OBJECTIVE

The primary objective of this project was to build a machine learning model capable of accurately predicting flight prices. By analysing various factors such as airline, flight class, duration, source, destination, number of stops, and time to departure, the model aims to assist users in making informed travel decisions.

Specific Goals:

- Pre-process and clean raw flight data to make it ready for analysis and modelling.
- Perform exploratory data analysis (EDA) to uncover patterns and insights.
- Build and evaluate multiple machine learning models to select the most effective one for predicting flight prices.
- Ensure scalability and reliability of the system by leveraging pipelines and reusable components.
- Deploy a trained model that can predict prices for unseen flight data.

PROBLEM STATEMENT

Flight prices are highly dynamic, varying due to multiple factors like time of booking, number of stops, and flight duration. Travelers often struggle to predict flight prices, which leads to either overpayment or delayed bookings.

Key Challenges Identified:

1. Cleaning and standardizing inconsistent data formats, such as duration and stop details.
2. Handling categorical and numerical features for effective pre-processing.
3. Selecting the best model from a range of algorithms based on metrics like mean squared error (MSE) and R-squared values.
4. Managing multivariate relationships between features and their collective impact on flight price.

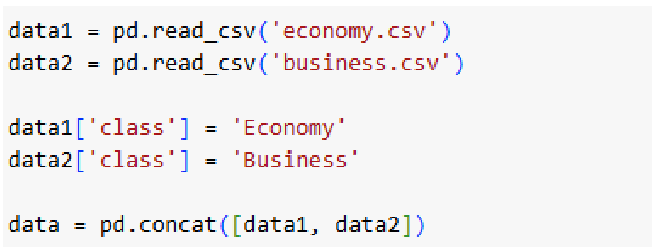
Expected Outcome:
A trained machine learning model capable of providing accurate price predictions based on user-provided flight details.

METHODOLOGIES USED

This project followed a structured approach involving data cleaning, exploratory analysis, and machine learning model development. Here are the steps in detail:

3.1 Data Collection:

- Two datasets were loaded: economy.csv and business.csv.
- A new column, class, was added to differentiate between economy and business class data. Both datasets were concatenated into a single DataFrame for analysis.



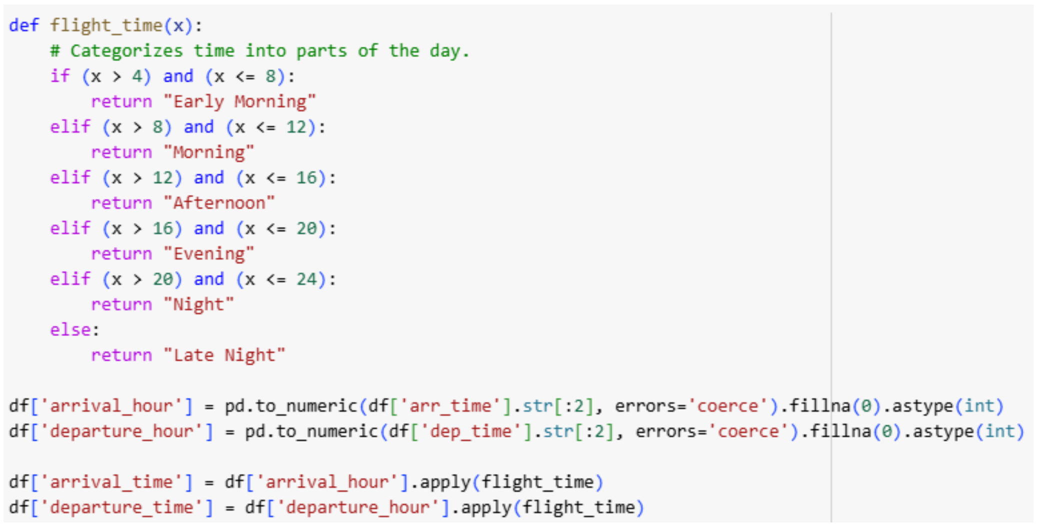
```
data1 = pd.read_csv('economy.csv')
data2 = pd.read_csv('business.csv')

data1['class'] = 'Economy'
data2['class'] = 'Business'

data = pd.concat([data1, data2])
```

3.2 Data Cleaning:

Data cleaning included multiple transformations to prepare the dataset for modeling:

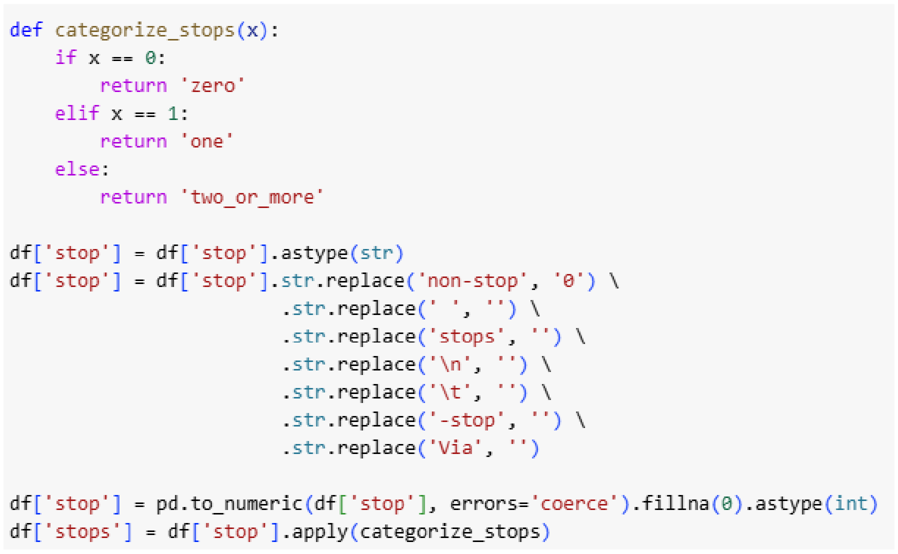
- **Time Categorization:**

The arrival_time and departure_time were categorized into segments like "Morning," "Evening," and "Night" based on extracted hours.



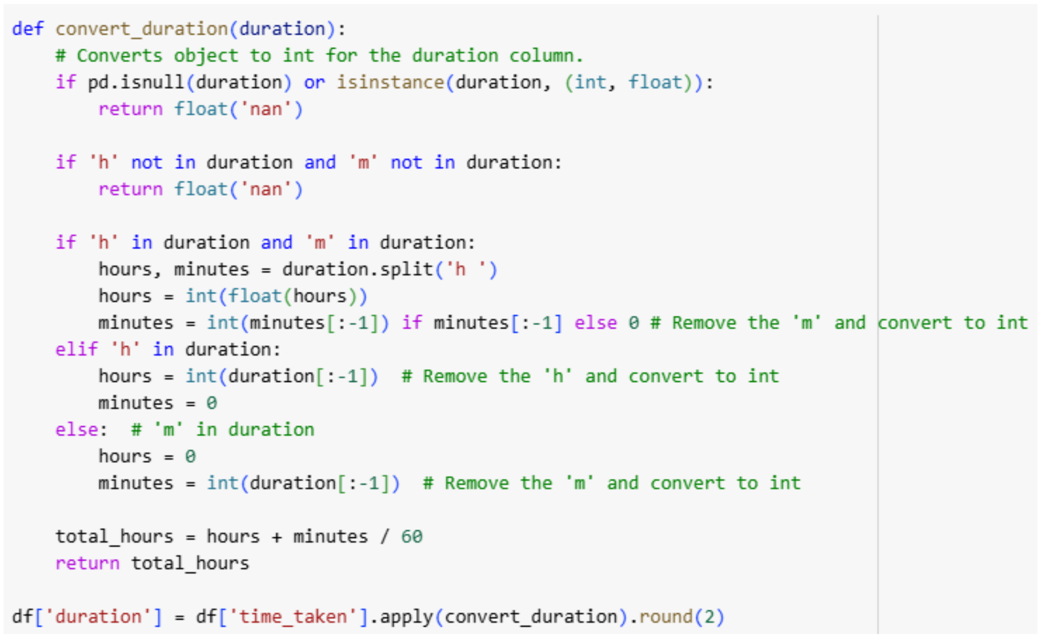
```python
def flight_time(x):
    # Categorizes time into parts of the day.
    if (x > 4) and (x <= 8):
        return "Early Morning"
    elif (x > 8) and (x <= 12):
        return "Morning"
    elif (x > 12) and (x <= 16):
        return "Afternoon"
    elif (x > 16) and (x <= 20):
        return "Evening"
    elif (x > 20) and (x <= 24):
        return "Night"
    else:
        return "Late Night"

df['arrival_hour'] = pd.to_numeric(df['arr_time'].str[:2], errors='coerce').fillna(0).astype(int)
df['departure_hour'] = pd.to_numeric(df['dep_time'].str[:2], errors='coerce').fillna(0).astype(int)

df['arrival_time'] = df['arrival_hour'].apply(flight_time)
df['departure_time'] = df['departure_hour'].apply(flight_time)
```

- **Stop Categorization:**
  The number of stops was cleaned and categorized into zero, one, and two_or_more to simplify the feature.



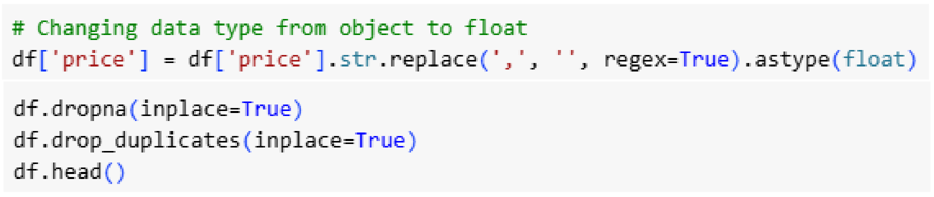
```python
def categorize_stops(x):
    if x == 0:
        return 'zero'
    elif x == 1:
        return 'one'
    else:
        return 'two_or_more'

df['stop'] = df['stop'].astype(str)
df['stop'] = df['stop'].str.replace('non-stop', '0') \
                        .str.replace(' ', '') \
                        .str.replace('stops', '') \
                        .str.replace('\n', '') \
                        .str.replace('\t', '') \
                        .str.replace('-stop', '') \
                        .str.replace('Via', '')

df['stop'] = pd.to_numeric(df['stop'], errors='coerce').fillna(0).astype(int)
df['stops'] = df['stop'].apply(categorize_stops)
```

- **Duration Conversion:**
  Durations were standardized into hours to maintain consistency. For example, "2h 30m" was converted to 2.5 hours.



```python
def convert_duration(duration):
    # Converts object to int for the duration column.
    if pd.isnull(duration) or isinstance(duration, (int, float)):
        return float('nan')

    if 'h' not in duration and 'm' not in duration:
        return float('nan')

    if 'h' in duration and 'm' in duration:
        hours, minutes = duration.split('h ')
        hours = int(float(hours))
        minutes = int(minutes[:-1]) if minutes[:-1] else 0 # Remove the 'm' and convert to int
    elif 'h' in duration:
        hours = int(duration[:-1])  # Remove the 'h' and convert to int
        minutes = 0
    else:  # 'm' in duration
        hours = 0
        minutes = int(duration[:-1])  # Remove the 'm' and convert to int

    total_hours = hours + minutes / 60
    return total_hours

df['duration'] = df['time_taken'].apply(convert_duration).round(2)
```

- **Price Cleaning:**
  The price column was converted from strings with commas to numeric data for analysis.
- **Missing and Duplicated Values:**
  All rows with missing or duplicate values were removed to ensure data integrity.



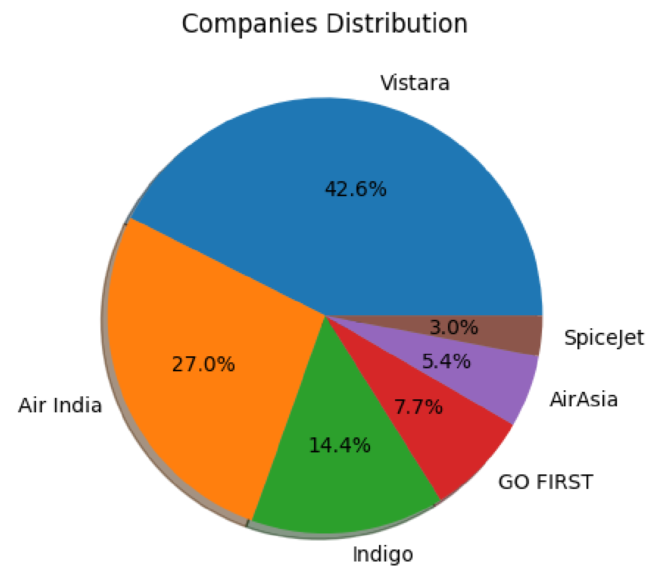
```python
# Changing data type from object to float
df['price'] = df['price'].str.replace(',', '', regex=True).astype(float)
```

```python
df.dropna(inplace=True)
df.drop_duplicates(inplace=True)
df.head()
```

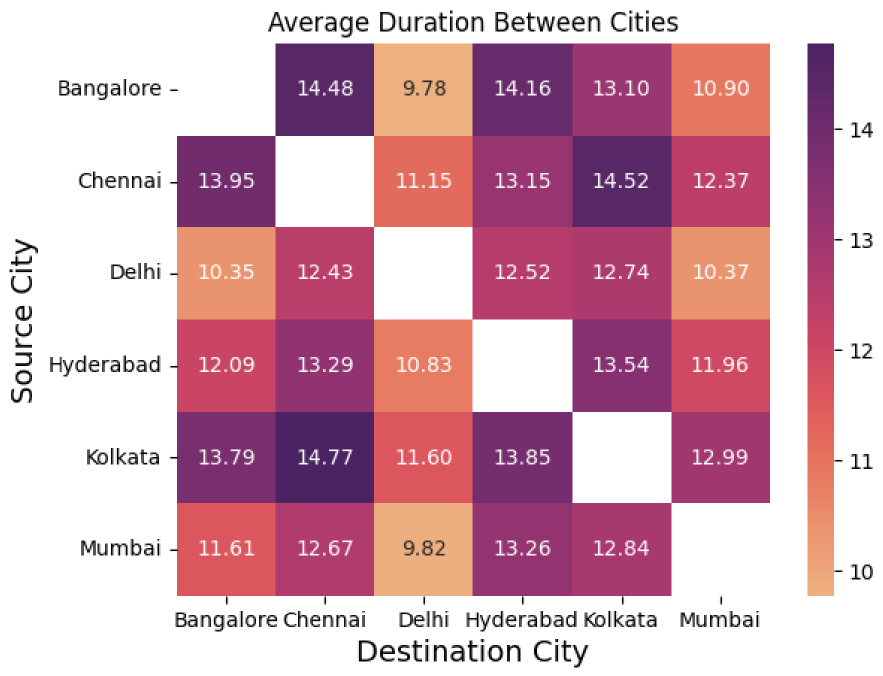
3.3 Exploratory Data Analysis (EDA):

EDA provided insights into data patterns and distributions:

- **Univariate Analysis:**
  Visualization of individual feature distributions, such as the proportion of flights by airline and average prices by flight class.
  Example: A pie chart showcasing airline distribution.

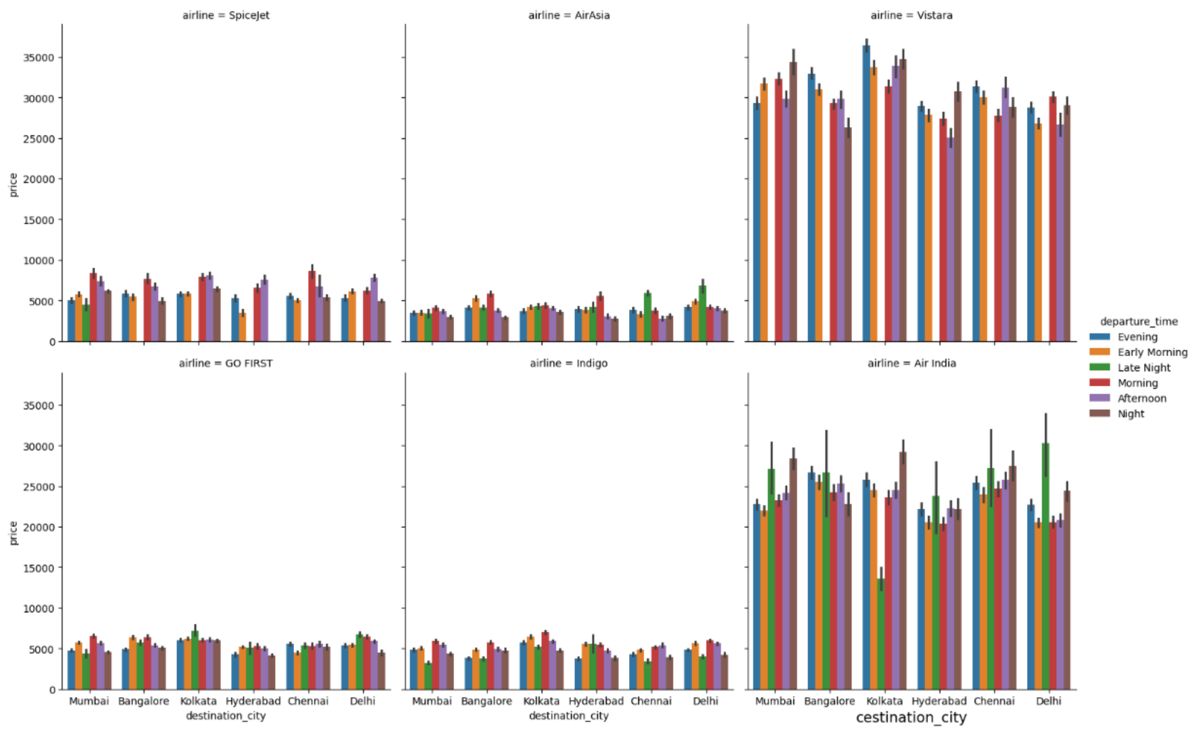


- **Bivariate Analysis:**
  Relationships between two features, such as price vs. duration and departure time vs. average prices.
  Example: Heatmaps of average flight durations and prices between source and destination cities.



Source City \ Destination City	Bangalore	Chennai	Delhi	Hyderabad	Kolkata	Mumbai
Bangalore		14.48	9.78	14.16	13.10	10.90
Chennai	13.95		11.15	13.15	14.52	12.37
Delhi	10.35	12.43		12.52	12.74	10.37
Hyderabad	12.09	13.29	10.83		13.54	11.96
Kolkata	13.79	14.77	11.60	13.85		12.99
Mumbai	11.61	12.67	9.82	13.26	12.84	

- **Multivariate Analysis:**
  Combined relationships of three or more features, such as average price trends across airlines, departure times, and destinations.
  Example: A bar chart grouping prices by departure times for each airline.



3.4 Feature Engineering:

- Categorical features were one-hot encoder, while numerical features were scaled using StandardScaler.

3.5 Feature Selection:

To avoid overfitting and improve efficiency, only the most relevant features were selected for model training. Feature importance was determined using techniques such as:

- **Correlation Analysis:**
  A heatmap was used to identify highly correlated features. Features with low correlation to the target (price) or high correlation among themselves were dropped.

- **Model-Based Selection:**
  A RandomForestRegressor was used for feature importance extraction. Features contributing significantly to model accuracy were retained.

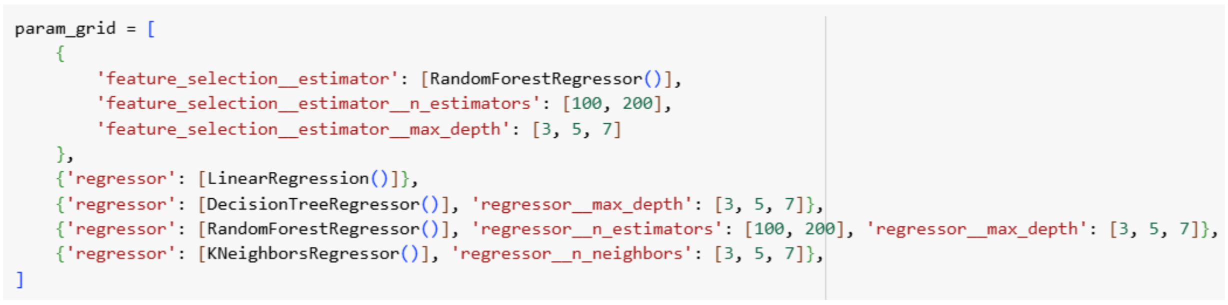
Final Features Retained:

Airline, Duration, Number of Stops, Flight Class, Source, Destination, Days Left to Departure

3.6 Machine Learning Models:

Various regression algorithms were implemented to predict flight prices. These included:

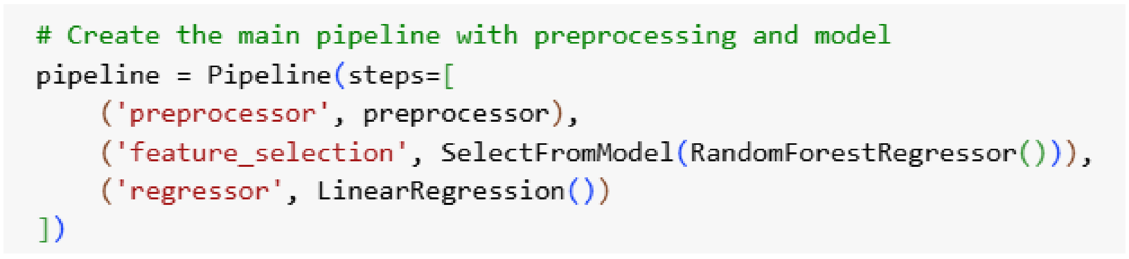
- **Linear Regression**
- **Decision Tree Regressor**
- **Random Forest Regressor**
- **K-Nearest Neighbors (KNN)**



```
param_grid = [
    {
        'feature_selection__estimator': [RandomForestRegressor()],
        'feature_selection__estimator__n_estimators': [100, 200],
        'feature_selection__estimator__max_depth': [3, 5, 7]
    },
    {'regressor': [LinearRegression()]},
    {'regressor': [DecisionTreeRegressor()], 'regressor__max_depth': [3, 5, 7]},
    {'regressor': [RandomForestRegressor()], 'regressor__n_estimators': [100, 200], 'regressor__max_depth': [3, 5, 7]},
    {'regressor': [KNeighborsRegressor()], 'regressor__n_neighbors': [3, 5, 7]},
]
```

3.7 Pipeline and Hyperparameter Tuning:

A Pipeline was created to streamline preprocessing and model training. A GridSearchCV was used for hyperparameter optimization.



```
# Create the main pipeline with preprocessing and model
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('feature_selection', SelectFromModel(RandomForestRegressor())),
    ('regressor', LinearRegression())
])
```

```
grid_search = GridSearchCV(pipeline, param_grid, cv=3, scoring='neg_mean_squared_error')
grid_search.fit(xtrain, ytrain)
```

3.8 Model Evaluation

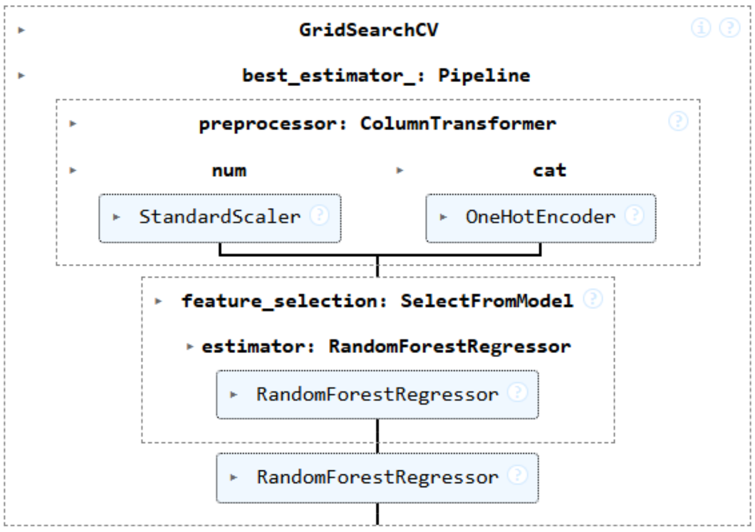
```
best_model = grid_search.best_estimator_
y_pred = best_model.predict(xtest)

mse = mean_squared_error(ytest, y_pred)
r2 = r2_score(ytest, y_pred)

print(f"Best Model: {best_model}")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

3.9 Saving the Model

```
joblib.dump(best_model, 'best_flight_price_model.joblib')
```

RESULTS

- **Best Model:** Random Forest Regressor with optimized hyperparameters.
- **Metrics Achieved:**
  - Mean Squared Error (MSE): 39798553.49
  - R-squared (R^2): 0.922
- **Insights:**
  - Prices were highly correlated with features like days_left and class.
  - Evening and Night flights had consistently higher prices.
  - Business class prices were significantly higher than economy prices, regardless of other factors.



```
GridSearchCV
best_estimator_: Pipeline
preprocessor: ColumnTransformer
num                         cat
StandardScaler              OneHotEncoder
feature_selection: SelectFromModel
estimator: RandomForestRegressor
RandomForestRegressor
RandomForestRegressor
```

CHALLENGES

- Data Inconsistency:

  Cleaning data for features like duration and stops required multiple transformations.
- High Dimensionality of Categorical Features:

  One-hot encoding of airlines, cities, and times increased feature space, impacting model complexity.
- Model Selection and Overfitting:

  Decision trees tended to overfit, while simpler models like linear regression underperformed.
- Handling Multicollinearity:

  Correlations between features like duration and stops affected model performance, requiring careful feature engineering.