



## LOW LEVEL DESIGN (LDD)

# 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	Initial LLD v 1.0	Laizin v

# Low level design (LLD)

## Contents

Document version control.....	2
Abstract.....	3
1 Introduction	
1.1 Why low-level document.....	5
1.2 Scope.....	5
2 Architecture	
2.1 Data collection.....	6
2.2 EDA and FE.....	6
2.3 Preprocessor pickling.....	11
2.4 Model training and pickling.....	14
2.5 Prediction process.....	14
3 User interface	
3.1 Landing page introduction.....	16
3.2 Inputting features.....	16
3.3 Usage of web frameworks.....	17
3.4 Navigation layout.....	18
4 Future improvements	
4.1 upgrading plans and improvements.....	18
5 Conclusion.....	19

# 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

# Low level design (LLD)

## 1. Introduction

### 1.1 why Low-level document

Low-level documentation (LLD) is created during the development process to provide detailed information about individual components. This documentation is intended for software developers and programmers who will be implementing the system, and provides a guide to the users for the understanding of how the application is developed

1. Implements the design aspects of the app
2. Explains the architecture of the project

- reliability
- maintainability
- use cases

### 1.2 Scope

The scope of a High-Level Design (HLD) document refers to the boundaries and limitations of the system being designed. It defines what the system will and will not do, what features and functionalities it will have, and what user requirements it will meet. The scope of the HLD document is critical in ensuring that the system meets the intended purpose and that the stakeholders' expectations are met

## 2. 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

## Low level design (LLD)

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

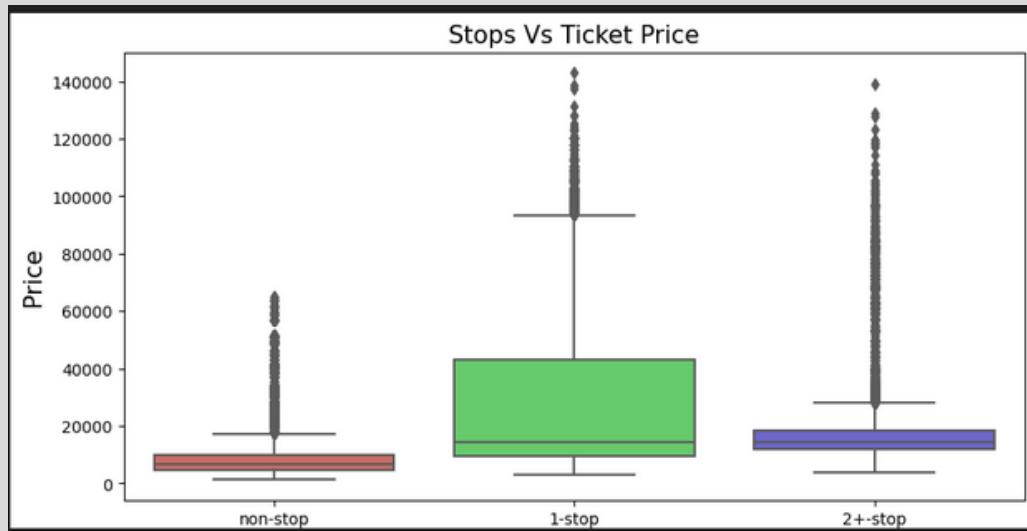
We had **452087** rows of data initially

	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

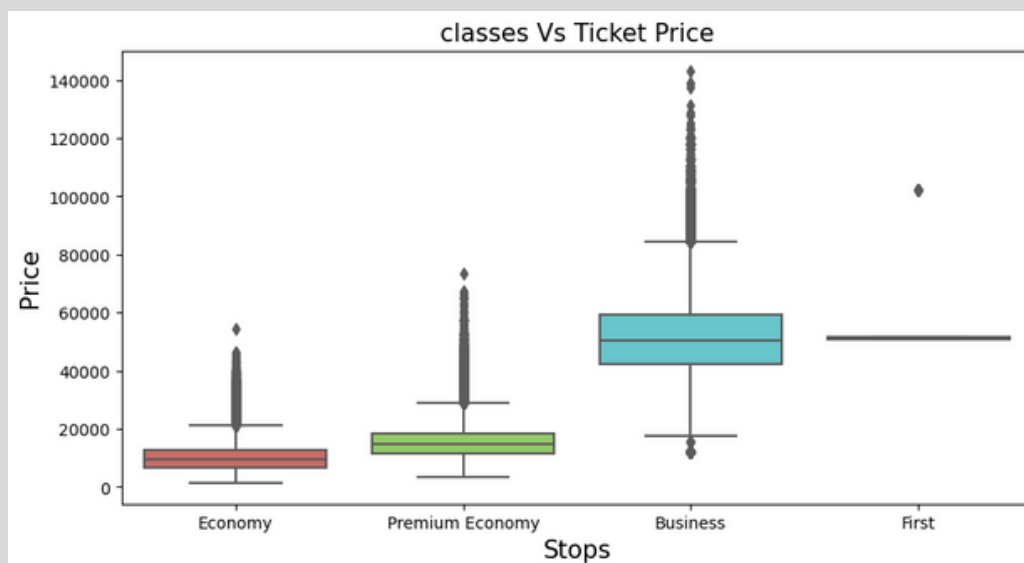
We then checked if there are any null values in the dataset. fortunately we do not have any null values in our dataset. if there were some, we will delete the row which contains the null values as we have a huge dataset

# Low level design (LLD)

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



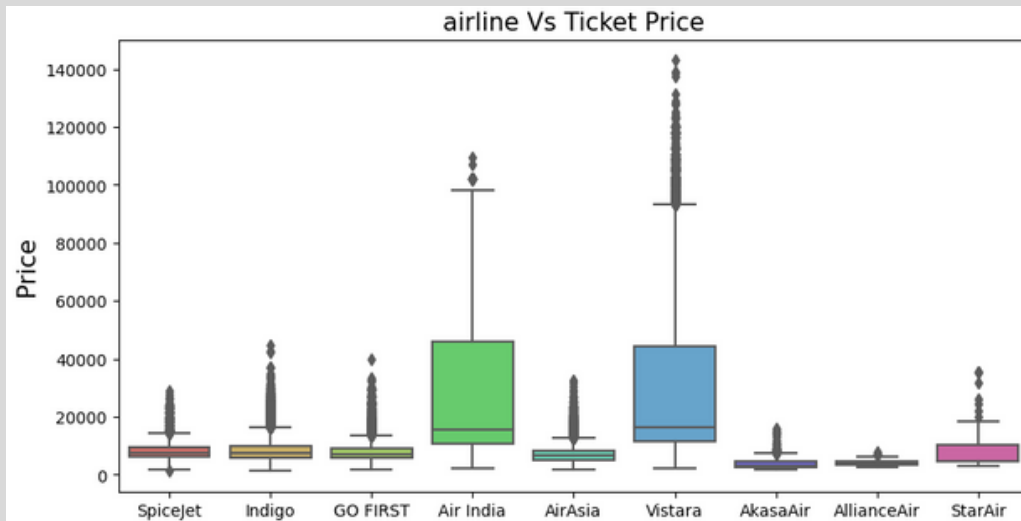
next we have checked the variation of fare with the classes of flights



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



# Low level design (LLD)



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 comparatively

```
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

## Low level design (LLD)

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

# Low level design (LLD)

```
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

# Low level design (LLD)

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

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

Onehot encoder is used to encode all columns 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)
```

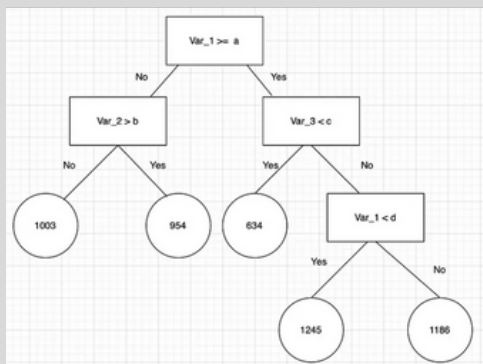
## 2.4 Model training and pickling

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.



# Low level design (LLD)

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)
```

# Low level design (LLD)

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

# Low level design (LLD)

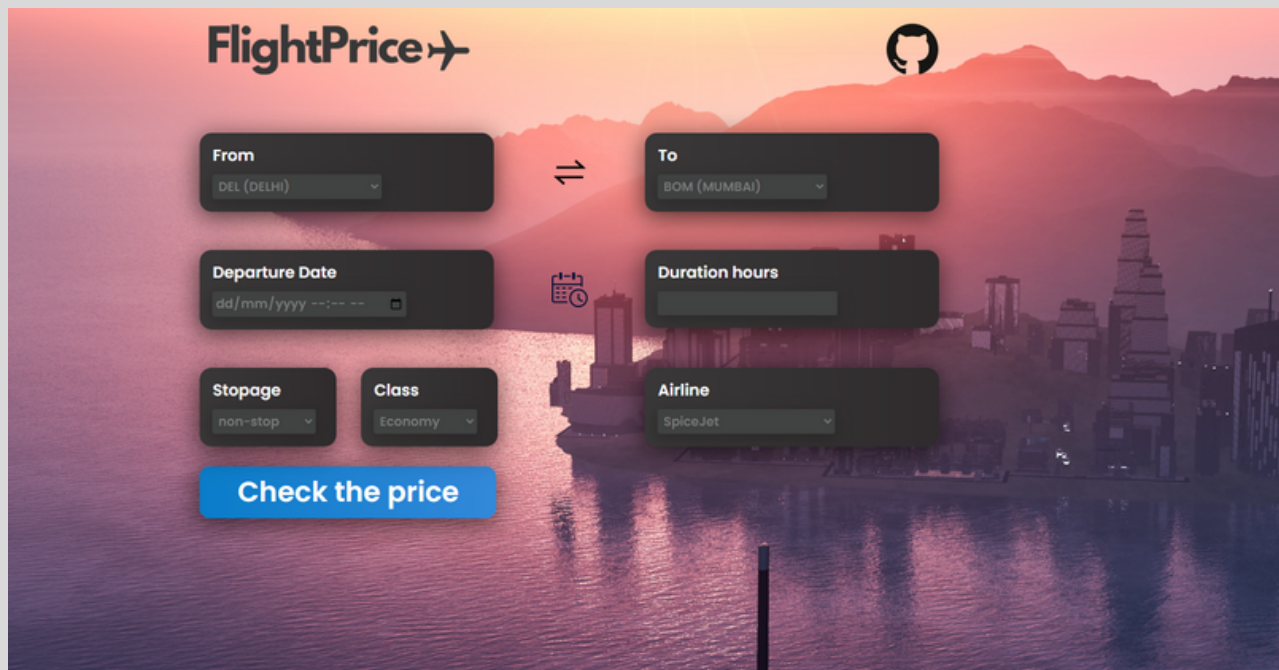
## 3. User interface (UI)

### 3.1 Landing page introduction

As our flask web app contains only one page that acts as the whole user interface to our machine learning app

the page is designed using **HTML, CSS and bootstrap framework**

we have also used jinja templates inside our web page

The image shows a web application interface for flight pricing. At the top left is the logo "FlightPrice" with a plane icon. At the top right is a GitHub logo. The main form consists of several input fields: "From" (with a dropdown menu showing "DEL (DELHI)"), "To" (with a dropdown menu showing "BOM (MUMBAI)"), "Departure Date" (with a date picker showing "dd/mm/yyyy --:-- --"), "Duration hours" (with a text input field), "Stopage" (with a dropdown menu showing "non-stop"), "Class" (with a dropdown menu showing "Economy"), and "Airline" (with a dropdown menu showing "SpiceJet"). A blue button labeled "Check the price" is at the bottom. The background is a scenic image of a city skyline at sunset.

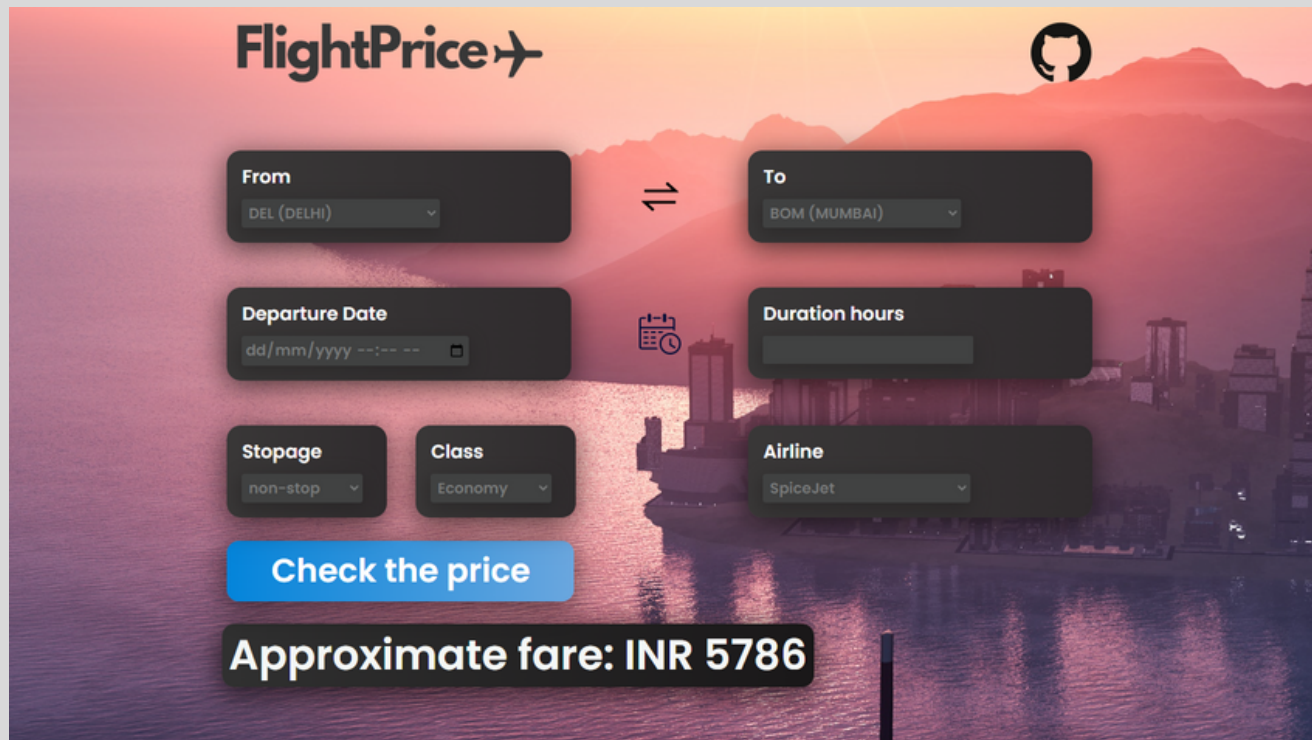
### 3.2 Inputting features

there are 7 input fields we have given . 5 of them are to select from multiple options . one is to select date and time . one is to input the duration of flight and this can be given as integer or floating point value

the button is provided as a submit button to make the prediction by redirecting to a python function. the predicted value will be appeared right below the submit button . "GET" method requests will be processed in python code and convert the necessary datas into the formats we need



# Low level design (LLD)



The screenshot shows a web application for flight booking. At the top, the logo "FlightPrice" with an airplane icon is on the left, and a GitHub icon is on the right. The main form consists of several input fields: "From" (DEL (DELHI)), "To" (BOM (MUMBAI)), "Departure Date" (dd/mm/yyyy), "Duration hours", "Stopage" (non-stop), "Class" (Economy), and "Airline" (SpiceJet). A blue button labeled "Check the price" is positioned below the form. At the bottom, a black box displays the "Approximate fare: INR 5786". The background features a scenic view of a city skyline at sunset.

## 3.3 Usage of web frameworks

We have used Flask , HTML , CSS and bootstrap to create the user interface , ie a web app

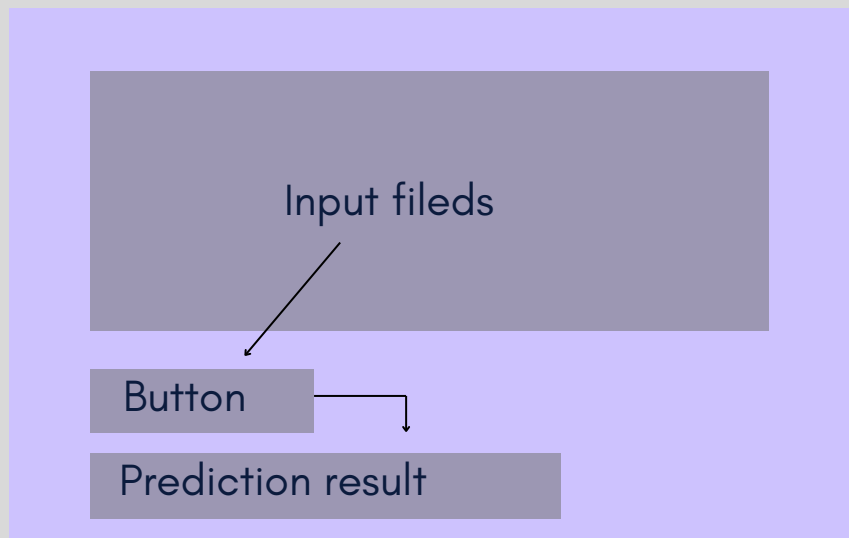
**Flask** is a micro web framework written in Python. It is designed to be simple and lightweight, making it easy to use for small to medium-sized web applications. Flask is based on the Werkzeug toolkit and the Jinja2 templating engine, and it provides a lot of built-in features and tools that make it easy to build web applications quickly

**Bootstrap** is a popular open-source front-end web development framework that provides developers with a set of pre-designed HTML, CSS, and JavaScript templates, components, and tools for building responsive and mobile-first websites and web applications. Bootstrap was initially developed by Twitter, and it is now maintained by the Bootstrap team and a large community of developers.

# Low level design (LLD)

## 3.4 Navigation layout

As shown in the screenshot , the navigation very simple in our application . input fields are given in grids. which is easy to understand. there is only one action a user can , that is submitting the data to make the prediction . the button is responsible to submit the data to make prediction



## 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