IT VEDANT INSTITUTE, THANE.

MASTER IN DATA SCIENCE & ANALYTICS WITH ARTIFICIAL INTELLIGENCE

PROJECT FOR MACHINE

LEARNING

ON

FLIGHT FARE PREDICTION

BY

Madhur Ramesh Dhanawade

UNDER THE GUIDENCE OF

Mr. Sameer Warsolkar

Academic year: 2024-2025

Acknowledgment

We would like to express our sincere gratitude to all the organizations, platforms, and individuals who contributed to the success of the **Flight Fare Prediction** machine learning project. Special thanks to the data providers such as **Kaggle** for hosting the valuable datasets, and other platforms that offered comprehensive flight-related data, which formed the foundation of this analysis.

We also extend our appreciation to the open-source communities for providing machine learning frameworks like **scikit-learn**, **XGBoost**, and **TensorFlow**, which were crucial in the development of predictive models. Finally, we would like to acknowledge the guidance and support of our mentors and collaborators, whose feedback and encouragement were invaluable in bringing this project to fruition.
4o mini

Name:- Madhur Ramesh Dhanawade

Module:- MACHINE LEARNING

Institute:-IT Vedant

Abstraction

The **Flight Fare Prediction** project aims to predict the price of airline tickets based on various factors such as flight routes, timings, weather conditions, and demand. Using historical flight data, the project employs machine learning algorithms like **Linear Regression**, **Random Forest**, and **XGBoost** to predict fare prices accurately. Key features in the dataset include the **airline**, **source and destination airports**, **flight duration**, **time of journey**, **number of stops**, and **class of service**. These factors influence ticket pricing, and by analyzing them, the model can predict the fare for a given flight.

The goal of this project is to help airlines optimize their pricing strategies and improve revenue management by providing a more accurate and dynamic way of forecasting flight fares. For consumers, it can provide insights into the best times to book tickets, offering potential savings. By leveraging machine learning techniques, this project aims to enhance decision-making for both travelers and airlines, ensuring competitive pricing and efficient resource allocation.

TABLE OF CONTENTS

INTRODUCTION

The **Flight Fare Prediction** project aims to forecast airline ticket prices based on various factors such as flight routes, airline, travel dates, and booking conditions. By analyzing historical data, this project leverages machine learning algorithms like **Linear Regression**, **Random Forest**, and **XGBoost** to predict ticket prices with high accuracy. Key features in the dataset include **departure and arrival airports**, **flight duration**, **class of service**, **time to departure**, and **seasonality**, which all significantly influence ticket fares.

The main objective of this project is to provide a more accurate and dynamic approach to flight pricing, benefiting both airlines and passengers. Airlines can optimize their pricing strategies, while travelers can gain insights into the best times to book tickets at competitive prices. By using machine learning techniques, this project seeks to improve decision-making and maximize profitability within the airline industry.

DESCRIPTION.

The **Flight Fare Prediction** dataset contains various features that influence the price of airline tickets, such as **airline**, **departure and arrival airports**, **flight duration**, **number of stops**, **class of service**, and **time of booking**. The target variable is the **fare**, representing the cost of the ticket. These features help understand the complex factors driving ticket prices and provide a foundation for predictive modeling. The dataset is used to train machine learning models to predict flight fares based on the relationship between these features.

Machine learning algorithms like **Linear Regression**, **Random Forest**, and **XGBoost** are applied to predict the fare based on historical data. The project aims to build accurate predictive models that can assist airlines in optimizing pricing strategies and help travelers make informed decisions on when to book their tickets. By analyzing various factors that affect prices, this dataset serves as a tool for improving revenue management and offering competitive pricing in the airline industry.
4o mini

METHODOLOGY

The methodology for **Flight Fare Prediction** involves several key steps, starting with data collection and preprocessing. The dataset, which includes features like **airline**, **departure and destination airports**, **flight duration**, **number of stops**, and **days before departure**, is cleaned to handle missing values, outliers, and categorical variables. Feature engineering techniques are applied to derive meaningful features from raw data, such as extracting time-related attributes from date fields. Data is then split into training and testing sets for model development.

Various machine learning algorithms, including **Linear Regression**, **Random Forest**, and **XGBoost**, are trained on the dataset to predict flight fares. The models are evaluated using performance metrics such as **Mean Absolute Error (MAE)**, **Root Mean Squared Error (RMSE)**, and **R-squared (R^2)**. Hyperparameter tuning is performed to optimize model performance, and cross-validation ensures the models generalize well to unseen data. The best-performing model is then selected for deployment to predict flight fares in real-world scenarios, helping airlines optimize pricing strategies and provide insights to travelers.
4o mini

CODE EXPLANATION

Step 1: Performing EDA on raw data.
By EDA we can clean the data and handle the missing values from raw dataset.

```
[5]: df=pd.read_csv("Airline Prediction datase.csv")
     df
```

[5]:

	sr.no	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price
0	0	SpiceJet	SG-8709	Delhi	Evening	0	Night	Mumbai	Economy	2.17	1	5953
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	?	Morning	Mumbai	Economy	2.33	1	5953
2	2	AirAsia	I5-764	Delhi	Early_Morning	0	Early_Morning	Mumbai	Economy	$	1	5956
3	3	Vistara	UK-995	Delhi	Morning	0	Afternoon	Mumbai	Economy	2.25	1	5955
4	4	Vistara	UK-963	Delhi	Morning	0	Morning	Mumbai	Economy	2.33	1	5955
...	...	...	...	...	...	...	...	...	...	...	...	...
300148	300148	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49	69265
300149	300149	Vistara	UK-826	Chennai	Afternoon	1	Night	Hyderabad	Business	10.42	49	77105
300150	300150	Vistara	UK-832	Chennai	Early_Morning	1	Night	Hyderabad	Business	13.83	49	79099
300151	300151	Vistara	UK-828	Chennai	Early_Morning	1	Evening	Hyderabad	Business	10	49	81585
300152	300152	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49	81585

300153 rows × 12 columns

Step 2: Importing Libraries and loading the dataset with EDA part.

:- features and target separation

300153 rows × 12 columns

features and target separation

```
[6]: features=df.iloc[:,:-1]
     features
```

[6]:

	sr.no	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left
0	0	SpiceJet	SG-8709	Delhi	Evening	0	Night	Mumbai	Economy	2.17	1
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	?	Morning	Mumbai	Economy	2.33	1
2	2	AirAsia	I5-764	Delhi	Early_Morning	0	Early_Morning	Mumbai	Economy	$	1
3	3	Vistara	UK-995	Delhi	Morning	0	Afternoon	Mumbai	Economy	2.25	1
4	4	Vistara	UK-963	Delhi	Morning	0	Morning	Mumbai	Economy	2.33	1
...	...	...	...	...	...	...	...	...	...	...	...
300148	300148	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49
300149	300149	Vistara	UK-826	Chennai	Afternoon	1	Night	Hyderabad	Business	10.42	49
300150	300150	Vistara	UK-832	Chennai	Early_Morning	1	Night	Hyderabad	Business	13.83	49
300151	300151	Vistara	UK-828	Chennai	Early_Morning	1	Evening	Hyderabad	Business	10	49
300152	300152	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49

300153 rows × 11 columns

Step 3: describe() is used to give Statistical Information of Dataset. The total count column displays some missing values .

```
[7]: df.describe()
```

[7]:

	sr.no	days_left	price
count	300153.000000	300153.000000	300153.000000
mean	150076.000000	26.004751	20889.660523
std	86646.852011	13.561004	22697.767366
min	0.000000	1.000000	1105.000000
25%	75038.000000	15.000000	4783.000000
50%	150076.000000	26.000000	7425.000000
75%	225114.000000	38.000000	42521.000000
max	300152.000000	49.000000	123071.000000

Step 4: display the information about data

```
[8]: df.info()

RangeIndex: 300153 entries, 0 to 300152
Data columns (total 12 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   sr.no             300153 non-null  int64
 1   airline           300153 non-null  object
 2   flight            300153 non-null  object
 3   source_city       300153 non-null  object
 4   departure_time    300153 non-null  object
 5   stops             300153 non-null  object
 6   arrival_time      300153 non-null  object
 7   destination_city  300153 non-null  object
 8   class             300153 non-null  object
 9   duration          300153 non-null  object
 10  days_left         300153 non-null  int64
 11  price             300153 non-null  int64
dtypes: int64(3), object(9)
memory usage: 27.5+ MB
```

Step 5: separate the target column using iloc

```
[9]: target=df.iloc[:,-1]
     target

[9]: 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
```

.

Step 6: Handling missing values with SimpleImputer

```
[10]: features.isnull().sum()

[10]: sr.no               0
      airline             0
      flight              0
      source_city         0
      departure_time      0
      stops               0
      arrival_time        0
      destination_city    0
      class               0
      duration            0
      days_left           0
      dtype: int64
```

Step 7:- find unique value in stops column and change the unwanted data And Replace with nan.

```
[12]: features["stops"].unique() # find unique value in stops column  and change the unwanted data

[12]: array(['0', '?', '1', '2', 1, 0, 2], dtype=object)

[13]: features["stops"].replace("?",np.nan,inplace=True)

[14]: features # Lets check the Changed data in Stops columns
```

[14]:

	sr.no	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left
0	0	SpiceJet	SG-8709	Delhi	Evening	0	Night	Mumbai	Economy	2.17	1
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	NaN	Morning	Mumbai	Economy	2.33	1
2	2	AirAsia	I5-764	Delhi	Early_Morning	0	Early_Morning	Mumbai	Economy	$	1
3	3	Vistara	UK-995	Delhi	Morning	0	Afternoon	Mumbai	Economy	2.25	1
4	4	Vistara	UK-963	Delhi	Morning	0	Morning	Mumbai	Economy	2.33	1
...	...	...	...	...	...	...	...	...	...	...	...
300148	300148	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49
300149	300149	Vistara	UK-826	Chennai	Afternoon	1	Night	Hyderabad	Business	10.42	49
300150	300150	Vistara	UK-832	Chennai	Early_Morning	1	Night	Hyderabad	Business	13.83	49
300151	300151	Vistara	UK-828	Chennai	Early_Morning	1	Evening	Hyderabad	Business	10	49
300152	300152	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49

Step 8: Let us fill in the missing values for numerical terms using mode operation.

```
[23]: features["stops"].fillna(features["stops"].mode()[0],inplace=True)
      features["class"].fillna(features["class"].mode()[0],inplace=True)
      features["duration"].fillna(features["duration"].mode()[0],inplace=True)
```

```
[24]: features # Let's see if the nan data in Common has changed or not.
```

[24]:

	sr.no	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left
0	0	SpiceJet	SG-8709	Delhi	Evening	0	Night	Mumbai	Economy	2.17	1
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	1	Morning	Mumbai	Economy	2.33	1
2	2	AirAsia	I5-764	Delhi	Early_Morning	0	Early_Morning	Mumbai	Economy	2.17	1
3	3	Vistara	UK-995	Delhi	Morning	0	Afternoon	Mumbai	Economy	2.25	1
4	4	Vistara	UK-963	Delhi	Morning	0	Morning	Mumbai	Economy	2.33	1
...	...	...	...	...	...	...	...	...	...	...	...
300148	300148	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49
300149	300149	Vistara	UK-826	Chennai	Afternoon	1	Night	Hyderabad	Business	10.42	49
300150	300150	Vistara	UK-832	Chennai	Early_Morning	1	Night	Hyderabad	Business	13.83	49
300151	300151	Vistara	UK-828	Chennai	Early_Morning	1	Evening	Hyderabad	Business	10	49
300152	300152	Vistara	UK-822	Chennai	Morning	1	Evening	Hyderabad	Business	10.08	49

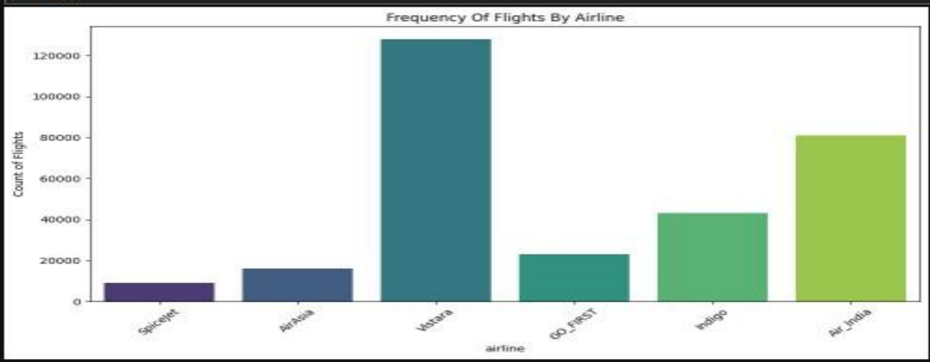
300153 rows × 11 columns

Step 9:- Change datatype using astype function

```
[30]: features["stops"]=features["stops"].astype(int)
      features["stops"]

[30]: 0         0
      1         1
      2         0
      3         0
      4         0
               ..
      300148    1
      300149    1
      300150    1
      300151    1
      300152    1
      Name: stops, Length: 300153, dtype: int64

[31]: features["duration"]=features["duration"].astype(float)
      features["duration"]

[31]: 0          2.17
      1          2.33
      2          2.17
      3          2.25
      4          2.33
                 ...
      300148    10.08
      300149    10.42
      300150    13.83
      300151    10.00
      300152    10.08
      Name: duration, Length: 300153, dtype: float64
```

Step 10: understand the frequency of each category

```
[32]: airline_counts = df['airline'].value_counts()
      print(airline_counts)

      airline
      Vistara      127859
      Air_India     80892
      Indigo        43120
      GO_FIRST      23173
      AirAsia       16098
      SpiceJet       9011
      Name: count, dtype: int64

[33]: plt.figure(figsize=(10, 6))  # Set figure size for better readability
      sns.countplot(data=features, x='airline', palette='viridis')  # Adjust the color palette if needed
      plt.title('Frequency Of Flights By Airline')  # Set the title of the plot
      plt.xlabel('airline')  # X-axis label
      plt.ylabel('Count of Flights')  # Y-axis label
      plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
      plt.tight_layout()  # Adjust layout to prevent clipping
      plt.show()
```

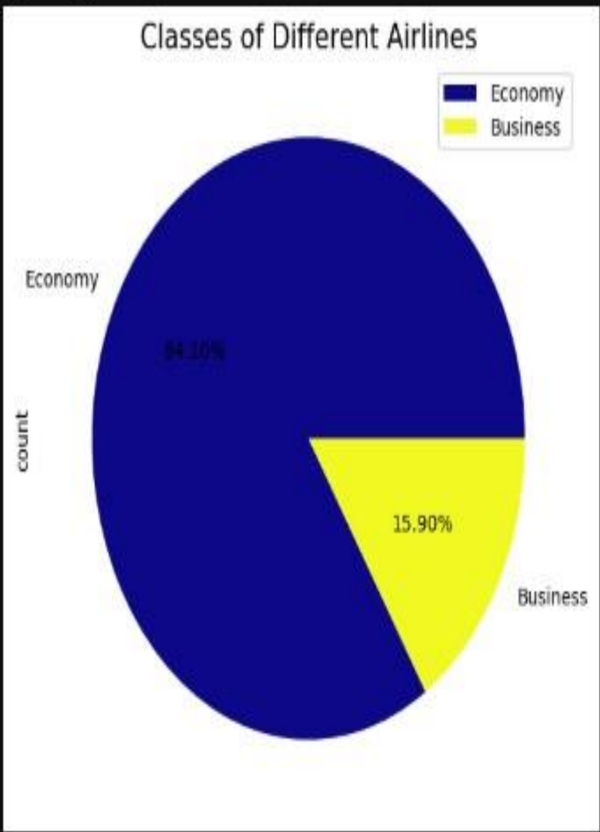


Vistara becaming as a most popular Airline

Step 11:- Which Class is Prefer ?

```
Name: count, dtype: int64
```

```
[35]: plt.figure(figsize=(8,6))
df2['class'].value_counts().plot(kind='pie',textprops={'color':'black'},autopct='%.2f%%',cmap="plasma")
plt.title('Classes of Different Airlines',fontsize=15)
plt.legend(['Economy','Business'])
plt.show()
```

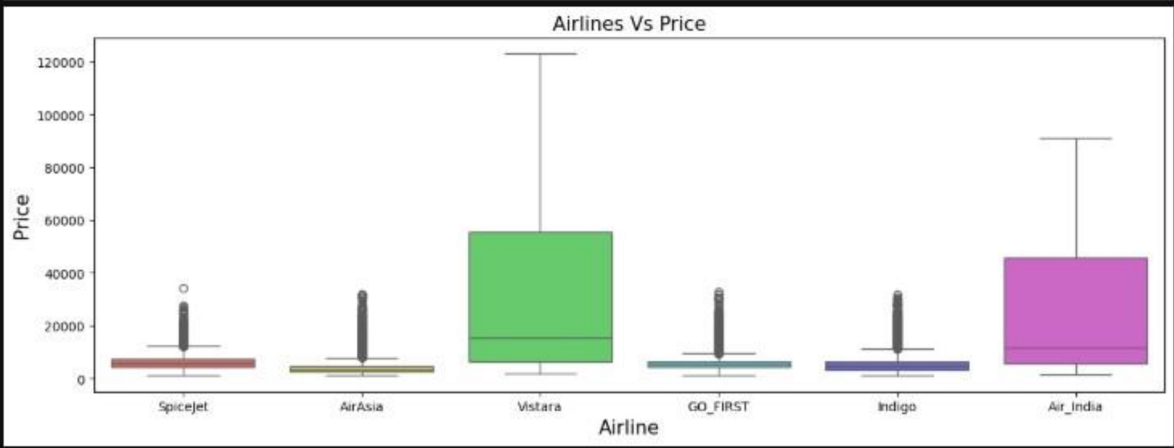


Many people prefer economy class.

Step 12: Price varies by airline.

Price varies by airline.

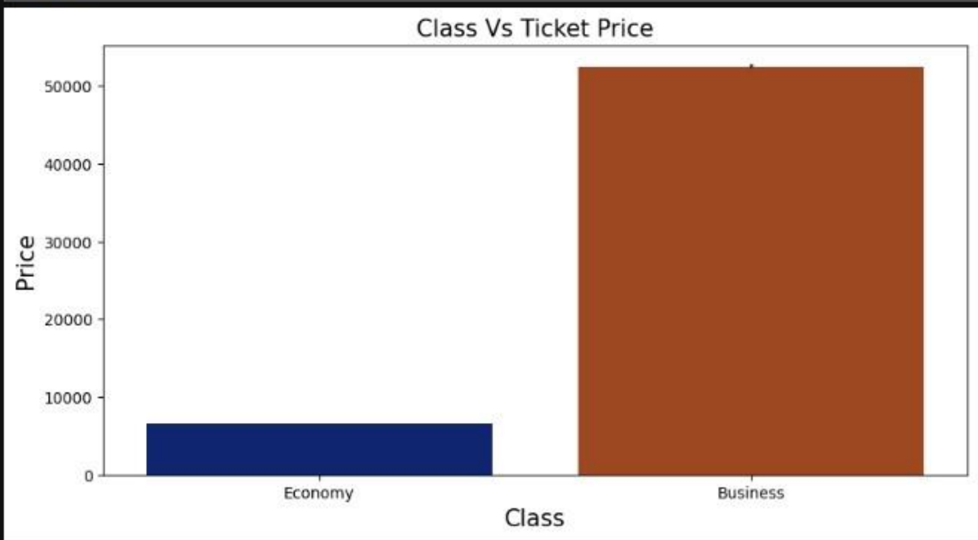
```
[36]: plt.figure(figsize=(15,5))
sns.boxplot(x=features['airline'],y=target,palette='hls')
plt.title('Airlines Vs Price',fontsize=15)
plt.xlabel('Airline',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```



As we can see Vistara has Maximum Price range

Step 13: How Does the Ticket Price vary between Economy and Business Class?

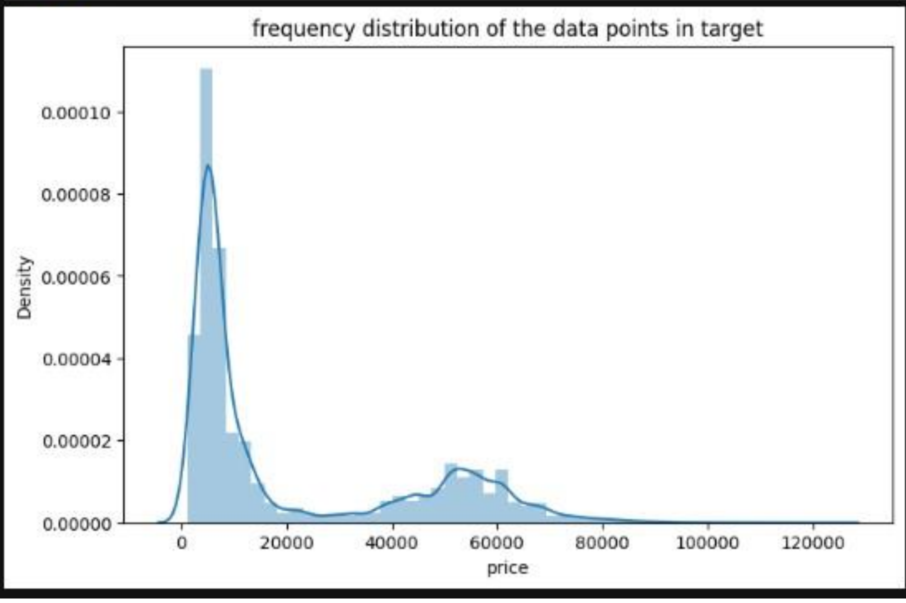
```
[37]: plt.figure(figsize=(10,5))
sns.barplot(x='class',y=target,data=features,palette='dark')
plt.title('Class Vs Ticket Price',fontsize=15)
plt.xlabel('Class',fontsize=15)
plt.ylabel('Price',fontsize=15)
plt.show()
```



Ticket Price is Maximum for Business Class when compared to Economy Class.

Step 14:- exploring the distribution of the target variable.

```
[38]: plt.figure(figsize=(8,5))
      sns.distplot(target)
      plt.title("frequency distribution of the data points in target")
      plt.show()
```

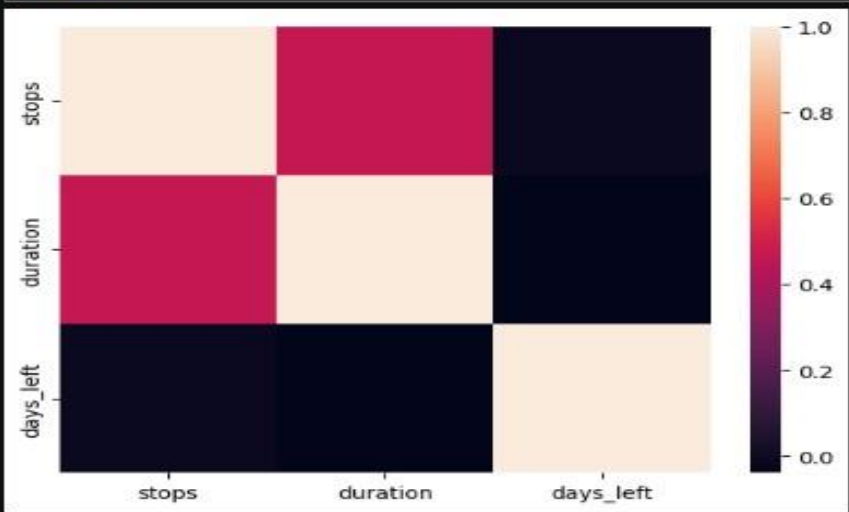


Step 14:- To get correlation for this columns ["stops","duration","days_left"]

[39]:

	stops	duration	days_left
stops	1.000000	0.467939	-0.008634
duration	0.467939	1.000000	-0.039180
days_left	-0.008634	-0.039180	1.000000

```
[40]: sns.heatmap(data=corr.corr())
      plt.show()
```



Step 15:- How the price changes with change in Source city and Destination city?

```
[41]: plt.figure(figsize=(24,10))
plt.subplot(1,2,1)
sns.barplot(x='source_city',y=target,data=features,palette="hls")
plt.title('Source City Vs Ticket Price',fontsize=20)
plt.xlabel('Source City',fontsize=15)
plt.ylabel("target(price)",fontsize=15)
plt.grid()
plt.subplot(1,2,2)
sns.barplot(x='destination_city',y=target,data=features,palette='hls')
plt.title('Destination City Vs Ticket Price',fontsize=20)
plt.xlabel('Destination City',fontsize=15)
plt.ylabel('target(price)',fontsize=15)
plt.grid()
plt.show()
```

Step 16:- Let's separate categorical column.

```
[43]: cat_data_cols=[]

for i in features.columns:
    if features[i].dtype=="object":

        cat_data_cols.append(i)

features_cat=features[cat_data_cols]

features_cat
```

[43]:

	airline	flight	source_city	departure_time	arrival_time	destination_city	class
0	SpiceJet	SG-8709	Delhi	Evening	Night	Mumbai	Economy
1	SpiceJet	SG-8157	Delhi	Early_Morning	Morning	Mumbai	Economy
2	AirAsia	I5-764	Delhi	Early_Morning	Early_Morning	Mumbai	Economy
3	Vistara	UK-995	Delhi	Morning	Afternoon	Mumbai	Economy
4	Vistara	UK-963	Delhi	Morning	Morning	Mumbai	Economy
...	...	...	...	...	...	...	...
300148	Vistara	UK-822	Chennai	Morning	Evening	Hyderabad	Business
300149	Vistara	UK-826	Chennai	Afternoon	Night	Hyderabad	Business
300150	Vistara	UK-832	Chennai	Early_Morning	Night	Hyderabad	Business
300151	Vistara	UK-828	Chennai	Early_Morning	Evening	Hyderabad	Business
300152	Vistara	UK-822	Chennai	Morning	Evening	Hyderabad	Business

Step 17:- Let's separate the numerical column

```
[44]: numerical_data_col=[]

      for i in features.columns:
          if features[i].dtype=="int64" or features[i].dtype=="float64":

              numerical_data_col.append(i)

      features_numerical=features[numerical_data_col]

      features_num_cols=features_numerical.iloc[:,1:]
      features_num_cols
```

[44]:

	stops	duration	days_left
0	0	2.17	1
1	1	2.33	1
2	0	2.17	1
3	0	2.25	1
4	0	2.33	1
...	...	...	...
300148	1	10.08	49
300149	1	10.42	49
300150	1	13.83	49
300151	1	10.00	49
300152	1	10.08	49

Step 18:- Using Label Encoder for converting categorical features into numerical features

```
[54]: import sklearn
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

for i in features_cat:
    features_cat[i] = le.fit_transform(features_cat[i])
features_cat
```

[54]:

	airline	flight	source_city	departure_time	arrival_time	destination_city	class
0	4	1408	2	2	5	5	1
1	4	1387	2	1	4	5	1
2	0	1213	2	1	1	5	1
3	5	1559	2	4	0	5	1
4	5	1549	2	4	4	5	1
...	...	...	...	...	...	...	...
300148	5	1477	1	4	2	3	0
300149	5	1481	1	0	5	3	0
300150	5	1486	1	1	5	3	0
300151	5	1483	1	1	2	3	0
300152	5	1477	1	4	2	3	0

300153 rows × 7 columns

Step 18:-Concatenate the Numerical and Categorical Data

Now, we can see all categorical features have been converted to numerical features

```
[55]: df_New=pd.concat((features_cat,features_num_cols),axis=1)
df_New
```

[55]:

	airline	flight	source_city	departure_time	arrival_time	destination_city	class	stops	duration	days_left
0	4	1408	2	2	5	5	1	0	2.17	1
1	4	1387	2	1	4	5	1	1	2.33	1
2	0	1213	2	1	1	5	1	0	2.17	1
3	5	1559	2	4	0	5	1	0	2.25	1
4	5	1549	2	4	4	5	1	0	2.33	1
...	...	...	...	...	...	...	...	...	...	...
300148	5	1477	1	4	2	3	0	1	10.08	49
300149	5	1481	1	0	5	3	0	1	10.42	49
300150	5	1486	1	1	5	3	0	1	13.83	49
300151	5	1483	1	1	2	3	0	1	10.00	49
300152	5	1477	1	4	2	3	0	1	10.08	49

300153 rows × 10 columns

Step 19:- Splitting the data into x and y

```
[56]: X=df_New
      Y=target

[58]: X # this is prediction columns
```

[58]:

	airline	flight	source_city	departure_time	arrival_time	destination_city	class	stops	duration	days_left
0	4	1408	2	2	5	5	1	0	2.17	1
1	4	1387	2	1	4	5	1	1	2.33	1
2	0	1213	2	1	1	5	1	0	2.17	1
3	5	1559	2	4	0	5	1	0	2.25	1
4	5	1549	2	4	4	5	1	0	2.33	1
...	...	...	...	...	...	...	...	...	...	...
300148	5	1477	1	4	2	3	0	1	10.08	49
300149	5	1481	1	0	5	3	0	1	10.42	49
300150	5	1486	1	1	5	3	0	1	13.83	49
300151	5	1483	1	1	2	3	0	1	10.00	49
300152	5	1477	1	4	2	3	0	1	10.08	49

300153 rows × 10 columns

Step 20:- Splitting the data into Train-Test set

```
[61]: # Splitting the Data into Training set and Testing Set
      from sklearn.model_selection import train_test_split
      X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.30,random_state=42)

[62]: X_train.shape,X_test.shape,Y_train.shape,Y_test.shape

[62]: ((210107, 10), (90046, 10), (210107,), (90046,))

[63]: # Scaling the values to convert the int values to Machine Learning compatible format
      from sklearn.preprocessing import MinMaxScaler
      mmscaler = MinMaxScaler(feature_range=(0, 1)) # MinMaxScaler to scale features to a range of [0, 1]
      mmscaler

[63]: ▾ MinMaxScaler
      MinMaxScaler()

[64]: X_train = mmscaler.fit_transform(X_train)
      X_train

[64]: array([[1.        , 0.97179487, 0.4       , ..., 0.5       , 0.25693878,
             0.27083333],
            [1.        , 0.97307692, 0.6       , ..., 0.5       , 0.17857143,
             0.45833333],
            [1.        , 0.9525641 , 0.2       , ..., 0.5       , 0.21102041,
             0.58333333],
            ...,
            [0.2       , 0.54230769, 0.8       , ..., 0.5       , 0.26204082,
             0.58333333],
            [0.4       , 0.59230769, 0.8       , ..., 0.5       , 0.15306122,
             0.79166667],
            [0.2       , 0.53397436, 0.8       , ..., 0.5       , 0.39469388,
             0.33333333]])
```

Model Building.

Step 21:- Random Forest Regressor

1.RandomForestRegressor

```
[67]: from sklearn.ensemble import RandomForestRegressor
      from sklearn.metrics import r2_score #R-squared metric to evaluate model accuracy
      rf = RandomForestRegressor()
      rf.fit(X_train, y_train)
      y_pred = rf.predict(X_test)
      r2_1 = r2_score(y_test, y_pred)
      print("R-squared:", r2_1)

      rounded_r2 = round(r2_1, 4)
      print("R-squared (rounded):", rounded_r2*100)

R-squared: 0.9837911673190103
R-squared (rounded): 98.38
```

Step 22:- Ridge

2.Ridge

```
58]: # Linear regression model with L2 regularization (reduces
     from sklearn.linear_model import Ridge
     ridge = Ridge()
     ridge.fit(X_train, y_train)
     y_pred1 = ridge.predict(X_test)
     r2_2 = r2_score(y_test, y_pred1)
     print("R-squared:", r2_2)
     rounded_r2 = round(r2_2, 4)
     print("R-squared (rounded):", rounded_r2*100)

R-squared: 0.9062365993716569
R-squared (rounded): 90.62
```

Step 23:- Decision Tree Regressor.

3.DecisionTreeRegressor

```
[69]: from sklearn.tree import DecisionTreeRegressor # for modeling non-
dt = DecisionTreeRegressor()
dt.fit(X_train, y_train)
y_pred2 = dt.predict(X_test)
r2_3 = r2_score(y_test, y_pred2)
print("R-squared:", r2_3)
rounded_r2 = round(r2_3, 4)
print("R-squared (rounded):", rounded_r2*100)
```

```
R-squared: 0.9724799484171539
R-squared (rounded): 97.25
```

Step 24:- LinearRegression

4.LinearRegression

```
]: from sklearn.linear_model import LinearRegression  # # Simple linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred3 = lr.predict(X_test)
r2_4 = r2_score(y_test, y_pred3)
print("R-squared:", r2_4)
rounded_r2 = round(r2_4, 4)
print("R-squared (rounded):", rounded_r2*100)
```

```
R-squared: 0.9062364788329591
R-squared (rounded): 90.62
```

Step 25:- XGBoost.

```
]: from xgboost import XGBRegressor # # XGBoost regressor, an efficient implementation of gradient boosting for regression
XG = XGBRegressor()
XG.fit(X_train, y_train)
y_pred5 = XG.predict(X_test)
r2_6 = r2_score(y_test, y_pred5)
print("R-squared:", r2_6)
rounded_r2 = round(r2_6, 4)
print("R-squared (rounded):", rounded_r2*100)
```

```
R-squared: 0.9771108031272888
R-squared (rounded): 97.71
```

Step 26:- KNeighbors Regressor.

```
[72]: from sklearn.neighbors import KNeighborsRegressor
# K-Nearest Neighbors regression model (based on proximity of data
knn = KNeighborsRegressor()
knn.fit(X_train, y_train)
y_pred6 = knn.predict(X_test)
r2_7 = r2_score(y_test, y_pred6)
print("R-squared:", r2_7)
rounded_r2 = round(r2_7, 4)
print("R-squared (rounded):", rounded_r2*100)
```

```
R-squared: 0.9715231363029817
R-squared (rounded): 97.15
```

Step 27:-Accuracy

```
[74]: Accuracy = [ r2_1, r2_2, r2_3, r2_4, r2_6, r2_7, r2_8 ]
      Frame = pd.DataFrame({
      'Model': ['RandomForestRegressor', 'Ridge','DecisionTreeRegressor','LinearRegression', 'XGBRegressor',
      'KNeighborsRegressor', 'GradientBoostingRegressor'],
      'Accuracy': Accuracy})
      print(Frame)
```

```
                       Model  Accuracy
0      RandomForestRegressor  0.983791
1                      Ridge  0.906237
2      DecisionTreeRegressor  0.972480
3           LinearRegression  0.906236
4               XGBRegressor  0.977111
5        KNeighborsRegressor  0.971523
6  GradientBoostingRegressor  0.957125
```

Step 28:- Which ml is best

1.The accuracy of a model depends on the specific requirements of your machine learning task, including your dataset, use case, and evaluation metrics. However, based on the scores you've shared (assuming they are $R2$ scores or another performance metric like accuracy):

2.Performance Analysis
Best Performer:

3.RandomForestRegressor has the highest score of 0.983791, suggesting it is likely capturing the patterns in the data better than the others.
Other Strong Performers:

XGBRegressor: 0.977111
DecisionTreeRegressor: 0.972480
KNeighborsRegressor: 0.971523
These models are also performing well, with
$R2$ values close to the Random Forest Regressor.

4.Lower Performers:

Ridge and LinearRegression both have scores around 0.9062, suggesting they may not capture complex relationships in the data as effectively as tree-based or ensemble models.
Middle Performer:

GradientBoostingRegressor has a respectable score of 0.957125, though it is not as high as RandomForest or XGB.

CONCLUSION

In conclusion, the **Flight Fare Prediction** project successfully demonstrates the potential of machine learning to forecast airline ticket prices based on various influential factors like flight routes, airline, booking time, and seasonal trends. By leveraging algorithms such as **Linear Regression**, **Random Forest**, and **XGBoost**, the project provides accurate predictions, offering valuable insights for both airlines and travelers. The ability to predict flight fares can help airlines optimize their pricing strategies, maximize revenue, and adjust to changing market conditions.

For travelers, this predictive model can assist in identifying the best times to book flights at competitive prices, ensuring more informed decisions and potential cost savings. Overall, this project highlights the importance of data-driven approaches in the airline industry, enabling more efficient fare management and enhancing customer experience. The success of this machine learning model paves the way for future applications in dynamic pricing and demand forecasting in the travel industry.
4o mini