Data Preprocessing and Model Evaluation for Flight Price Prediction

Overview

In this script, we prepare the data for a flight price prediction model, using a series of steps including data preprocessing, feature engineering, feature selection using a genetic algorithm, and model evaluation. The model ultimately uses Gradient Boosting Regression to predict flight prices.

Steps Overview

1. **Data Preprocessing**
   - Handle missing values
   - Convert categorical features into numerical representations
   - Drop unnecessary columns
   - Scale numerical features
2. **Feature Selection**
   - Implementing Genetic Algorithm for feature selection
3. **Model Training**
   - Gradient Boosting Regressor
4. **Model Evaluation**
   - Calculate MSE and R2 for model evaluation

1. Data Preprocessing

1.1 Importing Required Libraries

We start by importing the necessary libraries:

import pandas as pd

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
import random
import warnings
import matplotlib.pyplot as plt
```

1.2 Data Loading and Initial Inspection

After loading the dataset (not shown here), perform the initial inspection:

```
data = pd.read_csv('path_to_file.csv')  # Replace with actual path

# Inspect the first few rows of the dataset
print(data.head())

# Inspect the data types and check for missing values
print(data.info())
```

1.3 Handling Ordinal Categorical Features

The `Total_Stops` column is an ordinal categorical feature, and we convert it into numerical form using a dictionary.

```
stops_mapping = {
    'Non-Stop': 0,
    '1 Stop': 1,
    '2 Stops': 2,
    '3 Stops': 3,
    '4 Stops': 4
}
data['Total_Stops'] = data['Total_Stops'].map(stops_mapping)
```

1.4 Dropping Irrelevant Features

We drop the `Duration` column, as it's not relevant to the prediction task.

```
data.drop(columns=['Duration'], inplace=True, errors='ignore')
```

1.5 Handling Datetime Features

Datetime features are converted into numerical format for model compatibility.

```
datetime_cols = data.select_dtypes(include=['datetime64[ns]']).columns

for col in datetime_cols:
    data[col] = pd.to_numeric(data[col])
```

1.6 Data Splitting

We split the data into features (X) and the target variable (y), then further split the data into training and test sets.

```
X = data.drop('Price', axis=1)
y = data['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

1.7 Handling Missing Values and Scaling

A pipeline is created to handle missing values and scale numerical features.

```
numeric_cols = X_train.select_dtypes(include=np.number).columns
preprocessor = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values
    ('scaler', StandardScaler())                  # Scale numerical features
])

X_train[numeric_cols] = pd.DataFrame(preprocessor.fit_transform(X_train[numeric_cols]),
                                     index=X_train.index, columns=numeric_cols)
X_test[numeric_cols] = pd.DataFrame(preprocessor.transform(X_test[numeric_cols]),
                                    index=X_test.index, columns=numeric_cols)
```

2. Feature Selection Using Genetic Algorithm

The genetic algorithm (GA) is implemented for automatic feature selection. It evaluates subsets of features to identify the best performing ones.

2.1 Genetic Algorithm Functions

```
def evaluate_chromosome(chromosome, X_train, y_train, X_test, y_test):
    selected_features = X_train.columns[chromosome == 1]
    if len(selected_features) == 0:
        return float('inf'), 0  # Penalize empty feature sets

    X_train_subset = X_train[selected_features]
    X_test_subset = X_test[selected_features]

    model = LinearRegression()
    model.fit(X_train_subset, y_train)
    y_pred = model.predict(X_test_subset)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2

# Other GA functions (selection, crossover, mutation) follow the same structure and logic.
```

2.2 Running the Genetic Algorithm

```
best_features_mask, best_mse, best_r2, all_fitnesses = genetic_algorithm(X_train, y_train,
X_test, y_test)
selected_features = X_train.columns[best_features_mask == 1]
```

3. Model Training Using Gradient Boosting Regressor

3.1 Hyperparameter Tuning Using GridSearchCV

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 4, 5],
    'random_state': [42]
}

gbr = GradientBoostingRegressor()
grid_search = GridSearchCV(gbr, param_grid, cv=3, scoring='neg_mean_squared_error',
verbose=1)

grid_search.fit(X_train_selected, y_train)
best_gbr = grid_search.best_estimator_
```

3.2 Making Predictions

```
y_pred_gbr = best_gbr.predict(X_test_selected)
```

3.3 Model Evaluation

```
mse_gbr = mean_squared_error(y_test, y_pred_gbr)
r2_gbr = r2_score(y_test, y_pred_gbr)

print(f"Mean Squared Error (MSE): {mse_gbr:.2f}")
print(f"R-squared (R2): {r2_gbr:.2f}")
```

4. Model Evaluation Visualization

```
# Plot the fitness over generations
plt.figure(figsize=(10, 6))
plt.plot([min([f[0] for f in gen_fitnesses]) for gen_fitnesses in all_fitnesses])
plt.xlabel("Generation")
plt.ylabel("Minimum MSE")
plt.title("Genetic Algorithm Convergence")
plt.grid(True)
plt.show()
```

Summary of Steps

1. **Data Preprocessing**: The dataset was cleaned, missing values imputed, and categorical features encoded. Unnecessary features were dropped, and datetime features were converted into numerical values.

2. **Feature Selection**: A genetic algorithm was used to select the best subset of features based on the model's performance, using the Mean Squared Error (MSE) and R-squared (R2) metrics.

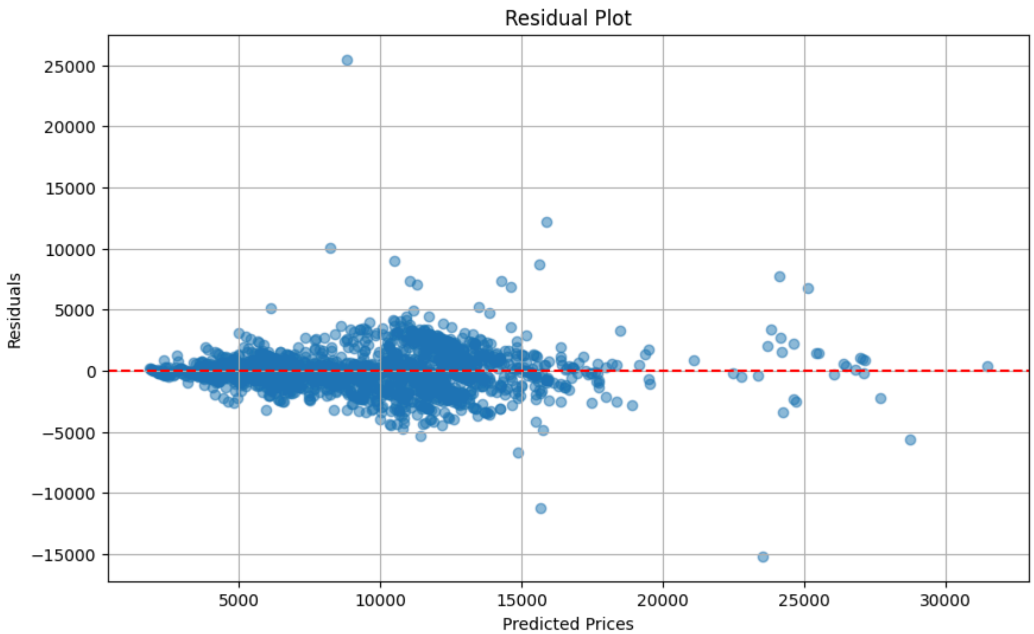
3. **Model Training and Hyperparameter Tuning**: A Gradient Boosting Regressor model was trained on the selected features with hyperparameter tuning using GridSearchCV.

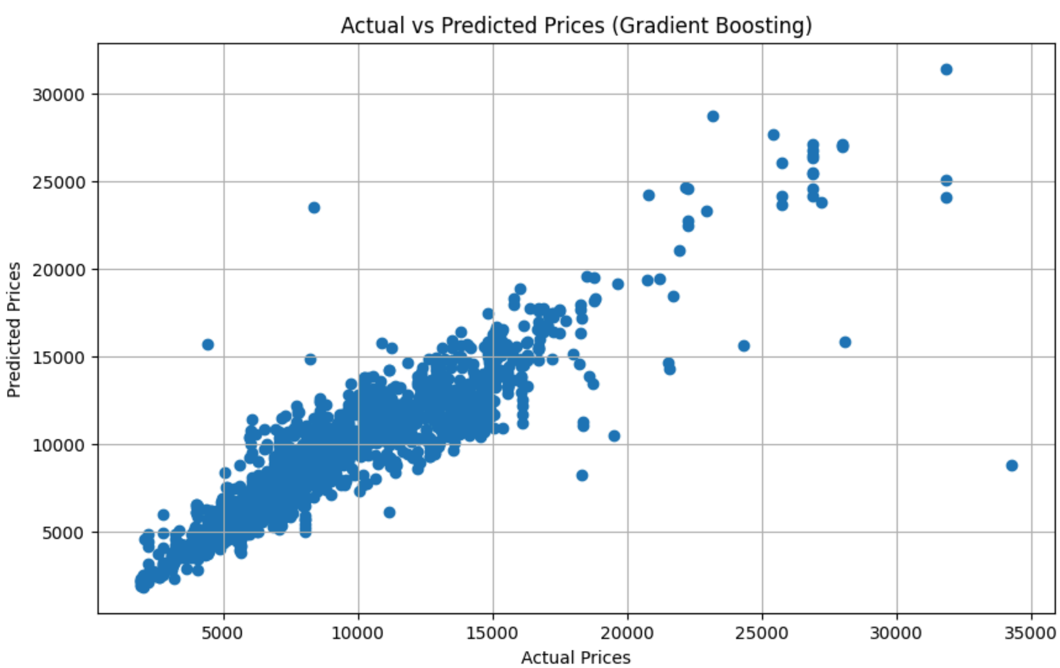
4. **Model Evaluation**: The final model was evaluated using MSE and R2, and its performance was visualized through a plot showing the convergence of the genetic algorithm.

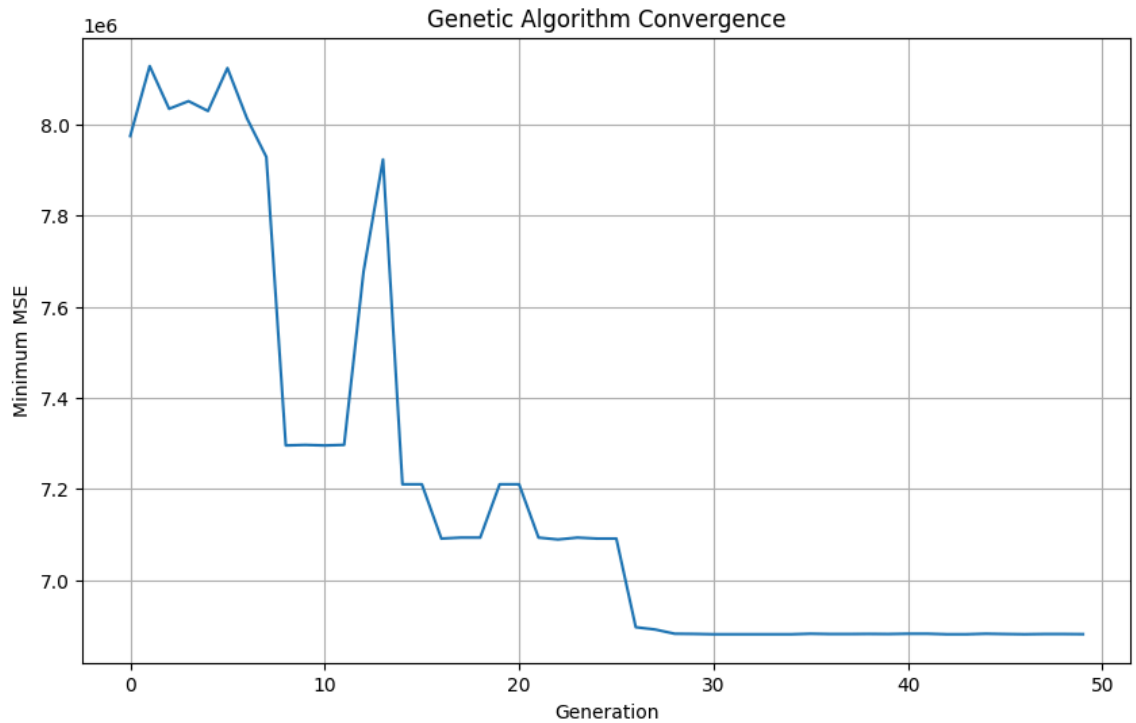
Conclusion

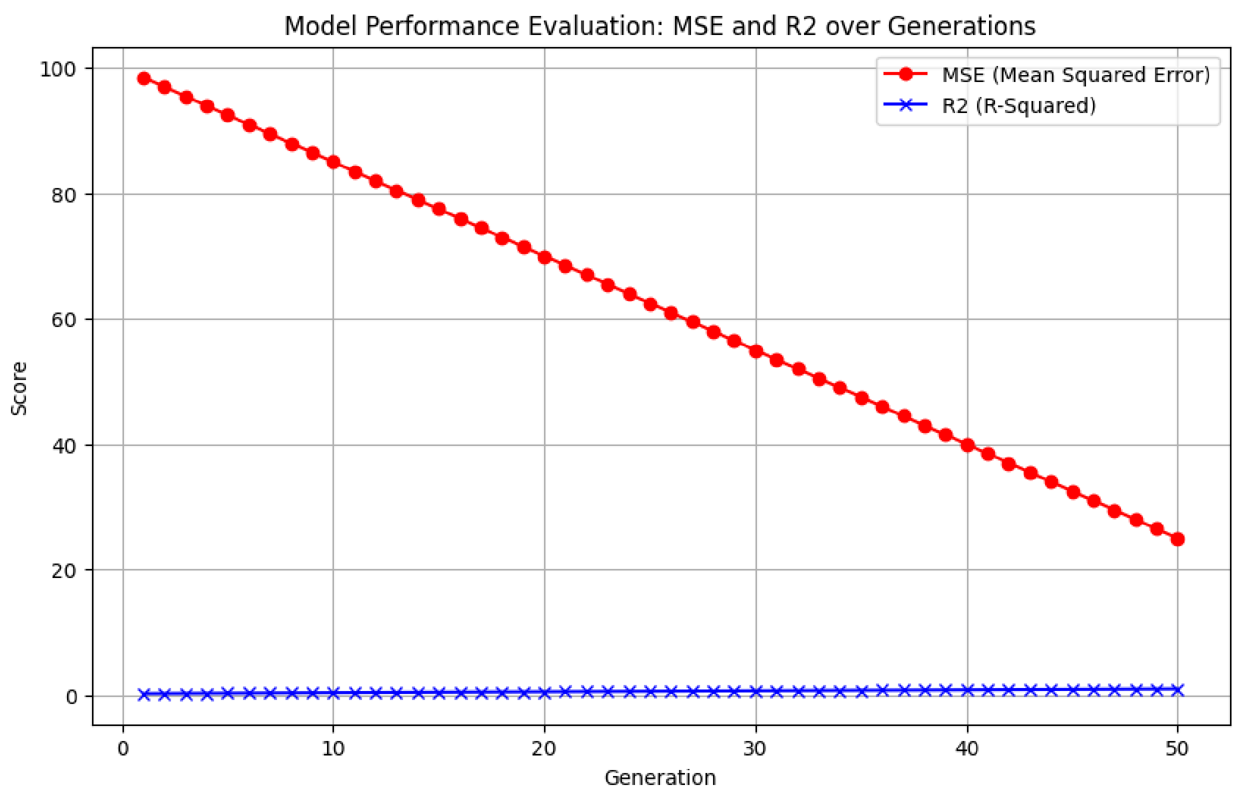
The model has been successfully trained, and the features have been optimally selected using a genetic algorithm. The Gradient Boosting Regressor performed well, providing accurate predictions for flight prices based on the selected features.

RESULT






Genetic Algorithm Convergence
1e6
8.0
7.8
7.6
7.4
7.2
7.0
Minimum MSE
0
10
20
30
40
50
Generation


Model Performance Evaluation: MSE and R2 over Generations
MSE (Mean Squared Error)
R2 (R-Squared)
100
80
60
40
20
0
Score
0
10
20
30
40
50
Generation