



Data Science Intern at Data Glacier

Week 4: Deployment on Flask

- Name: Salih Eren Yüzbaşıoğlu
- Batch Code: LISUM20
- Date: 12 May 2023
- Submitted to: Data Glacier

Table of Contents:

1. Introduction	3
2. Data Information	4
>2.1 Attribute Information	4
3. Build Machine Learning Model	5
4. Turning Model into Flask Framework	6
>4.1 App.py	7
>4.2 Home.html.....	8
>4.3 request.html	9
>4.4 Running Procedure	10

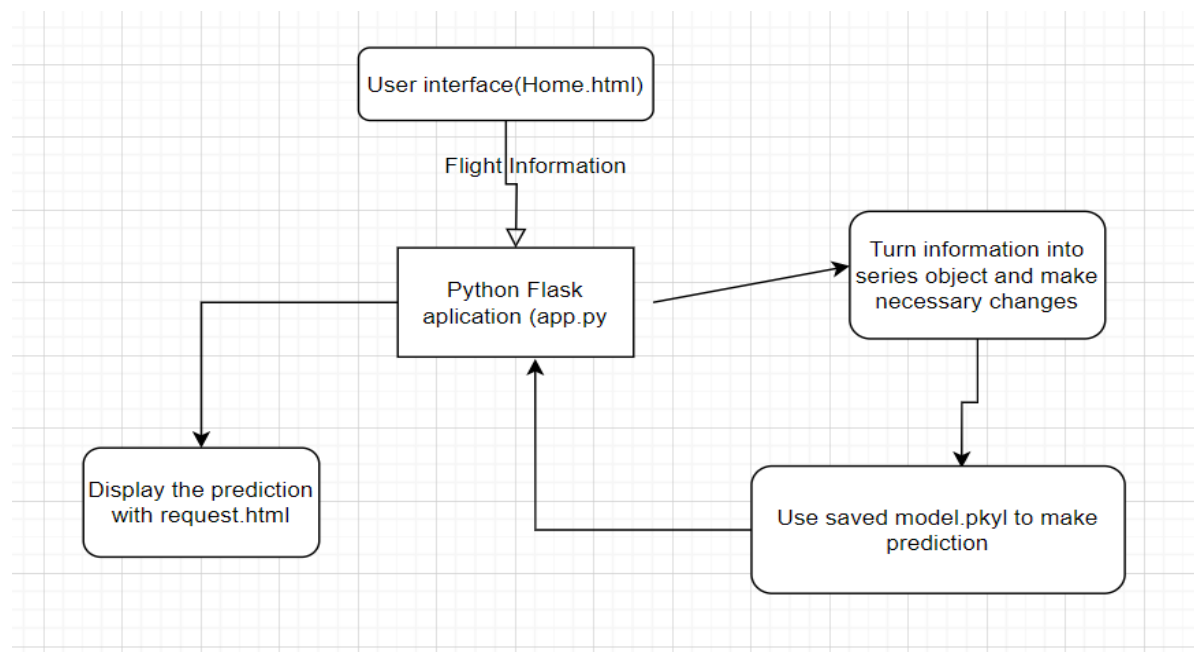
1. Introduction

This project involves deploying a machine learning model for flight price prediction using the Flask web framework. The model we will be using is a random forest regressor that has been trained on a dataset of flight prices, departure and arrival times, airlines, and other relevant features.

The goal of this project is to create a web application that allows users to input information about a flight, such as the departure and arrival cities, departure and return dates, and number of passengers, and get a predicted flight price based on the trained random forest model.

To build the random forest model, we will be using Python and several libraries, including NumPy, Pandas, and scikit-learn. We will also be using the Pickle library to serialize the trained model and save it for later use.

In addition to building the machine learning model, we will be using the Flask framework to create a web application that integrates the model and allows users to interact with it through a user-friendly web interface.



2. Data Information

The sample were extracted from the link

<https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction>.

DATA COLLECTION AND METHODOLOGY

Octoparse scraping tool was used to extract data from the website. Data was collected in two parts: one for economy class tickets and another for business class tickets. A total of 300261 distinct flight booking options were extracted from the site. Data was collected for 50 days, from February 11th to March 31st, 2022.

Data source was secondary data and was collected from Ease my trip website.

DATASET

Dataset contains information about flight booking options from the website Easemytrip for flight travel between India's top 6 metro cities. There are 300261 datapoints and 11 features in the cleaned dataset.

2.1 Attribute Information

The various features of the cleaned dataset are explained below:

- 1) Airline: The name of the airline company is stored in the airline column. It is a categorical feature having 6 different airlines.
- 2) Flight: Flight stores information regarding the plane's flight code. It is a categorical feature.
- 3) Source City: City from which the flight takes off. It is a categorical feature having 6 unique cities.
- 4) Departure Time: This is a derived categorical feature obtained created by grouping time periods into bins. It stores information about the departure time and have 6 unique time labels.
- 5) Stops: A categorical feature with 3 distinct values that stores the number of stops between the source and destination cities.
- 6) Arrival Time: This is a derived categorical feature created by grouping time intervals into bins. It has six distinct time labels and keeps information about the arrival time.

7) Destination City: City where the flight will land. It is a categorical feature having 6 unique cities.

8) Class: A categorical feature that contains information on seat class; it has two distinct values: Business and Economy.

9) Duration: A continuous feature that displays the overall amount of time it takes to travel between cities in hours.

10) Days Left: This is a derived characteristic that is calculated by subtracting the trip date by the booking date.

11) Price: Target variable stores information of the ticket price.

Sample was .csv file which I turned into a data frame here is the head of data frame.

```
Data.head()
```

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956
3	3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955
4	4	Vistara	UK-963	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955

3. Building a Model

3.1 Cleaning and Updating the Data

Deleting unnecessary columns.

```
Data = Data.drop("Unnamed: 0",axis=1)
Data = Data.drop("flight",axis=1)
Data.head()
```

	airline	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
0	SpiceJet	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953
1	SpiceJet	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953
2	AirAsia	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956
3	Vistara	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955
4	Vistara	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955

Changing stops and class columns into numbers.

```
def stopChanger(a):
    if a=="zero":
        return 0
    elif a=="one":
        return 1
    else:
        return 2
Data["stops"] = Data["stops"].apply(lambda x: stopChanger(x))
Data["source_city"].value_counts()
```

```
Delhi      61343
Mumbai     60896
Bangalore  52061
Kolkata    46347
Hyderabad  40806
Chennai    38700
Name: source_city, dtype: int64
```

```
# create a dictionary to map each category to an integer
mapping = {'Economy': 0, 'Business': 1}

# replace each category with its corresponding integer
Data["class"] = Data["class"].replace(mapping)
```

We create dummies for airline, arrival time and departure time since they are not connected to price in linear way. We also create dummies for city pairs with source and destination total of 25 dummies. We did not do 5 dummies for each because source city is not important if it is not connected with destination city since every pair has its own route with different length and different condition.

```
: # create a new DataFrame with a binary indicator variable for each category
one_hot = pd.get_dummies(Data["airline"], prefix='airline')

# concatenate the new DataFrame with the original DataFrame
Data = pd.concat([Data, one_hot], axis=1)

# drop the original "arrival_time" column
Data = Data.drop("airline", axis=1)

: one_hot = pd.get_dummies(Data["arrival_time"], prefix='arrival_time')
Data = pd.concat([Data, one_hot], axis=1)
Data = Data.drop("arrival_time", axis=1)
one_hot = pd.get_dummies(Data["departure_time"], prefix='departure_time')
Data = pd.concat([Data, one_hot], axis=1)
Data = Data.drop("departure_time", axis=1)

: # create a new column that contains the combination of "source_city" and "destination_city"
Data["city_pair"] = Data["source_city"] + "_" + Data["destination_city"]

# create a new DataFrame with a binary indicator variable for each city pair
one_hot = pd.get_dummies(Data["city_pair"], prefix='city_pair')

# concatenate the new DataFrame with the original DataFrame
Data = pd.concat([Data, one_hot], axis=1)

# drop the original "source_city", "destination_city", and "city_pair" columns
Data = Data.drop(["source_city", "destination_city", "city_pair"], axis=1)
```

3.2 Machine Learning Model

Training the model. We select such small test size because our main goal is not to test it in notebook well but to test it well in web app.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.01, random_state = 42)
```

```
from sklearn.ensemble import RandomForestRegressor

# Get the best model
reg_rf_meta = RandomForestRegressor(n_jobs=-1, verbose=2)

# Refit on the entire training set
reg_rf_meta.fit(X_train, y_train)

# Make a prediction for a new flight with a given feature vector X_test and price y_test
```

Scores of the prediction.

```
y_pred = reg_rf_meta.predict(X_test)

[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent workers.
[Parallel(n_jobs=12)]: Done 17 tasks | elapsed: 0.0s
[Parallel(n_jobs=12)]: Done 100 out of 100 | elapsed: 0.0s finished
```

```
reg_rf_meta.score(X_train, y_train)

[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent workers.
[Parallel(n_jobs=12)]: Done 17 tasks | elapsed: 0.9s
[Parallel(n_jobs=12)]: Done 100 out of 100 | elapsed: 3.6s finished

0.9973934757224987
```

```
reg_rf_meta.score(X_test, y_test)

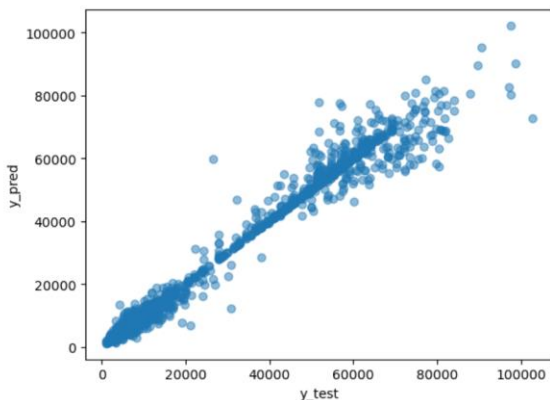
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent workers.
[Parallel(n_jobs=12)]: Done 17 tasks | elapsed: 0.0s
[Parallel(n_jobs=12)]: Done 100 out of 100 | elapsed: 0.0s finished

0.9860011948027996
```

The machine learning model used for flight price prediction takes nine input features into account, namely stop count, class type, days left, departure time, arrival time, destination city, source city, duration of flight, and airline. These features are used to predict the price of a flight ticket. However, it's worth noting that the absence of the departure day and month information may have an impact on the model's accuracy. This is because the price of a flight ticket can vary significantly depending on the time of year and the day of the week, which is typically associated with the level of demand. Without this information, the model may not be able to capture these fluctuations accurately. Nonetheless, over 0.986 score is amazing for a model that does not know the day or the month.

Scatter Plot

```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



We can see that between 0 and 20000 and 50000 and 90000 model has higher standard deviation.

Error metrics and model dumping:

```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 1006.1556322927572
MSE: 7259616.200890675
RMSE: 2694.3674955155384
```

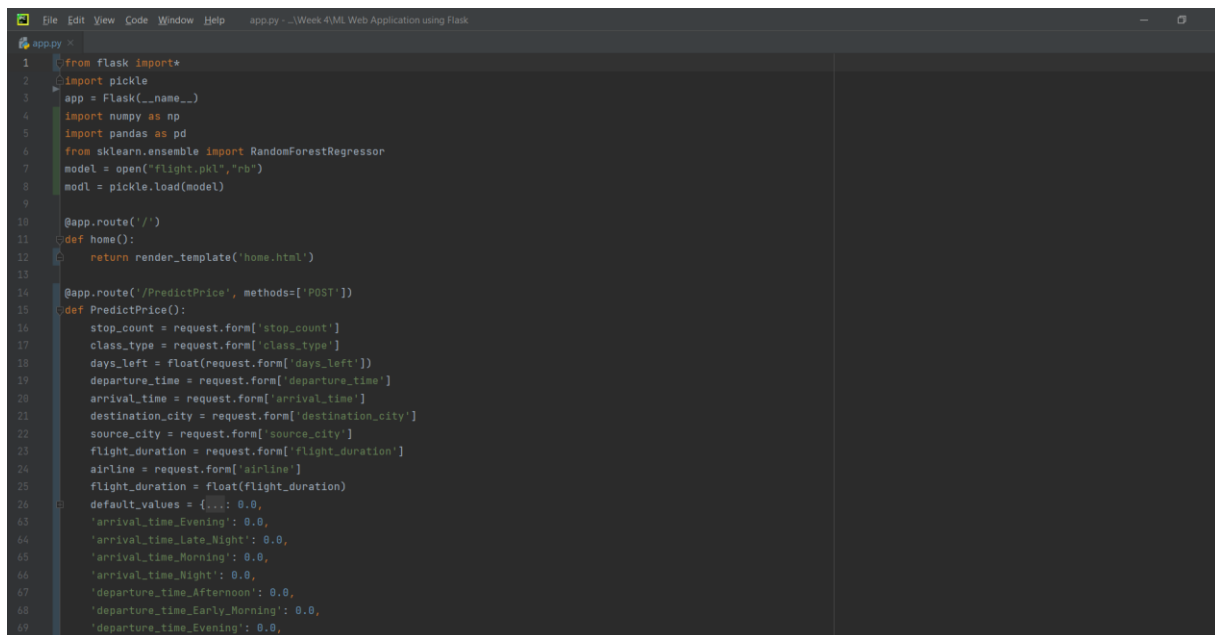
```
import pickle
with open('flight.pkl', 'wb') as f:
    pickle.dump(reg_rf_meta, f)
```

4. Turning Model into Web Application

We develop a web application that consists of a simple web page with a form field that lets us enter a message. After submitting the message to the web application, it will render it on a new page which gives us a result of flight prediction.

4.1 App.py

The app.py file contains the main code that will be executed by the Python interpreter to run the Flask web application.



```
1 from flask import*
2 import pickle
3 app = Flask(__name__)
4 import numpy as np
5 import pandas as pd
6 from sklearn.ensemble import RandomForestRegressor
7 model = open("flight.pkl","rb")
8 mdl = pickle.load(model)
9
10 @app.route('/')
11 def home():
12     return render_template('home.html')
13
14 @app.route('/PredictPrice', methods=['POST'])
15 def PredictPrice():
16     stop_count = request.form['stop_count']
17     class_type = request.form['class_type']
18     days_left = float(request.form['days_left'])
19     departure_time = request.form['departure_time']
20     arrival_time = request.form['arrival_time']
21     destination_city = request.form['destination_city']
22     source_city = request.form['source_city']
23     flight_duration = request.form['flight_duration']
24     airline = request.form['airline']
25     flight_duration = float(flight_duration)
26     default_values = {'stop_count': 0.0,
27                       'arrival_time_Evening': 0.0,
28                       'arrival_time_Late_Night': 0.0,
29                       'arrival_time_Morning': 0.0,
30                       'arrival_time_Night': 0.0,
31                       'departure_time_Afternoon': 0.0,
32                       'departure_time_Early_Morning': 0.0,
33                       'departure_time_Evening': 0.0}
```



```
File Edit View Code Window Help app.py - Week 4 ML Web Application using Flask
76 'airline_indigo': 0.0,
77 'airline_spicejet': 0.0,
78 'airline_vistara': 0.0
79 }
80 def stopChanger(a):
81     if a=="zero":
82         return 0
83     elif a=="one":
84         return 1
85     else:
86         return 2
87 mapping = {'Economy': 0, 'Business': 1}
88 default_values = pd.Series(default_values)
89 default_values.loc['airline_'+ airline]=1.0
90 default_values.loc['city_pair_'+ source_city+'_'+destination_city]=1.0
91 default_values.loc['departure_time_'+ departure_time]=1.0
92 default_values.loc['stops'] = stopChanger(stop_count)
93 default_values.loc['arrival_time_'+ arrival_time]=1
94 default_values.loc['class']=mapping[class_type]
95 default_values.loc['duration']= flight_duration
96 default_values.loc['days_left']= days_left
97 print(default_values)
98 if request.method=="POST":
99     prediction = modl.predict(default_values.to_frame().T)
100
101     prediction = prediction/82
102     prediction = float(prediction)
103     return render_template('request.html', y=format(prediction, '.2f'))
104
105 if __name__ == "__main__":
106     app.run(debug=True)
```

- The first few lines import the necessary libraries, including Flask, NumPy, Pandas, scikit-learn, and Pickle.
- The code then opens a trained machine learning model from a file using the Pickle library.
- Two routes are defined using the "@app.route" decorator. The first route ("/") renders a template for the home page of the web application. The second route ("/PredictPrice") accepts a POST request containing form data submitted by the user, and uses the trained machine learning model to predict the flight price based on the input data.
- In the "/PredictPrice" route, the form data submitted by the user is extracted using the "request.form" object, and is then processed and transformed into a format suitable for input into the trained machine learning model.
- The transformed data is then used to make a prediction using the trained machine learning model, and the resulting predicted flight price is returned to the user as a response to the original POST request.
- The final few lines start the Flask application and run it in debug mode.

4.2 Home.html

```

<!DOCTYPE html>
<html>
<head>
    <title>Flight Search</title>
</head>
<body>
    <h1>Flight Price Guesser</h1>
    <form action="/PredictPrice" method="post">
        <label for="stop_count">Stop Count:</label>
        <select name="stop_count" id="stop_count">
            <option value="zero">0 stop</option>
            <option value="one">1 stop</option>
            <option value="2">2 or more stops</option>
        </select><br><br>

        <label for="class_type">Class Type:</label>
        <select name="class_type" id="class_type">
            <option value="Economy">Economy</option>
            <option value="Business">Business</option>
        </select><br><br>

        <label for="days_left">Days Left Until Flight:</label>
        <input type="number" name="days_left" id="days_left" min="1" max="49"><br><br>

        <label for="departure_time">Departure Time:</label>
        <select name="departure_time" id="departure_time">
            <option value="Morning">Morning</option>
            <option value="Early_Morning">Early Morning</option>
            <option value="Afternoon">Afternoon</option>
            <option value="Evening">Evening</option>
            <option value="Night">Night</option>
            <option value="Late_Night">Late Night</option>
        </select><br><br>

        <label for="arrival_time">Arrival Time:</label>
        <select name="arrival_time" id="arrival_time">
            <option value="Morning">Morning</option>
            <option value="Early_Morning">Early Morning</option>
            <option value="Afternoon">Afternoon</option>
            <option value="Evening">Evening</option>
            <option value="Night">Night</option>
            <option value="Late_Night">Late Night</option>
        </select><br><br>

        <label for="destination_city">Destination City:</label>
        <select name="destination_city" id="destination_city">
            <option value="Delhi">Delhi</option>
            <option value="Mumbai">Mumbai</option>
            <option value="Bangalore">Bangalore</option>
            <option value="Kolkata">Kolkata</option>
            <option value="Hyderabad">Hyderabad</option>
            <option value="Chennai">Chennai</option>
        </select><br><br>
    </form>

```

```

<label for="arrival_time">Arrival Time:</label>
<select name="arrival_time" id="arrival_time">
  <option value="Morning">Morning</option>
  <option value="Early_Morning">Early Morning</option>
  <option value="Afternoon">Afternoon</option>
  <option value="Evening">Evening</option>
  <option value="Night">Night</option>
  <option value="Late_Night">Late Night</option>
</select><br><br>

<label for="destination_city">Destination City:</label>
<select name="destination_city" id="destination_city">
  <option value="Delhi">Delhi</option>
  <option value="Mumbai">Mumbai</option>
  <option value="Bangalore">Bangalore</option>
  <option value="Kolkata">Kolkata</option>
  <option value="Hyderabad">Hyderabad</option>
  <option value="Chennai">Chennai</option>
</select><br><br>

<label for="source_city">Source City:</label>
<select name="source_city" id="source_city">
  <option value="Delhi">Delhi</option>
  <option value="Mumbai">Mumbai</option>
  <option value="Bangalore">Bangalore</option>
  <option value="Kolkata">Kolkata</option>
  <option value="Hyderabad">Hyderabad</option>
  <option value="Chennai">Chennai</option>
</select><br><br>

<label for="flight_duration">Flight Duration (hours):</label>
<input type="number" name="flight_duration" id="flight_duration" min="0" step="0.01"><br><br>

<label for="airline">Airline:</label>
<select name="airline" id="airline">
  <option value="Vistara">Vistara</option>
  <option value="Air_India">Air India</option>
  <option value="Indigo">Indigo</option>
  <option value="GO_FIRST">GO FIRST</option>
  <option value="AirAsia">AirAsia</option>
  <option value="SpiceJet">SpiceJet</option>
</select><br><br>

<input type="submit" value="Search">
</form>
</body>
</html>

```

- The document starts with a DOCTYPE declaration that tells the web browser what type of document to expect.
- The HTML document is contained within <html> and </html> tags.
- The <head> section contains metadata about the document, including the title of the page (which appears in the browser's title bar).
- The <body> section contains the main content of the document, which in this case is a form for users to input data.
- The <h1> tag creates a heading that says "Flight Price Guesser".
- The <form> tag creates a form that allows users to input data for the machine learning model. The "action" attribute specifies where the form data should be sent when the user submits the form (in this case, to the "/PredictPrice" route of the Flask application). The "method" attribute specifies the HTTP method to use when submitting the form data (in this case, "POST").
- Within the <form> tag, there are several <label> tags that create labels for the input fields in the form. Each <label> tag is associated with an <input> or <select> tag that creates the corresponding form field.
- The various <select> tags create dropdown menus that allow users to select options for the flight's stop count, class type, departure time, arrival time, destination city, source city, and airline.
- The <input> tags create fields for users to input the number of days left until the flight and the flight duration in hours.

- The final `<input>` tag creates a submit button that the user can click to submit the form data.
- The `</form>` tag closes the form.
- Finally, the `</body>` and `</html>` tags close the document.

4.2 request.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Flight Price Guesser - Prediction Result</title>
</head>
<body>
  <h1>Flight Price Guesser - Prediction Result</h1>
  <p>Price Prediction: {{ y }}$</p>
</body>
</html>
```

- The document starts with a DOCTYPE declaration, indicating that it is an HTML document.
- The `<html>` tag indicates the start of the HTML document, while the `<head>` and `<body>` tags provide the structure for the document.
- The `<title>` tag sets the title of the HTML document to "Flight Price Guesser - Prediction Result".
- The `<h1>` tag creates a heading that says "Flight Price Guesser - Prediction Result".
- The `<p>` tag creates a paragraph that displays the predicted flight price. The predicted flight price is inserted into the HTML template using Flask's templating engine, which uses the double curly braces `{{ }}` to indicate a variable. In this case, the variable "y" contains the predicted flight price returned by the model. The dollar sign "\$" is added after the price to indicate the currency.
- Finally, the `</body>` and `</html>` tags close the HTML document.

4.3 Running Procedure

Once we have done all the above, we can start running the API by executing the command from the Terminal:

```
PS C:\Users\Salih\Desktop\DataGlacier\DataGlacierInternship\Week 4\ML Web Application using Flask> python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 129-526-058
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now we could open a web browser and navigate to <http://127.0.0.1:5000/>, we should see a simple website with the content like so.



The screenshot shows a web browser window with the address bar displaying <http://127.0.0.1:5000/>. The page title is "Flight Price Guesser". The form contains the following fields and controls:

- Stop Count: 2 or more stops (dropdown menu)
- Class Type: Economy (dropdown menu)
- Days Left Until Flight: 12 (text input)
- Departure Time: Afternoon (dropdown menu)
- Arrival Time: Afternoon (dropdown menu)
- Destination City: Bangalore (dropdown menu)
- Source City: Chennai (dropdown menu)
- Flight Duration (hours): 12 (text input)
- Airline: GO FIRST (dropdown menu)
- Search button

Output Page:



The screenshot shows a web browser window with the address bar displaying <http://127.0.0.1:5000/PredictPrice>. The page title is "Flight Price Guesser - Prediction Result". The output text is:

Price Prediction: 98.205