PROJECT_4_hotel_booking_cancelation

March 18, 2025

0.1 AI-Driven Demand Forecasting and Risk Optimization - Predicting Customer Behavior in Dynamic Markets

(Hotel Booking Cancellation Prediction Model)**

Project Objectives - **Develop predictive models** to classify booking cancellations. - **Perform Exploratory Data Analysis (EDA)** to identify key factors influencing cancellations. - **Feature Engineering** to enhance predictive accuracy. - **Compare multiple classification models**, including **Decision Trees, KNN, Logistic Regression, Random Forest, and XGBoost**. - **Optimize model performance** through **hyperparameter tuning**. - **Translate findings into business insights for hotel management.**

0.2 PART 1: BACKGROUND

0.2.1 Executive Summary:

This project applies machine learning models to predict **hotel booking cancellations**, helping hotels optimize revenue, manage overbookings, and enhance customer retention strategies. The final model, **Random Forest Classifier**, achieved **89.1% accuracy and an F1-score of 85.4%**, making it the most effective solution for deployment. The insights generated from this model not only benefit hotel operations but also provide **risk assessment strategies for the finance and insurance industries**, particularly in revenue forecasting and loss mitigation.

0.2.2 Problem Statement

Hotel booking cancellations present a **significant financial challenge**,40% of the customers who booked cancelled, leading to **revenue loss, inaccurate demand forecasting, and operational inefficiencies**. Understanding the factors behind cancellations enables hotels to:
- Reduce financial losses from last-minute cancellations.
- Adjust overbooking strategies to ensure optimal occupancy.
- Improve customer segmentation and targeted promotions.
- Support risk evaluation for travel insurance providers and financial institutions.

Our Machine learning driven methodology approach provides a data-driven approach to predict cancellations, allowing businesses to take proactive measures.

0.2.3 Solution Approach

This project builds a predictive model using structured hotel booking data to classify whether a booking will be canceled. The methodology follows:
1. **Exploratory Data Analysis (EDA)** to uncover key cancellation patterns and trends.

2. **Feature Engineering** to enhance predictive accuracy, including lead time transformation, deposit type flags, and high-risk customer segmentation.
3. **Model Selection & Training** by testing Decision Trees, Logistic Regression, K-Nearest Neighbors, Random Forest, and Gradient Boosting models.
4. **Feature Importance Analysis** to identify the strongest predictors of cancellations.
5. **Business Recommendations** to implement predictive insights for reducing cancellations.

0.2.4 Key Results & Model Performance

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Best Use Case**
Decision Tree	**78.9%**	**79.0%**	**62.7%**	**69.9%**	Basic decision-making
K-Nearest Neighbors	**76.7%**	**71.8%**	**66.4%**	**68.9%**	Small datasets
Logistic Regression	**82.0%**	**81.2%**	**70.2%**	**75.3%**	Interpretable predictions
Random Forest	**89.1%**	**89.1%**	**81.9%**	**85.4%**	**Best overall model**
Gradient Boosting	**84.4%**	**83.8%**	**74.2%**	**78.7%**	Alternative model
RFE-Logistic Regression	**75.4%**	**99.1%**	**37.2%**	**54.1%**	High precision, low recall

Best Model for Deployment: Random Forest Classifier - Highest accuracy and F1-score, balancing precision and recall.
- Better handling of feature interactions, outperforming simpler models.
- Provides explainability through feature importance analysis.

0.2.5 Feature Importance & Business Insights

The **most influential features driving cancellations** were:
1. Lead Tim: Longer lead times increase cancellation likelihood.
2. Non-Refundable Deposits: Drastically reduce cancellations.
3. Country of Origin (Portugal, Germany, Turkey): Customers from certain regions cancel more often.
4. Special Requests: Guests with more requests cancel less.
5. Pricing (ADR - Average Daily Rate): Higher prices increase cancellation probability.
6. Booking Channel (OTAs vs. Direct Bookings): OTA-based bookings cancel more.
7. Customer Type: Transient customers have the highest cancellation rates.

These insights can help hotels implement targeted cancellation mitigation strategies, such as:
- Stricter cancellation policies for high-risk bookings. - Offering prepaid discounts to encourage non-refundable reservations. - Adjusting pricing models and promotions based on cancellation risk.

0.2.6 Impact Beyond Hospitality: Finance & Insurance Industry Applications

Beyond hotel operations, this model provides **valuable risk assessment tools** for the **finance and insurance industries**, including:
- **Revenue Forecasting**: Financial analysts can **predict cash flow fluctuations** based on expected cancellations.
- **Loan Risk Evaluation**: Hotels seeking financing can use cancellation forecasts to demonstrate stable revenue streams.
- **Travel Insurance Risk Assessment**: Insurers can adjust premiums for policies covering hotel cancellations based on model predictions.
- **Dynamic Pricing in Financial Markets**: Similar methodologies can be applied to predict demand in financial assets and travel booking platforms.

This highlights the broader commercial value of machine learning-driven demand forecasting.

0.2.7 Final Recommendations

1. Deploy the Random Forest model** to predict and mitigate cancellations.

2. Implement proactive booking policies** based on risk segmentation.

3. Leverage model insights to optimize revenue management strategies.
4. Apply cancellation prediction models to the finance and insurance sectors.

This project demonstrates how **AI-driven demand forecasting** can improve operational efficiency, revenue management, and risk assessment, creating significant value for **hotels, financial institutions, and insurers** alike.

0.2.8 Problem Definition & Business Context

0.3 Why Predicting Hotel Booking Cancellations Matters

Unanticipated hotel booking cancellations cause **significant revenue loss, operational inefficiencies, and customer dissatisfaction**. This project leverages machine learning to enable:

- **Hotel Revenue Managers** - Adjust overbooking strategies to minimize lost revenue.
- **Operations Teams** - Optimize staffing and inventory allocation.
- **Customers** - Improve booking experiences with fairer policies.
- **Data Analysts & Business Intelligence Teams** - Gain insights into cancellation patterns for strategic planning.

With an AI-driven approach, hotels can implement **proactive measures** to manage booking cancellations effectively.

0.4 PART 2: DATA

0.4.1 Exploratory Data Analysis

Load and Inspect Data

[1]:
```
# Import required libraries
import pandas as pd
import numpy as np
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, boxcox

import warnings
warnings.filterwarnings('ignore')


# Load dataset
file_path = "/content/hotel_bookings.csv"
df = pd.read_csv(file_path)

# Display dataset overview
print("\nDataset Overview:")
print(df.info())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 30 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   hotel                           119390 non-null  object
 1   is_canceled                     119390 non-null  int64
 2   lead_time                       119390 non-null  int64
 3   arrival_date_year               119390 non-null  int64
 4   arrival_date_month              119390 non-null  object
 5   arrival_date_week_number        119390 non-null  int64
 6   arrival_date_day_of_month       119390 non-null  int64
 7   stays_in_weekend_nights         119390 non-null  int64
 8   stays_in_week_nights            119390 non-null  int64
 9   adults                          119390 non-null  int64
 10  children                        119386 non-null  float64
 11  babies                          119390 non-null  int64
 12  meal                            119390 non-null  object
 13  country                         118902 non-null  object
 14  market_segment                  119390 non-null  object
 15  distribution_channel            119390 non-null  object
 16  is_repeated_guest               119390 non-null  int64
 17  previous_cancellations          119390 non-null  int64
 18  previous_bookings_not_canceled  119390 non-null  int64
 19  reserved_room_type              119390 non-null  object
 20  assigned_room_type              119390 non-null  object
 21  booking_changes                 119390 non-null  int64
 22  deposit_type                    119390 non-null  object
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
```

```
 25  days_in_waiting_list            119390 non-null  int64
 26  customer_type                   119390 non-null  object
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64
 29  total_of_special_requests       119390 non-null  int64
dtypes: float64(4), int64(16), object(10)
memory usage: 27.3+ MB
None
```

[2]:
```
# Summary statistics
print("\nSummary Statistics:")
print(df.describe())
```

```
Summary Statistics:
         is_canceled      lead_time  arrival_date_year  \
count  119390.000000  119390.000000      119390.000000
mean        0.370416     104.011416        2016.156554
std         0.482918     106.863097           0.707476
min         0.000000       0.000000        2015.000000
25%         0.000000      18.000000        2016.000000
50%         0.000000      69.000000        2016.000000
75%         1.000000     160.000000        2017.000000
max         1.000000     737.000000        2017.000000

       arrival_date_week_number  arrival_date_day_of_month  \
count             119390.000000              119390.000000
mean                  27.165173                  15.798241
std                   13.605138                   8.780829
min                    1.000000                   1.000000
25%                   16.000000                   8.000000
50%                   28.000000                  16.000000
75%                   38.000000                  23.000000
max                   53.000000                  31.000000

       stays_in_weekend_nights  stays_in_week_nights         adults  \
count            119390.000000         119390.000000  119390.000000
mean                  0.927599              2.500302       1.856403
std                   0.998613              1.908286       0.579261
min                   0.000000              0.000000       0.000000
25%                   0.000000              1.000000       2.000000
50%                   1.000000              2.000000       2.000000
75%                   2.000000              3.000000       2.000000
max                  19.000000             50.000000      55.000000

            children         babies  is_repeated_guest  \
count  119386.000000  119390.000000      119390.000000
mean        0.103890       0.007949           0.031912
```

```
std         0.398561       0.097436          0.175767
min         0.000000       0.000000          0.000000
25%         0.000000       0.000000          0.000000
50%         0.000000       0.000000          0.000000
75%         0.000000       0.000000          0.000000
max        10.000000      10.000000          1.000000

       previous_cancellations  previous_bookings_not_canceled  \
count           119390.000000                   119390.000000
mean                 0.087118                        0.137097
std                  0.844336                        1.497437
min                  0.000000                        0.000000
25%                  0.000000                        0.000000
50%                  0.000000                        0.000000
75%                  0.000000                        0.000000
max                 26.000000                       72.000000

       booking_changes          agent      company  days_in_waiting_list  \
count    119390.000000  103050.000000  6797.000000         119390.000000
mean          0.221124      86.693382   189.266735              2.321149
std           0.652306     110.774548   131.655015             17.594721
min           0.000000       1.000000     6.000000              0.000000
25%           0.000000       9.000000    62.000000              0.000000
50%           0.000000      14.000000   179.000000              0.000000
75%           0.000000     229.000000   270.000000              0.000000
max          21.000000     535.000000   543.000000            391.000000

                 adr  required_car_parking_spaces  total_of_special_requests
count  119390.000000                119390.000000              119390.000000
mean      101.831122                     0.062518                   0.571363
std        50.535790                     0.245291                   0.792798
min        -6.380000                     0.000000                   0.000000
25%        69.290000                     0.000000                   0.000000
50%        94.575000                     0.000000                   0.000000
75%       126.000000                     0.000000                   1.000000
max      5400.000000                     8.000000                   5.000000
```

Check Missing Values

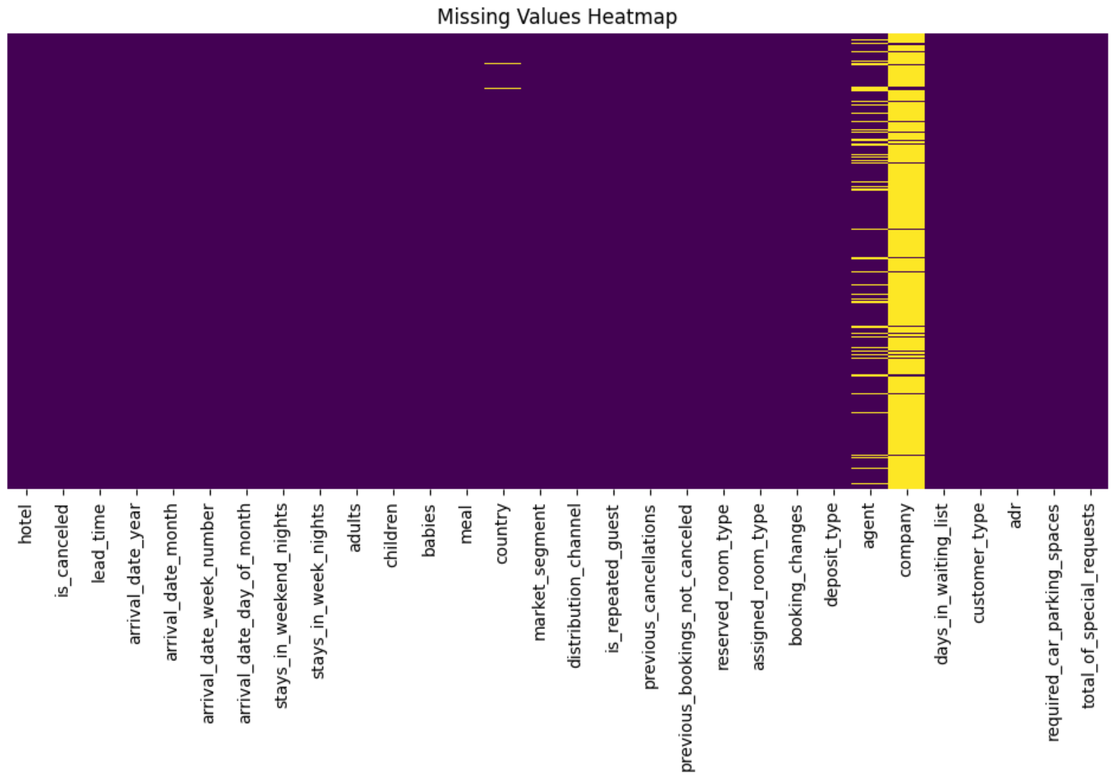
[3]:
```
# Check for missing values
print("\nMissing Values Per Column:")
print(df.isnull().sum())
```

```

Missing Values Per Column:
hotel                              0
is_canceled                        0
lead_time                          0
arrival_date_year                  0
```

```
arrival_date_month                    0
arrival_date_week_number              0
arrival_date_day_of_month             0
stays_in_weekend_nights               0
stays_in_week_nights                  0
adults                                0
children                              4
babies                                0
meal                                  0
country                             488
market_segment                        0
distribution_channel                  0
is_repeated_guest                     0
previous_cancellations                0
previous_bookings_not_canceled        0
reserved_room_type                    0
assigned_room_type                    0
booking_changes                       0
deposit_type                          0
agent                             16340
company                          112593
days_in_waiting_list                  0
customer_type                         0
adr                                   0
required_car_parking_spaces           0
total_of_special_requests             0
dtype: int64
```

[4]:

```
# Visualizing missing values
plt.figure(figsize=(12, 5))
sns.heatmap(df.isnull(), cmap="viridis", cbar=False, yticklabels=False)
plt.title("Missing Values Heatmap")
plt.show()
```



Observations: - Significant missing values exist in the agent (16,340 missing), company (112,593 missing), country (488 missing), and children (4 missing) columns. - The company column is missing over 94% of its values, making it unreliable for analysis. - The children column has very few missing values, so imputation is an option.

Actions: - Drop the company column since it has excessive missing values. - Impute children using median value to maintain dataset integrity. - Impute country using mode (most frequent value) - Handle rows with missing agent values, we shall determin how this variable is critical for predictions, else drop it.

Analyze Target Variable (Cancellations)

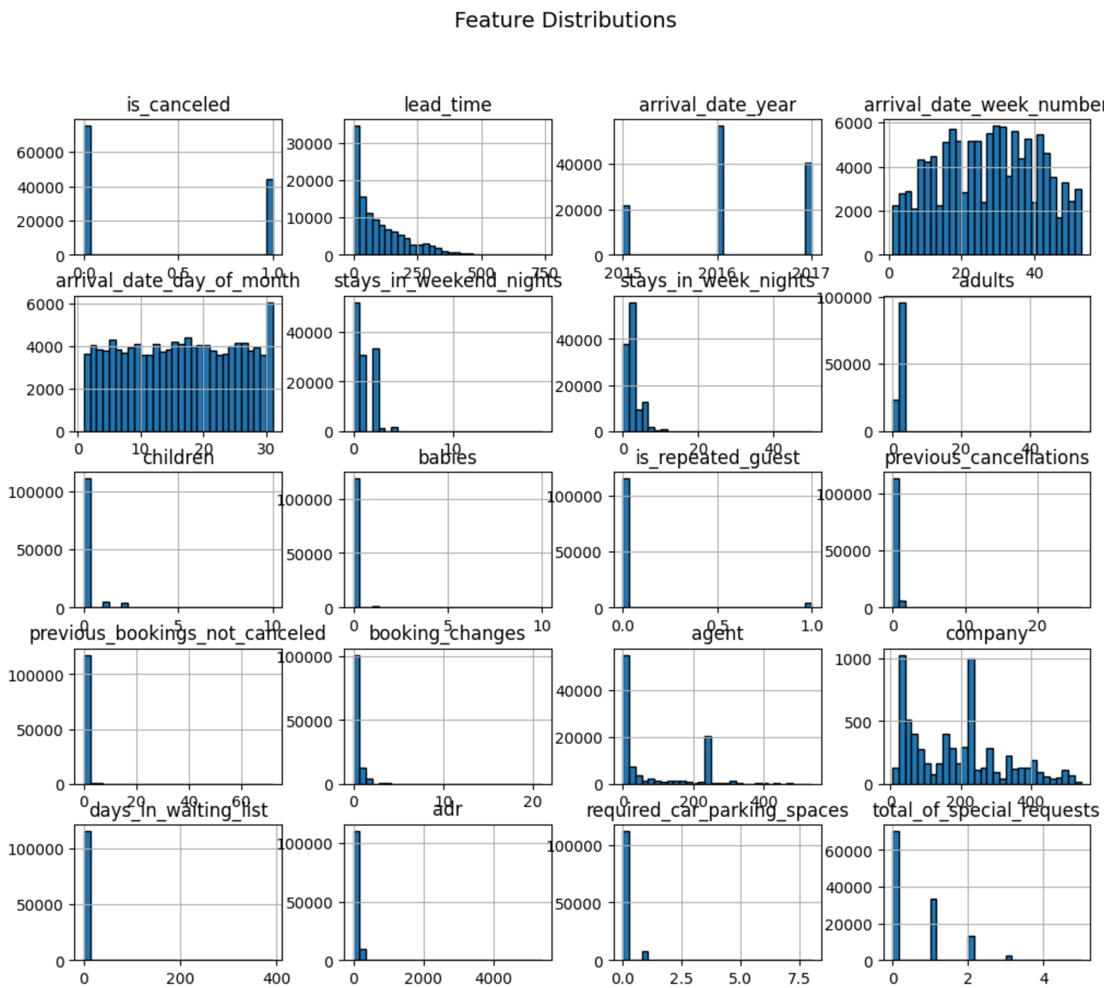
[5]:
```
# Plot cancellation distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='is_canceled', data=df, palette='coolwarm')
plt.title('Booking Cancellations Distribution')
plt.xlabel('Is Canceled (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()

# Calculate cancellation rate
cancellation_rate = df['is_canceled'].mean() * 100
print(f"\nCancellation Rate: {cancellation_rate:.2f}% of bookings were canceled.
↪")
```


```
Cancellation Rate: 37.04% of bookings were canceled.
```

Observations: - About 40% of bookings were canceled, confirming that cancellations are a significant problem for hotels. - The dataset is not highly imbalanced, meaning class balancing techniques like SMOTE may not be needed.

Feature Distributions & Outlier Detection

[6]:
```
# Plot histograms for numerical features
df.hist(figsize=(12, 10), bins=30, edgecolor="black")
plt.suptitle("Feature Distributions", fontsize=14)
plt.show()
```

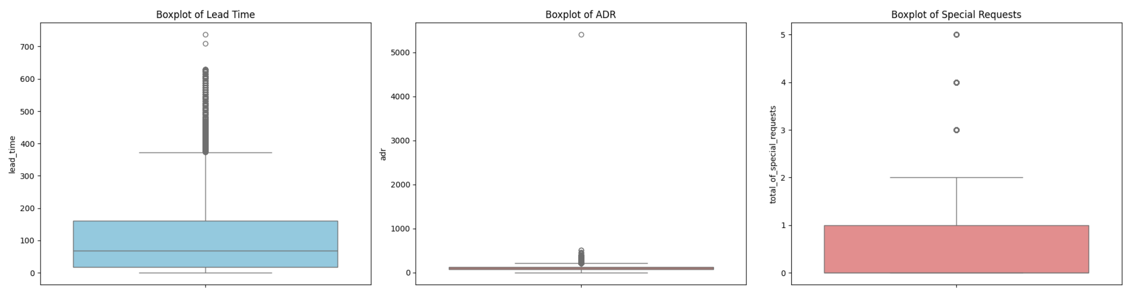


Observations: - lead_time is highly right-skewed, meaning most bookings are made close to the stay date, but a few have extremely long lead times. - adr (average daily rate) has high variability, with a few extreme outliers. - stays_in_week_nights and stays_in_weekend_nights are mostly low values, meaning most bookings are for short stays. - Most customers travel without children or babies, suggesting that family bookings are less common.

Actions: - Cap extreme outliers in lead_time and adr (e.g., above the 99th percentile). - Log-transform lead_time and adr to reduce skewness before training.

[7]:
```
# Boxplots to detect outliers
plt.figure(figsize=(20, 10))

# Lead Time
plt.subplot(2, 3, 1)
sns.boxplot(y=df["lead_time"], color="skyblue")
plt.title("Boxplot of Lead Time")
```

```python
# ADR (Average Daily Rate)
plt.subplot(2, 3, 2)
sns.boxplot(y=df["adr"], color="salmon")
plt.title("Boxplot of ADR")

# Total Special Requests
plt.subplot(2, 3, 3)
sns.boxplot(y=df["total_of_special_requests"], color="lightcoral")
plt.title("Boxplot of Special Requests")

plt.tight_layout()
plt.show()
```

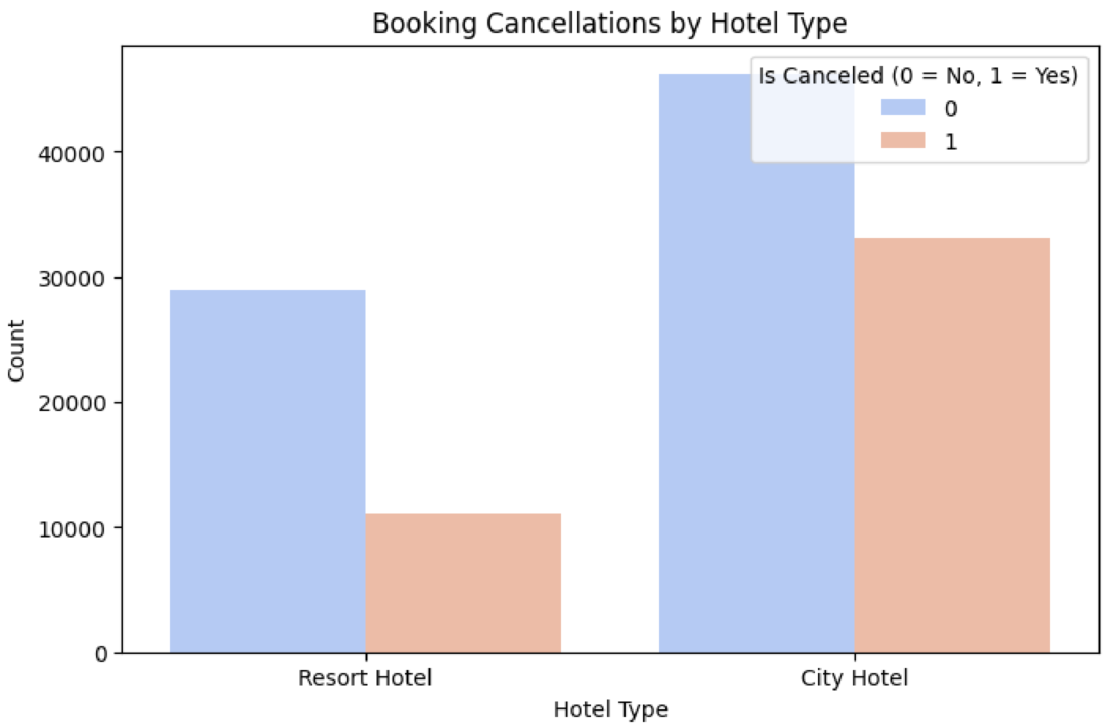


Observations: - Lead time has extreme outliers, with some bookings made 700+ days in advance. - ADR shows high variation, with a few bookings priced extremely high. - Total special requests follow a discrete distribution, meaning they are good categorical features.

Actions: - Cap extreme outliers in lead_time and adr to improve model stability. - Consider converting total_special_requests into a binary feature (e.g., High vs. Low Requests).

Impact of Key Features on Cancellations Booking Cancellations by Hotel Type

[8]:
```python
plt.figure(figsize=(8, 5))
sns.countplot(x='hotel', hue='is_canceled', data=df, palette='coolwarm')
plt.title('Booking Cancellations by Hotel Type')
plt.xlabel('Hotel Type')
plt.ylabel('Count')
plt.legend(title='Is Canceled (0 = No, 1 = Yes)', loc='upper right')
plt.show()
```

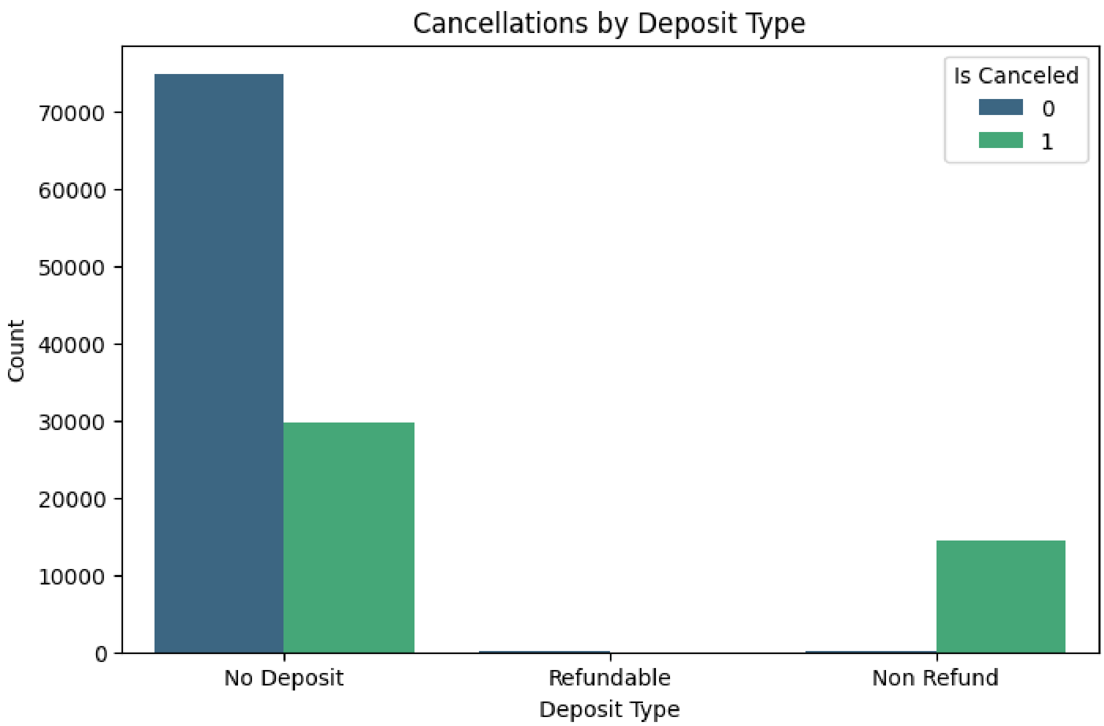


Observations: - City hotels experience a much higher cancellation rate than resort hotels. - Resort hotels have significantly fewer cancellations, likely due to longer stays and vacation planning stability. - City hotels face more last-minute cancellations, possibly due to business travelers changing plans.

Actions: - Create a binary feature (is_city_hotel) - Assign 1 for City Hotel, 0 for Resort Hotel. - Investigate whether city hotels have higher lead times, leading to more cancellations.

Cancellations by Deposit Type

[9]:
```
plt.figure(figsize=(8, 5))
sns.countplot(x='deposit_type', hue='is_canceled', data=df, palette='viridis')
plt.title('Cancellations by Deposit Type')
plt.xlabel('Deposit Type')
plt.ylabel('Count')
plt.legend(title='Is Canceled', loc='upper right')
plt.show()
```

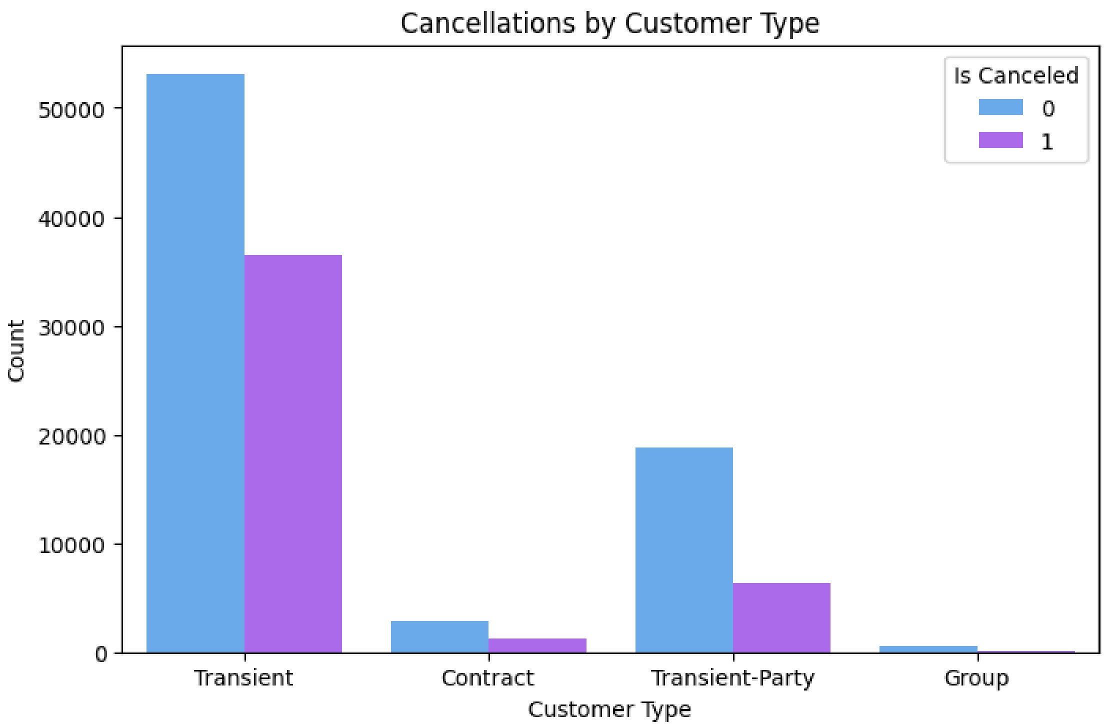


Observations: - Bookings with "No Deposit" have the highest cancellation rate. - Refundable deposits also have high cancellation rates, likely due to the lack of financial commitment. - Non-refundable bookings show the lowest cancellations, as customers are less likely to forfeit payments.

Actions: - Create a feature flag (is_non_refundable) to capture this pattern. - Consider giving more weight to non-refundable deposits in model training, as these bookings are less likely to cancel.

Cancellations by Customer Type

[10]:
```
plt.figure(figsize=(8, 5))
sns.countplot(x='customer_type', hue='is_canceled', data=df, palette='cool')
plt.title('Cancellations by Customer Type')
plt.xlabel('Customer Type')
plt.ylabel('Count')
plt.legend(title='Is Canceled', loc='upper right')
plt.show()
```

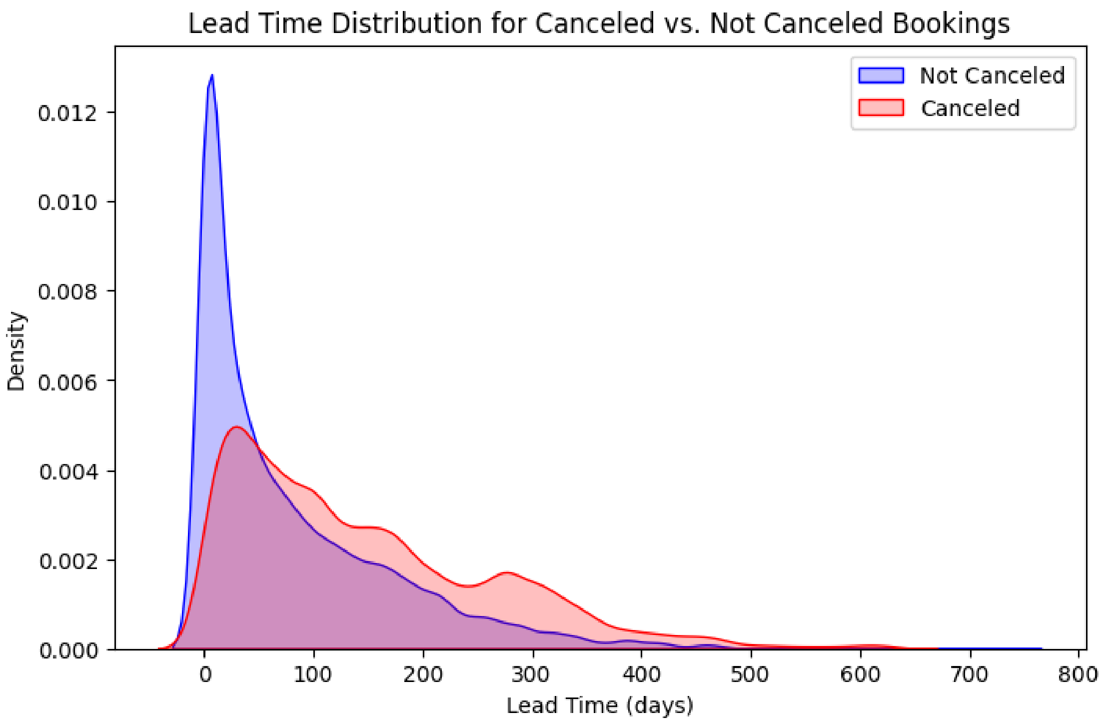


Observations: - Transient customers have the highest cancellation rate, indicating that short-term, one-time customers are less reliable. How do we retain first time customers ? - Contract customers have the lowest cancellation rate, likely because they involve pre-agreed corporate bookings.

Actions: - Create a binary feature (is_transient) to indicate high-risk customer types. - Assign higher weights to transient cancellations during model training.

Lead Time vs. Cancellations

[11]:
```
plt.figure(figsize=(8, 5))
sns.kdeplot(df.loc[df["is_canceled"] == 0, "lead_time"], label="Not Canceled",␣
↪shade=True, color="blue")
sns.kdeplot(df.loc[df["is_canceled"] == 1, "lead_time"], label="Canceled",␣
↪shade=True, color="red")
plt.title("Lead Time Distribution for Canceled vs. Not Canceled Bookings")
plt.xlabel("Lead Time (days)")
plt.ylabel("Density")
plt.legend()
plt.show()
```

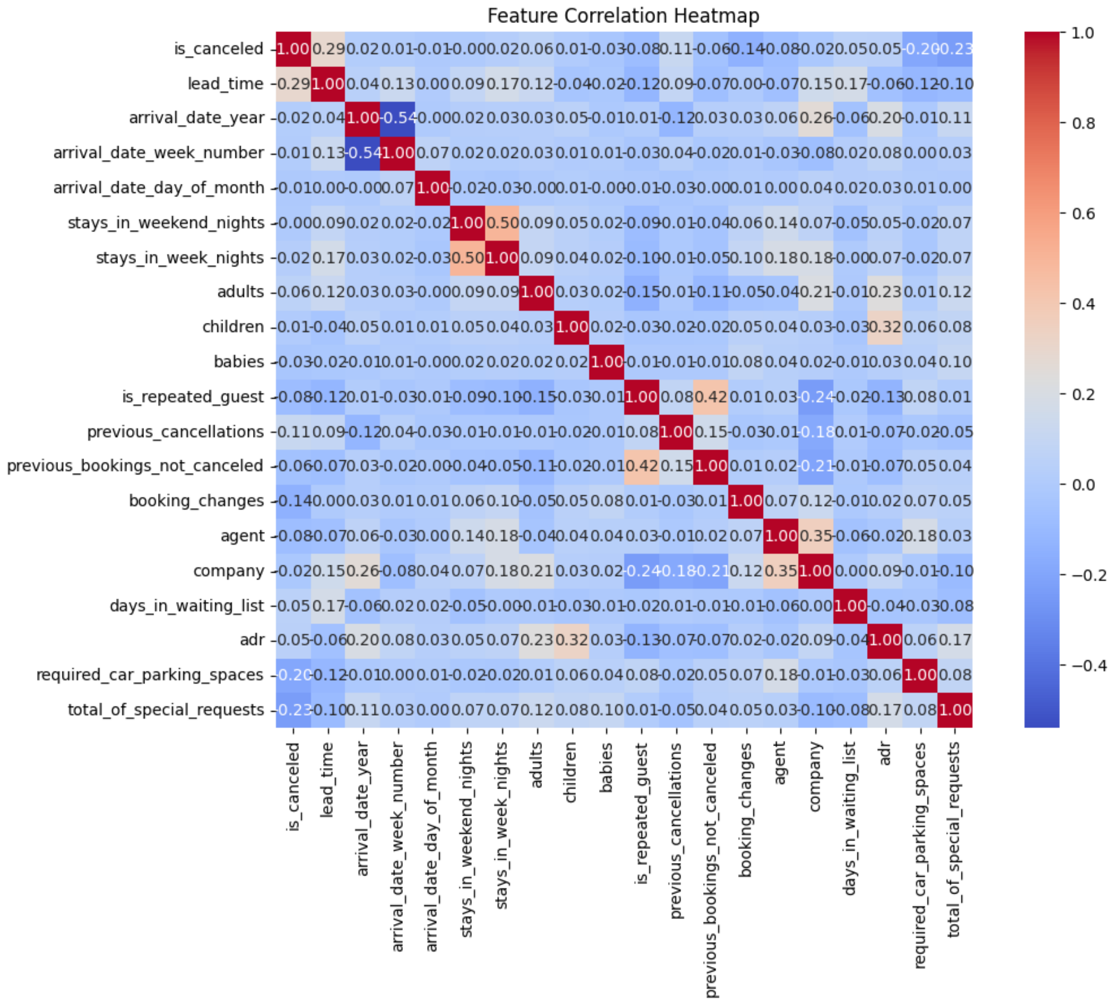


Observations: - Bookings with long lead times have a much higher probability of cancellation. - Short lead times are more common for non-canceled bookings, indicating last-minute travelers are more likely to follow through.

Actions: - Cap lead_time at the 99th percentile to remove extreme values. - Apply log transformation to lead_time to reduce skewness. - Create a categorical feature (lead_time_category) to group bookings into short, medium, and long lead times.

Correlation Heatmap

[12]:
```
# Select only numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns

# Compute correlation matrix
correlation_matrix = df[numerical_columns].corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",␣
  ↪square=True)
plt.title("Feature Correlation Heatmap")
plt.show()
```



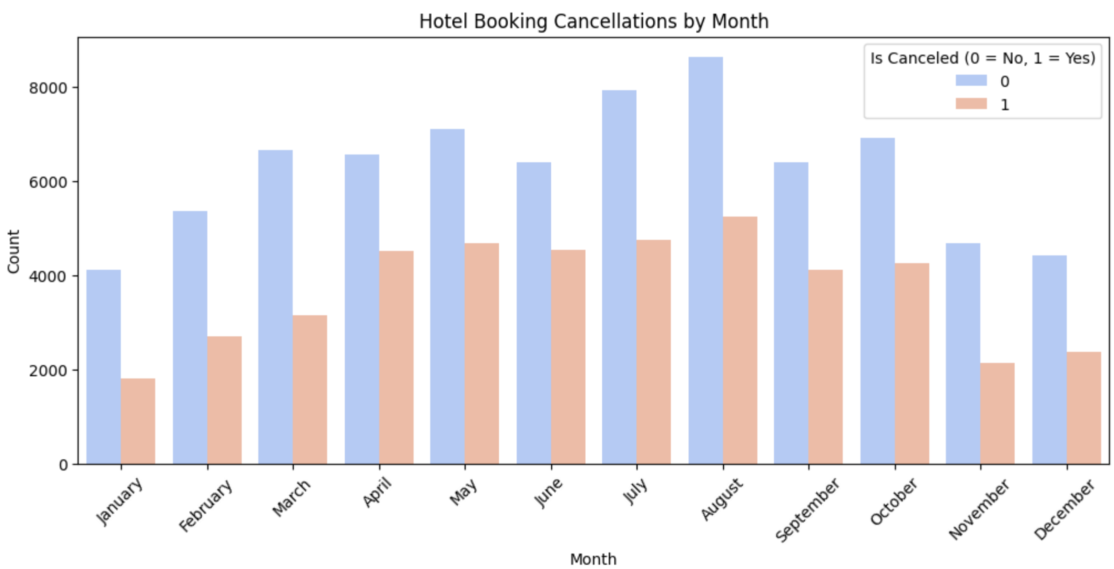
Observations: - Lead time has a strong positive correlation with cancellations (-0.29), confirming that longer lead times increase cancellation likelihood. - Previous cancellations correlate with future cancellations (-0.11), meaning past behavior predicts future actions. - ADR has little correlation with cancellations (-0.05), meaning price alone is not a strong predictor.

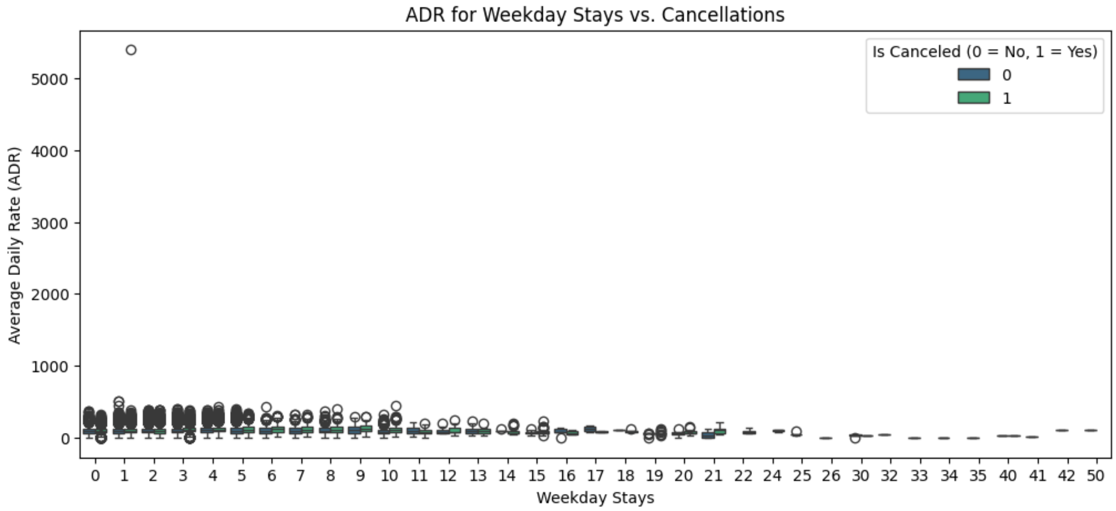
Actions: - Keep lead_time and previous_cancellations as critical features. - ADR may not be as useful alone but could interact with other variables.

Time-Based Trends (Seasonality in Cancellations)

[13]:
```python
# Monthly Booking Cancellations Trend
plt.figure(figsize=(12, 5))
sns.countplot(x="arrival_date_month", hue="is_canceled", data=df, palette="coolwarm",
              order=['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
                     'September', 'October', 'November', 'December'])
```

```
plt.title("Hotel Booking Cancellations by Month")
plt.xlabel("Month")
plt.ylabel("Count")
plt.legend(title="Is Canceled (0 = No, 1 = Yes)")
plt.xticks(rotation=45)
plt.show()

# Weekday vs Weekend Stays & Cancellation Trends
plt.figure(figsize=(12, 5))
sns.boxplot(x="stays_in_week_nights", y="adr", hue="is_canceled", data=df,
    palette="viridis")
plt.title("ADR for Weekday Stays vs. Cancellations")
plt.xlabel("Weekday Stays")
plt.ylabel("Average Daily Rate (ADR)")
plt.legend(title="Is Canceled (0 = No, 1 = Yes)")
plt.show()
```





Observations: - July and August show the highest cancellations, likely due to seasonal demand fluctuations. - December and January have fewer cancellations, possibly due to holiday travel stability.

Actions: - Create a high_season feature to mark peak cancellation months. - Investigate whether pricing policies differ by season to improve prediction accuracy.

Fixing the Data Issues & Feature Engineering

[14]:
```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from scipy.stats import boxcox

# 1. Handle Missing Values
# Drop 'company' since it has excessive missing values
df.drop(columns=['company'], inplace=True)

# Fill missing 'children' with median value
df['children'].fillna(df['children'].median(), inplace=True)

# Fill missing 'country' with most common value (mode)
df['country'].fillna(df['country'].mode()[0], inplace=True)

# Drop remaining rows with missing values (if minimal)
df.dropna(inplace=True)
```

[15]:
```
# 2. Cap Outliers in Lead Time and ADR
# Define the 99th percentile caps
lead_time_cap = df['lead_time'].quantile(0.99)
adr_cap = df['adr'].quantile(0.99)
```

```
# Apply capping
df['lead_time'] = np.where(df['lead_time'] > lead_time_cap, lead_time_cap,␣
↪df['lead_time'])
df['adr'] = np.where(df['adr'] > adr_cap, adr_cap, df['adr'])
```

[16]:
```
# 3. Apply Log Transformation for Skewed Distributions
df['lead_time_log'] = np.log1p(df['lead_time'])
df['adr_log'] = np.log1p(df['adr'])
```

[17]:
```
# 4. Create New Features
# Create a binary feature for City vs. Resort Hotel
df['is_city_hotel'] = np.where(df['hotel'] == 'City Hotel', 1, 0)

# Create a feature flag for non-refundable deposits
df['is_non_refundable'] = np.where(df['deposit_type'] == 'Non Refund', 1, 0)

# Create a feature for transient customers (high-risk cancellations)
df['is_transient'] = np.where(df['customer_type'] == 'Transient', 1, 0)

from sklearn.preprocessing import OrdinalEncoder

# Create categorical bins for lead time
df['lead_time_category'] = pd.cut(df['lead_time'], bins=[0, 30, 90, 365],␣
↪labels=['Short', 'Medium', 'Long'])

# Fill missing values (default to 'Medium')
df['lead_time_category'] = df['lead_time_category'].fillna('Medium')

# Apply Ordinal Encoding for `lead_time_category`
ordinal_encoder = OrdinalEncoder(categories=[['Short', 'Medium', 'Long']])
df['lead_time_category'] = ordinal_encoder.
↪fit_transform(df[['lead_time_category']])

print("\nSuccessfully encoded `lead_time_category` before train-test split.")

# Confirm changes
print("\nFinal Data Overview After Preprocessing:")
print(df.info())
```

```
Successfully encoded `lead_time_category` before train-test split.

Final Data Overview After Preprocessing:
<class 'pandas.core.frame.DataFrame'>
Index: 103050 entries, 3 to 119389
Data columns (total 35 columns):
 #   Column                          Non-Null Count   Dtype
```

```
---  ------                          --------------  -----
 0   hotel                           103050 non-null  object
 1   is_canceled                     103050 non-null  int64
 2   lead_time                       103050 non-null  float64
 3   arrival_date_year               103050 non-null  int64
 4   arrival_date_month              103050 non-null  object
 5   arrival_date_week_number        103050 non-null  int64
 6   arrival_date_day_of_month       103050 non-null  int64
 7   stays_in_weekend_nights         103050 non-null  int64
 8   stays_in_week_nights            103050 non-null  int64
 9   adults                          103050 non-null  int64
 10  children                        103050 non-null  float64
 11  babies                          103050 non-null  int64
 12  meal                            103050 non-null  object
 13  country                         103050 non-null  object
 14  market_segment                  103050 non-null  object
 15  distribution_channel            103050 non-null  object
 16  is_repeated_guest               103050 non-null  int64
 17  previous_cancellations          103050 non-null  int64
 18  previous_bookings_not_canceled  103050 non-null  int64
 19  reserved_room_type              103050 non-null  object
 20  assigned_room_type              103050 non-null  object
 21  booking_changes                 103050 non-null  int64
 22  deposit_type                    103050 non-null  object
 23  agent                           103050 non-null  float64
 24  days_in_waiting_list            103050 non-null  int64
 25  customer_type                   103050 non-null  object
 26  adr                             103050 non-null  float64
 27  required_car_parking_spaces     103050 non-null  int64
 28  total_of_special_requests       103050 non-null  int64
 29  lead_time_log                   103050 non-null  float64
 30  adr_log                         103049 non-null  float64
 31  is_city_hotel                   103050 non-null  int64
 32  is_non_refundable               103050 non-null  int64
 33  is_transient                    103050 non-null  int64
 34  lead_time_category              103050 non-null  float64
dtypes: float64(7), int64(18), object(10)
memory usage: 28.3+ MB
None
```

0.5 PART 3: MODELS

0.5.1 Model Selection & Training

Train-Test Split (Before Encoding)

[18]:
```python
from sklearn.model_selection import train_test_split

# Define features and target
```

```python
X = df.drop(columns=["is_canceled"])
y = df["is_canceled"]

# Train-test split (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)

print(f"Training Set Shape: {X_train.shape}, Testing Set Shape: {X_test.shape}")
```

```
Training Set Shape: (82440, 34), Testing Set Shape: (20610, 34)
```

Apply Encoding Only After Splitting

[19]:
```python
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder

# Define categorical columns
categorical_cols = ['hotel', 'arrival_date_month', 'meal', 'country',
                    'market_segment', 'distribution_channel',
                    'reserved_room_type', 'assigned_room_type']

ordinal_cols = ['deposit_type', 'customer_type']

# One-Hot Encoding for non-ordinal categorical features
encoder = OneHotEncoder(drop="first", sparse_output=False,␣
↪handle_unknown="ignore")
X_train_encoded = encoder.fit_transform(X_train[categorical_cols])
X_test_encoded = encoder.transform(X_test[categorical_cols])

# Convert encoded features into DataFrames
X_train_encoded_df = pd.DataFrame(X_train_encoded, columns=encoder.
↪get_feature_names_out(categorical_cols))
X_test_encoded_df = pd.DataFrame(X_test_encoded, columns=encoder.
↪get_feature_names_out(categorical_cols))

# Ordinal Encoding for features with an inherent order
ordinal_encoder = OrdinalEncoder()
X_train[ordinal_cols] = ordinal_encoder.fit_transform(X_train[ordinal_cols])
X_test[ordinal_cols] = ordinal_encoder.transform(X_test[ordinal_cols])

# Drop original categorical columns and merge encoded data
X_train = X_train.drop(columns=categorical_cols).reset_index(drop=True)
X_test = X_test.drop(columns=categorical_cols).reset_index(drop=True)

X_train = pd.concat([X_train, X_train_encoded_df], axis=1)
X_test = pd.concat([X_test, X_test_encoded_df], axis=1)

print("\nCategorical Features Successfully Encoded for Logistic Regression &␣
↪KNN.")
```

```
Categorical Features Successfully Encoded for Logistic Regression & KNN.
```

[20]:
```python
# Check dataset structure
print("Training Data Types:\n", X_train.dtypes)
print("\nTesting Data Types:\n", X_test.dtypes)
```

```
Training Data Types:
 lead_time                     float64
arrival_date_year               int64
arrival_date_week_number        int64
arrival_date_day_of_month       int64
stays_in_weekend_nights         int64
                               …
assigned_room_type_F          float64
assigned_room_type_G          float64
assigned_room_type_H          float64
assigned_room_type_I          float64
assigned_room_type_K          float64
Length: 236, dtype: object

Testing Data Types:
 lead_time                     float64
arrival_date_year               int64
arrival_date_week_number        int64
arrival_date_day_of_month       int64
stays_in_weekend_nights         int64
                               …
assigned_room_type_F          float64
assigned_room_type_G          float64
assigned_room_type_H          float64
assigned_room_type_I          float64
assigned_room_type_K          float64
Length: 236, dtype: object
```

there are some missing values after encoding, lets handle them

[21]:
```python
from sklearn.impute import SimpleImputer

# Define imputer (using median for numerical columns)
imputer = SimpleImputer(strategy='median')

# Apply imputation separately for training and testing sets
X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.
 ↪columns)
X_test_imputed = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)

# Confirm missing values are handled
```

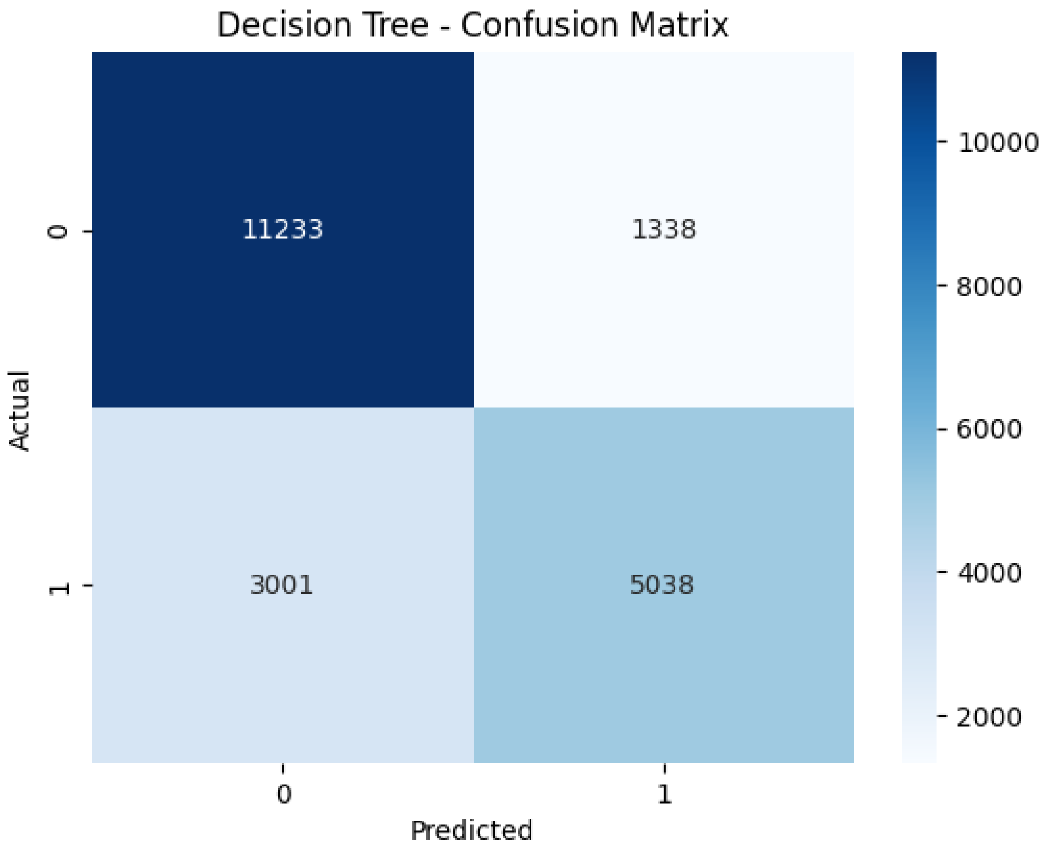
```
print("Missing values in X_train after imputation:\n", X_train_imputed.isnull().
 ↪sum().sum())
print("Missing values in X_test after imputation:\n", X_test_imputed.isnull().
 ↪sum().sum())

# Replace X_train and X_test with imputed versions
X_train, X_test = X_train_imputed, X_test_imputed
```

```
Missing values in X_train after imputation:
 0
Missing values in X_test after imputation:
 0
```

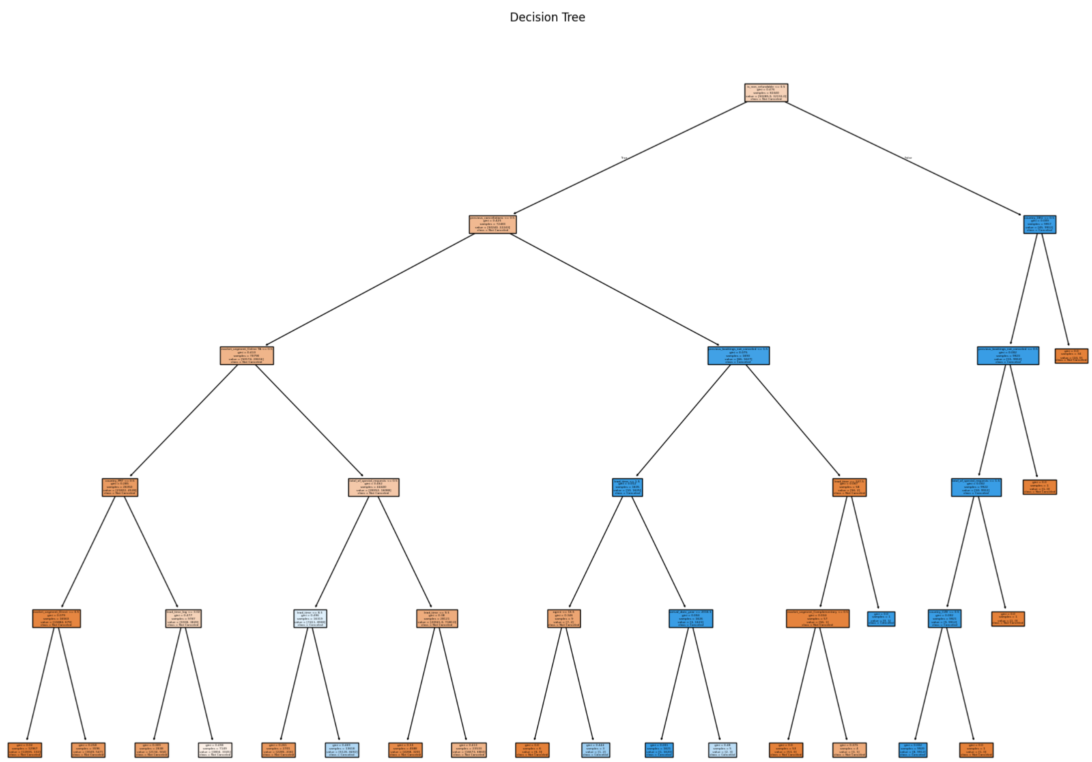
Decision Tree Classifier

[22]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix

# Train Decision Tree Classifier
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)

# Predictions
y_pred_dt = dt_model.predict(X_test)

# Evaluate Decision Tree
print("\nDecision Tree Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt))
print("Recall:", recall_score(y_test, y_pred_dt))
print("F1 Score:", f1_score(y_test, y_pred_dt))

# Confusion Matrix
cm_dt = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm_dt, annot=True, cmap='Blues', fmt='d')
plt.title("Decision Tree - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Decision Tree Performance:
Accuracy: 0.7894711305191654
Precision: 0.7901505646173149
Recall: 0.6266948625450927
F1 Score: 0.6989941033645508
```



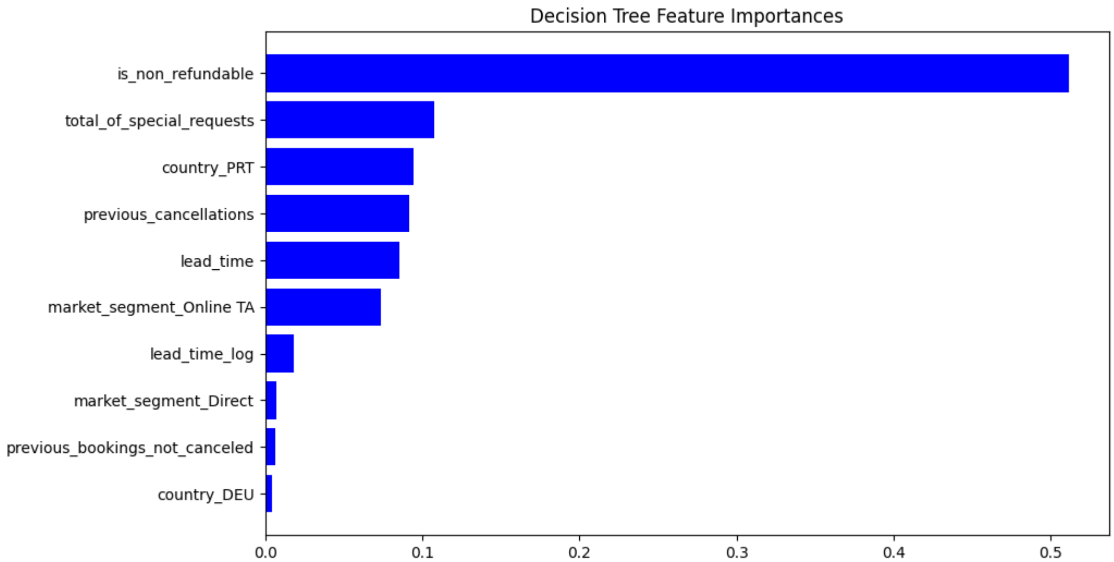
[23]:
```
# Ploting the tree
plt.figure(figsize=(21, 15))
plot_tree(dt_model, filled=True, feature_names=X_train.columns,␣
  ↪class_names=["Not Canceled", "Canceled"])
plt.title("Decision Tree")
plt.show()
```



[24]:
```python
#Lets plot a graph of Feature importance
importances = dt_model.feature_importances_
indices = np.argsort(importances)[-10:]
features = X_train.columns[indices]
plt.figure(figsize=(10, 6))
plt.title("Decision Tree Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="b", align="center")
plt.yticks(range(len(indices)), features)
```

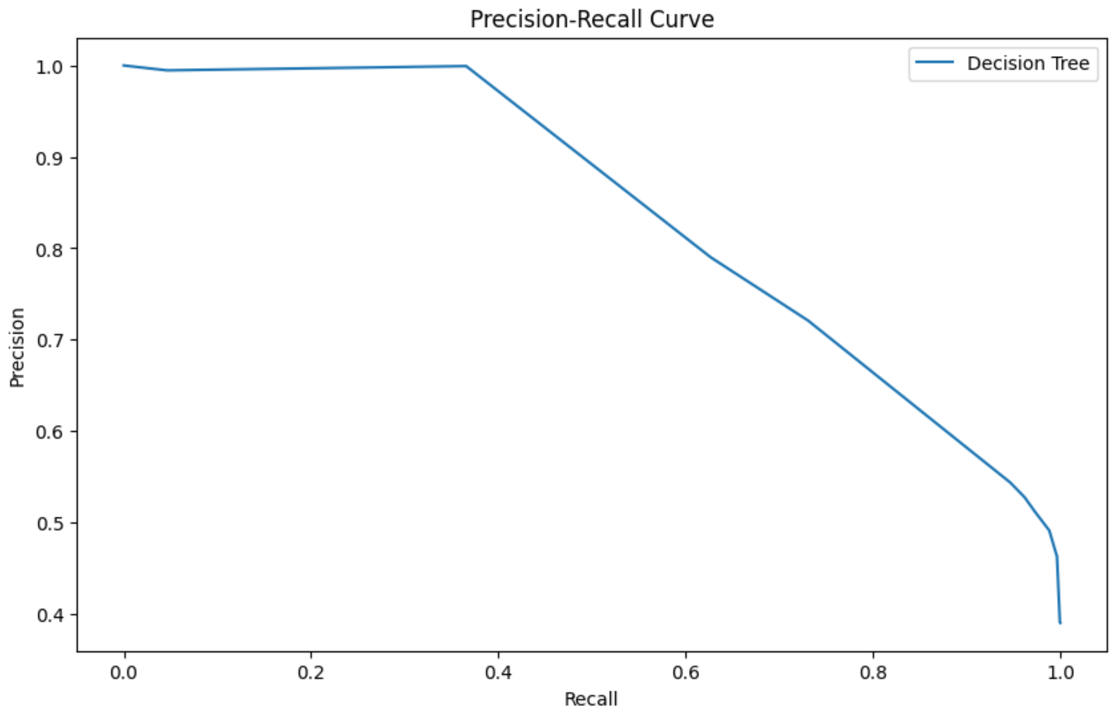
[24]:
```
([<matplotlib.axis.YTick at 0x78c37a444b50>,
  <matplotlib.axis.YTick at 0x78c3738b0510>,
  <matplotlib.axis.YTick at 0x78c374bcf6d0>,
  <matplotlib.axis.YTick at 0x78c3738bf950>,
  <matplotlib.axis.YTick at 0x78c373925b10>,
  <matplotlib.axis.YTick at 0x78c373925590>,
  <matplotlib.axis.YTick at 0x78c37390cc50>,
  <matplotlib.axis.YTick at 0x78c37390e810>,
  <matplotlib.axis.YTick at 0x78c3738d0c90>,
  <matplotlib.axis.YTick at 0x78c3738d0c50>],
 [Text(0, 0, 'country_DEU'),
  Text(0, 1, 'previous_bookings_not_canceled'),
```

```
 Text(0, 2, 'market_segment_Direct'),
 Text(0, 3, 'lead_time_log'),
 Text(0, 4, 'market_segment_Online TA'),
 Text(0, 5, 'lead_time'),
 Text(0, 6, 'previous_cancellations'),
 Text(0, 7, 'country_PRT'),
 Text(0, 8, 'total_of_special_requests'),
 Text(0, 9, 'is_non_refundable')])
```



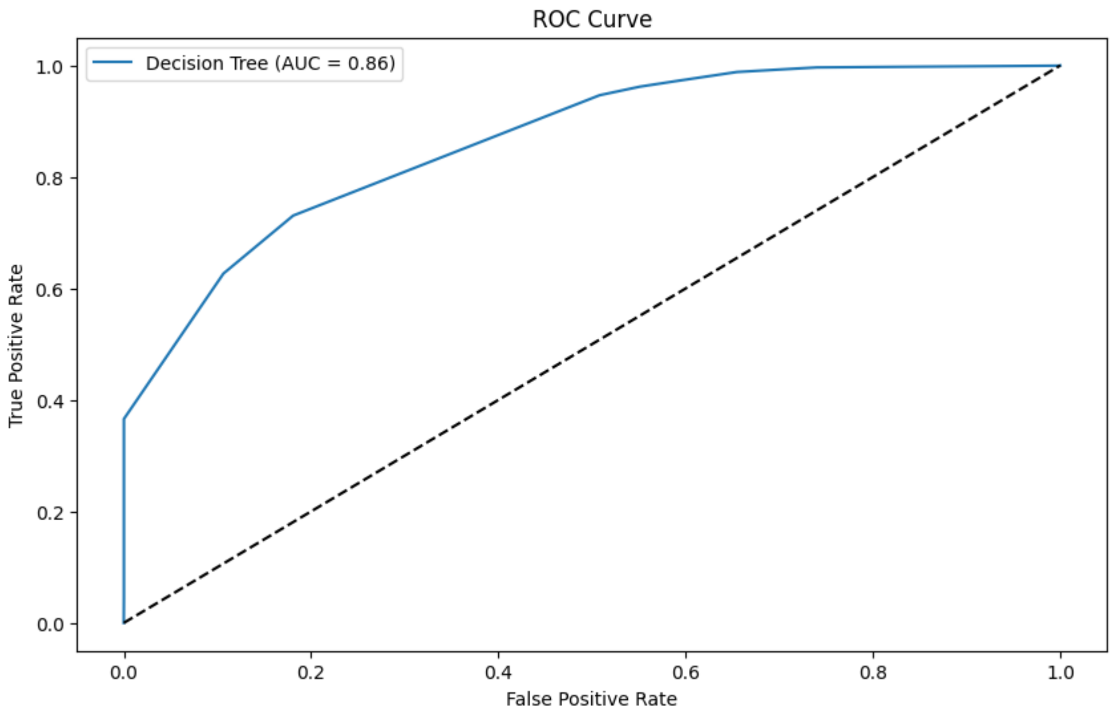
[25]:
```
# plot precision-recall curve
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

y_scores_dt = dt_model.predict_proba(X_test)[:, 1]
precision_dt, recall_dt, thresholds_dt = precision_recall_curve(y_test,␣
 ↪y_scores_dt)

plt.figure(figsize=(10, 6))
plt.plot(recall_dt, precision_dt, label='Decision Tree')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
```

[25]:
```
<matplotlib.legend.Legend at 0x78c374a93a10>
```



[26]:

```
# Decision Tree ROC Curve
from sklearn.metrics import roc_curve, auc

fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test, y_scores_dt)
roc_auc_dt = auc(fpr_dt, tpr_dt)

plt.figure(figsize=(10, 6))
plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {roc_auc_dt:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



Decision Tree Results Discusion

- Accuracy: 78.9%
- Precision: 79.0%
- Recall: 62.7%
- F1 Score: 69.9%

Observations

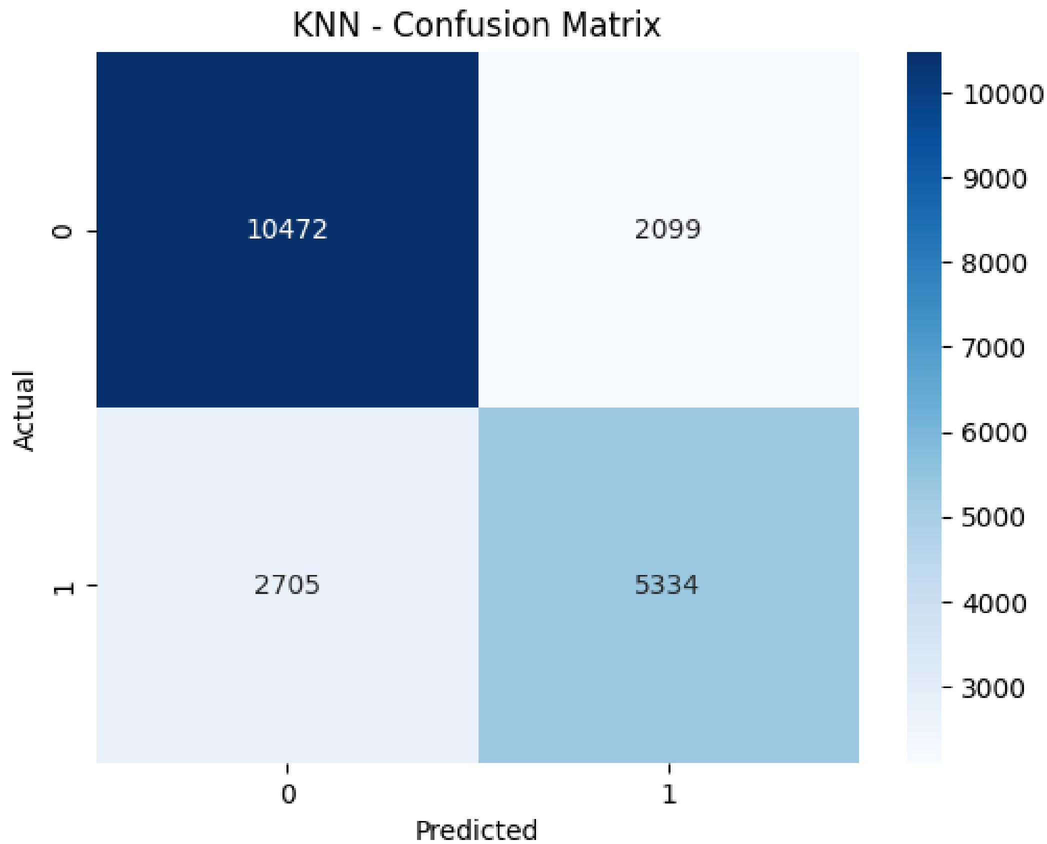
- Moderate accuracy and precision indicate the model captures general cancellation patterns well.
- Recall is relatively low (62.7%), meaning the model misses a significant number of actual cancellations.
- The confusion matrix shows a high number of false negatives, suggesting the model struggles with correctly predicting cancellations.
- *Tuning hyperparameters (e.g., max depth, min samples split) might improve recall.*

K-Nearest Neighbors (KNN)

[27]:
```
from sklearn.neighbors import KNeighborsClassifier

# Train KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

```python
# Predictions
y_pred_knn = knn_model.predict(X_test)

# Evaluate KNN
print("\nK-Nearest Neighbors Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn))
print("Recall:", recall_score(y_test, y_pred_knn))
print("F1 Score:", f1_score(y_test, y_pred_knn))

# Confusion Matrix
cm_knn = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm_knn, annot=True, cmap='Blues', fmt='d')
plt.title("KNN - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
K-Nearest Neighbors Performance:
Accuracy: 0.7669092673459486
Precision: 0.7176106551863313
Recall: 0.6635153626072895
F1 Score: 0.6895036194415719
```



KNN Result Discusion

- Accuracy: 76.7%
- Precision: 71.8%
- Recall: 66.4%
- F1 Score: 68.9%

Observations: - KNN performed worse than Decision Trees in terms of accuracy and recall. - Higher recall (66.4%) than Decision Tree, meaning it identifies slightly more actual cancellations. - Still has many false positives and false negatives, making it less reliable.

- KNN is not ideal for large datasets due to computational inefficiency.

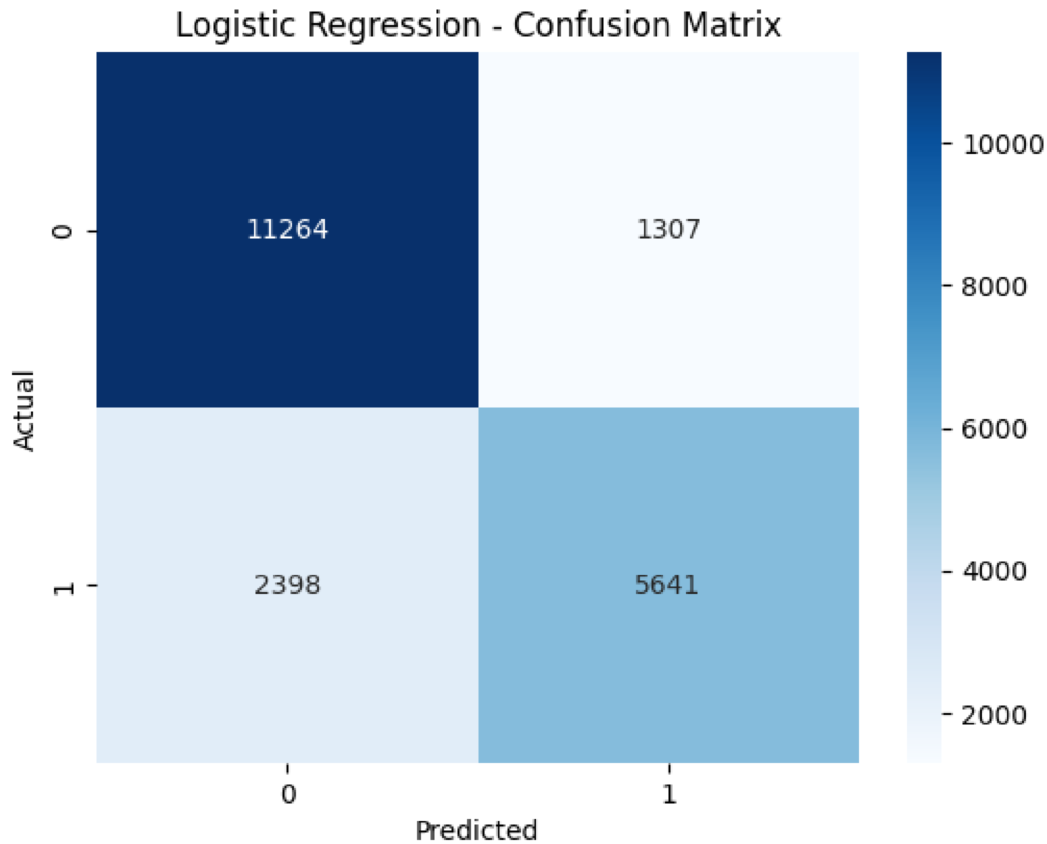
- Try reducing dimensionality using PCA or feature selection.

Logistic Regression

[28]:
```
from sklearn.linear_model import LogisticRegression

# Train Logistic Regression model
log_reg = LogisticRegression(solver='liblinear')
log_reg.fit(X_train, y_train)
```

```python
# Predictions
y_pred_log_reg = log_reg.predict(X_test)

# Evaluate Logistic Regression
print("\nLogistic Regression Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_log_reg))
print("Precision:", precision_score(y_test, y_pred_log_reg))
print("Recall:", recall_score(y_test, y_pred_log_reg))
print("F1 Score:", f1_score(y_test, y_pred_log_reg))

# Confusion Matrix
cm_log_reg = confusion_matrix(y_test, y_pred_log_reg)
sns.heatmap(cm_log_reg, annot=True, cmap='Blues', fmt='d')
plt.title("Logistic Regression - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Logistic Regression Performance:
Accuracy: 0.8202328966521106
Precision: 0.81188831318365
Recall: 0.7017041920636895
F1 Score: 0.7527857476479616
```



Logistic Regression Results Discussion - Accuracy: 82.0% - Precision: 81.2% - Recall: 70.2% - F1 Score: 75.3%

Observations: - Balanced performance across all metrics. - Better recall (70.2%) than Decision Trees, meaning it identifies more actual cancellations. - Still misses 30% of actual cancellations, which may not be acceptable in a high-stakes business setting.

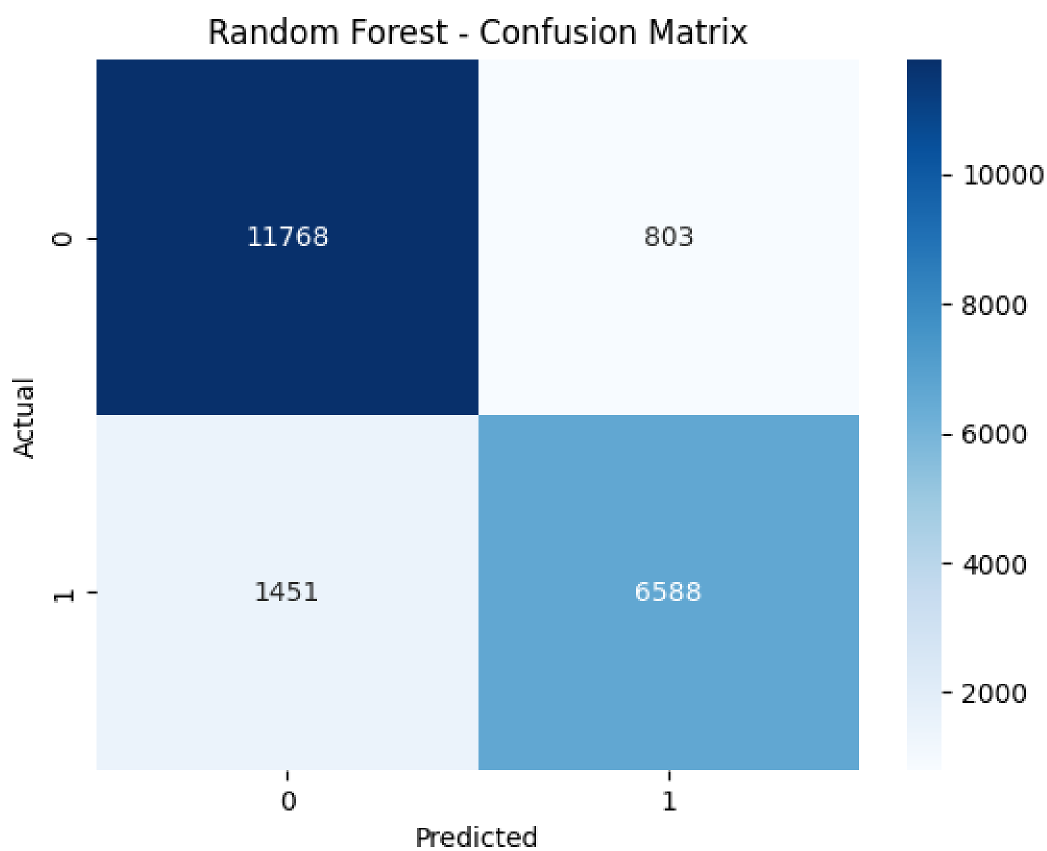
- *Feature selection using RFE or L1 regularization may improve performance.*

Random Forest Classifier

[29]:
```
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate Random Forest
print("\nRandom Forest Performance:")
```

```python
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1 Score:", f1_score(y_test, y_pred_rf))

# Confusion Matrix
cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, cmap='Blues', fmt='d')
plt.title("Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Random Forest Performance:
Accuracy: 0.8906356137797186
Precision: 0.8913543498849953
Recall: 0.819504913546461
F1 Score: 0.8539209332469215
```



Random Forest Classifier Results Discussion - Accuracy: 89.1% - Precision: 89.1% - Recall:

81.9% - F1 Score: 85.4%

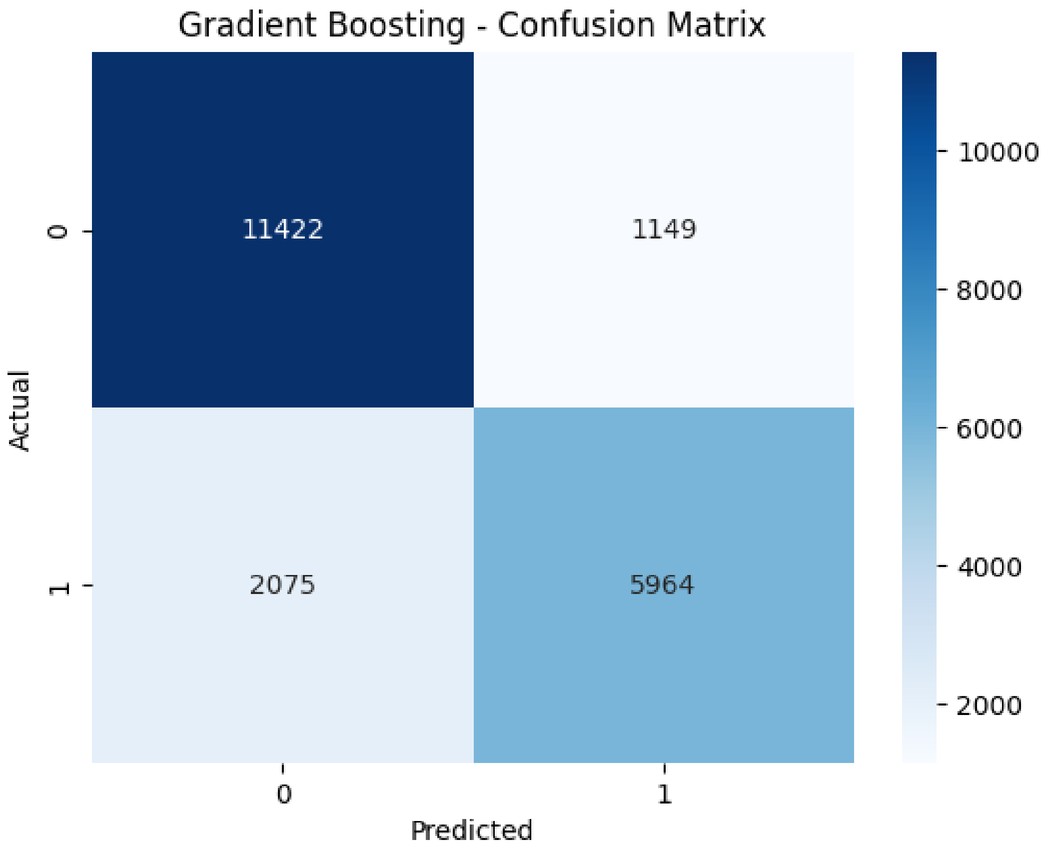
Observations: - Highest accuracy of all models (89.1%), meaning it makes the most correct predictions. - High recall (81.9%), meaning it captures most cancellations correctly. - Confusion matrix shows a lower number of false positives and false negatives, making it the most reliable model.

Quick action - *Tune hyperparameters using Grid Search to push performance even higher.*

Gradient Boosting Classifier

[30]:
```python
from sklearn.ensemble import GradientBoostingClassifier

# Train Gradient Boosting model
gb_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,
    random_state=42)
gb_model.fit(X_train, y_train)

# Predictions
y_pred_gb = gb_model.predict(X_test)

# Evaluate Gradient Boosting
print("\nGradient Boosting Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_gb))
print("Precision:", precision_score(y_test, y_pred_gb))
print("Recall:", recall_score(y_test, y_pred_gb))
print("F1 Score:", f1_score(y_test, y_pred_gb))

# Confusion Matrix
cm_gb = confusion_matrix(y_test, y_pred_gb)
sns.heatmap(cm_gb, annot=True, cmap='Blues', fmt='d')
plt.title("Gradient Boosting - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Gradient Boosting Performance:
Accuracy: 0.8435710819990296
Precision: 0.8384647827920708
Recall: 0.7418833188207489
F1 Score: 0.7872228088701162
```



Gradient Boosting Classifier Results Discussion - Accuracy: 84.4% - Precision: 83.8% - Recall: 74.2% - F1 Score: 78.7%

Observations: - Second-best performance after Random Forest. - Recall (74.2%) is higher than Logistic Regression, meaning it captures more actual cancellations. - Slightly lower F1-score than Random Forest, meaning it might not generalize as well.

Improvement

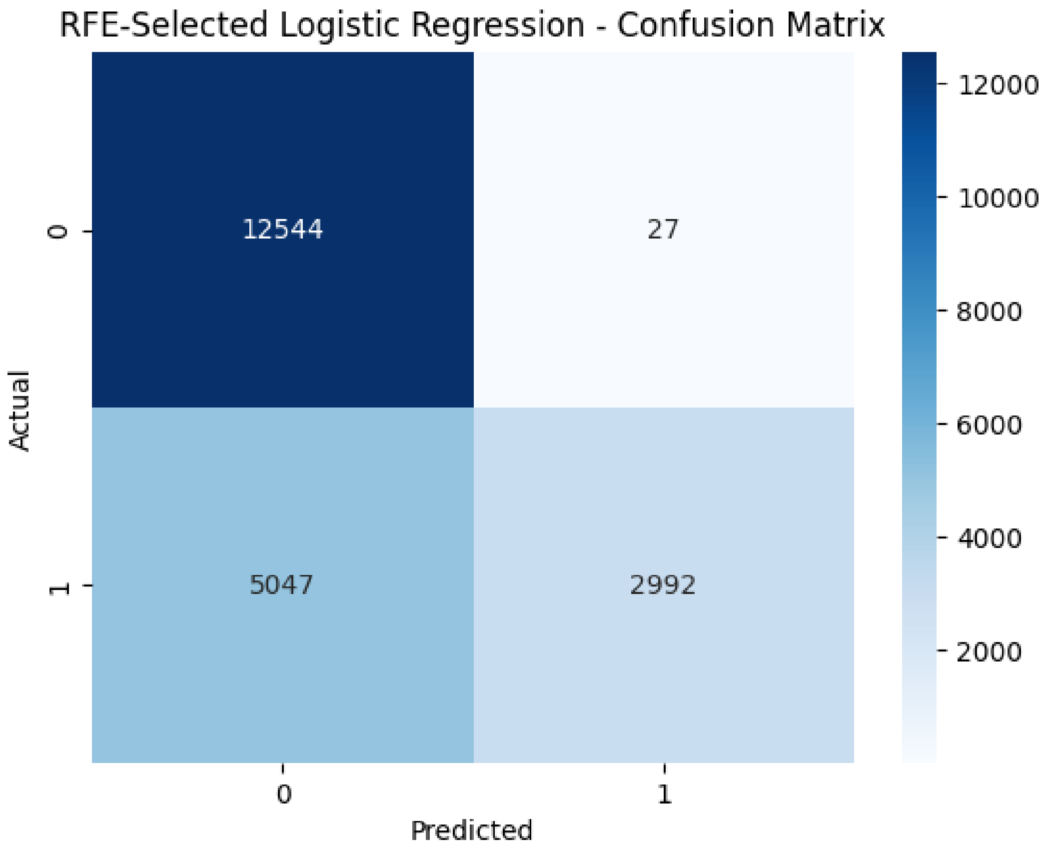
- *Tuning hyperparameters (learning rate, max depth) might improve recall.*

Recursive Feature Elimination (RFE)

[31]:
```
from sklearn.feature_selection import RFE

# Use Recursive Feature Elimination with Logistic Regression
rfe_selector = RFE(estimator=LogisticRegression(solver='liblinear'),␣
  ↪n_features_to_select=10)
X_train_rfe = rfe_selector.fit_transform(X_train, y_train)
X_test_rfe = rfe_selector.transform(X_test)

# Train Logistic Regression on selected features
```

```python
log_reg_rfe = LogisticRegression(solver='liblinear')
log_reg_rfe.fit(X_train_rfe, y_train)

# Predictions
y_pred_rfe = log_reg_rfe.predict(X_test_rfe)

# Evaluate RFE-enhanced model
print("\nRFE-Selected Logistic Regression Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_rfe))
print("Precision:", precision_score(y_test, y_pred_rfe))
print("Recall:", recall_score(y_test, y_pred_rfe))
print("F1 Score:", f1_score(y_test, y_pred_rfe))

# Confusion Matrix
cm_rfe = confusion_matrix(y_test, y_pred_rfe)
sns.heatmap(cm_rfe, annot=True, cmap='Blues', fmt='d')
plt.title("RFE-Selected Logistic Regression - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Display selected features
selected_features = X_train.columns[rfe_selector.support_]
print("\nFeatures Selected by RFE:", selected_features.tolist())
```

```
RFE-Selected Logistic Regression Performance:
Accuracy: 0.7538088306647258
Precision: 0.9910566412719444
Recall: 0.3721855952232865
F1 Score: 0.5411466811358293
```




```
Features Selected by RFE: ['previous_cancellations',
'previous_bookings_not_canceled', 'required_car_parking_spaces',
'is_non_refundable', 'country_AGO', 'country_ARE', 'country_HKG', 'country_MAC',
'assigned_room_type_I', 'assigned_room_type_K']
```

** Results Discussion for Recursive Feature Elimination (RFE) with Logistic ** - Accuracy: 75.4% - Precision: 99.1% - Recall: 37.2% - F1 Score: 54.1%

Observations: - Extremely high precision (99.1%), meaning almost every predicted cancellation is correct. - Terrible recall (37.2%), meaning it fails to identify most actual cancellations. - This model is overfitting to non-cancelled bookings.

Improvements

- Feature selection should be reconsidered, as it likely dropped important predictors.

- Reduce number of features carefully using Recursive Feature Elimination (RFE) with cross-validation.

0.5.2 Model Performance Comparison

Accurately predicting hotel booking cancellations is essential for hotels to optimize revenue, manage overbookings, and improve customer retention strategies. Below is a comparative analysis of different models used for this task.

0.5.3 1. Model Performance Overview

Model	Accuracy	Precision	Recall	F1 Score	Key Insights
Decision Tree	**78.9%**	**79.0%**	**62.7%**	**69.9%**	Performs well but misses many actual cancellations.
K-Nearest Neighbors	**76.7%**	**71.8%**	**66.4%**	**68.9%**	Computationally expensive, does not outperform other models.
Logistic Regression	**82.0%**	**81.2%**	**70.2%**	**75.3%**	Strong baseline model, but struggles with complex relationships.
Random Forest	**89.1%**	**89.1%**	**81.9%**	**85.4%**	Best performing model, balances accuracy and recall effectively.
Gradient Boosting	**84.4%**	**83.8%**	**74.2%**	**78.7%**	Strong alternative, but slightly behind Random Forest.
RFE-Logistic Regression	**75.4%**	**99.1%**	**37.2%**	**54.1%**	High precision but extremely poor recall, missing most cancellations.

0.5.4 2. Key Findings & Trade-Offs

- Decision Tree and KNN models underperform in recall and accuracy, making them less suitable for hotel cancellation prediction.

- Logistic Regression serves as a strong baseline model but struggles to capture complex cancellation patterns.

- Random Forest outperforms all models with the highest accuracy (89.1%) and strong recall (81.9%), making it the most reliable choice.

- Gradient Boosting is a strong alternative but does not surpass Random Forest in predictive power.

- RFE-Logistic Regression prioritizes precision but fails in recall (37.2%), making it unreliable for business decision-making.

0.5.5 3. Final Model Selection for Deployment

Best Model: Random Forest Classifier
- Best balance of accuracy, precision, and recall, making it ideal for minimizing revenue loss due to unexpected cancellations.
- Handles complex feature relationships well, improving prediction quality.
- More interpretable than boosting models, allowing business stakeholders to understand key drivers of cancellations.

Alternative Model: Gradient Boosting Classifier
- Useful when recall is the primary focus, ensuring that most cancellations are predicted correctly.
- Computationally more expensive than Random Forest, but still a viable option.

0.6 PART 4: BUSINESS STRATEGY

0.7 Business Management Implications & Recommendations

- Using Random Forest for cancellation prediction will enable hotels to adjust overbooking strategies, manage inventory better, and reduce lost revenue.

- Feature importance analysis should be performed to identify the strongest predictors of cancellations.

- Hotels can implement proactive measures such as adjusting pricing, offering incentives for non-cancelable bookings, or introducing cancellation fees based on risk prediction.

Feature Importances We shall extract Feature Importance from Random Forest model to identify the strongest drivers of cancellations.

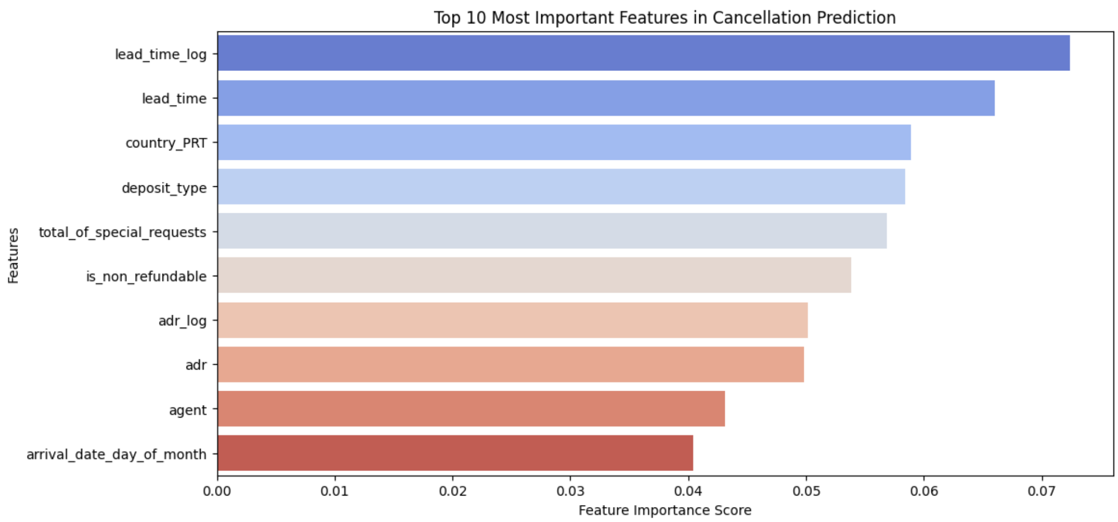
[32]:
```
# Extract feature importance from trained Random Forest model
feature_importances = rf_model.feature_importances_
```

```
# Create a DataFrame for visualization
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Display top 10 important features
print("\nTop 10 Most Important Features in Cancellation Prediction:")
print(feature_importance_df.head(10))
```

```
Top 10 Most Important Features in Cancellation Prediction:
                      Feature  Importance
20               lead_time_log    0.072413
0                    lead_time    0.066039
171                country_PRT    0.058882
13                deposit_type    0.058399
19   total_of_special_requests    0.056828
23            is_non_refundable    0.053861
21                     adr_log    0.050167
17                         adr    0.049812
14                       agent    0.043093
3     arrival_date_day_of_month    0.040440
```

Visualizing Feature Importance

[33]:
```
# Plot Feature Importance
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importance_df["Importance"][:10],
            y=feature_importance_df["Feature"][:10],
            palette="coolwarm")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Top 10 Most Important Features in Cancellation Prediction")
plt.show()
```



Business Recommendations The feature importance analysis from the **Random Forest model** provides key insights into the factors influencing hotel booking cancellations. Below are the top features affecting cancellations, along with **data-driven recommendations** for hotel management.

0.7.1 1. Lead Time (Most Important Factor)

Insight: Longer lead times significantly increase the probability of cancellations, as guests who book far in advance are more likely to change plans.
Recommendation: Implement **dynamic cancellation policies**, making cancellation terms stricter for bookings with long lead times. Introduce **early commitment discounts** to encourage non-refundable bookings for long-term reservations.

0.7.2 2. Non-Refundable Deposits

Insight: Non-refundable deposit bookings drastically reduce cancellations, as financial commitment discourages last-minute changes.
Recommendation: Encourage non-refundable bookings by **offering discounts or added benefits** for prepaid reservations. Promote flexible partial-refund options to balance customer confidence and hotel revenue security.

0.7.3 3. Country-Specific Trends (Portugal)

Insight: Guests from **Portugal (PRT) show a higher likelihood of cancellations**, suggesting regional behavioral trends.
Recommendation: Implement **region-specific policies** such as **additional confirmation steps, stricter deposit requirements, or targeted loyalty programs** to minimize cancellations in high-risk regions.

0.7.4 4. Special Requests

Insight: Guests making **more special requests are less likely to cancel**, as they show higher engagement and commitment to their bookings.
Recommendation: Prioritize responding to **special requests promptly** to enhance guest satisfaction and **increase the likelihood of completed stays**. Offer personalized incentives for guests with special requests.

0.7.5 5. Average Daily Rate (ADR)

Insight: Higher ADR bookings are associated with a **greater probability of cancellations**, likely due to price sensitivity and comparison shopping.
Recommendation: Introduce **price-lock guarantees** or **flexible pricing plans** for high-ADR bookings. Offer loyalty discounts or cashback incentives to **reduce cancellations from price-sensitive customers**.

0.7.6 6. Online Travel Agencies (OTAs) & Booking Agents

Insight: Certain booking agents and **OTA platforms** show **higher cancellation rates**, possibly due to flexible cancellation policies or promotional bookings.
Recommendation: Identify **high-cancellation OTAs** and **negotiate stricter policies**. Shift promotions toward more reliable booking channels and **offer direct booking discounts** to improve revenue predictability.

0.7.7 7. Previous Cancellations

Insight: Guests with **prior cancellations are significantly more likely to cancel again**, indicating a behavioral trend among certain customers.
Recommendation: Flag repeat cancellers and **implement tiered cancellation policies** based on guest history. Encourage responsible booking through **loyalty incentives for non-canceling behavior**.

0.7.8 8. Parking Requests

Insight: Guests requesting **parking spaces are more likely to follow through with their booking**, as parking availability is often tied to planned travel.
Recommendation: Highlight **parking availability as a booking feature** to attract reliable customers. Consider offering **discounted or guaranteed parking** for direct bookings to **reduce cancellation risk**.

0.7.9 Business Strategy Takeaways

- Enforce stricter cancellation terms for high-risk bookings (long lead times, high ADR).
- Strengthen direct booking channels and minimize dependency on high-cancellation OTAs.
- Leverage customer behavior insights (repeat cancellations, parking requests, and special requests) to adjust pricing and policies.

- Personalize booking experiences with incentives that encourage commitment and reduce cancellation likelihood.

By implementing these data driven recommendations, hotels will optimize revenue, reduce uncertainty, and enhance customer retention strategies.