✈ Flight Fare Prediction using Machine Learning

Problem Statement

Flight fare prediction involves applying data science and machine learning to estimate the cost of airline tickets based on various flight details. These include airline, travel date, source and destination cities, number of stops, and duration. By using statistical modeling on historical flight data, machine learning models can identify pricing patterns and forecast ticket fares. This enables smarter travel planning and supports dynamic pricing strategies.

Target:

a) Design a predictive model using machine learning algorithms to forecast the ticket fare of a flight based on its features – this is a regression problem.

b) Design an analytical model to identify and analyze the most significant factors influencing flight fares, helping airlines and travelers understand pricing trends – this is an insight-driven exploratory analysis task.

Dataset Description:

Column Name	Description
`Unnamed: 0`	Index/serial number, not relevant for analysis
`airline`	Airline name (e.g., Vistara, IndiGo, etc.)
`flight`	Flight number (e.g., UK-706)
`source_city`	City from which the flight departs
`departure_time`	Time of the day when flight departs (Morning, Evening, etc.)
`stops`	Number of stops (non-stop, one stop, two or more)
`arrival_time`	Time of the day when flight arrives
`destination_city`	City where flight lands
`class`	Travel class (`Economy` , `Business`)
`duration`	Total duration of flight (in hours, float)
`days_left`	Days left for departure from the date of booking
`price`	**Target Variable** – Flight fare (in INR)

Importing Necessary Libraies

In [34]:

```
#Data Manipulation Libraries
import pandas as pd
import numpy as np

# Data  Visualization Libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Warning Libraries
import warnings
warnings.filterwarnings("ignore")

# Data preprocessing Libraries
from sklearn.preprocessing import OneHotEncoder
OHE = OneHotEncoder()

from sklearn.preprocessing import OrdinalEncoder
ORE = OrdinalEncoder()

# Model_selection Libraries
from sklearn.model_selection import train_test_split

# Machine Learning Model
from sklearn.ensemble import RandomForestRegressor
LAR = RandomForestRegressor()
# RFR = RandomForestRegressor()
from sklearn.linear_model import Lasso
# LAR = Lasso(alpha=0.1)

# Evaluation Metrics
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

# GUI Libraries
import tkinter as tk
from tkinter import ttk, messagebox
```

Importing Dataset

In [35]:

```
airline_dataset = pd.read_csv("AirlinesDataset.csv")
```

Exploratory Data Analysis

Dataset before elimination of uncessunnecessary columns

In [36]:

```
airline_dataset
```

Out[36]:

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_le
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	
3	3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	
4	4	Vistara	UK-963	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	
...	...	...	...	...	...	...	...	...	...	...	.
300148	300148	Vistara	UK-822	Chennai	Morning	one	Evening	Hyderabad	Business	10.08	4
300149	300149	Vistara	UK-826	Chennai	Afternoon	one	Night	Hyderabad	Business	10.42	4
300150	300150	Vistara	UK-832	Chennai	Early_Morning	one	Night	Hyderabad	Business	13.83	4
300151	300151	Vistara	UK-828	Chennai	Early_Morning	one	Evening	Hyderabad	Business	10.00	4
300152	300152	Vistara	UK-822	Chennai	Morning	one	Evening	Hyderabad	Business	10.08	4

300153 rows × 12 columns

*The columns named 'Unnamed: 0' and 'flight' are not required for training this machine learning model, so they have been eliminated.

Eliminating Uncessunnecessary columns from the dataset

In [37]:

```
airline_dataset = airline_dataset.drop("Unnamed: 0",axis=1)
airline_dataset = airline_dataset.drop("flight",axis=1)
airline_dataset = airline_dataset.drop("duration",axis=1)
airline_dataset = airline_dataset.drop("arrival_time",axis=1)
```

Checking first 10 entries of the dataset -

In [38]:

```
airline_dataset.head(10)
```

Out[38]:

	airline	source_city	departure_time	stops	destination_city	class	days_left	price
0	SpiceJet	Delhi	Evening	zero	Mumbai	Economy	1	5953
1	SpiceJet	Delhi	Early_Morning	zero	Mumbai	Economy	1	5953
2	AirAsia	Delhi	Early_Morning	zero	Mumbai	Economy	1	5956
3	Vistara	Delhi	Morning	zero	Mumbai	Economy	1	5955
4	Vistara	Delhi	Morning	zero	Mumbai	Economy	1	5955
5	Vistara	Delhi	Morning	zero	Mumbai	Economy	1	5955
6	Vistara	Delhi	Morning	zero	Mumbai	Economy	1	6060
7	Vistara	Delhi	Afternoon	zero	Mumbai	Economy	1	6060
8	GO_FIRST	Delhi	Early_Morning	zero	Mumbai	Economy	1	5954
9	GO_FIRST	Delhi	Afternoon	zero	Mumbai	Economy	1	5954

Checking last 10 entries of the dataset -

In [39]:
```
airline_dataset.tail(10)
```

Out[39]:

	airline	source_city	departure_time	stops	destination_city	class	days_left	price
300143	Air_India	Chennai	Early_Morning	one	Hyderabad	Business	49	51345
300144	Air_India	Chennai	Evening	one	Hyderabad	Business	49	51345
300145	Air_India	Chennai	Morning	one	Hyderabad	Business	49	51345
300146	Air_India	Chennai	Early_Morning	one	Hyderabad	Business	49	51345
300147	Air_India	Chennai	Early_Morning	one	Hyderabad	Business	49	68739
300148	Vistara	Chennai	Morning	one	Hyderabad	Business	49	69265
300149	Vistara	Chennai	Afternoon	one	Hyderabad	Business	49	77105
300150	Vistara	Chennai	Early_Morning	one	Hyderabad	Business	49	79099
300151	Vistara	Chennai	Early_Morning	one	Hyderabad	Business	49	81585
300152	Vistara	Chennai	Morning	one	Hyderabad	Business	49	81585

Shape of the Dataset -

In [40]:
```
print("Shape of the above dataset:",airline_dataset.shape)
print("Numbers of columns:",12)
print("Numbers of Rows:",300153)
```

```
Shape of the above dataset: (300153, 8)
Numbers of columns: 12
Numbers of Rows: 300153
```

Columns Names -

In [41]:
```
airline_dataset.columns
```

Out[41]:
```
Index(['airline', 'source_city', 'departure_time', 'stops', 'destination_city',
       'class', 'days_left', 'price'],
      dtype='object')
```

Checking for null or missing values -

In [42]:
```
airline_dataset.isnull().sum()
```

Out[42]:
```
airline             0
source_city         0
departure_time      0
stops               0
destination_city    0
class               0
days_left           0
price               0
dtype: int64
```

Obtaining some information about the dataset -

In [43]:
```
airline_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype 
---  ------            --------------   ----- 
 0   airline           300153 non-null  object
 1   source_city       300153 non-null  object
 2   departure_time    300153 non-null  object
 3   stops             300153 non-null  object
 4   destination_city  300153 non-null  object
 5   class             300153 non-null  object
 6   days_left         300153 non-null  int64 
 7   price             300153 non-null  int64 
dtypes: int64(2), object(6)
memory usage: 18.3+ MB
```

Datatypes present in the dataset -

In [44]:
```
airline_dataset.dtypes
```

Out[44]:

```
airline             object
source_city         object
departure_time      object
stops               object
destination_city    object
class               object
days_left            int64
price                int64
dtype: object
```

Separating categorical columns from dataset

In [45]:

```
categorical_Columns = []
for i in airline_dataset.columns:
    if airline_dataset[i].dtype == object:
        categorical_Columns.append(i)
categorical_Columns
```

Out[45]:

```
['airline',
 'source_city',
 'departure_time',
 'stops',
 'destination_city',
 'class']
```

Separating numerical columns from dataset

In [46]:

```
numerical_Columns = []
for i in airline_dataset.columns:
    if airline_dataset[i].dtype != object:
        numerical_Columns.append(i)
numerical_Columns
```

Out[46]:

```
['days_left', 'price']
```

Merging both columns for Checking distinct values

In [47]:

```
combined_columns = categorical_Columns + numerical_Columns
combined_columns
```

Out[47]:

```
['airline',
 'source_city',
 'departure_time',
 'stops',
 'destination_city',
 'class',
 'days_left',
 'price']
```

Checking distinct values in each column

In [48]:

```
value_counts_dict = {}
for column in airline_dataset.columns:
    value_counts_dict[column] = airline_dataset[column].value_counts()
for column, counts in value_counts_dict.items():
    print("-" *(len(column)+4) )
    print("|",column,"|")
    print("-" *(len(column)+4) )
    print(counts)
    print("================")
```

```
-----------
| airline |
-----------
airline
Vistara      127859
Air_India     80892
Indigo        43120
GO_FIRST      23173
AirAsia       16098
SpiceJet       9011
Name: count, dtype: int64
================
---------------
| source_city |
---------------
source_city
Delhi        61343
Mumbai       60896
Bangalore    52061
Kolkata      46347
Hyderabad    40806
```

```
Chennai      38700
Name: count, dtype: int64
================
------------------
| departure_time |
------------------
departure_time
Morning          71146
Early_Morning    66790
Evening          65102
Night            48015
Afternoon        47794
Late_Night        1306
Name: count, dtype: int64
================
---------
| stops |
---------
stops
one            250863
zero            36004
two_or_more     13286
Name: count, dtype: int64
================
--------------------
| destination_city |
--------------------
destination_city
Mumbai       59097
Delhi        57360
Bangalore    51068
Kolkata      49534
Hyderabad    42726
Chennai      40368
Name: count, dtype: int64
================
---------
| class |
---------
class
Economy     206666
Business     93487
Name: count, dtype: int64
================
-------------
| days_left |
-------------
days_left
25    6633
18    6602
39    6593
32    6585
26    6573
24    6542
19    6537
31    6534
33    6532
40    6531
41    6525
28    6522
38    6512
20    6502
30    6501
42    6497
22    6494
36    6490
21    6479
37    6476
43    6472
44    6436
17    6419
11    6417
34    6412
13    6404
23    6401
29    6397
12    6381
27    6360
14    6349
15    6340
45    6314
35    6291
16    6272
```

```
46    6160
49    6154
48    6078
47    6069
10    5822
8     5767
6     5740
7     5703
9     5665
5     5392
4     5077
3     4248
2     4026
1     1927
Name: count, dtype: int64
================
---------
| price |
---------
price
54608    1445
2339     1442
54684    1390
60978    1383
60508    1230
         ...
10767       1
10946       1
12206       1
12972       1
5607        1
Name: count, Length: 12157, dtype: int64
================
```

Encoding the columns for better accuracy of the model

In [49]:

```python
airline_dataset_encoded = airline_dataset.copy()
```

*we applied One-Hot Encoding to the columns['airline', 'source_city', 'departure_time', 'arrival_time', 'destination_city'] because they are categorical features, and machine learning models require numerical input to perform calculations.

In [50]:

```python
OHE = OneHotEncoder(drop='first', sparse_output=False, handle_unknown='ignore')
onehot_encoding_columns = ['airline', 'source_city', 'departure_time', 'destination_city']
columns_to_encode = airline_dataset_encoded[onehot_encoding_columns]
encoded_array = OHE.fit_transform(columns_to_encode)
encoded_df = pd.DataFrame(encoded_array,columns=OHE.get_feature_names_out(onehot_encoding_columns),index=airlin
airline_dataset_encoded.drop(columns=onehot_encoding_columns, inplace=True)
airline_dataset_encoded = pd.concat([airline_dataset_encoded, encoded_df], axis=1)
airline_dataset_encoded.head(10)
```

Out[50]:

	stops	class	days_left	price	airline_Air_India	airline_GO_FIRST	airline_Indigo	airline_SpiceJet	airline_Vistara	source_city
0	zero	Economy	1	5953	0.0	0.0	0.0	1.0	0.0	
1	zero	Economy	1	5953	0.0	0.0	0.0	1.0	0.0	
2	zero	Economy	1	5956	0.0	0.0	0.0	0.0	0.0	
3	zero	Economy	1	5955	0.0	0.0	0.0	0.0	1.0	
4	zero	Economy	1	5955	0.0	0.0	0.0	0.0	1.0	
5	zero	Economy	1	5955	0.0	0.0	0.0	0.0	1.0	
6	zero	Economy	1	6060	0.0	0.0	0.0	0.0	1.0	
7	zero	Economy	1	6060	0.0	0.0	0.0	0.0	1.0	
8	zero	Economy	1	5954	0.0	1.0	0.0	0.0	0.0	
9	zero	Economy	1	5954	0.0	1.0	0.0	0.0	0.0	

10 rows × 24 columns

*We applied ordinal encoding because the number of stops has a natural increasing order that affects fare. This helps the model understand that more stops usually imply longer/cheaper flights.

In [51]:

```python
stops_order = [['zero', 'one', 'two_or_more']]
ORE = OrdinalEncoder(categories=stops_order)
airline_dataset_encoded['stops_encoded'] = ORE.fit_transform(airline_dataset_encoded[['stops']])
airline_dataset_encoded['stops'] = airline_dataset_encoded['stops_encoded']
airline_dataset_encoded.drop(columns='stops_encoded', inplace=True)
```

*We used ordinal encoding because flight classes (Economy < Business) have a clear price hierarchy. Encoding preserves this relationship so the model can learn its impact on fare.

In [52]:

```
class_order = [['Economy', 'Business']]
ORE_class = OrdinalEncoder(categories=class_order)
airline_dataset_encoded['class_encoded'] = ORE_class.fit_transform(airline_dataset_encoded[['class']])
airline_dataset_encoded['class'] = airline_dataset_encoded['class_encoded']
airline_dataset_encoded.drop(columns='class_encoded', inplace=True)
```

Random sample of the dataset just to ensure that the encoding is accurate

In [53]:

```
airline_dataset_encoded.sample(20)
```

Out[53]:

	stops	class	days_left	price	airline_Air_India	airline_GO_FIRST	airline_Indigo	airline_SpiceJet	airline_Vistara	source_
140908	1.0	0.0	39	4056	0.0	0.0	0.0	0.0	0.0	
213140	1.0	1.0	16	57017	0.0	0.0	0.0	0.0	1.0	
107132	2.0	0.0	32	7106	0.0	0.0	0.0	0.0	1.0	
58056	2.0	0.0	30	10006	0.0	0.0	0.0	0.0	1.0	
90207	1.0	0.0	32	2723	0.0	0.0	0.0	0.0	0.0	
64553	1.0	0.0	19	4772	0.0	0.0	1.0	0.0	0.0	
15502	1.0	0.0	28	5040	0.0	0.0	1.0	0.0	0.0	
70196	1.0	0.0	49	4977	1.0	0.0	0.0	0.0	0.0	
166593	1.0	0.0	13	4453	0.0	0.0	1.0	0.0	0.0	
234326	1.0	1.0	4	65517	0.0	0.0	0.0	0.0	1.0	
145441	2.0	0.0	27	7123	1.0	0.0	0.0	0.0	0.0	
209992	0.0	1.0	32	36712	0.0	0.0	0.0	0.0	1.0	
33430	1.0	0.0	36	6172	0.0	0.0	0.0	0.0	1.0	
22603	1.0	0.0	16	5686	1.0	0.0	0.0	0.0	0.0	
205510	1.0	0.0	36	3979	1.0	0.0	0.0	0.0	0.0	
136140	1.0	0.0	6	13524	1.0	0.0	0.0	0.0	0.0	
210486	1.0	1.0	36	50969	1.0	0.0	0.0	0.0	0.0	
119245	2.0	0.0	44	9847	0.0	0.0	0.0	0.0	1.0	
276517	0.0	1.0	36	24122	1.0	0.0	0.0	0.0	0.0	
219124	0.0	1.0	1	38470	0.0	0.0	0.0	0.0	1.0	

20 rows × 24 columns

Splitting features and target

In [54]:

```
X = airline_dataset_encoded.drop("price",axis=1)
Y = airline_dataset_encoded["price"]
```

Displaying the Features -

In [55]:

```
X
```

Out[55]:

	stops	class	days_left	airline_Air_India	airline_GO_FIRST	airline_Indigo	airline_SpiceJet	airline_Vistara	source_city_Che
0	0.0	0.0	1	0.0	0.0	0.0	1.0	0.0	
1	0.0	0.0	1	0.0	0.0	0.0	1.0	0.0	
2	0.0	0.0	1	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	1	0.0	0.0	0.0	0.0	1.0	
4	0.0	0.0	1	0.0	0.0	0.0	0.0	1.0	
...	...	...	...	...	...	...	...	...	
300148	1.0	1.0	49	0.0	0.0	0.0	0.0	1.0	
300149	1.0	1.0	49	0.0	0.0	0.0	0.0	1.0	
300150	1.0	1.0	49	0.0	0.0	0.0	0.0	1.0	
300151	1.0	1.0	49	0.0	0.0	0.0	0.0	1.0	
300152	1.0	1.0	49	0.0	0.0	0.0	0.0	1.0	

300153 rows × 23 columns

Displaying the Target -

In [56]:

```
Y
```

Out[56]:

```
0          5953
1          5953
2          5956
3          5955
4          5955
          ...
300148    69265
300149    77105
300150    79099
300151    81585
300152    81585
Name: price, Length: 300153, dtype: int64
```

Splitting Training and testing data

In [57]:

```
X_train , X_test , Y_train , Y_test  = train_test_split(X,Y , test_size= 0.2 ,random_state=14)
```

Training the model -

In [58]:

```
LAR.fit(X_train, Y_train)
```

Out[58]:

```
▾ RandomForestRegressor ⓘ ?

RandomForestRegressor()
```

Predicting the test data -

In [59]:

```
y_pred = LAR.predict(X_test)
```

Evaluating the accuracy of the model -

In [60]:

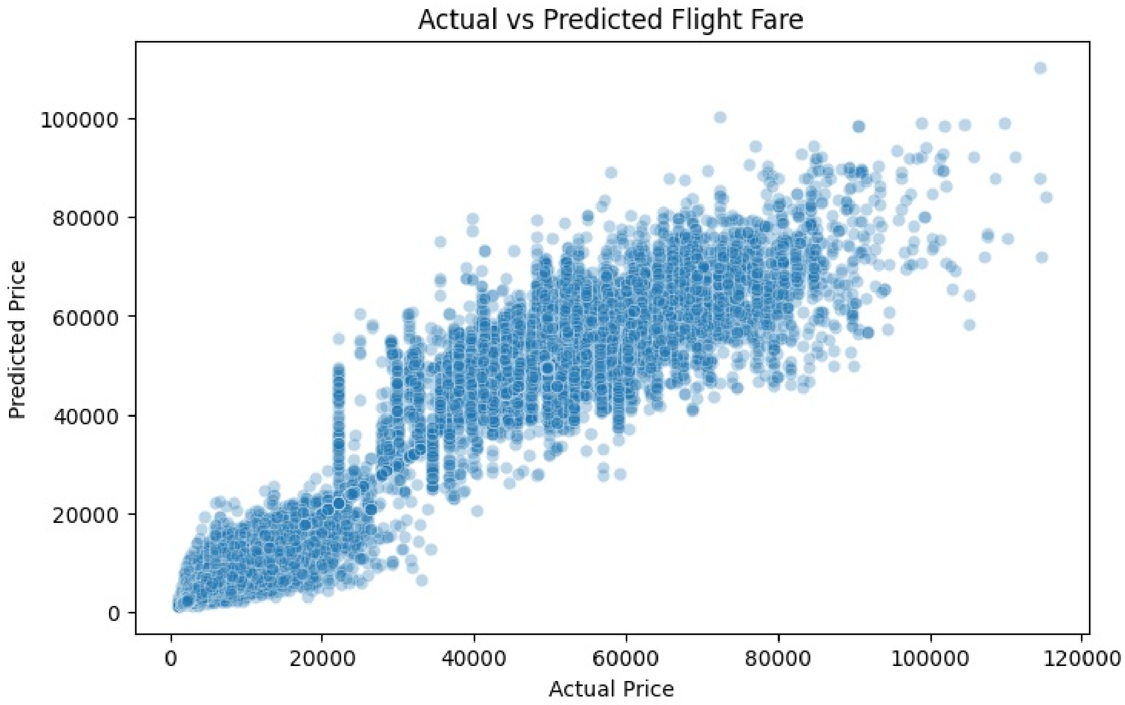
```
mse = mean_squared_error(Y_test, y_pred)
r2 = r2_score(Y_test, y_pred)

print("Mean Squared Error:", mse)
print("R² Score:", r2)
print("Predictions:",y_pred)
```

```
Mean Squared Error: 20425400.763711303
R² Score: 0.9604541327637137
Predictions: [ 8032.7136039  62495.6386014  44633.01467857 ... 54608.
  9110.571       7167.47266638]
```

In [61]:

```
plt.figure(figsize=(8,5))
sns.scatterplot(x=Y_test, y=y_pred, alpha=0.3)
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Flight Fare")
plt.show()
```



Predictive System -

In [62]:

```python
airline_categories = ['AirAsia', 'AirIndia', 'GoFirst', 'Indigo', 'SpiceJet', 'Vistara']
city_categories = ['Bangalore', 'Chennai', 'Delhi', 'Hyderabad', 'Kolkata', 'Mumbai']
time_categories = ['Early_Morning', 'Evening', 'Morning', 'Night']
stop_categories = ['zero', 'one', 'two_or_more']
class_categories = ['Economy', 'Business']

def predict_fare():
    try:
        airline = airline_var.get()
        source = source_var.get()
        departure = departure_var.get()
        stops = stops_var.get()
        dest = destination_var.get()
        f_class = class_var.get()
        days = int(days_left_var.get())

        input_df = pd.DataFrame([[airline, source, departure, dest]],
                                columns=['airline', 'source_city', 'departure_time', 'destination_city'])
        encoded_array = OHE.transform(input_df)
        encoded_df = pd.DataFrame(encoded_array,
                                  columns=OHE.get_feature_names_out(['airline', 'source_city', 'departure_time'
                                  index=[0])


        stops_encoded = ORE.transform([[stops]])[0][0]
        class_encoded = ORE_class.transform([[f_class]])[0][0]


        final_input = pd.DataFrame([[stops_encoded, class_encoded, days]], columns=['stops', 'class', 'days_lef
        final_input = pd.concat([final_input, encoded_df], axis=1)


        missing = set(X.columns) - set(final_input.columns)
        for col in missing:
            final_input[col] = 0
        final_input = final_input[X.columns]


        result = LAR.predict(final_input)[0]
        result_label.config(text=f"✈  Estimated Flight Fare: ₹{round(result, 2)}")

    except Exception as e:
        messagebox.showerror("Error", f"Something went wrong: {e}")
```

Graphical User Interface -

In [63]:

```python
root = tk.Tk()
root.title("Flight Fare Predictor")

tk.Label(root, text="Select Flight Details", font=("Arial", 14)).grid(row=0, column=0, columnspan=2, pady=10)

tk.Label(root, text="Airline").grid(row=1, column=0)
```

```python
airline_var = ttk.Combobox(root, values=airline_categories)
airline_var.grid(row=1, column=1)

tk.Label(root, text="Source City").grid(row=2, column=0)
source_var = ttk.Combobox(root, values=city_categories)
source_var.grid(row=2, column=1)

tk.Label(root, text="Departure Time").grid(row=3, column=0)
departure_var = ttk.Combobox(root, values=time_categories)
departure_var.grid(row=3, column=1)

tk.Label(root, text="Stops").grid(row=4, column=0)
stops_var = ttk.Combobox(root, values=stop_categories)
stops_var.grid(row=4, column=1)

tk.Label(root, text="Destination City").grid(row=5, column=0)
destination_var = ttk.Combobox(root, values=city_categories)
destination_var.grid(row=5, column=1)

tk.Label(root, text="Class").grid(row=6, column=0)
class_var = ttk.Combobox(root, values=class_categories)
class_var.grid(row=6, column=1)

tk.Label(root, text="Days Left").grid(row=7, column=0)
days_left_var = tk.Spinbox(root, from_=0, to=365)
days_left_var.grid(row=7, column=1)

tk.Button(root, text="Predict Fare", command=predict_fare).grid(row=8, column=0, columnspan=2, pady=10)

result_label = tk.Label(root, text="", font=("Arial", 12), fg="green")
result_label.grid(row=9, column=0, columnspan=2)

root.mainloop()
```