DYNAMIC FLIGHT AND HOTEL PRICE PREDICTOR

Problem Statement:

In Today's Dynamic Travel Market, Consumers struggle to discern the best times to book flight and hotel arrangements due to unpredictable price fluctuations influenced by seasonal demand, market trends, and availability. Existing solutions, such as price comparison websites and travel agents, lack the sophistication to accurately predict these fluctuations, leaving travellers at risk of overspending or missing out on savings. Thus, there is a pressing need for a comprehensive solution leveraging machine learning to analyse historical pricing data, seasonal trends, and market demand, providing actionable insights for optimal booking times. The proposed Dynamic Flight and Hotel Price Predictor addresses this gap by employing advanced algorithms to forecast price trends accurately, offering personalized recommendations and real-time alerts to ensure users make informed decisions, enhancing their travel experience and saving money in the process.

Problem Description:

- Challenges Faced by Consumers: Consumers encounter challenges in determining the best timing for booking flights and hotel accommodations to secure optimal prices.
- Factors Influencing Price Fluctuations: Price fluctuations in tickets and accommodations are influenced by factors like seasonal demand, market trends, and availability, posing difficulties for travellers in decision-making.
- Limitations of Existing Solutions: Existing solutions, such as price comparison websites and traditional travel agents, lack robust predictive capabilities and fail to leverage advanced data analytics and machine learning.
- Need for a Comprehensive Solution: There's a pressing need for a comprehensive solution employing machine learning to predict price fluctuations effectively.

- Proposed Platform Features: The proposed platform analyzes historical pricing data, seasonal trends, market demand, and other relevant factors to provide actionable insights on optimal booking times.
- Personalized Recommendations: It aims to offer personalized recommendations on the best time to book flights and accommodations, leveraging continuous analysis of pricing data and market dynamics.
- Real-time Alerts and Notifications: Real-time alerts and notifications will be provided to ensure users capitalize on potential savings opportunities.
- Empowering Travelers: Ultimately, the platform seeks to empower travelers with informed decision-making, securing the best deals on travel arrangements and enhancing their overall travel experience while saving them money.

Solution Approach

User Profiles: Users create personalized profiles with their travel preferences, including:

- Budget constraints.
- Preferred airlines or hotel chains.
- Travel dates and flexibility.

The system learns from user interactions to tailor recommendations.

Smart Price Prediction Algorithms:

Implementing advanced machine learning models to predict flight and hotel prices:

- Regression Models: Use linear regression, decision trees, or gradient boosting to estimate prices based on historical data.
- Time-Series Models: Employ ARIMA or Prophet to capture temporal patterns.

Considering features such as booking date, departure date, lead time, and market events.

Real-Time Monitoring and Alerts:

- Deploy the model in a cloud-based environment.
- Continuously monitor real-time data:
    - New bookings.
    - Market changes (e.g., flash sales, demand spikes).

- Trigger alerts when prices are expected to change significantly.

User-Friendly Interface:

Develop a web or mobile app for travelers:

Flight Price Tracker: Users input their desired route and travel dates. The system provides real-time updates on price fluctuations.

Hotel Price Watch: Users receive notifications when hotel rates drop below their specified budget.

Booking Recommendations: Based on historical trends, suggest optimal booking times.

Personalized Itineraries:

Combine flight and hotel data to create dynamic itineraries:

Budget Optimization: Recommend cost-effective options.

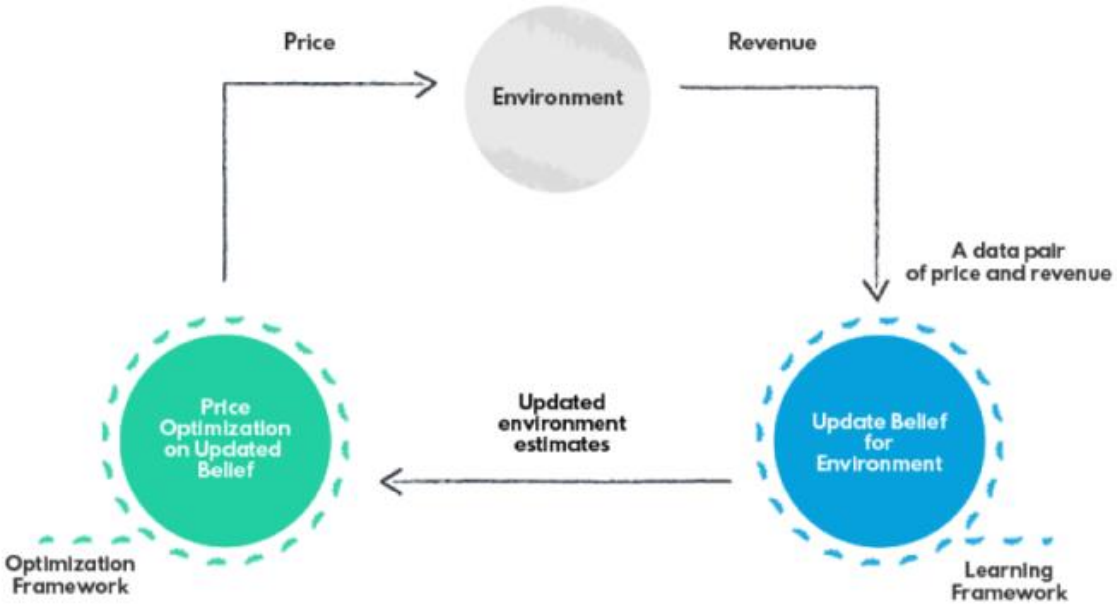
Flexible Dates: Propose alternative travel dates for better prices

BENEFITS

1. Flexible Travel Planning: Armed with insights into price trends, travelers have the flexibility to adjust their travel dates or destinations to align with more affordable options. This flexibility can be particularly beneficial for those with flexible schedules or who are seeking the best value for their money.
2. Improved Budget Management: Predictive pricing information allows travelers to plan and budget their trips more effectively. By knowing when prices are expected to be high or low, travelers can allocate their travel funds more efficiently, ensuring they get the most out of their travel budget without overspending.
3. Enhanced Competitiveness for Businesses: Airlines, hotels, and travel agencies can leverage predictive pricing models to stay competitive in the market. By offering dynamic pricing that reflects demand fluctuations, businesses can attract price-conscious travelers while maximizing revenue and occupancy rates. This allows them to maintain a competitive edge in the industry and adapt to changing market conditions more effectively.

How Dynamic Pricing Works?

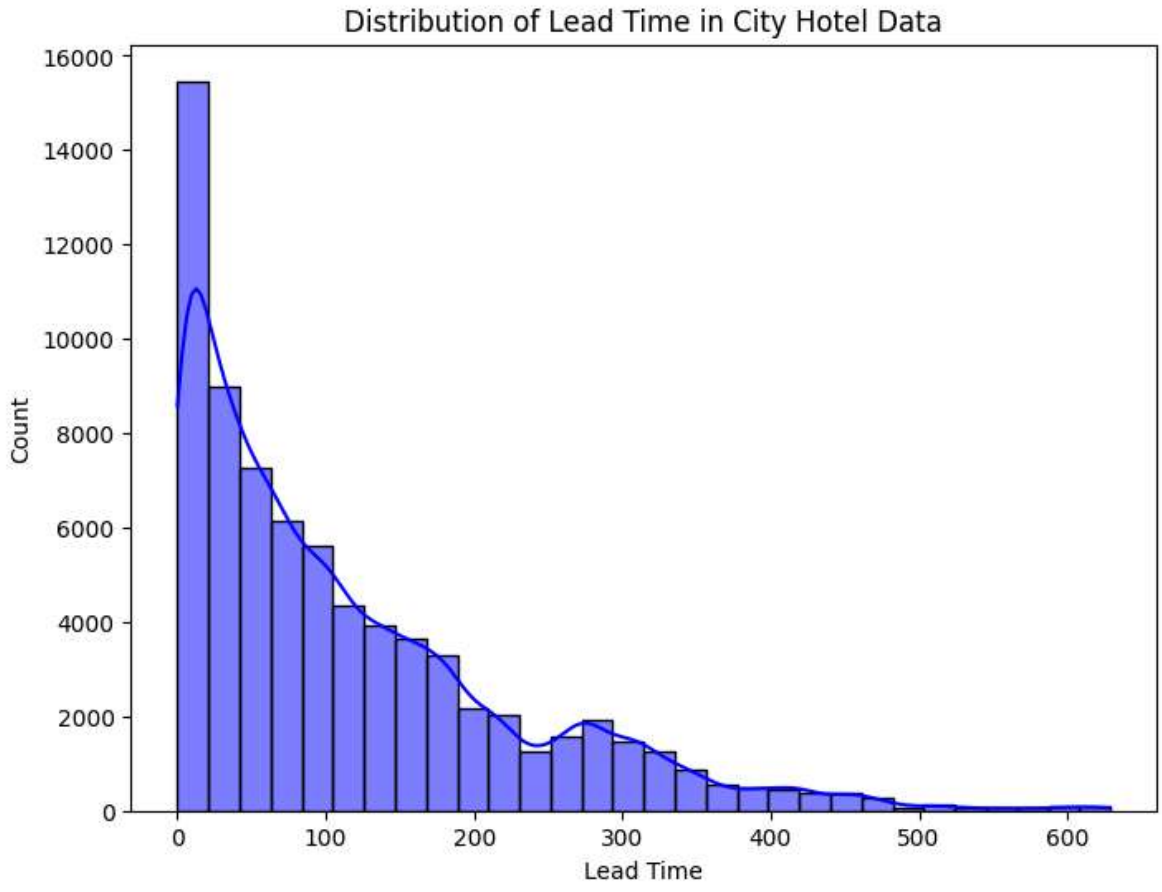
Dynamic pricing strategies replace fixed prices with fluctuating prices, calculated and updated in an automated way based on a bunch of variables. The main goal of dynamic pricing is to maximize revenue and profit by adjusting prices according to real-time market demand and supply. By doing so, businesses can adapt to changes in the market and stay competitive.

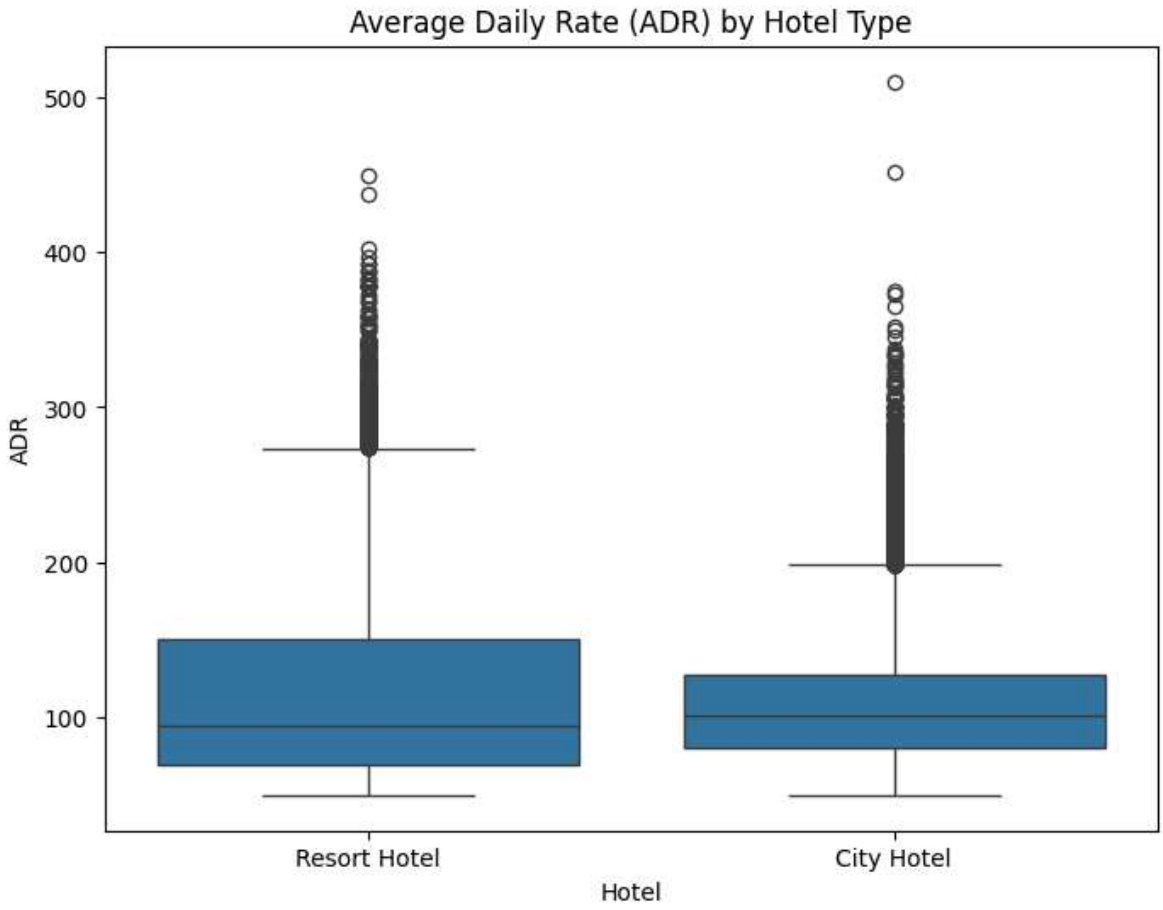


Nithya Johny & Shraddha

In [1]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from datetime import datetime

# Load data
city_hotel_prep = pd.read_csv("/content/City_hotel_Prep.csv")
cleaned_hotel_bookings = pd.read_csv("/content/hotel_bookings.csv")

# Drop 'Unnamed: 0' columns if they are just indices
if 'Unnamed: 0' in city_hotel_prep.columns:
    city_hotel_prep.drop(columns=['Unnamed: 0'], inplace=True)
if 'Unnamed: 0' in cleaned_hotel_bookings.columns:
    cleaned_hotel_bookings.drop(columns=['Unnamed: 0'], inplace=True)

# Encode categorical variables
city_hotel_prep['hotel'] = city_hotel_prep['hotel'].astype(str)
cleaned_hotel_bookings['hotel'] = cleaned_hotel_bookings['hotel'].astype(str)

# Combine both datasets to ensure all labels are encountered during fitting
combined_hotel_data = pd.concat([city_hotel_prep, cleaned_hotel_bookings], ig

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'hotel' column
combined_hotel_data['hotel'] = label_encoder.fit_transform(combined_hotel_dat
```

In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Example: Histogram of lead_time in city_hotel_prep dataset
plt.figure(figsize=(8, 6))
sns.histplot(city_hotel_prep['lead_time'], bins=30, kde=True, color='blue')
plt.title('Distribution of Lead Time in City Hotel Data')
plt.xlabel('Lead Time')
plt.ylabel('Count')
plt.show()

# Example: Boxplot of adr in cleaned_hotel_bookings dataset
plt.figure(figsize=(8, 6))
sns.boxplot(x='hotel', y='adr', data=cleaned_hotel_bookings)
plt.title('Average Daily Rate (ADR) by Hotel Type')
plt.xlabel('Hotel')
plt.ylabel('ADR')
plt.show()
```

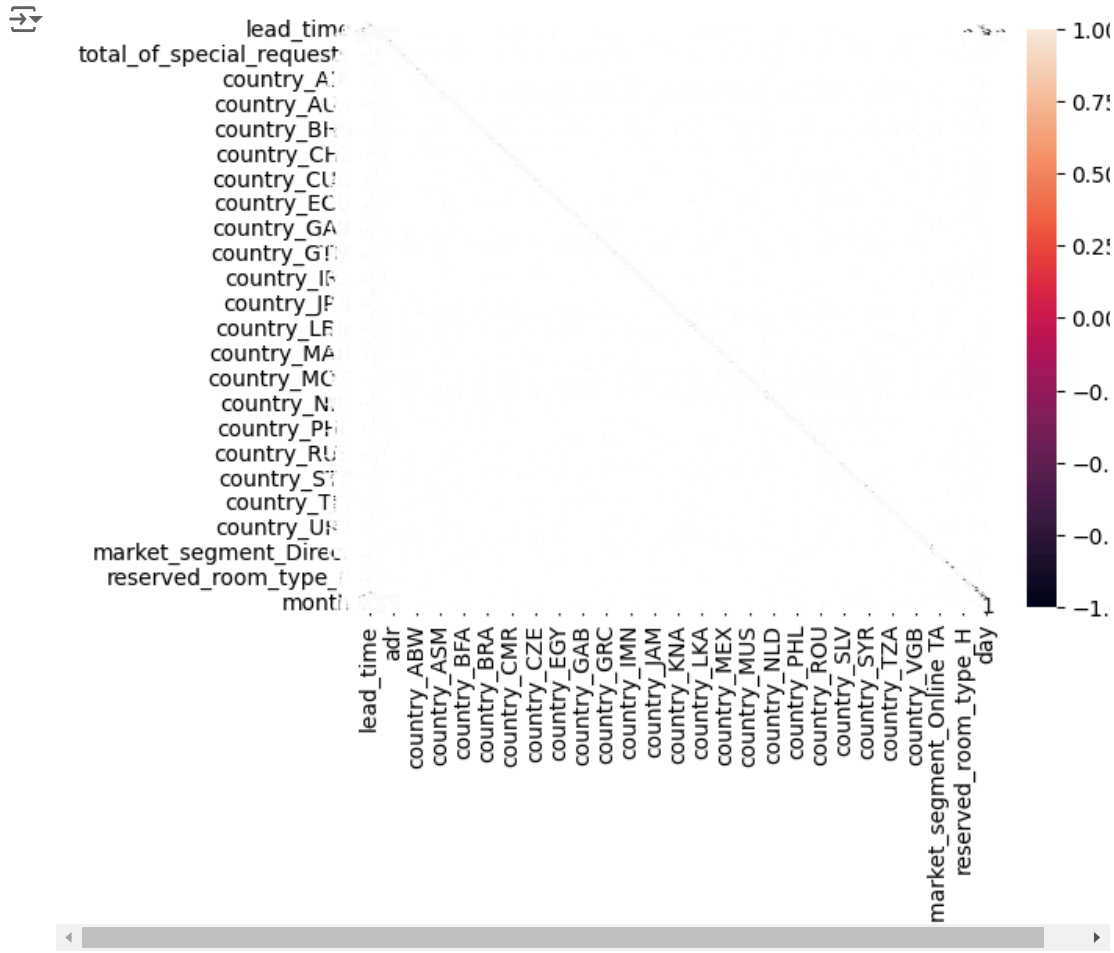



Average Daily Rate (ADR) by Hotel Type
ADR
500
400
300
200
100
Resort Hotel
City Hotel
Hotel

In [2]:
```
# Check if 'date' column exists in the DataFrame
if 'date' in combined_hotel_data.columns:
    # Convert date column to datetime format and extract relevant features
    combined_hotel_data['date'] = pd.to_datetime(combined_hotel_data['date'],

    # Extract date features
    combined_hotel_data['arrival_date_year'] = combined_hotel_data['date'].dt
    combined_hotel_data['arrival_date_month'] = combined_hotel_data['date'].d
    combined_hotel_data['arrival_date_week_number'] = combined_hotel_data['da
    combined_hotel_data['arrival_date_day_of_month'] = combined_hotel_data['d

    # Drop the original 'date' column if no longer needed
    combined_hotel_data.drop(columns=['date'], inplace=True)
else:
    print("Warning: 'date' column not found in combined_hotel_data. Proceedin

# Define categorical and numerical features
categorical_features = ['hotel', 'meal', 'country', 'market_segment', 'reserv
numerical_features = ['lead_time', 'arrival_date_year', 'arrival_date_month',
                      'arrival_date_day_of_month', 'stays_in_weekend_nights',
                      'adults', 'total_of_special_requests']

# Preprocessing pipeline
# Define preprocessing steps for numerical and categorical features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Handle missing values wit
    ('scaler', StandardScaler())  # Standardize numerical features
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  #
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # One-hot encode cate
])

# Combine preprocessing steps using ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Split data into train and test sets
if 'adr' in combined_hotel_data.columns:
    X = combined_hotel_data.drop(['adr'], axis=1)  # Features
    y = combined_hotel_data['adr']  # Target variable

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
else:
    raise KeyError("Column 'adr' not found in combined_hotel_data DataFrame. (

# Define models to evaluate
models = [
    ('Linear Regression', LinearRegression()),
    ('Decision Tree', DecisionTreeRegressor(random_state=42))
]

# Evaluate models
```

```
for name, model in models:
    # Create a pipeline with preprocessing and the current model
    regressor = Pipeline(steps=[('preprocessor', preprocessor),
                                ('regressor', model)])

    # Fit the pipeline
    regressor.fit(X_train, y_train)

    # Predict on test set
    y_pred = regressor.predict(X_test)

    # Evaluate metrics
    mse = mean_squared_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    r2 = r2_score(y_test, y_pred)

    # Print results
    print(f"Model: {name}")
    print(f"Mean Squared Error: {mse:.2f}")
    print(f"Root Mean Squared Error: {rmse:.2f}")
    print(f"R-squared: {r2:.2f}")
    print("="*50)
```

```
Model: Linear Regression
Mean Squared Error: 905.83
Root Mean Squared Error: 30.10
R-squared: 0.48
==================================================
Model: Decision Tree
Mean Squared Error: 128.20
Root Mean Squared Error: 11.32
R-squared: 0.93
==================================================
```




```
#Model Selection and Training
#Split the Data
#Split into training and testing sets
from sklearn.model_selection import train_test_split

X = combined_hotel_data.drop('adr', axis=1)
y = combined_hotel_data['adr']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


#Train models like Linear Regression, Decision Trees, and Random Forest
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    print(f"{name} model trained.")
```

```
Linear Regression model trained.
Decision Tree model trained.
Random Forest model trained.
```

```
#Use Grid Search or Random Search:

from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}

grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred = best_model.predict(X_test)
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared:", r2_score(y_test, y_pred))


##Evaluate using metrics:
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

y_pred = best_model.predict(X_test)
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R-squared:", r2_score(y_test, y_pred))



#Perform cross-validation:

from sklearn.model_selection import cross_val_score

scores = cross_val_score(best_model, X, y, cv=5, scoring='neg_mean_squared_error')
print("Cross-validation scores:", scores)


#Identify important features

importances = best_model.feature_importances_
feature_names = X.columns
feature_importances = pd.Series(importances, index=feature_names)
feature_importances = feature_importances.sort_values(ascending=False)
print(feature_importances)


#Save the model
import joblib

joblib.dump(best_model, 'hotel_price_prediction_model.pkl')


#Create a web interface using Flask or Streamlit:

import streamlit as st
import joblib

model = joblib.load('hotel_price_prediction_model.pkl')

st.title("Hotel Price Prediction")
lead_time = st.number_input("Lead Time")
stays_in_weekend_nights = st.number_input("Stays in Weekend Nights")
stays_in_week_nights = st.number_input("Stays in Week Nights")
adults = st.number_input("Number of Adults")
# Add inputs for other features...

if st.button("Predict"):
    prediction = model.predict([[lead_time, stays_in_weekend_nights, stays_in_week_nights, adults,    ]])
```

```
    prediction = model.predict([[lead_time, stays_in_weekend_nights, stays_in_week_nights, adults, ...]])
    st.write(f"Predicted Price: {prediction[0]}")
```

Start coding or generate with AI.

Rishi and Deekshith

Business Modeling for Dynamic Flight and Hotel Price Prediction

Subscription-Based Model:

Freemium Tier: Provide basic features for free to attract users and build a customer base.
Paid Tier: Offer advanced features and insights, such as more accurate price predictions, personalized recommendations, and exclusive deals, for a subscription fee.

User Conversion Strategy:

Engagement: Use engaging content and tools in the free tier to demonstrate value.
Personalization: Offer personalized offers and insights to free users as a teaser for the paid features.
Limited-Time Offers: Provide discounts on the subscription fee for first-time users or during special promotions.
User Education: Regularly update users about the benefits of the paid tier through newsletters and in-app notifications.

Financial Equation for Dynamic Flight and Hotel Price Prediction Assumptions:

Initial Price of Subscription (P): $100/month
Growth Rate (r): 4% per month (this growth rate can be due to increased user adoption and market expansion)
Time Interval (t): Number of months

Financial Equation:

$$Y = P \times (1 + r)t$$

Where:
Y = Revenue over time
P = Initial subscription price

r = Growth rate
t = Time interval (in months)
Equation with Values:
$Y=100\times(1+0.04)^t$

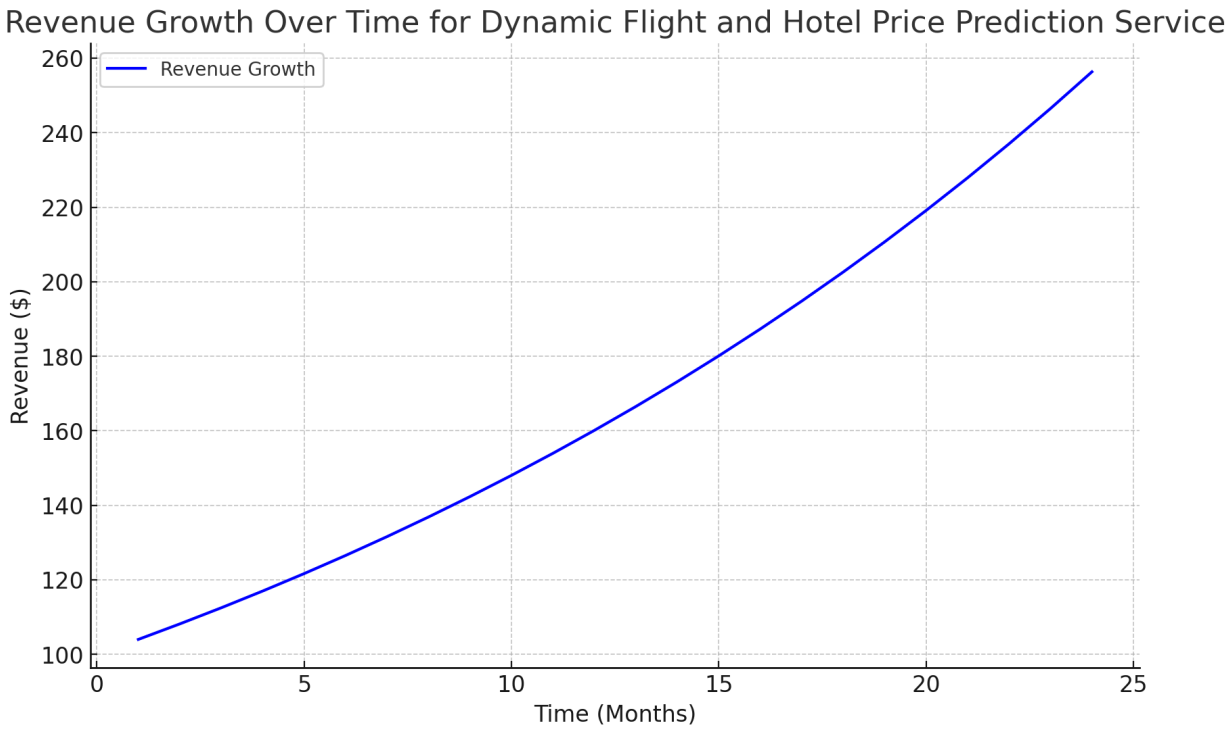
Financial Projection

Here are the calculations for the first few months to provide a sense of the growth pattern:

Month 1:
$Y1=100\times(1+0.04)1=100\times1.04=104$
Month 2:
$Y2=100\times(1+0.04)2=100\times1.0816=108.16$
Month 3:
$Y3=100\times(1+0.04)3=100\times1.124864=112.49$
Month 12:
$Y12=100\times(1+0.04)12=100\times1.601032=160.10$
Month 24:
$Y24=100\times(1+0.04)24=100\times2.563038=256.30$

This example illustrates how the subscription revenue grows over time with a 4% monthly growth rate. Let's now plot the graph to visualize this growth.

Here is the graph showing the revenue growth over 24 months for the dynamic flight and hotel price prediction service.



Financial Equation Recap:

$Y=100\times(1+0.04)t$
Month 1: $104
Month 12: $160.10
Month 24: $256.30

The graph visualizes this growth, indicating a steady increase in revenue as the number of months progresses. The 4% monthly growth rate leads to significant revenue expansion over time.