Airline Flight Price Prediction - Final Project

Project Overview

This project implements a machine learning solution to predict airline flight prices based on various flight characteristics. The project includes comprehensive data analysis, preprocessing, feature engineering, and comparison of multiple regression models to identify the best performing algorithm.

Table of Contents

- Dataset Description
- Project Structure
- Installation & Requirements
- Data Processing Pipeline
- Exploratory Data Analysis
- Model Development
- Results
- Usage
- Key Insights
- Future Improvements

Dataset Description

File: `airlines_flights_data.csv`

The dataset contains **300,154 flight records** with the following features:

Original Features

Feature	Type	Description
`index`	Numeric	Record identifier (dropped during preprocessing)
`airline`	Categorical	Airline company name
`flight`	Categorical	Flight code (dropped during preprocessing)
`source_city`	Categorical	Departure city
`departure_time`	Categorical	Time period of departure
`stops`	Categorical	Number of stops (zero, one, two_or_more)
`arrival_time`	Categorical	Time period of arrival (dropped during preprocessing)
`destination_city`	Categorical	Arrival city
`class`	Categorical	Travel class (Economy/Business)
`duration`	Numeric	Flight duration in hours

Feature	Type	Description
`days_left`	Numeric	Days until departure
`price`	Numeric	**Target variable** - Ticket price

Data Characteristics

- **No missing values** in any column
- **No duplicate records**
- Price originally in INR (Indian Rupees), converted to USD (₹87.04 = $1)
- Multiple airlines: SpiceJet, AirAsia, Vistara, Air India, Indigo, GO_FIRST
- Six departure time periods: Early_Morning, Morning, Afternoon, Evening, Night, Late_Night

---

Project Structure

```
DEPI/
├── final project[1].ipynb          # Main Jupyter notebook
├── airlines_flights_data.csv       # Dataset
└── README.md                        # This documentation
```

Installation & Requirements

Required Libraries

```
pandas>=1.3.0
numpy>=1.21.0
matplotlib>=3.4.0
seaborn>=0.11.0
scikit-learn>=1.0.0
xgboost>=1.5.0
```

Installation

```
pip install pandas numpy matplotlib seaborn scikit-learn xgboost
```

Data Processing Pipeline

1. Data Loading & Initial Exploration

```
df = pd.read_csv("airlines_flights_data.csv")
df.info()
df.isna().sum()
df.duplicated().sum()
```

Findings:

- ✅ No missing values
- ✅ No duplicates
- Dataset ready for processing

2. Feature Engineering

Dropped Features

```
df.drop(["arrival_time", 'flight', 'index'], axis=1, inplace=True)
```

Rationale:

- `index`: Redundant identifier
- `flight`: Flight code doesn't contribute to price prediction
- `arrival_time`: High correlation with departure_time + duration

Price Conversion

```
df['price'] = (df['price']/87.04).round()
```

Converted prices from INR to USD for better interpretability.

3. Feature Categorization

Categorical Features (6):

- `airline`, `source_city`, `departure_time`, `destination_city`, `class`, `stops`

Numerical Features (2):

- `duration`, `days_left`

4. Outlier Treatment

Applied **IQR method** to `duration` feature:

```
q1 = df['duration'].quantile(0.25)
q3 = df['duration'].quantile(0.75)
```

```
iqr = q3 - q1
lower_bound = q1 - iqr * 1.5
upper_bound = q3 + iqr * 1.5

df['duration'] = df['duration'].clip(lower=lower_bound, upper=upper_bound)
```

5. Encoding

Label Encoding applied to all categorical features using `LabelEncoder()`.

6. Feature Scaling

StandardScaler applied to ensure all features have:

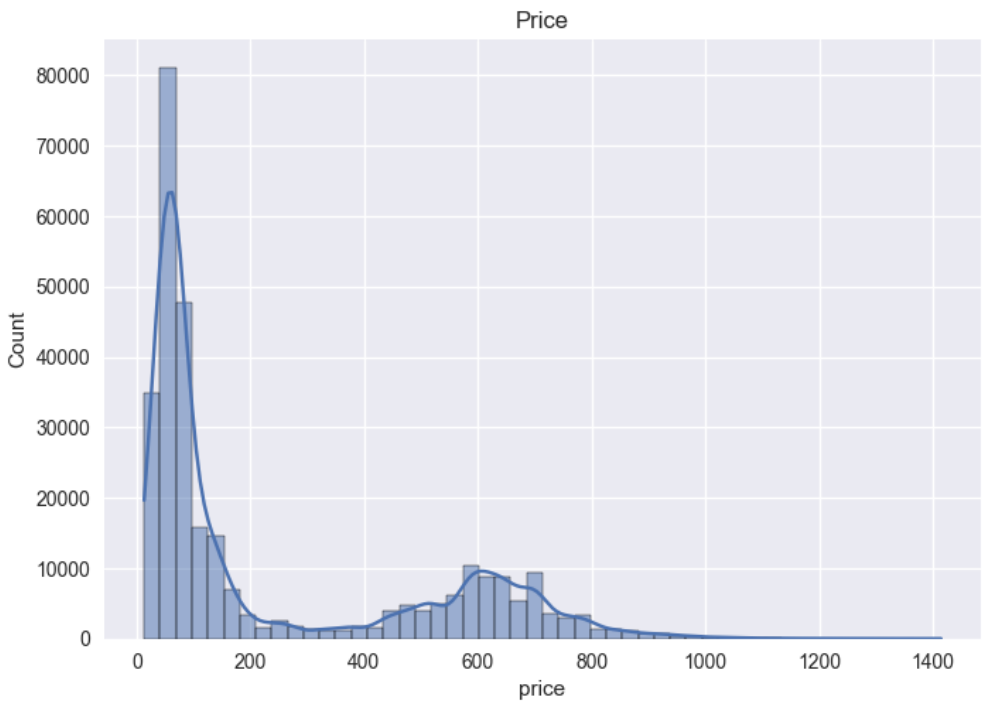
- Mean = 0
- Standard deviation = 1

```
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Exploratory Data Analysis

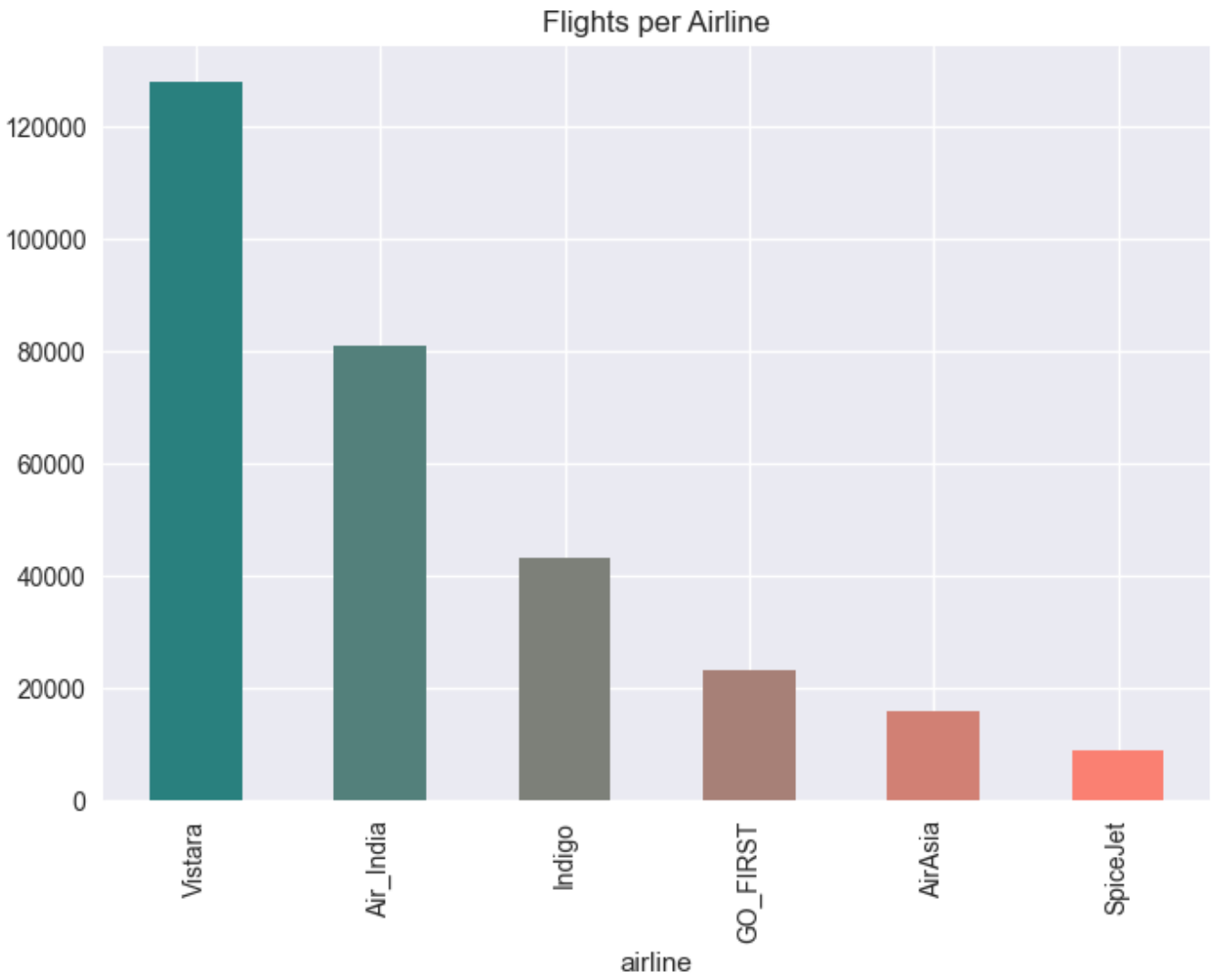
1. Price Distribution

- **Type:** Histogram with KDE
- **Finding:** Price distribution shows right-skewed pattern



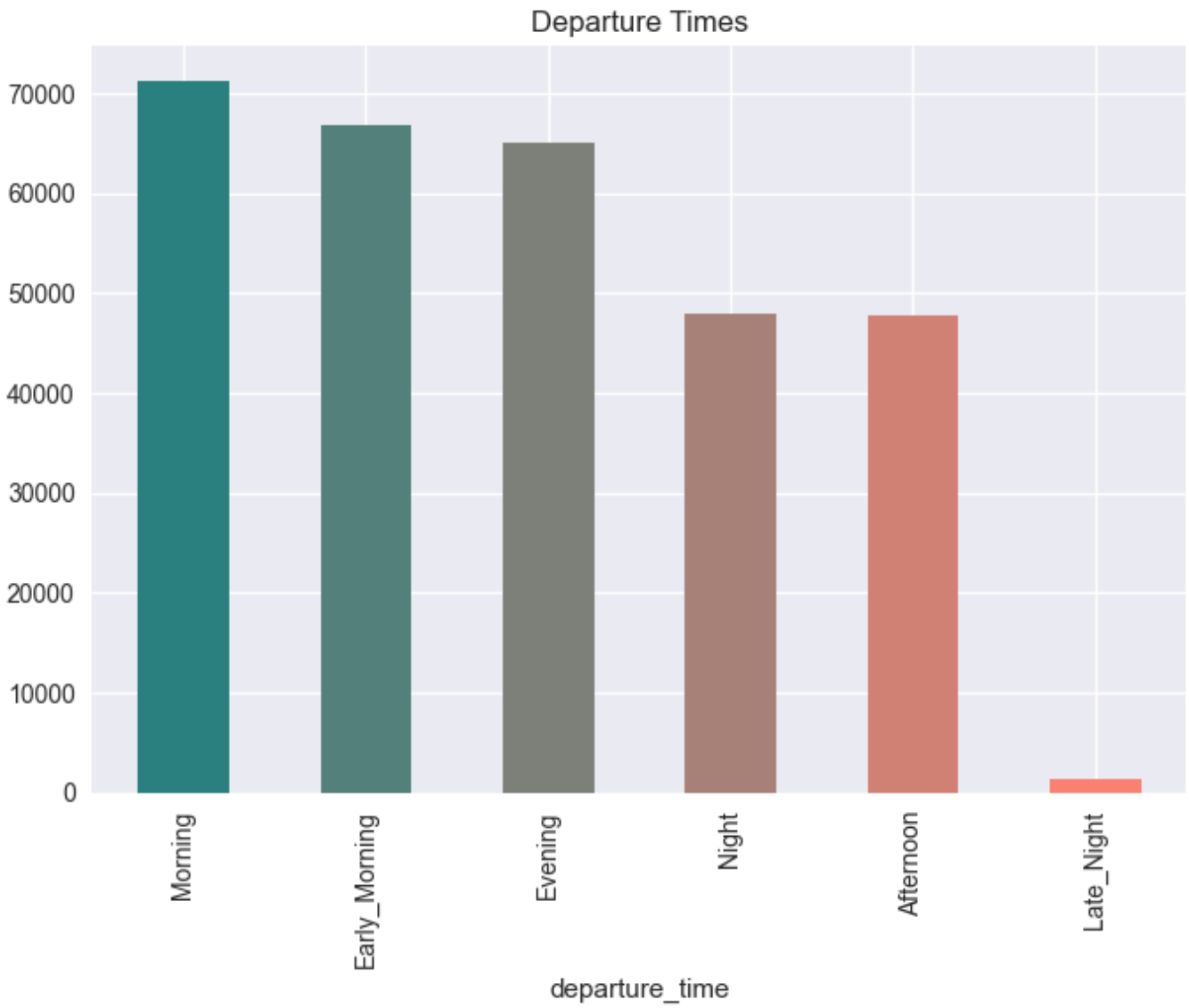
2. Flights per Airline

- **Type:** Bar chart
- **Finding:** Distribution of flights across different airlines



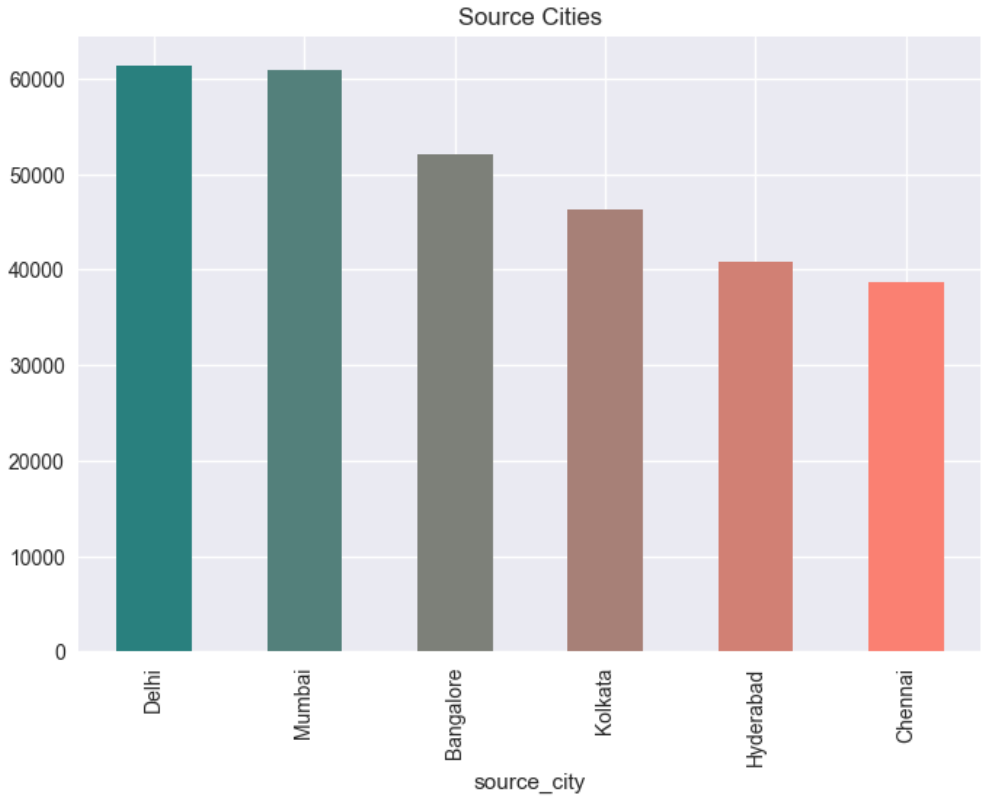
3. Departure Times Analysis

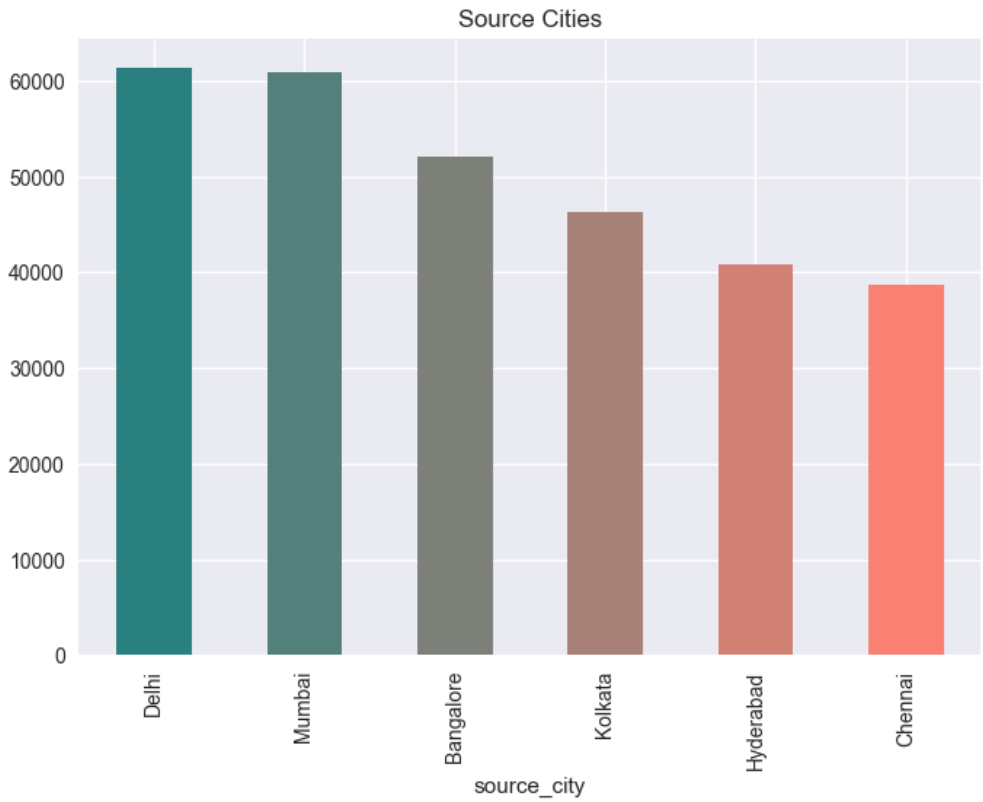
- **Type:** Bar chart
- **Finding:** Distribution of flights across different time periods



4. City Analysis

- **Source Cities:** Distribution of departure cities
- **Destination Cities:** Distribution of arrival cities





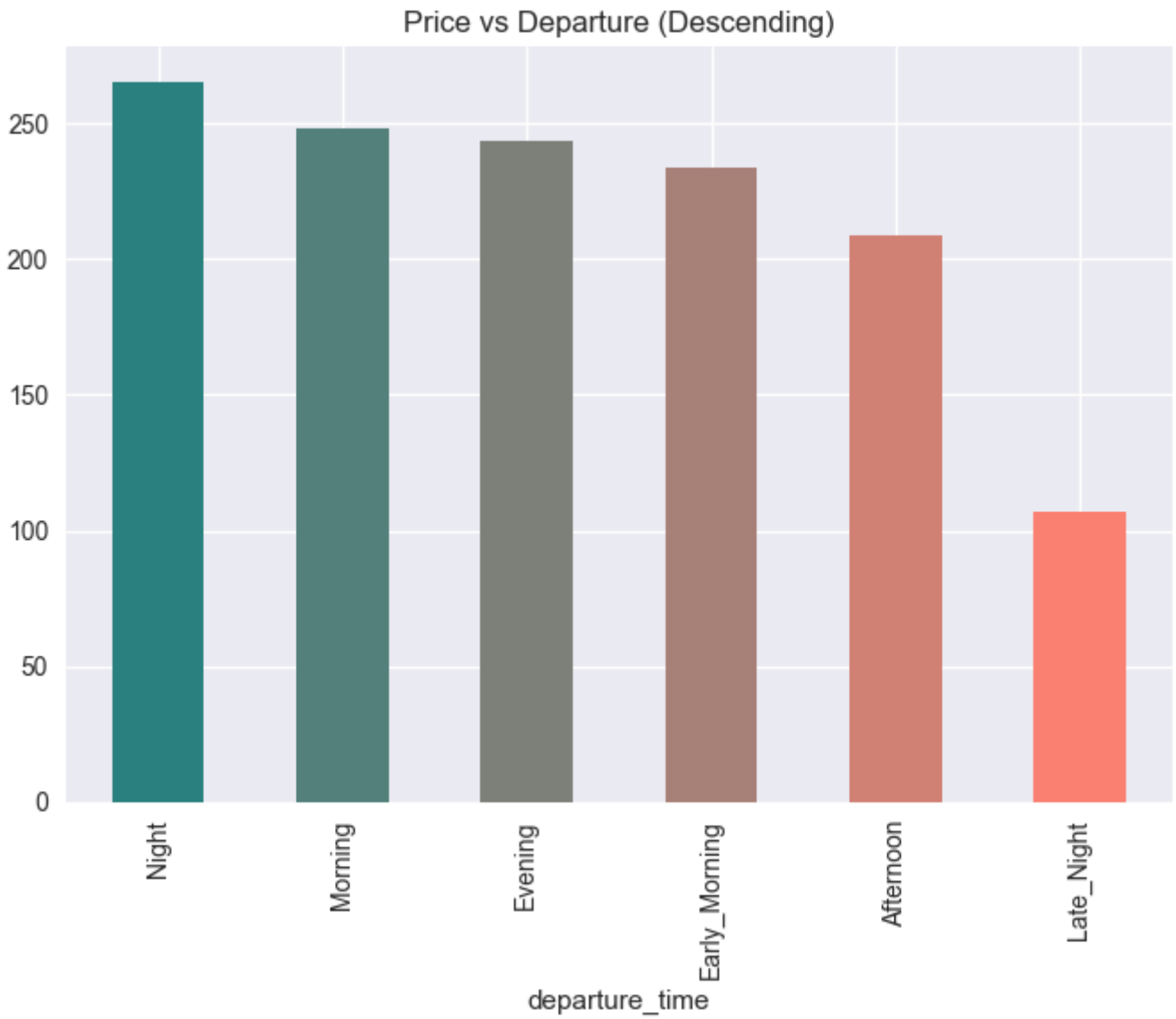
5. Price Analysis

Average Price by Airline

- Bar chart showing which airlines charge more on average

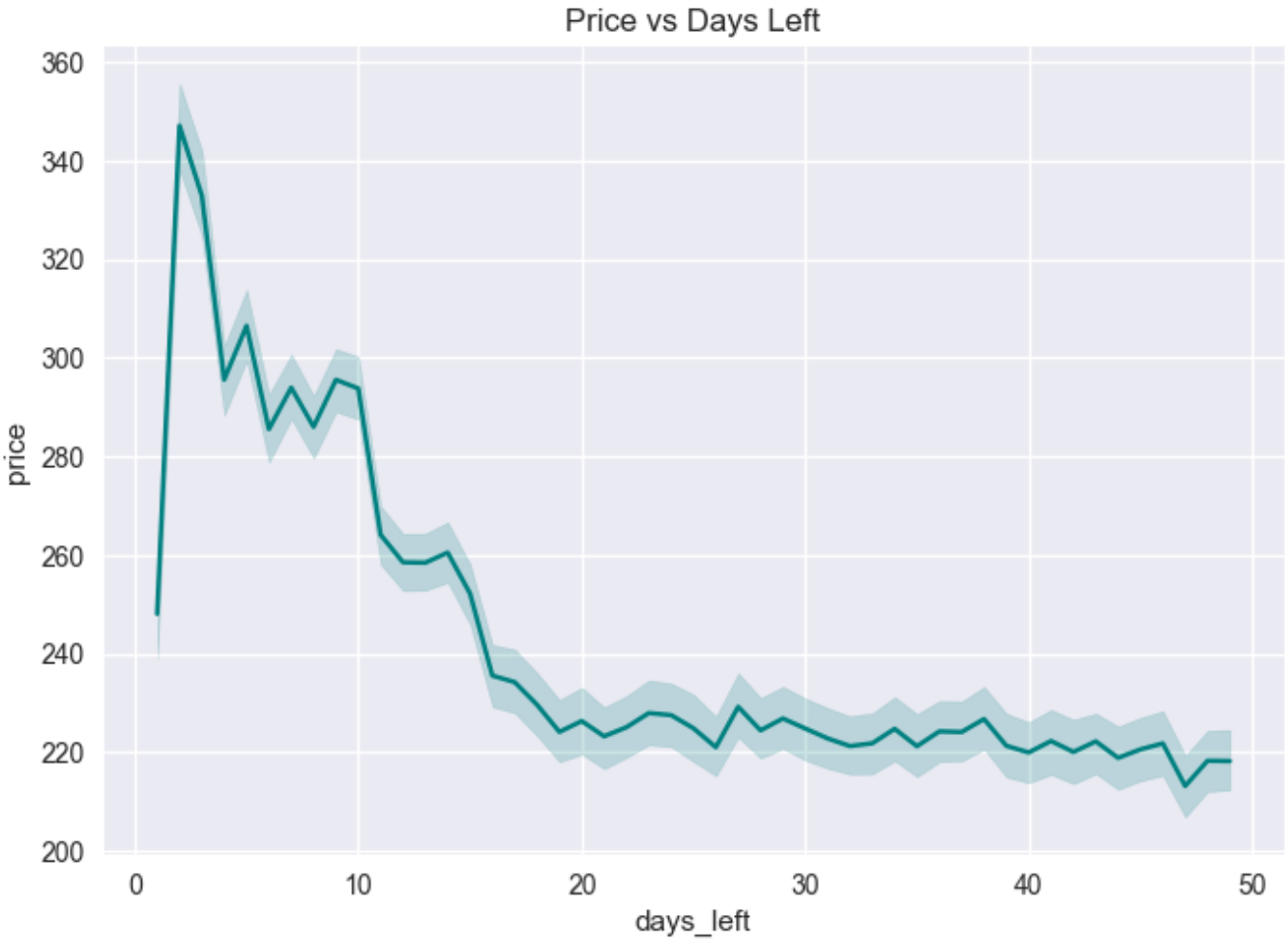
Price vs Departure Time

- Bar chart (descending order)
- **Insight:** Certain departure times command premium prices



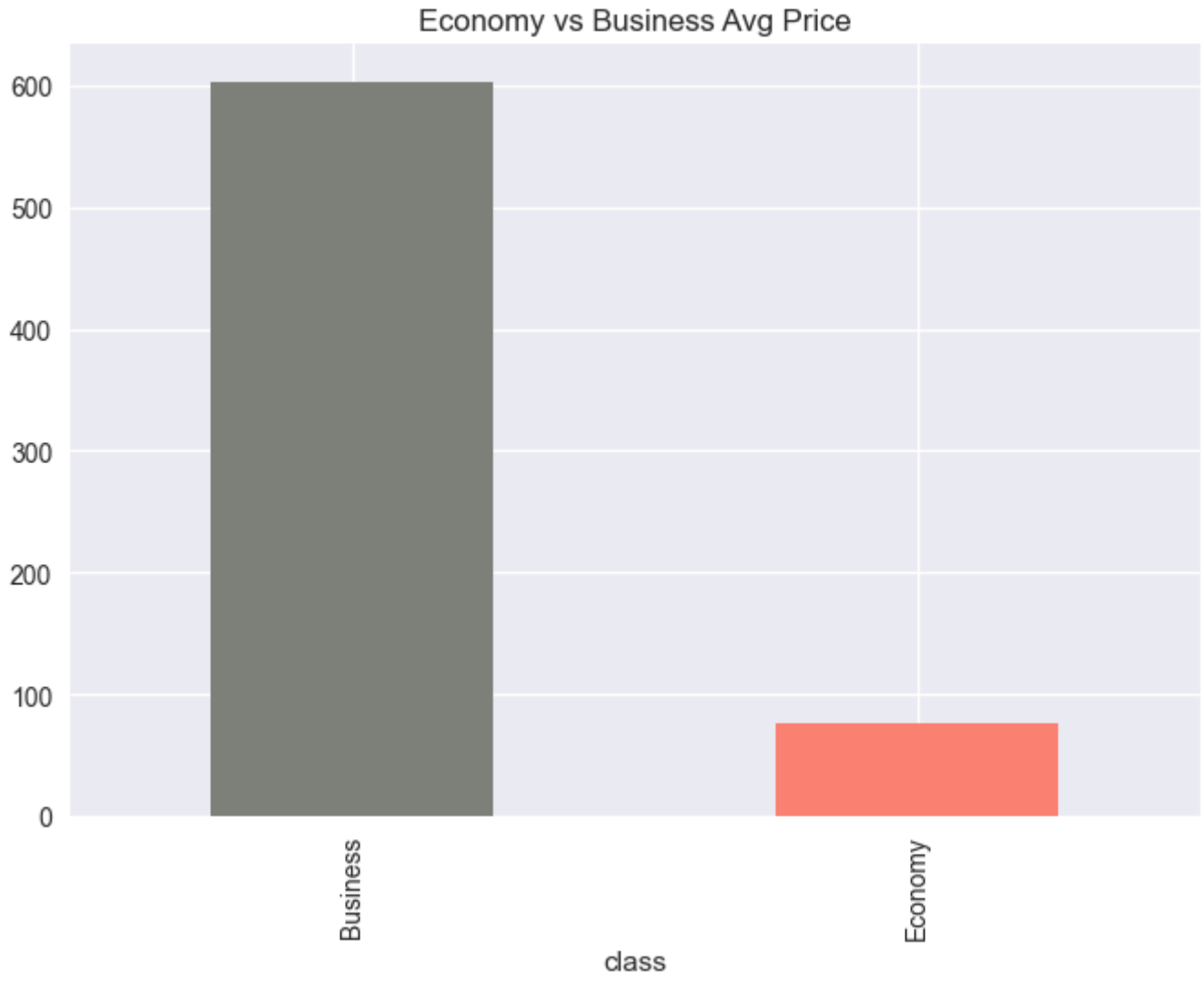
Price vs Days Left

- Line plot showing price trend
- **Insight:** Price changes as departure date approaches



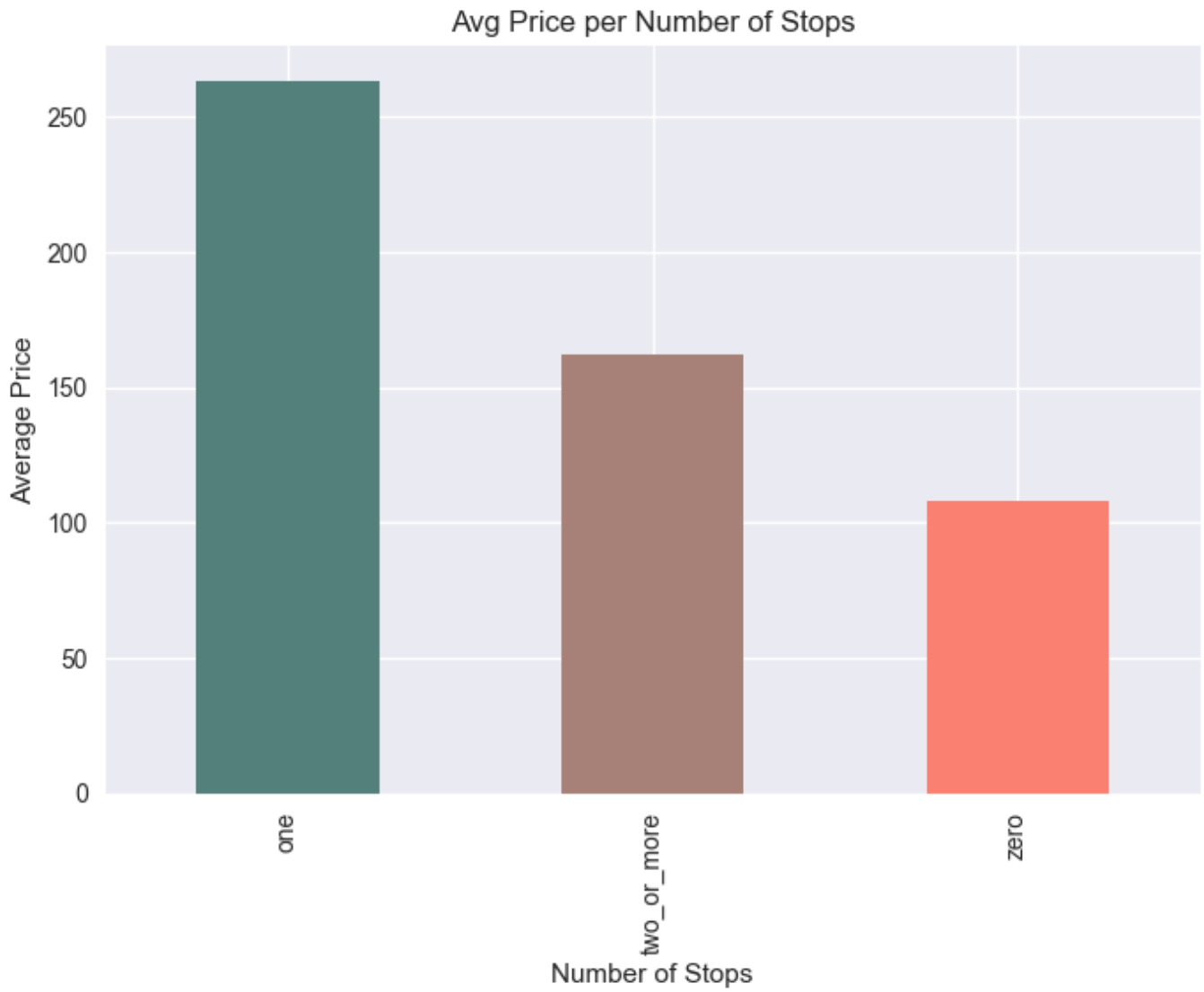
Economy vs Business

- **Finding:** Business class significantly more expensive than Economy



Price vs Number of Stops

- Bar chart showing impact of layovers
- **Insight:** Direct flights (zero stops) typically priced differently than multi-stop flights



6. Outlier Detection

- Box plots for all numerical features (`duration`, `days_left`, `price`)

Color Scheme

Custom color gradient (teal to salmon) for consistent and professional visualizations:

```
import matplotlib.colors as mcolors
colors = mcolors.LinearSegmentedColormap.from_list("", ["teal", "salmon"])
```

Model Development

Train-Test Split

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

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=101)
```

- **Training Set:** 70%
- **Test Set:** 30%
- **Random State:** 101 (for reproducibility)

Models Implemented

Four regression models were trained and compared:

1. Random Forest Regressor

```
from sklearn.ensemble import RandomForestRegressor

RF_model = RandomForestRegressor()
RF_model.fit(x_train_scaled, y_train)
```

Advantages:

- Handles non-linear relationships
- Resistant to overfitting
- Feature importance insights

2. Linear Regression

```
from sklearn.linear_model import LinearRegression

reg_model = LinearRegression()
reg_model.fit(x_train_scaled, y_train)
```

Advantages:

- Simple and interpretable
- Fast training
- Baseline model

3. Support Vector Regression (SVM)

```
from sklearn.svm import SVR
```

```
svm = SVR()
svm.fit(x_train_scaled, y_train)
```

Advantages:

- Effective in high-dimensional spaces
- Robust to outliers

4. XGBoost Regressor

```
from xgboost import XGBRegressor

xgb_model = XGBRegressor()
xgb_model.fit(x_train_scaled, y_train)
```

Advantages:

- State-of-the-art performance
- Handles missing values
- Built-in regularization

Evaluation Metrics

For each model, we calculated:

1. **R^2 Score (Coefficient of Determination)**
    - Percentage of variance explained by the model
    - Range: 0-100% (higher is better)
2. **Mean Squared Error (MSE)**
    - Average squared difference between predictions and actual values
    - Lower is better

```
from sklearn.metrics import mean_squared_error, r2_score

y_pred = model.predict(x_test_scaled)
print(f"R2: {r2_score(y_test, y_pred) * 100:.2f}%")
print("MSE:", mean_squared_error(y_test, y_pred))
```

Results

Model Performance Comparison

Model	R^2 Score	MSE	Performance

Model	R^2 Score	MSE	Performance
Random Forest	98.42%	1070.48	☆☆☆☆☆ Excellent
XGBoost	97.45%	1724.02	☆☆☆☆☆ Excellent
SVM	94.02%	4049.49	☆☆☆☆ Very Good
Linear Regression	90.41%	6498.11	☆☆☆ Good

Best Performing Model

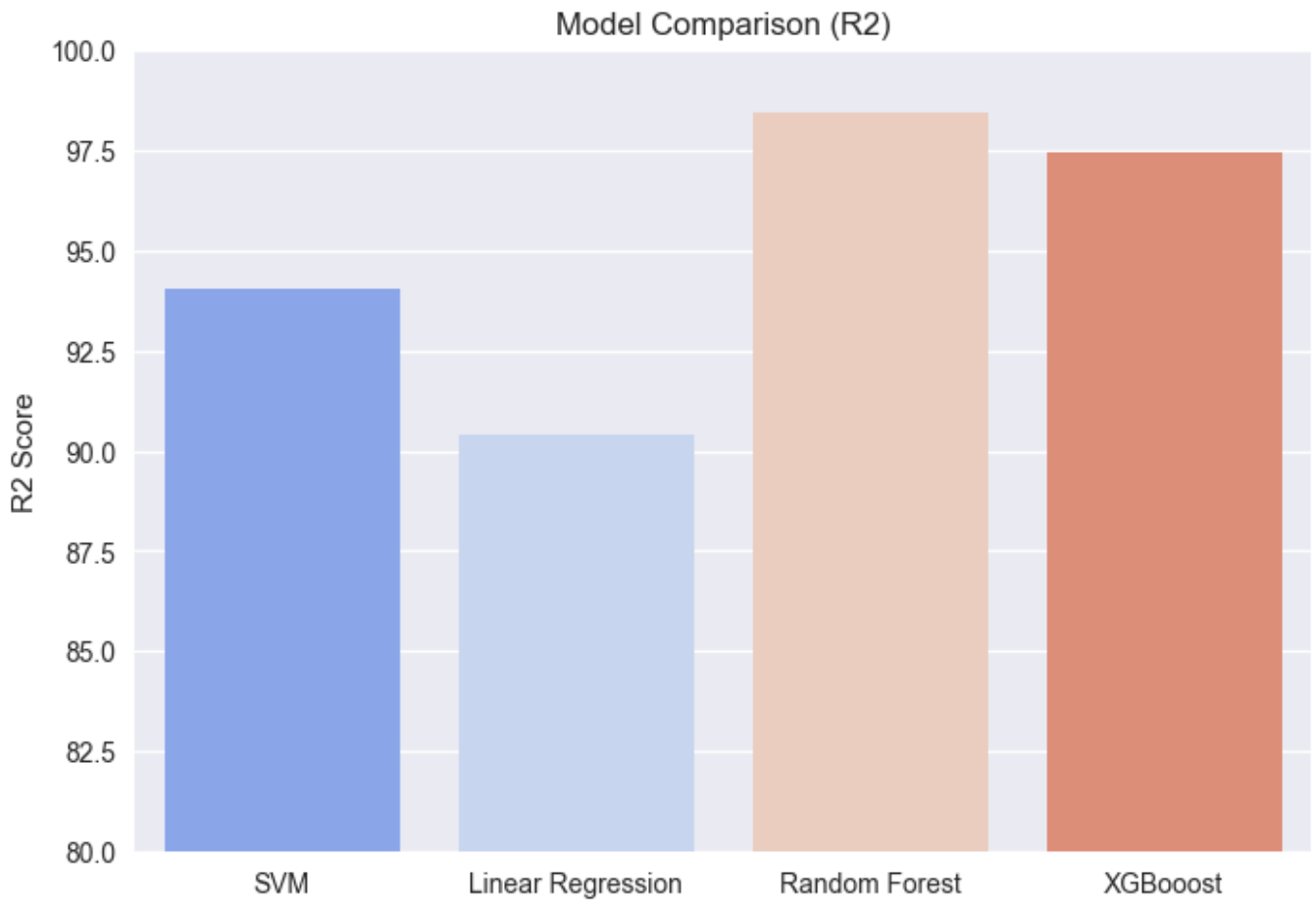
Random Forest emerged as the top performer with an R^2 score of **98.42%** and the lowest MSE of **1070.48**, indicating excellent predictive capability. **XGBoost** follows closely with an R^2 of **97.45%**.

Visualization

A comparative bar plot displays all model performances:

```
models = ["SVM", "Linear Regression", "Random Forest", "XGBooost"]
scores = [R2_for_svm, R2_for_Linear_Regression, R2_for_RF, R2_for_XGboost]

sns.barplot(x=models, y=scores, palette="coolwarm")
plt.ylim(80, 100)
plt.ylabel("R2 Score")
plt.title("Model Comparison (R2)")
```



Usage

Running the Complete Pipeline

1. **Open the notebook:**

```
jupyter notebook "final project[1].ipynb"
```

2. **Execute cells sequentially:**
   - Import libraries
   - Load and explore data
   - Visualize data
   - Preprocess features
   - Train models
   - Evaluate and compare

Making Predictions

```
# Example: Predict price for new flight data
new_flight = [[encoded_airline, encoded_source, encoded_departure,
               encoded_stops, encoded_destination, encoded_class,
               duration, days_left]]

new_flight_scaled = scaler.transform(new_flight)
predicted_price = RF_model.predict(new_flight_scaled)
print(f"Predicted Price: ${predicted_price[0]:.2f}")
```

Key Insights

From EDA

1. **Departure Time Impact:** Certain time slots command premium pricing
2. **Days Left Correlation:** Price varies significantly based on booking advance
3. **Class Premium:** Business class substantially more expensive than Economy
4. **Stops Effect:** Number of layovers directly impacts pricing
5. **Airline Variation:** Different airlines have different pricing strategies
6. **Duration Factor:** Longer flights generally cost more

From Model Training

1. **Tree-based models** (Random Forest, XGBoost) outperform linear models
2. **Ensemble methods** provide superior accuracy for this problem
3. **Feature scaling** is critical for SVM and Linear Regression performance
4. **Non-linear relationships** exist between features and price

Future Improvements

1. Model Enhancements

- **Hyperparameter tuning** using GridSearchCV or RandomizedSearchCV
- **Cross-validation** for more robust performance estimates
- **Feature importance analysis** to identify key price drivers
- **Ensemble stacking** combining multiple models

2. Feature Engineering

- **Time-based features:** Month, day of week, season
- **Route popularity:** Frequency of specific routes
- **Price history:** Historical pricing patterns
- **Demand indicators:** Holiday periods, events

3. Advanced Analysis

- **Temporal analysis:** Price trends over time
- **Route-specific models:** Specialized models for high-traffic routes
- **Competitor analysis:** Cross-airline price comparison
- **Dynamic pricing:** Real-time prediction updates

4. Additional Features

- **Confidence intervals** for predictions
- **Feature contribution explanation** (SHAP values)
- **Alternative metrics:** MAE, RMSE, MAPE
- **Segmentation analysis:** Different models for different customer segments

---

Technical Notes

Reproducibility

- Random state set to **101** for consistent results
- All preprocessing steps documented
- Model parameters explicitly stated

Performance Considerations

- Dataset size: 300K+ records
- Training time varies by model:
  - Linear Regression: Seconds
  - Random Forest: Minutes
  - XGBoost: Minutes
  - SVM: Longest (consider subset for testing)

Limitations

1. **Geographic scope:** India-focused dataset
2. **Time period:** Snapshot data, not time-series
3. **External factors:** Doesn't account for fuel prices, economic conditions
4. **Seasonal variation:** May not capture full yearly patterns