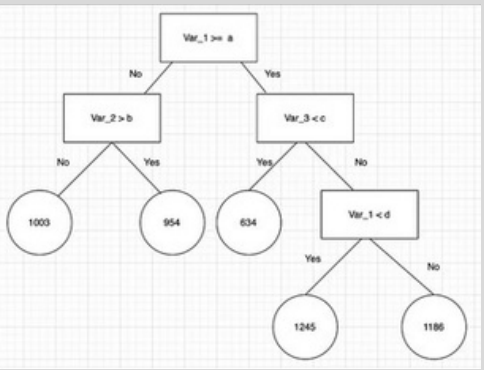
## 2.4 Model training and pickling

Model training is a process in machine learning where a model is developed by training it on a dataset. In this process, the model is fed a set of inputs along with their corresponding outputs, and it learns to recognize patterns in the data and make predictions based on those patterns. The goal of model training is to optimize the model's performance, so it can accurately predict outputs for new inputs.

Pickling is a process of serializing Python objects into a binary format that can be stored or transmitted over the network. In the context of machine learning, pickling is often used to save trained models to disk, so they can be easily reloaded and used to make predictions on new data without having to retrain the model from scratch. By pickling a trained model, you can save the state of the model along with its weights and parameters, allowing you to recreate the model exactly as it was trained, even if the original code or environment is no longer available.

***Random forest regressor*** is the machine learning technique we have used to train the model on

In Random Forest Regression, each decision tree is trained on a random subset of the features and a random subset of the data. This helps to reduce overfitting and improve the model's generalization performance. During prediction, the model takes the average of the predictions made by all the decision trees to make a final prediction.



# ARCHITECTURE

We trained the preprocessed data on random forest regressor and then calculated MSE

MSE - 41636963.86023311

***Mean squared error (MSE)*** is the average of the summation of the squared difference between the actual output value and the predicted output value. Our goal is to reduce the MSE as much as possible

After finalizing the model we have saved the model as a pickle object . so we can load the object anytime to make prediction on new data

```
import pickle

# save the model to a file
with open('random_forest_model.pkl', 'wb') as f:
    pickle.dump(rf, f)
```

## 2.5 Prediction process

After saving both preprocessor and model as pickle objects . we can now go ahead into testing our machine learning algorithm on new data

for that , we have to load the both preprocessor and model we have saved using load object from pickle library

```
with open('model/preprocessor.pkl', 'rb') as file:
    preprocessing_pipeline = pickle.load(file)
with open('model/random_forest_model.pkl', 'rb') as f:
    model = pickle.load(f)
```

# ARCHITECTURE

First we have to create a pandas dataframe that includes all the feature input we need to make the prediction , the inputs we need are

***Airline***
***Classes***
***Source***
***Departure***
***Total_stops***
***Arrival***
***Destination***
***Duration_in_hours***
***Days_left***
***Year***
***Month***
***Day***
***Day_of_week***

After creating the dataframe , we need to feed the dataframe to our loaded preprocessor and assign it as preprocessed input. this preprocessed input is then fit on our model to make prediction

```
preprocessed_new_data = preprocessing_pipeline.transform(new_data)

# Make predictions using the trained model
predictions = model.predict(preprocessed_new_data)
print("predicted price is : ",predictions)
```

As the algorithm is used to make predictions based on attributes , we can cross check the predictions by checking out the real world flight fares in travel websites by filtering the results that matches our features we have given to our model to make prediction

# 3. Technical specifications

## 3.1 core technology used

Machine learning is a core technology used in the flight price prediction app, enabling accurate and data-driven predictions. Machine learning is a branch of artificial intelligence that focuses on developing algorithms and models capable of learning patterns and making predictions or decisions based on data. In the context of the app

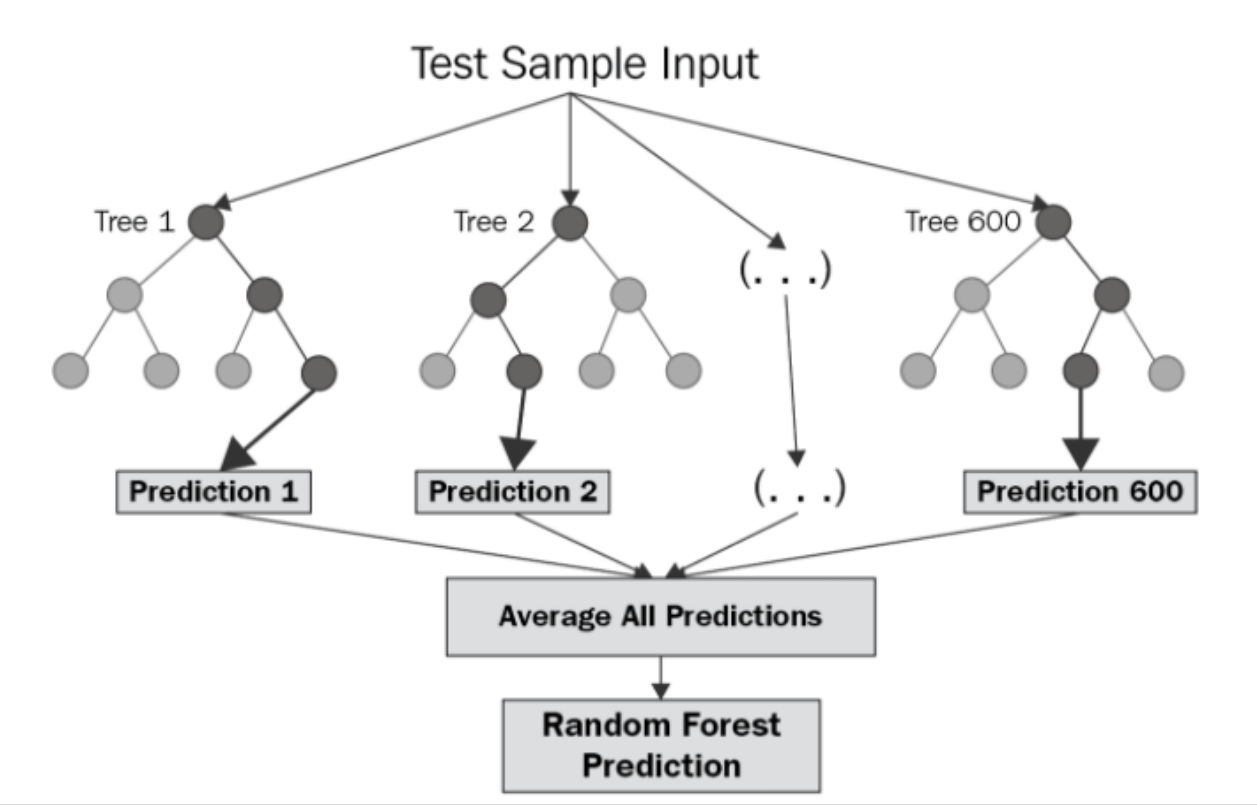
Machine learning algorithms analyze historical flight data, including factors such as departure time, airline, route, and seasonality, to identify patterns and relationships that influence flight prices. These algorithms are trained using labeled historical data, where the actual flight prices are known. Once trained, the models can make predictions on new, unseen data, providing estimates of future flight prices. Machine learning techniques applied in the app include Random Forest Regression, a powerful ensemble learning method that combines multiple decision trees to make accurate predictions. By leveraging machine learning, the flight price prediction app can provide users with valuable insights and predictions, empowering them to make informed decisions when booking flights.

## Random forest regressor

After testing out the few machine learning models , we have concluded that the random forest regressor gives the best results among all the other models

# ARCHITECTURE

The Random Forest Regressor is a popular machine learning algorithm employed in the flight price prediction app. It is an ensemble learning method that combines multiple decision trees to make accurate predictions. In the case of regression tasks, such as predicting flight prices, the Random Forest Regressor leverages a collection of decision trees to generate an ensemble prediction. Each decision tree is constructed using a random subset of the training data and a random subset of the input features. During training, the decision trees learn to split the data based on different feature thresholds, recursively creating nodes and leaves that represent different decision paths. When making predictions, each tree in the ensemble independently generates a prediction, and the final prediction is obtained by averaging or taking the majority vote of the individual tree predictions.




pic : https://levelup.gitconnected.com

## 3.2 Perfomance measurment

1. Mean Squared Error (MSE): MSE measures the average squared difference between the predicted and actual values. A lower MSE indicates better accuracy, with zero being the optimal value.
2. Root Mean Squared Error (RMSE): RMSE is the square root of MSE and provides a measure of the average magnitude of the prediction errors. Like MSE, a lower RMSE signifies higher accuracy.
3. Mean Absolute Error (MAE): MAE calculates the average absolute difference between the predicted and actual values. It provides a measure of the average prediction error magnitude.
4. R-squared (R2) Score: R2 score indicates the proportion of the variance in the target variable that can be explained by the model. It ranges from 0 to 1, with 1 representing a perfect fit. Higher R2 scores signify better accuracy.

## 3.3 Technology stack

1. Python  for backend
2. Flask framework for web part
3. HTML, CSS, Bootstrap for web page design
4. Heroku for deployment
5. VS code as IDE
6. Scikit-learn for model training

# 4. Future Improvements

## 4.1 Upgrading plans and improvements

Currently the application can predict fares for very limited flight routes as our training data was so limited . in future we can add more to our training data and make prediction for more routes

Add more data of flight routes

- Add more airlines
- Continuous upgradation
- Fare comparing for nearest dates
-

# 5. Conclusion

Its concluding that we have explained the Low level design (LLD) for the flight fare prediction application using machine learning . we have seen the architecture, flow of working and user interface. also we have discussed the future improvements that can be integrated to this application