Dynamic Airfare Prediction: A Machine Learning Approach to Enhancing Travel Cost Management

Project Report

by

Naresh Gajula
Nishanta Bhanu Palla

973-652-1701
857-397-7871

gajula.na@northeastern.edu
palla.n@northeastern.edu

Percentage of Effort Contributed by Student 1: 50%

Percentage of Effort Contributed by Student 2: 50%

Signature of Student 1: Naresh Gajula

Signature of Student 2: Nishant Bhanu Palla

Submission Date: 04-19-2024

Predicting Flight Fares by ML

Background:

The airline industry stands as one of the most critical sectors in global transportation, facilitating the movement of people and goods across vast distances. With millions of flights operated daily, the pricing dynamics within this industry are complex, influenced by a multitude of factors such as fuel prices, demand patterns, competition, and operational costs. In such a dynamic environment, accurate forecasting of flight fares becomes imperative for both airlines and travelers. Predictive models offer a promising solution to this challenge, leveraging historical data to anticipate future fare trends and optimize pricing strategies. By developing a robust machine learning model for predicting flight fares, we aim to contribute to the efficiency and competitiveness of the airline industry while empowering travelers to make more informed booking decisions.

Problem Description:

At the heart of our project lies the fundamental challenge of predicting flight fares with precision and reliability. The objectives are twofold: first, to build a predictive model capable of forecasting future fares with a high degree of accuracy, and second, to gain insights into the underlying factors driving fare fluctuations. By addressing these objectives, we aim to provide airlines with actionable intelligence to optimize revenue management strategies and empower travelers with the knowledge to secure the best deals. Our analysis encompasses a comprehensive exploration of historical flight fare data, encompassing diverse routes, airlines, and travel periods. Through advanced analytics techniques, we seek to unravel the complexities of fare dynamics and offer practical solutions to enhance pricing transparency and efficiency in the air travel market.

Analysis:

Our analysis unfolds in multiple stages, beginning with exploratory data analysis (EDA) to understand the distribution, trends, and relationships within the flight fare dataset. We delve deep into the various attributes such as departure/arrival times, airline preferences, route popularity, and seasonal variations to uncover patterns and insights. Feature engineering plays a crucial role in transforming raw data into meaningful predictors, enhancing the predictive power of our model. Leveraging state-of-the-art machine learning algorithms, we embark on model development, experimenting with regression, ensemble methods, and other advanced techniques to capture the complex interplay of factors influencing fare dynamics. Rigorous evaluation and validation ensure the robustness and reliability of our predictive model, paving the way for actionable insights and recommendations.

Results:

The culmination of our analysis yields promising results, demonstrating the efficacy of our predictive model in forecasting flight fares with remarkable accuracy. Through meticulous feature selection and model refinement, we achieve significant improvements in predictive performance, surpassing traditional forecasting methods. Our findings shed light on the key drivers of fare fluctuations, including seasonal demand variations, competitive pricing strategies, and external factors such as fuel costs and economic conditions. Furthermore, we provide actionable recommendations for airlines to optimize pricing strategies and for travelers to capitalize on favorable booking opportunities. By leveraging the power of data-driven insights, we empower stakeholders across the air travel ecosystem to navigate the complexities of fare dynamics with confidence and efficiency.

Conclusion:

In conclusion, our project underscores the transformative potential of predictive analytics in shaping the future of air travel. By harnessing the vast reservoir of historical flight fare data, we unlock actionable insights that drive value for airlines, travelers, and industry stakeholders alike. The development of a robust predictive model represents a significant step towards enhancing pricing transparency, optimizing revenue management, and improving the overall travel experience. Moving forward, continued investment in data analytics and machine learning holds the key to unlocking new frontiers in airfare prediction, ultimately fostering a more efficient, competitive, and consumer-friendly aviation industry.

In [2]:
```python
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Data Analysis

In [3]:
```python
data = pd.read_csv("N:\Data Mining\Air fare\Cleaned_dataset.csv", encoding='ISO-8859-1')
```

In [4]:
```python
data.head()
```

Out[4]:

	Date_of_journey	Journey_day	Airline	Flight_code	Class	Source	Departure	Total_stops	Arrival	Destination	Duration_in_hours	Days_left	Fare
0	2023-01-16	Monday	SpiceJet	SG-8169	Economy	Delhi	After 6 PM	non-stop	After 6 PM	Mumbai	2.0833	1	5335
1	2023-01-16	Monday	Indigo	6E-2519	Economy	Delhi	After 6 PM	non-stop	Before 6 AM	Mumbai	2.3333	1	5899
2	2023-01-16	Monday	GO FIRST	G8-354	Economy	Delhi	After 6 PM	non-stop	Before 6 AM	Mumbai	2.1667	1	5801
3	2023-01-16	Monday	SpiceJet	SG-8709	Economy	Delhi	After 6 PM	non-stop	After 6 PM	Mumbai	2.0833	1	5794
4	2023-01-16	Monday	Air India	AI-805	Economy	Delhi	After 6 PM	non-stop	After 6 PM	Mumbai	2.1667	1	5955

Checking for missing values

In [5]:
```python
data.isnull().sum()
```

Out[5]:
```
Date_of_journey      0
Journey_day          0
Airline              0
Flight_code          0
Class                0
Source               0
Departure            0
Total_stops          0
Arrival              0
Destination          0
Duration_in_hours    0
Days_left            0
Fare                 0
dtype: int64
```

In [6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452088 entries, 0 to 452087
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Date_of_journey    452088 non-null  object 
 1   Journey_day        452088 non-null  object 
 2   Airline            452088 non-null  object 
 3   Flight_code        452088 non-null  object 
 4   Class              452088 non-null  object 
 5   Source             452088 non-null  object 
 6   Departure          452088 non-null  object 
 7   Total_stops        452088 non-null  object 
 8   Arrival            452088 non-null  object 
 9   Destination        452088 non-null  object 
 10  Duration_in_hours  452088 non-null  float64
 11  Days_left          452088 non-null  int64  
 12  Fare               452088 non-null  int64  
dtypes: float64(1), int64(2), object(10)
memory usage: 44.8+ MB
```

In [7]:
```
data.describe()
```

Out[7]:

	Duration_in_hours	Days_left	Fare
count	452088.000000	452088.000000	452088.000000
mean	12.349222	25.627902	22840.100890
std	7.431478	14.300846	20307.963002
min	0.750000	1.000000	1307.000000
25%	6.583300	13.000000	8762.750000
50%	11.333300	26.000000	13407.000000
75%	16.500000	38.000000	35587.000000
max	43.583300	50.000000	143019.000000

correlation matrix

In [8]:

```
import pandas as pd
import plotly.express as px


# Create the correlation matrix
corr_matrix = data.corr()
corr_matrix
```

```
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\3396527796.py:6: FutureWarning: The default value of numeric_only in DataFrame.corr is de
precated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this war
ning.
  corr_matrix = data.corr()
```

Out[8]:

	Duration_in_hours	Days_left	Fare
Duration_in_hours	1.000000	-0.032878	0.179909
Days_left	-0.032878	1.000000	-0.087852
Fare	0.179909	-0.087852	1.000000

In [9]:

```
# Create the heatmap
fig = px.imshow(corr_matrix)
fig.show()
```

In [9]:
```
data.shape
```

Out[9]:
```
(452088, 13)
```

Remove duplicates

In [10]:
```
data = data.dropna()
data.drop_duplicates( keep=False, inplace=True)
data = data.reset_index(drop = True)
data.shape
```

Out[10]: (440087, 13)

In [11]:
```
data1 = data.groupby(['Airline', 'Flight_code'], as_index=False).count()
counts = data1.Airline.value_counts().reset_index()
counts.columns = ['Airline', 'Count']
counts
```

Out[11]:

	Airline	Count
0	Indigo	702
1	Air India	171
2	Vistara	165
3	AirAsia	106
4	GO FIRST	104
5	SpiceJet	92
6	AkasaAir	51
7	AllianceAir	10
8	StarAir	4

In [12]:
```
data2 = data.groupby(['Flight_code','Airline','Class'],as_index=False).count()
class_counts = data2['Class'].value_counts()
class_counts_df = pd.DataFrame({'Class': class_counts.index, 'Count': class_counts.values})
class_counts_df
```

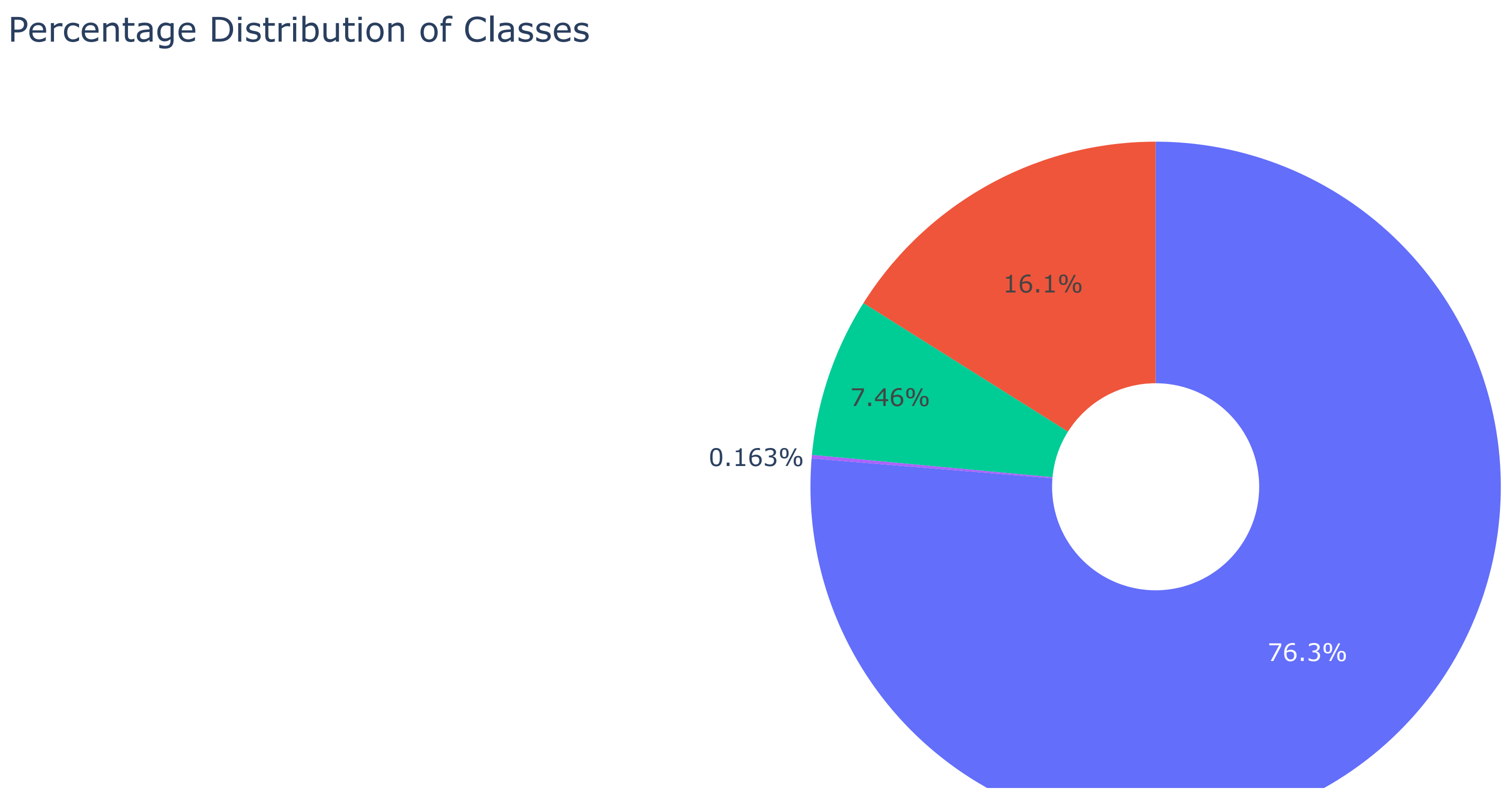
Out[12]:

	Class	Count
0	Economy	1401
1	Business	295
2	Premium Economy	137
3	First	3

Visualizations:

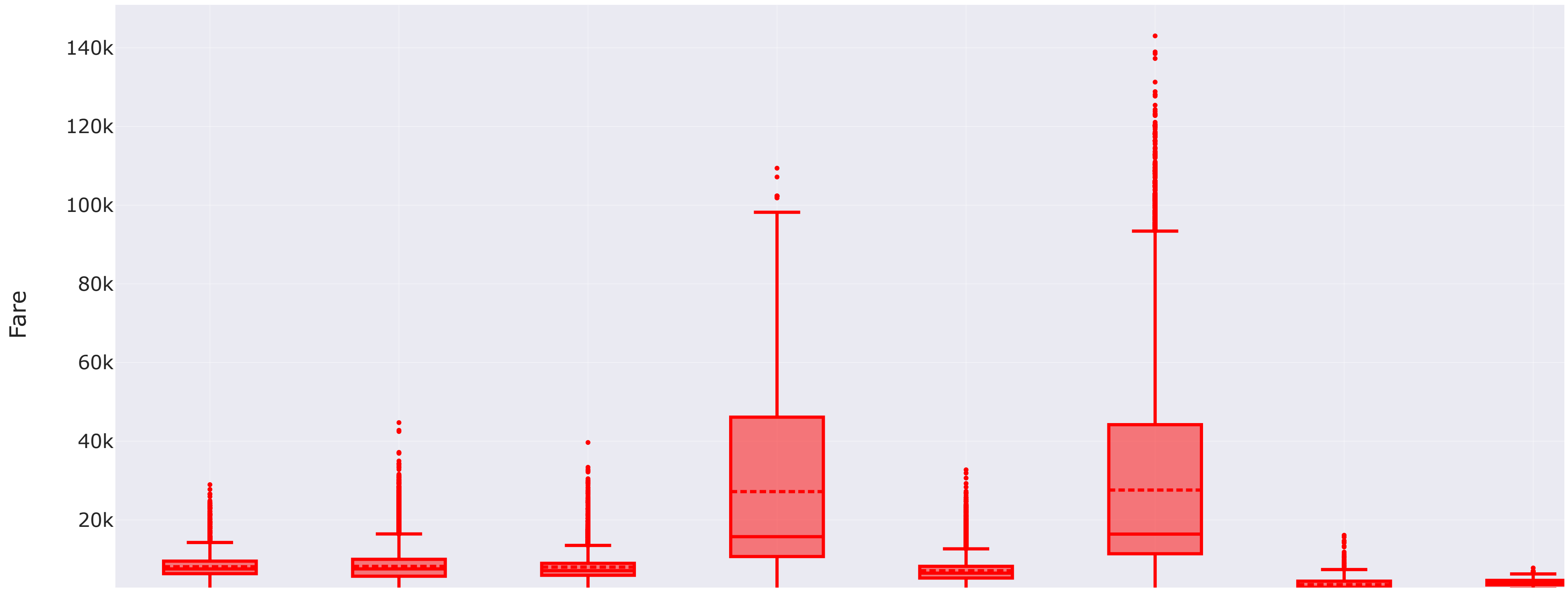
percentage distribution of classes by pie chart

In [14]:
```
import plotly.graph_objects as go

fig = go.Figure(data=[go.Pie(labels=class_counts_df['Class'],
                             values=class_counts_df['Count'],
                             hole=.3)])

fig.update_layout(title='Percentage Distribution of Classes',
                  font=dict(size=12),
                  legend=dict(orientation="h"))

fig.show()
```



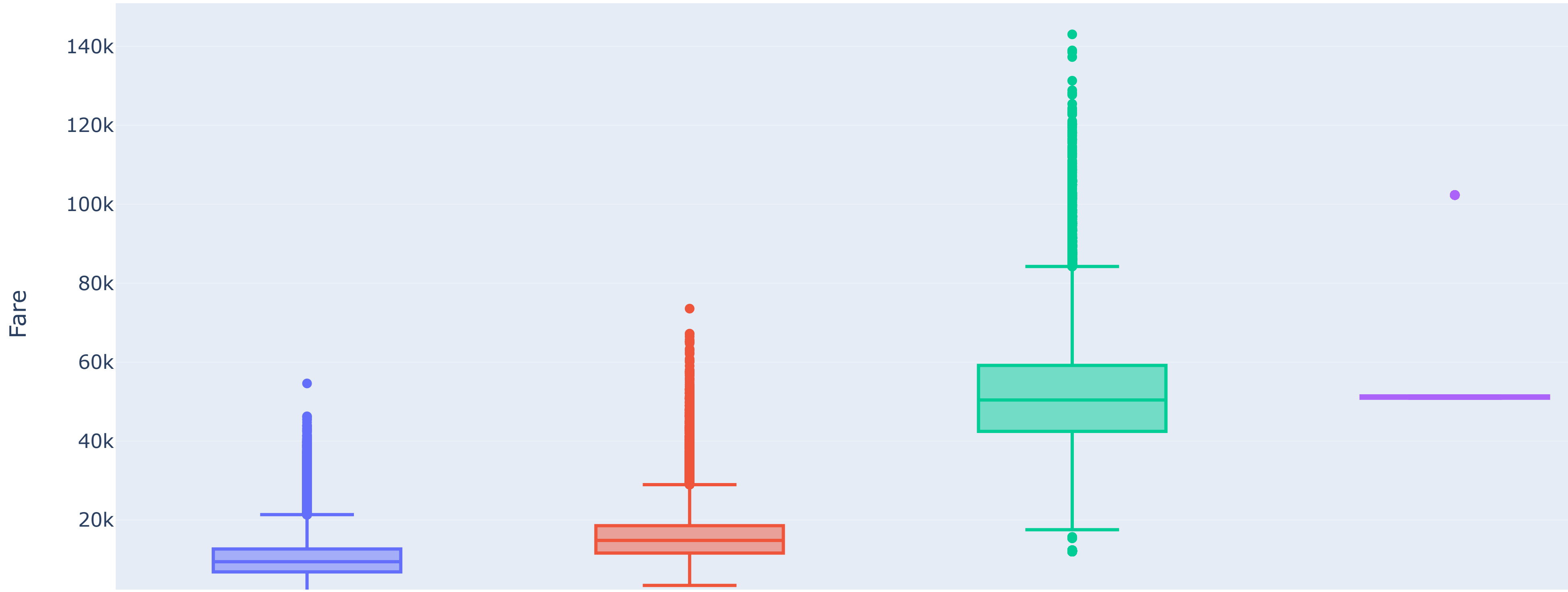
Boxplot of ticket prices by each airline

In [15]:
```
import plotly.express as px

fig = px.box(data, y='Fare', x='Airline', color_discrete_sequence=["orange"], points='all', template='seaborn')
fig.update_traces(marker=dict(color='red', size=3), boxmean=True, boxpoints='outliers')
fig.show()
```



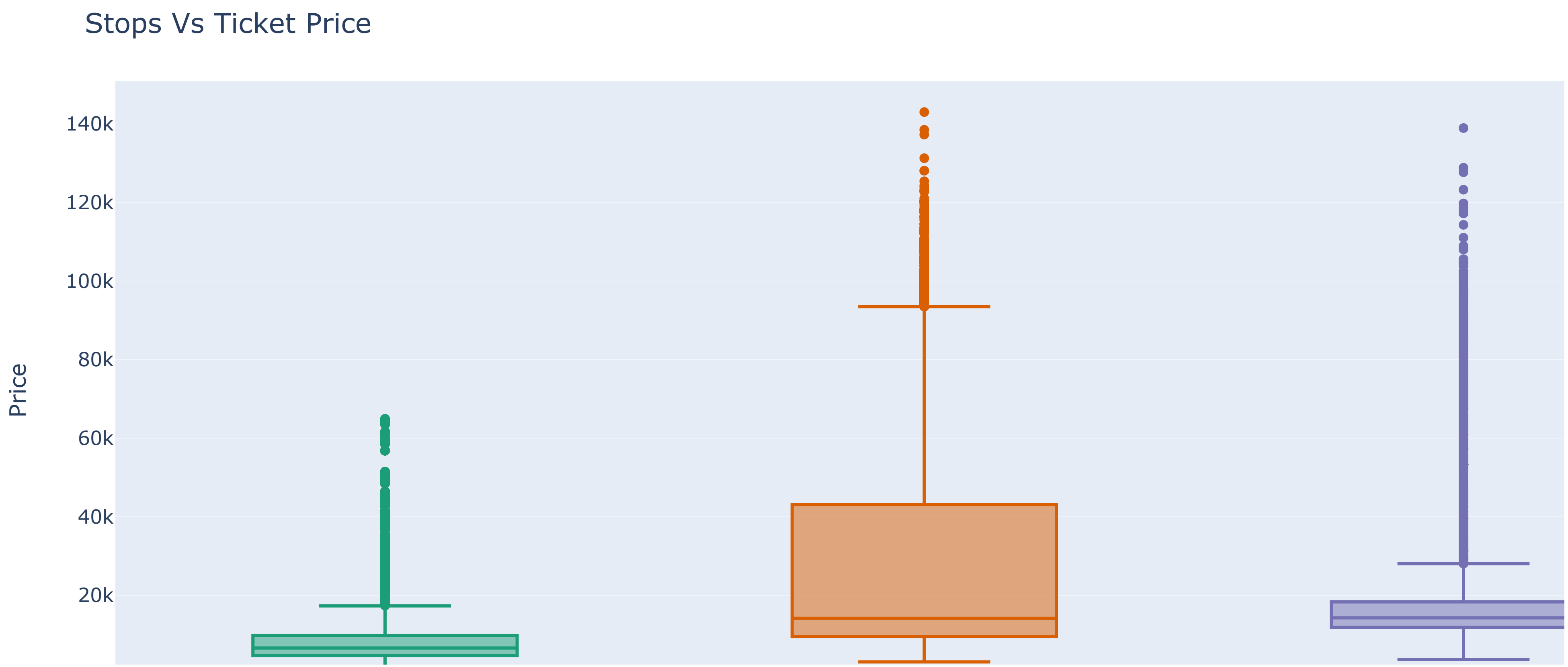
Boxplot of ticket prices by each Class

In [16]:
```
fig = px.box(data, x="Class", y="Fare", color="Class", hover_data=["Airline"])
fig.update_traces(quartilemethod="exclusive")
fig.show()
```



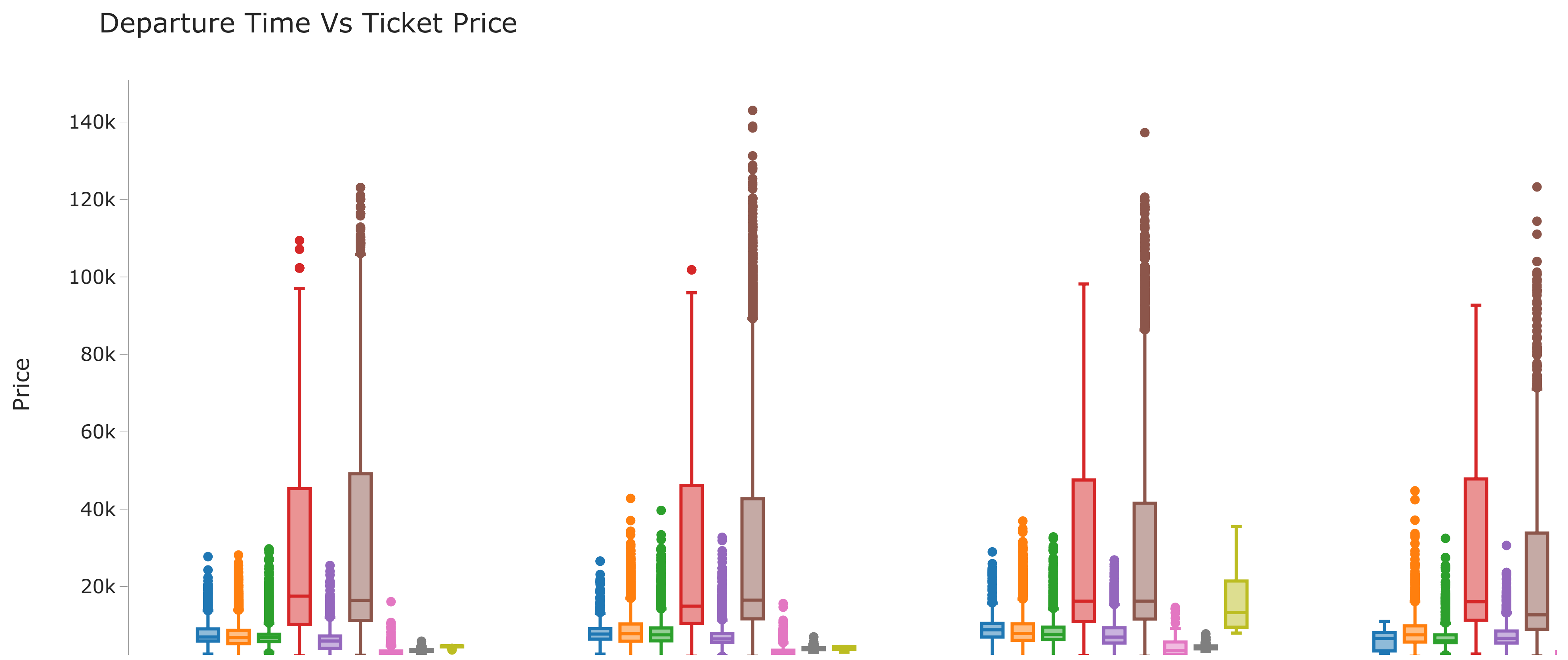
Boxplot of ticket prices by number of stops

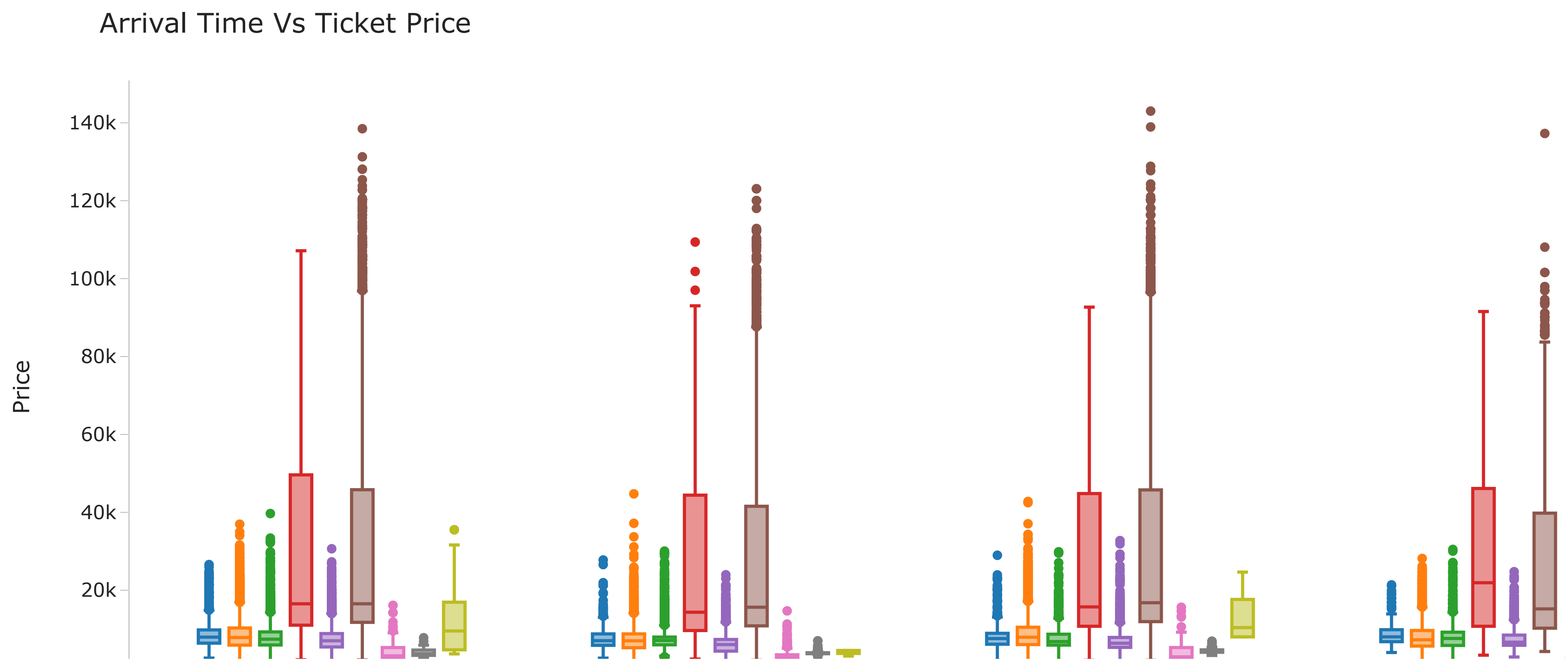
In [17]:
```
import plotly.express as px

fig = px.box(data, x='Total_stops', y='Fare', color='Total_stops', color_discrete_sequence=px.colors.qualitative.Dark2)
fig.update_layout(title='Stops Vs Ticket Price', xaxis_title='Stops', yaxis_title='Price')
fig.show()
```



Boxplot of ticket prices by departure time

In [19]:
```
fig = px.box(data, x='Departure', y='Fare', color='Airline', template='simple_white')
fig.update_layout(title='Departure Time Vs Ticket Price', xaxis_title='Departure Time', yaxis_title='Price')
fig.show()

fig = px.box(data, x='Arrival', y='Fare', color='Airline', template='simple_white')
fig.update_layout(title='Arrival Time Vs Ticket Price', xaxis_title='Arrival Time', yaxis_title='Price')
fig.show()
```

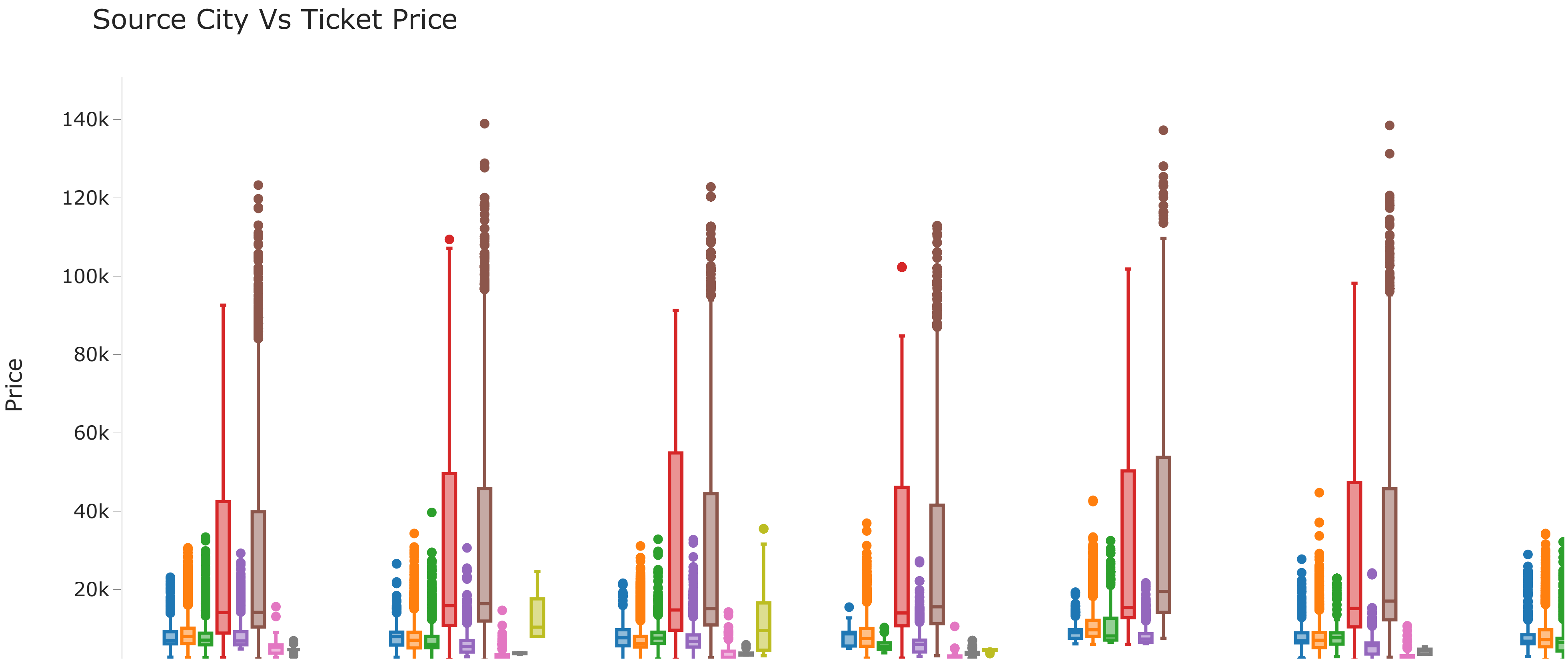

Departure Time Vs Ticket Price
Price
140k
120k
100k
80k
60k
40k
20k

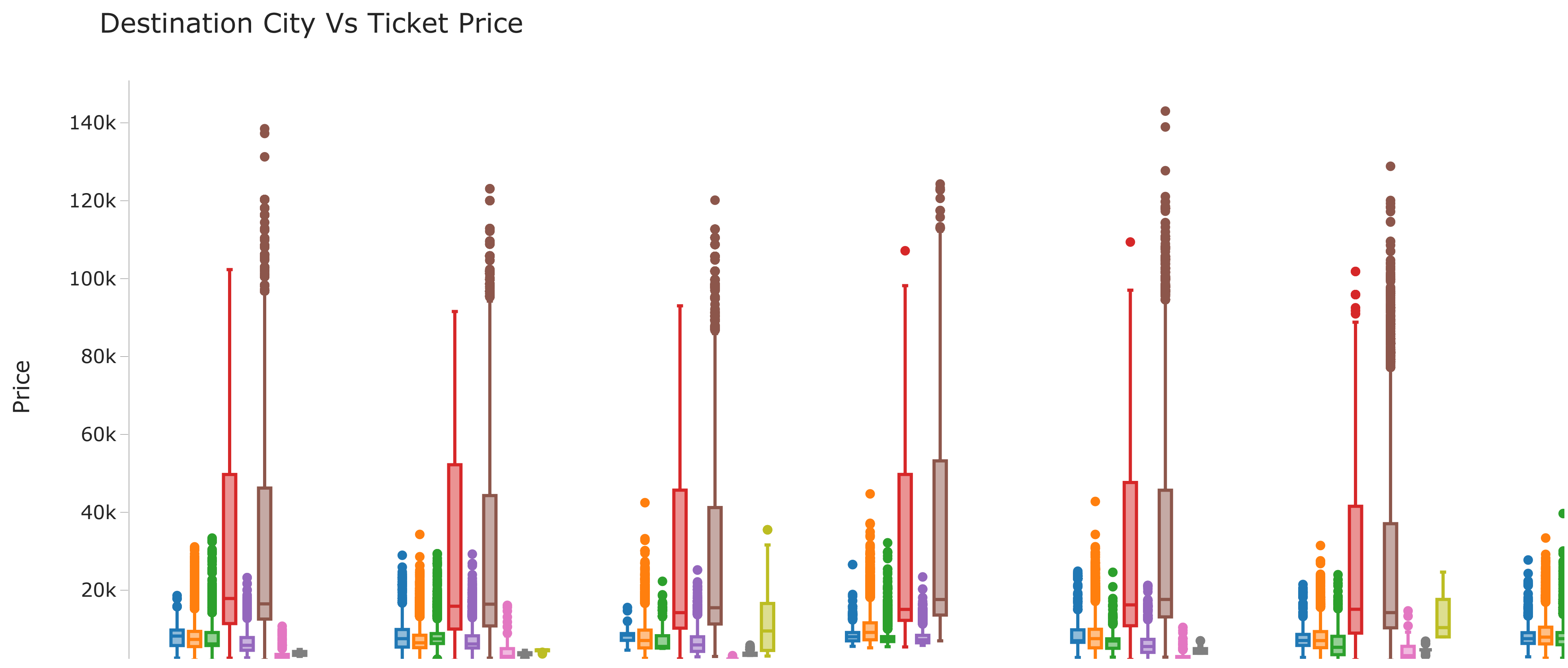


Boxplot of ticket prices by arrival time

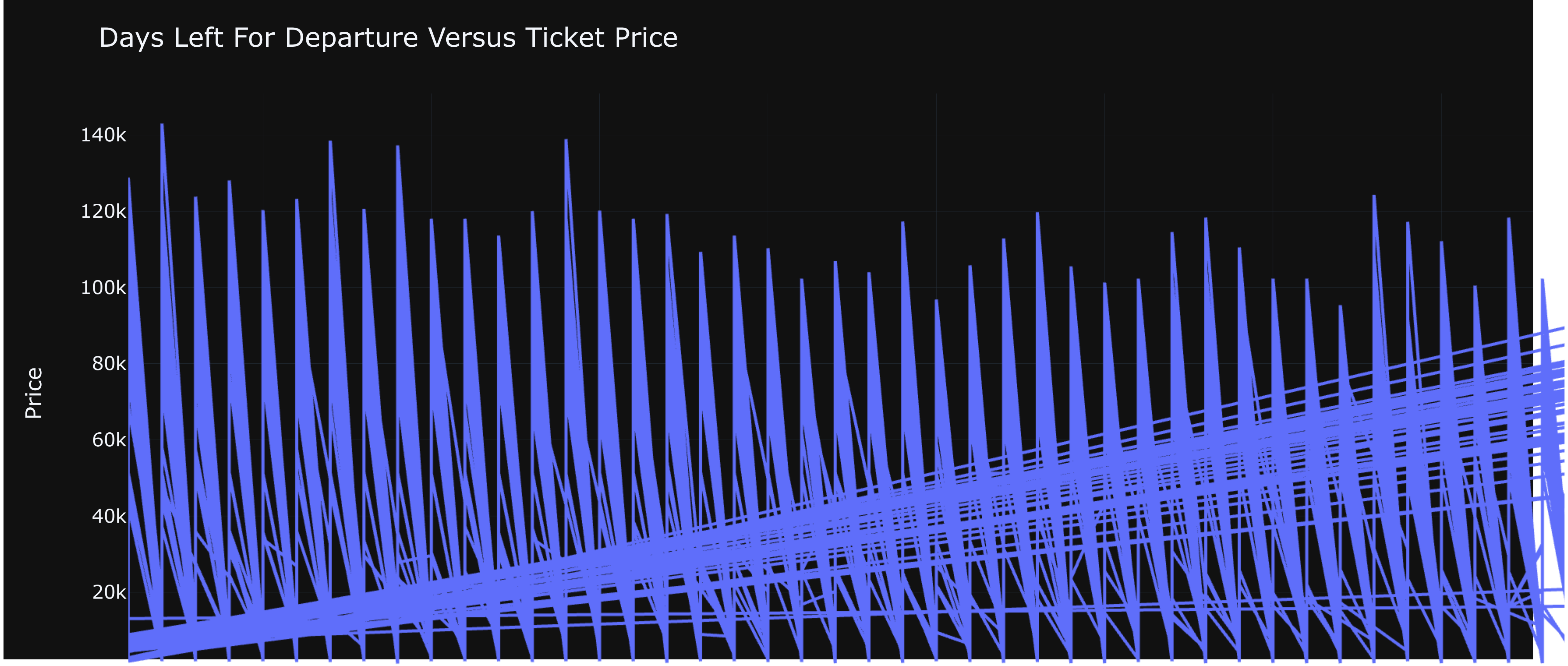
In [20]:

```
fig = px.box(data, x='Source', y='Fare', color='Airline', template='simple_white')
fig.update_layout(title='Source City Vs Ticket Price', xaxis_title='Source City', yaxis_title='Price')
fig.show()

fig = px.box(data, x='Destination', y='Fare', color='Airline', template='simple_white')
fig.update_layout(title='Destination City Vs Ticket Price', xaxis_title='Destination City', yaxis_title='Price')
fig.show()
```


Source City Vs Ticket Price
Price
140k
120k
100k
80k
60k
40k
20k



In [21]:
```
import plotly.express as px

fig = px.line(data, x='Days_left', y='Fare', template='plotly_dark',
              labels={'Days_left': 'Days Left for Departure', 'Fare': 'Price'})
fig.update_layout(title='Days Left For Departure Versus Ticket Price', xaxis_title='Days Left for Departure', yaxis_title='Price')
fig.show()
```



In [13]:
```
data.groupby(['Flight_code','Source','Destination','Airline','Class'],as_index=False).count().groupby(['Source','Destination'],as_index=Fal
```

Out[13]:

	Source	Destination	Flight_code
0	Ahmedabad	Bangalore	85
1	Ahmedabad	Chennai	82
2	Ahmedabad	Delhi	64
3	Ahmedabad	Hyderabad	77
4	Ahmedabad	Kolkata	88
5	Ahmedabad	Mumbai	50
6	Bangalore	Ahmedabad	136
7	Bangalore	Chennai	97
8	Bangalore	Delhi	181
9	Bangalore	Hyderabad	119

In [14]:

```
data.groupby(['Airline','Source','Destination'],as_index=False)['Fare'].mean().head(10)
```

Out[14]:

	Airline	Source	Destination	Fare
0	Air India	Ahmedabad	Bangalore	30898.056017
1	Air India	Ahmedabad	Chennai	31986.554209
2	Air India	Ahmedabad	Delhi	25284.740260
3	Air India	Ahmedabad	Hyderabad	28618.727551
4	Air India	Ahmedabad	Kolkata	30114.170294
5	Air India	Ahmedabad	Mumbai	31228.560304
6	Air India	Bangalore	Ahmedabad	28063.483264
7	Air India	Bangalore	Chennai	28978.088460
8	Air India	Bangalore	Delhi	23134.751645
9	Air India	Bangalore	Hyderabad	27742.954733

Dimension Reduction

In [21]:
```
from sklearn.decomposition import PCA

# Instantiate PCA with desired number of components
pca = PCA(n_components=2)

# Fit PCA to your data and transform it
x_train_pca = pca.fit_transform(x_train)
x_test_pca = pca.transform(x_test)

# Now we can use x_train_pca and x_test_pca for modeling
```

In [22]:
```
from sklearn.decomposition import PCA

# Extract numerical features for PCA
numerical_features = data[['Date_of_journey', 'Journey_day', 'Duration_in_hours', 'Days_left']]

# Initialize PCA with desired number of components
pca = PCA(n_components=2)

# Fit PCA to the numerical features
pca.fit(numerical_features)

# Transform the numerical features to their principal components
numerical_pca = pca.transform(numerical_features)

# Replace original numerical features with principal components
data[['PCA1', 'PCA2']] = numerical_pca
```

In [25]:
```
# Perform one-hot encoding using pandas
categorical_encoded = pd.get_dummies(categorical_features, columns=['Airline', 'Flight_code', 'Class', 'Source', 'Departure', 'Total_stops'

# Concatenate the original DataFrame with the encoded categorical features
data_encoded = pd.concat([data, categorical_encoded], axis=1)

# Drop the original categorical features
data_encoded.drop(columns=categorical_features.columns, inplace=True)
```

In [24]:
```
from sklearn.preprocessing import OneHotEncoder

# Assuming you have a DataFrame 'df' containing your data
# Extract categorical features for One-Hot Encoding
categorical_features = data[['Airline', 'Flight_code', 'Class', 'Source', 'Departure', 'Total_stops', 'Arrival', 'Destination']]

# Initialize One-Hot Encoder
encoder = OneHotEncoder()

# Fit and transform the categorical features
```

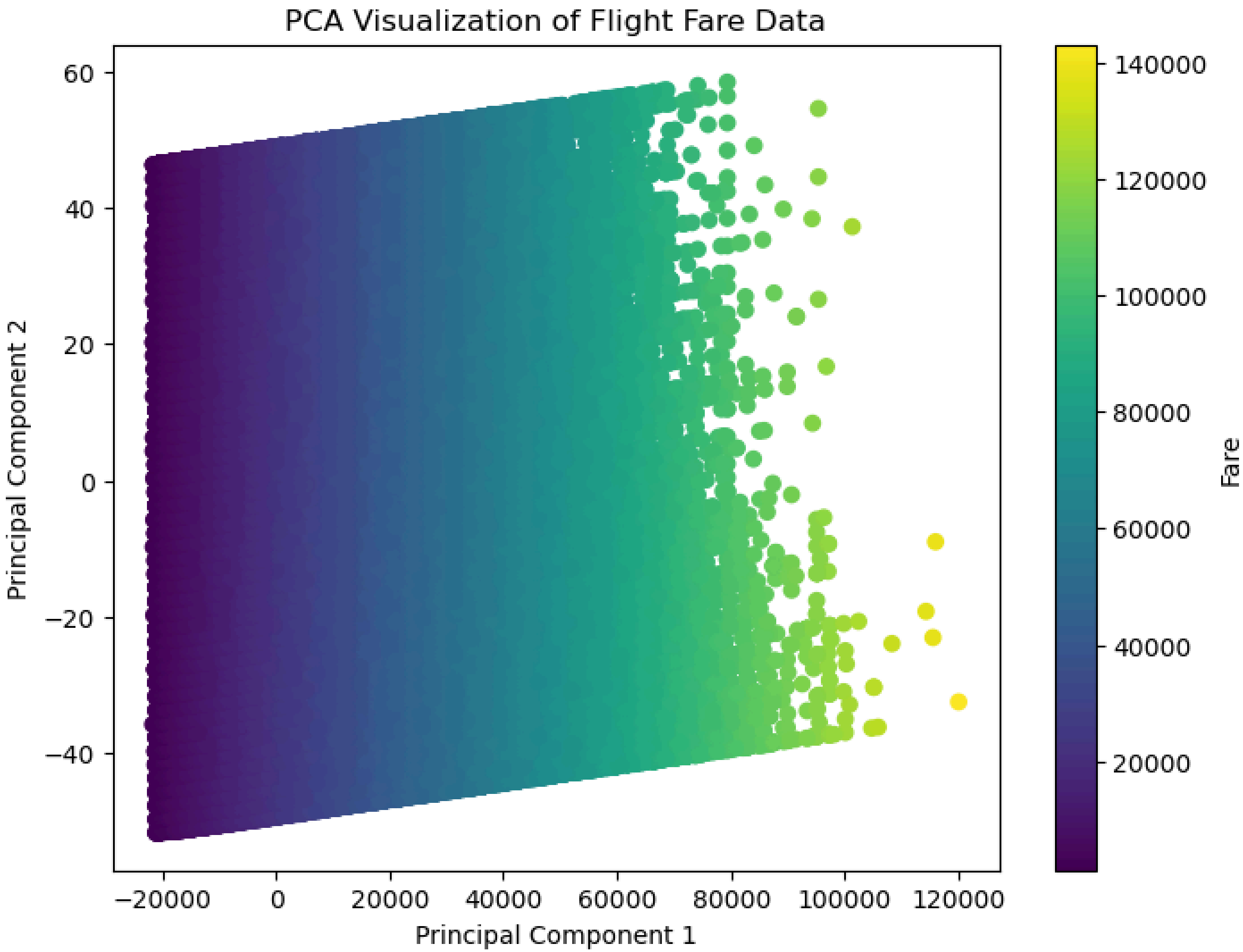
```python
categorical_encoded = encoder.fit_transform(categorical_features)

# Convert the encoded features to a DataFrame
categorical_df = pd.DataFrame(categorical_encoded.toarray(), columns=encoder.get_feature_names_out(categorical_features.columns))

# Concatenate the original DataFrame with the encoded categorical features
df_encoded = pd.concat([data, categorical_df], axis=1)

# Drop the original categorical features
df_encoded.drop(columns=categorical_features.columns, inplace=True)
```

In [26]:

```python
from sklearn.decomposition import PCA

# Initialize PCA with desired number of components
pca = PCA(n_components=2)

# Fit PCA to the data
pca.fit(df_encoded)

# Transform the data to its principal components
X_pca = pca.transform(df_encoded)
```

In [27]:

```python
import matplotlib.pyplot as plt

# Visualize the principal components
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_encoded['Fare'], cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Visualization of Flight Fare Data')
plt.colorbar(label='Fare')
plt.show()
```



In [28]:
```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, df_encoded['Fare'], test_size=0.2, random_state=42)
```

In [29]:
```
# Initialize the machine learning model (e.g., Linear Regression)
model = LinearRegression()

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

Out[29]:
```
▾ LinearRegression
LinearRegression()
```

Predictive Performance Evaluation:

In [30]:
```python
# Predict the fares for the testing data
y_pred = model.predict(X_test)

# Evaluate the performance of the model using appropriate metrics (e.g., Mean Absolute Error, Mean Squared Error, R-squared)
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('Mean Absolute Error:', mae)
print('Mean Squared Error:', mse)
print('R-squared:', r2)
```

```
Mean Absolute Error: 0.0008121091575577499
Mean Squared Error: 1.0082983759049834e-06
R-squared: 0.9999999999999976
```

In [31]:
```python
# Example for Linear Regression
from sklearn.linear_model import LinearRegression

# Assuming we have X_train, X_test, y_train, y_test defined and X_encoded, y_encoded transformed data
model_lr = LinearRegression()
model_lr.fit(X_train, y_train)
y_pred_lr = model_lr.predict(X_test)

# Calculate mean squared error
mse_lr = mean_squared_error(y_test, y_pred_lr)
print('Linear Regression - Mean Squared Error:', mse_lr)
```

```
Linear Regression - Mean Squared Error: 1.0082983759049834e-06
```

In [34]:
```python
from sklearn.tree import DecisionTreeRegressor

# Initialize the Decision Tree Regressor model
model_dt = DecisionTreeRegressor()

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

```python
# Make predictions
y_pred_dt = model_dt.predict(X_test)

# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)

print('Decision Tree Regression - Mean Squared Error:', mse_dt)
print('Decision Tree Regression - R-squared:', r2_dt)
```

```
Decision Tree Regression - Mean Squared Error: 755.699856847463
Decision Tree Regression - R-squared: 0.9999981826807223
```

In [36]:
```python
from sklearn.ensemble import GradientBoostingRegressor

# Initialize the Gradient Boosting Regressor model
model_gb = GradientBoostingRegressor()

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

# Make predictions
y_pred_gb = model_gb.predict(X_test)

# Evaluate the model
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

print('Gradient Boosting Regression - Mean Squared Error:', mse_gb)
print('Gradient Boosting Regression - R-squared:', r2_gb)
```

```
Gradient Boosting Regression - Mean Squared Error: 17359.83363374449
Gradient Boosting Regression - R-squared: 0.9999582527903962
```

In [37]:
```python
from sklearn.metrics import r2_score

# Predictions for each model
y_pred_lr = model_lr.predict(X_test)
y_pred_dt = model_dt.predict(X_test)
y_pred_gb = model_gb.predict(X_test)

# Calculate R-squared values
r2_lr = r2_score(y_test, y_pred_lr)
r2_dt = r2_score(y_test, y_pred_dt)
r2_gb = r2_score(y_test, y_pred_gb)

# Print R-squared values
print('Linear Regression - R-squared:', r2_lr)
```

```
print('Decision Tree Regression - R-squared:', r2_dt)
print('Gradient Boosting Regression - R-squared:', r2_gb)
```

```
Linear Regression - R-squared: 0.9999999999999976
Decision Tree Regression - R-squared: 0.9999981826807223
Gradient Boosting Regression - R-squared: 0.9999582527903962
```

Model Building:

In [15]:
```
# Coverting the labels into a numeric form using Label Encoder
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for col in data.columns:
    if data[col].dtype=='object':
        data[col]=le.fit_transform(data[col])
```

In [16]:
```
# storing the Dependent Variables in X and Independent Variable in Y
x=data.drop(['Fare'],axis=1)
y=data['Fare']
```

In [17]:
```
 #Splitting the Data into Training set and Testing Set
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[17]:
```
((308060, 12), (132027, 12), (308060,), (132027,))
```

In [18]:
```
# Scaling the values to convert the int values to Machine Languages
from sklearn.preprocessing import MinMaxScaler
mmscaler=MinMaxScaler(feature_range=(0,1))
x_train=mmscaler.fit_transform(x_train)
x_test=mmscaler.fit_transform(x_test)
x_train=pd.DataFrame(x_train)
x_test=pd.DataFrame(x_test)
```

In [19]:
```
a={'Model Name':[], 'Mean_Absolute_Error_MAE':[] ,'Adj_R_Square':[] ,'Root_Mean_Squared_Error_RMSE':[] ,'Mean_Absolute_Percentage_Error_MAP
Results=pd.DataFrame(a)
Results.head()
```

Out[19]:

Model Name	Mean_Absolute_Error_MAE	Adj_R_Square	Root_Mean_Squared_Error_RMSE	Mean_Absolute_Percentage_Error_MAPE	Mean_Squared_Error_MSE	Root_Mean_Squ

In [20]:

```python
# Build the Regression / Regressor models

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn import linear_model
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xgb
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import GradientBoostingRegressor
from tqdm.notebook import tqdm_notebook


# Create objects of Regression / Regressor models with default hyper-parameters

modelmlg = LinearRegression()
modeldcr = DecisionTreeRegressor()
#modelbag = BaggingRegressor()
modelrfr = RandomForestRegressor()
modelSVR = SVR()
modelXGR = xgb.XGBRegressor()
modelKNN = KNeighborsRegressor(n_neighbors=5)
modelETR = ExtraTreesRegressor()
modelRE=Ridge()
modelLO=linear_model.Lasso(alpha=0.1)

modelGBR = GradientBoostingRegressor(loss='squared_error', learning_rate=0.1, n_estimators=100, subsample=1.0,
                                     criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1,
                                     min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0,
                                     init=None, random_state=None, max_features=None,
                                     alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False,
                                     validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)

# Evalution matrix for all the algorithms

MM = [modelmlg, modeldcr, modelETR, modelGBR, modelXGR, modelRE, modelLO]

for models in tqdm_notebook(MM):

    # Fit the model with train data

    models.fit(x_train, y_train)

    # Predict the model with test data
```

```python
y_pred = models.predict(x_test)

# Print the model name
#print('Model Name: ', models)

# Evaluation metrics for Regression analysis

from sklearn import metrics

# print('Mean Absolute Error (MAE):', round(metrics.mean_absolute_error(y_test, y_pred),3))
# print('Mean Squared Error (MSE):', round(metrics.mean_squared_error(y_test, y_pred),3))
# print('Root Mean Squared Error (RMSE):', round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)),3))
# print('R2_score:', round(metrics.r2_score(y_test, y_pred),6))
# print('Root Mean Squared Log Error (RMSLE):', round(np.log(np.sqrt(metrics.mean_squared_error(y_test, y_pred))),3))

# Define the function to calculate the MAPE - Mean Absolute Percentage Error

def MAPE (y_test, y_pred):
    y_test, y_pred = np.array(y_test), np.array(y_pred)
    return np.mean(np.abs((y_test - y_pred) / y_test)) * 100

# Evaluation of MAPE

result = MAPE(y_test, y_pred)
# print('Mean Absolute Percentage Error (MAPE):', round(result, 2), '%')
# Calculate Adjusted R squared values

r_squared = round(metrics.r2_score(y_test, y_pred),6)
adjusted_r_squared = round(1 - (1-r_squared)*(len(y)-1)/(len(y)-x.shape[1]-1),6)
# print('Adj R Square: ', adjusted_r_squared)
# print('------------------------------------------------------------------------------------------------------------')
#-------------------------------------------------------------------------------------------
new_row = {'Model Name' : models,
           'Mean_Absolute_Error_MAE' : metrics.mean_absolute_error(y_test, y_pred),
           'Adj_R_Square' : adjusted_r_squared,
           'Root_Mean_Squared_Error_RMSE' : np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
           'Mean_Absolute_Percentage_Error_MAPE' : result,
           'Mean_Squared_Error_MSE' : metrics.mean_squared_error(y_test, y_pred),
           'Root_Mean_Squared_Log_Error_RMSLE': np.log(np.sqrt(metrics.mean_squared_error(y_test, y_pred))),
           'R2_score' : metrics.r2_score(y_test, y_pred)}
Results = Results.append(new_row, ignore_index=True)
```

```
  0%|          | 0/7 [00:00<?, ?it/s]
```

```
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
C:\Users\rajen\AppData\Local\Temp\ipykernel_36036\168480723.py:89: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  Results = Results.append(new_row, ignore_index=True)
```

In [38]:

```
Results
```

Out[38]:

	Model Name	Mean_Absolute_Error_MAE	Adj_R_Square	Root_Mean_Squared_Error_RMSE	Mean_Absolute_Percentage_Error_MAPE	Me
0	LinearRegression()	12362.228696	0.461041	15010.490597	92.868237	
1	DecisionTreeRegressor()	2136.125141	0.935700	5184.689506	9.756202	
2	(ExtraTreeRegressor(random_state=1145937469), ...	1818.452069	0.963324	3915.698477	8.418442	
3	([DecisionTreeRegressor(criterion='friedman_ms...	3945.987181	0.902126	6396.605812	21.716427	
4	XGBRegressor(base_score=None, booster=None, ca...	2692.244562	0.948711	4630.555360	13.976330	
5	Ridge()	12362.211827	0.461041	15010.488388	92.867585	
6	Lasso(alpha=0.1)	12362.244481	0.461042	15010.480087	92.868490	

The results indicate that the Extra Trees Regression model exhibits the lowest MAE and MAPE values, along with the highest adjusted R-squared and R-squared values. These metrics suggest that the Extra Trees Regression model outperforms other models, making it the most effective choice for this dataset.

In [ ]: