In [ ]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns

clean_data = pd.read_csv(r"C:\Users\Pratik\Desktop\CU Denver Courses\Computing BANA 6620\Codes\Codes\Test\Clean_
data = pd.DataFrame(clean_data)

super_host = pd.read_csv(r"C:\Users\Pratik\Desktop\CU Denver Courses\Computing BANA 6620\Codes\Codes\Test\super_
superdf = pd.DataFrame(super_host)

nonsuper_host = pd.read_csv(r"C:\Users\Pratik\Desktop\CU Denver Courses\Computing BANA 6620\Codes\Codes\Test\no
nsuperdf = pd.DataFrame(nonsuper_host)

num_columns = ['price', 'bathrooms', 'bedrooms', 'beds',
               'host_response_rate', 'host_acceptance_rate', 'host_total_listings_count']



# Summary statistics for numerical columns
print("Summary Statistics:")
superdf[num_columns].describe()
nsuperdf[num_columns].describe()


# Most popular categories ROOM TYPE
superdf['room_type_Hotel room'].sum()
superdf['room_type_Private room'].sum()
superdf['room_type_Shared room'].sum()

nsuperdf['room_type_Hotel room'].sum()
nsuperdf['room_type_Private room'].sum()
nsuperdf['room_type_Shared room'].sum()

superdf['instant_bookable_t'].sum()
nsuperdf['instant_bookable_t'].sum()

# Assuming you have three boolean columns: 'Hotel room', 'private room', and 'shared room'
mask = (superdf['room_type_Hotel room'] == False) & (superdf['room_type_Private room'] == False) & (superdf['ro
count = mask.sum()  # sum() on a boolean Series counts the number of True values
print("Number of rows with all three false:", count)

nsmask = (nsuperdf['room_type_Hotel room'] == False) & (nsuperdf['room_type_Private room'] == False) & (nsuperd
count = nsmask.sum()  # sum() on a boolean Series counts the number of True values
print("Number of rows with all three false:", count)

# ----------PIE CHART FOR ROOM TYPES------------------

# Pie Chart of super vs non-super room types
def determine_room_type(row):
    if row['room_type_Hotel room'] == True:
        return 'Hotel room'
    elif row['room_type_Private room'] == True:
        return 'Private room'
    elif row['room_type_Shared room'] == True:
        return 'Shared room'
    else:
        # If all three are False, it's an entire place
        return 'Entire place'

data['final_room_type'] = data.apply(determine_room_type, axis=1)

# Filter for superhosts and non-superhosts
superhosts_data = data[data['host_is_superhost_t'] == True]
non_superhosts_data = data[data['host_is_superhost_t'] == False]

# Count the occurrences of each final room type
superhost_counts = superhosts_data['final_room_type'].value_counts()
non_superhost_counts = non_superhosts_data['final_room_type'].value_counts()

# Create subplots for side-by-side pie charts
fig, axes = plt.subplots(1, 2, figsize=(16, 8))  # Increase figure size

wedges, texts, autotexts = axes[0].pie(
    superhost_counts,
    autopct='%1.1f%%',
    startangle=90,
    colors=sns.color_palette("Set2"),
    labeldistance=1.2,   # Move labels further out
    pctdistance=1.1,     # Move percentages out as well
```

```
    textprops={'fontsize': 12}
)

axes[0].set_title('Super Host\'s Room Types')
axes[0].axis('equal')

wedges, texts, autotexts = axes[1].pie(
    non_superhost_counts,
    autopct='%1.1f%%',
    startangle=90,
    colors=sns.color_palette("Set2"),
    labeldistance=1.2,
    pctdistance=1.1,
    textprops={'fontsize': 12}
)
axes[1].legend(wedges, non_superhost_counts.index, title="Super Host\'s Room Types", loc="best")
# axes[1].set_title('Non-Super Host\'s Room Types')
axes[1].axis('equal')

plt.tight_layout()
plt.show()


# ------------PRICE BOXPLOT-------------------


palette = sns.color_palette("bright")
plt.figure(figsize=(10,8))
sns.boxplot(x='host_is_superhost_t', y='price', data=data,
            palette=palette, showfliers=False)  # superhosts in red)
plt.title('Super Host vs. Non-Super Host by Prices')
plt.xticks(rotation=45)
plt.show()

#Briana python work
#EDA_setup

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv(r"C:\Users\Pratik\Desktop\CU Denver Courses\Computing BANA 6620\Codes\Codes\Test\Clean_AirBNI
data.head()
data.columns
data.isnull().sum()

# Turn columns with T/F into 1 for True and 0 for False
data[['host_identity_verified_dum_t',
    'room_type_Hotel room',
    'room_type_Private room',
    'room_type_Shared room',
    'instant_bookable_t',
    'host_is_superhost_t']] = data[['host_identity_verified_dum_t',
                                  'room_type_Hotel room',
                                  'room_type_Private room',
                                  'room_type_Shared room',
                                  'instant_bookable_t',
                                  'host_is_superhost_t']].astype(int)

data.head()

# Turning neighborhood_cleansed into  dummy variables
neighborhood_dummies = pd.get_dummies(data['neighbourhood_cleansed'], prefix='neighborhood', drop_first=True)

# Add the new columns to the dataset
data = pd.concat([data, neighborhood_dummies], axis=1)

# Drop the original neighbourhood_cleansed column
#data = data.drop(columns=['neighbourhood_cleansed'])

data.columns
data.head()

# make sure the neighborhood columns are integers
data[['neighborhood_auraria', 'neighborhood_baker', 'neighborhood_barnum',
    'neighborhood_barnumwest', 'neighborhood_bearvalley', 'neighborhood_belcaro', 'neighborhood_berkeley', 'nei
    'neighborhood_cbd', 'neighborhood_chaffeepark', 'neighborhood_cheesmanpark', 'neighborhood_cherrycreek', 'n
    'neighborhood_cityparkwest', 'neighborhood_civiccenter', 'neighborhood_clayton', 'neighborhood_cole', 'neig
    'neighborhood_congresspark', 'neighborhood_corymerrill', 'neighborhood_countryclub', 'neighborhood_dia', 'n
    'neighborhood_fivepoints', 'neighborhood_fortlogan', 'neighborhood_gatewaygreenvalleyranch', 'neighborhood_
    'neighborhood_hale', 'neighborhood_hampden', 'neighborhood_hampdensouth', 'neighborhood_harveypark', 'neighl
    'neighborhood_hilltop', 'neighborhood_indiancreek', 'neighborhood_jeffersonpark', 'neighborhood_lincolnpark
```

```
    'neighborhood_marston', 'neighborhood_montbello', 'neighborhood_montclair', 'neighborhood_northcapitolhill'
    'neighborhood_overland', 'neighborhood_plattpark', 'neighborhood_regis', 'neighborhood_rosedale', 'neighborl
    'neighborhood_southmoorpark', 'neighborhood_southparkhill', 'neighborhood_speer', 'neighborhood_stapleton',
    'neighborhood_university', 'neighborhood_universityhills', 'neighborhood_universitypark', 'neighborhood_val
    'neighborhood_washingtonparkwest', 'neighborhood_washingtonvirginiavale', 'neighborhood_wellshire', 'neighb
    'neighborhood_barnumwest', 'neighborhood_bearvalley', 'neighborhood_belcaro', 'neighborhood_berkeley', 'nei
    'neighborhood_cbd', 'neighborhood_chaffeepark', 'neighborhood_cheesmanpark', 'neighborhood_cherrycreek', 'n
    'neighborhood_cityparkwest', 'neighborhood_civiccenter', 'neighborhood_clayton', 'neighborhood_cole', 'neig
    'neighborhood_congresspark', 'neighborhood_corymerrill', 'neighborhood_countryclub', 'neighborhood_dia', 'n
    'neighborhood_fivepoints', 'neighborhood_fortlogan', 'neighborhood_gatewaygreenvalleyranch', 'neighborhood_
    'neighborhood_hale', 'neighborhood_hampden', 'neighborhood_hampdensouth', 'neighborhood_harveypark', 'neigh
    'neighborhood_hilltop', 'neighborhood_indiancreek', 'neighborhood_jeffersonpark', 'neighborhood_lincolnpark
    'neighborhood_marston', 'neighborhood_montbello', 'neighborhood_montclair', 'neighborhood_northcapitolhill'
    'neighborhood_overland', 'neighborhood_plattpark', 'neighborhood_regis', 'neighborhood_rosedale', 'neighborl
    'neighborhood_southmoorpark', 'neighborhood_southparkhill', 'neighborhood_speer', 'neighborhood_stapleton',
    'neighborhood_university', 'neighborhood_universityhills', 'neighborhood_universitypark', 'neighborhood_val
    'neighborhood_washingtonparkwest', 'neighborhood_washingtonvirginiavale', 'neighborhood_wellshire', 'neighb

data.head()



# Exhibit 3

## EDA: Grouped bar chart of average review scores for superhost and non supershost

# Separate the data into superhosts (1) and non-superhosts (0)
df_scores_SH = data[data['host_is_superhost_t'] == 1][['review_scores_rating', 'review_scores_accuracy',
                                                    'review_scores_cleanliness', 'review_scores_checkin',
                                                    'review_scores_communication', 'review_scores_location',
                                                    'review_scores_value']]
df_scores_nonSH = data[data['host_is_superhost_t'] == 0][['review_scores_rating', 'review_scores_accuracy',
                                                      'review_scores_cleanliness', 'review_scores_checkin',
                                                      'review_scores_communication', 'review_scores_location',
                                                      'review_scores_value']]

# average of each review score category
scores_SH_avg = df_scores_SH.mean()
scores_nonSH_avg = df_scores_nonSH.mean()

# Create a grouped bar chart
categories = scores_SH_avg.index
x = np.arange(len(categories))
width = 0.35
fig, ax = plt.subplots(figsize=(10, 6))

# Plot bars for Superhosts and Non-Superhosts
bars1 = ax.bar(x - width/2, scores_SH_avg, width, label='Superhosts', color='skyblue')
bars2 = ax.bar(x + width/2, scores_nonSH_avg, width, label='Non-Superhosts', color='salmon')

ax.set_xlabel('Review Categories')
ax.set_ylabel('Average Score')
ax.set_title('Average Review Scores by Superhost Status')
ax.set_xticks(x)
ax.set_xticklabels(categories, rotation=45, ha='right')  # Rotate for better readability
ax.legend(title='Host Type', loc='lower left')

# values on top of the bars
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.1f}', xy=(bar.get_x() + bar.get_width()/2, height),
                    xytext=(0, 3), textcoords='offset points', ha='center', fontsize=9)

plt.tight_layout()
plt.show()




# Exhibit 5

## Top 10 Neighborhoods via  k means clustering

from sklearn.cluster import KMeans

# relevant columns for neighborhoods and review scores
neighborhood_columns = ['neighborhood_auraria', 'neighborhood_baker', 'neighborhood_barnum', 'neighborhood_barn
                        'neighborhood_bearvalley', 'neighborhood_belcaro', 'neighborhood_berkeley', 'neighborho
                        'neighborhood_cbd', 'neighborhood_chaffeepark', 'neighborhood_cheesmanpark', 'neighborh
                        'neighborhood_citypark', 'neighborhood_cityparkwest', 'neighborhood_civiccenter', 'neig
                        'neighborhood_cole', 'neighborhood_collegeviewsouthplatte', 'neighborhood_congresspark'
                        'neighborhood_countryclub', 'neighborhood_dia', 'neighborhood_eastcolfax', 'neighborhoo
```

```python
                        'neighborhood_fivepoints', 'neighborhood_fortlogan', 'neighborhood_gatewaygreenvalleyran
                        'neighborhood_goldsmith', 'neighborhood_hale', 'neighborhood_hampden', 'neighborhood_han
                        'neighborhood_harveyparksouth', 'neighborhood_highland', 'neighborhood_hilltop', 'neighb
                        'neighborhood_jeffersonpark', 'neighborhood_lincolnpark', 'neighborhood_lowryfield', 'ne
                        'neighborhood_marston', 'neighborhood_montbello', 'neighborhood_montclair', 'neighborhoo
                        'neighborhood_northeastparkhill', 'neighborhood_northparkhill', 'neighborhood_overland'
                        'neighborhood_regis', 'neighborhood_rosedale', 'neighborhood_rubyhill', 'neighborhood_sl
                        'neighborhood_southmoorpark', 'neighborhood_southparkhill', 'neighborhood_speer', 'neigh
                        'neighborhood_sunnyside', 'neighborhood_sunvalley', 'neighborhood_unionstation', 'neighb
                        'neighborhood_universityhills', 'neighborhood_universitypark', 'neighborhood_valverde',
                        'neighborhood_virginiavillage', 'neighborhood_washingtonpark', 'neighborhood_washington
                        'neighborhood_washingtonvirginiavale', 'neighborhood_wellshire', 'neighborhood_westcolfa
                        'neighborhood_westwood', 'neighborhood_whittier', 'neighborhood_windsor']


review_columns = ['review_scores_rating', 'review_scores_accuracy', 'review_scores_cleanliness', 'review_scores_
                  'review_scores_communication', 'review_scores_location', 'review_scores_value']


# Top 5 neighborhoods based on listing counts
neighborhood_listing_counts = data[neighborhood_columns].sum()
top_10_neighborhoods_by_count = neighborhood_listing_counts.nlargest(10).index

# Collect review scores for the top 5 neighborhoods
neighborhood_scores_list = []

for neighborhood in top_10_neighborhoods_by_count:
    neighborhood_scores = data[data[neighborhood] == 1][review_columns].mean()
    neighborhood_scores['review_scores_rating'] = data[data[neighborhood] == 1]['review_scores_rating'].mean()
    neighborhood_scores_list.append(neighborhood_scores)

# Create a DataFrame for the top 10 neighborhoods review scores
top_10_neighborhoods_data = pd.DataFrame(neighborhood_scores_list, columns=review_columns)

# KMeans clustering
kmeans = KMeans(n_clusters=2, random_state=42)
top_10_neighborhoods_data_values = np.array(top_10_neighborhoods_data)

# Fit the KMeans model
kmeans.fit(top_10_neighborhoods_data_values)

# Assign clusters to the neighborhoods
top_10_neighborhoods_df = pd.DataFrame({
    'Neighborhood': top_10_neighborhoods_by_count,
    'Cluster': kmeans.labels_,
    'review_scores_rating': top_10_neighborhoods_data['review_scores_rating']
})

# Display results
print("\nK-means Clustering of the Top 10 Neighborhoods Based on Listing Counts and \n their average review sco
print(top_10_neighborhoods_df)



# Exhibit 8
from sklearn.linear_model import LinearRegression
X = data[['years_as_host']]  # Independent variable
y = data['review_scores_rating']  # Dependent variable

# Fit the model
model = LinearRegression()
model.fit(X, y)

print(f"Coefficient for years_as_host: {model.coef_}")
print(f"Intercept: {model.intercept_}")

# statsmodel for years as host and review scores rating
import statsmodels.api as sm
X = sm.add_constant(X)  # Adds intercept to the model
model_sm = sm.OLS(y, X).fit()
print(model_sm.summary())

"""There is a significant relationship between the number of years a
host has been active and their review scores rating."""
"""As hosts gain more experience (years as host), their review scores rating tends to improve."""


#jigna
# Step 1: Filter data into superhosts and non-superhosts
data = pd.read_csv(r"C:\Users\Pratik\Desktop\CU Denver Courses\Computing BANA 6620\Codes\Codes\Test\Clean_AirBN
superhosts = data[data['host_is_superhost_t'] == True]
non_superhosts = data[data['host_is_superhost_t'] == False]
```

```python
# Step 2: Calculate neighborhood distribution for both groups
superhost_distribution = superhosts['neighbourhood_cleansed'].value_counts(normalize=True)
non_superhost_distribution = non_superhosts['neighbourhood_cleansed'].value_counts(normalize=True)

# Step 3: Identify the top 10 neighborhoods based on total proportion
top_10_neighborhoods = (superhost_distribution + non_superhost_distribution).nlargest(10)

# Combine distributions for visualization
top_10_df = pd.DataFrame({
    "Superhosts": superhost_distribution[top_10_neighborhoods.index].fillna(0),
    "Non-Superhosts": non_superhost_distribution[top_10_neighborhoods.index].fillna(0)
}).fillna(0)

# Step 4: Plot the top 10 neighborhoods
ax = top_10_df.plot(kind='bar', figsize=(12, 6), width=0.8, color=["blue", "orange"])
plt.title("Top 10 Neighborhoods: Superhosts vs Non-Superhosts", fontsize=16)
plt.ylabel("Proportion of Listings", fontsize=12)
plt.xlabel("Neighborhoods", fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.legend(title="Host Type", loc="upper right")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Add annotations (percentage values above bars)
for p in ax.patches:
    ax.annotate(
        f"{p.get_height():.2%}",  # Format as percentage
        (p.get_x() + p.get_width() / 2, p.get_height()),
        ha='center',
        va='bottom',
        fontsize=9,
    )

plt.tight_layout()
plt.show()

# Step 1: Filter data into superhosts and non-superhosts
superhosts = data[data['host_is_superhost_t'] == True]
non_superhosts = data[data['host_is_superhost_t'] == False]

# Step 2: Calculate averages for response and acceptance rates
metrics = {
    "Response Rate (%)": [
        superhosts['host_response_rate'].mean(),
        non_superhosts['host_response_rate'].mean(),
    ],
    "Acceptance Rate (%)": [
        superhosts['host_acceptance_rate'].mean(),
        non_superhosts['host_acceptance_rate'].mean(),
    ],
}

# Convert to a DataFrame for plotting
comparison_df = pd.DataFrame(metrics, index=["Superhosts", "Non-Superhosts"])

# Step 3: Create bar plots for visual comparison
ax = comparison_df.plot(kind="bar", figsize=(10, 6), width=0.8, color=['blue', 'orange'])

# Step 4: Add percentage titles above the bars
for p in ax.patches:
    ax.annotate(
        f"{p.get_height():.1f}%",  # Percentage title
        (p.get_x() + p.get_width() / 2, p.get_height() + 1),
        ha='center',
        fontsize=10,
    )

# Step 5: Enhance the chart with labels, title, and legend
plt.title("Comparison of Metrics: Superhosts vs Non-Superhosts", fontsize=16)
plt.ylabel("Percentage (%)", fontsize=12)
plt.xlabel("Host Type", fontsize=12)
plt.xticks(rotation=0, fontsize=10)
plt.legend(title="Metrics", loc="upper left")
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Step 6: Show the plot
plt.tight_layout()
plt.show()

#pratik
# Basic Information
print("Shape of the dataset:", data.shape)
print("\nColumns in the dataset:\n", data.columns)
```

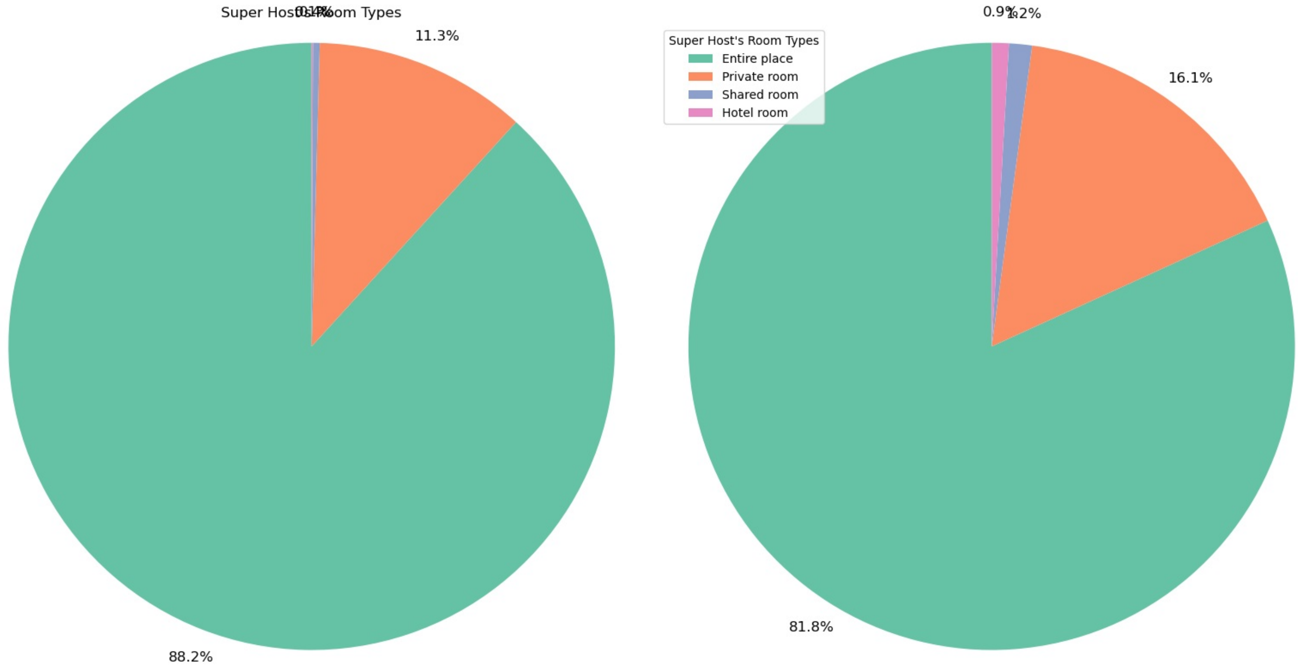
```python
# Display first few rows
print("\nFirst few rows of the dataset:\n", data.head())

# Check for missing data
missing_data = data.isnull().sum().sort_values(ascending=False)
print("\nMissing Data Summary:\n", missing_data[missing_data > 0])

# missing_data = data.isnull().sum()
# missing_data

# Impute missing values
data['description'].fillna("No Description", inplace=True)
data['neighborhood_overview'].fillna("No Overview", inplace=True)

columns_to_drop = ['host_about']
data= data.drop(columns=columns_to_drop, axis=1)

data = data.drop(['license'],axis=1)

#drop redundant avaialibility-related columns
columns_to_drop = ['availability_30', 'availability_60', 'availability_90']
data = data.drop(columns=columns_to_drop, axis=1)
print("Remaining Columns:\n", data.columns)

# Drop redundant review-related columns
columns_to_drop = ['number_of_reviews', 'number_of_reviews_l30d']
data = data.drop(columns=columns_to_drop, axis=1)

#Drop redundant review-score rating  and just keeping in gernal review_score_rating alone

data['average_review_score'] = data[
    ['review_scores_accuracy', 'review_scores_cleanliness', 'review_scores_checkin',
     'review_scores_communication', 'review_scores_location', 'review_scores_value']
].mean(axis=1)

columns_to_drop = ['review_scores_accuracy','review_scores_cleanliness',
                   'review_scores_checkin','review_scores_communication',
                    'review_scores_location','review_scores_value' ]

data = data.drop(columns=columns_to_drop, axis=1)

correlation = data[['review_scores_rating', 'average_review_score']].corr()
print("Correlation between review_scores_rating and average_review_score:\n", correlation)

columns_to_drop = ['average_review_score']
data = data.drop(columns=columns_to_drop, axis=1)
print("Remaining Columns:\n", data.columns)
'''''
Strengths of the Dataset

Target Variable Present:
The column host_is_superhost_t is clean and ready for use as the target variable.
No missing values here, which is crucial

Well-Chosen Features:
Includes a mix of numerical (price, reviews_per_month, availability_365) and
categorical features (room_type_Private room, license_bool).

Relevant columns like license_bool and host_identity_verified_dum_t
help capture trust and professionalism.

Consolidated review and availability features (e.g., review_scores_rating, availability_365).

Boolean Indicators:

Columns like description_bool, license_bool,
and neighborhood_overview_bool capture the presence/absence
of key attributes without redundancy.

Some features might still have high correlations (e.g., host_response_rate
 and host_acceptance_rate).
lets check this first before moving ahead.
'''

import seaborn as sns
import matplotlib.pyplot as plt

# Select numerical features
numerical_cols = [
    'host_response_rate', 'host_acceptance_rate', 'price',
    'availability_365', 'reviews_per_month', 'review_scores_rating'
]
```

```python
# Compute the correlation matrix
correlation_matrix = data[numerical_cols].corr()
correlation_matrix

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()

'''
#Key Observations
host_response_rate and host_acceptance_rate:
Correlation: 0.356 → Weak to moderate positive correlation.
Conclusion: These features are not strongly correlated,
so we should keep both as they likely provide complementary information.

price:
Low correlation with all other features (≤0.03).
Conclusion: price appears independent and should be retained as a critical predictor.

availability_365:
Slight negative correlations with host_response_rate (-0.062)
and reviews_per_month (-0.090).
Conclusion: availability_365 provides unique information about
yearly availability and should be retained.

reviews_per_month and review_scores_rating:
Weak positive correlation (0.102).
Conclusion: These features offer distinct insights and should both be retained.
'''

# Check if necessary columns exist for response time visualization
if 'response_time_in_hours' in data.columns and 'host_is_superhost_t' in data.columns:

    # Calculate mean response time for superhost and non-superhost
    mean_response_time = data.groupby('host_is_superhost_t')['response_time_in_hours'].mean()

    # Plot the line chart
    plt.figure(figsize=(10, 6))
    mean_response_time.plot(kind='line', marker='o', color='g')
    plt.title("Mean Response Time for Superhosts vs Non-Superhosts")
    plt.xlabel("Superhost Status (False = Non-Superhost, True = Superhost)")
    plt.ylabel("Mean Response Time (Hours)")
    plt.xticks([0, 1], labels=["Non-Superhost", "Superhost"])
    plt.grid()
    plt.show()
else:
    print("Required columns ('response_time_in_hours', 'host_is_superhost_t') are missing from the dataset.")

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score, roc_curve, accuracy_score,confusion_matrix,Co
import matplotlib.pyplot as plt

# Define features and target
features = ['host_response_rate', 'host_acceptance_rate', 'price',
            'availability_365', 'reviews_per_month', 'review_scores_rating']
target = 'host_is_superhost_t'

X = data[features]
y = data[target]

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train logistic regression model
model = LogisticRegression(random_state=42)
model.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = model.predict(X_test_scaled)
y_prob = model.predict_proba(X_test_scaled)[:, 1]

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

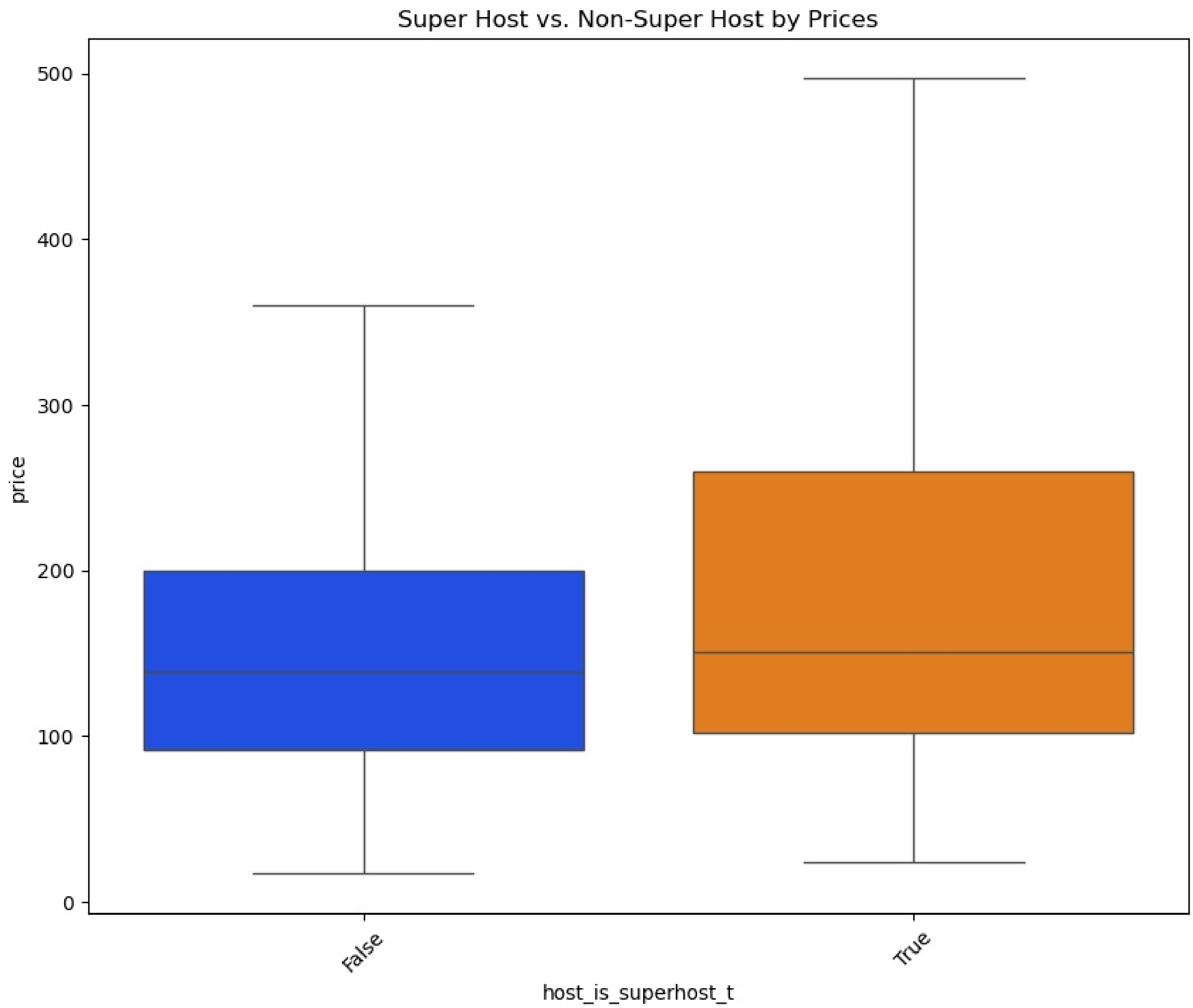
```
# Visualize confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Superhost', 'Superhost'], yticklal
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# ROC-AUC Score
roc_auc = roc_auc_score(y_test, y_prob)
print(f"ROC-AUC Score: {roc_auc:.2f}")

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
'''
1. Key Metrics

Precision:
For False (Non-Superhosts): 69% -Out of all instances predicted as non-superhosts,69% was correct
For True (Superhosts): 69% - Out of all instances predicted as superhosts, 69% were correct.

Recall:

For False (Non-Superhosts):48% - The model correctly identified 48% of actual non-superhosts.
For True (Superhosts): 85% - The model correctly identified 85% of actual superhosts.

F1-Score:
For False (Non-Superhosts): 0.57 - Indicates moderate balance of precision and recall for non -superhost.
For True (Superhosts): 0.76 - Good Balance of precision and recall for superhost

Accuracy:
Overall, the model is 69% accurate in its predictions.

Class Imbalance:Higher recall for superhosts (85%) indicates the model performs
better at identifying superhosts compared to non-superhosts (48%).


The performance metrics indicate a moderate class imbalance, particularly in the recall for non-superhosts (Fal

Understanding the Imbalance - Class Distribution
'''

# Check class distribution
class_counts = data['host_is_superhost_t'].value_counts()
print("Class Distribution:\n", class_counts)

## since there is no significance difference in the count, may be we can try fixing
#using class weights
# Train logistic regression with class weights
model_weighted = LogisticRegression(class_weight="balanced", random_state=42)
model_weighted.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_weighted = model_weighted.predict(X_test_scaled)
print("Classification Report (Weighted Logistic Regression):")
print(classification_report(y_test, y_pred_weighted))

#it slightly improved but further can be improved lets adjust threshold
#threshold Adjustment
# Predict probabilities
y_prob = model_weighted.predict_proba(X_test_scaled)[:, 1]

# Adjust the threshold
threshold = 0.4
y_pred_adjusted = (y_prob >= threshold).astype(int)

# Evaluate with adjusted threshold
print("Classification Report with Adjusted Threshold:")
print(classification_report(y_test, y_pred_adjusted))

from sklearn.ensemble import RandomForestClassifier
```

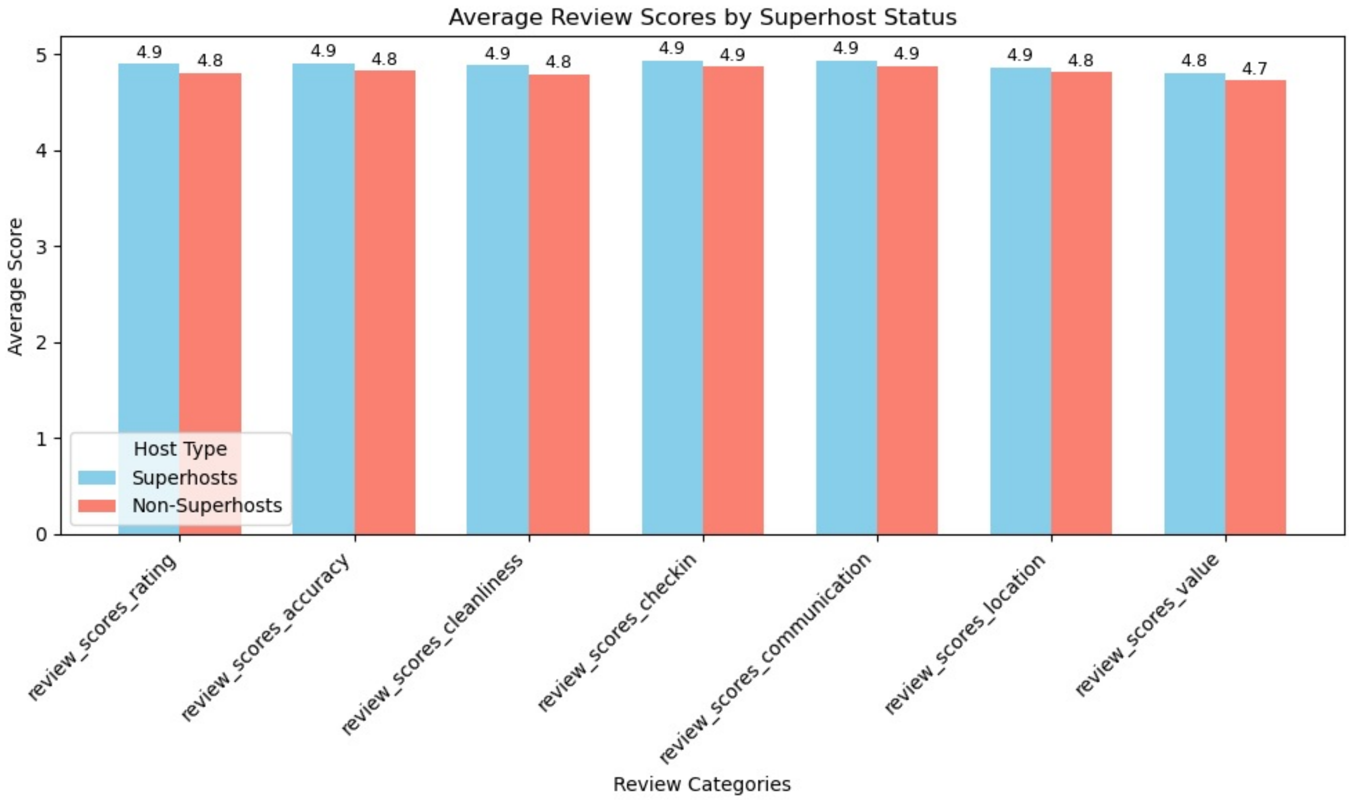
```python
from sklearn.metrics import classification_report

#The adjusted threshold has improved recall for the majority class (True: Superhost)
#but at the expense of recall for the minority class (False: Non-Superhost).
#Here:Given that threshold adjustment only moderately balances recall
#and precision, moving to a more sophisticated model is the logical next step.
```

```
Summary Statistics:
Number of rows with all three false: 2481
Number of rows with all three false: 1803
```

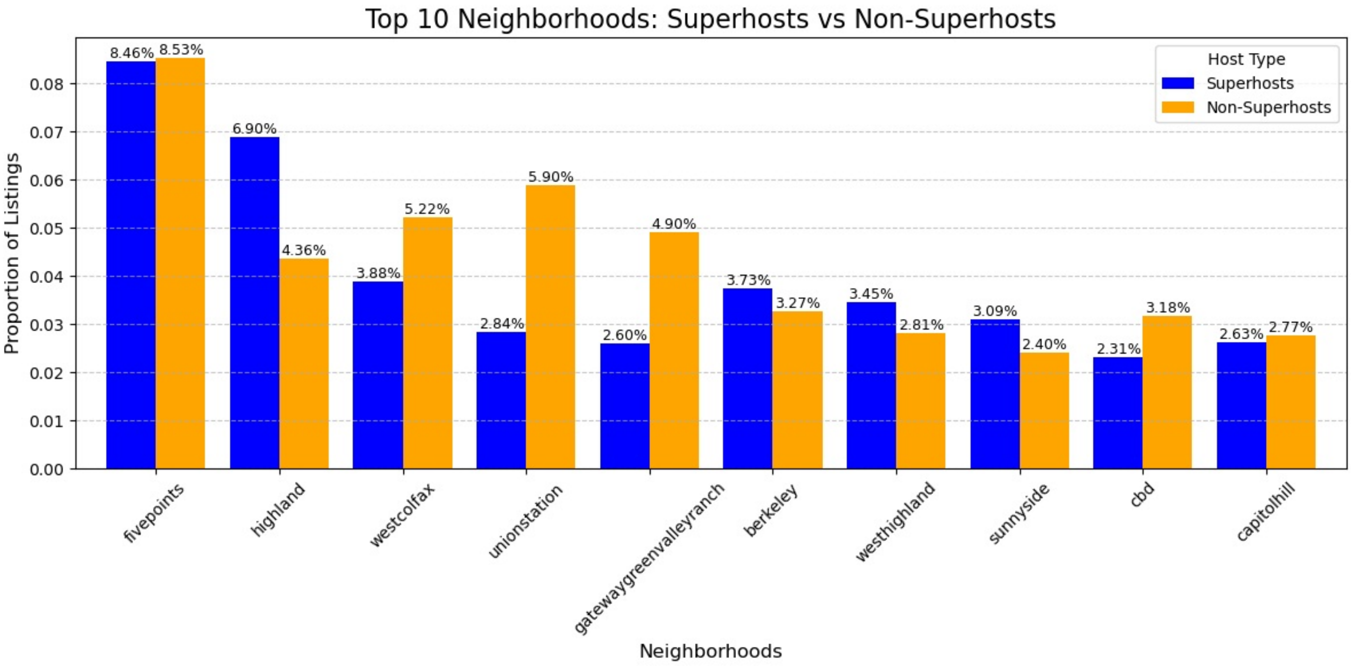


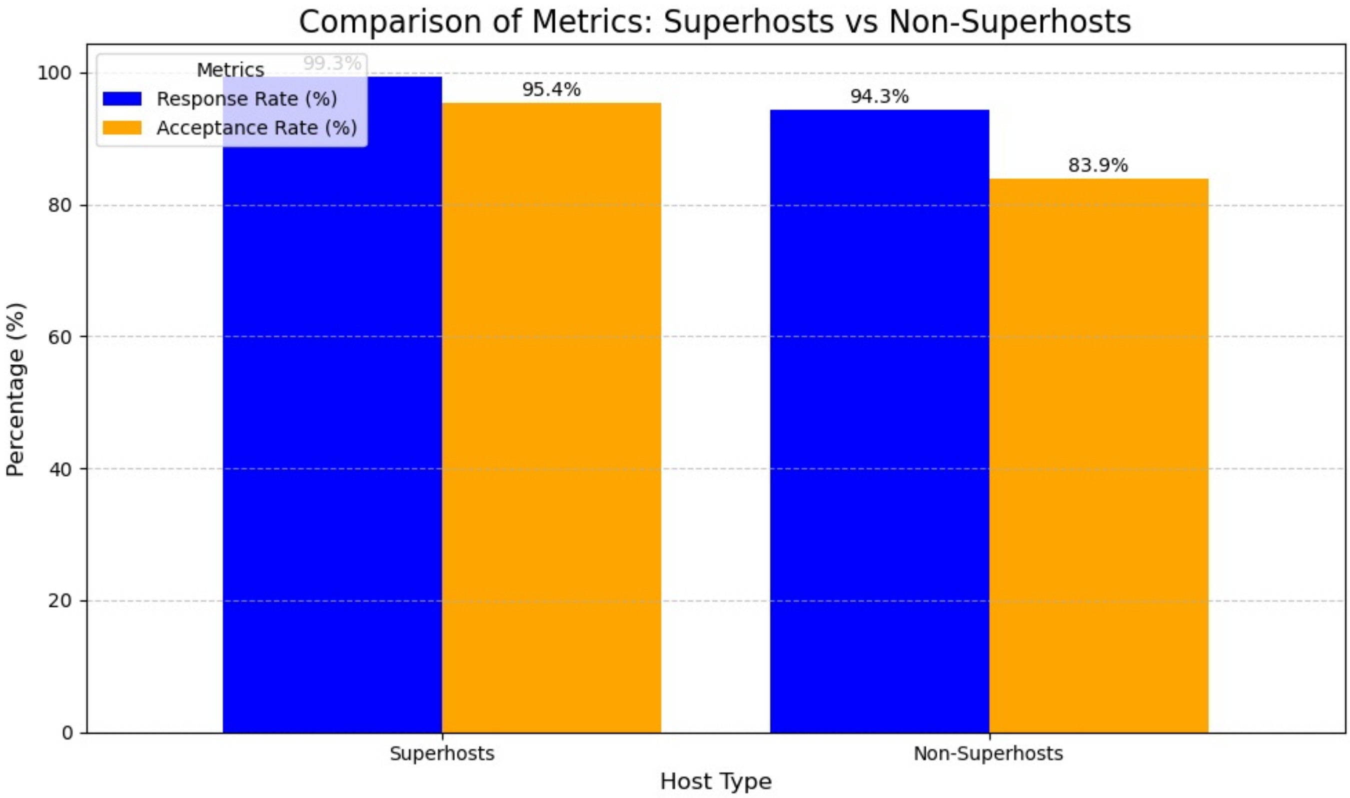

```
<ipython-input-13-cc48914e6dd6>:111: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable
to `hue` and set `legend=False` for the same effect.

  sns.boxplot(x='host_is_superhost_t', y='price', data=data,
<ipython-input-13-cc48914e6dd6>:111: UserWarning: The palette list has more values (10) than needed (2), which m
ay not be intended.
  sns.boxplot(x='host_is_superhost_t', y='price', data=data,
```


Super Host vs. Non-Super Host by Prices
price
0
100
200
300
400
500
False
True
host_is_superhost_t




```
c:\Users\Pratik\anaconda3\envs\bana6620\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The de
fault value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress t
he warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\Pratik\anaconda3\envs\bana6620\lib\site-packages\sklearn\cluster\_kmeans.py:1436: UserWarning: KMeans i
s known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can av
oid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
K-means Clustering of the Top 10 Neighborhoods Based on Listing Counts and
 their average review score rating:
                          Neighborhood  Cluster  review_scores_rating
0              neighborhood_fivepoints        1              4.874956
1                neighborhood_highland        1              4.891810
2              neighborhood_westcolfax        1              4.869168
3            neighborhood_unionstation        0              4.801168
4   neighborhood_gatewaygreenvalleyranch        0              4.781186
5                neighborhood_berkeley        1              4.893810
6            neighborhood_westhighland        1              4.886630
7               neighborhood_sunnyside        1              4.911670
8                   neighborhood_baker        1              4.874535
9             neighborhood_capitolhill        1              4.862678
Coefficient for years_as_host: [0.00926964]
Intercept: 4.789292004632663
                             OLS Regression Results
================================================================================
Dep. Variable:     review_scores_rating   R-squared:                       0.015
Model:                              OLS   Adj. R-squared:                  0.015
Method:                   Least Squares   F-statistic:                     77.54
Date:                  Wed, 11 Dec 2024   Prob (F-statistic):           1.76e-18
Time:                          15:20:18   Log-Likelihood:                 25.827
No. Observations:                  5016   AIC:                            -47.65
Df Residuals:                      5014   BIC:                            -34.61
Df Model:                             1
Covariance Type:              nonrobust
=================================================================================
                    coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
const             4.7893      0.009    543.438      0.000       4.772       4.807
years_as_host     0.0093      0.001      8.806      0.000       0.007       0.011
==============================================================================
Omnibus:                     6279.333   Durbin-Watson:                   1.873
Prob(Omnibus):                  0.000   Jarque-Bera (JB):          1198888.775
Skew:                          -6.742   Prob(JB):                         0.00
Kurtosis:                      77.529   Cond. No.                         22.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```



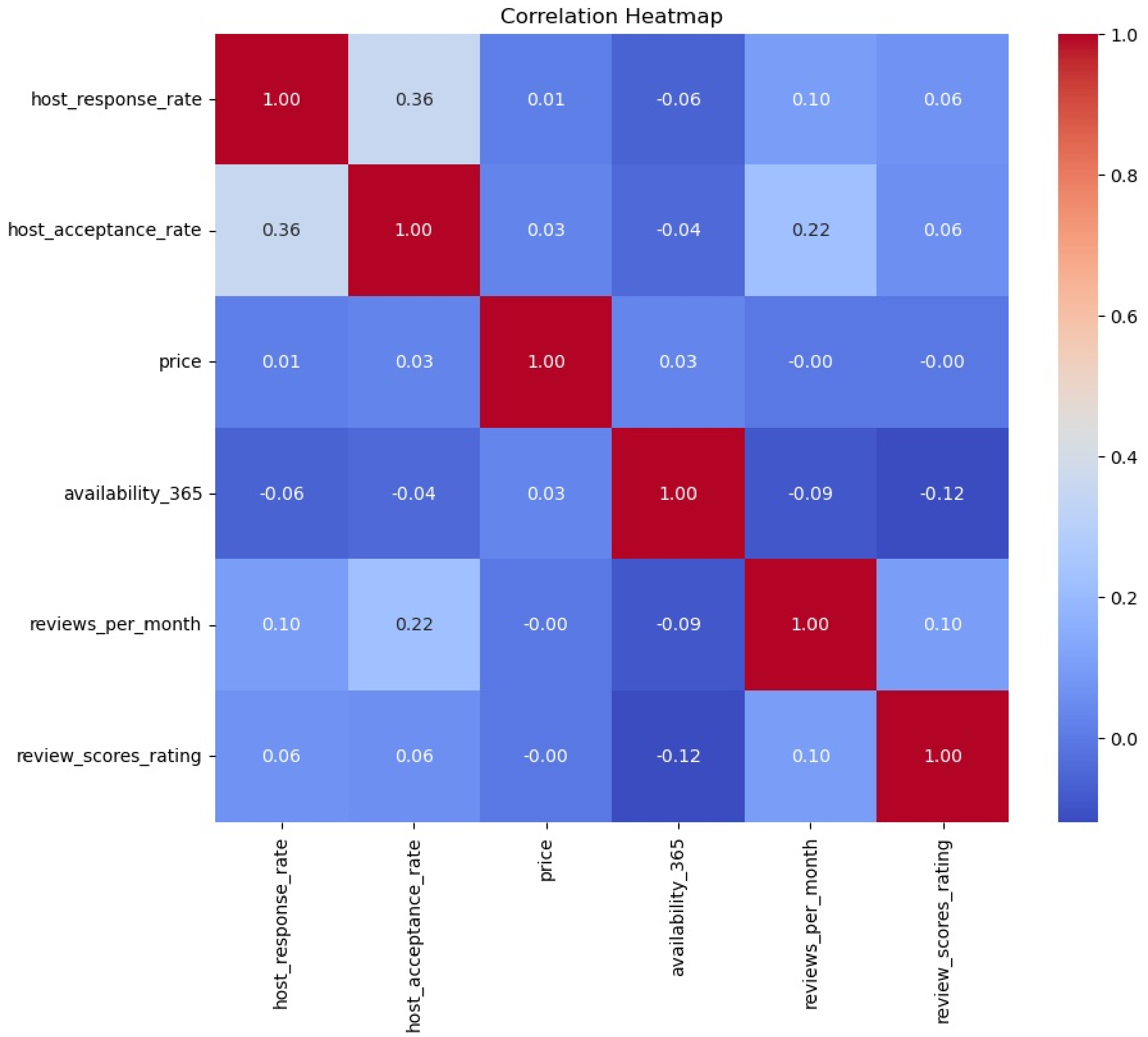


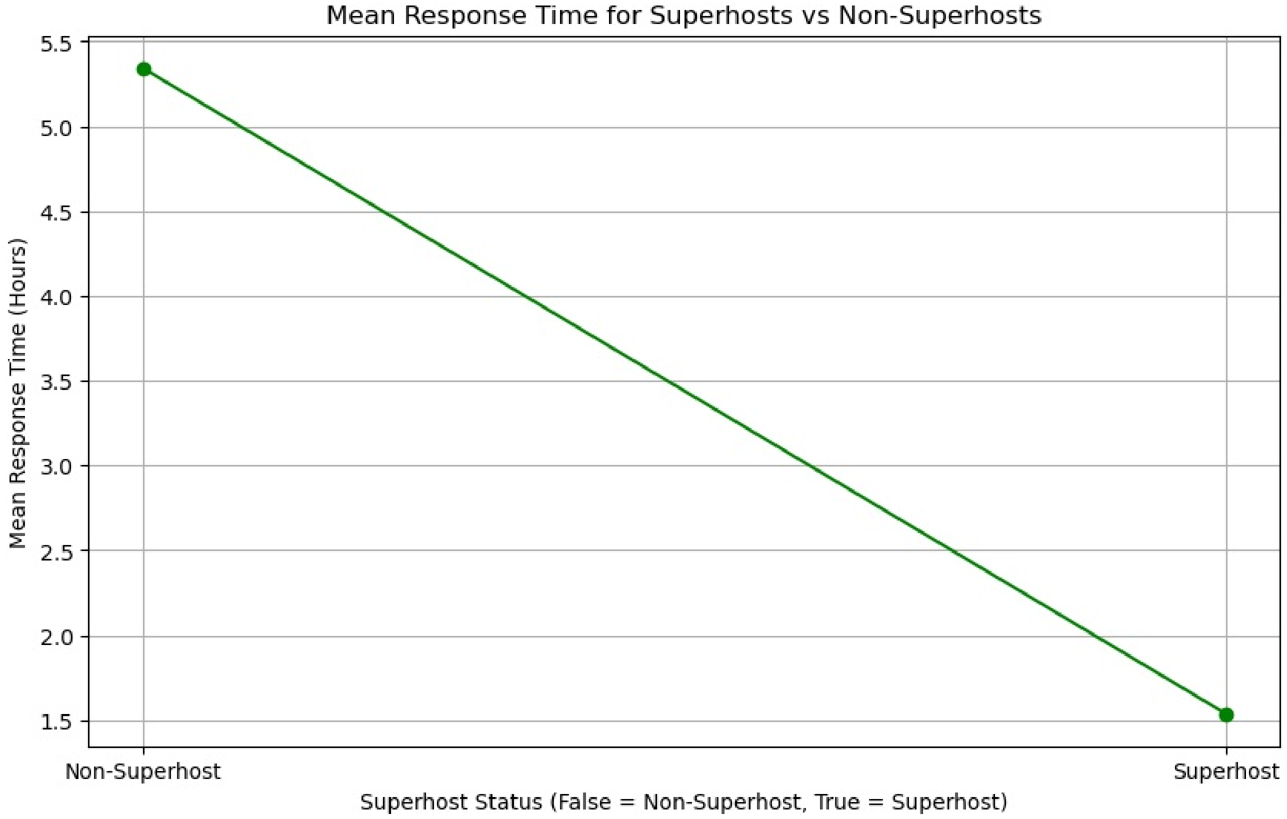

```
Shape of the dataset: (5016, 47)

Columns in the dataset:
 Index(['id', 'description', 'neighborhood_overview', 'host_name', 'host_about',
       'host_response_rate', 'host_acceptance_rate',
       'host_total_listings_count', 'neighbourhood_cleansed', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'availability_30',
       'availability_60', 'availability_90', 'availability_365',
       'number_of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d',
       'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'license', 'calculated_host_listings_count',
       'reviews_per_month', 'years_as_host', 'days_since_last_review',
       'description_bool', 'neighborhood_overview_bool', 'license_bool',
       'host_about_bool', 'response_time_in_hours',
       'host_identity_verified_dum_t', 'room_type_Hotel room',
       'room_type_Private room', 'room_type_Shared room', 'instant_bookable_t',
       'host_is_superhost_t'],
      dtype='object')

First few rows of the dataset:
         id                                        description  \
```

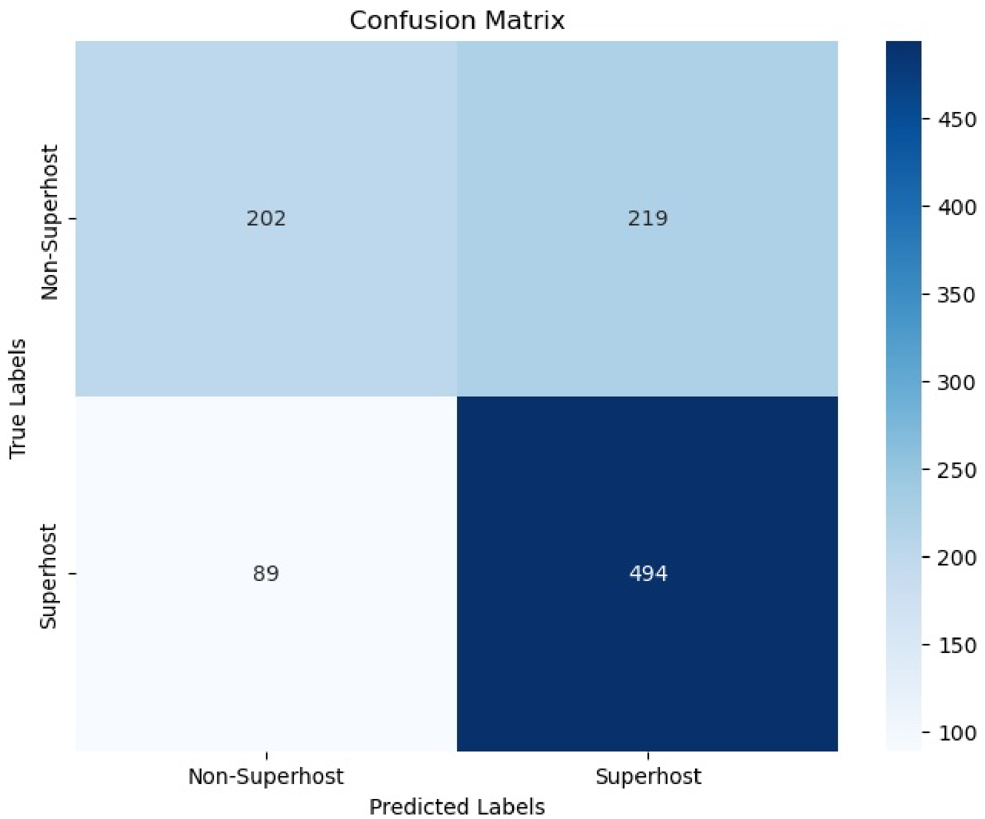
```
0    360.0  Enjoy the famous Colorado weather and unplug i...
1    590.0  Large guest room in my home, where I also live...
2    592.0  This room is in the basement.  It does not hav...
3   1940.0  Private studio with separate entrance in histo...
4  21745.0  Thank you for visiting my King Bed Room site! ...

                              neighborhood_overview            host_name  \
0  The cottage is located in the center of Lower ...  Jennifer & Giovanni
1  I love the diversity of my neighborhood and it...                 Jill
2                                                NaN                 Jill
3  Walking through the Baker historical neighborh...               Joanne
4  I love my Uptown neighborhood, which is within...            Alexandra

                                          host_about  host_response_rate  \
0  We are artists and tinkerers.\r\n \r\nWe enjoy...               100.0
1  I am friendly and I love meeting people from a...               100.0
2  I am friendly and I love meeting people from a...               100.0
3  I've had the good fortune to travel to many co...               100.0
4  Denver native, former teacher, musician, chapl...               100.0

   host_acceptance_rate  host_total_listings_count neighbourhood_cleansed  \
0                  98.0                          4               highland
1                  93.0                          2          northparkhill
2                  93.0                          2          northparkhill
3                  99.0                         13                  baker
4                 100.0                          4       northcapitolhill

   accommodates  ...  neighborhood_overview_bool  license_bool  \
0             2  ...                           0             0
1             3  ...                           0             0
2             2  ...                           1             0
3             2  ...                           0             0
4             2  ...                           0             1

   host_about_bool response_time_in_hours  host_identity_verified_dum_t  \
0                0                    1.0                          True
1                0                    1.0                          True
2                0                    1.0                          True
3                0                    1.0                          True
4                0                    2.0                          True

   room_type_Hotel room  room_type_Private room  room_type_Shared room  \
0                 False                   False                  False
1                 False                    True                  False
2                 False                    True                  False
3                 False                   False                  False
4                 False                    True                  False

   instant_bookable_t  host_is_superhost_t
0               False                 True
1               False                 True
2               False                 True
3               False                 True
4               False                 True

[5 rows x 47 columns]

Missing Data Summary:
 host_about               1999
license                  1752
neighborhood_overview    1670
description                71
dtype: int64
Remaining Columns:
 Index(['id', 'description', 'neighborhood_overview', 'host_name',
       'host_response_rate', 'host_acceptance_rate',
       'host_total_listings_count', 'neighbourhood_cleansed', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'availability_365',
       'number_of_reviews', 'number_of_reviews_ltm', 'number_of_reviews_l30d',
       'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'calculated_host_listings_count',
       'reviews_per_month', 'years_as_host', 'days_since_last_review',
       'description_bool', 'neighborhood_overview_bool', 'license_bool',
       'host_about_bool', 'response_time_in_hours',
       'host_identity_verified_dum_t', 'room_type_Hotel room',
       'room_type_Private room', 'room_type_Shared room', 'instant_bookable_t',
       'host_is_superhost_t'],
      dtype='object')
Correlation between review_scores_rating and average_review_score:
                       review_scores_rating  average_review_score
```

```
review_scores_rating              1.000000            0.900738
average_review_score              0.900738            1.000000
Remaining Columns:
 Index(['id', 'description', 'neighborhood_overview', 'host_name',
       'host_response_rate', 'host_acceptance_rate',
       'host_total_listings_count', 'neighbourhood_cleansed', 'accommodates',
       'bathrooms', 'bedrooms', 'beds', 'amenities', 'price',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'availability_365',
       'number_of_reviews_ltm', 'review_scores_rating',
       'calculated_host_listings_count', 'reviews_per_month', 'years_as_host',
       'days_since_last_review', 'description_bool',
       'neighborhood_overview_bool', 'license_bool', 'host_about_bool',
       'response_time_in_hours', 'host_identity_verified_dum_t',
       'room_type_Hotel room', 'room_type_Private room',
       'room_type_Shared room', 'instant_bookable_t', 'host_is_superhost_t'],
      dtype='object')
```

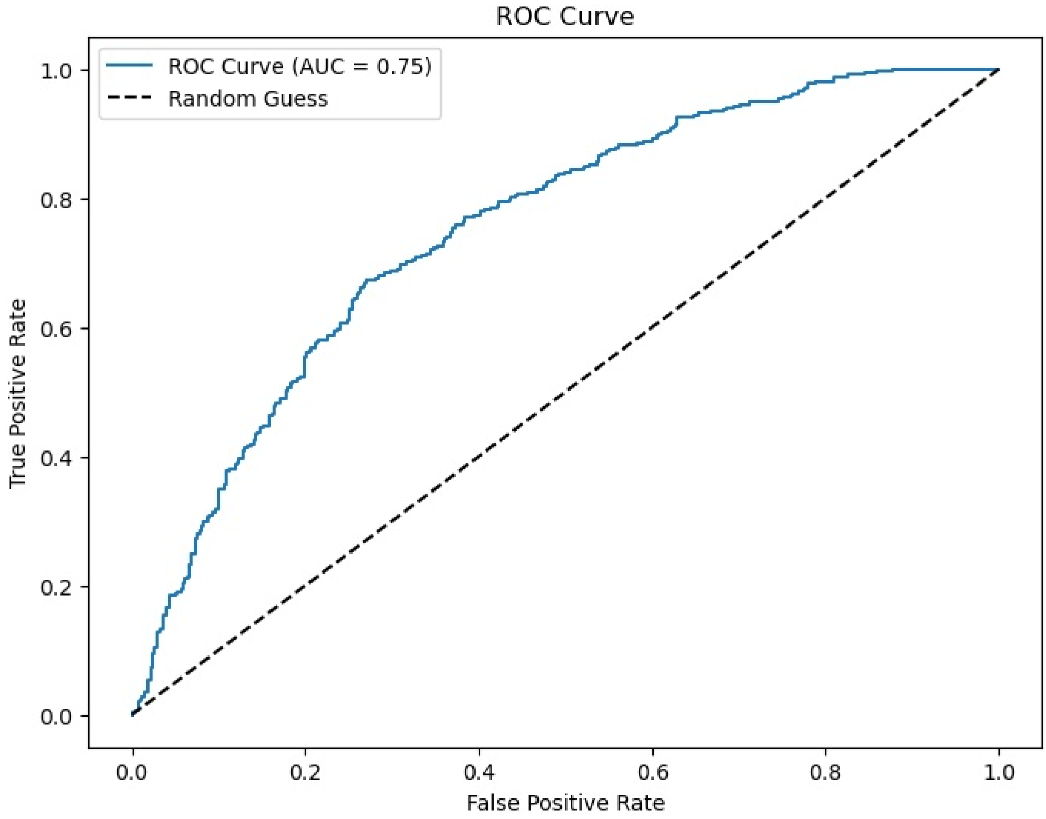
```
<ipython-input-13-cc48914e6dd6>:426: FutureWarning: A value is trying to be set on a copy of a DataFrame or Seri
es through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on w
hich we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)'
or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['description'].fillna("No Description", inplace=True)
<ipython-input-13-cc48914e6dd6>:427: FutureWarning: A value is trying to be set on a copy of a DataFrame or Seri
es through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on w
hich we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)'
or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['neighborhood_overview'].fillna("No Overview", inplace=True)
```








```
Classification Report:
              precision    recall  f1-score   support

       False       0.69      0.48      0.57       421
        True       0.69      0.85      0.76       583

    accuracy                           0.69      1004
   macro avg       0.69      0.66      0.66      1004
weighted avg       0.69      0.69      0.68      1004

ROC-AUC Score: 0.75
```




```
Class Distribution:
 host_is_superhost_t
True     2812
False    2204
Name: count, dtype: int64
Classification Report (Weighted Logistic Regression):
              precision    recall  f1-score   support

       False       0.67      0.57      0.61       421
        True       0.72      0.80      0.76       583

    accuracy                           0.70      1004
   macro avg       0.70      0.68      0.69      1004
weighted avg       0.70      0.70      0.70      1004

Classification Report with Adjusted Threshold:
              precision    recall  f1-score   support

       False       0.72      0.42      0.53       421
        True       0.68      0.88      0.77       583

    accuracy                           0.69      1004
   macro avg       0.70      0.65      0.65      1004
weighted avg       0.70      0.69      0.67      1004
```

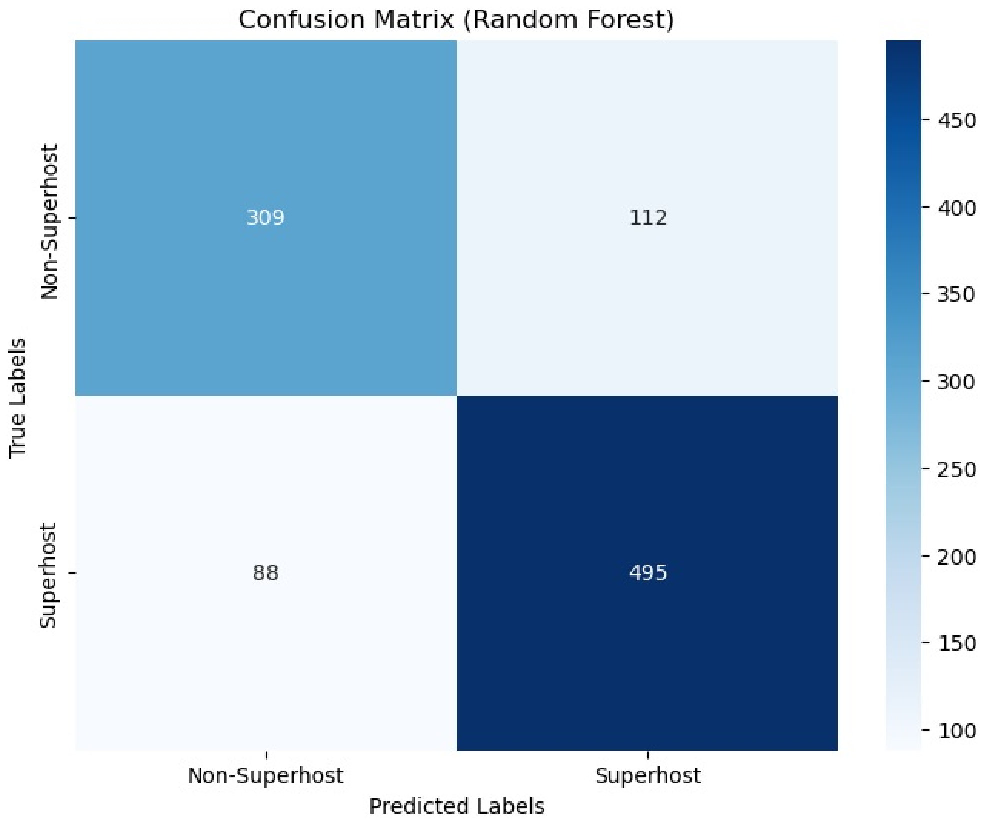
In [ ]:

```
# Train Random Forest with class weights
rf_model = RandomForestClassifier(class_weight="balanced", random_state=42)
rf_model.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred_rf = rf_model.predict(X_test_scaled)
y_prob_rf = rf_model.predict_proba(X_test_scaled)[:, 1]

print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

from sklearn.metrics import confusion_matrix
import seaborn as sns

# Confusion Matrix for Random Forest
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
```

```python
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Superhost', 'Superhost'], yticl
plt.title("Confusion Matrix (Random Forest)")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

'''
The Random Forest model shows a significant improvement
in overall performance compared to the adjusted threshold logistic regression.
'''

import pickle

# Save the trained Random Forest model
with open("random_forest_model.pkl", "wb") as model_file:
    pickle.dump(rf_model, model_file)

# Save the scaler
with open("scaler.pkl", "wb") as scaler_file:
    pickle.dump(scaler, scaler_file)

from langchain.chat_models import ChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
import yaml
import os
import sys  # Added for proper exit handling

# Load OpenAI API Key
OPENAI_API_KEY = yaml.safe_load(open("credentials.yml"))["openai"]
os.environ["OPENAI_API_KEY"] = OPENAI_API_KEY

# Initialize LangChain Chat LLM with the correct model
llm = ChatOpenAI(model="gpt-3.5-turbo", temperature=0.7)

# Function to validate user input
def validate_input(prompt, min_val, max_val, dtype):
    """
    Validates user input to ensure it falls within the specified range or allows exiting.
    """
    while True:
        try:
            value = input(f"{prompt} ({min_val} to {max_val}, or type 'exit' to quit): ").strip()
            if value.lower() == "exit":
                print("Bye! Thank you for using the Airbnb Superhost Chatbot!")
                sys.exit(0)  # Cleanly terminate the program
            value = dtype(value)
            if min_val <= value <= max_val:
                return value
            else:
                print(f"Value must be between {min_val} and {max_val}. Please try again.")
        except ValueError:
            print(f"Invalid input. Please enter a valid {dtype.__name__} value.")

# LangChain Prompt for Predictions
prompt = PromptTemplate(
    input_variables=["response_rate", "acceptance_rate", "price", "availability", "reviews", "rating"],
    template=(
        "Given the following metrics:\n"
        "- Response Rate: {response_rate}%\n"
        "- Acceptance Rate: {acceptance_rate}%\n"
        "- Price per Night: ${price}\n"
        "- Availability (days/year): {availability}\n"
        "- Reviews per Month: {reviews}\n"
        "- Rating: {rating}/5\n\n"
        "Classify if this host qualifies as a Superhost and provide actionable recommendations "
        "if they do not qualify. Ensure your explanation references the metrics provided."
    )
)

# Function to interact with LangChain for prediction and recommendations
def get_prediction_and_recommendations(inputs):
    """
    Uses LangChain to classify Superhost status and generate recommendations.
    """
    chain = LLMChain(llm=llm, prompt=prompt)
    result = chain.run(inputs)
    return result

# Chatbot function
def chatbot_with_langchain():
    """
```

```python
    Airbnb Superhost Chatbot using LangChain for predictions and recommendations.
    """
    print("Welcome to the Airbnb Superhost Chatbot!")
    print("Provide the following inputs about your hosting profile (type 'exit' to quit):")

    # Collect user inputs
    inputs = {
        "response_rate": validate_input("Host Response Rate", 0, 100, float),
        "acceptance_rate": validate_input("Host Acceptance Rate", 0, 100, float),
        "price": validate_input("Average Price per Night", 20, 1000, float),
        "availability": validate_input("Availability in the Next 365 Days", 0, 365, int),
        "reviews": validate_input("Reviews Per Month", 0.01, 26, float),
        "rating": validate_input("Review Scores Rating", 1, 5, float),
    }

    # Get predictions and recommendations
    print("\n--- Prediction and Recommendations ---")
    response = get_prediction_and_recommendations(inputs)
    #print("\n".join(response.split(". ")))
    print(response)



# Run the chatbot
if __name__ == "__main__":
    try:
        chatbot_with_langchain()
    except KeyboardInterrupt:
        print("\nBye! Thank you for using the Airbnb Superhost Chatbot!")
    except SystemExit:
        # Suppress the SystemExit traceback completely
        pass
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

       False       0.78      0.73      0.76       421
        True       0.82      0.85      0.83       583

    accuracy                           0.80      1004
   macro avg       0.80      0.79      0.79      1004
weighted avg       0.80      0.80      0.80      1004
```



```
Welcome to the Airbnb Superhost Chatbot!
Provide the following inputs about your hosting profile (type 'exit' to quit):

--- Prediction and Recommendations ---
Based on the metrics provided, the host does not qualify as a Superhost. To qualify as a Superhost on most platf
orms, hosts typically need to maintain a high level of performance across various metrics.

Here's why this host does not qualify:


1. Response Rate: The response rate of 89.0% is quite good, but to qualify as a Superhost, platforms usually req
uire a response rate of 90% or higher. The host should aim to respond promptly to all inquiries to improve this
metric.

2. Acceptance Rate: The acceptance rate of 85.0% is slightly below the typical threshold for Superhost status, w
hich is usually around 88% or higher. Hosts should consider accepting more booking requests to improve this metr
ic.

3. Reviews per Month: The host receives 10.0 reviews per month, which is a good indicator of guest satisfaction.
However, to qualify as a Superhost, hosts usually need to maintain a consistently high number of positive review
s.

4. Rating: The host's rating of 4.0/5 is decent, but Superhosts typically have ratings of 4.8 or higher. Hosts s
hould aim to provide exceptional service to increase their overall rating.

Actionable recommendations for the host to improve their chances of qualifying as a Superhost include:
- Increase response rate to at least 90% by promptly responding to all inquiries.
- Improve acceptance rate by accepting more booking requests.
- Encourage guests to leave positive reviews to increase the number of reviews per month.
- Strive to provide exceptional service to increase overall rating to 4.8 or higher.

By focusing on these areas of improvement, the host can work towards qualifying as a Superhost and enhancing the
ir reputation on the platform.
```

In [ ]:

```python
if __name__ == "__main__":
    try:
        chatbot_with_langchain()
    except KeyboardInterrupt:
        print("\nBye! Thank you for using the Airbnb Superhost Chatbot!")
    except SystemExit:
        # Suppress the SystemExit traceback completely
        pass
```

```
Welcome to the Airbnb Superhost Chatbot!
Provide the following inputs about your hosting profile (type 'exit' to quit):
Bye! Thank you for using the Airbnb Superhost Chatbot!
```