Flight Ticket Price Prediction: Project Report

Authors : Matheo NGUYEN, Rayan BELABBAS, Pierre CHARTIER

1. Introduction

Understanding the factors that influence flight ticket prices can be useful for travelers, travel agencies, and airline companies alike. This project aims to build a machine learning model that can predict flight prices based on various features such as the airline, route, travel class, and time until departure.
The goal is not only to accurately predict prices but also to understand the impact of different variables on these prices.

---

2. Problem Definition

The primary objective is to predict the price of a flight ticket based on a set of structured features.

This is a **supervised regression problem**, where the target variable is the price of the flight (in Indian Rupees). The model should be able to:

- Provide accurate price estimations.
- Identify key features that influence ticket prices.

---

3. Dataset Description (Kaggle)

Source: Kaggle - Flight Price Prediction
Overview:

- Over 300,000 rows of domestic flight records in India.
- Collected over 50 consecutive days by scraping the EaseMyTrip portal.

Features:

- `airline` : Name of the airline
- `flight` : Flight number
- `source_city`, `destination_city` : Cities of departure and arrival
- `departure_time`, `arrival_time` : Scheduled times of travel
- `stops` : Number of layovers
- `class` : Economy or Business
- `duration` : Travel duration
- `days_left` : Number of days before departure
- `price` : Final ticket price in INR (**target variable**)

---

4. Data Preprocessing

Several preprocessing steps were taken to prepare the dataset:

1. **Whitespace Stripping:** All categorical variables had their leading/trailing whitespaces removed.
2. **Data Type Casting:** Ensured all categorical features were treated as `str` type.
3. **Missing Values:** Rows with missing values were dropped.
4. **Index Resetting:** After dropping rows, the DataFrame index was reset to maintain consistency.

```
# Example: Clean categorical columns
categorical_cols = ["airline", "flight", "source_city", "departure_time",
"stops", "arrival_time", "destination_city", "class"] df[categorical_cols] =
df[categorical_cols].apply(lambda col:
col.str.strip()).astype(str)
```

---

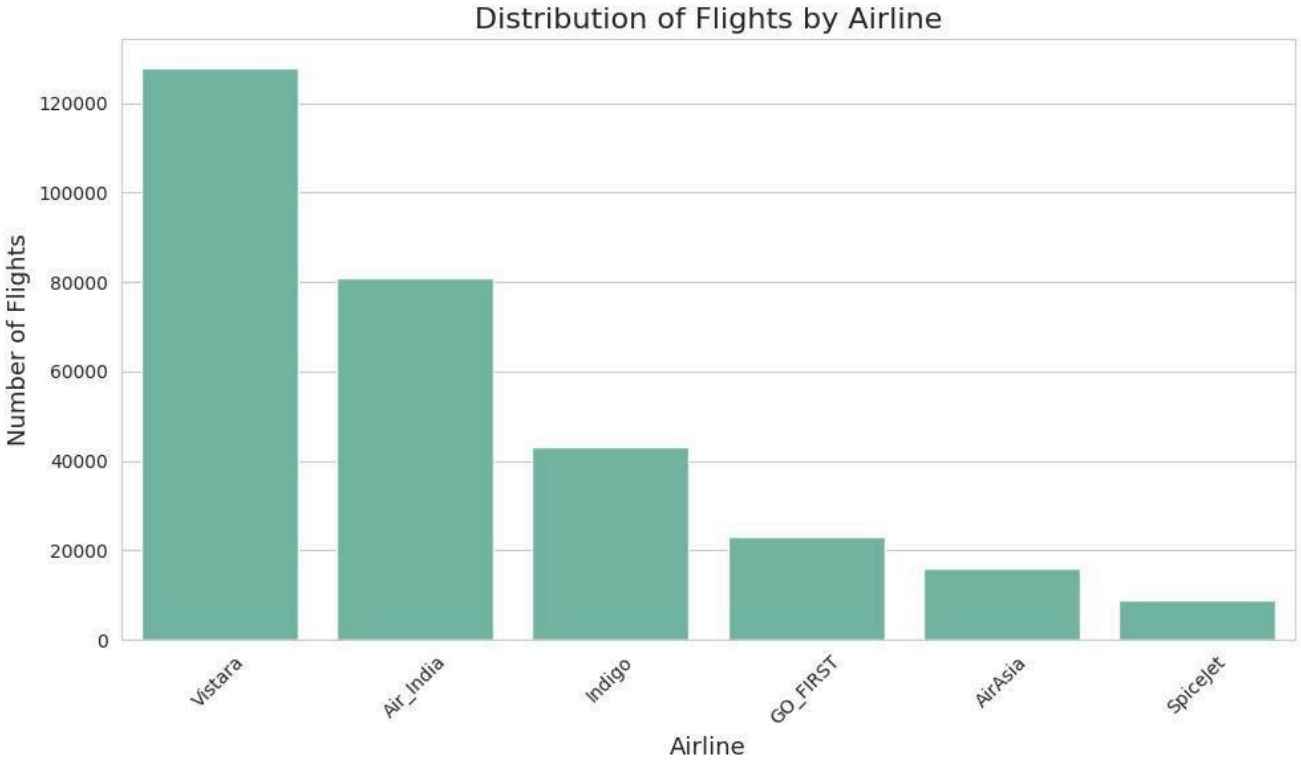
5. Data Visualization

Key Insights:

- **Airline Distribution:** IndiGo and Air India have the highest number of flights.
  - It's important to note that this could bias our model.
- **Seat Class Distribution:** ~80% Economy, ~20% Business class.
  - This is to be expected, but it is important to check.
- **Price Variation by Airline:** Full-service airlines (like Air India) tend to have higher average prices.
- **Number of Stops vs Price:** Direct flights are generally more expensive than connecting ones.

Visualizations:

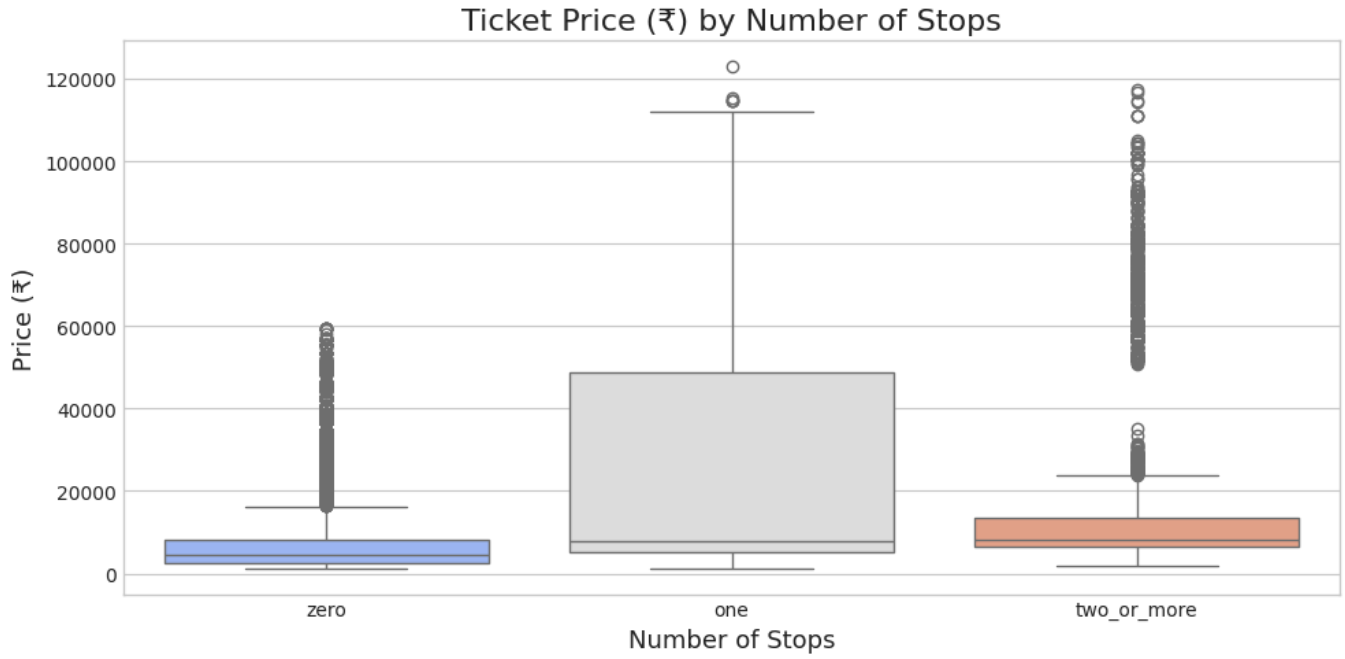
Count plots for categorical distribution.



- **Vistara** leads with ~128,000 flights, followed by **Air India** (~80,000) and **Indigo** (~43,000). Their network breadth drives both market presence and price diversity.
- **SpiceJet** has the **fewest flights** (~9,000), indicating a smaller market share in the dataset.

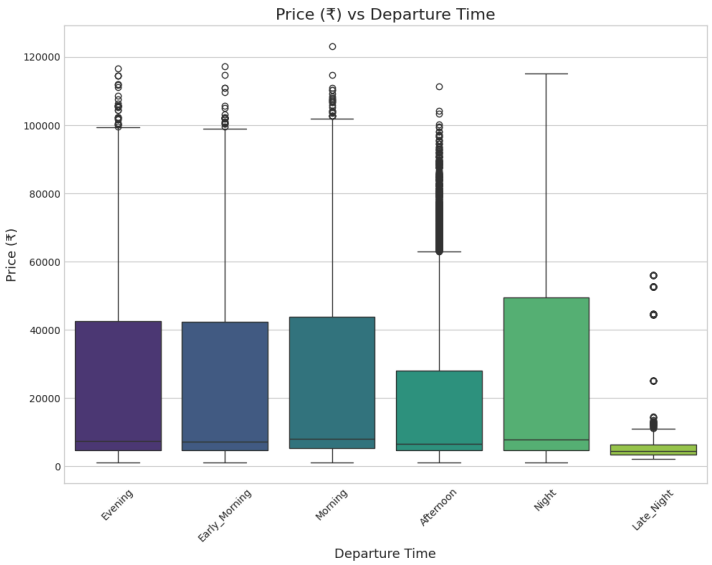
Boxplots for price comparisons.

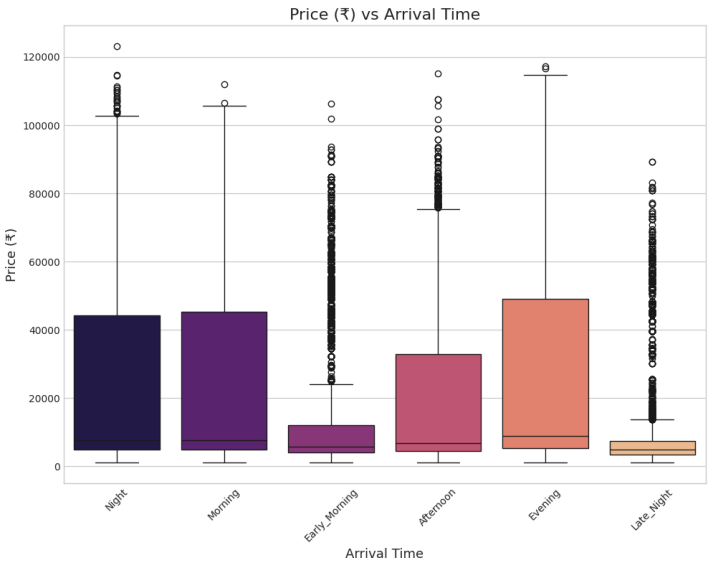
- **Vistara** and **Air India** exhibits the **widest price range**, with multiple high-end outliers. This reflects its premium service offerings.
- **AirAsia**, **GO FIRST**, **Air Asia** and **SpiceJet**, positioned as low-cost carriers, show **tight price distributions**, with minimal high-end outliers.



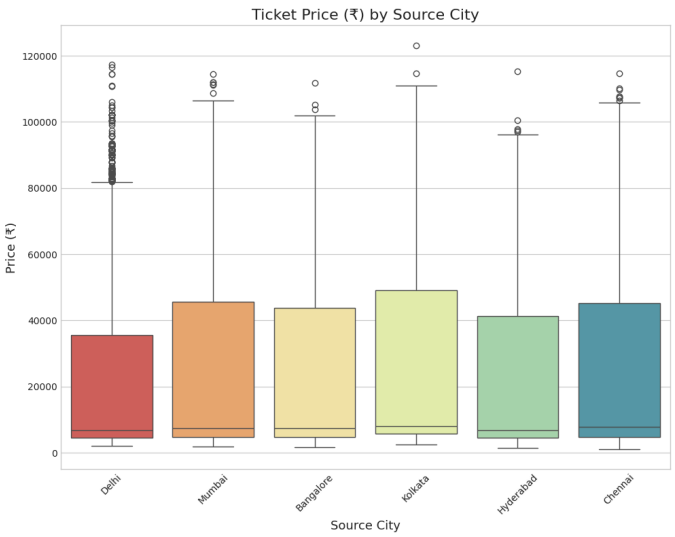
- Surprisingly, flights with **one stop** have a **higher median** and greater variability than direct flights.

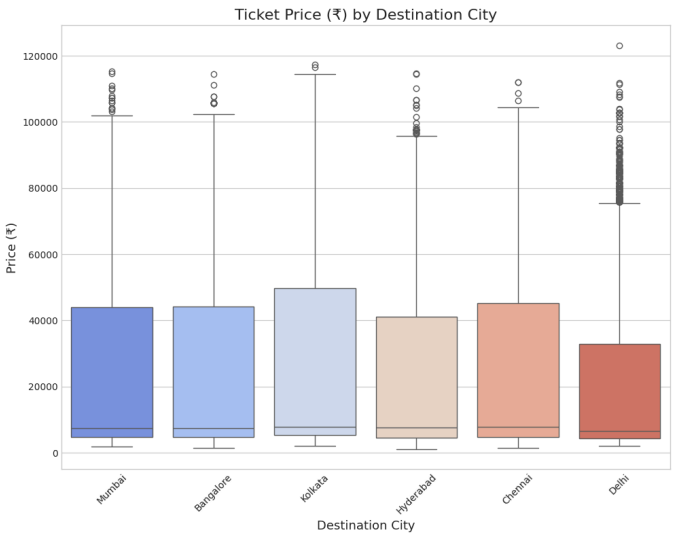
- **Direct flights** remain the **cheapest on average**, appealing to budget-conscious travelers.
- Flights with **two or more stops** trend towards **higher costs**, likely due to complex routings, increased operational costs, and fewer low-cost options on multi-stop legs.





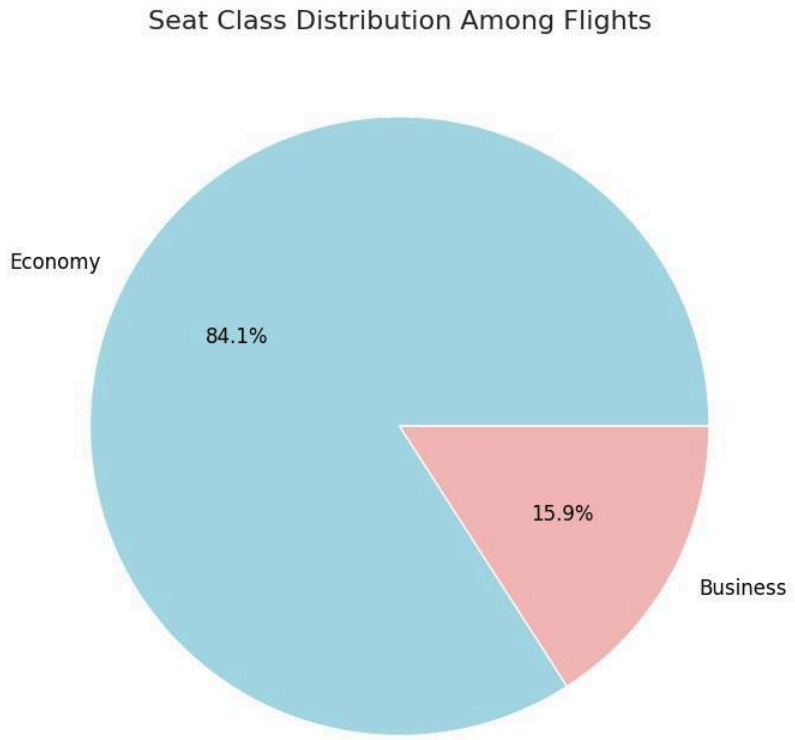
- Demand peaks for **Morning** and **Early Morning** slots, pushing prices up. Business travelers often prefer these slots maybe for productivity.
- **Afternoon** departures offer the **lowest prices**.
- **Night** and **Late Night** slots show moderate pricing, influenced by reduced convenience.





- **Evening** and **Night** arrivals command **higher fares**, as passengers favor arriving before the day ends or after work hours.
- **Early Morning** arrivals have **lowest prices**, corresponding with lower demand for very early arrivals and potential need for additional overnight stays.

Pie charts for class distribution



- **Economy class** represents **84%** of all seat bookings, highlighting its dominance in the domestic market.
- **Business class** accounts for **16%**, indicating a smaller but significant premium segment.
- This skew underscores the importance of tailoring pricing models predominantly for the economy segment, while capturing high-margin opportunities in business class.

6. Model Evaluation and Visualization Report

This section presents a complete pipeline for training, evaluating, and visualizing machine learning models for predicting flight ticket prices.

6.1. Data Encoding

To prepare the dataset, all categorical features are label-encoded:

```
df_bk = df.copy()
    if df[col].dtype == 'object':
        df[col] = le.fit_transform(df[col])
```

A backup of the original DataFrame is saved as `df_bk` to preserve untransformed values for future analysis. Then, each object-type column is encoded into numerical form using `LabelEncoder` .

6.2. Feature Selection and Train-Test Split

We have defined our target variable as the price and the other columns as features.

```
X = df.drop('price', axis=1)
y = df['price']
```

Then the dataset is split:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

70% of the data is used for training, and 30% is kept for testing.

6.3. Feature Scaling

Features are scaled to a 0-1 range using MinMaxScaler:

```
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=X.columns)
X_test = pd.DataFrame(scaler.transform(X_test), columns=X.columns)
```

This transformation is crucial for ensuring that all numerical features are on the same scale. We use MinMaxScaler to adjust each feature individually such that the minimum becomes 0 and the maximum becomes 1.

6.4. Model Initialization

Three regression models are initialized:

```
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Extra Trees': ExtraTreesRegressor(random_state=42)
}
```

These models were chosen to offer a balanced comparison across different regression algorithms:

- **Linear Regression**: A simple, interpretable baseline model.
- **Random Forest Regressor**: It is robust to outliers and reduces overfitting by averaging multiple decision trees.
- **Extra Trees Regressor**: Similar to Random Forest, but introduces more randomness in the split selection, which can improve generalization and training speed.

6.5. Model Evaluation Function

A custom function evaluates each model's performance:

```
def evaluate_model(name, model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae = mean_absolute_error(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    rmsle = np.log1p(rmse)
    mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100
    adj_r2 = 1 - (1 - r2) * (len(y_test) - 1) / (len(y_test) -
X_test.shape[1] - 1)

    return {
        'Model': name,
        'MAE': round(mae, 2),

        'RMSE': round(rmse, 2),
        'MAPE': round(mape, 2),

        'R2': round(r2, 4),
        'Adj_R2': round(adj_r2, 4)
    }
```

Metrics computed include:

- **MAE**: Mean Absolute Error
- **RMSE**: Root Mean Squared Error
- **MAPE**: Mean Absolute Percentage Error
- **R2**: Coefficient of Determination
- **Adjusted R2: R2** adjusted for the number of features

6.6. Training and Comparing Models

All models are evaluated using a loop:

```
results = []
for name, model in models.items():
    results.append(evaluate_model(name, model, X_train, X_test, y_train,
y_test))

results_df = pd.DataFrame(results).sort_values(by='Adj_R2',
ascending=False).reset_index(drop=True)
```

The results are stored in a DataFrame, sorted by adjusted R2 score.
We choose the R^2 score (coefficient of determination) because it measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides a normalized metric that indicates model performance so :

- An R^2 score of **1.0** means perfect prediction.
- An R^2 score of **0.0** means the model predicts no better than the mean.
- A **higher R^2** indicates a better fit.

---

7. Best Model Prediction

Finally, the full performance comparison table is printed:

Model	MAE	RMSE	MAPE	R2	Adj_R2
Random Forest	804.99	2192.93	5.40	0.9907	0.9907
Extra Trees	830.76	2313.55	5.55	0.9896	0.9896
Linear Regression	4623.77	7003.56	43.67	0.9047	0.9047

Conclusion: Random Forest as the Best Model

Based on the evaluation metrics, Random Forest Regressor stands out as the most effective model:

- It has the lowest average prediction error (MAE).
- It achieves the smallest root mean squared error (RMSE), indicating high precision.
- It shows the lowest percentage error (MAPE), demonstrating consistent relative accuracy.
- It explains the highest proportion of variance in the target variable (R^2 score).
- It maintains strong explanatory power even after adjusting for feature count (Adjusted R^2).
- Compared to Extra Trees, it is slightly more accurate.
- Compared to Linear Regression, it performs significantly better in every metric.

Then we choose **Random Forest** :

```
best_model = models['Extra Trees']
y_pred = best_model.predict(X_test)
```

Predictions are compared against actual prices:

```
result = pd.DataFrame({
    'Price_actual': y_test,
    'Price_pred': y_pred
}, index=y_test.index)
result = df_bk.join(result)
```

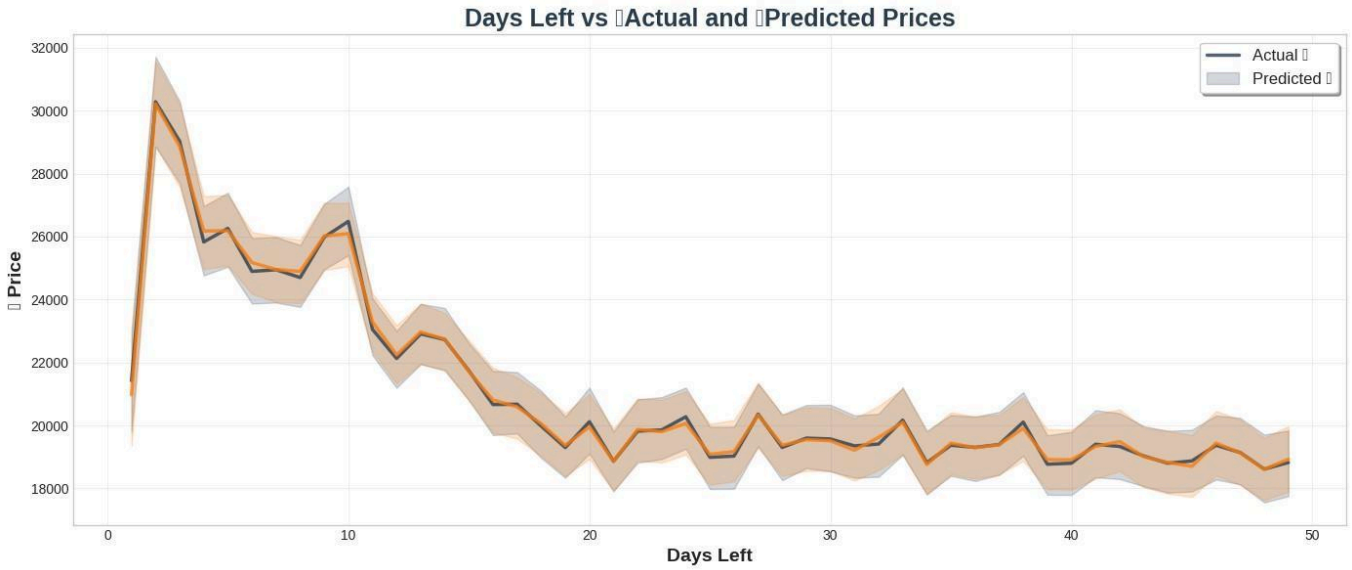
8. Results Visualization

8.1. Days Left vs Price

A line plot compares actual and predicted prices relative to days left:

```
sns.lineplot(data=result, x='days_left', y='Price_actual', color=colors['dark'], linewidth=2.5, alpha=0.8) sns.lineplot(data=result, x='days_left', y='Price_pred', color=colors['secondary'], linewidth=2.5, alpha=0.8)

plt.title('Days Left vs ₹Actual and ₹Predicted Prices', fontsize=18, fontweight='bold', color=colors['dark']) plt.xlabel('Days Left', fontsize=14, fontweight='bold') plt.ylabel('₹ Price', fontsize=14, fontweight='bold') plt.legend(['Actual ₹', 'Predicted ₹'], fontsize=12, frameon=True, fancybox=True, shadow=True) plt.grid(True, alpha=0.3) plt.tight_layout() plt.show()
```



Price trend over time: The model follows reality well, with prices varying from 50 to 20 days before departure, then increasing significantly right up to the day of departure.

The time-series view confirms that the model not only predicts individual prices accurately but also reproduces the overall seasonal pattern in price dynamics with appropriate confidence estimates.

8.2. Actual vs Predicted

A scatter plot compares actual vs predicted prices:

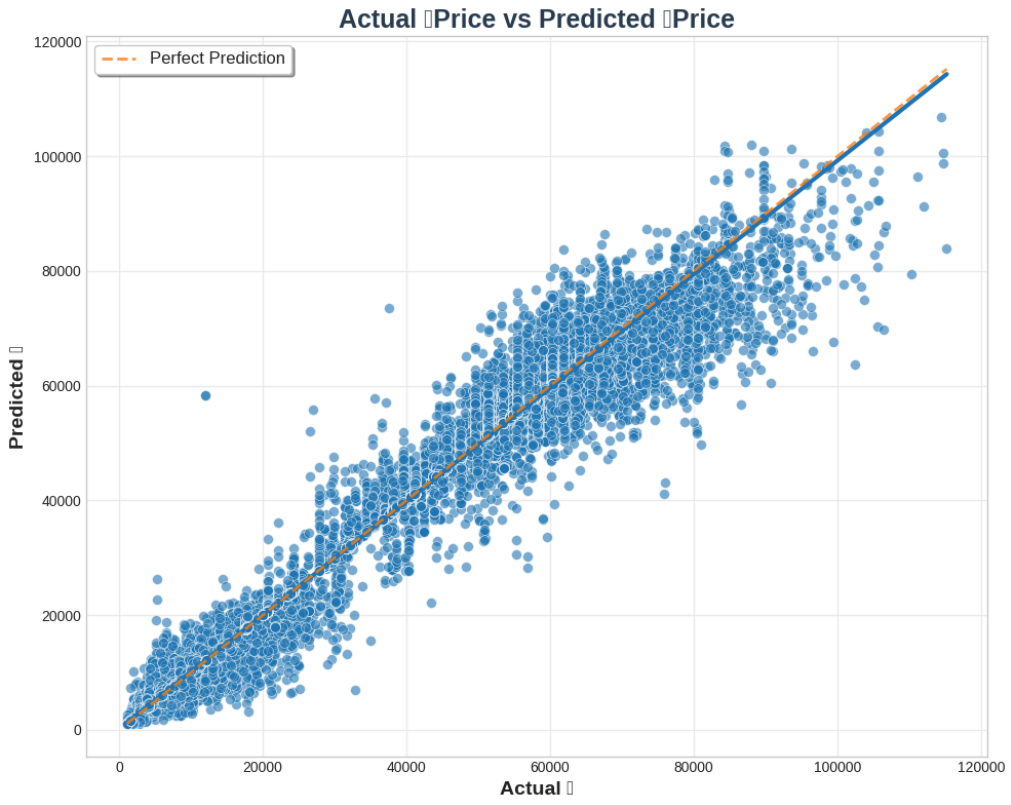
```
plt.figure(figsize=(10,8))
plt.style.use('seaborn-v0_8-whitegrid')

plt.scatter(result['Price_actual'], result['Price_pred'],
           color=colors['primary'], alpha=0.6, s=50, edgecolor='white',
linewidth=0.5)

sns.regplot(x='Price_actual', y='Price_pred', data=result,
           color=colors['primary'], scatter=False, line_kws={'linewidth':
3})

min_val = min(result['Price_actual'].min(), result['Price_pred'].min())
max_val = max(result['Price_actual'].max(), result['Price_pred'].max())
plt.plot([min_val, max_val], [min_val, max_val],
         color=colors['secondary'], linestyle='--', linewidth=2, alpha=0.8,
         label='Perfect Prediction')

plt.title('Actual ₹Price vs Predicted ₹Price',
          fontsize=18, fontweight='bold', color=colors['dark'])
plt.xlabel('Actual ₹', fontsize=14, fontweight='bold')
plt.ylabel('Predicted ₹', fontsize=14, fontweight='bold')
plt.legend(fontsize=12, frameon=True, fancybox=True, shadow=True)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



9. Responsible AI Practices

In building and evaluating our flight price prediction model, we have taken bests practices to ensure fairness, transparency, and ethical use of the system:

- **Bias and Fairness Assessment**: We reviewed the dataset to identify potential sources of bias, such as overrepresentation or underrepresentation of specific airlines, cities, or travel classes.
- **Transparency and Explainability**: Ensemble methods like Random Forest are less interpretable than linear models, feature importance analysis can still provide insights into the most influential variables.
- **Data Privacy**: The dataset used does not contain any personally identifiable information.
- **Robustness and Reliability**: Multiple models were evaluated using a diverse set of metrics to ensure that the final system is not only accurate but also robust to various input conditions and outliers.
- **Avoiding Overfitting**: Adjusted R^2 and proper train/test splitting were used to assess generalization performance and avoid overfitting on training data.
- **Reproducibility**: The code uses fixed random states (42) and standard pipelines for preprocessing and training, making results reproducible across environments and runs.
- **Limitations and Ethical Use**: This model is designed solely for educational and analytical purposes.

10. Conclusion

The flight price prediction project demonstrates how data-driven approaches can provide accurate, interpretable insights into a real-world problem. With Random Forest achieving over 90% R^2 accuracy, this system can be deployed in:

- **Travel pricing assistants**
- **Budget recommendation systems**
- **Market analysis dashboards**