In [ ]:
```
import pandas as pd
df=pd.read_csv("Clean_Dataset.csv");
```

In [ ]:
```
df.head()
```

Out[ ]:

| | Unnamed: 0 | airline | flight | source_city | departure_time | stops | arrival_time | destination_city | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | SpiceJet | SG-8709 | Delhi | Evening | zero | Night | Mumbai | Ecor |
| **1** | 1 | SpiceJet | SG-8157 | Delhi | Early_Morning | zero | Morning | Mumbai | Ecor |
| **2** | 2 | AirAsia | I5-764 | Delhi | Early_Morning | zero | Early_Morning | Mumbai | Ecor |
| **3** | 3 | Vistara | UK-995 | Delhi | Morning | zero | Afternoon | Mumbai | Ecor |
| **4** | 4 | Vistara | UK-963 | Delhi | Morning | zero | Morning | Mumbai | Ecor |

In [ ]:
```
df.describe
df
```

Out[ ]:

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

300153 rows × 12 columns

In [ ]:
```
desired_sample_size = 10000

# Randomly sample the dataset
sampled_df = df.sample(n=desired_sample_size, random_state=42)
```

In [ ]:
```
sampled_df.describe
sampled_df
```

Out[ ]:

| | Unnamed: 0 | airline | flight | source_city | departure_time | stops | arrival_time | destination_cit |
|---|---|---|---|---|---|---|---|---|
| **27131** | 27131 | Air_India | AI-506 | Delhi | Morning | one | Early_Morning | Kolkat |
| **266857** | 266857 | Vistara | UK-706 | Kolkata | Morning | one | Night | Mumba |
| **141228** | 141228 | Vistara | UK-772 | Kolkata | Morning | one | Night | Bangalor |
| **288329** | 288329 | Vistara | UK-824 | Chennai | Night | one | Morning | Dell |
| **97334** | 97334 | Air_India | AI-501 | Bangalore | Afternoon | one | Night | Mumba |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **27874** | 27874 | Air_India | AI-803 | Delhi | Early_Morning | one | Evening | Kolkat |
| **56233** | 56233 | GO_FIRST | G8-575 | Mumbai | Early_Morning | one | Afternoon | Bangalor |
| **98649** | 98649 | Vistara | UK-814 | Bangalore | Night | one | Early_Morning | Mumba |
| **250921** | 250921 | Vistara | UK-854 | Bangalore | Evening | zero | Night | Mumba |
| **244645** | 244645 | Air_India | AI-660 | Mumbai | Evening | one | Morning | Chenna |

10000 rows × 12 columns

In [ ]:
```
df.corr(numeric_only=True)
```

Out[ ]:

| | Unnamed: 0 | duration | days_left | price |
|---|---|---|---|---|
| **Unnamed: 0** | 1.000000 | 0.159007 | 0.014638 | 0.761177 |
| **duration** | 0.159007 | 1.000000 | -0.039157 | 0.204222 |
| **days_left** | 0.014638 | -0.039157 | 1.000000 | -0.091949 |
| **price** | 0.761177 | 0.204222 | -0.091949 | 1.000000 |

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Unnamed: 0          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
price               0
dtype: int64
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

# Fit and transform the 'departure_time' column
sampled_df['departure_time_encoded'] = le.fit_transform(sampled_df['departure_time'])

# Display the DataFrame with the new encoded column
print(sampled_df[['departure_time', 'departure_time_encoded']])
```

```
       departure_time  departure_time_encoded
27131         Morning                      4
266857        Morning                      4
141228        Morning                      4
288329          Night                      5
97334       Afternoon                      0
...               ...                    ...
27874   Early_Morning                      1
56233   Early_Morning                      1
98649           Night                      5
250921        Evening                      2
244645        Evening                      2

[10000 rows x 2 columns]
```

In [ ]:
```
le = LabelEncoder()

# Fit and transform the 'departure_time' column
sampled_df['arrival_time_encoded'] = le.fit_transform(sampled_df['arrival_time'])

# Display the DataFrame with the new encoded column
print(sampled_df[['arrival_time', 'arrival_time_encoded']])
```

```
         arrival_time  arrival_time_encoded
27131   Early_Morning                     1
266857          Night                     5
141228          Night                     5
288329        Morning                     4
97334           Night                     5
...               ...                   ...
27874         Evening                     2
56233       Afternoon                     0
98649   Early_Morning                     1
250921          Night                     5
244645        Morning                     4

[10000 rows x 2 columns]
```

In [ ]:
```
le = LabelEncoder()

# Fit and transform the 'departure_time' column
sampled_df['class_time_encoded'] = le.fit_transform(sampled_df['class'])

# Display the mapping between original values and encoded values
print("Mapping of departure_time values to numerical labels:")
for original, encoded in zip(le.classes_, le.transform(le.classes_)):
    print(f"{original}: {encoded}")

# Display the DataFrame with the new encoded column
print(sampled_df[['class', 'class_time_encoded']])
```

```
Mapping of departure_time values to numerical labels:
Business: 0
Economy: 1
           class  class_time_encoded
27131    Economy                   1
266857  Business                   0
141228   Economy                   1
288329  Business                   0
97334    Economy                   1
...          ...                 ...
27874    Economy                   1
56233    Economy                   1
98649    Economy                   1
250921  Business                   0
244645  Business                   0

[10000 rows x 2 columns]
```

VISUALIZATION

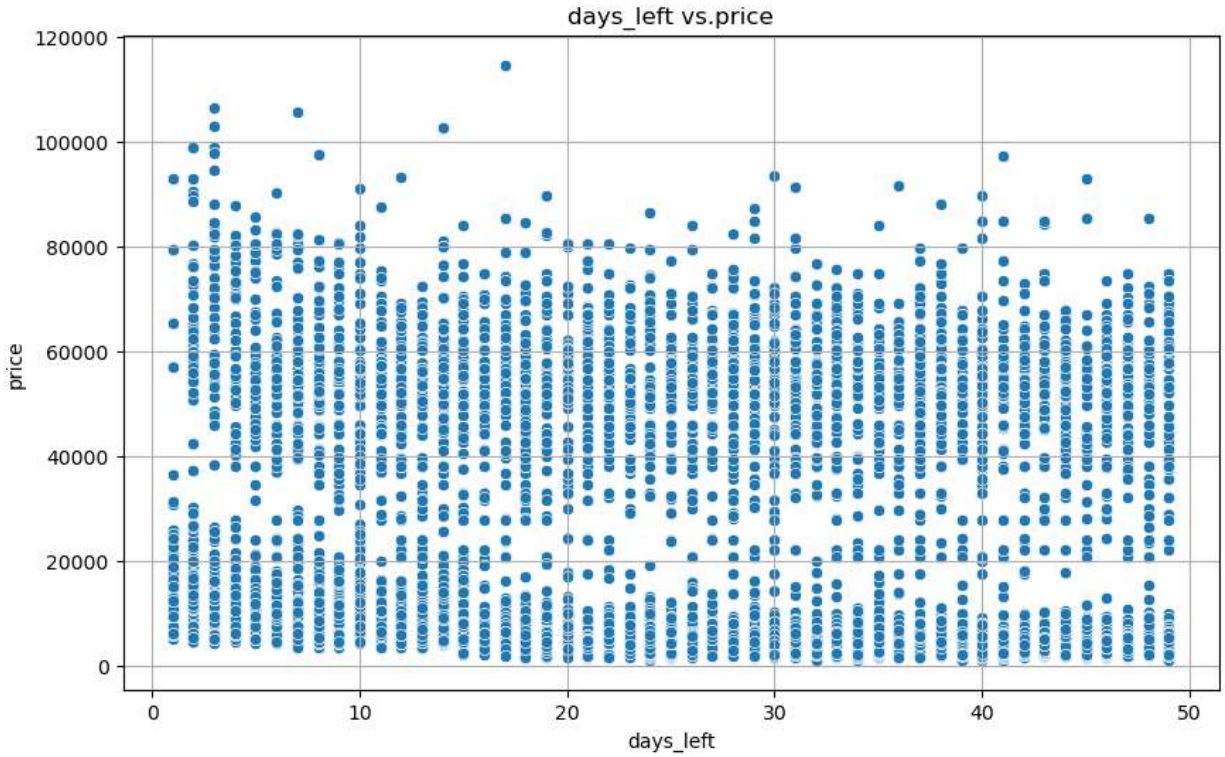
In [ ]:
```
sampled_df
```

Out[ ]:

| | Unnamed: 0 | airline | flight | source_city | departure_time | stops | arrival_time | destination_cit |
|---|---|---|---|---|---|---|---|---|
| **27131** | 27131 | Air_India | AI-506 | Delhi | Morning | one | Early_Morning | Kolkat |
| **266857** | 266857 | Vistara | UK-706 | Kolkata | Morning | one | Night | Mumba |
| **141228** | 141228 | Vistara | UK-772 | Kolkata | Morning | one | Night | Bangalor |
| **288329** | 288329 | Vistara | UK-824 | Chennai | Night | one | Morning | Dell |
| **97334** | 97334 | Air_India | AI-501 | Bangalore | Afternoon | one | Night | Mumba |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **27874** | 27874 | Air_India | AI-803 | Delhi | Early_Morning | one | Evening | Kolkat |
| **56233** | 56233 | GO_FIRST | G8-575 | Mumbai | Early_Morning | one | Afternoon | Bangalor |
| **98649** | 98649 | Vistara | UK-814 | Bangalore | Night | one | Early_Morning | Mumba |
| **250921** | 250921 | Vistara | UK-854 | Bangalore | Evening | zero | Night | Mumba |
| **244645** | 244645 | Air_India | AI-660 | Mumbai | Evening | one | Morning | Chenna |

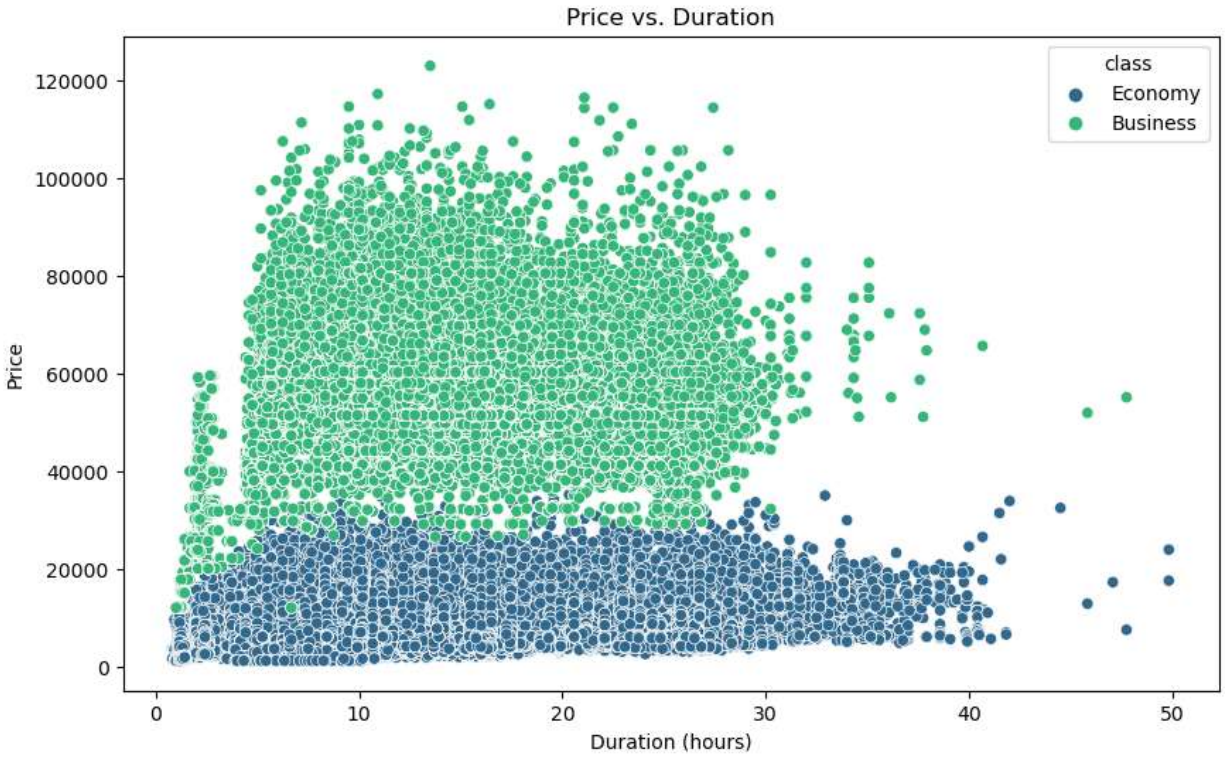
10000 rows × 15 columns

In [ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.scatterplot(x='days_left', y='price', data=sampled_df)
plt.title('days_left vs.price')
plt.xlabel('days_left')
plt.ylabel('price')
plt.grid(True)
plt.show()
```




Scatter Plot for Price vs. Duration:

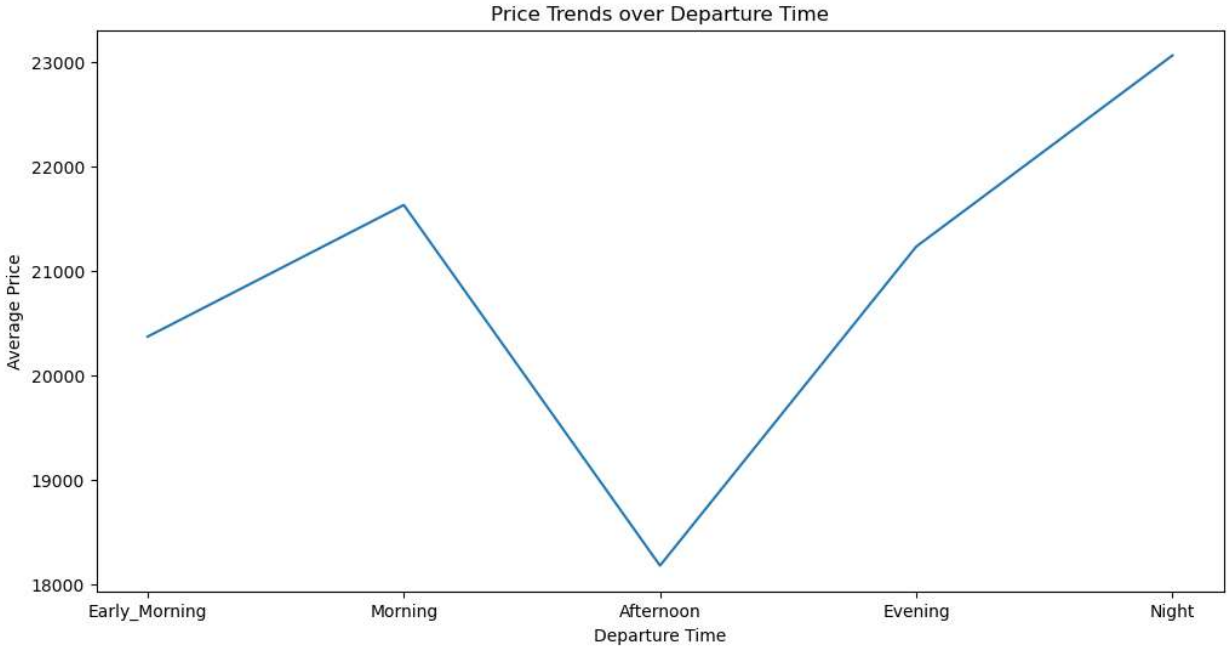
In [ ]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='duration', y='price', data=df, hue='class', palette='viridis')
plt.title('Price vs. Duration')
plt.xlabel('Duration (hours)')
plt.ylabel('Price')
plt.show()
```



Line Chart for Price Trends over Time:

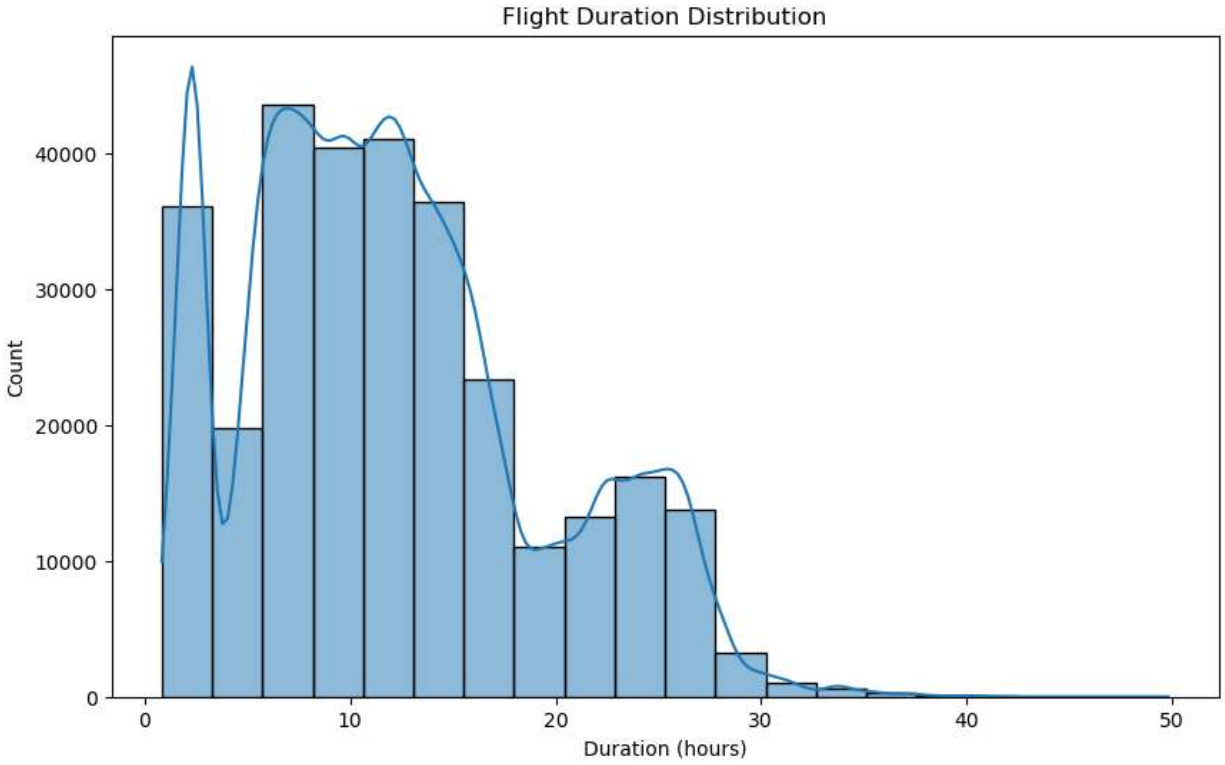
In [ ]:
```
plt.figure(figsize=(12, 6))
df['departure_time'] = pd.Categorical(df['departure_time'], categories=['Early_Morning
sns.lineplot(x='departure_time', y='price', data=df, estimator='mean', ci=None)
plt.title('Price Trends over Departure Time')
plt.xlabel('Departure Time')
plt.ylabel('Average Price')
plt.show()
```

```
C:\Users\pradh\AppData\Local\Temp\ipykernel_52028\823941320.py:3: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(x='departure_time', y='price', data=df, estimator='mean', ci=None)
```



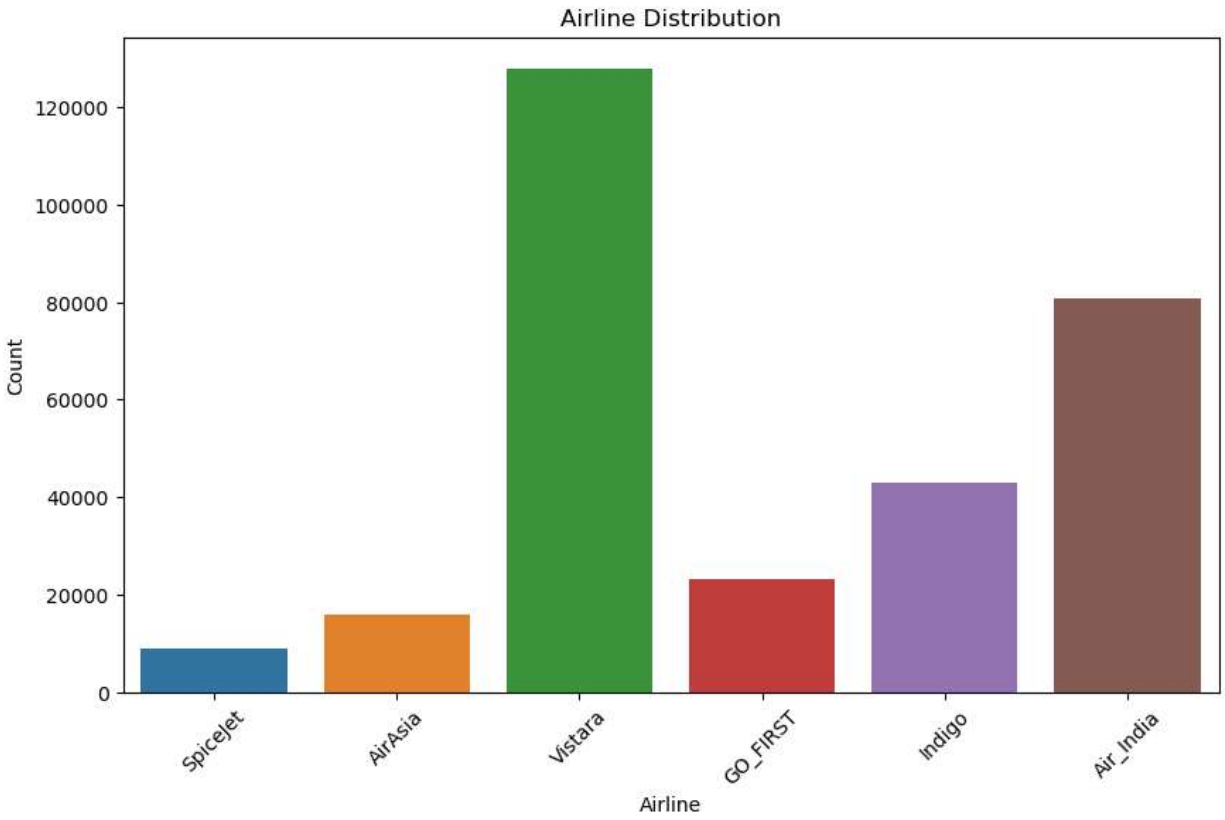

Histogram for Flight Durations:

In [ ]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df['duration'], bins=20, kde=True)
plt.title('Flight Duration Distribution')
plt.xlabel('Duration (hours)')
plt.ylabel('Count')
plt.show()
```

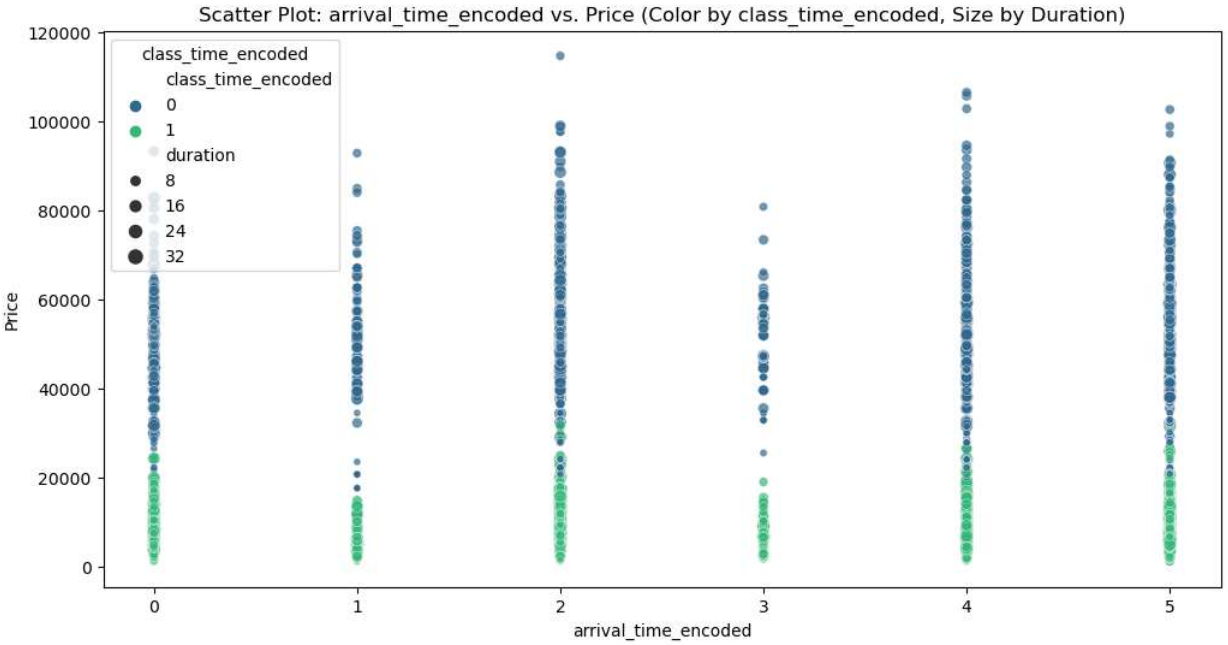



Bar Chart for Airline Distribution:

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(x='airline', data=df)
plt.title('Airline Distribution')
plt.xlabel('Airline')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.show()
```

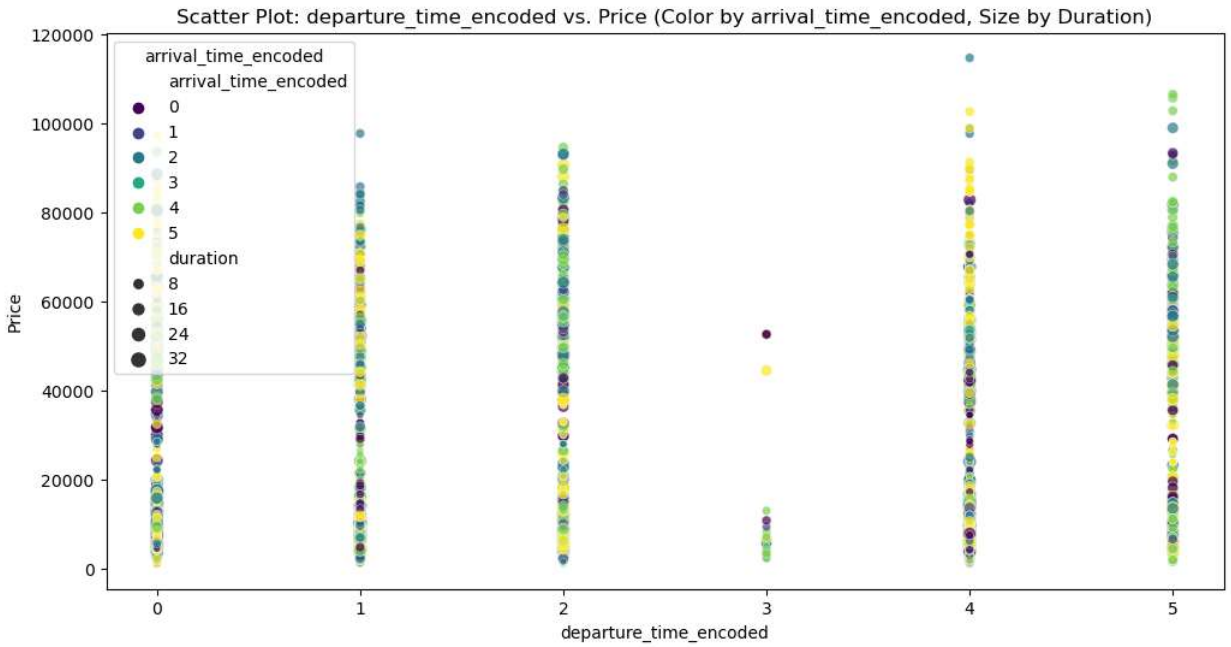



In [ ]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='arrival_time_encoded', y='price', hue='class_time_encoded', size='d
plt.title('Scatter Plot: arrival_time_encoded vs. Price (Color by class_time_encoded,
plt.xlabel('arrival_time_encoded')
plt.ylabel('Price')
plt.legend(title='class_time_encoded')
plt.show()
```




In [ ]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='departure_time_encoded', y='price', hue='arrival_time_encoded', siz
plt.title('Scatter Plot: departure_time_encoded vs. Price (Color by arrival_time_encod
plt.xlabel('departure_time_encoded')
```

```
plt.ylabel('Price')
plt.legend(title='arrival_time_encoded')
plt.show()
```




In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
```

K-Nearnest Neighbors (KNN)

In [ ]:
```
# Assuming you have your features (X) and target variable (y) defined as follows
X = sampled_df[['duration', 'days_left','departure_time_encoded','class_time_encoded',
y = sampled_df['price']

# Split the data into a training set and a testing set (e.g., 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Create a K-Nearest Neighbors (KNN) model
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can adjust the number of n

# Train the KNN model on the training data
knn_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn_classifier.predict(X_test)

# Calculate and print evaluation metrics
accuracy_k = accuracy_score(y_test, y_pred)
precision_k = precision_score(y_test, y_pred, average='weighted')  # Use 'micro', 'mac
f1_k = f1_score(y_test, y_pred, average='weighted')  # Use 'micro', 'macro', or 'weigh
recall_k = recall_score(y_test, y_pred, average='weighted')  # Use 'micro', 'macro', c

print(f'Accuracy: {accuracy_k}')
print(f'Precision: {precision_k}')
print(f'F1 Score: {f1_k}')
print(f'Recall: {recall_k}')
```

```
Accuracy: 0.019
Precision: 0.014541405081846258
F1 Score: 0.01482783644460115
Recall: 0.019
```

Decision tree

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
```

In [ ]:
```
# Create a Decision Tree classifier
decision_tree = DecisionTreeClassifier(random_state=42)

# Fit the model to the training data
decision_tree.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = decision_tree.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Calculate precision
precision = precision_score(y_test, y_pred, average='weighted')  # You can use 'micro'

# Calculate F1 score
f1 = f1_score(y_test, y_pred, average='weighted')  # You can use 'micro', 'macro', or

# Calculate recall
recall = recall_score(y_test, y_pred, average='weighted')  # You can use 'micro', 'mac

# Print the metrics
print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'F1 Score: {f1}')
print(f'Recall: {recall}')
```

```
Accuracy: 0.116
Precision: 0.12751229187479185
F1 Score: 0.11127835335160347
Recall: 0.116
```

Support Vector Machine (SVM)

In [ ]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score
from sklearn.model_selection import train_test_split

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Create an SVM model
svm_classifier = SVC(random_state=42)

# Fit the model to the training data
svm_classifier.fit(X_train, y_train)

# Make predictions on the testing data
y_pred_svm = svm_classifier.predict(X_test)
```

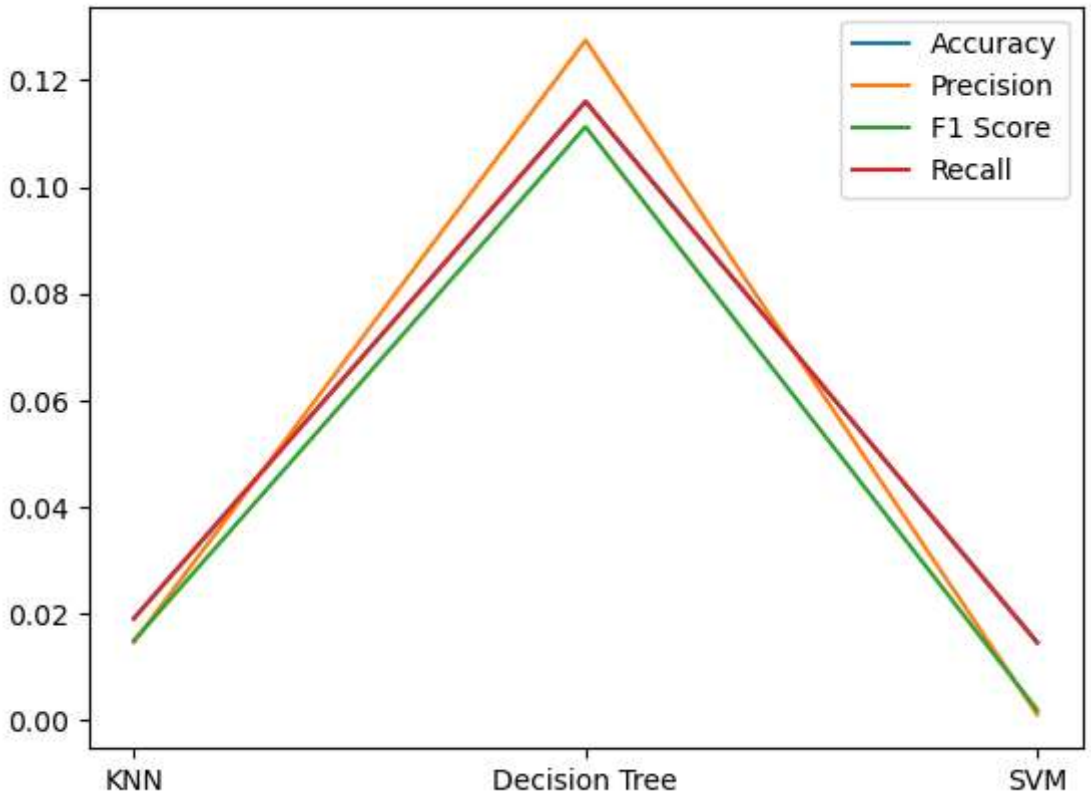
```
# Calculate and print the evaluation metrics
accuracy_s = accuracy_score(y_test, y_pred_svm)
precision_s = precision_score(y_test, y_pred_svm, average='weighted')  # Use 'micro',
f1_s = f1_score(y_test, y_pred_svm, average='weighted')  # Use 'micro', 'macro', or 'w
recall_s = recall_score(y_test, y_pred_svm, average='weighted')  # Use 'micro', 'macro

print(f'Accuracy: {accuracy_s}')
print(f'Precision: {precision_s}')
print(f'F1 Score: {f1_s}')
print(f'Recall: {recall_s}')
```

```
Accuracy: 0.0145
Precision: 0.0009855002984213602
F1 Score: 0.0018170765599032665
Recall: 0.0145
```

In [ ]:
```
models = [ 'KNN', 'Decision Tree','SVM']
acc_scores = [accuracy_k, accuracy,accuracy_s ]
prec_scores = [precision_k,precision,precision_s ]
f1_scores = [f1_k,f1,f1_s]
recall_scores = [recall_k, recall,recall_s]

best_model = acc_scores.index(max(acc_scores))
print(f'Best model based on accuracy: {models[best_model]}')

best_model = prec_scores.index(max(prec_scores))
print(f'Best model based on precision: {models[best_model]}')

best_model = f1_scores.index(max(f1_scores))
print(f'Best model based on f1 score: {models[best_model]}')

best_model = recall_scores.index(max(recall_scores))
print(f'Best model based on recall: {models[best_model]}')
```

```
Best model based on accuracy: Decision Tree
Best model based on precision: Decision Tree
Best model based on f1 score: Decision Tree
Best model based on recall: Decision Tree
```

In [ ]:
```
plt.plot(models, acc_scores, label='Accuracy')
plt.plot(models, prec_scores, label='Precision')
plt.plot(models, f1_scores, label='F1 Score')
plt.plot(models, recall_scores, label='Recall')
plt.legend()
plt.show()
```




In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Perform feature scaling (Standardization) if needed
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [ ]:
```
from sklearn.linear_model import LinearRegression

# Initialize the regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
# Prompt the user for input
duration = float(input("Enter the duration (in hours): "))
days_left = int(input("Enter the number of days left for the flight: "))
departure_time_encoded=int(input("Enter the departure time  for the flight:afternoon=0
class_time_encoded = int(input("Enter the Class type for the flight: 1is for Economy a
arrival_time_encoded=int(input("Enter the arrival time for the flight: fternoon=0After

# Create a user input dictionary with the values provided by the user
user_input = {
```

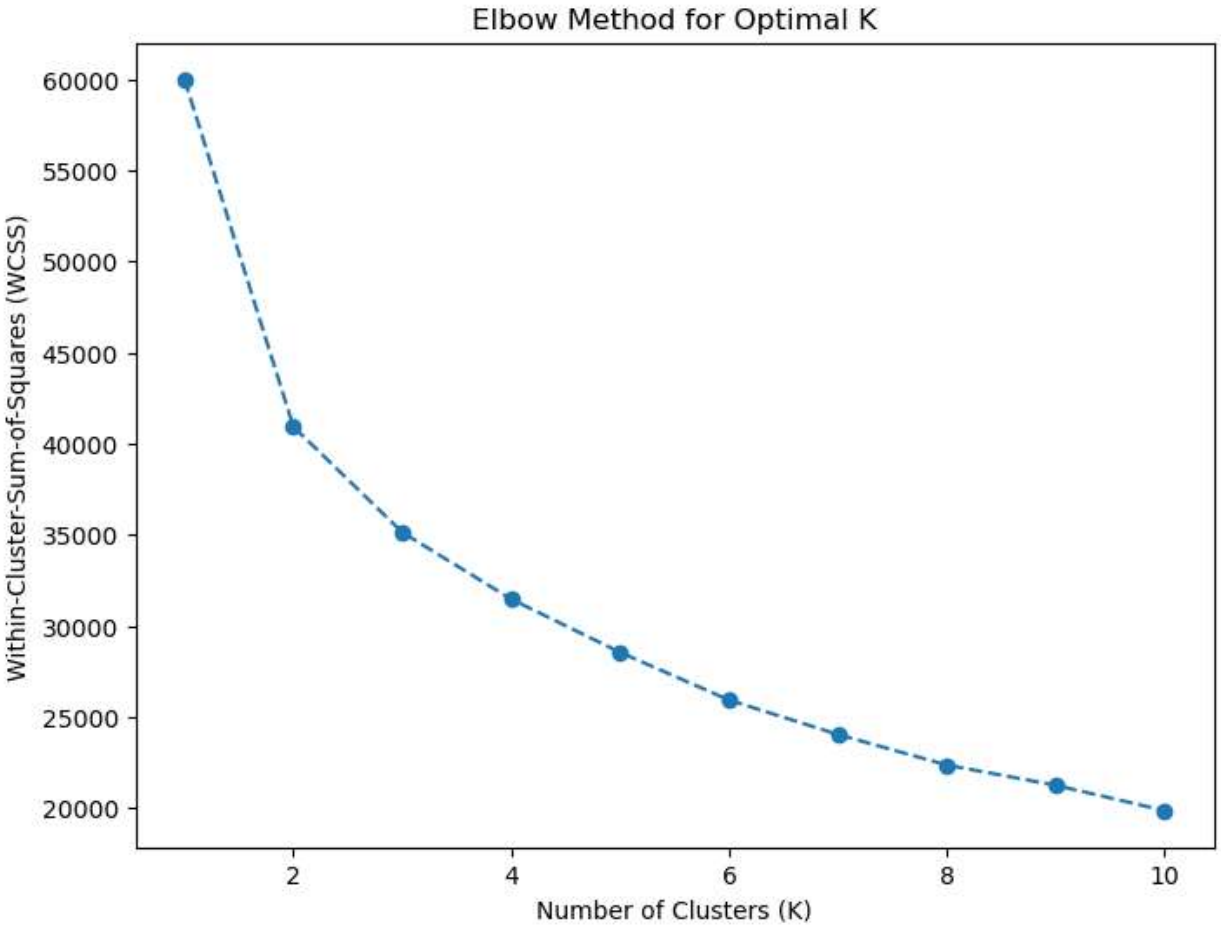
```
    'duration': duration,
    'days_left': days_left,
    'departure_time_encoded':departure_time_encoded,
    'class_time_encoded':class_time_encoded,
    'arrival_time_encoded':arrival_time_encoded
}

# Create a DataFrame from the user input
user_df = pd.DataFrame(user_input, index=[0])

# Perform feature scaling on the user input data
#The user input data is standardized using the same scaler that was used to scale the
user_df = scaler.transform(user_df)

# Make price predictions
predicted_prices = model.predict(user_df)
print(f"Predicted Price: {predicted_prices[0]}")
```

```
Predicted Price: 7466.098959281715
```

## K-MEANS CLUSTERING

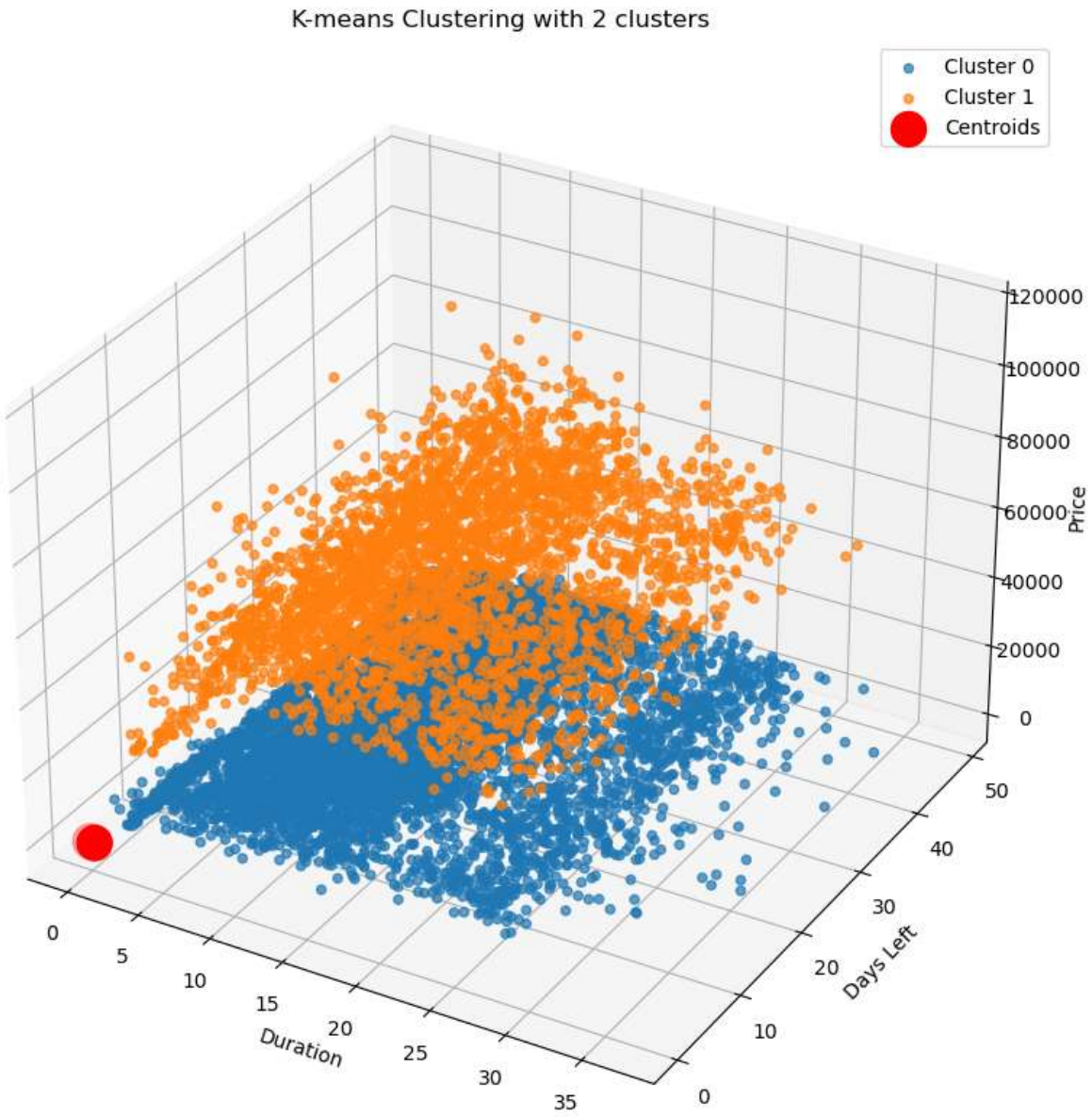
In [ ]:

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
features_for_clustering = ['duration', 'days_left','departure_time_encoded','class_tim
scaler = StandardScaler()
df_scaled = scaler.fit_transform(sampled_df[features_for_clustering])

# Determine the optimal number of clusters using the Elbow Method
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_st
    kmeans.fit(df_scaled)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Within-Cluster-Sum-of-Squares (WCSS)')
plt.show()
```




In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D  # Importing 3D plotting library

# Assuming sampled_df is your DataFrame
optimal_k = 2  # Choose the optimal K value based on the Elbow Method

# Select the features for clustering
features_for_clustering = ['duration', 'days_left', 'departure_time_encoded', 'class_t

# Standardize the features
scaler = StandardScaler()
df_scaled = scaler.fit_transform(sampled_df[features_for_clustering])

# Apply K-means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300, n_init=10, randc
sampled_df['cluster'] = kmeans.fit_predict(df_scaled)

# Visualize the clusters in 3D space
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

for i in range(optimal_k):
    cluster_data = sampled_df[sampled_df['cluster'] == i]
    ax.scatter(cluster_data['duration'], cluster_data['days_left'], cluster_data['pric

ax.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], kmeans.cluste
           s=300, c='red', label='Centroids')
ax.set_xlabel('Duration')
ax.set_ylabel('Days Left')
ax.set_zlabel('Price')
ax.set_title(f'K-means Clustering with {optimal_k} clusters')
```

```
plt.legend()
plt.show()
```




In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Assuming 'price' is your target variable
X = sampled_df[['duration', 'days_left', 'departure_time_encoded', 'class_time_encoded
y = sampled_df['price']

# Split the data into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"R-squared (R2): {r2}")
```

```
Mean Squared Error (MSE): 58337125.78427896
R-squared (R2): 0.891465119334148
```