Flight Fare Prediction Project

1. As first step, we load our `FlightFare_Dataset` from Project Directory, using Pandas `read_excel` method
2. Then, we perform Feature Exploration and Engineering to transform our dataset
3. Once done, we use a Feature Selection technique to select the most important features
4. At this point, we train a Random Forest Regressor Model
5. As next step, we do hyper-parameter tuning (using `RandomGridSearch`) to build the best model
6. Finally, we export Model `.pkl` file back to Project Directory
7. Towards the end, we proceed to Model Deployment step

Set up Environment

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import metrics
import seaborn as sns
sns.set()
```

```
# Mount Google Drive - applicable, if working on Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Set Working Directory - if working on Google Drive
%cd /content/drive/MyDrive/ML-project/flight-fare-prediction

# # Set Working Directory - if working on Local Machine
# import os
# os.chdir('/Users//replace_me')
```

```
/content/drive/MyDrive/ML-project/flight-fare-prediction
```

Load Dataset

```
# Load dataset from Project folder
dataset = pd.read_excel("Data_Train.xlsx")

# To stretch head function output to the notebook width
pd.set_option('display.max_columns', None)

dataset.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Dur
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR →	05:50	13:15	7

Next steps: View recommended plots

```
dataset.info()       # Print Data Types
```

```
<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
```

```
dataset.shape
```

```
(10683, 11)
```

Check for missing values

```
dataset.isnull().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
```

```
dataset.dropna(inplace = True)
```

```
dataset.isnull().sum()
```

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

Pre Processing Task

```
dataset.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Dur
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR →	05:50	13:15	7

Next steps: View recommended plots

Handling Object Data

`Date_of_Journey`, `Dep_Time`, `Arrival_Time`, `Duration` are object datatype. To derive numeric features on these, we use pandas `to_datetime` method to convert object data type to datetime datatype.

Attribute `.dt.day` will extract day from the date

Attribute `.dt.month` will extract month from that date

[ ] ↳ *7 cells hidden*

Handling Categorical Data

`Airline`, `Source`, `Destination`, `Route`, `Total_Stops`, `Additional_Info` **are all categorical.** One can find many ways to handle categorical data, like:

1. **Nominal data** --> data is not in any order --> **OneHotEncoder** is used in this case
2. **Ordinal data** --> data is in order --> **LabelEncoder** is used in this case

```
# Feature engineering on: Airline
dataset["Airline"].value_counts()
```

```
Airline
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: count, dtype: int64
```

```
# As Airline is Nominal Categorical data we will perform OneHotEncoding
Airline = dataset[["Airline"]]
Current_Airline_List = Airline['Airline']
New_Airline_List = []

for carrier in Current_Airline_List:
  if carrier in ['Jet Airways', 'IndiGo', 'Air India', 'SpiceJet',
       'Multiple carriers', 'GoAir', 'Vistara', 'Air Asia']:
    New_Airline_List.append(carrier)
  else:
    New_Airline_List.append('Other')

Airline['Airline'] = pd.DataFrame(New_Airline_List)
Airline['Airline'].value_counts()
```

```
<ipython-input-97-5653ef352d3c>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  Airline['Airline'] = pd.DataFrame(New_Airline_List)
Airline
Jet Airways          3849
IndiGo               2053
Air India            1750
Multiple carriers    1196
SpiceJet              818
Vistara               479
Air Asia              319
GoAir                 194
Other                  23
Name: count, dtype: int64
```

```
Airline = pd.get_dummies(Airline, drop_first= True)
Airline.head()
```

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Multiple carriers	Airline_Other	Airline_SpiceJet	Airline_Vistara
0	False	False	True	False	False	False	False	False
1	True	False	False	False	False	False	False	False
2	False	False	False	True	False	False	False	False
3	False	False	True	False	False	False	False	False

Next steps: View recommended plots

```
# Feature engineering on: Source
dataset["Source"].value_counts()
```

```
Source
Delhi       4536
Kolkata     2871
Banglore    2197
```

```
Mumbai       697
Chennai      381
Name: count, dtype: int64
```

```
# As Source is Nominal Categorical data we will perform OneHotEncoding
Source = dataset[["Source"]]
Source = pd.get_dummies(Source, drop_first= True)
# drop_first= True means we drop the first column to prevent multicollinearity
Source.head()
```

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	False	False	False	False
1	False	False	True	False
2	False	True	False	False
3	False	False	True	False
4	False	False	False	False

Next steps: View recommended plots

```
# Feature engineering on: Destination
dataset["Destination"].value_counts()
```

```
Destination
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
Hyderabad     697
Kolkata       381
Name: count, dtype: int64
```

```
# Renaming destination 'New Delhi' to 'Delhi' - to match with Source
Destination = dataset[["Destination"]]
Current_Destination_List = Destination['Destination']
New_Destination_List = []

for value in Current_Destination_List:
  if value in ['New Delhi']:
    New_Destination_List.append('Delhi')
  else:
    New_Destination_List.append(value)

Destination['Destination'] = pd.DataFrame(New_Destination_List)

# As Destination is Nominal Categorical data we will perform OneHotEncoding
Destination = pd.get_dummies(Destination, drop_first = True)
Destination.head()
```

```
<ipython-input-102-f6df4e290dd3>:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  Destination['Destination'] = pd.DataFrame(New_Destination_List)
```

	Destination_Cochin	Destination_Delhi	Destination_Hyderabad	Destination_Kolkata
0	False	True	False	False
1	False	False	False	False
2	True	False	False	False
3	False	False	False	False
4	False	True	False	False

Next steps: View recommended plots

```
dataset["Additional_Info"].value_counts()
```

```
Additional_Info
No info                         8344
In-flight meal not included     1982
No check-in baggage included     320
1 Long layover                    19
Change airports                    7
Business class                     4
No Info                            3
```

```
1 Short layover                1
Red-eye flight                 1
2 Long layover                 1
Name: count, dtype: int64
```

```
# Additional_Info contains almost 80% no_info

# Route and Total_Stops are related to each other
dataset.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

```
# Feature engineering on: Total_Stops
dataset["Total_Stops"].value_counts()
```

```
Total_Stops
1 stop      5625
non-stop    3491
2 stops     1520
3 stops       45
4 stops        1
Name: count, dtype: int64
```

```
# As this is case of Ordinal Categorical type we perform LabelEncoder
# Here Values are assigned with corresponding keys
dataset.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
dataset.head()
```

	Airline	Source	Destination	Total_Stops	Price	journey_day	journey_month	dep_hour	dep_min	arrival_hour	arrival_min	Dura
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30	

Next steps: View recommended plots

```
# Concatenate dataframe --> train_data + Airline + Source + Destination
data_train = pd.concat([dataset, Airline, Source, Destination], axis = 1) # axis = 1 signifies column
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

data_train.head()
```

	Total_Stops	Price	journey_day	journey_month	dep_hour	dep_min	arrival_hour	arrival_min	Duration_hours	Duration_mins	Airl
0	0	3897	24	3	22	20	1	10	2	50	
1	2	7662	1	5	5	50	13	15	7	25	
2	2	13882	9	6	9	25	4	25	19	0	
3	1	6218	12	5	18	5	23	30	5	25	
4	1	13302	1	3	16	50	21	35	4	45	

```
data_train.shape
```

```
(10682, 26)
```

Feature Selection

Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods:

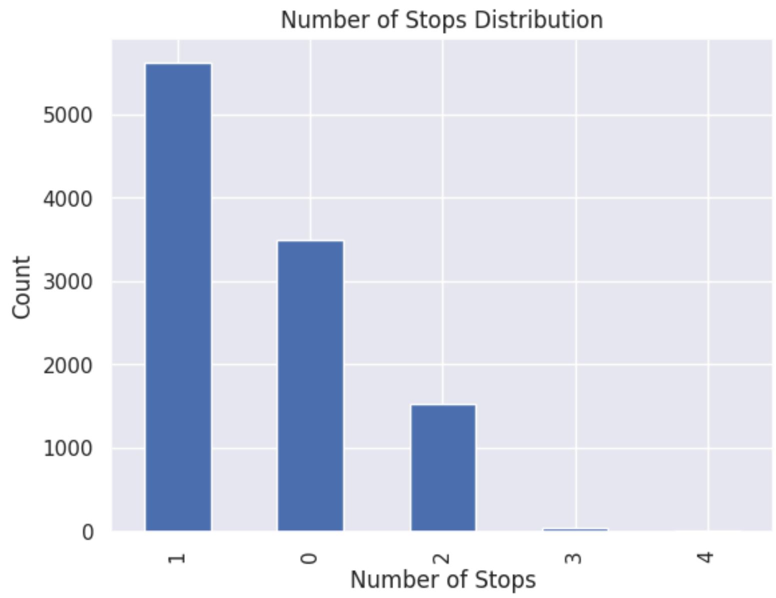
1. **feature_importance_**: To check for relative feature importance
2. **Variable Inflation Factor (VIF)**: To check for multicollinearity

[ ] ↳ *9 cells hidden*

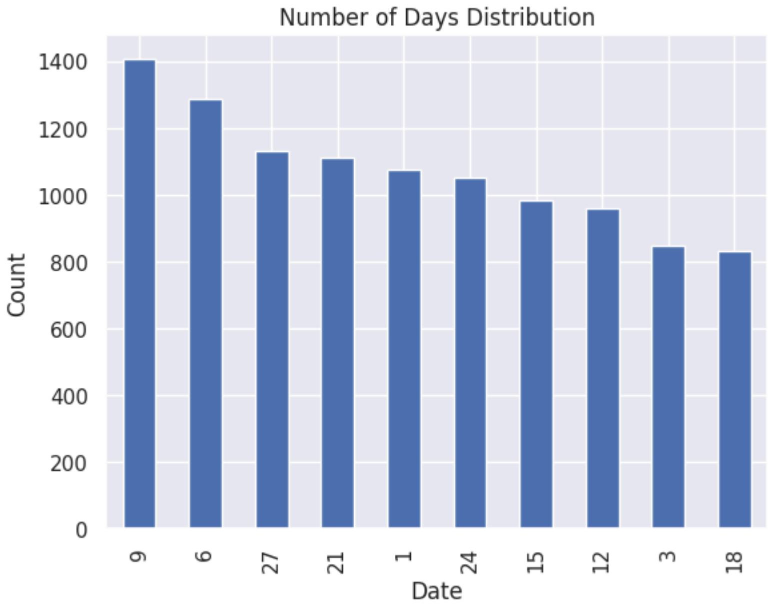
Univariate Analysis

```
#univariate analysis
total_stops_counts=X['Total_Stops'].value_counts()
count_journey_days=X['journey_day'].value_counts()
count_journey_month=X['journey_month'].value_counts()
air_india_counts=X['Airline_Air India'].value_counts()
goair_counts=X['Airline_GoAir'].value_counts()
jetairways_counts=X['Airline_Jet Airways'].value_counts()
indigo_counts=X['Airline_IndiGo'].value_counts()
multi_carriers_counts=X['Airline_Multiple carriers'].value_counts()
other_airline_counts=X['Airline_Other'].value_counts()
spicejet_counts=X['Airline_SpiceJet'].value_counts()
vistara_counts=X['Airline_Vistara'].value_counts()
src_chennai=X['Source_Chennai'].value_counts()
src_kolkata=X['Source_Kolkata'].value_counts()
src_mumbai=X['Source_Mumbai'].value_counts()
des_delhi=X['Destination_Delhi'].value_counts()
des_cochin=X['Destination_Cochin'].value_counts()
des_hyderabad=X['Destination_Hyderabad'].value_counts()
des_kolkata=X['Destination_Kolkata'].value_counts()

mean_dep_hour=X['dep_hour'].mean()
mean_dep_min=X['dep_min'].mean()
mean_arrival_hour=X['arrival_hour'].mean()
mean_arrival_min=X['arrival_min'].mean()
mean_duration_hours=X['Duration_hours'].mean()
mean_duration_min=X['Duration_mins'].mean()
```

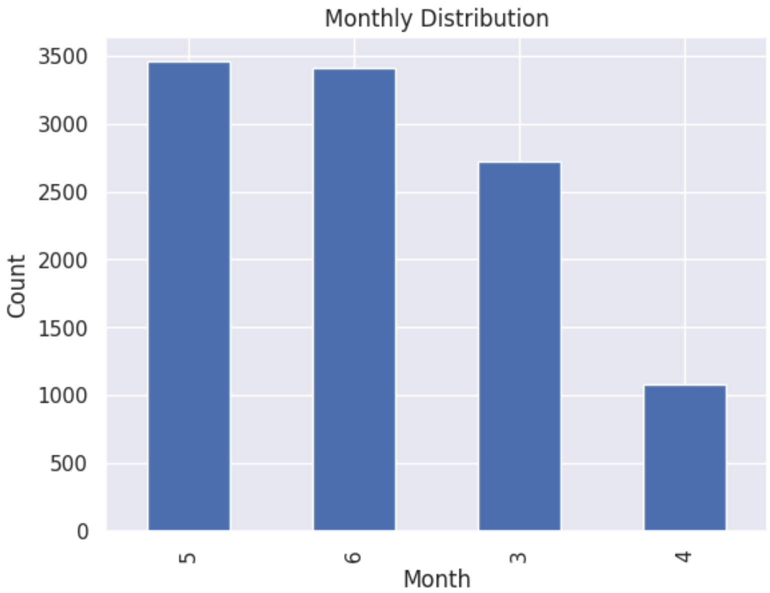
```
import matplotlib.pyplot as plt

total_stops_counts.plot(kind='bar')

# Adding labels and title
plt.xlabel('Number of Stops')
plt.ylabel('Count')
plt.title('Number of Stops Distribution')

# Displaying the plot
plt.show()
```



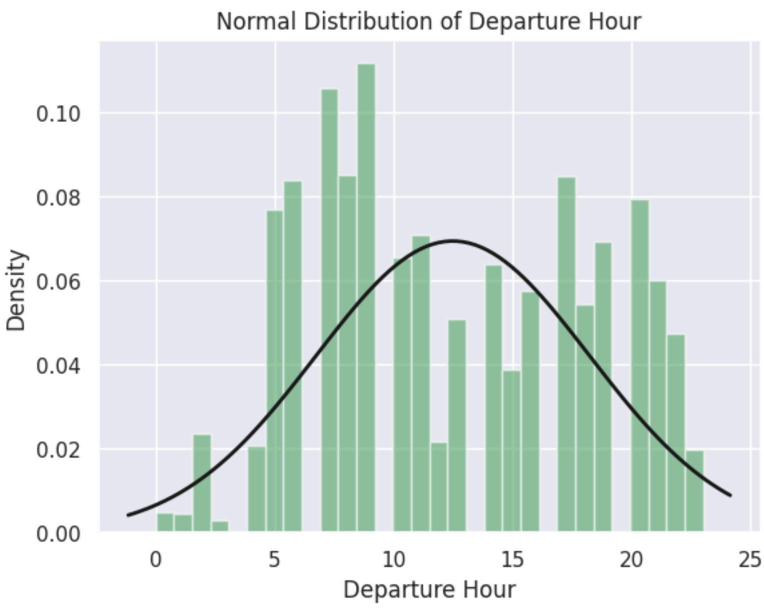

```
count_journey_days.plot(kind='bar')

# Adding labels and title
plt.xlabel('Date')
plt.ylabel('Count')
plt.title('Number of Days Distribution')

# Displaying the plot
plt.show()
```



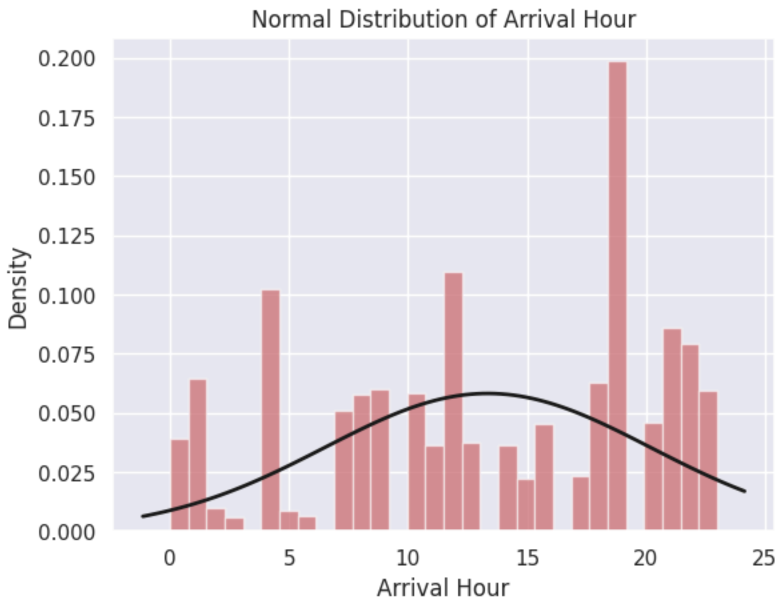

```
count_journey_month.plot(kind='bar')

# Adding labels and title
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Monthly Distribution')

# Displaying the plot
plt.show()
```



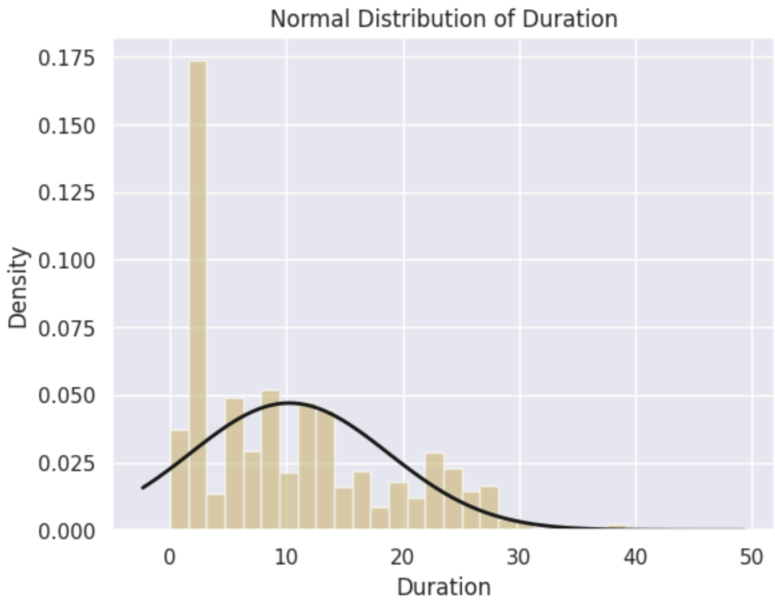
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Assuming 'dep_hour' is the name of the column
dep_hour_values = X['dep_hour']

# Fit a normal distribution to the data
mu, std = norm.fit(dep_hour_values)

# Plot the histogram
plt.hist(dep_hour_values, bins=30, density=True, alpha=0.6, color='g')

# Plot the PDF (Probability Density Function)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)

# Adding labels and title
plt.xlabel('Departure Hour')
plt.ylabel('Density')
plt.title('Normal Distribution of Departure Hour')

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




```
# Assuming 'dep_hour' is the name of the column
arrival_hour_values = X['arrival_hour']

# Fit a normal distribution to the data
mu, std = norm.fit(arrival_hour_values)

# Plot the histogram
plt.hist(arrival_hour_values, bins=30, density=True, alpha=0.6, color='r')

# Plot the PDF (Probability Density Function)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)

# Adding labels and title
plt.xlabel('Arrival Hour')
plt.ylabel('Density')
plt.title('Normal Distribution of Arrival Hour')

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




```
# Assuming 'dep_hour' is the name of the column
dur_hour_values = X['Duration_hours']

# Fit a normal distribution to the data
mu, std = norm.fit(dur_hour_values)

# Plot the histogram
plt.hist(dur_hour_values, bins=30, density=True, alpha=0.6, color='y')

# Plot the PDF (Probability Density Function)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)

# Adding labels and title
plt.xlabel('Duration')
plt.ylabel('Density')
plt.title('Normal Distribution of Duration')

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



Multivariate Analysis

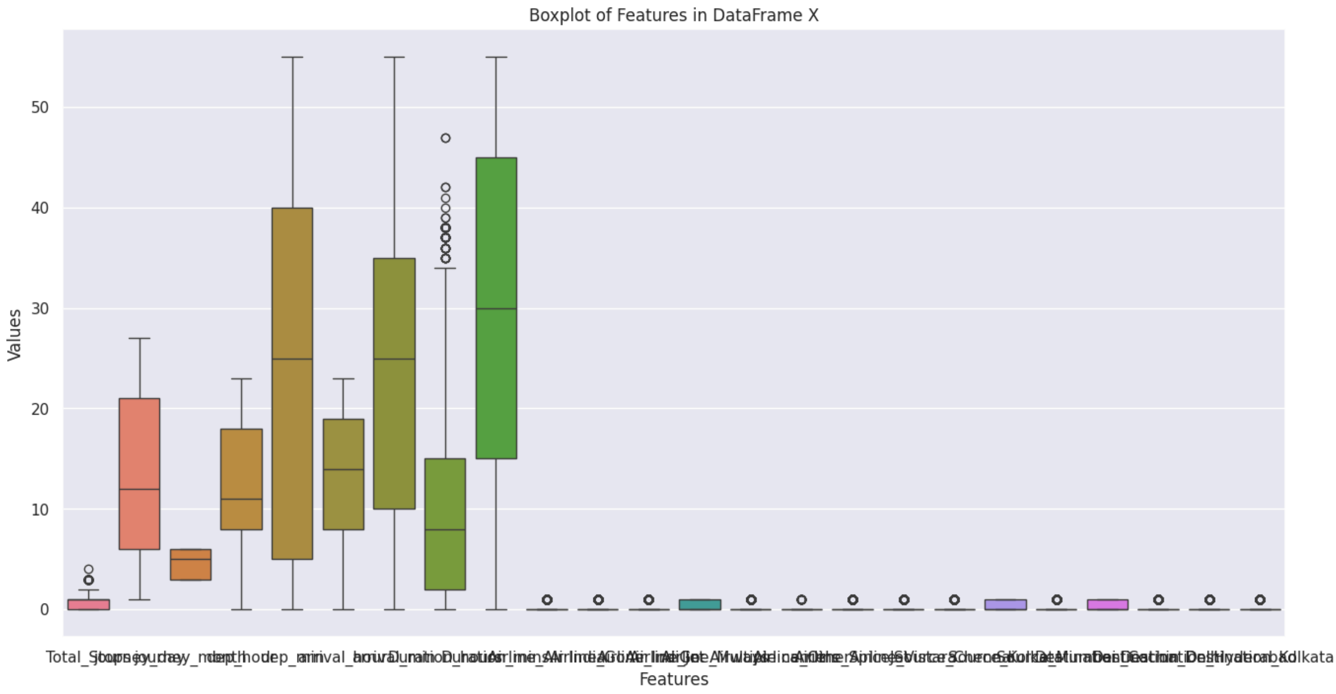
```
# import seaborn as sns
# import matplotlib.pyplot as plt

# Create pairplot
# pairplot = sns.pairplot(X)
# plt.savefig('pairplot.png')  # Save the image as 'pairplot.png'
# plt.show()
```

OUTLIER DETECTION

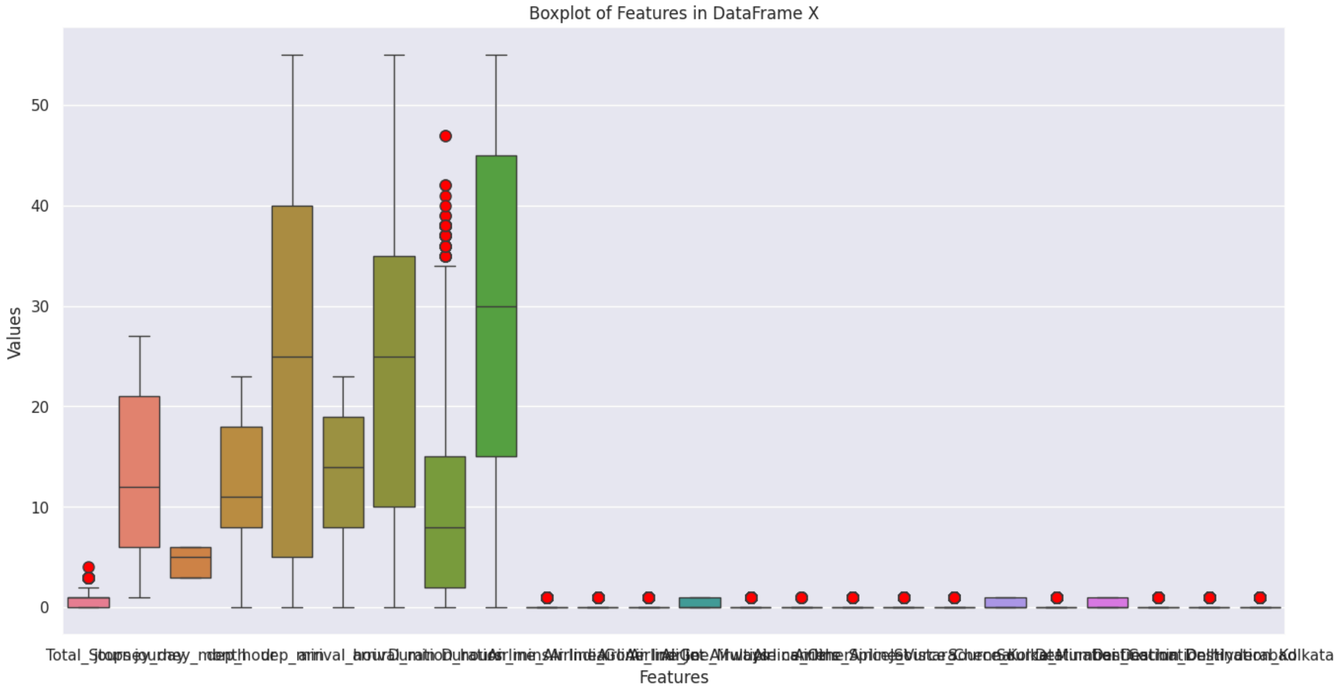
```
X.columns
```

```
Index(['Total_Stops', 'journey_day', 'journey_month', 'dep_hour', 'dep_min',
       'arrival_hour', 'arrival_min', 'Duration_hours', 'Duration_mins',
       'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Multiple carriers', 'Airline_Other',
       'Airline_SpiceJet', 'Airline_Vistara', 'Source_Chennai',
       'Source_Kolkata', 'Source_Mumbai', 'Destination_Cochin',
       'Destination_Delhi', 'Destination_Hyderabad', 'Destination_Kolkata'],
      dtype='object')
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have your DataFrame X
# Convert object data types to numeric, coerce invalid parsing to NaN
X = X.apply(pd.to_numeric, errors='coerce')

# Dropping columns with non-numeric data if needed (optional)
# X = X.select_dtypes(include=[np.number])

# Plotting boxplot for each numeric column
plt.figure(figsize=(16, 8))
sns.boxplot(data=X)
plt.xlabel("Features")
plt.ylabel("Values")
plt.title("Boxplot of Features in DataFrame X")

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




```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data: Assuming you have a DataFrame X with the specified columns
# Convert all columns to numeric data types, coercing invalid parsing to NaN
X = X.apply(pd.to_numeric, errors='coerce')

# Create a boxplot with seaborn
plt.figure(figsize=(16, 8))

# Customize the appearance of outliers using flierprops
flierprops = {
    'marker': 'o',     # Set the marker type (e.g., 'o' for circle)
    'markersize': 8,   # Set the size of the marker
    'markerfacecolor': 'red',  # Set the marker face color to red
    'linestyle': 'none' # No lines for the markers
}

# Plot the boxplot
sns.boxplot(data=X, flierprops=flierprops)

# Add labels and title
plt.xlabel("Features")
plt.ylabel("Values")
plt.title("Boxplot of Features in DataFrame X")

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



ANALYSE CORRELATIONS

```
# Data Train Columns: 'Total_Stops', 'Price', 'journey_day', 'journey_month', 'dep_hour',
#        'dep_min', 'arrival_hour', 'arrival_min', 'Duration_hours',
#        'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
#        'Airline_Jet Airways', 'Airline_Multiple carriers', 'Airline_Other',
#        'Airline_SpiceJet', 'Airline_Vistara', 'Source_Chennai', 'Source_Delhi',
#        'Source_Kolkata', 'Source_Mumbai', 'Destination_Cochin',
#        'Destination_Delhi', 'Destination_Hyderabad', 'Destination_Kolkata'

Z = data_train.copy()
```

```
Z.drop(['dep_hour', 'dep_min', 'arrival_hour','arrival_min'], axis=1, inplace=True)
Z.columns
```

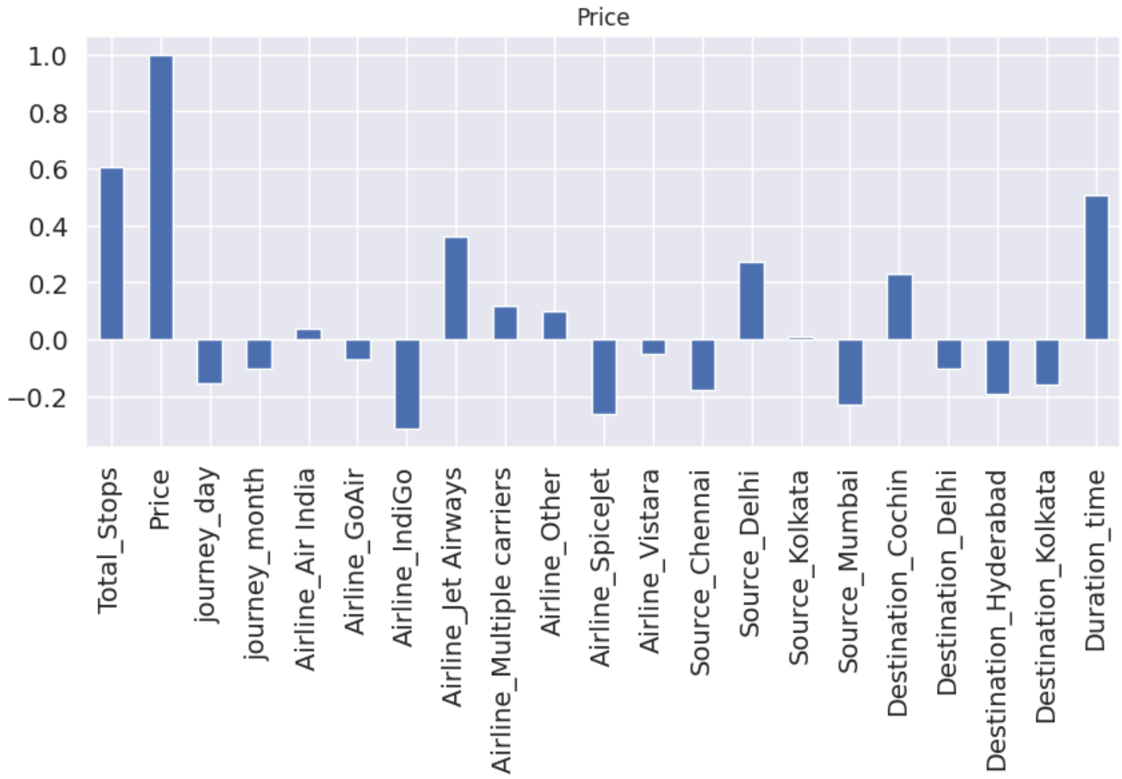
```
Index(['Total_Stops', 'Price', 'journey_day', 'journey_month',
       'Duration_hours', 'Duration_mins', 'Airline_Air India', 'Airline_GoAir',
       'Airline_IndiGo', 'Airline_Jet Airways', 'Airline_Multiple carriers',
       'Airline_Other', 'Airline_SpiceJet', 'Airline_Vistara',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata'],
      dtype='object')
```

```
Z['Duration_time'] = Z['Duration_hours'] * 60 + Z['Duration_mins']
Z.drop(['Duration_hours','Duration_mins'], axis=1, inplace=True)
display(Z.head())
```

	Total_Stops	Price	journey_day	journey_month	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Multiple carriers	Airlin
0	0	3897	24	3	False	False	True	False	False	
1	2	7662	1	5	True	False	False	False	False	
2	2	13882	9	6	False	False	False	True	False	
3	1	6218	12	5	False	False	True	False	False	
4	1	13302	1	3	False	False	True	False	False	

```
Z.corr().loc['Price'].plot(kind= 'bar', figsize=(10,4), grid=True, fontsize=14,title='Price' )
```

<Axes: title={'center': 'Price'}>




```
Z.drop(['Airline_Air India','Airline_GoAir', 'Airline_Multiple carriers', 'Airline_Other','Airline_Vistara', 'Source_Chennai','Source_Kc
         'Destination_Delhi', 'Destination_Kolkata' ], axis=1, inplace=True)
display(Z.head())
```

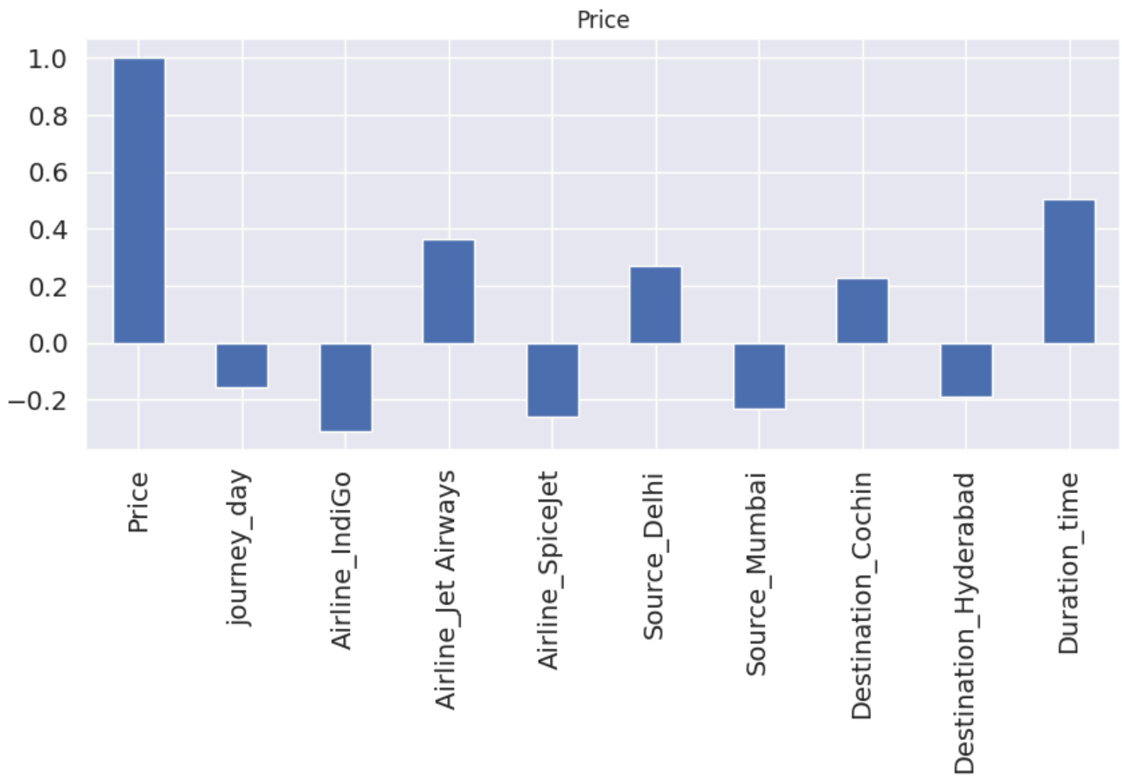
	Total_Stops	Price	journey_day	journey_month	Airline_IndiGo	Airline_Jet Airways	Airline_SpiceJet	Source_Delhi	Source_Mumbai	Desti
0	0	3897	24	3	True	False	False	False	False	
1	2	7662	1	5	False	False	False	False	False	
2	2	13882	9	6	False	True	False	True	False	
3	1	6218	12	5	True	False	False	False	False	

```
Z.corr().loc['Price'].plot(kind= 'bar', figsize=(10,4), grid=True, fontsize=14,title='Price' )
```

```
<Axes: title={'center': 'Price'}>
```


```
Z.drop(['journey_month', 'Total_Stops'], axis=1, inplace=True)
Z.corr().loc['Price'].plot(kind= 'bar', figsize=(10,4), grid=True, fontsize=14,title='Price' )
```

```
<Axes: title={'center': 'Price'}>
```




```
Z.drop(['Airline_SpiceJet'], axis=1, inplace=True)
Z.corr().loc['Price'].plot(kind= 'bar', figsize=(10,4), grid=True, fontsize=14,title='Price' )
```

```
<Axes: title={'center': 'Price'}>
```


```
display(Z.columns)
display(Z.shape)
Z.head()
```

```
Index(['Price', 'journey_day', 'Airline_IndiGo', 'Airline_Jet Airways',
       'Source_Delhi', 'Source_Mumbai', 'Destination_Cochin',
       'Destination_Hyderabad', 'Duration_time'],
      dtype='object')
(10682, 9)
```

	Price	journey_day	Airline_IndiGo	Airline_Jet Airways	Source_Delhi	Source_Mumbai	Dest
0	3897	24	True	False	False	False	
1	7662	1	False	False	False	False	
2	13882	9	False	True	True	False	
3	6218	12	True	False	False	False	

Next steps: View recommended plots

```
train_df=Z.copy()
```

Dimensionality Reduction Using PCA

Using 3 PCS

```
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

train_df=Z.copy()

# Separate features (X) and target variable (y) from train_df
X = train_df.drop('Price', axis=1)  # Features
y = train_df['Price']  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=None)

# Standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components=3)  # Reduce to 3 principal components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Now you have X_train_pca, X_test_pca, y_train, and y_test ready for training and testing your model
# You can use X_train_pca and y_train for training your model and X_test_pca, y_test for evaluating its performance
```

Using LR

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Initialize Linear Regression model
linear_reg = LinearRegression()

# Train the model on the PCA-transformed training data
linear_reg.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = linear_reg.predict(X_test_pca)

# Evaluate the model
lr_mse = mean_squared_error(y_test, y_pred)
lr_rmse = mean_squared_error(y_test, y_pred, squared=False)  # calculating RMSE from MSE
lr_mae = mean_absolute_error(y_test, y_pred)

print("Mean Squared Error (MSE):", lr_mse)
print("Root Mean Squared Error (RMSE):", lr_rmse)
print("Mean Absolute Error (MAE):", lr_mae)
```

```
Mean Squared Error (MSE): 13649027.132757816
Root Mean Squared Error (RMSE): 3694.458977002968
Mean Absolute Error (MAE): 2608.3320743232753
```

Using Random Forests as the Training Algorithm

Training Model with MSE as an error metric

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Initialize Random Forest regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=None)

# Train the model on the PCA-transformed training data
rf_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = rf_regressor.predict(X_test_pca)

# Evaluate the model
rf_mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", rf_mse)
```

```
Mean Squared Error: 10816875.291248582
```

Training Model with RMSE as an Error Metric

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Initialize Random Forest regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the PCA-transformed training data
rf_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = rf_regressor.predict(X_test_pca)

# Calculate MSE
mse = mean_squared_error(y_test, y_pred)

# Calculate RMSE
rf_rmse = np.sqrt(mse)

print("Root Mean Squared Error:", rf_rmse)
```

```
Root Mean Squared Error: 3285.5765015593643
```

Using MAE as an error metric

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Initialize Random Forest regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model on the PCA-transformed training data
rf_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = rf_regressor.predict(X_test_pca)

# Calculate MAE
rf_mae = mean_absolute_error(y_test, y_pred)

print("Mean Absolute Error:", rf_mae)
```

```
Mean Absolute Error: 2254.4253580850764
```

Using KNN as the training algorithm

Using MSE as an error metric

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Initialize KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)

# Train the model on the PCA-transformed training data
knn_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = knn_regressor.predict(X_test_pca)

# Evaluate the model
knn_mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", knn_mse)
```

```
Mean Squared Error: 10296472.20565014
```

Using RMSE as an error metric

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Initialize KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)

# Train the model on the PCA-transformed training data
knn_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = knn_regressor.predict(X_test_pca)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
knn_rmse=np.sqrt(mse)
print("Root Mean Squared Error:", knn_rmse)
```

```
Root Mean Squared Error: 3208.8116500739243
```

Using MAE as an EM

```
# Calculate MAE
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error

# Initialize KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)

# Train the model on the PCA-transformed training data
knn_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = knn_regressor.predict(X_test_pca)
knn_mae = mean_absolute_error(y_test, y_pred)

print("Mean Absolute Error:", knn_mae)
```

```
Mean Absolute Error: 2212.7521047708137
```

Using SVR as the Training Algorithm

[ ] ↳ *6 cells hidden*

Analysing the Error Metrics

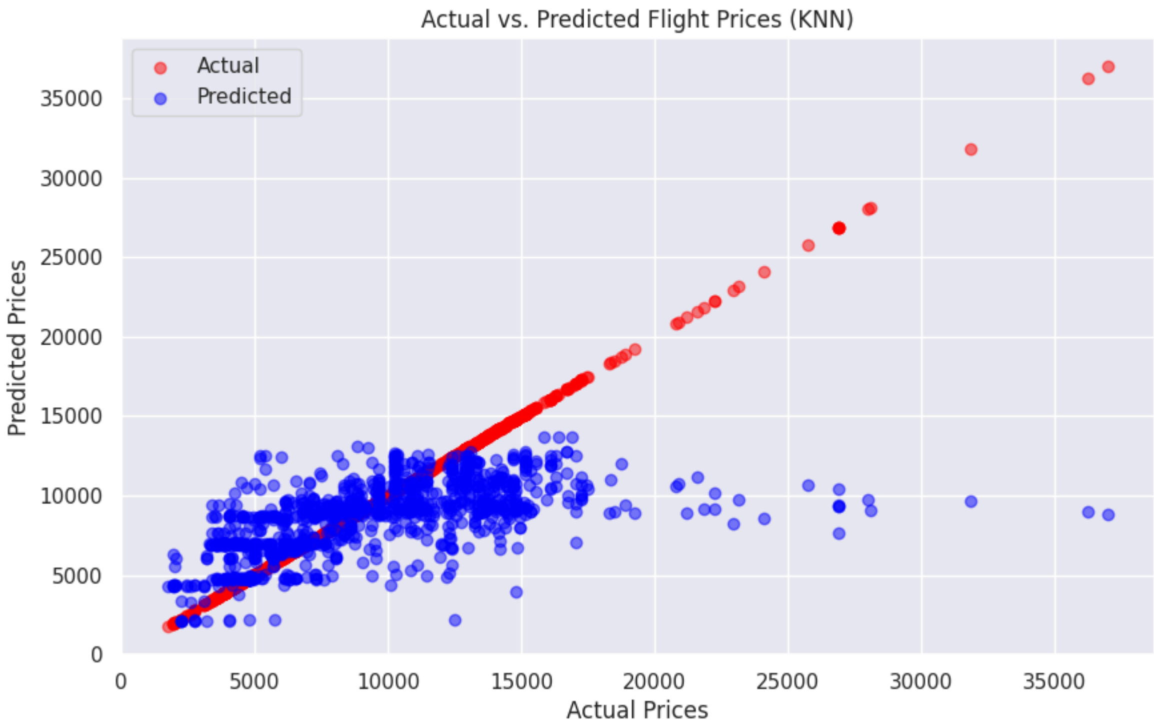
[ ] ↳ *2 cells hidden*

Visualising the Algorithm

Double-click (or enter) to edit

Scatter Plot

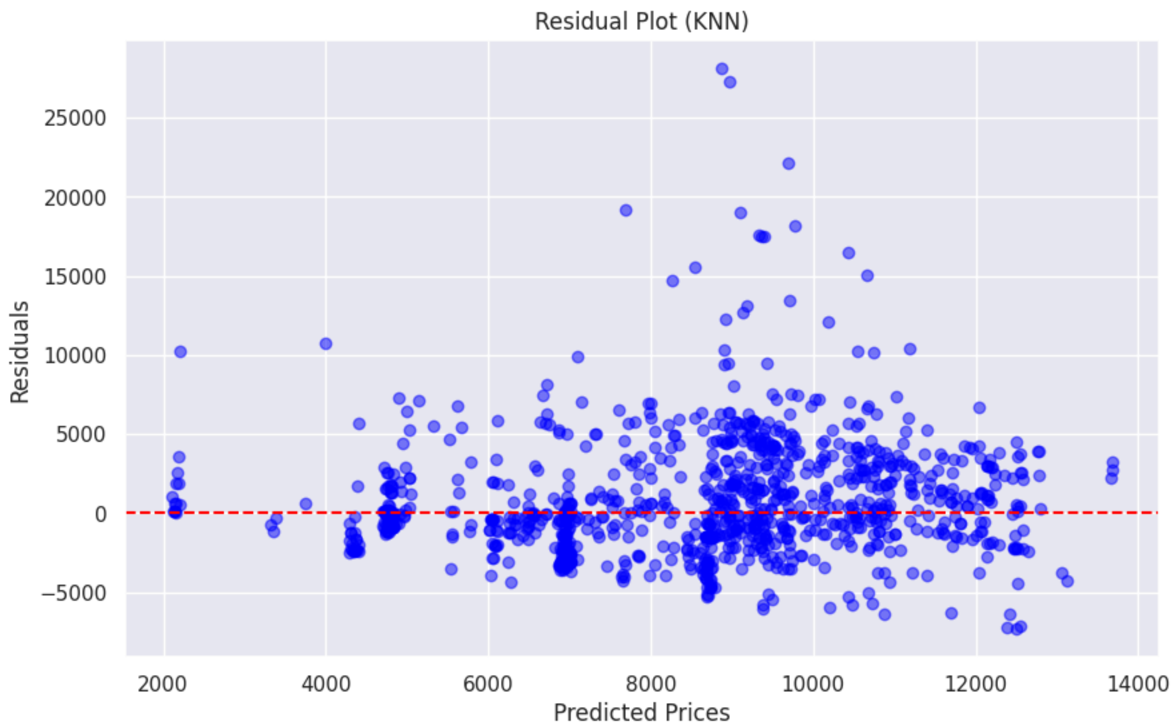
```
import matplotlib.pyplot as plt

# Create a scatter plot for actual vs. predicted flight prices
plt.figure(figsize=(10, 6))

# Plot actual prices in red
plt.scatter(y_test, y_test, color='red', label='Actual', alpha=0.5)

# Plot predicted prices in blue
plt.scatter(y_test, y_pred, color='blue', label='Predicted', alpha=0.5)

# Add labels and title
plt.title('Actual vs. Predicted Flight Prices (KNN)')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.legend()
plt.show()
```



Double-click (or enter) to edit

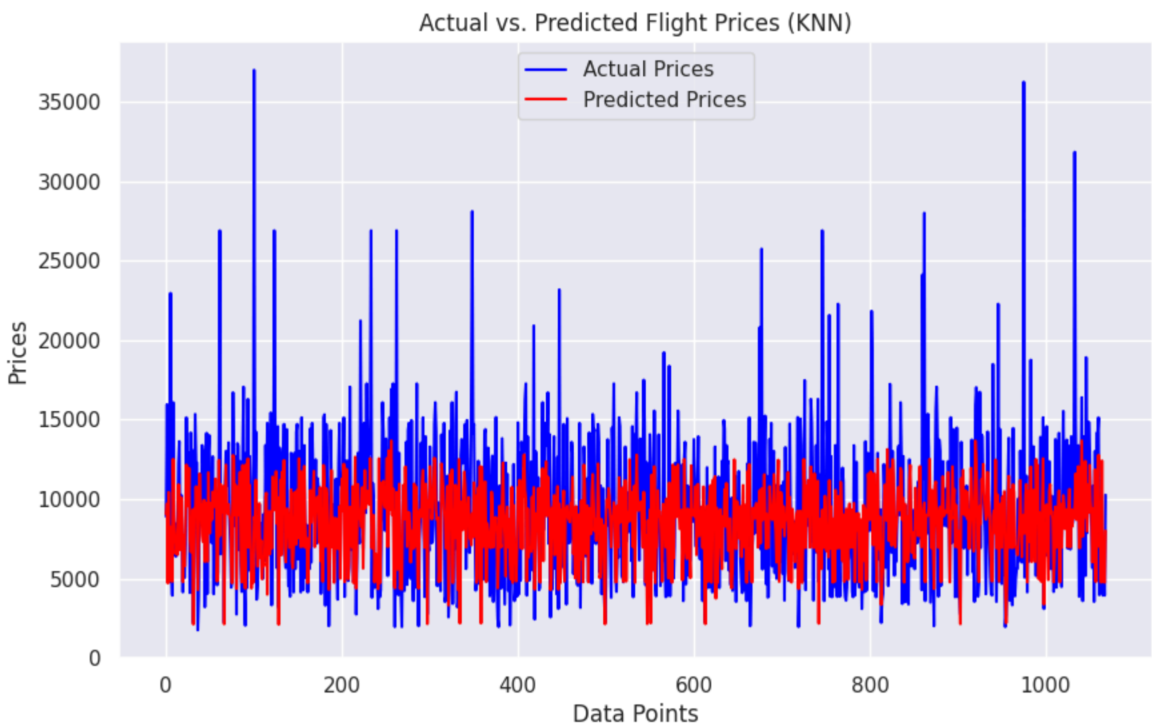
Residual Plot

```
import matplotlib.pyplot as plt

# Calculate residuals
residuals = y_test - y_pred

# Create a residual plot
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals, color='blue', alpha=0.5)
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Residual Plot (KNN)')
plt.xlabel('Predicted Prices')
plt.ylabel('Residuals')
plt.show()
```



Actual vs Predicted Line Plot

```
import matplotlib.pyplot as plt

# Create a line plot of actual and predicted prices
plt.figure(figsize=(10, 6))
plt.plot(range(len(y_test)), y_test, color='blue', label='Actual Prices')
plt.plot(range(len(y_test)), y_pred, color='red', label='Predicted Prices')
plt.title('Actual vs. Predicted Flight Prices (KNN)')
plt.xlabel('Data Points')
plt.ylabel('Prices')
plt.legend()
plt.show()
```



Interpretting the model

```
pip install numpy pandas scikit-learn shap
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.2.2)
Requirement already satisfied: shap in /usr/local/lib/python3.10/dist-packages (0.45.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.4.0)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.66.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (24.0)
Requirement already satisfied: slicer==0.0.7 in /usr/local/lib/python3.10/dist-packages (from shap) (0.0.7)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.41.1)
```

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import shap
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Initialize KNN regressor
knn_regressor = KNeighborsRegressor(n_neighbors=5)

# Train the model on the PCA-transformed training data
knn_regressor.fit(X_train_pca, y_train)

# Predict on the PCA-transformed test data
y_pred = knn_regressor.predict(X_test_pca)
```

```
display("Xtrain: ",X_train_pca.shape)
display("ytrain: ",y_train.shape)
display("ypred",y_pred.shape)
display("xtest",X_test_pca.shape)
```