ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC XÃ HỘI VÀ NHÂN VĂN

KHOA THƯ VIỆN – THÔNG TIN HỌC

PHÂN TÍCH DỮ LIỆU CHO QUẢN LÝ

Giảng viên hướng dẫn: ThS. Trần Đình Anh Huy

Sinh viên thực hiện: Huỳnh Phương Vi – 2156210151

Tp. Hồ Chí Minh, ngày 6 tháng 1 năm 2024

MỤC LỤC

I. TỔNG QUAN:

1. Mục đích của báo cáo: Báo cáo này nhằm mục đích dự đoán giá vé máy bay trong tương lai và tìm ra mô hình tốt nhất để dự đoán giá vé máy bay.
2. Phạm vi của báo cáo: Tập trung vào dự đoán giá vé máy bay dựa trên bộ dữ liệu ghi lại lịch sử những chuyến bay bao gồm những thông tin đi kèm trong phạm vi năm 2019.
3. Dataset sử dụng : /kaggle/input/flight-price-prediction/Clean_Dataset.csv
4. Công cụ sử dụng: Google Colab
5. Ngôn ngữ sử dụng để phân tích: Python.
6. Phương pháp nghiên cứu: Báo cáo sử dụng phương pháp phân tích dữ liệu định lượng và phương pháp học máy (machine learning):

- Phân tích mô tả (khám phá dữ liệu, EDA)
- Phân tích dự đoán bằng mô hình hồi quy tuyến tính (đơn biến, đa biến, hồi quy OLS)

II. DỮ LIỆU:

1. Khái quát về Dataset:

- Tập dữ liệu chứa thông tin về các tùy chọn đặt vé máy bay giữa 6 thành phố đô thị hàng đầu của Ấn Độ.
- Có 10684 điểm dữ liệu và 11 biến trong tập dữ liệu.

2. Xác định biến và phân loại biến:

- **Giới thiệu về ý nghĩa của 11 biến trong tập dữ liệu:**

- **Airline:** Cột chứa tên của các hãng hàng không. Có 6 hãng hàng không: IndiGo, AirIndia, JetAirway, GoAir, SpiceJet, JetAirway, Mutiples Carriers,
- **Date_of_Journey:** Cột chứa ngày chuyến bay cất cánh.
- **Source:** Thành phố nơi chuyến bay cất cánh
- **Destination:** Thành phố nơi chuyến bay đáp cánh.
- **Route:** Tuyến đường đi của chuyến bay (thể hiện ký hiệu từ thành phố nào qua thành phố nào)
- **Dep_time:** Thời gian khởi hành của chuyến bay.
- **Arrival_time:** Thời gian hạ cánh của chuyến bay.
- **Duration:** Tổng thời gian cần thiết để di chuyển giữa các thành phố tính bằng giờ.
- **Total_Stops:** Tổng số điểm dừng giữa thành phố nguồn và thành phố đích đến.
- **Addinational_Info:** Thông tin kèm theo nếu có.
- **Price:** Giá vé máy bay.

- **Xác định biến:**
  Tập dữ liệu chứa duy nhất một biến số là Price.
  Những biến còn lại đều là biến phân loại

⇨ Do đó khi tiến hành xử lý dữ liệu trước khi dự đoán, cần dùng kỹ thuật decoding để chuyển biến phân loại về số.

III. PHÂN TÍCH DỮ LIỆU

1. Import dữ liệu và khai báo các thư viện:

- Đây là bước đầu tiên trong trước khi đi vào khai phá dữ liệu.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import statsmodels.api as sm
import plotly
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from plotly.offline import iplot
!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip
from pandas_profiling import ProfileReport
```

⇨ **Tổng hợp những thư viện dùng trong quá trình phân tích:**

- NumPy là thư viện mã nguồn mở cung cấp các đối tượng mảng đa chiều và các hàm để xử lý các đối tượng mảng.
- Pandas là thư viện mã nguồn mở cung cấp các đối tượng khung dữ liệu và các hàm để xử lý các dữ liệu bảng.
- Seaborn là thư viện mã nguồn mở cung cấp các giao diện đồ họa người dùng (GUI) cho các thư viện trực quan hóa dữ liệu khác, chẳng hạn như Matplotlib. Seaborn thường được sử dụng để tạo các biểu đồ và đồ thị dữ liệu trực quan và đẹp mắt.
- Matplotlib là thư viện mã nguồn mở cung cấp các hàm để tạo các biểu đồ và đồ thị dữ liệu.
- Statsmodels là thư viện mã nguồn mở cung cấp các hàm để phân tích thống kê dữ liệu.
- Plotly là thư viện mã nguồn mở cung cấp các hàm để tạo các biểu đồ và đồ thị dữ liệu tương tác.
- Px là thư viện con của Plotly cung cấp các hàm để tạo các biểu đồ và đồ thị dữ liệu theo phong cách Excel.
- Ff là thư viện con của Plotly cung cấp các hàm để tạo các biểu đồ và đồ thị dữ liệu tùy chỉnh.

- Go là thư viện con của Plotly cung cấp các hàm để tạo các biểu đồ và đồ thị dữ liệu từ các đối tượng JSON.
- Thư viện from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error cung cấp các hàm để đánh giá hiệu suất của các mô hình học máy.
- Thư viện from sklearn.model_selection import train_test_split cung cấp các hàm để chia dữ liệu thành dữ liệu đào tạo và dữ liệu kiểm tra.
- Thư viện from sklearn.linear_model import LinearRegression cung cấp một triển khai của mô hình hồi quy tuyến tính.
- Thư viện from plotly.offline import iplot cung cấp các hàm để hiển thị các biểu đồ và đồ thị dữ liệu trong trình duyệt web.

2. Load file data vào Dataframe trên Google Colab:

```
df = pd.read_excel('Data_Train.xlsx')
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

3. EDA – Exploratory Data Analysis:

EDA – Phân tích dữ liệu thăm dò là quá trình mô tả dữ liệu bằng các kỹ thuật thống kê và trực quan hoá nhằm tập trung vào các khía cạnh quan trọng của dữ liệu để tiếp tục phân tích. Điều này bao gồm cả việc kiểm tra tập dữ liệu từ nhiều góc độ, mô tả và tóm tắt nó mà không đưa ra bất kỳ giả định nào khác về nội dung của nó. EDA là một bước quan trọng cần phải thực hiện trước khi đi sâu vào mô hình thống kê hoặc học máy.

A. Xem xét điểm bất thường của dữ liệu

- Kiểm tra số columns trong bộ dữ liệu:

```
df.columns
```

```
Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
       'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
       'Additional_Info', 'Price'],
      dtype='object')
```

```
[ ] list(df)
```

```
['Airline',
 'Date_of_Journey',
 'Source',
 'Destination',
 'Route',
 'Dep_Time',
 'Arrival_Time',
 'Duration',
 'Total_Stops',
 'Additional_Info',
 'Price']
```

```
df.shape
```

```
(10683, 11)
```

⇨ Tập dữ liệu có tổng cộng 11 cột và 10683 dòng.

```
df.info(verbose = True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

⇨ Hầu hết dữ liệu hoàn toàn là biến phân loại, chỉ có duy nhất biến Price là biến số
=> Cần dùng kĩ thuật Dummy Encoding để mã hóa những biến phân loại trước khi xây dựng model.

- Kiểm tra Giá trị khuyết trong bộ dữ liệu:

```
df.isna().sum()
```

```
Airline            0
Date_of_Journey    0
Source             0
Destination        0
Route              1
Dep_Time           0
Arrival_Time       0
Duration           0
Total_Stops        1
Additional_Info    0
Price              0
dtype: int64
```

⇨ Có giá trị khuyết ở hai cột Route và Total_Stops.

- Miêu tả dữ liệu biến số:

```
[15] df.describe()
```

	Price
count	10683.000000
mean	9087.064121
std	4611.359167
min	1759.000000
25%	5277.000000
50%	8372.000000
75%	12373.000000
max	79512.000000

Trung bình giá vé máy bay là 9087 trong khi min là 4611 và max là 79512 => lệch phải, giá thấp được mua nhiều hơn là số lượng giá cao. Độ lệch chuẩn lớn.

- Xem xét điểm bất thường của dữ liệu: nếu nguồn và đích có cùng tên, hãy

```
[ ] df[(df['Source']==df['Destination'])]
```

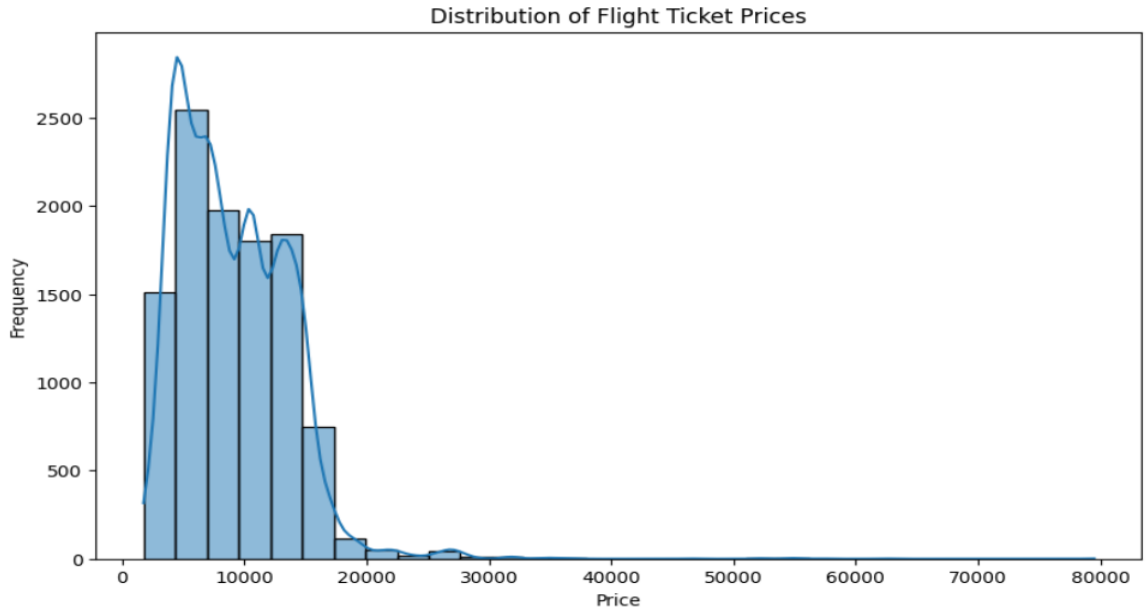
Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price

bỏ hàng

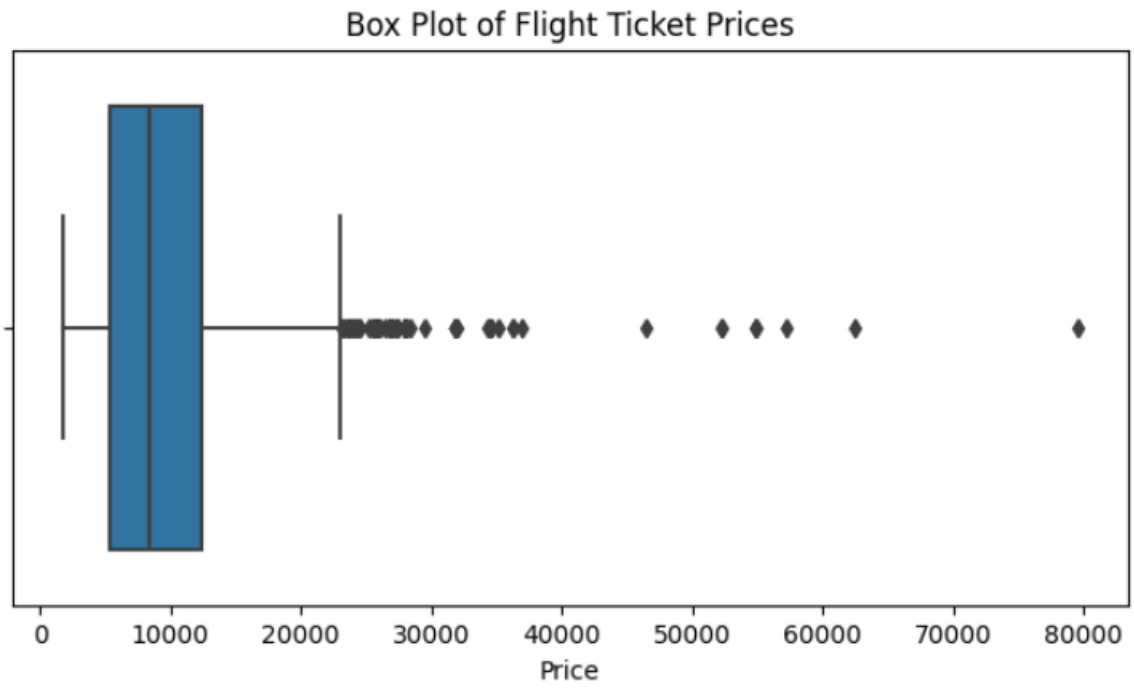
- **Kiểm tra Phân phối của biến Price cũng như các tham số thống kê:**

```
plt.figure(figsize=(10, 6))
sns.histplot(df['Price'], bins=30, kde=True)
plt.title('Distribution of Flight Ticket Prices')
```

```
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
price_mean = df['Price'].mean()
price_median = df['Price'].median()
price_std = df['Price'].std()
price_min =df['Price'].min()
price_max = df['Price'].max()
print(f"Mean Price: {price_mean}")
print(f"Median Price: {price_median}")
print(f"Standard Deviation of Price: {price_std}")
print(f"Minimum Price: {price_min}")
print(f"Maximum Price: {price_max}")
plt.figure(figsize=(8, 4))
sns.boxplot(x=df['Price'])
plt.title('Box Plot of Flight Ticket Prices')
plt.xlabel('Price')
plt.show()
```




```
Mean Price: 9087.064120565385
Median Price: 8372.0
Standard Deviation of Price: 4611.3591668171175
Minimum Price: 1759
Maximum Price: 79512
```



⇨ Giá bị lệch phải có mức cao nhất từ 0 đến 20.000, điều đó có nghĩa là các chuyến bay có giá nằm trong phạm vi này được bán thường xuyên, biểu đồ cũng hiển thị giá trị ngoại lệ từ năm 20000.

⇨ Có outliers ở tất cả các hãng hàng không trong khoảng từ 20000 đến 80000

- Kiểm tra giá trị trùng lặp trong bộ dữ liệu:

```
print('Tổng số dòng: ',df.shape[0])
print('Tổng số cột: ',df.shape[1])
dupplicate_row = len(df)-len(df.drop_duplicates())
print('Số dòng bị trùng lặp: ',dupplicate_row)
```

```
Tổng số dòng:  10683
Tổng số cột:  11
Số dòng bị trùng lặp:  220
```

⇨ **Có 220 dòng bị trùng lặp => dưới 10% trên tổng số 10683 nên sẽ cân nhắc để xóa giá trị trùng lặp ở bước Preprocessing.**

4. PREPROCESSING

A. Xử lý những biến về dạng số:

- **Biến Total Stops có thể chuyển về thành số thay vì biến phân loại:**

```
df['Total_Stops'] = df['Total_Stops'].str.extract(r'(\d+)').fillna('0')
```

```
df['Total_Stops'] = df['Total_Stops'].astype(str).astype(int)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10683 non-null  int64 
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(2), object(9)
memory usage: 918.2+ KB
```

```
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	0	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1	No info	13302

- **Tạo biến Phân loại buổi trong ngày dựa trên giờ khởi hành (Dep_time và giờ đáp (Arrival_time):**

```
def condition(x):
    if x>=6 and x <= 12:
        return "Morning"
    elif x>=12 and x<=18:
        return "Afternoon"
    elif x>18 and x<24:
        return 'Evening'
    else:
      return 'Night'
```

```
df['Deps'] = df['Dep_hr'].apply(condition)
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional_Info	Price	Day_of_Journey	t	Dep_hr	Dep_min	Arrival_hr	Arrival_min	Duration_hr	Duration_min	Deps
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	21	22	20	1	10	2	50	Evening
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	5	2	5	50	13	15	7	25	Night
2	Jet Airways	2019-09-06	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	6	3	9	25	4	25	19	00	Morning
3	IndiGo	2019-12-05	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	5	2	18	5	23	30	5	25	Afternoon
4	IndiGo	2019-01-03	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	3	0	16	50	21	35	4	45	Afternoon

```
def condition(y):
    if y>=6 and y<= 12:
        return "Morning"
    elif y>=12 and y<=18:
        return "Afternoon"
    elif y>18 and y<24:
        return 'Evening'
    else:
      return 'Night'
```

```
df['Arrivals'] = df['Arrival_hr'].apply(condition)
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional_Info	Price	Day_of_Journey	t	Dep_hr	Dep_min	Arrival_hr	Arrival_min	Duration_hr	Duration_min	Deps	Arrivals
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	21	22	20	1	10	2	50	Evening	Night
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	5	2	5	50	13	15	7	25	Night	Afternoon
2	Jet Airways	2019-09-06	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	6	3	9	25	4	25	19	00	Morning	Night
3	IndiGo	2019-12-05	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	5	2	18	5	23	30	5	25	Afternoon	Evening
4	IndiGo	2019-01-03	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	3	0	16	50	21	35	4	45	Afternoon	Evening

- **Chuyển 4 biến ngày và giờ về thành kiểu Datetime:**
- **Date_of_Journey: ở biến này convert lại thành biến t đơn giản để phục vụ cho việc chạy model.**

```
df['Date_of_Journey']=pd.to_datetime(df['Date_of_Journey'])
df['Day_of_Journey']=(df['Date_of_Journey']).dt.day
df["t"] = df["Date_of_Journey"].dt.day- df["Day_of_Journey"].min()
df
```

```
<ipython-input-91-a57afc80d016>:1: UserWarning:

Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Specify a format to ensure consisten
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Day_of_Journey	t
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	0	No info	3897	24	21
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2	No info	7662	5	2
2	Jet Airways	2019-09-06	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2	No info	13882	6	3
3	IndiGo	2019-12-05	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1	No info	6218	5	2
4													
...													
10678													
10679													
10680													
10681	Vistara	2019-01-03	Banglore	New Delhi	BLR → DEL	11:30	14:10	2h 40m	0	No info	12648	3	0
10682	Air India	2019-09-05	Delhi	Cochin	DEL → GOI → BOM → COK	10:55	19:15	8h 20m	2	No info	11753	5	2

```
df['Arrival_hr']=pd.to_datetime(df['Arrival_Time']).dt.hour
df['Arrival_min']=pd.to_datetime(df['Arrival_Time']).dt.minute
```

```
df.drop(["Arrival_Time"],axis=1,inplace=True)
```

10683 rows × 13 columns

- **Dep_time: Chuyển biến về thành kiểu datetime và phân ra thành cột giờ và phút_sau đó xóa biến ban đầu là Dep_time vì sẽ không sử dụng tới nữa:**
- **Arrival_time: Tương tự như biến Dep_time:**

- **Ở cột Dep_hr và Arrival_hr, chuyển về thành hai biến phân loại bao gồm 4 giá trị (Morning, Afternoon, Evening, Night)**

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional_Info	Price	Day_of_Journey	t	Dep_hr	Dep_min	Arrival_hr	Arrival_min	Duration_hr	Duration_min	Deps	Arrivals
0	IndiGo	2019-03-24	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	21	22	20	1	10	2	50	Evening	Night
1	Air India	2019-01-05	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	5	2	5	50	13	15	7	25	Night	Afternoon
2	Jet Airways	2019-09-06	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	6	3	9	25	4	25	19	00	Morning	Night
3	IndiGo	2019-12-05	Kolkata	Banglore	CCU → NAG → BLR	1 stop	No info	6218	5	2	18	5	23	30	5	25	Afternoon	Evening
4	IndiGo	2019-01-03	Banglore	New Delhi	BLR → NAG → DEL	1 stop	No info	13302	3	0	16	50	21	35	4	45	Afternoon	Evening

- **Duration:**

```
duration=df['Duration'].str.split(' ',expand=True) #split duration datapoints based on space ' '
duration[1].fillna('00m',inplace=True)   #fill all "NAN" with '00m'
df['Duration_hr']=duration[0].apply(lambda x: x[:-1]) #select the item at index o and leave the last one (in this case the 'h')
df['Duration_min']=duration[1].apply(lambda x: x[:-1]) #select the item at index 1 and leave the last one (in this case the 'm')
```

```
df.drop(["Duration"],axis=1,inplace=True)
```

```
def condition(x):
    if x>=6 and x <= 12:
        return "Morning"
    elif x>=12 and x<=18:
        return "Afternoon"
    elif x>18 and x<24:
        return 'Evening'
    else:
      return 'Night'
```

```
df['Deps'] = df['Dep_hr'].apply(condition)
df.head()
```

```
def condition(y):
    if y>=6 and y<= 12:
        return "Morning"
    elif y>=12 and y<=18:
        return "Afternoon"
    elif y>18 and y<24:
        return 'Evening'
    else:
      return 'Night'
```

```
df['Arrivals'] = df['Arrival_hr'].apply(condition)
df.head()
```

```
df.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Total_Stops	Additional_Info	Price	Dep_hr	Dep_min	Arrival_hr	Arrival_min	Duration_hr	Duration_min
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	0	No info	3897	22	20	1	10	2	50
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	2	No info	7662	5	50	13	15	7	25
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	2	No info	13882	9	25	4	25	19	00
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	1	No info	6218	18	5	23	30	5	25
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	1	No info	13302	16	50	21	35	4	45

B. Xử lý missing value (giá trị khuyết):

```
df.dropna(how='any',inplace=True)
df.isnull().sum()
```

```
Airline           0
Date_of_Journey   0
Source            0
Destination       0
Route             0
Total_Stops       0
Additional_Info   0
Price             0
Day_of_Journey    0
t                 0
Dep_hr            0
Dep_min           0
Arrival_hr        0
Arrival_min       0
Duration_hr       0
Duration_min      0
Arrivals          0
Deps              0
dtype: int64
```

C. Xử lý giá trị lặp lại trong data (Duplicates values):

```
df.drop_duplicates(keep=False, inplace=True)
```

```
print('Tổng số dòng: ',df.shape[0])
print('Tổng số cột: ',df.shape[1])
dupplicate_row = len(df)-len(df.drop_duplicates())
print('Số dòng bị trùng lặp: ',dupplicate_row)
```

```
Tổng số dòng:  10263
Tổng số cột:  18
Số dòng bị trùng lặp:  0
```

D. Xử lý Outliers ở biến Price (Phân phối cho thấy có giá trị ngoại lai ở biến Price):

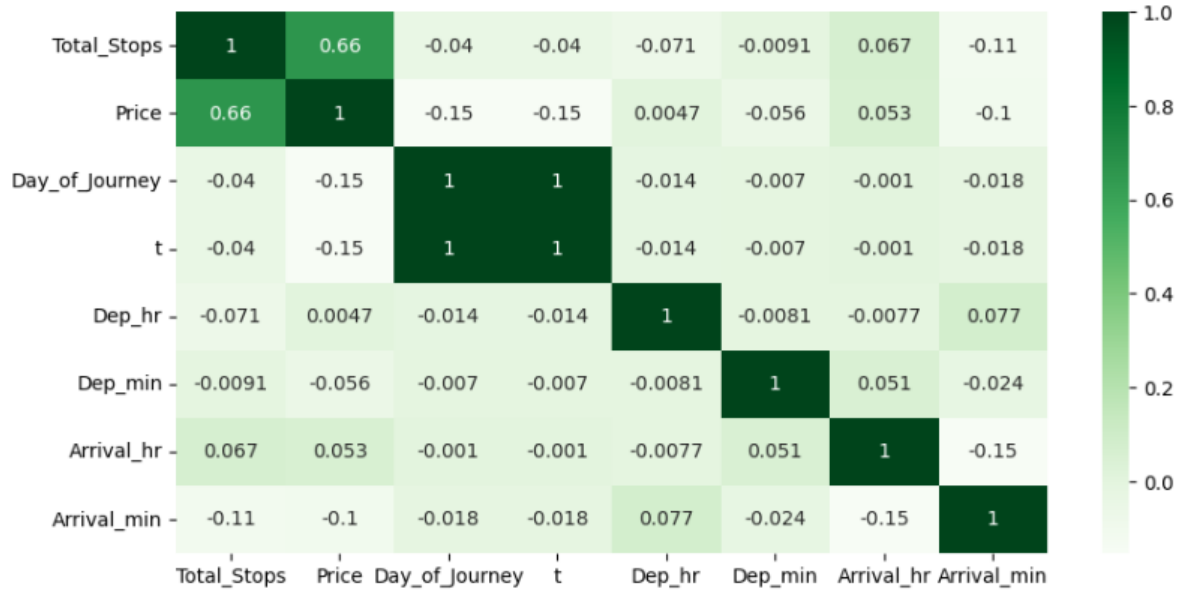
```
continuous_column_list = ['Price']

for column_name in continuous_column_list:
    while True:
        Q1 = df[column_name].quantile(0.25)
        Q3 = df[column_name].quantile(0.75)
        IQR = Q3 - Q1

        df_cleaned_outlier = df[~((df[column_name] < (Q1 - 1.5 * IQR)) | (df[column_name] > (Q3 + 1.5 * IQR)))]

        if len(df_cleaned_outlier) == len(df):
            break
        df = df_cleaned_outlier
print(df.shape[0])
print(df.shape[1])
```

```
10164
14
```

E. Bỏ các cột có thể gây ra hiện tượng đa cộng tuyến:

```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True, cmap="Greens");
```

```
<ipython-input-115-c459dbe5cd81>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. S
```



- Không có cột có thể gây ra hiện tượng đa cộng tuyến.

VIẾT FUNCTION TỔNG HỢP TẤT CẢ CÁC BƯỚC PREPROCESSING TRÊN:

```
def clean_data(df):
```

```
    # Chuyển cột 'Date_of_Journey' thành kiểu dữ liệu
datetime
    df['Date_of_Journey'] =
pd.to_datetime(df['Date_of_Journey'], format='%d/%m/%Y')

    # Tạo cột 'Day_of_Journey' từ 'Date_of_Journey'
    df['Day_of_Journey'] = df['Date_of_Journey'].dt.day

    # Tính toán cột 't'
    df['t'] = df['Date_of_Journey'].dt.day -
df['Day_of_Journey'].min()

    # Xử lý cột 'Dep_Time'
    df['Dep_hr'] = pd.to_datetime(df['Dep_Time']).dt.hour
    df['Dep_min'] =
pd.to_datetime(df['Dep_Time']).dt.minute
    df.drop(["Dep_Time"], axis=1, inplace=True)

    # Xử lý cột 'Arrival_Time'
    df['Arrival_hr'] =
pd.to_datetime(df['Arrival_Time']).dt.hour
    df['Arrival_min'] =
pd.to_datetime(df['Arrival_Time']).dt.minute
    df.drop(["Arrival_Time"], axis=1, inplace=True)

    # Xử lý cột 'Duration'
    duration = df['Duration'].str.split(' ', expand=True)
    duration[1].fillna('00m', inplace=True)
    df['Duration_hr'] = duration[0].apply(lambda x:
int(x[:-1]))
    df['Duration_min'] = duration[1].apply(lambda x:
int(x[:-1]))
    df.drop(["Duration"], axis=1, inplace=True)

    # Xử lý cột 'Dep_hr' và 'Arrival_hr' để tạo cột mới dựa
trên thời gian
    df['Deps'] = df['Dep_hr'].apply(lambda x: "Morning" if
6 <= x <= 12 else ("Afternoon" if 12 < x <= 18 else
("Evening" if 18 < x < 24 else 'Night')))
```

```
    df['Arrivals'] = df['Arrival_hr'].apply(lambda x:
"Morning" if 6 <= x <= 12 else ("Afternoon" if 12 < x <= 18
else ("Evening" if 18 < x < 24 else 'Night')))

    # Xử lý cột 'Total_Stops'
    df['Total_Stops'] =
df['Total_Stops'].str.extract(r'(\d+)').fillna('0').astype(
int)

    # Xóa các hàng có giá trị NaN
    df.dropna(how='any', inplace=True)

    # Loại bỏ các bản ghi trùng lặp
    df.drop_duplicates(keep=False, inplace=True)

    #xử lý outlier biến Price

    continuous_column_list = ['Price']

    for column_name in continuous_column_list:
        while True:
             Q1 = df[column_name].quantile(0.25)
             Q3 = df[column_name].quantile(0.75)
             IQR = Q3 - Q1

             df_cleaned_outlier = df[~((df[column_name] <
(Q1 - 1.5 * IQR)) | (df[column_name] > (Q3 + 1.5 * IQR)))]

             if len(df_cleaned_outlier) == len(df):
                 break
             df = df_cleaned_outlier
    print(df.shape[0])
    print(df.shape[1])

    df = df.drop(['Day_of_Journey','Date_of_Journey',
'Route', 'Additional_Info', 'Dep_hr',
'Dep_min',  'Arrival_hr','Arrival_min'], axis=1)

    df.info()
```

```
    return df

def optimize_data(df):
    # Clean data
    df_cleaned = clean_data(df)

    return df_cleaned

df = pd.read_excel('Data_Train.xlsx')

df_optimized = optimize_data(df)

df_optimized.head()
```

5. **DATAVISUALIZATION:**

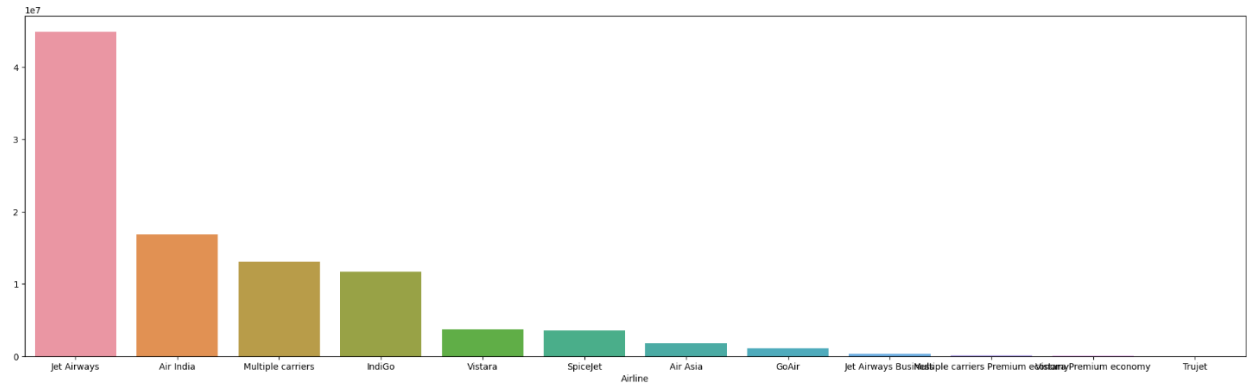
- **Lợi nhuận của mỗi chuyến bay:**

```
airline_price_data = df.groupby('Airline')['Price'].sum().sort_values(ascending=False)
airline_price_data
```

```
Airline
Jet Airways                          44817461
Air India                            16838841
Multiple carriers                    13039603
IndiGo                               11648071
Vistara                               3734451
SpiceJet                              3548717
Air Asia                              1783293
GoAir                                 1137045
Jet Airways Business                   350152
Multiple carriers Premium economy      148445
Vistara Premium economy                 26887
Trujet                                   4140
Name: Price, dtype: int64
```

```
plt.figure(figsize=(25,7))
sns.barplot(x=airline_price_data.index, y=airline_price_data.values)
```

```
<Axes: xlabel='Airline'>
```

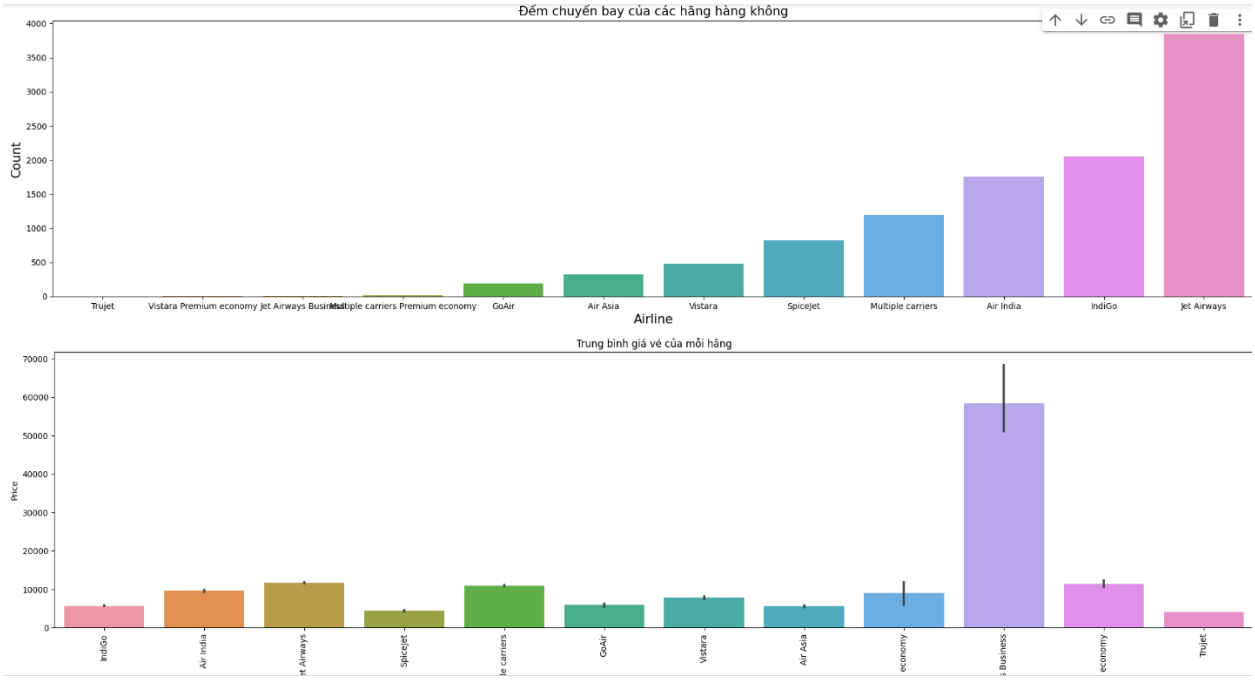


⇨ Hãng bay Jet Airways có số người chi tiền vào nhiều nhất, xếp thứ hai là Air India. Tuy nhiên mức độ chi tiền vào vé bay có giá cao của khách hàng thuộc Air India nhiều hơn, đường trên biểu đồ của Air India phổ rộng và cao ở khoảng giữa 5000 đến 15000.

- **Số lượng chuyến bay giữa các hãng hàng không, so sánh giá:**

```
count_airline = df['Airline'].value_counts().sort_values()
count_airline
```

```
Trujet                             1
Vistara Premium economy            3
Jet Airways Business               6
Multiple carriers Premium economy  13
GoAir                              194
Air Asia                           319
Vistara                            479
SpiceJet                           818
Multiple carriers                  1196
Air India                          1752
IndiGo                             2053
Jet Airways                        3849
Name: Airline, dtype: int64
```



⇨ Hãng bay Jet Aiways luôn chiếm tỷ lệ cao nhất về số lượng chuyến bay. Giá của Jet Airway Business cao nhất trong tổng số những hãng bay. Có thể lí giải do đây là loại hình Business.

⇨ Loại trừ Business thì giá thành của những hãng bay thì JetAirway có mức giá cao hơn so với những hãng còn lại tuy nhiên số lượng chuyến bay vẫn chiếm tỉ lệ cao nhất.

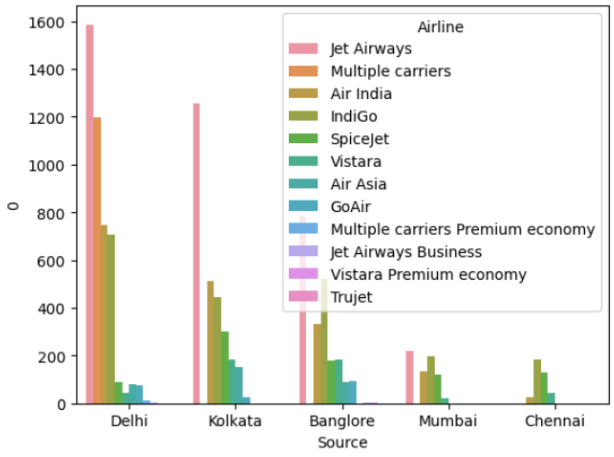
- **Hãng hàng không nào cung cấp nhiều chuyến bay nhất giữa thành phố nguồn và thành phố điểm đến?:**

```
airline_source_des = df.groupby(['Airline','Source','Destination']).size().sort_values(ascending=False).reset_index()
airline_source_des
```

	Airline	Source	Destination	0
0	Jet Airways	Delhi	Cochin	1586
1	Jet Airways	Kolkata	Banglore	1256
2	Multiple carriers	Delhi	Cochin	1196
3	Air India	Delhi	Cochin	747
4	IndiGo	Delhi	Cochin	705
5	Air India	Kolkata	Banglore	512
6	IndiGo	Kolkata	Banglore	445
7	Jet Airways	Banglore	New Delhi	418
8	Jet Airways	Banglore	Delhi	370
9	IndiGo	Banglore	Delhi	366
10	SpiceJet	Kolkata	Banglore	300
11	Jet Airways	Mumbai	Hyderabad	219
12	Air India	Banglore	New Delhi	212
13	IndiGo	Mumbai	Hyderabad	196
14	IndiGo	Chennai	Kolkata	184
15	Vistara	Kolkata	Banglore	183

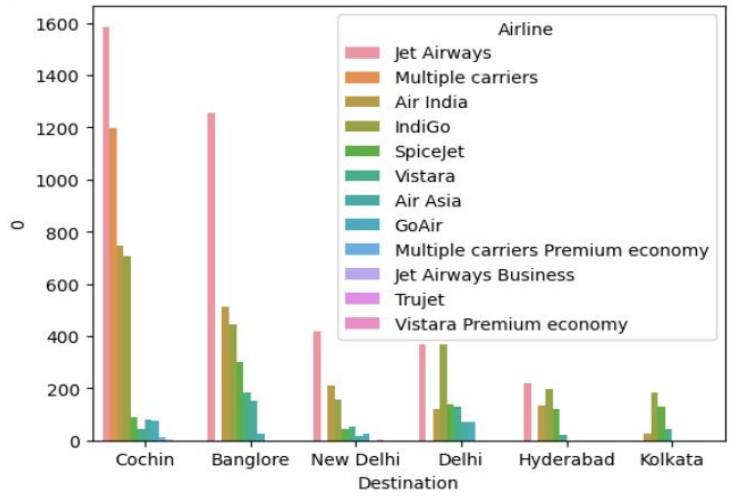
```
sns.barplot(data=airline_source, x='Source', y=0, hue='Airline')
```

```
<Axes: xlabel='Source', ylabel='0'>
```



```
sns.barplot(data=airline_des, x='Destination', y=0, hue='Airline')
```

```
<Axes: xlabel='Destination', ylabel='0'>
```



⇨ **Dù xuất phát hay hạ cánh thì Jet Airways vẫn luôn dẫn đầu về số lượng chuyến bay tại cả điểm nguồn và điểm đến.**

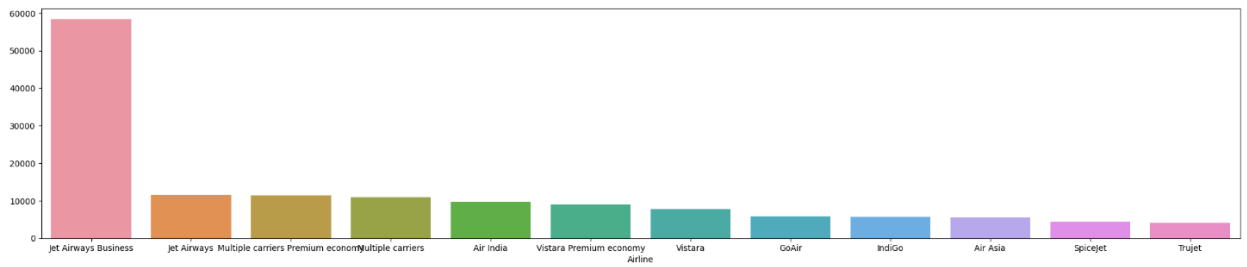
- **Trung bình giá vé của mỗi hãng:**

```
airline_mean_price = df.groupby('Airline')['Price'].mean().sort_values(ascending=False)
airline_mean_price
```

```
Airline
Jet Airways Business                58358.666667
Jet Airways                         11643.923357
Multiple carriers Premium economy   11418.846154
Multiple carriers                   10902.678094
Air India                            9611.210616
Vistara Premium economy              8962.333333
Vistara                              7796.348643
GoAir                                5861.056701
IndiGo                               5673.682903
Air Asia                             5590.260188
SpiceJet                             4338.284841
Trujet                               4140.000000
Name: Price, dtype: float64
```

```
plt.figure(figsize=(26,5))
sns.barplot(x=airline_mean_price.index, y=airline_mean_price.values)
```

```
<Axes: xlabel='Airline'>
```



⇨ Giá vé của hãng Jet Airways Business có giá vé trung bình cao nhất (do là loại hình Business).

⇨ Nếu không so sánh Loại hình Business thì giá vé của Jet Airways trung bình vẫn cao nhất và Giá vé của Trujet là trung bình thấp nhất.

- **Điểm đến và điểm xuất phát nào phổ biến nhất?**

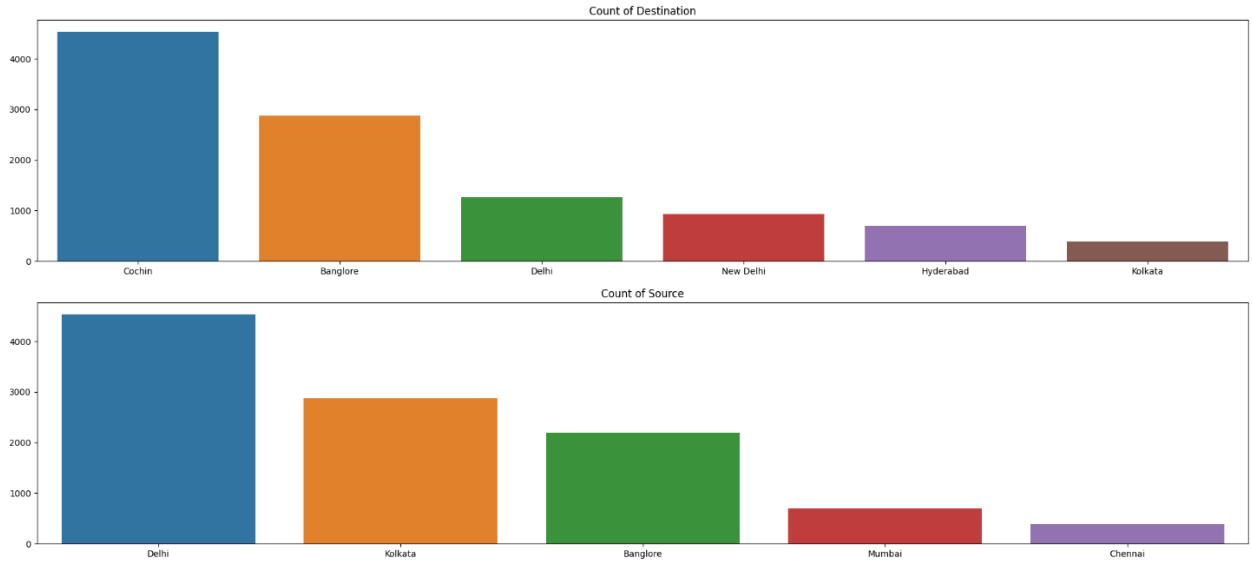
```
des_popular = df["Destination"].value_counts()
des_popular
```

```
Cochin       4537
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

```
source_popular = df["Source"].value_counts()
source_popular
```

```
Delhi       4537
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

```
plt.figure(figsize = (25,5))
sns.barplot(x=des_popular.index, y=des_popular.values)
plt.title("Count of Destination")

plt.figure(figsize = (25,5))
sns.barplot(x=source_popular.index, y=source_popular.values)
plt.title("Count of Source")
```



⇨ Người dân tại thành phố Cochin có số lượng chuyến bay khở hành cao nhất

⇨ Người dân tại thành phố Delhi có số lượng chuyến bay hạ cánh tới nhiều nhất

- **Đếm số chuyến bay vào mỗi buổi theo giờ khởi hành:**

```
count_deps = df["Deps"].value_counts()
count_deps
```

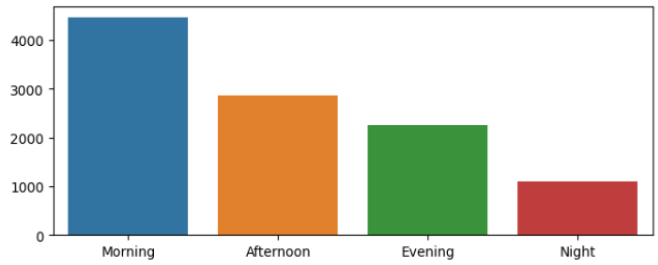
```
Morning      4461
Afternoon    2870
Evening      2258
Night        1094
Name: Deps, dtype: int64
```

```
mean_price_deps = df.groupby(['Deps'])['Price'].mean().sort_values(ascending=False)
mean_price_deps
```

```
Deps
Afternoon    9342.222648
Morning      9136.609280
Night        8842.213894
Evening      8783.494686
Name: Price, dtype: float64
```

```
plt.figure(figsize=(8, 3 ))
sns.barplot(x=count_deps.index, y=count_deps.values)
```

```
<Axes: >
```

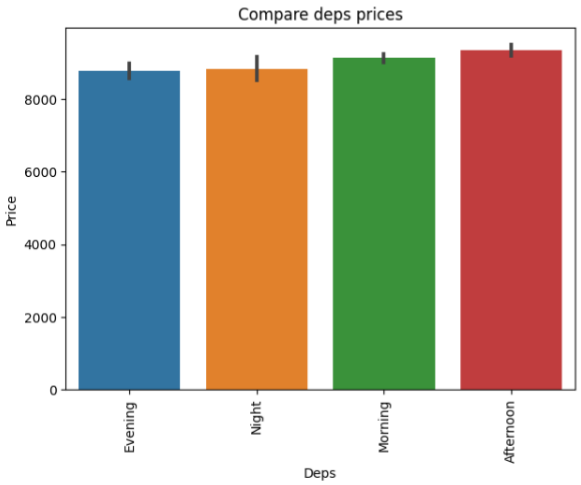


⇨ Người dân có xu hướng bay vào buổi sáng sớm nhiều nhất

⇨ Tuy nhiên thì giá vé máy bay vào buổi đêm có trung bình rẻ nhất.

- **So sánh giá của vé máy bay trong mỗi buổi theo giờ khởi hành:**

```
# Create a bar plot to compare deps prices
plt.figure(figsize=(7, 5))
sns.barplot(x='Deps', y='Price', data=df)
plt.title('Compare deps prices')
plt.xlabel('Deps')
plt.ylabel('Price')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.show()
```

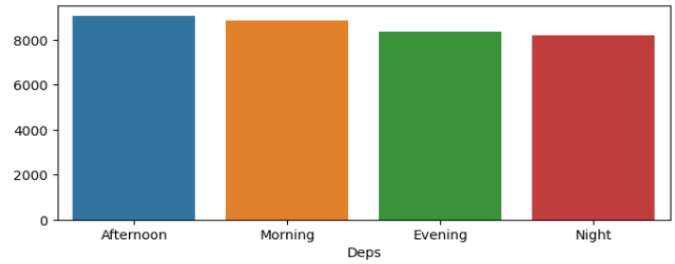


⇨ **Giá vé máy bay vào buổi tối thấp nhất.**

- **Trung bình giá vé vào mỗi buổi theo giờ bay:**

```
mean_price_deps = df.groupby(['Deps'])['Price'].mean().sort_values(ascending=False)
plt.figure(figsize=(8, 3 ))
sns.barplot(x=mean_price_deps.index, y=mean_price_deps.values)

print(mean_price_deps)
```

```
Deps
Afternoon    9087.323306
Morning      8889.650407
Evening      8368.563360
Night        8208.499496
Name: Price, dtype: float64
```



⇨ Giá vé vào buổi chiều trung bình cao nhất.

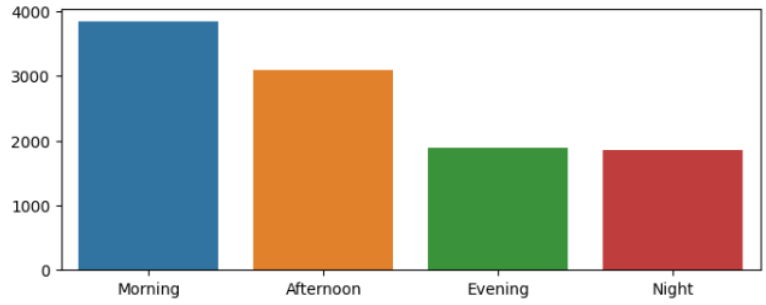
- **Đếm và số chuyến bay hạ cánh trong mỗi buổi phân theo giờ hạ cánh:**

```
count_arrival = df["Arrivals"].value_counts()
count_arrival
```

```
Evening      3665
Morning      2981
Afternoon    1816
Night        1702
Name: Arrivals, dtype: int64
```

```
plt.figure(figsize=(8, 3 ))
sns.barplot(x=count_deps.index, y=count_arrival.values)
```

```
<Axes: >
```



⇨ **Số chuyến bay hạ cánh vào buổi tối có số lượng cao nhất.**

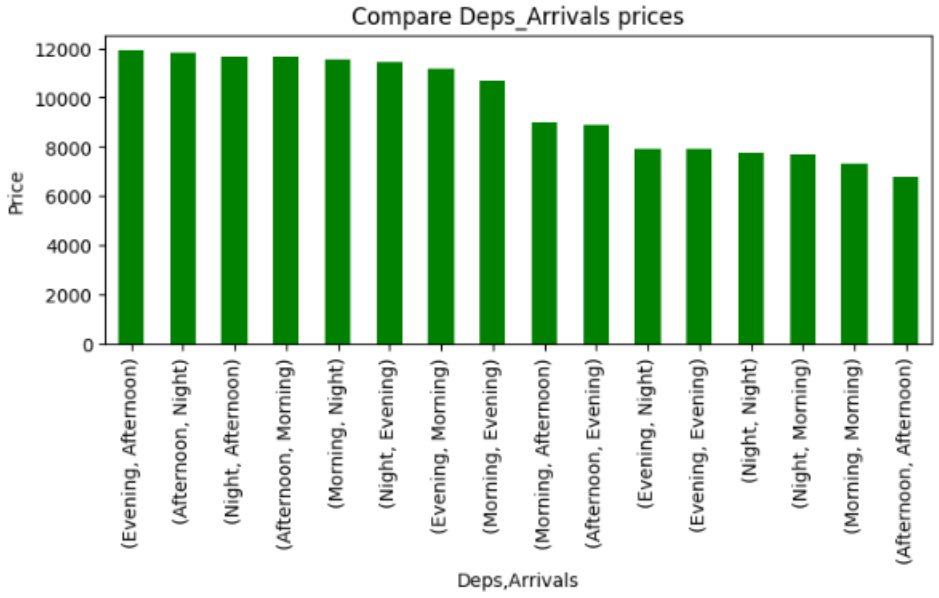
- **Giá thay đổi như thế nào dựa trên giờ khởi hành và giờ đáp:**

```
change_price = df.groupby(['Deps','Arrivals'])['Price'].mean().sort_values(ascending=False)
change_price
```

```
Deps       Arrivals
Evening    Afternoon    11946.270492
Afternoon  Night        11841.108607
Night      Afternoon    11678.467290
Afternoon  Morning      11643.538793
Morning    Night        11539.948148
Night      Evening      11456.362791
Evening    Morning      11155.400000
Morning    Evening      10660.232574
           Afternoon     8973.996904
Afternoon  Evening       8866.297771
Evening    Night         7908.796834
           Evening       7896.877996
Night      Night         7739.240343
           Morning       7713.218924
Morning    Morning       7290.753968
Afternoon  Afternoon     6790.099698
Name: Price, dtype: float64
```

```
fig = plt.figure(figsize = (8, 3))
change_price.plot(kind='bar', color = 'green')
plt.title('Compare Deps_Arrivals prices')
plt.ylabel('Price')
```

```
Text(0, 0.5, 'Price')
```



Giờ khởi hành và giờ cất cánh vào buổi chiều thì giá vé sẽ thấp nhất hoặc là cả hai đều vào buổi sáng

=> Có thể thấy tại giờ khởi hành thì số lượng chuyến bay khởi hành vào buổi sáng cất cánh nhiều nhất => Buổi chiều là giờ hạ cánh nhiều nhất => Nhưng xét về giá thay đổi thì mức giá vào buổi chiều khi khởi hành sẽ có giá trị trung bình rẻ nhất

- Giá vé dựa trên Source và Destination places:

```
df['Source'].value_counts()
```

```
Delhi       4537
Kolkata     2871
Banglore    2197
Mumbai       697
Chennai      381
Name: Source, dtype: int64
```

```
df['Destination'].value_counts()
```

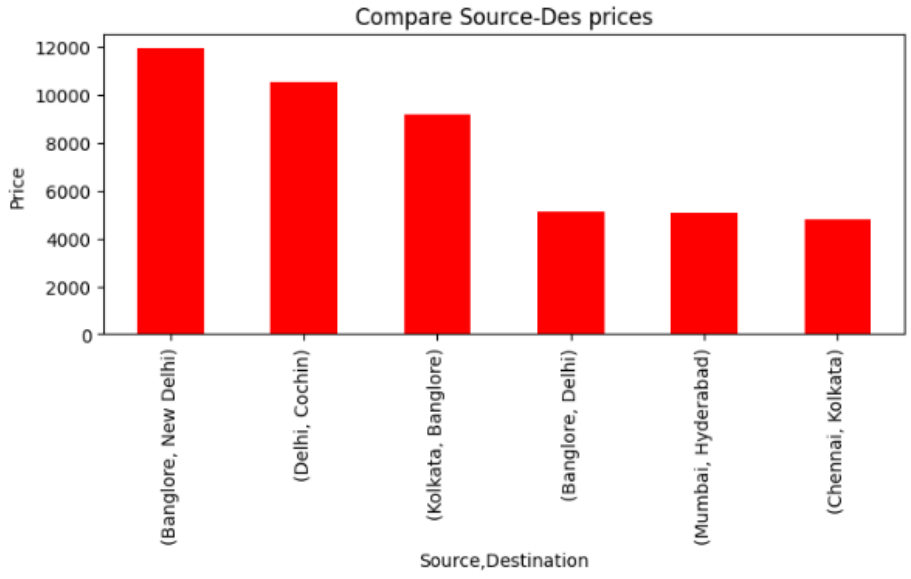
```
Cochin       4537
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: Destination, dtype: int64
```

```
price_source_des = df.groupby(['Source','Destination'])['Price'].mean().sort_values(ascending=False)
price_source_des
```

```
Source    Destination
Banglore  New Delhi      11917.716738
Delhi     Cochin         10539.439057
Kolkata   Banglore        9158.389411
Banglore  Delhi           5143.918577
Mumbai    Hyderabad       5059.708752
Chennai   Kolkata         4789.892388
Name: Price, dtype: float64
```

```
fig = plt.figure(figsize = (8, 3))
price_source_des.plot(kind='bar', color = 'red')
plt.title('Compare Source-Des prices')
plt.xlabel('Source,Destination')
plt.ylabel('Price')
```

```
Text(0, 0.5, 'Price')
```



Chennai -> Kolkata => có giá thấp nhất

- **Giá vé dựa trên Route:**

```
df['Total_Stops'].value_counts().sort_values(ascending=False)
```

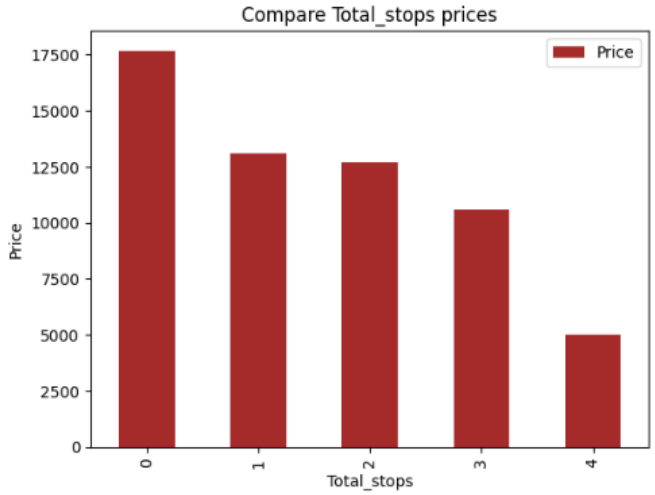
```
1    5625
0    3492
2    1520
3      45
4       1
Name: Total_Stops, dtype: int64
```

```
price_totalstops = df.groupby(['Total_Stops'])['Price'].mean().sort_values(ascending=False).reset_index()
price_totalstops
```

	Total_Stops	Price
0	4	17686.000000
1	3	13112.000000
2	2	12715.807895
3	1	10594.123556
4	0	5025.603379

```
fig = plt.figure(figsize = (8, 3))
price_totalstops.plot(kind='bar', color = 'brown')
plt.title('Compare Total_stops prices')
plt.xlabel('Total_stops')
plt.ylabel('Price')
```

```
Text(0, 0.5, 'Price')
<Figure size 800x300 with 0 Axes>
```



⇨ **Không có trạm dừng sẽ chiếm tỷ lệ cao nhất.**

IV. XÂY DỰNG MÔ HÌNH

1. Dummy Encoding:

Trước khi chạy mô hình, sử dụng kỹ thuật Dummy Encoding để chuyển các biến phân loại về dạng số:

Viết Function xử lý biến phân loại bằng phương pháp Dummy Encoding:

```
def encode_data(df):
    # Danh sách các biến liên tục và phân loại để dễ dàng xử lý
    continuous_column_list = ['Total_Stops', 'Price', 't', 'Duration_hr', 'Duration_min']
    all_columns = df.columns.tolist()
    categorical_column_list = [col for col in all_columns if col not in
continuous_column_list]

    # Chuyển đổi kiểu dữ liệu của các biến phân loại sang string
    df[categorical_column_list] = df[categorical_column_list].astype(str)
    df_categorical = pd.get_dummies(df[categorical_column_list], drop_first=True)

    # Kết hợp DataFrame của các biến liên tục và DataFrame biến phân loại đã chuyển đổi
    df_combined = df[continuous_column_list].join(df_categorical)

    return df_combined

def optimize_data(df):
    # Clean data
    df_cleaned = clean_data(df)

    # Encode data
    df_encoded = encode_data(df_cleaned)

    # Chuyển kiểu dữ liệu các cột
    df_cast = df_encoded.astype(int)

    # Loại bỏ cột 'Duration_hr'
    df_cast.drop(['Duration_hr'], axis=1, inplace=True)

    return df_cast

df = pd.read_excel('Data_Train.xlsx')

df_optimized = optimize_data(df)

df_optimized.head()
```

- **Kết quả**

	Total_Stops	Price	t	Duration_min	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	...	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata	Destination_New Delhi	Deps_Evening	Deps_Morning	Deps_
0	0	3897	23	50	0	0	1	0	0	0	...	0	0	0	1	1	0	
1	2	7662	0	25	1	0	0	0	0	0	...	0	0	0	0	0	0	
2	2	13882	8	0	0	0	0	1	0	0	...	0	0	0	0	0	1	
3	1	6218	11	25	0	0	1	0	0	0	...	0	0	0	0	0	0	
4	1	13302	0	45	0	0	1	0	0	0	...	0	0	0	1	0	0	

5 rows × 30 columns

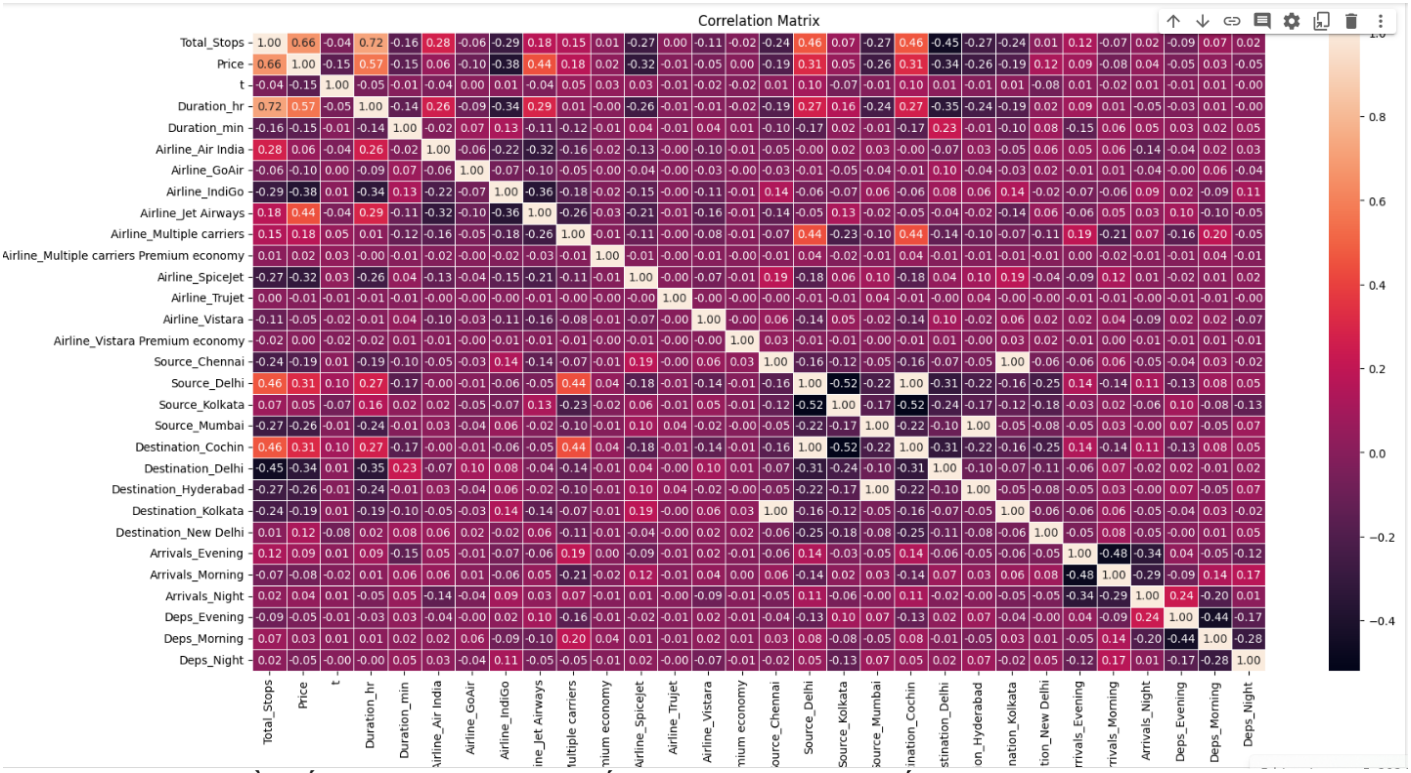
2. Phân tích mối tương quan giữa các biến – Tương quan Pearson:

Tương quan Pearson là thước đo phổ biến nhất để đánh giá mối tương quan giữa các biến, giá trị trong khoảng từ -1 đến 1:

- -1 biểu thị mối tương quan nghịch hoàn hảo (khi một biến tăng thì biến kia giảm).
- 0 biểu thị không có mối tương quan.
- 1 biểu thị mối tương quan dương hoàn hảo (khi một biến tăng thì biến kia cũng tăng).

⇨ Dùng headmap để thấy rõ tương quan giữa các biến:

```
pearsons = df_cast.corr()
plt.figure(figsize=(20, 10))
sns.heatmap(pearsons, annot=True,  fmt=".2f", linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



⇨ Nhận xét về mối tương quan giữa biến Price với những biến còn lại:

- Tương quan cao nhất là Price - Total_Stops: 0.66 => Số chặn dừng sẽ ảnh hưởng nhiều đến giá vé.

Kết luận:

- Chọn biến Total_Stops làm biến độc lập cho mô hình hồi quy đơn biến.
- Biến Total_stops và Duration_hr có giá trị gần nhau => sẽ gây ra hiện tượng đa cộng tuyến khi chạy mô hình hồi quy đa biến => Xóa Duration_hr.

3. MÔ HÌNH HỒI QUY ĐƠN BIẾN:

Dùng phương pháp bình phương nhỏ nhất OLS :

- OLS là một phương pháp ước lượng thống kê hiệu quả. OLS tìm ra các hệ số hồi quy sao cho tổng bình phương sai số (TSS) là nhỏ nhất. Điều này đảm bảo rằng mô hình ước lượng được gần với dữ liệu thực tế nhất có thể.
- OLS là một phương pháp ước lượng có thể được giải thích dễ dàng. OLS cho phép chúng ta hiểu được mối quan hệ giữa biến phụ thuộc và các biến độc lập.
- OLS là một phương pháp ước lượng có thể được mở rộng dễ dàng. OLS có thể được sử dụng để ước lượng các mô hình hồi quy tuyến tính đơn biến, đa biến, hồi quy với sai số cố định hoặc hồi quy với sai số ngẫu nhiên.

```
df_cast['Price'].unique()
```

```
array([ 3897,  7662, 13882, ..., 12352, 11733, 12648])
```

```
[132] import statsmodels.api as sm
      from sklearn.linear_model import LinearRegression
      from sklearn.model_selection import train_test_split
      from pandas.core.common import random_state
      from sklearn.metrics import mean_squared_error,mean_absolute_error ,r2_score
```

Do Hệ số tương quan của biến Total_Stops là lớn nhất trong tổng số biến => chọn biến cho mô hình hồi quy tuyến tính đơn biến.

```
[133] predict='Price'
      x=np.array(df_cast['Total_Stops'])
      y=np.array(df_cast[predict])

      # predict='Price'
      # x=np.array(df1['Duration_hr'])
      # y=np.array(df1[predict])

      # predict='Price'
      # x=np.array(df1['Airline_IndiGo'])
      # y=np.array(df1[predict])
```

```
[134] import statsmodels.api as sm
      X_sm = sm.add_constant(x)
      ols = sm.OLS(y,X_sm.astype(float)).fit()
```

```
print(ols.summary())
```

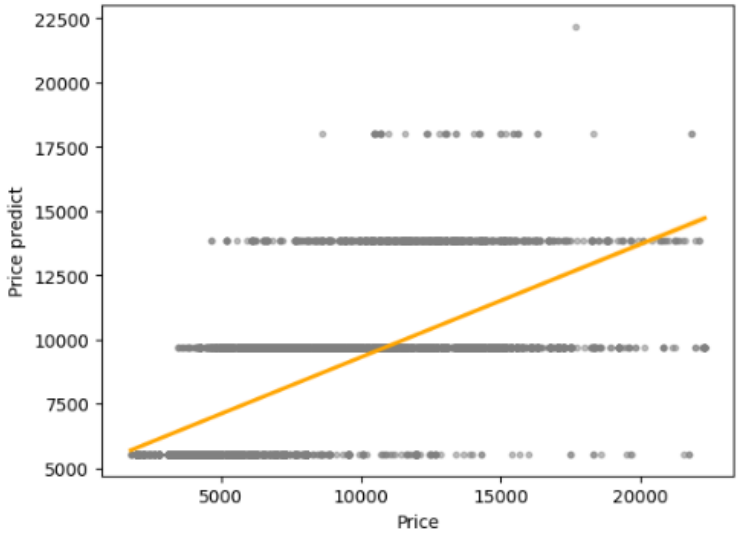
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.440
Model:                            OLS   Adj. R-squared:                  0.440
Method:                 Least Squares   F-statistic:                     7987.
Date:                Fri, 05 Jan 2024   Prob (F-statistic):               0.00
Time:                        14:24:04   Log-Likelihood:                -95922.
No. Observations:               10164   AIC:                         1.918e+05
Df Residuals:                   10162   BIC:                         1.919e+05
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       5525.4666     47.163    117.157      0.000    5433.018    5617.915
x1          4162.7264     46.578     89.371      0.000    4071.425    4254.028
==============================================================================
Omnibus:                      863.000   Durbin-Watson:                   2.004
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1163.363
Skew:                           0.717   Prob(JB):                    2.39e-253
Kurtosis:                       3.830   Cond. No.                         2.77
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

⇨ **Mô hình hồi quy đơn biến có ý nghĩa thống kê với biến Total_Stops có ý nghĩa nhưng hệ số R-squared chỉ có 0.440 =>. Nguyên nhân là do thiếu biến. Do đó ta đi tới mô hình hồi quy đa biến.**

Chart:

```
import seaborn as sns

sns.regplot(data=df_pred, x='Price', y='Price predict',
            scatter_kws=dict(color='gray', s=10, alpha=0.5),
            line_kws=dict(color='orange'))
```

```
<Axes: xlabel='Price', ylabel='Price predict'>
```



4. MÔ HÌNH HỒI QUY ĐA BIẾN:

⇨ **Xóa biến Duration_hr vì có khả năng gây ra hiện tượng đa cộng tuyến khi chạy mô hình hồi quy đa biến.**

⇨ **Do xây dựng mô hình dự đoán giá vé máy bay nên sẽ tiếp tục dùng biến Price là biến phụ thuộc và tất cả những biến còn lại là biến độc lập.**

a. **Sử dụng function Linear regression:**

```
predict='Price'
Xs = np.array(df_optimized.drop(['Price'], axis = 1))
y = np.array(df_optimized[predict])

reg = LinearRegression()
reg.fit(Xs, y)
```

```
LinearRegression()
```

```
print(reg.coef_)
print(reg.intercept_)
```

```
[ 2.85064680e+03 -7.07934772e+01  1.21332247e+00  1.54937264e+03
 -4.48758604e+01  1.53679825e+02  4.28920967e+03  4.76463917e+04
  3.42438407e+03  4.63856476e+03 -2.75543246e+02 -2.40054502e+03
  2.17885691e+03  3.23238944e+03 -5.35159154e+01 -4.54318255e+01
 -2.86484764e+02 -8.62986405e+02 -4.54318255e+01 -1.10613464e+03
 -8.62986405e+02 -5.35159154e+01  2.35455355e+03 -3.00062557e+02
 -3.15805360e+02 -3.24253955e+02 -9.19156930e+01 -2.00638321e+02
  2.57093614e+02]
5751.638575461406
```

```
reg.score(Xs, y)
```

```
0.6158483830905654
```

b. **Sử dụng mô hình hồi quy OLS trong hồi quy tuyến tính:**

```
predict='Price'
x1=np.array(df_cast.drop(['Price'], axis = 1))
y1=np.array(df_cast[predict])
```

```
import statsmodels.api as sm
X_sm = sm.add_constant(x1)
ols = sm.OLS(y1,X_sm.astype(float)).fit()
```

```
print(ols.summary())
```

OLS Regression Results

Dep. Variable:	y	R-squared:	0.643
Model:	OLS	Adj. R-squared:	0.642
Method:	Least Squares	F-statistic:	731.3
Date:	Fri, 05 Jan 2024	Prob (F-statistic):	0.00
Time:	14:24:15	Log-Likelihood:	-93631.
No. Observations:	10164	AIC:	1.873e+05
Df Residuals:	10138	BIC:	1.875e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4170.3711	137.366	30.360	0.000	3901.106	4439.636
x1	2606.8675	65.935	39.537	0.000	2477.622	2736.113
x2	-56.6497	2.779	-20.386	0.000	-62.097	-51.203
x3	13.7262	4.587	2.992	0.003	4.735	22.718
x4	2.4054	1.536	1.566	0.117	-0.605	5.416
x5	1525.4069	156.765	9.731	0.000	1218.117	1832.697
x6	23.5108	223.518	0.105	0.916	-414.629	461.650
x7	224.4307	148.693	1.509	0.131	-67.037	515.898
x8	4113.6192	147.428	27.903	0.000	3824.631	4402.608
x9	3265.7550	163.465	19.978	0.000	2945.331	3586.179
x10	4537.8030	689.630	6.580	0.000	3185.992	5889.614
x11	-262.2748	163.864	-1.601	0.110	-583.481	58.931
x12	-2077.2519	2435.105	-0.853	0.394	-6850.539	2696.035
x13	2219.6361	179.263	12.382	0.000	1868.245	2571.027
x14	3792.8049	1409.672	2.691	0.007	1029.568	6556.042
x15	497.3087	62.462	7.962	0.000	374.870	619.747
x16	605.1147	34.310	17.636	0.000	537.859	672.370
x17	978.4169	56.332	17.369	0.000	867.995	1088.839
x18	-318.6232	48.412	-6.581	0.000	-413.520	-223.726
x19	605.1147	34.310	17.636	0.000	537.859	672.370
x20	42.4280	73.994	0.573	0.566	-102.615	187.471
x21	-318.6232	48.412	-6.581	0.000	-413.520	-223.726
x22	497.3087	62.462	7.962	0.000	374.870	619.747
x23	2365.7261	81.483	29.033	0.000	2206.003	2525.449
x24	-280.1620	73.221	-3.826	0.000	-423.689	-136.635
x25	-261.7548	62.257	-4.204	0.000	-383.792	-139.718
x26	-294.2258	95.060	-3.095	0.002	-480.562	-107.890
x27	-57.2216	73.072	-0.783	0.434	-200.456	86.013
x28	-148.0376	74.938	-1.975	0.048	-294.931	-1.144
x29	216.6184	88.605	2.445	0.015	42.936	390.301

Omnibus:	840.854	Durbin-Watson:	2.021
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1548.070

⇨ **Mô hình chạy với kết quả khả quan là 0.643, tuy nhiên vẫn còn vài biến với p-values không có nghĩa thống kê (P-value thường được dùng để lựa chọn các biến có ý nghĩa thống kê trong quá trình xây dựng mô hình.)**
Xóa những biến có p-values > 0.05
Biến x3, x5, x6, x11, x12, x14, x20, x27 có p-value > 0. 05 => Không có ý nghĩa thống kê => Loại.

⇨ **Loại bỏ những biến đó ra khỏi mô hình và tiếp tục chạy như cũ.**

```
df_pred1 = df_optimized
```

```
predict='Price'
x2=np.array(df_pred1.drop(['Price', 'Duration_min',  'Airline_GoAir','Airline_IndiGo', 'Airline_SpiceJet', 'Airline_Trujet', 'Airline_Vistara Premium economy',   'Destination_Delhi', 'Arrivals_Evening
y2=np.array(df_pred1[predict])
```

```
import statsmodels.api as sm
X_sm = sm.add_constant(x2)
ols = sm.OLS(y2,X_sm.astype(float)).fit()
```

```
print(ols.summary())
```

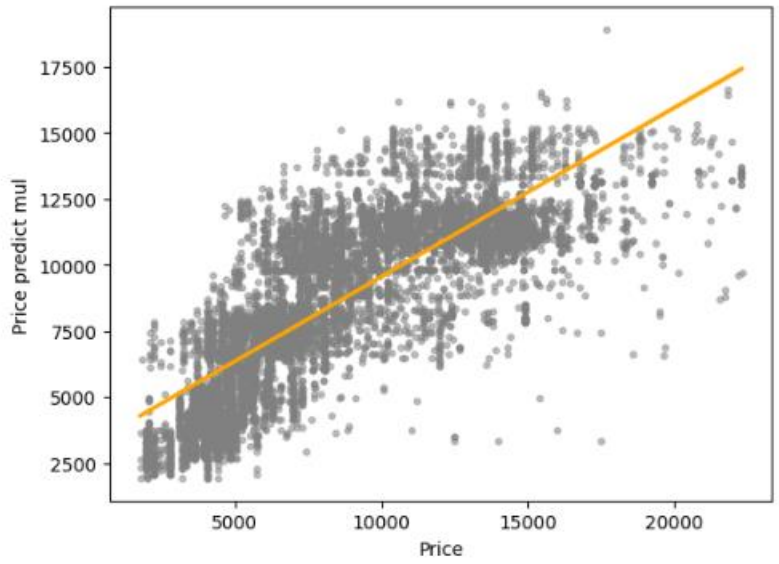
```
print(ols.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.615
Model:                            OLS   Adj. R-squared:                  0.614
Method:                 Least Squares   F-statistic:                     909.6
Date:                Tue, 09 Jan 2024   Prob (F-statistic):               0.00
Time:                        02:56:19   Log-Likelihood:                -96302.
No. Observations:               10263   AIC:                         1.926e+05
Df Residuals:                   10244   BIC:                         1.928e+05
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       4677.5256    113.301     41.284      0.000    4455.433    4899.618
x1          2851.5097     66.241     43.047      0.000    2721.664    2981.355
x2           -71.1447      3.394    -20.963      0.000     -77.797     -64.492
x3          1509.5087     99.636     15.150      0.000    1314.204    1704.814
x4          4249.9529     77.432     54.886      0.000    4098.171    4401.734
x5           4.76e+04   1180.381     40.324      0.000    4.53e+04    4.99e+04
x6          3349.0172    112.299     29.822      0.000    3128.889    3569.145
x7          4599.2166    802.947      5.728      0.000    3025.284    6173.149
x8          2153.4564    143.786     14.977      0.000    1871.608    2435.305
x9           474.5082     85.203      5.569      0.000     307.494     641.523
x10          505.1279     60.382      8.366      0.000     386.768     623.488
x11          795.6390    110.915      7.173      0.000     578.224    1013.054
x12         -326.4001     68.450     -4.768      0.000    -460.575    -192.225
x13          505.1279     60.382      8.366      0.000     386.768     623.488
x14         -326.4001     68.450     -4.768      0.000    -460.575    -192.225
x15          474.5082     85.203      5.569      0.000     307.494     641.523
x16         3463.7491    134.972     25.663      0.000    3199.178    3728.320
x17         -306.3272     85.173     -3.597      0.000    -473.283    -139.371
x18         -308.7250     72.997     -4.229      0.000    -451.813    -165.637
x19         -301.6076    111.686     -2.701      0.007    -520.533     -82.682
x20         -171.3262     69.361     -2.470      0.014    -307.288     -35.365
x21          324.3847     83.642      3.878      0.000     160.431     488.338
==============================================================================
Omnibus:                     5509.333   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           135635.887
Skew:                           2.070   Prob(JB):                         0.00
Kurtosis:                      20.322   Cond. No.                     4.03e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.44e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Chart mô hình hồi quy đa biến:

```
import seaborn as sns

sns.regplot(data=df_optimized, x='Price', y='Price predict mul',
            scatter_kws=dict(color='gray', s=10, alpha=0.5),
            line_kws=dict(color='orange'))
```

```
<Axes: xlabel='Price', ylabel='Price predict mul'>
```



Nhận xét:

- **Mô hình trên là mô hình có ý nghĩa nhất sau tất cả các mô hình với tất cả các biến đều có giá trị.**
- **P-values đạt < 0.05 => tất cả các biến đều có ý nghĩa thống kê.**
- **Durbin-Watson đạt => không có sự tương quan.**

- **Những biến không có ý nghĩa thống kê bao gồm**: 'Duration_min', 'Airline_GoAir','Airline_IndiGo', 'Airline_SpiceJet', 'Airline_Trujet', 'Airline_Vistara Premium economy', 'Destination_Delhi', 'Arrivals_Evening'

5. ĐÁNH GIÁ MÔ HÌNH – TIẾN HÀNH MÔ HÌNH HỒI QUY TIẾN HOẶC LÙI CHỌN MÔ HÌNH HỒI QUY TỐT NHẤT:

```
import statsmodels.api as sm

# R-squared
R_squared = ols.rsquared
# Giá trị tương quan
correlation = np.sqrt(R_squared)

print("Giá trị R-squared:", R_squared)
print("Giá trị tương quan:", correlation)
```

```
Giá trị R-squared: 0.6383494943481387
Giá trị tương quan: 0.7989677680283096
```

Giải thích hai chỉ số

- Giá trị R-squared là một thước đo độ phù hợp của mô hình hồi quy tuyến tính đa bội. Giá trị này đo lường mức độ biến thiên của biến phụ thuộc được giải thích bởi các biến độc lập. Giá trị R-squared càng cao thì mô hình càng phù hợp.
- Giá trị tương quan là một thước đo mối quan hệ tuyến tính giữa hai biến. Giá trị tương quan càng cao thì mối quan hệ tuyến tính giữa hai biến càng mạnh.

Kết luận dựa trên hai chỉ số đánh giá mô hình:

- Trong trường hợp này, giá trị R-squared là 0,642648394527124, cho thấy các biến độc lập có thể giải thích được 64,26% biến thiên của biến phụ thuộc. Giá trị tương quan là 0,8016535377125981, cho thấy hai biến có mối quan hệ tuyến tính dương mạnh.
- Mô hình hồi quy có độ phù hợp tốt. Giá trị R-squared là 0.638, tương ứng với 63.8% biến thiên của biến phụ thuộc (y) được giải thích bởi các biến độc lập (x). Giá trị này nằm trong khoảng từ 0.5 đến 1, được coi là mức độ phù hợp tốt của mô hình.
- Mối quan hệ giữa biến phụ thuộc và biến độc lập là tuyến tính. Giá trị tương quan (correlation) là 0.799, rất gần với 1. Điều này cho thấy mối quan hệ giữa hai biến là tuyến tính, nghĩa là khi biến độc lập tăng lên 1 đơn vị, biến phụ thuộc sẽ tăng lên 0.799 đơn vị.
- Từ hai chỉ số này, chúng ta có thể kết luận rằng mô hình hồi quy tuyến tính đa bội này phù hợp với dữ liệu và có mối quan hệ tuyến tính dương mạnh giữa biến phụ thuộc và các biến độc lập.

HỒI QUY TIẾN HOẶC LÙI – CHỌN MÔ HÌNH HỒI QUY TỐT NHẤT:

```
def backward_stepwise_regression(X, y, significance_level=0.05):
    remaining_features = list(X.columns)
    selected_features = []
    rsquared_values = []
    best_model = None

    while len(remaining_features) > 0:
        models = {}
        for feature in remaining_features:
            features = selected_features + [feature]
            X_temp = X[features]
            X_with_intercept = sm.add_constant(X_temp)
            model = sm.OLS(y, X_with_intercept).fit()
            models[feature] = model.rsquared_adj
```

```
        best_feature = max(models, key=models.get)
        remaining_features.remove(best_feature)
        selected_features.append(best_feature)
        rsquared_values.append(models[best_feature])

    best_features = selected_features[:len(selected_features)-1]  # Lấy danh sách biến trước khi dừng
    X_best = X[best_features] if best_features else X.copy()

    while True:
        X_with_intercept = sm.add_constant(X_best)
        best_model = sm.OLS(y, X_with_intercept).fit()
        p_values = best_model.pvalues[1:]  # Exclude intercept p-value
        max_p_value = p_values.max()

        if max_p_value > significance_level:
            feature_to_remove = p_values.idxmax()
            if feature_to_remove == 'const':
                break
            else:
                X_best = X_best.drop(columns=[feature_to_remove])
        else:
            break

    X_with_intercept = sm.add_constant(X_best)
    best_model = sm.OLS(y, X_with_intercept).fit()

    return best_model

# Sử dụng dữ liệu df_optimized từ quá trình xử lý trước đó
X = df_optimized.drop(columns=['Price'])  # Features
y = df_optimized['Price']  # Target

best_ols_model = backward_stepwise_regression(X, y)

if best_ols_model:
    print("Best model summary:")
    print(best_ols_model.summary())
```

Best model summary:

OLS Regression Results

Dep. Variable:	Price	R-squared:	0.616
Model:	OLS	Adj. R-squared:	0.615
Method:	Least Squares	F-statistic:	863.2
Date:	Sat, 06 Jan 2024	Prob (F-statistic):	0.00
Time:	16:30:11	Log-Likelihood:	-96296.
No. Observations:	10263	AIC:	1.926e+05
Df Residuals:	10243	BIC:	1.928e+05
Df Model:	19		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	4755.6966	115.513	41.170	0.000	4529.268	4982.125
Total_Stops	2838.9648	66.307	42.815	0.000	2708.989	2968.940
Airline_Jet Airways	4163.1384	81.426	51.128	0.000	4003.527	4322.749
Airline_Jet Airways Business	4.753e+04	1179.943	40.279	0.000	4.52e+04	4.98e+04
Airline_Multiple carriers	3280.8231	113.988	28.782	0.000	3057.385	3504.262
Destination_New Delhi	3469.5469	134.911	25.717	0.000	3205.094	3734.000
t	-70.8947	3.393	-20.896	0.000	-77.545	-64.244
Airline_Vistara	2052.8455	146.674	13.996	0.000	1765.335	2340.356
Airline_Air India	1425.2709	102.568	13.896	0.000	1224.218	1626.324
Source_Mumbai	-308.8115	68.606	-4.501	0.000	-443.292	-174.331
Airline_Multiple carriers Premium economy	4529.0789	802.786	5.642	0.000	2955.461	6102.696
Arrivals_Night	333.2250	83.637	3.984	0.000	169.279	497.171
Airline_SpiceJet	-408.3706	119.076	-3.429	0.001	-641.783	-174.958
Arrivals_Morning	-144.2689	69.772	-2.068	0.039	-281.036	-7.501
Source_Kolkata	822.1895	111.127	7.399	0.000	604.359	1040.020
Deps_Morning	-309.9995	72.960	-4.249	0.000	-453.014	-166.985
Deps_Evening	-318.4940	85.202	-3.738	0.000	-485.507	-151.481
Deps_Night	-314.2216	111.688	-2.813	0.005	-533.151	-95.292
Destination_Hyderabad	-308.8115	68.606	-4.501	0.000	-443.292	-174.331
Source_Chennai	1012.3762	171.316	5.909	0.000	676.564	1348.188
Source_Delhi	504.5588	60.350	8.360	0.000	386.260	622.857
Destination_Cochin	504.5588	60.350	8.360	0.000	386.260	622.857

Omnibus:	5518.469	Durbin-Watson:	1.994
Prob(Omnibus):	0.000	Jarque-Bera (JB):	136543.719
Skew:	2.073	Prob(JB):	0.00
Kurtosis:	20.381	Cond. No.	1.33e+17

Kết luận:

- Trong số những mô hình hồi quy đã chạy thì mô hình hồi quy tốt nhất là mô hình bao gồm 21 biến có ý nghĩa thống kê.
- Mô hình hồi quy có ý nghĩa thống kê với R-squared = 0.616 tương đối tốt.

6. ỨNG DỤNG CHẠY MÔ HÌNH HỒI QUY ĐỂ DỰ ĐOÁN GIÁ VÉ MÁY BAY

Yêu cầu:

- Người dùng nhập các tham số cần thiết mà nhóm đưa ra
- Kết quả giá vé máy bay, giá cao nhất và thấp nhất được dự báo bằng mô hình hồi quy tốt nhất ở yêu cầu 2

Cách làm:

- Tạp biến cho phép user nhập những giá trị của những biến có ý nghĩa thống kê của mô hình tốt nhất đã chạy.

- Kết quả bao gồm: Kết quả giá vé máy bay, giá cao nhất và thấp nhất được dự báo bằng mô hình hồi quy tốt nhất ở yêu cầu 2.

```
# Tạo dictionary chứa các giá trị cho biến phân loại
airline_options = {
    'Air India': 0,
    'Jet Airways': 0,
    'Jet Airways Business': 0,
    'Multiple carriers': 0,
    'Multiple carriers Premium economy': 0,
    'Vistara': 0,
    'SpiceJet': 0
}
source_options = {
    'Mumbai': 0,
    'Kolkata': 0,
    'Chennai': 0,
    'Delhi': 0
}
destination_options = {
    'New Delhi': 0,
    'Cochin': 0,
    'Hyderabad': 0
}
deps_options = {
    'Morning': 0,
```

```
    'Evening': 0,
    'Night': 0
}
arrivals_options = {
    'Morning': 0,
    'Night': 0
}
def select_value_from_list(options):
    print("Danh sách giá trị có sẵn:")
    for idx, key in enumerate(options.keys()):
        print(f"{idx + 1}. {key}")

    while True:
        choice = input("Nhập số thứ tự của giá trị bạn muốn chọn: ")
        if choice.isdigit() and 0 < int(choice) <= len(options):
            selected_key = list(options.keys())[int(choice) - 1]
            break
        else:
            print("Vui lòng nhập lại số thứ tự hợp lệ.")

    # Cập nhật giá trị được chọn thành 1 và các giá trị còn lại trong cùng một nhóm thành 0
    for key in options.keys():
        if key == selected_key:
            options[key] = 1
```

```
        else:
            options[key] = 0

    return options

# ... (Phần định nghĩa airline_options, source_options, destination_options, deps_options, arrivals_options)

# Tạo DataFrame chứa tất cả các cột với giá trị mặc định là 0
initial_columns = ['Total_Stops', 't', 'Duration_min'] + list(airline_options.keys()) + list(source_options.keys()) + list(destination_options.keys()) + list(deps_options.keys()) + list(arrivals_options.keys())
initial_values = [0] * len(initial_columns)

initial_df = pd.DataFrame([initial_values], columns=initial_columns)

# Nhập giá trị cho các biến liên tục
total_stops = int(input("Nhập giá trị cho Total_Stops: "))
t_value = int(input("Nhập giá trị cho t: "))
# Chọn giá trị cho các biến phân loại
airline_choice = select_value_from_list(airline_options)
source_choice = select_value_from_list(source_options)
destination_choice = select_value_from_list(destination_options)
deps_choice = select_value_from_list(deps_options)
arrivals_choice = select_value_from_list(arrivals_options)
```

```
# Gán giá trị 1 tương ứng với lựa chọn của người dùng vào DataFrame mới
user_input = {
    'Total_Stops': total_stops,
    't': t_value,
    **airline_choice,
    **source_choice,
    **destination_choice,
    **deps_choice,
    **arrivals_choice
}
# Cập nhật DataFrame mới với lựa chọn của người dùng
for col, val in user_input.items():
    initial_df.at[0, col] = val
# Sử dụng mô hình để dự đoán giá trị Price dựa trên dữ liệu người dùng nhập vào
# input_with_intercept = sm.add_constant(initial_df)
# predicted_price = best_ols_model.predict(input_with_intercept)


# print("Dữ liệu nhập của người dùng:")
# print(initial_df)
# print("\nGiá trị dự đoán là:", predicted_price[0])
# Sử dụng mô hình để dự đoán giá trị Price dựa trên dữ liệu người dùng nhập vào
input_with_intercept=sm.add_constant(initial_df)
predicted_values = best_ols_model.predict(input_with_intercept)
```

```
std_error = np.std(best_ols_model.resid)

# Tính toán khoảng dự đoán

min_predicted_price = predicted_values[0] - 1.96 * std_error

# Giả sử mức tin cậy 95%

max_predicted_price = predicted_values[0] + 1.96 * std_error

# Giả sử mức tin cậy 95%

print("Dữ liệu nhập của người dùng:")


print(initial_df)

print("\nGiá trị dự đoán là:", predicted_values[0])

print("Khoảng dự đoán 95%: từ", min_predicted_price, "đến",
max_predicted_price)
```

KẾT QUẢ:

```
Nhập giá trị cho Total_Stops: 1
Nhập giá trị cho t: 2
Danh sách giá trị có sẵn:
1. Air India
2. Jet Airways
3. Jet Airways Business
4. Multiple carriers
5. Multiple carriers Premium economy
6. Vistara
7. SpiceJet
Nhập số thứ tự của giá trị bạn muốn chọn: 1
Danh sách giá trị có sẵn:
1. Mumbai
2. Kolkata
3. Chennai
4. Delhi
Nhập số thứ tự của giá trị bạn muốn chọn: 2
Danh sách giá trị có sẵn:
1. New Delhi
2. Cochin
3. Hyderabad
Nhập số thứ tự của giá trị bạn muốn chọn: 1
Danh sách giá trị có sẵn:
1. Morning
2. Evening
3. Night
Nhập số thứ tự của giá trị bạn muốn chọn: 3
Nhập số thứ tự của giá trị bạn muốn chọn: 1
Danh sách giá trị có sẵn:
1. Morning
2. Night
Dữ liệu nhập của người dùng:
   Total_Stops  t  Duration_min  Air India  Jet Airways  Jet Airways Business  \
0            1  2             0          1            0                     0

   Multiple carriers  Multiple carriers Premium economy  Vistara  SpiceJet  \
0                  0                                  0        0         0

   ...  Chennai  Delhi  New Delhi  Cochin  Hyderabad  Morning  Evening  Night  \
0  ...        0      0          1       0          0        1        0      0

   Morning  Night
0        1      0

[1 rows x 22 columns]

Giá trị dự đoán là: 59306.369739536334
Khoảng dự đoán 95%: từ 53671.09784542673 đến 64941.64163364594
```

File dữ liệu:
https://docs.google.com/spreadsheets/d/1WcskCOp5ZcgHHAf2OT380U1z6lBNiTtMJHOjuK4a77I/edit?usp=sharing

File Google Colab phân tích dự đoán giá vé máy bay:
https://colab.research.google.com/drive/12V3zD-HTjCUC_JpsFbNsOesWqp-Ovygb?usp=sharing