flight-prediction-jupyter

June 25, 2024

[61]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error,r2_score
from sklearn.model_selection import KFold, cross_val_predict
import math
import missingno as msno
```

1) Exploratory Data Analysis (EDA)

[63]:
```python
# Importing the csv file and creating a dataframe using pandas
data = pd.read_csv("Clean_Dataset.csv")

# Printing the head of the dataframe
display(data.head())

# Printing the column names of the dataframe
print(data.columns)
```

```
   Unnamed: 0   airline   flight source_city departure_time stops  \
0           0  SpiceJet  SG-8709       Delhi        Evening  zero
1           1  SpiceJet  SG-8157       Delhi  Early_Morning  zero
2           2   AirAsia   I5-764       Delhi  Early_Morning  zero
3           3   Vistara   UK-995       Delhi        Morning  zero
4           4   Vistara   UK-963       Delhi        Morning  zero

    arrival_time destination_city    class  duration  days_left  price
0          Night           Mumbai  Economy      2.17          1   5953
1        Morning           Mumbai  Economy      2.33          1   5953
2  Early_Morning           Mumbai  Economy      2.17          1   5956
3      Afternoon           Mumbai  Economy      2.25          1   5955
4        Morning           Mumbai  Economy      2.33          1   5955
```

```
Index(['Unnamed: 0', 'airline', 'flight', 'source_city', 'departure_time',
       'stops', 'arrival_time', 'destination_city', 'class', 'duration',
```

```
       'days_left', 'price'],
      dtype='object')
```

[65]:
```
# General statistics
print(f'Data shape :\n {data.shape}\n')
print(f'Data types :\n {data.dtypes}\n')
display(data.iloc[:, 1:].describe()) # Excluding the fisrt Unname: 0 column
display(data.info())
```

```
Data shape :
 (300153, 12)

Data types :
 Unnamed: 0            int64
airline              object
flight               object
source_city          object
departure_time       object
stops                object
arrival_time         object
destination_city     object
class                object
duration            float64
days_left             int64
price                 int64
dtype: object
```

	duration	days_left	price
count	300153.000000	300153.000000	300153.000000
mean	12.221021	26.004751	20889.660523
std	7.191997	13.561004	22697.767366
min	0.830000	1.000000	1105.000000
25%	6.830000	15.000000	4783.000000
50%	11.250000	26.000000	7425.000000
75%	16.170000	38.000000	42521.000000
max	49.830000	49.000000	123071.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 12 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Unnamed: 0        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  float64
 10  days_left         300153 non-null  int64
 11  price             300153 non-null  int64
dtypes: float64(1), int64(3), object(8)
memory usage: 27.5+ MB
```

```
None
```

[ ]:
```
# Checking for missing values
display(data.isnull().sum())
msno.matrix(data)
```

No missing values detected

[16]:
```
# Checking for duplicate values if any according to unique valued column␣
↪Unnamed: 0
display(data.duplicated("Unnamed: 0").sum())
```

```
0
```

No duplicated values detected

[18]:
```
# Inspecting the unique values of some columns

categorical_columns = ["airline", "source_city", "destination_city",␣
↪"departure_time", "arrival_time", "stops", "class"]

for column in categorical_columns:
    print(f'{column} values:\n {data[column].unique()}\n')
```

```
airline values:
 ['SpiceJet' 'AirAsia' 'Vistara' 'GO_FIRST' 'Indigo' 'Air_India']

source_city values:
 ['Delhi' 'Mumbai' 'Bangalore' 'Kolkata' 'Hyderabad' 'Chennai']

destination_city values:
 ['Mumbai' 'Bangalore' 'Kolkata' 'Hyderabad' 'Chennai' 'Delhi']

departure_time values:
 ['Evening' 'Early_Morning' 'Morning' 'Afternoon' 'Night' 'Late_Night']

arrival_time values:
 ['Night' 'Morning' 'Early_Morning' 'Afternoon' 'Evening' 'Late_Night']

stops values:
```

```
 ['zero' 'one' 'two_or_more']

class values:
 ['Economy' 'Business']
```

Counting values

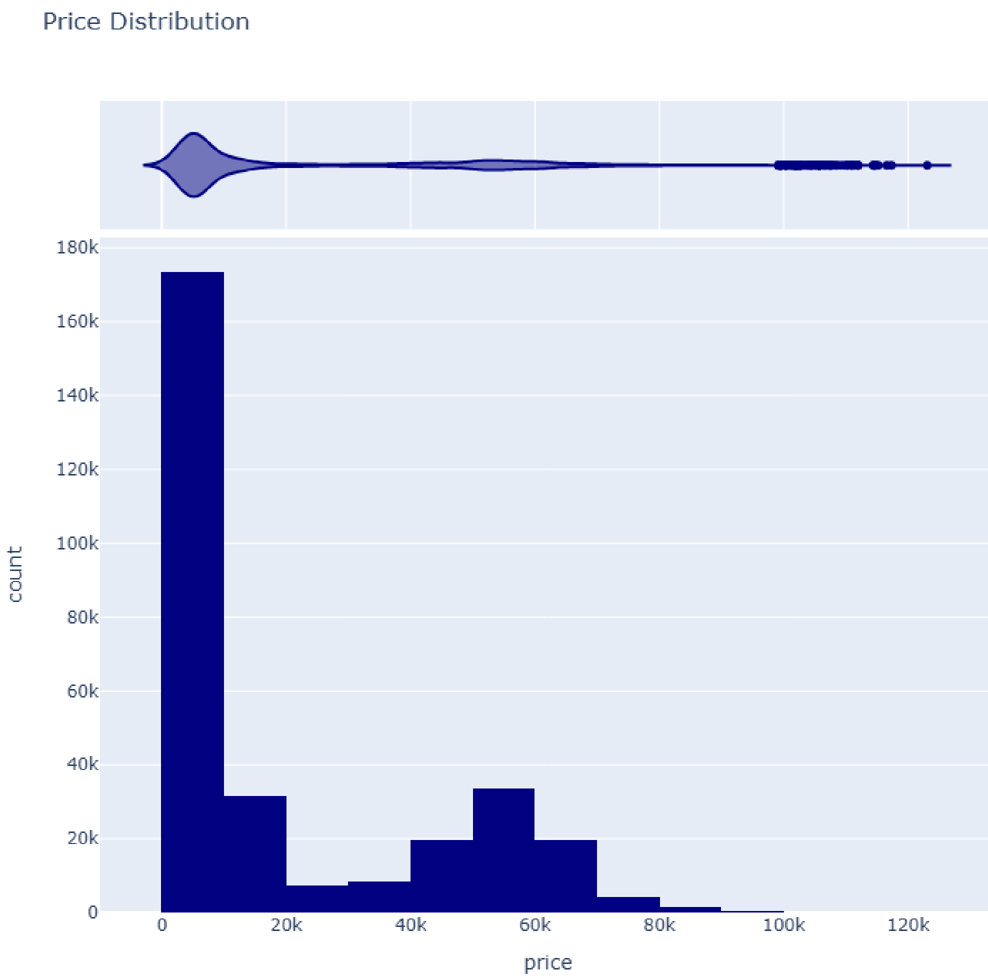
[ ]:
```
for column in categorical_columns:
    # Getting the value counts for the current column
    column_data = data[column].value_counts().reset_index() #converting␣
 ↪value_counts() into a dataframe
    column_data.columns = [column, 'count'] # Setting the name of the columns␣
 ↪in the dataframe

    # Create a histogram with Plotly Express
    fig = px.bar(column_data,
                 x=column,
                 y='count',
                 color=column,
                 color_continuous_scale='Viridis',
                 labels={'count': 'Count', column: column},
                 title=f"Frequency of {column}")

    # Show the plot
    fig.show()
```

Let s examine price distribution.

[27]:
```
display(data.iloc[:, -1:].describe())
price_range = data['price'].max() - data['price'].min()
print(f'The range of price is: {price_range}')


# Creating the histogram
fig = px.histogram(
    data,
    x='price',
    nbins=20,
    marginal='violin',
    title='Price Distribution',
    color_discrete_sequence=['navy'],
    width=800,
    height=750

)
# Show the plot
fig.show()
```

```
                price
count   300153.000000
mean     20889.660523
std      22697.767366
min       1105.000000
25%       4783.000000
50%       7425.000000
75%      42521.000000
max     123071.000000
```

```
The range of price is: 121966
```

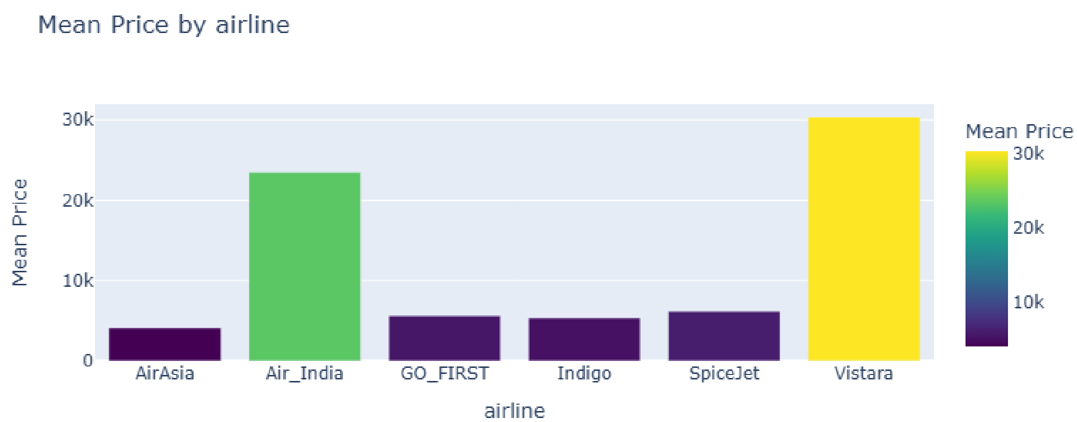


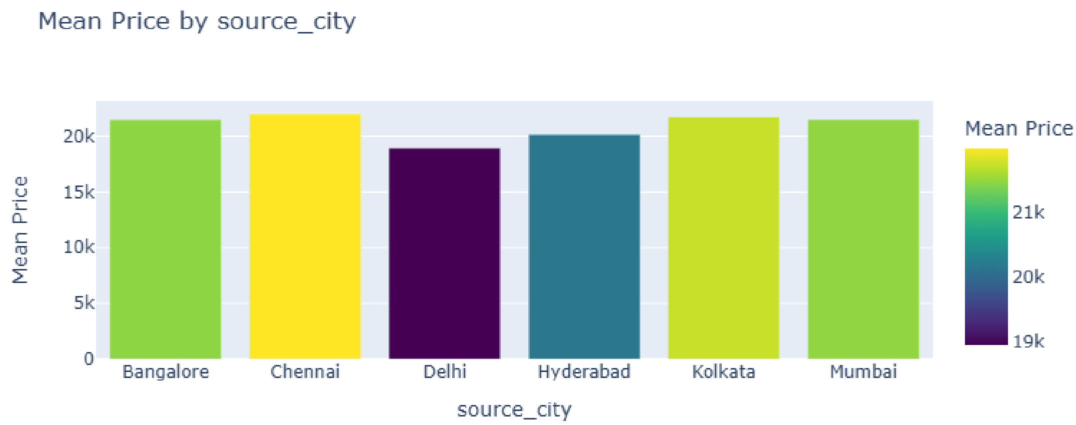
As we can see there are two edges. The first appears in low prices (0 < price < 15000) and the second in mid prices (500000 < price < 70000).

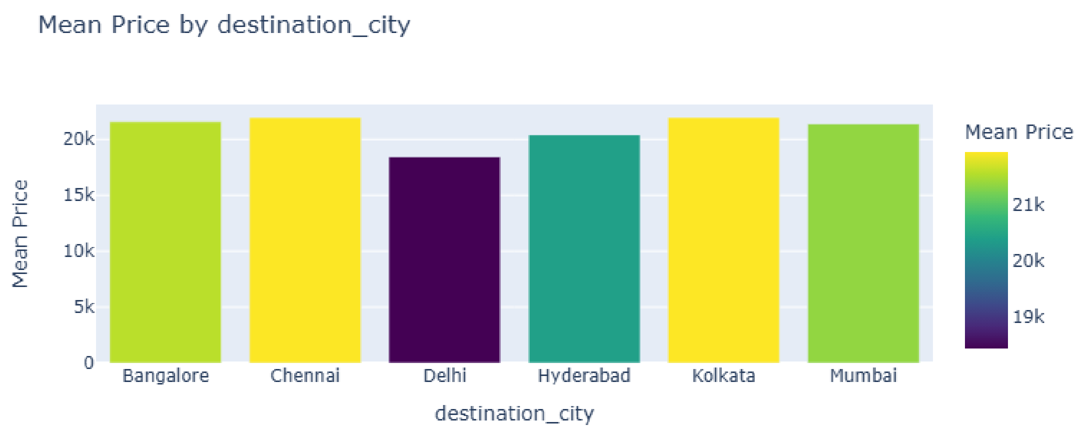
Let s check the mean price by each categorical column

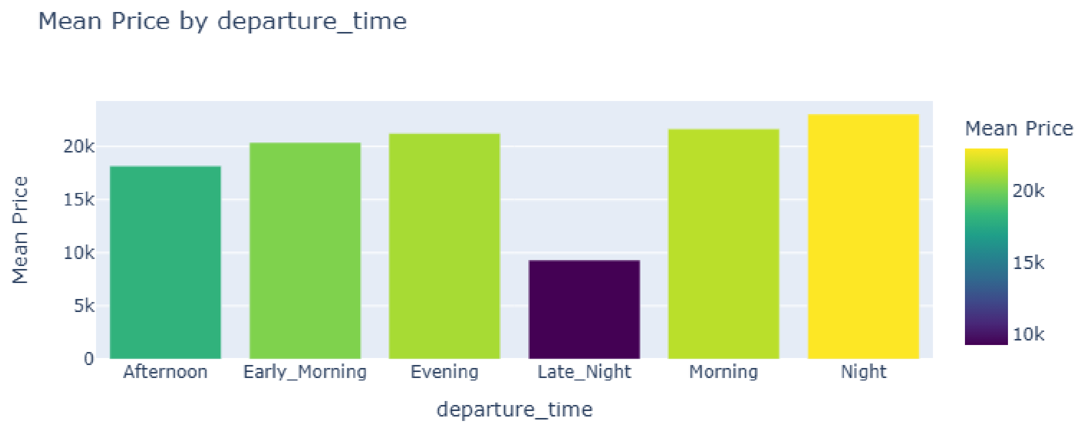
[30]:

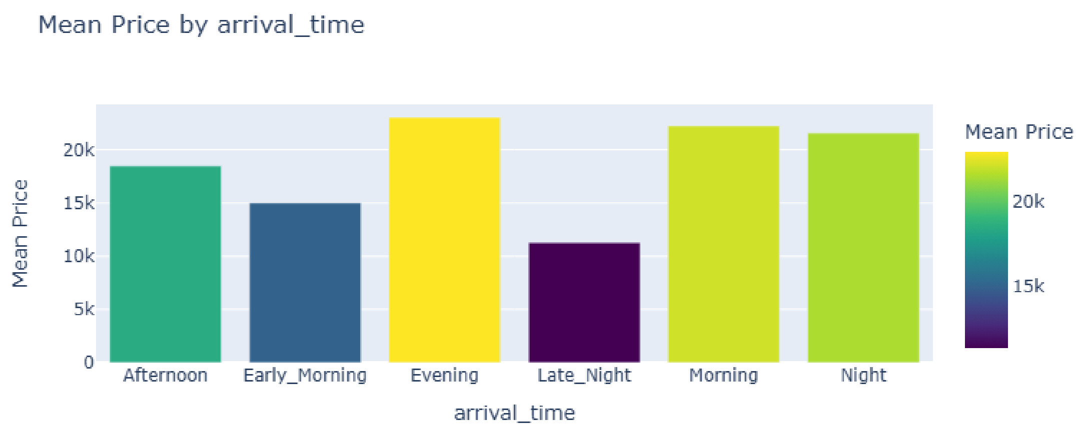
```
for column in categorical_columns:
    # Computing the mean price for each category
    mean_prices = data.groupby(column)['price'].mean().reset_index()

    # Creating a bar plot
    fig = px.bar(
        mean_prices,
        x=column,
        y='price',
        labels={'price': 'Mean Price', column: column},
        title=f"Mean Price by {column}",
        color='price',
        color_continuous_scale='Viridis'
    )

    # Show the plot
    fig.show()
```

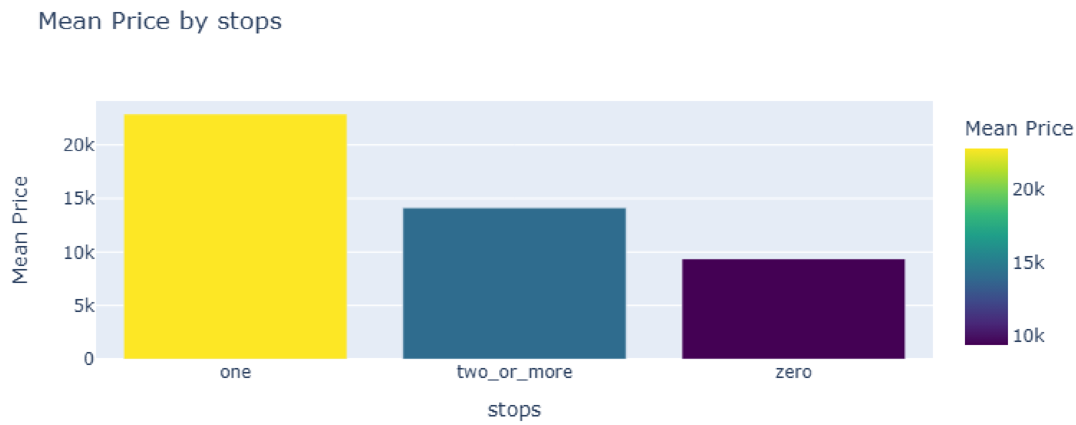


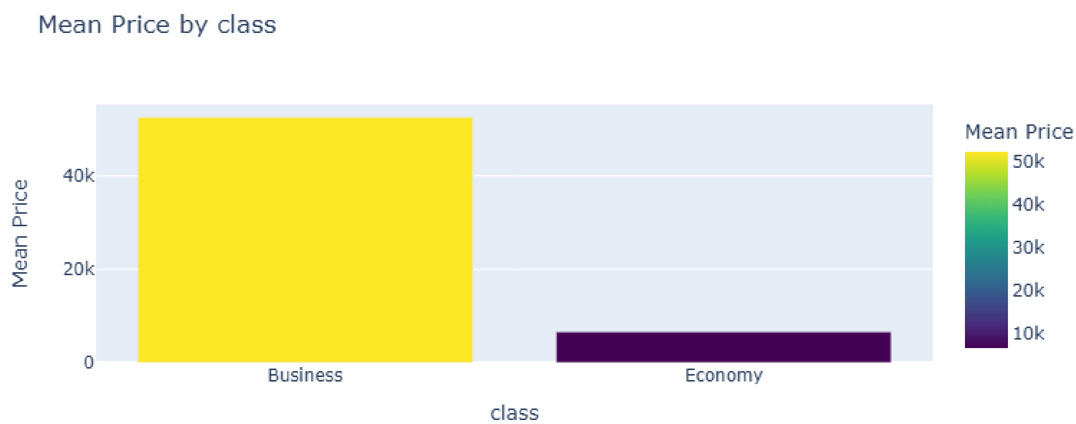

Mean Price by source_city
Mean Price
20k
15k
10k
5k
0
Bangalore
Chennai
Delhi
Hyderabad
Kolkata
Mumbai
source_city
Mean Price
21k
20k
19k


Mean Price by destination_city
Mean Price
20k
15k
10k
5k
0
Bangalore
Chennai
Delhi
Hyderabad
Kolkata
Mumbai
destination_city
Mean Price
21k
20k
19k


Mean Price by departure_time
Mean Price
20k
15k
10k
5k
0
Afternoon
Early_Morning
Evening
Late_Night
Morning
Night
departure_time
Mean Price
20k
15k
10k


Mean Price by arrival_time
Mean Price
20k
15k
10k
5k
0
Afternoon
Early_Morning
Evening
Late_Night
Morning
Night
arrival_time
Mean Price
20k
15k





As we can Business Class, flights with one stop and Vistara airline achieve higher mean prices. The flights with departure and arrival time “late night” have significant lower mean price over the other categories.

2) Modeling

We are going to proceed one_hot encoding for the columns: airline, source_city, destination_city and departure_time as they have a small number of non-numerical unique values .

Then we are going to turn the class column to binary as it contains ony two non numerical values and the the stops column values to numerical 0, 1 and 2.

Also we are going to drop the Unnamed: 0 and flight columns as the are not nessecary for our analysis and finaly we are going to let columns duration, days_left and price as they are.

[32]:
```
data = data.drop(["Unnamed: 0", "flight"], axis=1)
data["class"] = data["class"].apply(lambda x: 0 if x == "Economy" else 1)
data["stops"] = pd.factorize(data["stops"])[0]

display(data.head())
```

```
    airline source_city departure_time  stops   arrival_time destination_city  \
0  SpiceJet       Delhi        Evening      0          Night           Mumbai
1  SpiceJet       Delhi  Early_Morning      0        Morning           Mumbai
2   AirAsia       Delhi  Early_Morning      0  Early_Morning           Mumbai
3   Vistara       Delhi        Morning      0      Afternoon           Mumbai
4   Vistara       Delhi        Morning      0        Morning           Mumbai

   class  duration  days_left  price
0      0      2.17          1   5953
1      0      2.33          1   5953
2      0      2.17          1   5956
3      0      2.25          1   5955
4      0      2.33          1   5955
```

Preprocessing

We are going to proceed one_hot encoding for the columns: airline, source_city, destination_city and departure_time as they have a small number of non-numerical unique values .

Then we are going to turn the class column to binary as it contains ony two non numerical values and the the stops column values to numerical 0, 1 and 2.

Also we are going to drop the Unnamed: 0 and flight columns as the are not nessecary for our analysis and finaly we are going to let columns duration, days_left and price as they are.

[34]:
```
# Getting the dummies of source city column and joining them  to the dataframe
source_city_dummies = pd.get_dummies(data["source_city"], prefix="source_city")
data = data.join(source_city_dummies)

# Getting the dummies of destination_city column and joining them  to the␣
↪dataframe
destination_city_dummies = pd.get_dummies(data["destination_city"],␣
↪prefix="destination_city")
data = data.join(destination_city_dummies)

# Getting the dummies of airline column and joining them  to the dataframe
airline_dummies = pd.get_dummies(data["airline"], prefix="airline")
data = data.join(airline_dummies)

# Getting the dummies of arrival_time column and joining them  to the dataframe
arrival_time_dummies = pd.get_dummies(data["arrival_time"],␣
↪prefix="arrival_time")
data = data.join(arrival_time_dummies)
```

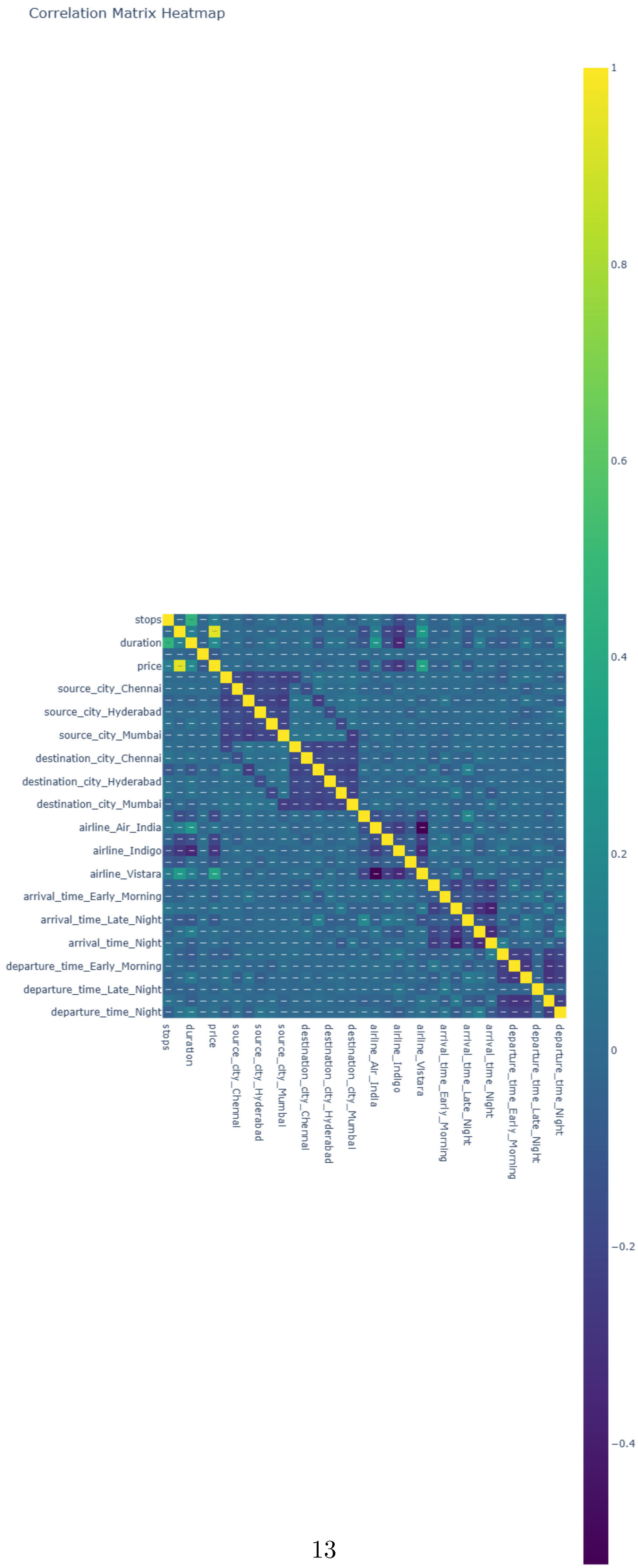
```
# Getting the dummies of departure_time column and joining them  to the␣
↪dataframe
departure_time_dummies = pd.get_dummies(data["departure_time"],␣
↪prefix="departure_time")
data = data.join(departure_time_dummies)

data = data.drop(["airline", "source_city", "destination_city",␣
↪"departure_time", "arrival_time"], axis=1)

display(data.head())
```

```
   stops  class  duration  days_left  price  source_city_Bangalore  \
0      0      0      2.17          1   5953                      0
1      0      0      2.33          1   5953                      0
2      0      0      2.17          1   5956                      0
3      0      0      2.25          1   5955                      0
4      0      0      2.33          1   5955                      0

   source_city_Chennai  source_city_Delhi  source_city_Hyderabad  \
0                    0                  1                      0
1                    0                  1                      0
2                    0                  1                      0
3                    0                  1                      0
4                    0                  1                      0

   source_city_Kolkata  …  arrival_time_Evening  arrival_time_Late_Night  \
0                    0  …                     0                        0
1                    0  …                     0                        0
2                    0  …                     0                        0
3                    0  …                     0                        0
4                    0  …                     0                        0

   arrival_time_Morning  arrival_time_Night  departure_time_Afternoon  \
0                     0                   1                         0
1                     1                   0                         0
2                     0                   0                         0
3                     0                   0                         0
4                     1                   0                         0

   departure_time_Early_Morning  departure_time_Evening  \
0                             0                       1
1                             1                       0
2                             1                       0
3                             0                       0
4                             0                       0
```

```
   departure_time_Late_Night  departure_time_Morning  departure_time_Night
0                          0                       0                     0
1                          0                       0                     0
2                          0                       0                     0
3                          0                       1                     0
4                          0                       1                     0

[5 rows x 35 columns]
```

Let s dive into the correlations between variables

[37]:
```
corr_matrix = data.corr()

# Creating the heatmap using Plotly Express
fig = px.imshow(
    corr_matrix,
    text_auto=True,
    color_continuous_scale='Viridis',
    title='Correlation Matrix Heatmap',
    width=2000,  # Adjust the width of the plot if needed
    height=2000
)

fig.show()
```


Correlation Matrix Heatmap
stops
duration
price
source_city_Chennai
source_city_Hyderabad
source_city_Mumbai
destination_city_Chennai
destination_city_Hyderabad
destination_city_Mumbai
airline_Air_India
airline_Indigo
airline_Vistara
arrival_time_Early_Morning
arrival_time_Late_Night
arrival_time_Night
departure_time_Early_Morning
departure_time_Late_Night
departure_time_Night
1
0.8
0.6
0.4
0.2
0
−0.2
−0.4

A) Positive Correlations:

The strongest positive correlation with price is class (0.938). This indicates that higher classes (likely higher fares) are positively correlated with price. duration (0.204) and stops (0.120) also show positive correlations with price, though weaker compared to class. arrival_time_Evening (0.056) and departure_time_Night (0.042) show some positive correlation with price as well.

B) Negative Correlations: The only notable negative correlation with price is days_left (-0.092), indicating that as the number of days left until the flight decreases, prices tend to increase slightly.

Regression Model

[39]:
```
# Excluding the target variable 'price' column
data_without_price = data.drop("price", axis=1)
X, y = data_without_price, data["price"]  # Features are everything except␣
↪'price', target is 'price'

# Initializing the RandomForestRegressor model
rf = RandomForestRegressor()

# Using K-Fold Cross Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Cross-validation predictions
y_pred = cross_val_predict(rf, X, y, cv=kf)

# Calculate metrics
r2 = r2_score(y, y_pred)
mae = mean_absolute_error(y, y_pred)
mse = mean_squared_error(y, y_pred)
rmse = math.sqrt(mse)

# Print metrics
print('R Squared:', r2)
print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('Root Mean Squared Error:', rmse)
```

```
R Squared: 0.9854754493812874
Mean Absolute Error: 1067.3186808936546
Mean Squared Error: 7482858.5990829235
Root Mean Squared Error: 2735.4814199849584
```

[41]:
```
# Creating a DataFrame for plotting
plot_data = pd.DataFrame({
    'Original Price': y,
```

```python
    'Predicted Price': y_pred
})

# Creating scatter plot
fig = px.scatter(
    plot_data,
    x='Original Price',
    y='Predicted Price',
    title='Predicted Prices vs Original Prices',
    labels={'Original Price': 'Original Price', 'Predicted Price': 'Predicted␣
↪Price'},
    width=800,
    height=600
)

fig.show()
```

A) Comparison between Mean Absolute Error (mae) and range Price (dependent Variable)

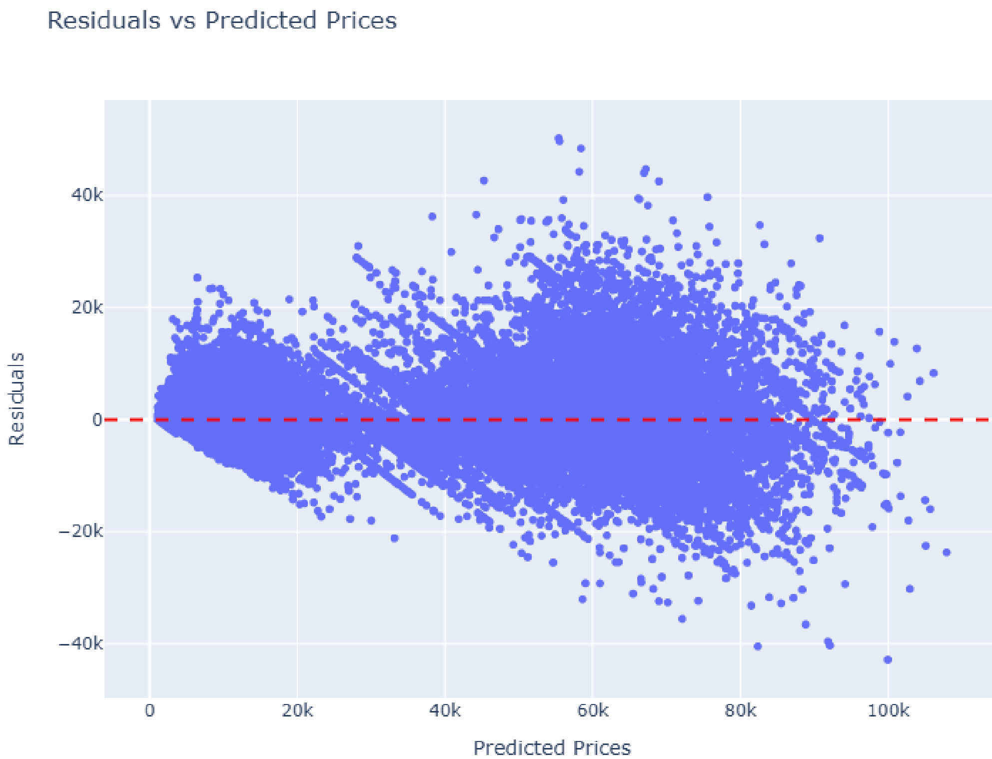
[44]:
```python
price_min = data["price"].min()
price_max = data["price"].max()
price_range = price_max - price_min
print(f'Price Range: {price_range}')

mae_percentage = (mae / price_range) * 100
print(f'mae as percentage of Range: {mae_percentage:.2f}%')
```

```
Price Range: 121966
mae as percentage of Range: 0.88%
```

Mean absolute error is 0.88% (LESS THAN 1))of the total range of prices, this iindicates very high model accuracy. This low percentage suggests that the average error is very small relative to the overall variability in our data and the model's predictions are very close to the actual values.

B) Residual Analysis

[47]:
```python
residuals = y - y_pred

# Creating a DataFrame for plotting
plot_data = pd.DataFrame({
    'Predicted Prices': y_pred,
    'Residuals': residuals
})

# Create scatter plot
fig = px.scatter(
    plot_data,
    x='Predicted Prices',
    y='Residuals',
    title='Residuals vs Predicted Prices',
    labels={'Predicted Prices': 'Predicted Prices', 'Residuals': 'Residuals'},
    width=800,
    height=600
)

# Adding horizontal line at y=0 (zero line)
fig.add_hline(y=0, line_dash="dash", line_color="red")

fig.show()
```



The residuals appear to be scattered somewhat randomly around the zero line, which is a good indication that the model has captured most of the underlying structure in the data.

There seems to be a slight funnel shape where the residuals become more spread out as the predicted prices increase. This suggests a possible issue with heteroscedasticity (i.e., the variance of the residuals increases with the predicted price).

C) Cross-Validation Consistency Check

[50]:
```
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(rf, X, y, cv=kf, scoring='neg_mean_absolute_error')
print(f'Cross-validated mae: {-cv_scores.mean()} (std: {cv_scores.std()})')
```

```
Cross-validated mae: 1066.2239258312663 (std: 7.938949695016515)
```

The Random Forest model has an estimated mean absolute error of approximately 1066.223, with a standard deviation of 7.938. This indicates that the model is performing consistently well across different folds, which is a positive sign for its generalization capability.

D) Acceptable range of mae

[53]:
```
avg_price = data["price"].mean()
print(f'Average Price: {avg_price}')
```

```
if mae < 0.1 * avg_price:  # Assuming 10% of average price as threshold
    print("mae is within acceptable range.")
else:
    print("mae is outside acceptable range.")
```

```
Average Price: 20889.660523133203
mae is within acceptable range.
```

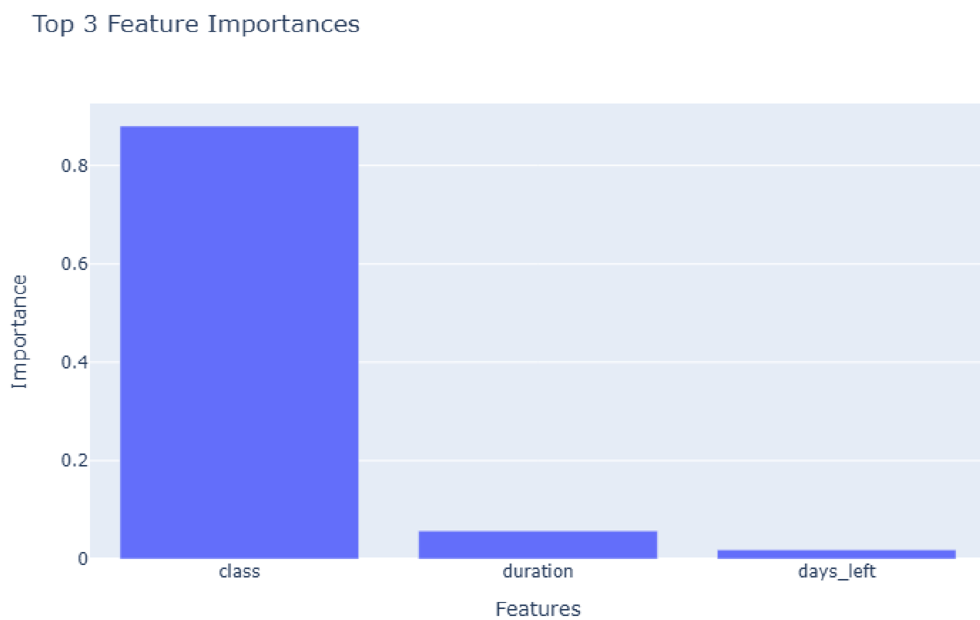
E) Feature Importance and Interpretation

[55]:
```
# Training the model on the entire dataset to get feature importances
rf.fit(X, y)

#Extracting feature names and their importances from the model
feature_names = rf.feature_names_in_ # Getting the names of the features used␣
 ↪by the model
feature_importances = rf.feature_importances_ # Getting the importance of each␣
 ↪feature

# Combining the feature names and their importances into a dictionary
importances =  dict(zip(feature_names, feature_importances))

#Sorting the features by their importances in descending order
sorted_importances = sorted(importances.items(), key=lambda x: x[1],␣
 ↪reverse=True) # importances.items() -list of tuples, x[1] for the second␣
 ↪ellement, reverse=True for descending order

#Printing the top three sorted list of features and their importances
top_three_importances = sorted_importances[:3]
display(top_three_importances)
```

```
[('class', 0.8801484748519044),
 ('duration', 0.05745470485847617),
 ('days_left', 0.018516566779171875)]
```

[57]:
```
# Creating a DataFrame for plotting
plot_data = pd.DataFrame({
    'Features': [x[0] for x in top_three_importances],
    'Importance': [x[1] for x in top_three_importances]
})

# Creating a bar plot
fig = px.bar(
    plot_data,
    x='Features',
    y='Importance',
    title='Top 3 Feature Importances',
    labels={'Features': 'Features', 'Importance': 'Importance'},
```

```
    width=700,
    height=500
)

fig.show()
```



The model relies heavily on ‘class’ feature to make its predictions so it plays a critical role in determining the outcome of the target variable.

Predicting price value given unprocessed data

[59]:
```
# Function to preprocess new data
def preprocess_new_data(new_data):
    # Apply the same transformations as the training data
    new_data["class"] = new_data["class"].apply(lambda x: 0 if x == "Economy"␣
 ↪else 1)
    new_data["stops"] = pd.factorize(new_data["stops"])[0]

    # One-hot encode categorical variables
    new_data = pd.get_dummies(new_data, columns=["airline", "source_city",␣
 ↪"destination_city", "departure_time", "arrival_time"])

    # Ensure the new data has the same columns as the training data
    missing_cols = set(X.columns) - set(new_data.columns)
    for col in missing_cols:
```

```
            new_data[col] = 0
    new_data = new_data[X.columns]

    return new_data

# Example new data for prediction
new_data = pd.DataFrame({
    "airline": ["IndiGo"],
    "source_city": ["Delhi"],
    "destination_city": ["Cochin"],
    "departure_time": ["Morning"],
    "arrival_time": ["Evening"],
    "stops": ["1 stop"],
    "class": ["Economy"],
    "duration": [180],
    "days_left": [30]
})

# Preprocess the new data
new_data_preprocessed = preprocess_new_data(new_data)

# Predict the price for the new data
predicted_price = rf.predict(new_data_preprocessed)

print("Predicted Price:", predicted_price)
```

```
Predicted Price: [6923.13]
```

[ ]:

```

```