PROJECT

HOME STAYS

About Project

Goal

The project aims to explore the data through `comprehensive exploratory data analysis (EDA)` and construct a `robust machine learning model` for the task, followed by detailed extraction of the model's insights.

Building a robust predictive model to estimate the `log_price` of homestay listings based on comprehensive analysis of their characteristics, amenities, and host information.

About Dataset

The dataset `"Homestays_Data.csv"` is provided for a project aimed at building a predictive model to estimate homestay prices based on various characteristics. With over 70,000 rows, it contains columns like 'log_price', 'host_since', 'amenities', 'last_review', 'room_type', 'property_type', and geographic coordinates and others.

DataLink : Home stays Data

Contents:

Some Visualization are missing you can see them

Technologies Used

1. pandas: Data manipulation and analysis in tabular format.

2. pickle: Serialization and deserialization of Python objects, often used for saving and loading machine learning models.

3. xgboost, lightgbm, CatBoostRegressor: Gradient boosting algorithms for regression tasks. These are powerful algorithms for predictive modeling.

4. sklearn.metrics.mean_squared_error, mean_absolute_error, r2_score: Evaluation metrics for regression models.

5. sklearn.model_selection.train_test_split: Splitting data into training and testing sets for model evaluation.

6. sklearn.linear_model.LinearRegression, Ridge, Lasso, ElasticNet: Linear regression and its regularized versions for regression tasks.

7. sklearn.tree.DecisionTreeRegressor: Decision tree-based regression algorithm.

8. sklearn.ensemble.RandomForestRegressor, GradientBoostingRegressor: Ensemble learning algorithms for regression tasks.

9. sklearn.feature_selection.RFE, SelectKBest, mutual_info_regression: Feature selection techniques for selecting the most important features for modeling.

10. sklearn.preprocessing.StandardScaler: Standardization of features by removing the mean and scaling to unit variance.

11. sklearn.decomposition.PCA: Dimensionality reduction using Principal Component Analysis.

12. numpy: Numerical computing, especially for arrays and matrices.

13. seaborn, matplotlib.pyplot, plotly.express, scipy.stats.skew: Data visualization libraries for generating insights and outcomes from data.

14. geopy.geocoders.Nominatim, geopy.distance.geodesic: Geocoding and distance calculation between geographic locations.

15. transformers.AutoModelForSequenceClassification, TFAutoModelForSequenceClassification, AutoTokenizer, AutoConfig, pipeline: Components from Hugging Face's Transformers library for natural language processing tasks like sequence classification.

16. sklearn.impute.KNNImputer: Imputation of missing values using k-nearest neighbors.

17. warnings: Python's built-in module for issuing warnings.

Problems & their solutions in Dataset

Missing Data & Imputation :

The missing values in the columns ['bathrooms', 'first_review', 'host_has_profile_pic', 'host_identity_verified', 'host_response_rate', 'host_since', 'last_review', 'neighbourhood', 'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds'] were handled using a combination of random imputation, mean/mode imputation, and KNN imputation.

Encoding Techniques:

I utilized two encoding techniques for model training: one-hot encoding and label encoding. Interestingly, the label encoding method yielded the best results, although I experimented with both encodings and attached the outcomes for your review.

Feature Selection:

1. Starting with filter methods: This method is chosen for its efficiency and ease of interpretation. It involves techniques like correlation analysis, information gain, and L1 regularization to identify potentially relevant features from the dataset.

2. Refining with wrapper or embedded methods: To further enhance feature selection, you've incorporated wrapper methods such as forward selection and recursive feature elimination (RFE), as well as embedded methods like LASSO regularization. These methods consider the interaction between features and can improve model performance, albeit at a higher computational cost. They're employed after filter methods have narrowed down the pool of candidate features.

3. Considering PCA: In cases of high-dimensional data, Principal Component Analysis (PCA) is being considered. PCA reduces the dimensionality of the dataset while retaining its most informative features. This step aims to streamline the dataset for more efficient modeling without sacrificing crucial information.

Ultimately, the project's focus on filter methods underscores a pragmatic approach, prioritizing simplicity and ease of implementation while ensuring efficient feature selection tailored to its objectives and constraints.

Model Selection:

For model selection, I experimented with a variety of algorithms, including simple baseline models such as Linear Regression, Decision Tree Regression, and Random Forest Regression, along with more advanced techniques like XGBoost, LightGBM, and CatBoost. After thorough evaluation, I ultimately selected XGBoost, LightGBM, and CatBoost as the final models for their superior performance and robustness in predicting the target variable.

Model Monitoring & Feature importance

For model monitoring, I relied on Evidently, a comprehensive toolkit for assessing model performance and monitoring model behavior over time. Additionally, for feature extraction insights, I employed SHAP (SHapley Additive exPlanations), a powerful technique for understanding the impact of features on model predictions and extracting meaningful insights from complex machine learning models.

Insights form the EDA

There are alot of findings i cant write it here you can refer to This Link

Model Insights

I H've evaluated three models: XGBoost, LightGBM, and CatBoost. Let's summarize their performance:

- **XGBoost:**
  - Parameters: Learning rate of 0.1, max depth of 3, and 100 estimators.
  - Evaluation Metrics: MSE of 0.172, MAE of 0.304, and R^2 of 0.664.

- **LightGBM:**
  - Parameters: Maximum depth of 3, subsample of 0.8, and colsample bytree of 0.8.
  - Evaluation Metrics: MSE of 0.173, MAE of 0.304, and R^2 of 0.664.

- **CatBoost:**
  - Parameters: Default parameters.
  - Evaluation Metrics: MSE of 0.171, MAE of 0.297, and R^2 of 0.667.

Among these models, CatBoost performed the best with the lowest MSE, MAE, and highest R^2 score, indicating superior accuracy in predicting the target variable.

Dataset Overview

In [54]:
```
# !pip install geopy

import pandas as pd  # for handling tabular data
import numpy as np  # for linear algebra

# for visualization
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from scipy.stats import skew

from geopy.geocoders import Nominatim
from geopy.distance import geodesic

# for machine learning
from scipy.special import softmax
from sklearn.impute import KNNImputer

import warnings

# Ignore all warnings
warnings.filterwarnings('ignore')

from wordcloud import WordCloud
from geopy.distance import great_circle
from sklearn.cluster import KMeans
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [55]:
```
df = pd.read_csv('Dataset\Homestays_Data(in).csv')
df.head()
```

Out[55]:

	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleaning_f
0	6901257	5.010635	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	3	1.0	Real Bed	strict	Tru
1	6304928	5.129899	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	7	1.0	Real Bed	strict	Tru
2	7919400	4.976734	Apartment	Entire home/apt	{TV,"Cable TV","Wireless Internet","Air condit...	5	1.0	Real Bed	moderate	Tru
3	13418779	6.620073	House	Entire home/apt	{TV,"Cable TV",Internet,"Wireless Internet",Ki...	4	1.0	Real Bed	flexible	Tru
4	3808709	4.744932	Apartment	Entire home/apt	{TV,Internet,"Wireless Internet","Air conditio...	2	1.0	Real Bed	moderate	Tru

5 rows × 29 columns

In [56]:
```
# Select 5 random samples from the DataFrame
df_sample = df.sample(n=5, random_state=42)

# Print the sampled DataFrame
df_sample

# ---------------------------------------
# Select 5 random samples from the DataFrame using the sample() method
# Set the random_state to 42 for reproducibility
# ---------------------------------------
```

Out[56]:

	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleani
4079	13662370	3.806662	House	Private room	{TV,Internet,"Wireless Internet","Air conditio...	2	1.5	Real Bed	strict	
33735	4765892	4.941642	Apartment	Entire home/apt	{TV,Internet,"Wireless Internet","Air conditio...	2	2.0	Real Bed	strict	
69475	21169968	4.941642	Apartment	Entire home/apt	{TV,"Cable TV","Wireless Internet","Air condit...	5	1.0	Real Bed	moderate	
454	7939196	4.867534	Apartment	Entire home/apt	{"Cable TV",Internet,"Wireless Internet","Air ...	6	1.0	Real Bed	strict	
25153	18161036	3.663562	House	Private room	{Internet,"Wireless Internet","Air conditionin...	2	1.0	Real Bed	flexible	

5 rows × 29 columns

In [57]:
```
# Basic Overview about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74111 entries, 0 to 74110
Data columns (total 29 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       74111 non-null  int64  
 1   log_price                74111 non-null  float64
 2   property_type            74111 non-null  object 
 3   room_type                74111 non-null  object 
 4   amenities                74111 non-null  object 
 5   accommodates             74111 non-null  int64  
 6   bathrooms                73911 non-null  float64
 7   bed_type                 74111 non-null  object 
 8   cancellation_policy      74111 non-null  object 
 9   cleaning_fee             74111 non-null  bool   
 10  city                     74111 non-null  object 
 11  description              74111 non-null  object 
 12  first_review             58247 non-null  object 
 13  host_has_profile_pic     73923 non-null  object 
 14  host_identity_verified   73923 non-null  object 
 15  host_response_rate       55812 non-null  object 
 16  host_since               73923 non-null  object 
 17  instant_bookable         74111 non-null  object 
 18  last_review              58284 non-null  object 
 19  latitude                 74111 non-null  float64
 20  longitude                74111 non-null  float64
 21  name                     74111 non-null  object 
 22  neighbourhood            67239 non-null  object 
 23  number_of_reviews        74111 non-null  int64  
 24  review_scores_rating     57389 non-null  float64
 25  thumbnail_url            65895 non-null  object 
 26  zipcode                  73145 non-null  object 
 27  bedrooms                 74020 non-null  float64
 28  beds                     73980 non-null  float64
dtypes: bool(1), float64(7), int64(3), object(18)
memory usage: 15.9+ MB
```

In [58]:
```
# About data shape
print(f"Number of Rows/Instances: {df.shape[0]}")
print(f"Number of Columns: {df.shape[1]}")
print('-'*30)
print(f"Number of object columns: {len(df.select_dtypes(include='object').columns)}")
print(f"Number of Int/Float columns: {len(df.select_dtypes(exclude='object').columns)}")

# ---------------------------------------
# - Number of Rows/Instances: This represents the total number of homestay listings in the dataset.
# - Number of Columns: This represents the total number of features or characteristics associated with each listing.
# - Number of object columns: These columns contain categorical or textual data.
# - Number of Int/Float columns: These columns contain numerical data.
# ---------------------------------------
```

```
Number of Rows/Instances: 74111
Number of Columns: 29
------------------------------
Number of object columns: 18
Number of Int/Float columns: 11
```

In [59]:
```
# Define columns representing details about each column
info_cols = [
    'num_of_unique_values',  # Number of unique values in the column
    '%_of_null_values',      # Percentage of null values in the column
    'dtype_of_columns',      # Data type of the column
    'count',                 # Count of non-null values
    'unique',                # Unique values in the column
    'top',                   # Most frequent value
    'freq',                  # Frequency of the most frequent value
    'mean',                  # Mean of numerical values
    'std',                   # Standard deviation of numerical values
    'min',                   # Minimum value
    '25%',                   # 25th percentile
    '50%',                   # 50th percentile (median)
    '75%',                   # 75th percentile
    'max'                    # Maximum value
]

# Get the names of the original columns
cols = df.columns

# Finding all the unique values in each column
unique_values = []

# Iterate through each column and get unique values
for col in df.columns:
    temp = list(df[col].unique())  # Get unique values for the column
    unique_values.append(temp)     # Append unique values to the list
```

In [60]:
```
# Calculate basic descriptive statistics for all columns
descriptive_stats = df.describe(include='all').fillna("not-applicable")
```

In [61]:
```
descriptive_stats
```

Out[61]:

	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleanin
count	74111.0	74111.0	74111	74111	74111	74111.0	73911.0	74111	74111	7
unique	not-applicable	not-applicable	35	3	67122	not-applicable	not-applicable	5	5	
top	not-applicable	not-applicable	Apartment	Entire home/apt	{}	not-applicable	not-applicable	Real Bed	strict	
freq	not-applicable	not-applicable	49003	41310	586	not-applicable	not-applicable	72028	32374	5
mean	11266617.102468	4.782069	not-applicable	not-applicable	not-applicable	3.155146	1.235263	not-applicable	not-applicable	appli
std	6081734.886894	0.717394	not-applicable	not-applicable	not-applicable	2.153589	0.582044	not-applicable	not-applicable	appli
min	344.0	0.0	not-applicable	not-applicable	not-applicable	1.0	0.0	not-applicable	not-applicable	appli
25%	6261964.5	4.317488	not-applicable	not-applicable	not-applicable	2.0	1.0	not-applicable	not-applicable	appli
50%	12254147.0	4.70953	not-applicable	not-applicable	not-applicable	2.0	1.0	not-applicable	not-applicable	appli
75%	16402260.5	5.220356	not-applicable	not-applicable	not-applicable	4.0	1.0	not-applicable	not-applicable	appli
max	21230903.0	7.600402	not-applicable	not-applicable	not-applicable	16.0	8.0	not-applicable	not-applicable	appli

11 rows × 29 columns

In [62]:
```
# Create a DataFrame to store insights about the data
column_Insight_df = pd.DataFrame(
    {
        info_cols[0]: df.nunique().to_list(),      # Number of unique values in each column
        info_cols[1]: df.isnull().mean(),          # Percentage of missing values in each column
        info_cols[2]: list(df.dtypes),             # Data types of each column
        info_cols[3]: list(df.count()),            # Non-null value count in each column
        info_cols[4]: unique_values                # Unique values in each column
    },
    index=cols  # Column names as index
)
```

In [63]:
```
column_Insight_df
```

Out[63]:

	num_of_unique_values	%_of_null_values	dtype_of_columns	count	unique
id	74111	0.000000	int64	74111	[6901257, 6304928, 7919400, 13418779, 3808709,...
log_price	767	0.000000	float64	74111	[5.010635294, 5.129898715, 4.976733742, 6.6200...
property_type	35	0.000000	object	74111	[Apartment, House, Condominium, Loft, Townhous...
room_type	3	0.000000	object	74111	[Entire home/apt, Private room, Shared room]
amenities	67122	0.000000	object	74111	[{"Wireless Internet","Air conditioning",Kitch...
accommodates	16	0.000000	int64	74111	[3, 7, 5, 4, 2, 6, 8, 1, 9, 10, 16, 11, 12, 14...
bathrooms	17	0.002699	float64	73911	[1.0, 1.5, 2.0, nan, 2.5, 3.0, 0.5, 4.5, 5.0, ...
bed_type	5	0.000000	object	74111	[Real Bed, Futon, Pull-out Sofa, Couch, Airbed]
cancellation_policy	5	0.000000	object	74111	[strict, moderate, flexible, super_strict_30, ...
cleaning_fee	2	0.000000	bool	74111	[True, False]
city	6	0.000000	object	74111	[NYC, SF, DC, LA, Chicago, Boston]
description	73474	0.000000	object	74111	[Beautiful, sunlit brownstone 1-bedroom in the...
first_review	2554	0.214057	object	58247	[6/18/2016, 8/5/2017, 4/30/2017, nan, 5/12/201...
host_has_profile_pic	2	0.002537	object	73923	[t, nan, f]
host_identity_verified	2	0.002537	object	73923	[t, f, nan]
host_response_rate	80	0.246913	object	55812	[nan, 100%, 71%, 68%, 67%, 83%, 50%, 90%, 86%,...
host_since	3087	0.002537	object	73923	[3/26/2012, 6/19/2017, 10/25/2016, 4/19/2015, ...
instant_bookable	2	0.000000	object	74111	[f, t]
last_review	1371	0.213558	object	58284	[7/18/2016, 9/23/2017, 9/14/2017, nan, 1/22/20...
latitude	74058	0.000000	float64	74111	[40.69652363, 40.76611542, 40.80810999, 37.772...
longitude	73973	0.000000	float64	74111	[-73.99161685, -73.98903992, -73.94375584, -12...
name	73350	0.000000	object	74111	[Beautiful brownstone 1-bedroom, Superb 3BR Ap...
neighbourhood	619	0.092726	object	67239	[Brooklyn Heights, Hell's Kitchen, Harlem, Low...
number_of_reviews	371	0.000000	int64	74111	[2, 6, 10, 0, 4, 3, 15, 9, 159, 82, 29, 13, 12...
review_scores_rating	54	0.225635	float64	57389	[100.0, 93.0, 92.0, nan, 40.0, 97.0, 99.0, 90....
thumbnail_url	65883	0.110861	object	65895	[https://a0.muscache.com/im/pictures/6d7cbbf7-...
zipcode	669	0.013035	object	73145	[11201, 10019, 10027, 94117, 20009, 94131, 902...
bedrooms	11	0.001228	float64	74020	[1.0, 3.0, 2.0, 0.0, 4.0, nan, 5.0, 6.0, 7.0, ...
beds	18	0.001768	float64	73980	[1.0, 3.0, 2.0, 7.0, 4.0, 6.0, 5.0, nan, 10.0,...

Evidently monitiring

In [11]:
```
# !pip install evidently
```

In [12]:
```
# import pandas as pd
# from sklearn import datasets
# from evidently.ui.workspace.cloud import CloudWorkspace
# from evidently.report import Report
# from evidently.metric_preset import DataQualityPreset, DataDriftPreset, TargetDriftPreset, RegressionPreset
```

In [13]:
```
# # Load the data for model monitoring from the provided CSV file
# monitoring_data = df.copy()

# # Rename the target column to 'target' for clarity
# monitoring_data.rename(columns={'log_price': 'target'}, inplace=True)
```

In [14]:
```
# # Create a CloudWorkspace instance with the provided token and URL
# workspace = CloudWorkspace(token="dG9rbgHfd9TQMR1Mup4FnQPNmSJvhu/fCmZ54q/kmiB6eZLw3wBQcLmwfGJhMDKPeCb9nntFsnf7pnH4oS5SDsmyl7k2g
#                            url="https://app.evidently.cloud")

# # Create a new project within the workspace for Home Stays Model Development
# project = workspace.create_project("Home Stays Model Dev")

# # Add a description to the project
# project.description = "Developing the model for production"

# # Save the changes to the project
# project.save()
```

In [15]:
```
# # Splitting the training data into reference data and current data
# reference_data = monitoring_data[:int(len(monitoring_data) * 0.8)]  # 80% of the training data for reference
# current_data = monitoring_data[int(len(monitoring_data) * 0.8):]     # Remaining 20% of the training data for current analysis
```

In [16]:
```
# # Create a data report object with data quality metrics
# data_report = Report(
#     metrics=[
#         DataQualityPreset(),
#     ]
# )

# # Run the data quality metrics on the reference and current data
# data_report.run(reference_data=reference_data, current_data=current_data)

# # Add the data report to the project in the workspace
# workspace.add_report(project_id=project.id, report=data_report)
```

In [17]:
```
# data_report
```

In [18]:
```
# data_drift_report = Report(
#        metrics=[
#            DataDriftPreset(),
#        ],
#     )
# data_drift_report.run(reference_data=reference_data, current_data=current_data)
# workspace.add_report(project.id, data_drift_report)
```

In [19]:
```
# data_drift_report
```

In [20]:
```
# num_target_drift_report = Report(metrics=[
#     TargetDriftPreset(),
# ])

# num_target_drift_report.run(reference_data=reference_data, current_data=current_data)
# workspace.add_report(project.id, num_target_drift_report)
```

In [21]:
```
# num_target_drift_report
```

In [22]:
```
# from evidently.metric_preset import RegressionPreset

# reg_performance_report = Report(metrics=[
#     RegressionPreset(),
# ])

# reg_performance_report.run(reference_data=reference_data, current_data=current_data)
# workspace.add_report(project.id, reg_performance_report)
```

In [23]:
```
# reg_performance_report
```

Data Cleaning and Preparation:

Real-world data often contains inconsistencies, missing values, errors, and formatting issues.

> Cleaning the data to ensure to its accuracy and usability:

1. Identifying and handle missing values (deletion, imputation).
2. Correcting the errors and inconsistencies.
3. Standardizeing data formats (dates, currencies, etc.).
4. Removeing the irrelevant or duplicate data points.

functions for checking outliers

In [76]:
```
# Method 1: Frequency Table and Visualization
def check_outliers_freq_table(data, col):
  # Create frequency table
  frequency_table = data[col].value_counts().reset_index(name='Frequency')
  frequency_table['Relative Frequency'] = frequency_table['Frequency'] / len(data)

  # Print frequency table
  print("Frequency Table:")
  print(frequency_table.to_string(index=True))

  # Visualize distribution (optional)
  frequency_table.plot(kind='bar', x=f'{col}', y='Frequency', title='Frequency Distribution')
  plt.show()

  # Identify potential outliers based on frequency (adjust threshold as needed)
  outlier_threshold = 0.1  # Adjust this based on your data and analysis goals
  outliers = frequency_table[frequency_table['Relative Frequency'] < outlier_threshold][col].tolist()
  print("\nPotential outliers based on frequency (less than", outlier_threshold, "relative frequency):")
  print(outliers)
```

```
# Method 2: IQR (for ordinal data)
def check_outliers_iqr(data):
  if not pd.api.types.is_categorical_dtype(data):
    print("IQR method is only applicable for ordinal data (categories with natural order).")
    return

  # Check if categories have an order (optional)
  categories = data.unique()
  if not all(categories[i] <= categories[i+1] for i in range(len(categories)-1)):
    print("Categories in your data do not seem to have a natural order. IQR method might not be suitable.")
    return

  # Calculate IQR
  Q1 = data.quantile(0.25)
  Q3 = data.quantile(0.75)
  IQR = Q3 - Q1

  # Define upper and lower bounds
  upper_bound = Q3 + (1.5 * IQR)
  lower_bound = Q1 - (1.5 * IQR)

  # Identify outliers using IQR rule
  outliers = data[(data > upper_bound) | (data < lower_bound)].tolist()
  print("\nPotential outliers based on IQR rule:")
  print(outliers)
```

functions for plotting

In [77]:

```
def plot_kde(data, column, shade=True, color="b", bw_method=0.5, xlabel=None, ylabel=None):
    """
    Create a KDE plot using Seaborn.

    Parameters:
    - data (DataFrame): The DataFrame containing the data to be plotted.
    - column (str): The column name for which KDE plot will be created.
    - shade (bool, optional): Whether to shade the area under the KDE curve. Default is True.
    - color (str, optional): The color of the KDE curve. Default is "b" (blue).
    - bw_method (float, optional): The bandwidth of the KDE. Default is 0.5.
    - xlabel (str, optional): Label for the x-axis. Default is None (uses column name).
    - ylabel (str, optional): Label for the y-axis. Default is None.

    Returns:
    - None
    """
    sns.set(style="whitegrid")  # Setting the style
    plt.figure(figsize=(10, 8))  # Setting figure size

    # Creating KDE plot
    sns.kdeplot(data[column], shade=shade, color=color, bw_method=bw_method)

    # Adding labels and title
    if not xlabel:
        xlabel = column  # Use column name as xlabel if not specified
    plt.xlabel(xlabel)
    if ylabel:
        plt.ylabel(ylabel)
    plt.title(f"{column} KDE Plot")  # Adding column name to the title

    # Display the plot
    plt.show()


def plot_kde_before_after_imputation(original_data, imputed_data, column, bw_method=0.5):
    """
    Create a KDE plot comparing the distribution before and after imputation.

    Parameters:
    - original_data (Series or DataFrame): The original data before imputation.
    - imputed_data (Series or DataFrame): The imputed data after imputation.
    - column (str): The column name for which KDE plot will be created.
    - bw_method (float, optional): The bandwidth of the KDE. Default is 0.5.

    Returns:
    - None
    """
    sns.set(style="whitegrid")  # Setting the style
    plt.figure(figsize=(10, 6))  # Setting figure size

    # Before imputation (original data)
    sns.kdeplot(original_data[column], label='Before Imputation', shade=True, color="b", bw_method=bw_method)

    # After imputation
    sns.kdeplot(imputed_data, label='After Imputation', linestyle='--', shade=True, color="g", bw_method=bw_method)

    # Customize the plot
    plt.title(f'Distribution of {column} (Before and After Imputation)')
    plt.xlabel(column)
    plt.ylabel('Density')
```

```
    plt.legend(title='Data')

    # Show the plot
    plt.show()


def create_seaborn_countplot(data, x, palette='viridis', title='Count Plot'):
    """
    Create a count plot using Seaborn with a specified color palette.

    Parameters:
    - data (DataFrame): The DataFrame containing the data to be plotted.
    - x (str): The column name of the categorical variable.
    - palette (str or list of colors, optional): The color palette to use. Default is 'viridis'.
    - title (str, optional): The title of the plot. Default is 'Count Plot'.

    Returns:
    - None
    """
    plt.figure(figsize=(33, 16))
    sns.countplot(x=x, data=data, palette=palette)
    plt.title(title)
    plt.show()


import plotly.express as px

def create_plotly_countplot(data, x, color='green', title='Count Plot'):
    """
    Create a count plot using Plotly Express with a specified color.

    Parameters:
    - data (DataFrame): The DataFrame containing the data to be plotted.
    - x (str): The column name of the categorical variable.
    - color (str, optional): The color to use. Default is 'green'.
    - title (str, optional): The title of the plot. Default is 'Count Plot'.

    Returns:
    - None
    """
    fig = px.histogram(data, x=x, color_discrete_sequence=[color],text_auto=True)
    fig.update_layout(title=title)
    fig.show()



def plot_barplot(df, xcol, ycol, title=None):
    """
    Create a barplot for a specified column in a DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.
        xcol (str): The name of the column to use for the x-axis.
        ycol (str): The name of the column to use for the y-axis.
        title (str, optional): The title of the plot. Defaults to None.

    Returns:
        matplotlib.axes._subplots.AxesSubplot: The created barplot.
    """
    # Set Seaborn style
    sns.set_style("whitegrid")

    # Create barplot
    plt.figure(figsize=(12, 8))  # Adjust size as needed
    ax = sns.barplot(x=df[xcol], y=df[ycol])

    # Add labels and title
    ax.set_xlabel(xcol)
    ax.set_ylabel('Count')
    ax.set_title(title if title else f'Barplot of {xcol}')

    # Rotate x-axis labels if needed
    ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

    # Show plot
    plt.show()

    return ax


def plot_correlation_heatmap(df, columns):
    """
    Plot a heatmap showing the correlation between specified columns in a DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.
        columns (list): A list of column names to include in the heatmap.

    Returns:
        None
```

```
    """
    # Set Seaborn style
    sns.set_style("whitegrid")

    # Calculate correlation matrix
    corr_matrix = df[columns].corr()

    # Plot heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")

    # Add title
    plt.title('Correlation Heatmap')

    # Show plot
    plt.show()


def calculate_mode(df, groupby_column, aggregate_columns):
    """
    Calculate the mode for each specified column grouped by another column.

    Parameters:
        df (DataFrame): The input DataFrame.
        groupby_column (str): The name of the column to group by.
        aggregate_columns (list): A list of column names to calculate the mode for.

    Returns:
        DataFrame: A DataFrame containing the modes for each specified column grouped by the groupby_column.
    """
    mode_data = {}
    for column in aggregate_columns:
        mode_data[f'mode_{column}'] = df.groupby(groupby_column)[column].agg(lambda x: x.mode().iloc[0])
    return pd.DataFrame(mode_data)


import seaborn as sns
import matplotlib.pyplot as plt

def plot_correlation_heatmap(df, columns):
    """
    Plot a heatmap showing the correlation between specified columns in a DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.
        columns (list): A list of column names to include in the heatmap.

    Returns:
        None
    """
    # Set Seaborn style
    sns.set_style("whitegrid")

    # Calculate correlation matrix
    corr_matrix = df[columns].corr()

    # Plot heatmap
    plt.figure(figsize=(33, 16))
    sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")

    # Add title
    plt.title('Correlation Heatmap')

    # Show plot
    plt.show()


def plot_lineplot(df, x_column, y_column):
    """
    Create a line plot for two specified columns in a DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.
        x_column (str): The name of the column for the x-axis.
        y_column (str): The name of the column for the y-axis.

    Returns:
        None
    """
    # Set Seaborn style
    sns.set_style("whitegrid")

    # Create line plot
    plt.figure(figsize=(33, 16))  # Adjust size as needed
    sns.lineplot(data=df, x=x_column, y=y_column)

    # Add labels and title
    plt.xlabel(x_column)
    plt.ylabel(y_column)
    plt.title(f'Line Plot of {y_column} vs {x_column}')
```

```
    # Show plot
    plt.show()
```

Utils

In [78]:
```
def analyze_skewness(data, column):
    """
    Calculate skewness of a column in a DataFrame and provide interpretation.

    Parameters:
    - data (DataFrame): The DataFrame containing the data.
    - column (str): The column name for which skewness will be calculated and interpreted.

    Returns:
    - None
    """
    skewness = data[column].skew()
    print("Skewness of the dataset:", skewness)

    # Interpretation
    if skewness < -1:
        print("The distribution is highly left-skewed.")
    elif -1 <= skewness < -0.5:
        print("The distribution is moderately left-skewed.")
    elif -0.5 <= skewness < 0.5:
        print("The distribution is approximately symmetric.")
    elif 0.5 <= skewness < 1:
        print("The distribution is moderately right-skewed.")
    else:
        print("The distribution is highly right-skewed.")




def random_imputation(df, column):
    """
    Perform random imputation for missing values in a DataFrame column.

    Parameters:
    - df (DataFrame): The input DataFrame.
    - column (str): The column name containing missing values to be imputed.

    Returns:
    - df_imputed (DataFrame): The DataFrame with missing values imputed using random sampling.
    """
    # Get indices of missing values in the specified column
    missing_indices = df[column].isnull()

    # Get non-missing values in the specified column
    non_missing_values = df.loc[~missing_indices, column]

    # Randomly sample non-missing values to impute missing values
    imputed_values = non_missing_values.sample(n=missing_indices.sum(), replace=True)

    # Update missing values in the specified column
    df_imputed = df.copy()
    df_imputed.loc[missing_indices, column] = imputed_values.values

    return df_imputed



def find_nearest_zipcode(latitude, longitude):
    """
    Find the nearest zip code based on latitude and longitude coordinates.

    Parameters:
    - latitude (float): Latitude coordinate.
    - longitude (float): Longitude coordinate.

    Returns:
    - nearest_zipcode (str): Zip code of the nearest location.
    """
    # Create geolocator object
    geolocator = Nominatim(user_agent="nearest_zipcode_finder")

    # Get the location information based on latitude and longitude
    location = geolocator.reverse((latitude, longitude), exactly_one=True)

    # Extract zip code from location information
    nearest_zipcode = location.raw['address'].get('postcode')

    return nearest_zipcode



def knn_impute(df, column_name, n_neighbors=5):
    """
```

```
    Perform k-Nearest Neighbors (kNN) imputation on a specified column in a DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.
        column_name (str): The name of the column to impute.
        n_neighbors (int): The number of neighbors to consider for imputation.

    Returns:
        DataFrame: The DataFrame with missing values in the specified column imputed using kNN imputation.
    """
    # Create a copy of the DataFrame with only the specified column
    df_missing = df[[column_name]].copy()

    # Instantiate the KNNImputer with the desired number of neighbors
    imputer = KNNImputer(n_neighbors=n_neighbors)

    # Fit the imputer on the data with missing values and transform the data
    imputed_data = imputer.fit_transform(df_missing)

    # Convert the imputed data back to a DataFrame
    imputed_df = pd.DataFrame(imputed_data, columns=[column_name])

    # Round and convert to integer
    imputed_df[column_name] = imputed_df[column_name].round().astype(int)

    # Replace the missing values in the original DataFrame with the imputed values
    df[column_name] = imputed_df[column_name]

    return df
```

Taking out null columns

In [79]:
```
# Method 1: Using isnull().any()
null_columns_1 = df.columns[df.isnull().any()]
print(null_columns_1)
```

```
Index(['first_review', 'host_has_profile_pic', 'host_identity_verified',
       'host_response_rate', 'host_since', 'last_review', 'neighbourhood',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds'],
      dtype='object')
```

In [80]:
```
df.isnull().sum()
```

Out[80]:
```
id                          0
log_price                   0
property_type               0
room_type                   0
amenities                   0
accommodates                0
bathrooms                   0
bed_type                    0
cancellation_policy         0
cleaning_fee                0
city                        0
description                 0
first_review            15864
host_has_profile_pic      188
host_identity_verified    188
host_response_rate      18299
host_since                188
instant_bookable            0
last_review             15827
latitude                    0
longitude                   0
name                        0
neighbourhood            6872
number_of_reviews           0
review_scores_rating    16722
thumbnail_url            8216
zipcode                   966
bedrooms                   91
beds                      131
amenities_count             0
has_wireless_internet       0
has_kitchen                 0
has_heating                 0
has_essentials              0
has_smoke_detector          0
dtype: int64
```

Handling Each column

Column Bathrooms

Note

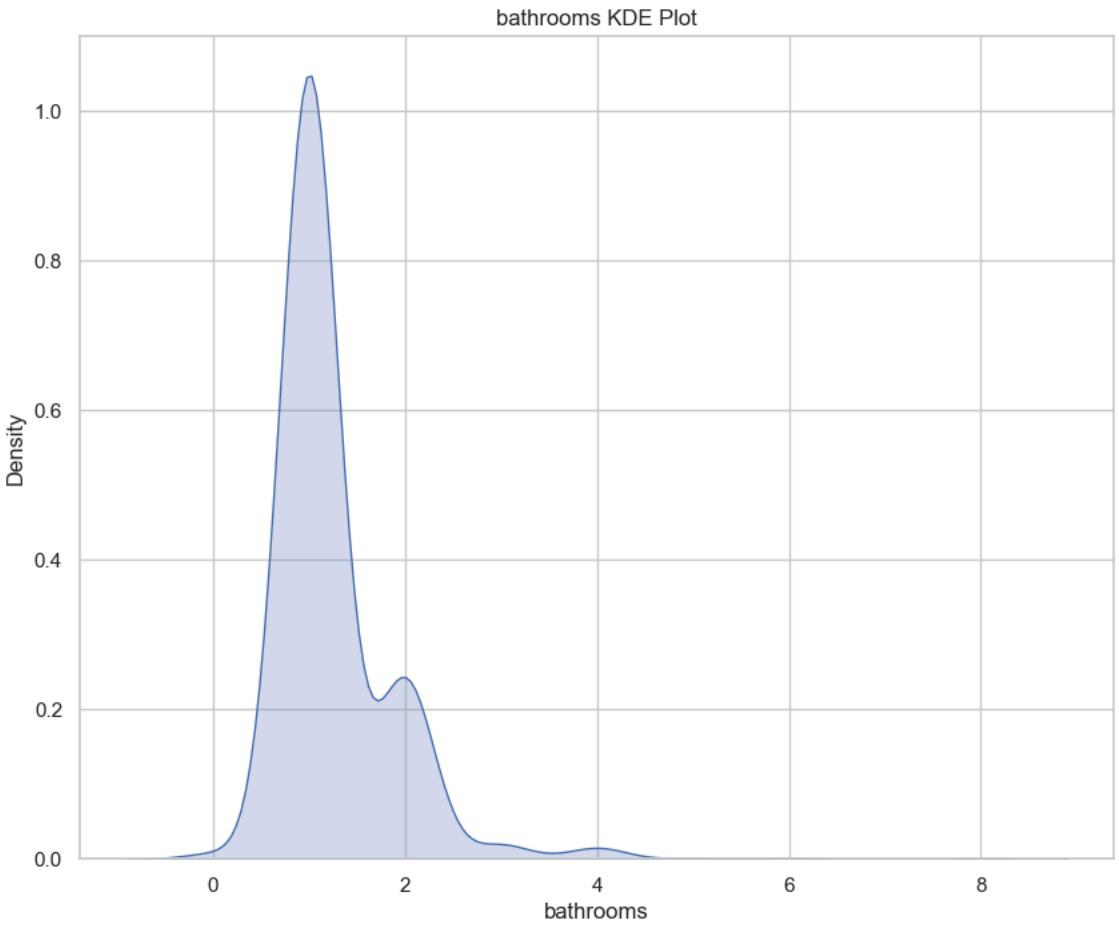
In real estate or housing datasets, having fractional values in the number of bathrooms can represent scenarios like having a bathroom with both a bathtub and a shower (counted as 1.5 bathrooms) or having a bathroom with additional features like a jacuzzi (counted as 2.5 bathrooms).

If you want to handle these fractional values, you can either:

1. **Round them**: Round the fractional values to the nearest integer, treating them as whole numbers of bathrooms.
2. **Keep them as is**: Preserve the fractional values, indicating partial bathrooms.

so i going with method 2 (i.e : **Keep them as is**)

In [81]:
```
plot_kde(df, column='bathrooms', shade=True, color="b", bw_method=0.5, ylabel="Density")
```



Calculating skewness

In [82]:
```
analyze_skewness(df, column='bathrooms')
```

```
Skewness of the dataset: 3.475813822494678
The distribution is highly right-skewed.
```

Data is Missing at Random (MAR)

- The data is highly right skewed(as it can amplify skewness) & continuous distribution, so using random imputation not best choice
- Iam going with Mode imputation
- In future i will use model based imputation for now iam going with mode

In [83]:
```
df = knn_impute(df, 'bathrooms', n_neighbors=5)
```

In [87]:
```
df['bathrooms']
```

Out[87]:
```
0        1
1        1
2        1
3        1
4        1
       ..
74106    1
74107    2
74108    1
74109    1
74110    1
Name: bathrooms, Length: 74111, dtype: int32
```

Checking for outliers

In [89]:
```
# check_outliers_freq_table(df,'bathrooms')
# check_outliers_iqr(df['bathrooms'])
```

Report :

- Handled missing data using mode imputation (iwill use more advanced techniques in future notebook)
- By using frequency tables i saw there are no outliers

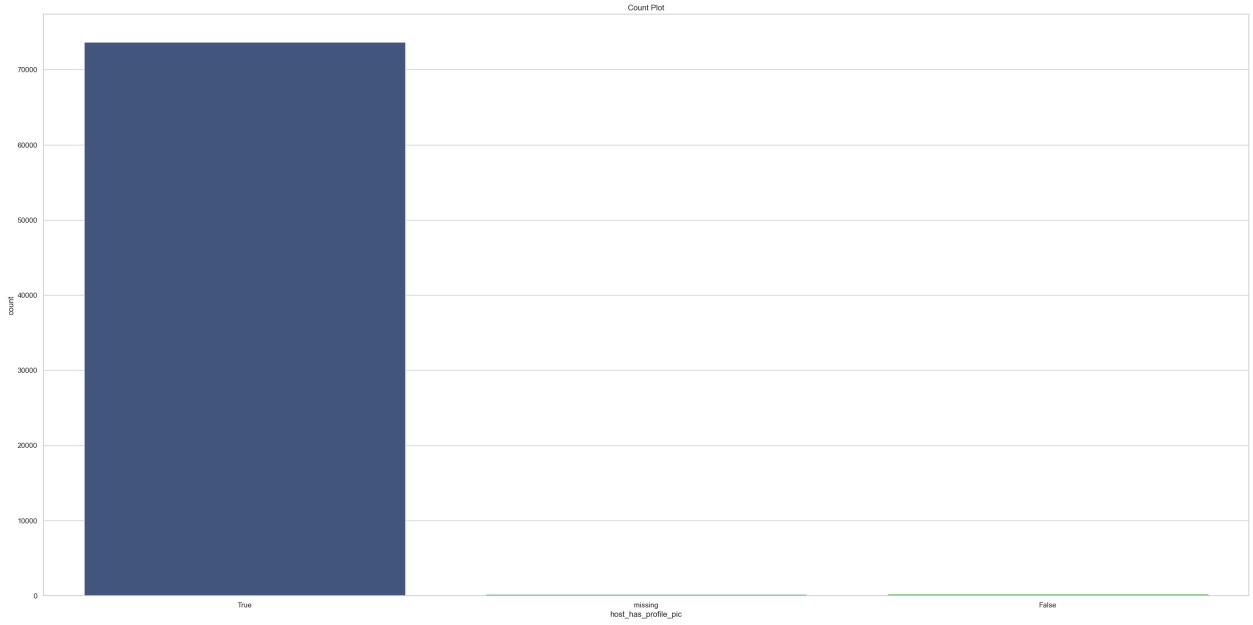
Columns: host_has_profile_pic host_identity_verified

replacing the null value with new category

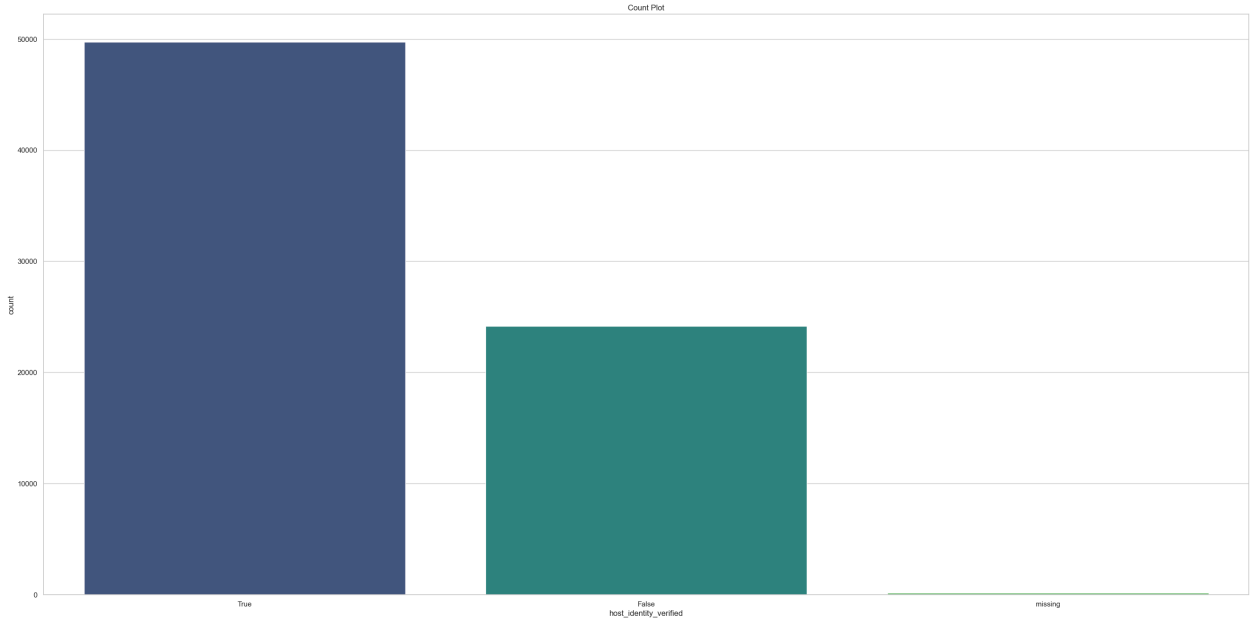
In [90]:
```
# df[['host_has_profile_pic','host_identity_verified']].fillna('m',inplace=True)  This one is not working
# reason: the there was a change in synatx of null values hence it doesnt work


# m : Missing value
df[['host_has_profile_pic','host_identity_verified']] = df[['host_has_profile_pic','host_identity_verified']].replace(np.nan, 'm'


# Changed the names of the values
df[['host_has_profile_pic','host_identity_verified']] = df[['host_has_profile_pic','host_identity_verified']].replace({'t':True,
```

Plotting the bar plot

In [91]:
```
create_seaborn_countplot(df, 'host_has_profile_pic')
```



In [92]:
```
create_seaborn_countplot(df, 'host_identity_verified')
```



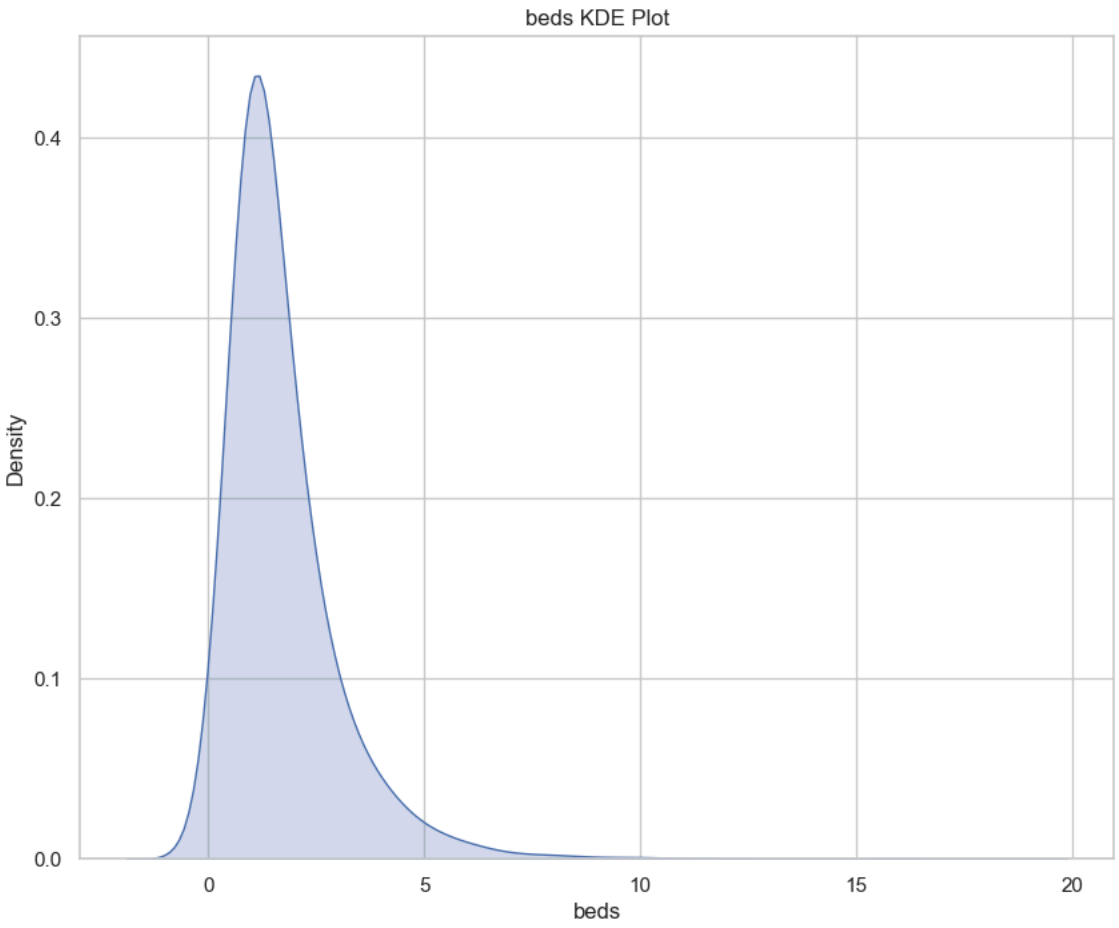
Report:

1. There are Equal missing values both columns
2. hence it is a categorical column if i use any other techniques for imputation it may leads lo bais
3. so i replace the null values with the new category
4. we can clearly see there are no outliers in the dataset

Column Beds

column Bathrooms

In [93]:
```
plot_kde(df, column='beds', shade=True, color="b", bw_method=0.5, ylabel="Density")
```



Calculating skewness

In [94]:
```
analyze_skewness(df, column='beds')
```

```
Skewness of the dataset: 3.358000200663669
The distribution is highly right-skewed.
```

Mode Imputing

In [95]:
```
df = knn_impute(df, 'beds', n_neighbors=5)
```

check the outliers

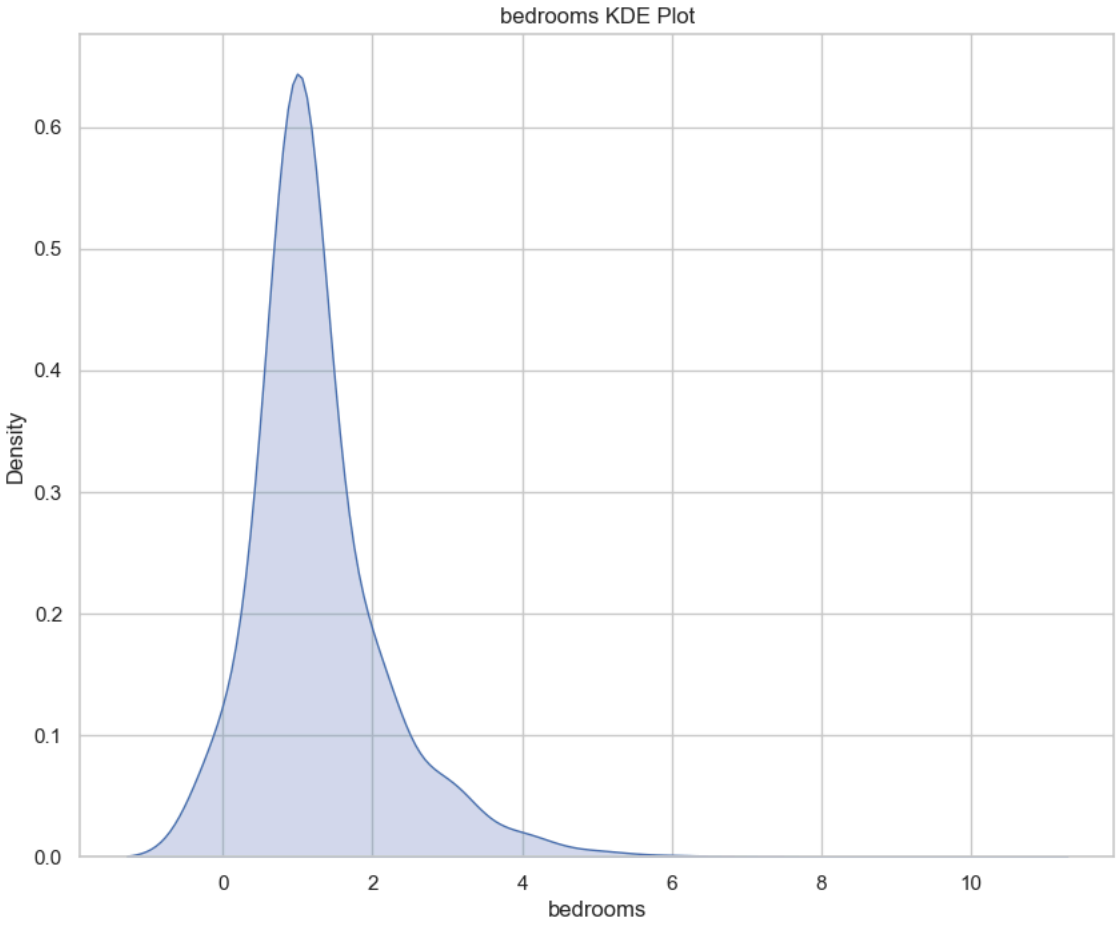
In [97]:
```
# check_outliers_freq_table(df,'beds')
# check_outliers_iqr(df['beds'])
```

Column Bedroom

Plotting KDE for Bedrooms to know which impuation can be best

1. List item
2. List item

In [98]:
```
plot_kde(df, column='bedrooms', shade=True, color="b", bw_method=0.5, ylabel="Density")
```



Calculate skewness

In [99]:
```
analyze_skewness(df, column='bedrooms')
```

```
Skewness of the dataset: 1.989848743689331
The distribution is highly right-skewed.
```

Mode Imputing

In [100…
```
df = knn_impute(df, 'bedrooms', n_neighbors=5)
```

Checking outliers

In [102…
```
# check_outliers_freq_table(df,'bedrooms')

# check_outliers_iqr(df['bedrooms'])
```

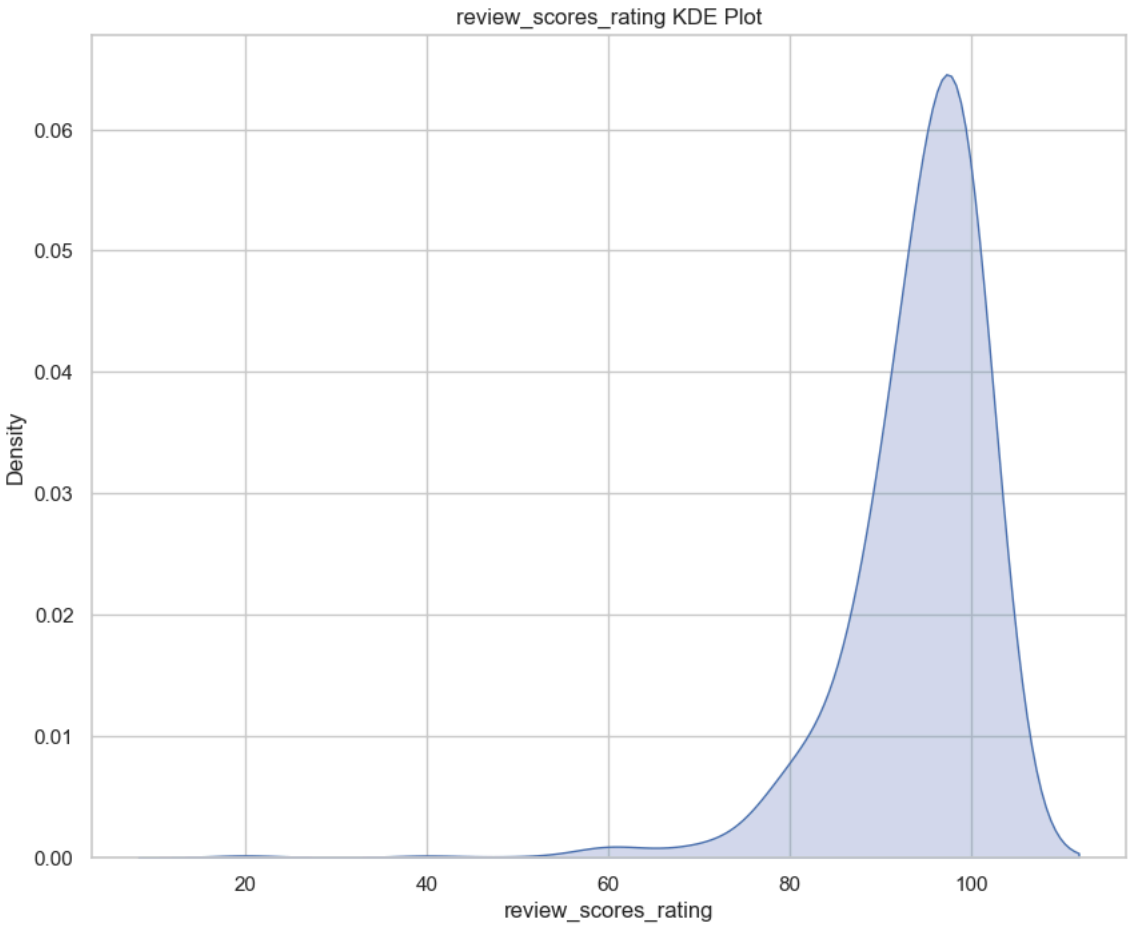
Report :

- Handled missing data using mode imputation (iwill use more advanced techniques in future notebook)
- By using frequency tables i saw there are no outliers

In [103…
```
df['review_scores_rating'].value_counts()
```

Out[103]:
```
100.0    16215
98.0      4374
97.0      4087
96.0      4081
95.0      3713
93.0      3647
90.0      2852
99.0      2631
94.0      2618
80.0      2163
92.0      2064
91.0      1615
89.0      1120
87.0      1119
88.0      1056
85.0       625
86.0       512
60.0       444
84.0       438
83.0       403
82.0       211
70.0       196
73.0       157
81.0       126
75.0       101
20.0        97
78.0        94
40.0        90
79.0        83
76.0        76
77.0        74
67.0        66
74.0        39
72.0        38
50.0        30
65.0        28
68.0        20
71.0        14
69.0        13
63.0        11
53.0        10
64.0        10
47.0         5
30.0         4
62.0         3
66.0         3
55.0         3
57.0         3
27.0         2
35.0         1
49.0         1
58.0         1
54.0         1
56.0         1
Name: review_scores_rating, dtype: int64
```

In [104…
```
plot_kde(df, column='review_scores_rating', shade=True, color="b", bw_method=0.5, ylabel="Density")
```



In [105…
```
analyze_skewness(df, column='review_scores_rating')
```

```
Skewness of the dataset: -3.3808606233341605
The distribution is highly left-skewed.
```

In [106…
```
df = knn_impute(df, 'review_scores_rating', n_neighbors=5)
```

---

column Thumbnail_url

In [107…
```
# df = df.drop('thumbnail_url', axis=1)
```

Report

Droping this column because it has no significance in the dataset

---

column Host_response_rate

In [108…
```
df['host_response_rate'] = df['host_response_rate'].str.split("%").str[0]
```

In [109…
```
df = random_imputation(df, column='host_response_rate')
```

In [110…
```
df['host_response_rate'] =  df['host_response_rate'].astype('int')
```

Report:

Hence data is continous so better to use random imputation becasuse it adds less bais

---

Datetime columns

In [111…
```
df[['last_review','host_since','first_review']]
```

Out[111]:

	last_review	host_since	first_review
0	7/18/2016	3/26/2012	6/18/2016
1	9/23/2017	6/19/2017	8/5/2017
2	9/14/2017	10/25/2016	4/30/2017
3	NaN	4/19/2015	NaN
4	1/22/2017	3/1/2015	5/12/2015
...	...	...	...
74106	NaN	3/24/2013	NaN
74107	4/15/2017	5/3/2016	8/15/2016
74108	9/10/2017	1/5/2012	1/3/2015
74109	NaN	9/17/2017	NaN
74110	4/30/2017	11/26/2012	9/5/2013

74111 rows × 3 columns

In [112…
```
# Convert multiple columns to datetime format
df[['last_review', 'host_since', 'first_review']] = df[['last_review', 'host_since', 'first_review']].apply(pd.to_datetime)
```

In [113…
```
df[['last_review', 'host_since', 'first_review']].isnull().sum()
```

Out[113]:
```
last_review     15827
host_since        188
first_review    15864
dtype: int64
```

In [114…
```
# Forward fill missing datetime values
df['last_review'] = df['last_review'].fillna(method='ffill')
df['host_since'] = df['host_since'].fillna(method='ffill')
df['first_review'] = df['first_review'].fillna(method='ffill')
```

Report:

since all the columns are datetime

in the begining i used mode imputation but i dont think it is suitable for that task soo

i used forward fill (each missing value is replaced with the last observed non-null value in the same column)

Feature Engineering

Extracting Information from Date Columns:

`first_review and last_review` : Extract features such as review duration or time since last review.

`host_since` : Calculate the tenure of the host.

In [115…
```
# Feature engineering for review duration
# Calculate the duration between the first and last review for each listing, in days
df['review_duration'] = (df['last_review'] - df['first_review']).dt.days

# Feature engineering for time since last review
# Calculate the number of days since the last review was posted
df['time_since_last_review'] = (pd.to_datetime('today') - df['last_review']).dt.days

# Feature engineering for host tenure
# Calculate the tenure of each host measured in days since they joined the platform
df['host_tenure'] = (pd.to_datetime('today') - df['host_since']).dt.days
```

Combining Features:

`number_of_reviews` and `review_scores_rating` : Calculate a new feature representing the average review score.

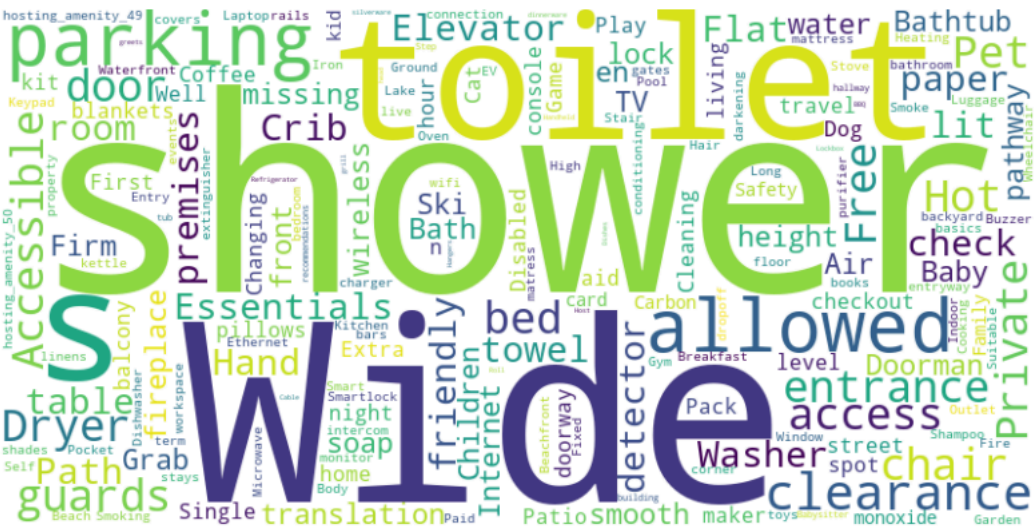
In [116…
```
# Calculate the average review score for each number_of_reviews group
average_review_score = df.groupby('number_of_reviews')['review_scores_rating'].mean().reset_index()
average_review_score.rename(columns={'review_scores_rating': 'average_review_score'}, inplace=True)

# Merge the average_review_score back to the original DataFrame
df = pd.merge(df, average_review_score, on='number_of_reviews', how='left')
```

Extracting the new columns from amenities

Extract information from the amenities list, such as the count of amenities or specific amenities present.

In [117…

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Function to extract amenities from JSON string
def extract_amenities(amenities_str):
    # Remove braces and quotes, then split by comma to get individual amenities
    amenities = amenities_str.strip('{}').replace('"', '').split(',')
    return amenities

# Extract amenities from all rows and collect unique amenities into a set
all_amenities = set()
for amenities_str in df['amenities']:
    amenities_list = extract_amenities(amenities_str)
    all_amenities.update(amenities_list)

# Create a string with all amenities separated by spaces
amenities_text = ' '.join(all_amenities)

# Generate word cloud
wordcloud = WordCloud(width=800, height=400, background_color ='white').generate(amenities_text)

# Display the generated word cloud
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [118…

```
def count_amenities(amenities_str):
    """
    Function to count the number of amenities
    """
    # Remove braces and quotes, then split by comma to get individual amenities
    amenities = amenities_str.strip('{}').replace('"', '').split(',')
    return len(amenities)

# Count the number of amenities for each listing
df['amenities_count'] = df['amenities'].apply(count_amenities)
```

In [119…

```
def extract_amenities(amenities_str):
    """
    Extracts amenities from a JSON string and creates a list of all amenities.

    Parameters:
    amenities_str (str): JSON string containing amenities.

    Returns:
    list: List of amenities extracted from the JSON string.
    """
    amenities = amenities_str.strip('{}').replace('"', '').split(',')
    return amenities

# Extract all amenities and create a list of all amenities across all listings
all_amenities = []
for amenities_str in df['amenities']:
    all_amenities.extend(extract_amenities(amenities_str))

# Calculate the frequency of each amenity
amenities_frequency = pd.Series(all_amenities).value_counts()
```

```
# Select the top 5 most common amenities
top_5_amenities = amenities_frequency.head(5).index.tolist()

print("Top 5 most common amenities:")
print(top_5_amenities)

# copying the list of important amenities
important_amenities = top_5_amenities

def check_important_amenity(amenities_str, important_amenity):
    """
    Checks if an important amenity is present in the amenities string.

    Parameters:
    amenities_str (str): String containing amenities.
    important_amenity (str): The important amenity to check for.

    Returns:
    int: 1 if the important amenity is present, 0 otherwise.
    """
    return 1 if important_amenity in amenities_str else 0

# Create binary columns for each important amenity with a prefix
prefix = 'has_'
for amenity in important_amenities:
    column_name = prefix + amenity.lower().replace(' ', '_')
    df[column_name] = df['amenities'].apply(lambda x: check_important_amenity(x, amenity))
```

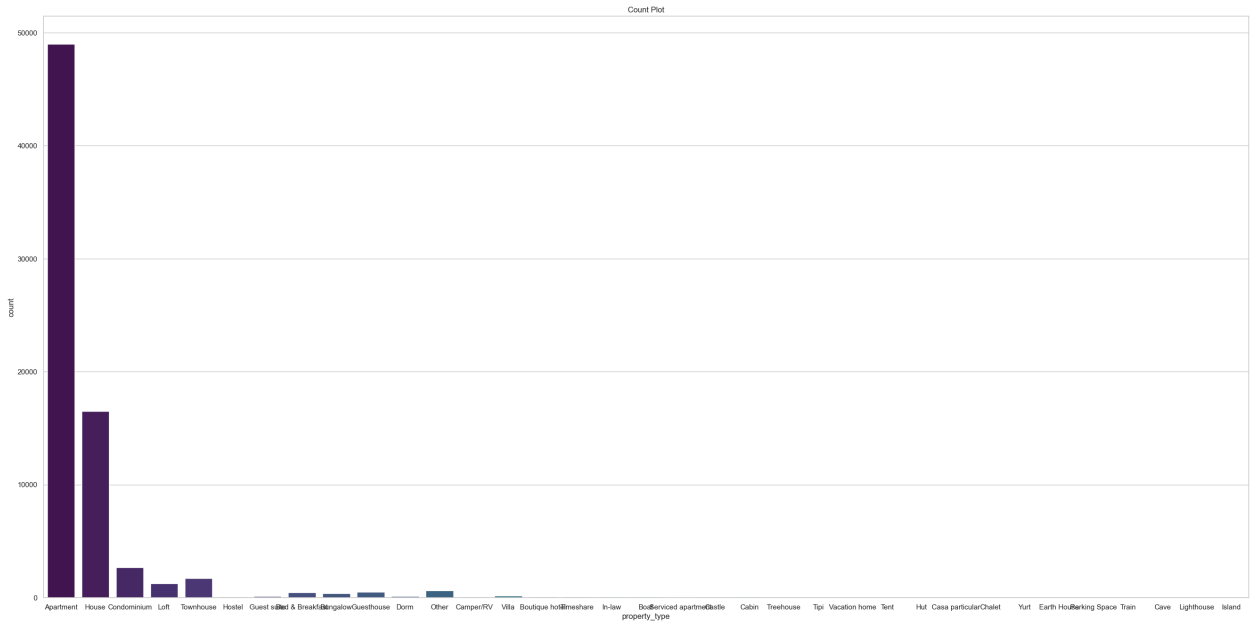
```
Top 5 most common amenities:
['Wireless Internet', 'Kitchen', 'Heating', 'Essentials', 'Smoke detector']
```

Exploratory Data Analysis (EDA)

Univarint Analysis

In [120…

```
create_seaborn_countplot(df, 'property_type')
```

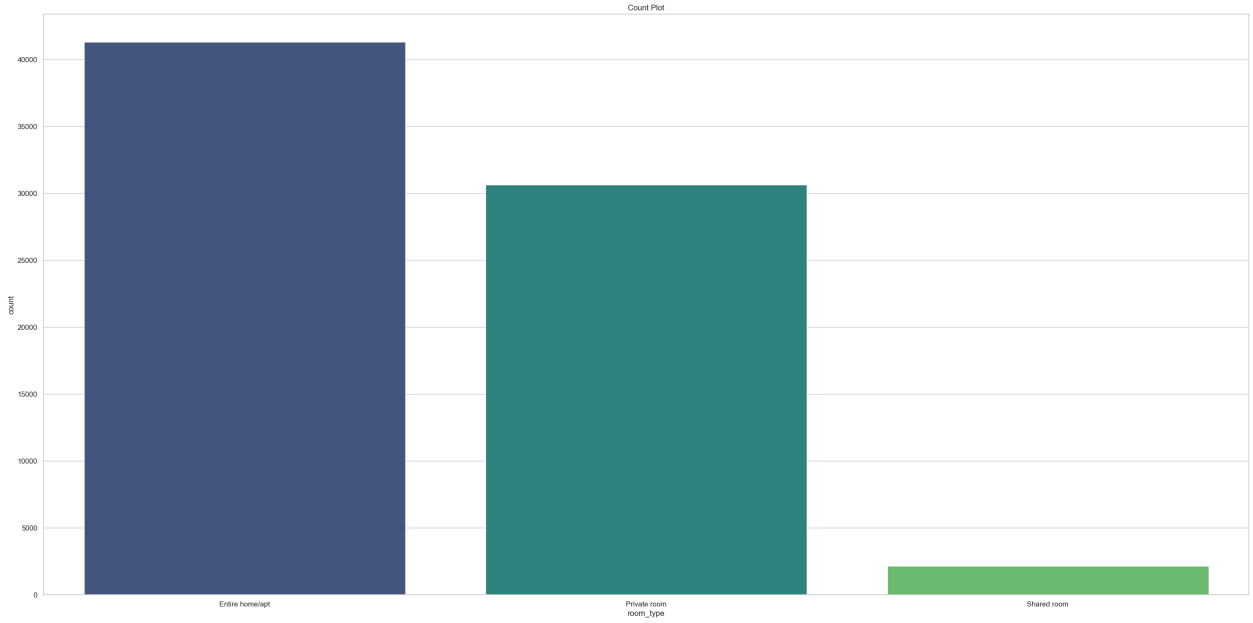


Report:

> - as we can see Apartment,House,condominium are top 3 properties; obviously reasons is most of the type of property are apartments in cities

In [121…

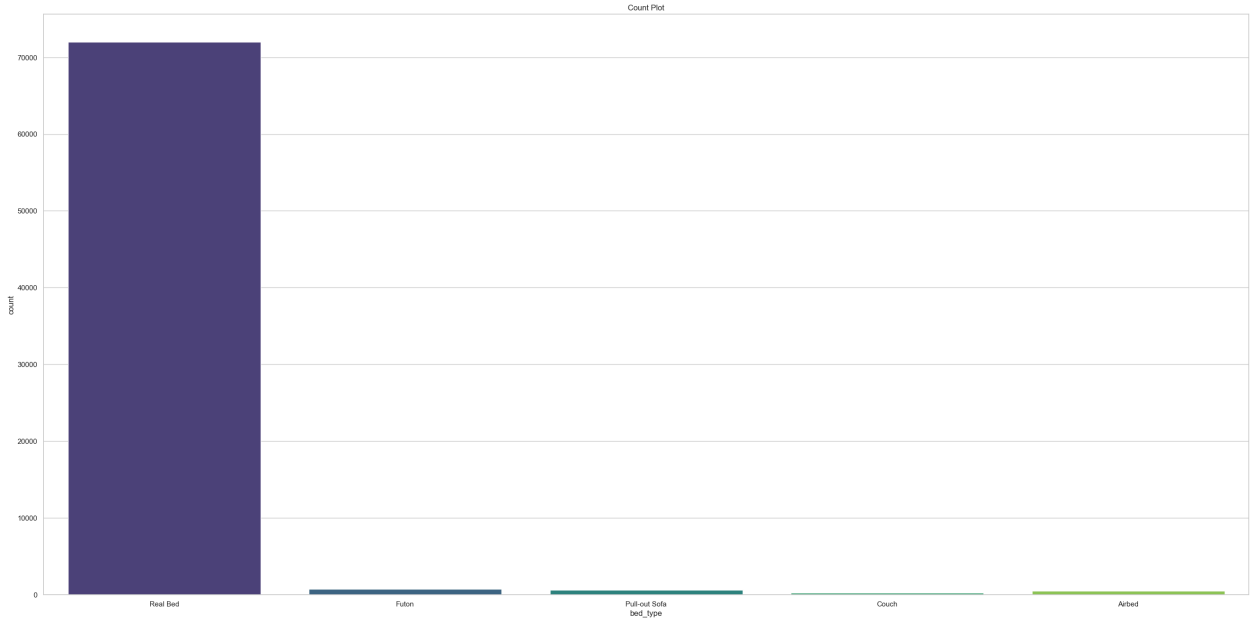
```
create_seaborn_countplot(df, 'room_type')
```



Report :

> - family: choose entire home/apt
> - couples/singles: choose private room
> - friends : shared rooms

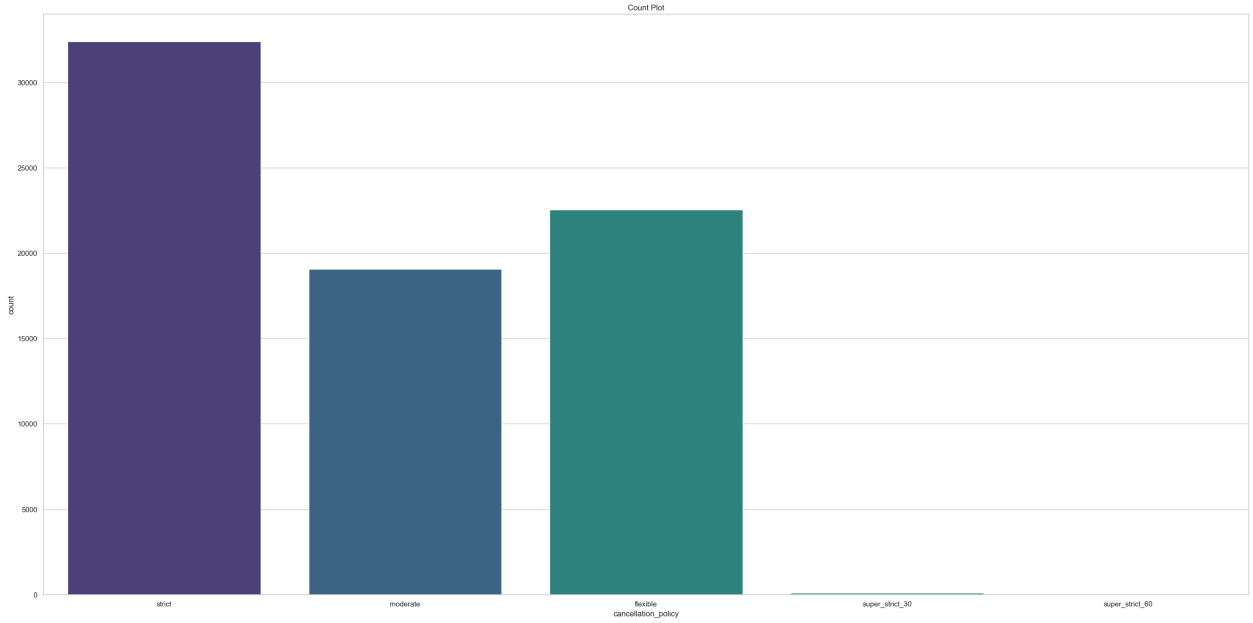
In [122…
```
create_seaborn_countplot(df, 'bed_type')
```



Report:

> We can see almost 97% rooms/homes are equiped with realbeds

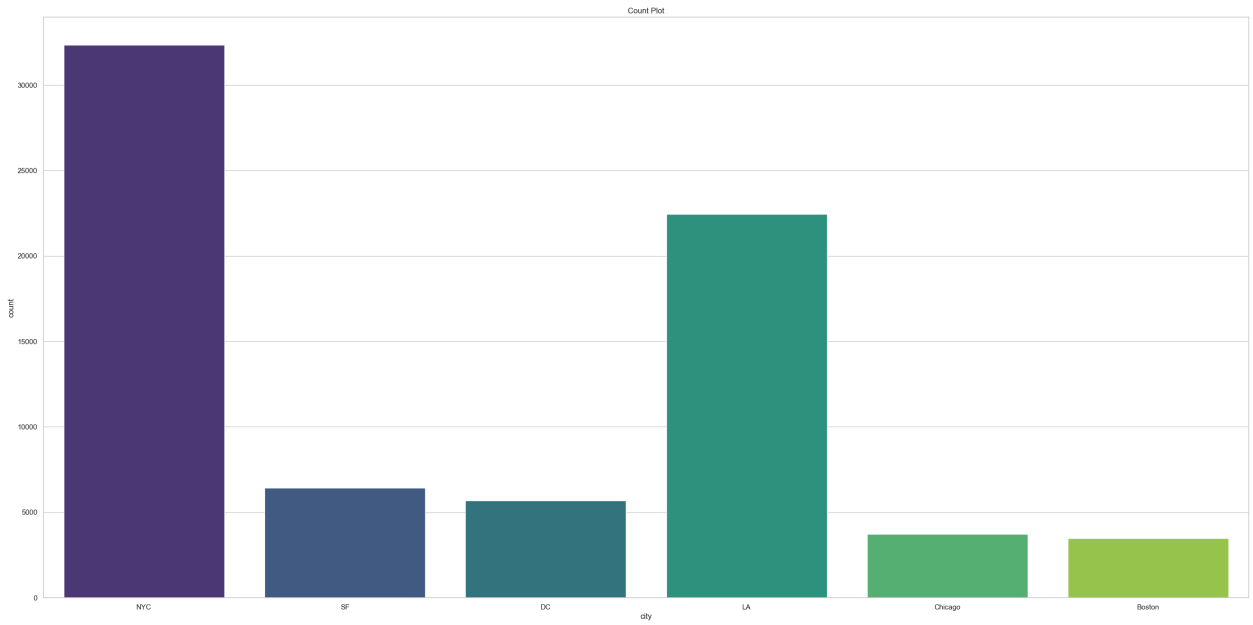
In [123…
```
create_seaborn_countplot(df, 'cancellation_policy')
```



Report:

> we can observe there are 5levels of different cancellation policecs among them `strict, moderate & flexible` are more

In [124…
```
create_seaborn_countplot(df, 'city')
```



Report:

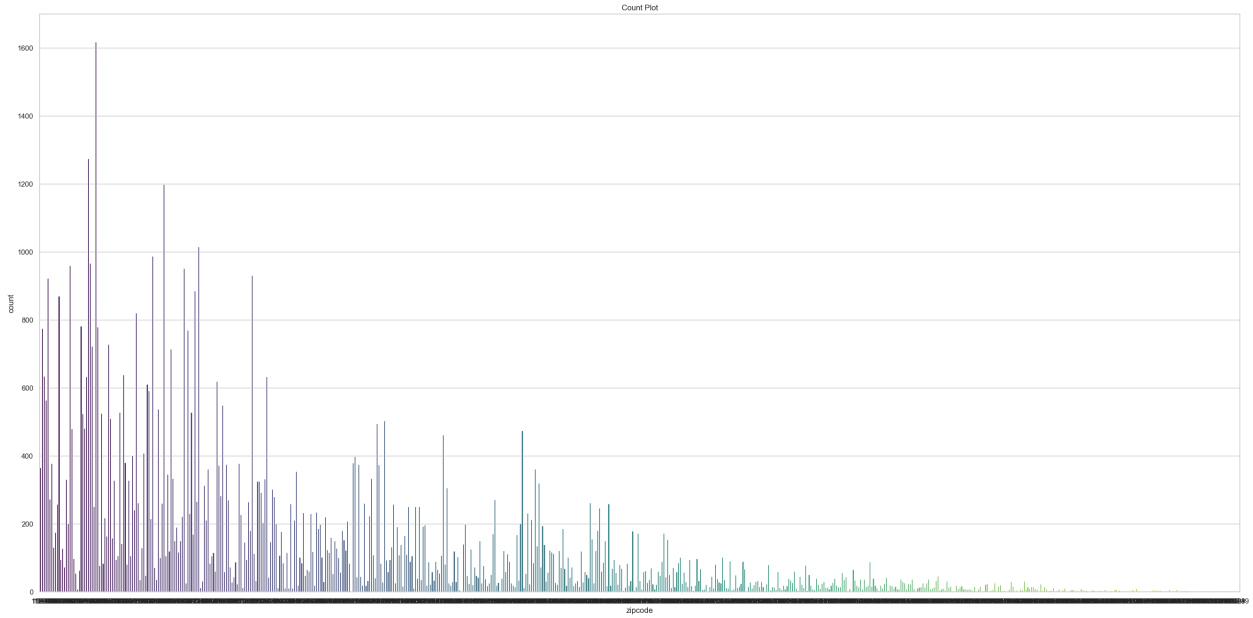
> Newyork city is most of the apartments/homes fallowed by Los angels

> san francisco - DC - have some what equal shares

> chicago & Boston also shares equal in numbers

In [125…
```
df['zipcode'] = df['zipcode'].str.split("-").str[0].str[:].str.split("\r").str[0]

# print('55555-57484\r\r\r\r\r\r\n\r\r\r\r\r\r\n\r\r\r\r\r\r\n94158'.split("-")[0].split("\r").str[0]) #Sample viz of above code
```

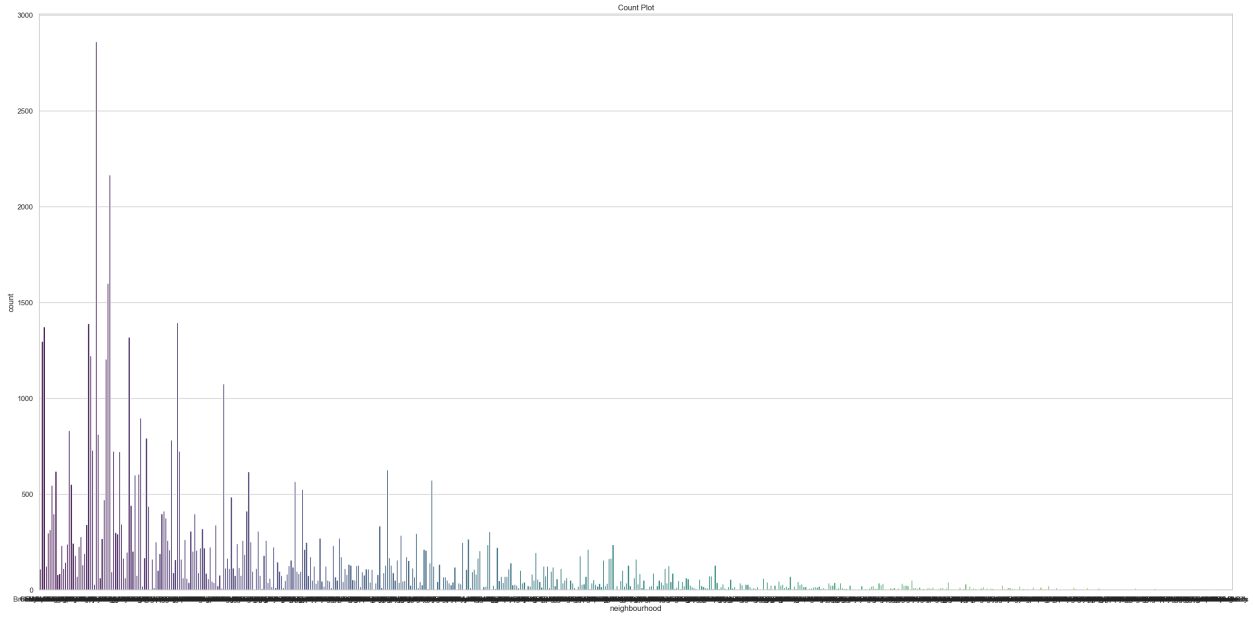
In [126…
```
create_seaborn_countplot(df, 'zipcode')
```



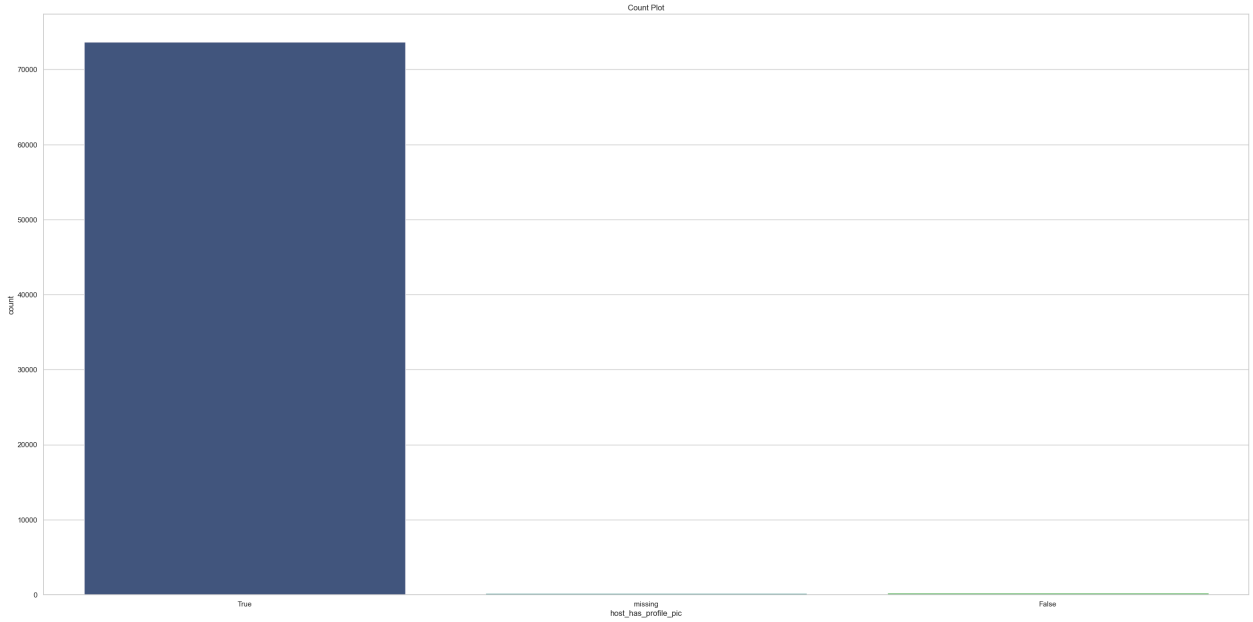
Report

> we can observe wide range of irregular pattern in the zipline count plot

In [127…
```
create_seaborn_countplot(df, 'neighbourhood')
```



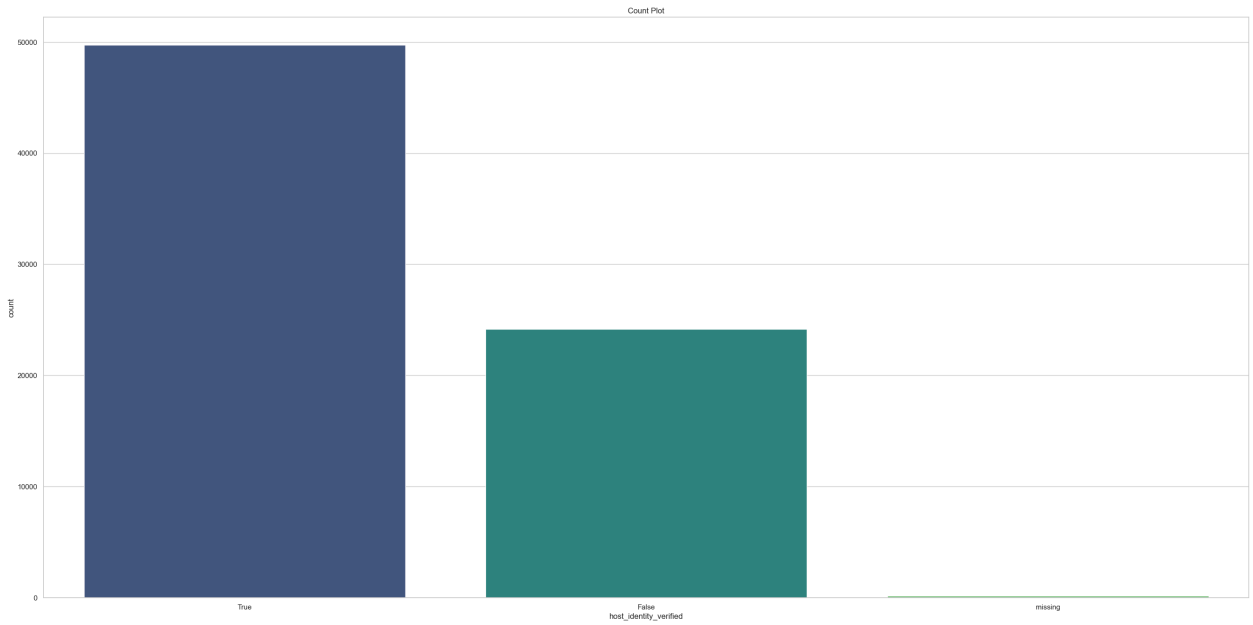
In [128…
```
create_seaborn_countplot(df, 'host_has_profile_pic')
```



Report

> most all them have profile pic

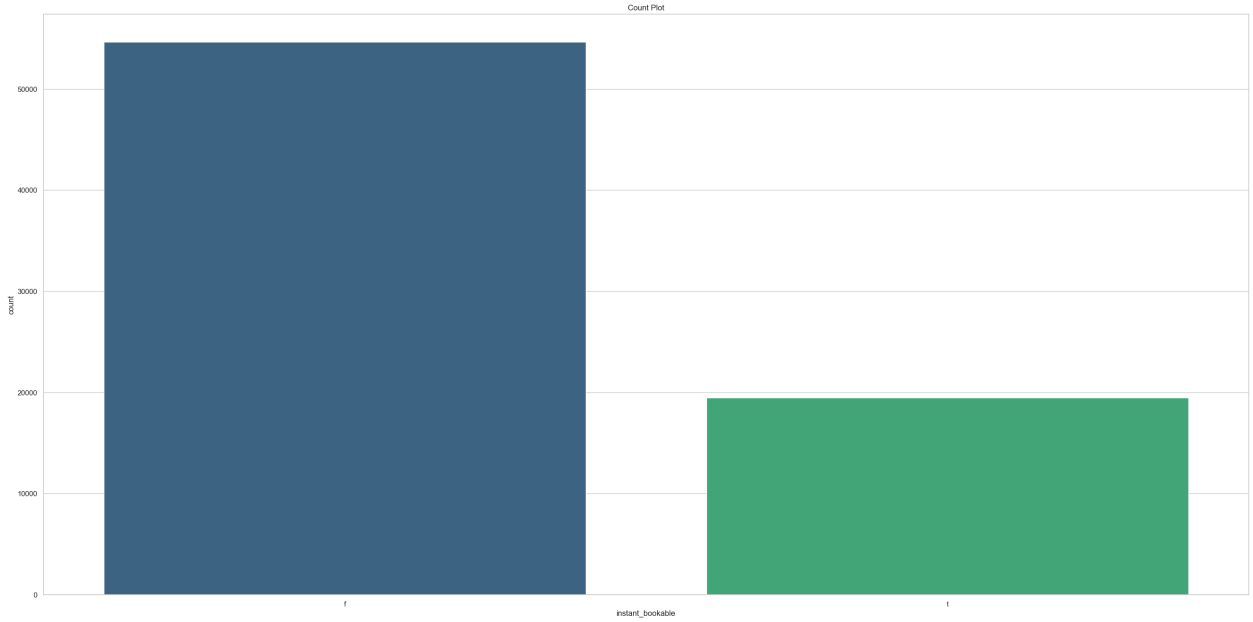
In [129…
```
create_seaborn_countplot(df, 'host_identity_verified')
```



Report:

> Most the hosts are verifed(49.0k) and some them are not verified(24k)

In [130…
```
create_seaborn_countplot(df, 'instant_bookable')
```



Report:

> Nearly 1/3 of them are not offering instanly bookable

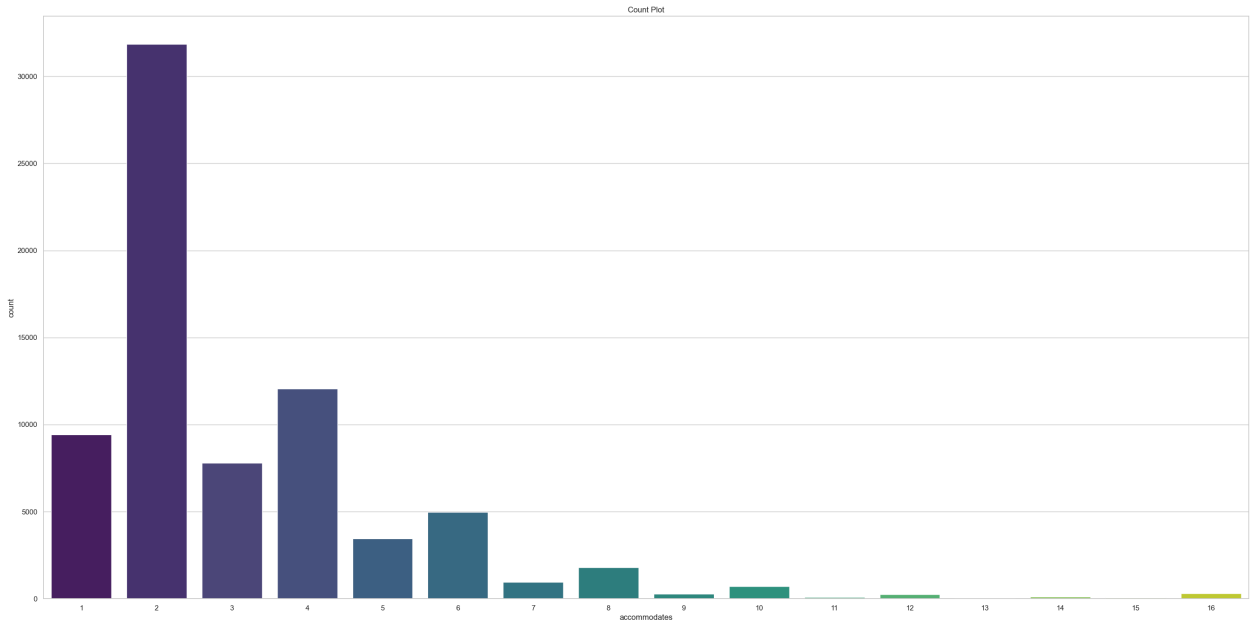
In [131…
```
df.select_dtypes(exclude='object').describe()
```

Out[131]:

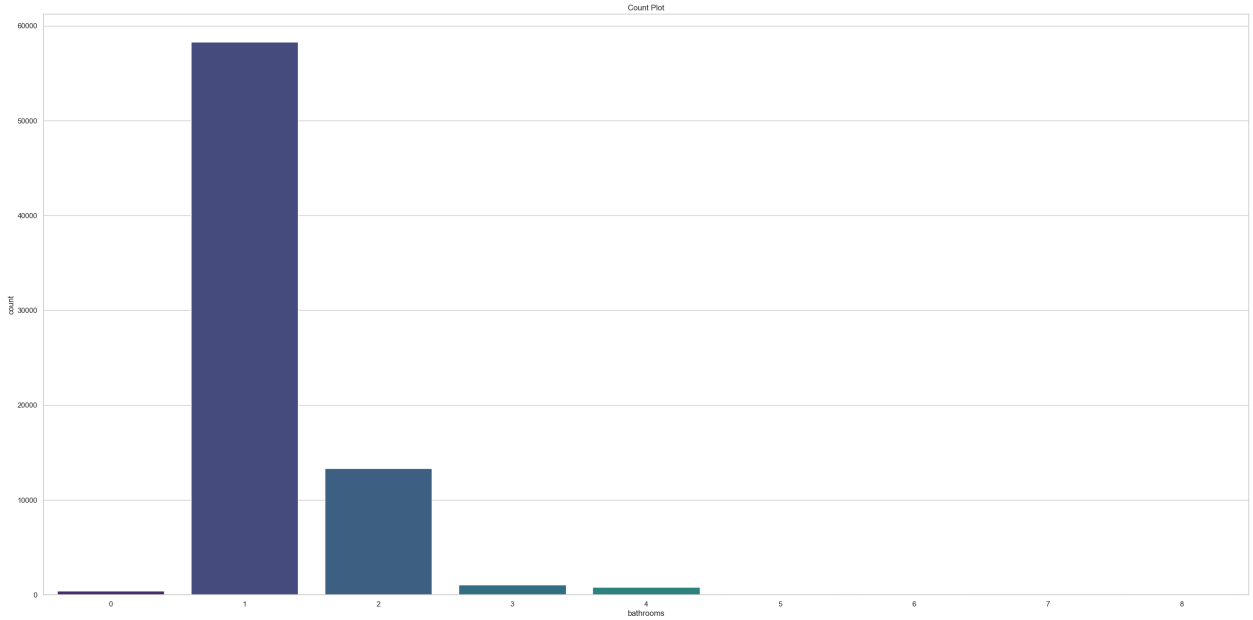
	id	log_price	accommodates	bathrooms	host_response_rate	latitude	longitude	number_of_reviews	review
count	7.411100e+04	74111.000000	74111.000000	74111.000000	74111.000000	74111.000000	74111.000000	74111.000000	
mean	1.126662e+07	4.782069	3.155146	1.250624	94.326011	38.445958	-92.397525	20.900568	
std	6.081735e+06	0.717394	2.153589	0.597852	16.387421	3.080167	21.705322	37.828641	
min	3.440000e+02	0.000000	1.000000	0.000000	0.000000	33.338905	-122.511500	0.000000	
25%	6.261964e+06	4.317488	2.000000	1.000000	100.000000	34.127908	-118.342374	1.000000	
50%	1.225415e+07	4.709530	2.000000	1.000000	100.000000	40.662138	-76.996965	6.000000	
75%	1.640226e+07	5.220356	4.000000	1.000000	100.000000	40.746096	-73.954660	23.000000	
max	2.123090e+07	7.600402	16.000000	8.000000	100.000000	42.390437	-70.985047	605.000000	

8 rows × 21 columns

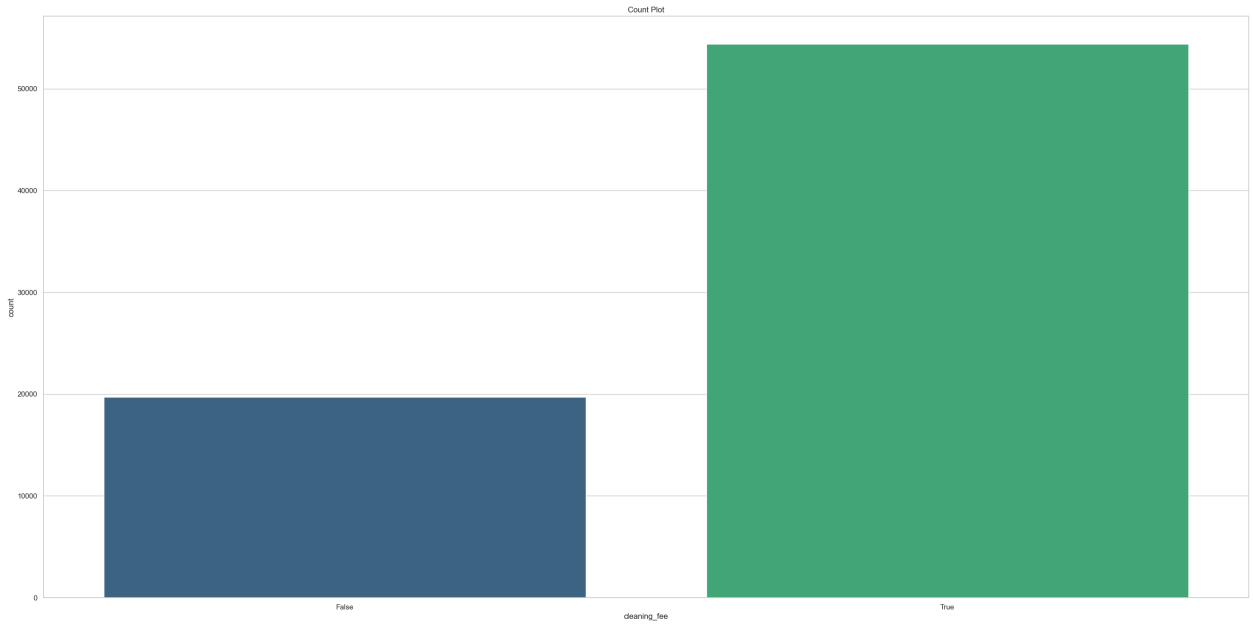
In [132…
```
create_seaborn_countplot(df, 'accommodates')
```



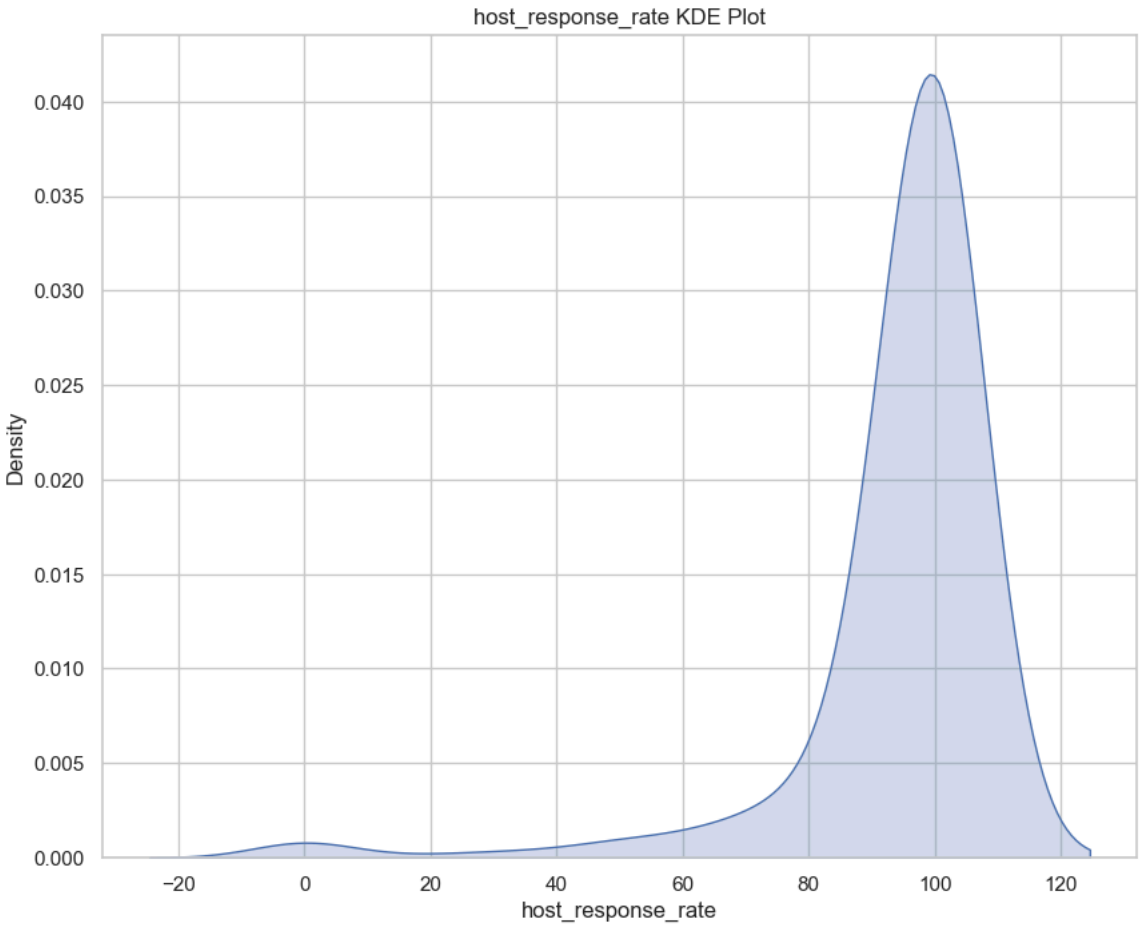
In [133…
```
create_seaborn_countplot(df, 'bathrooms')
```



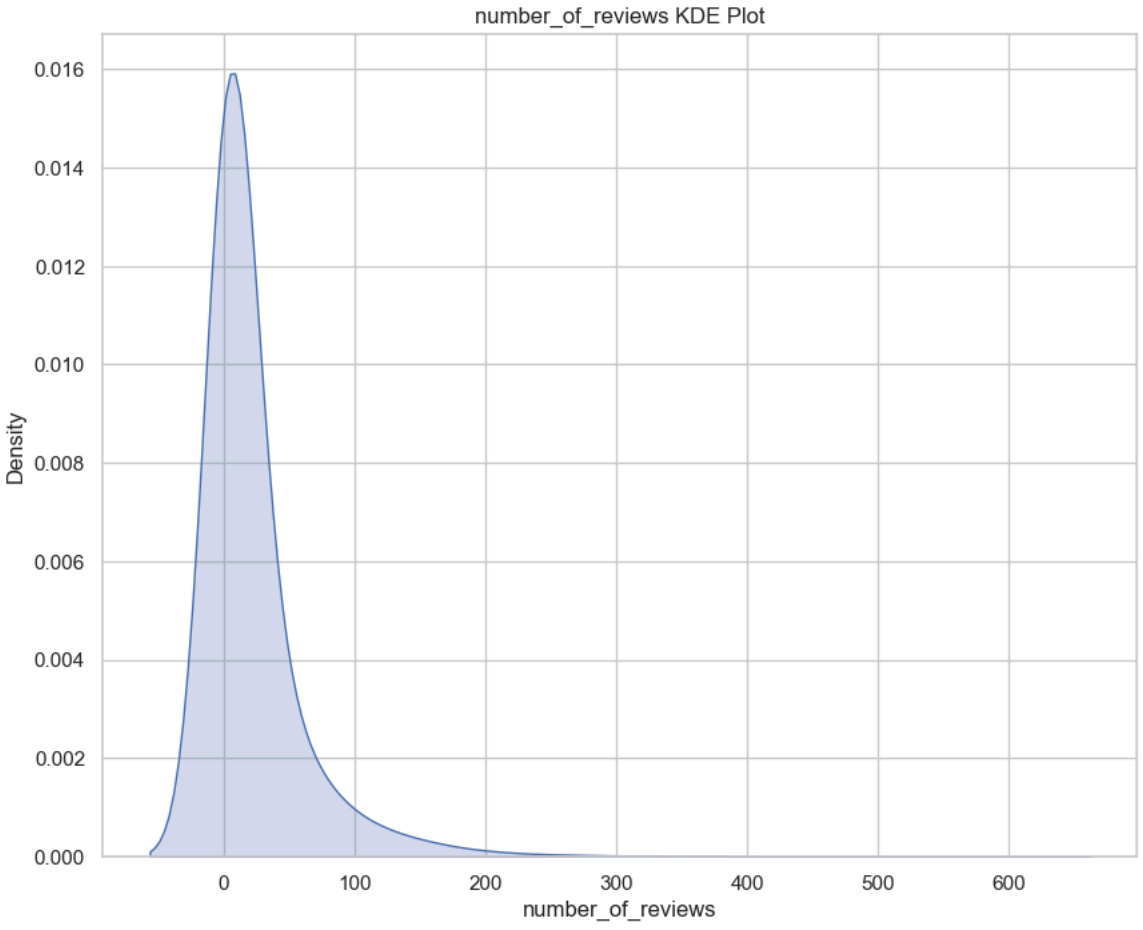
In [134…
```
create_seaborn_countplot(df, 'cleaning_fee')
```



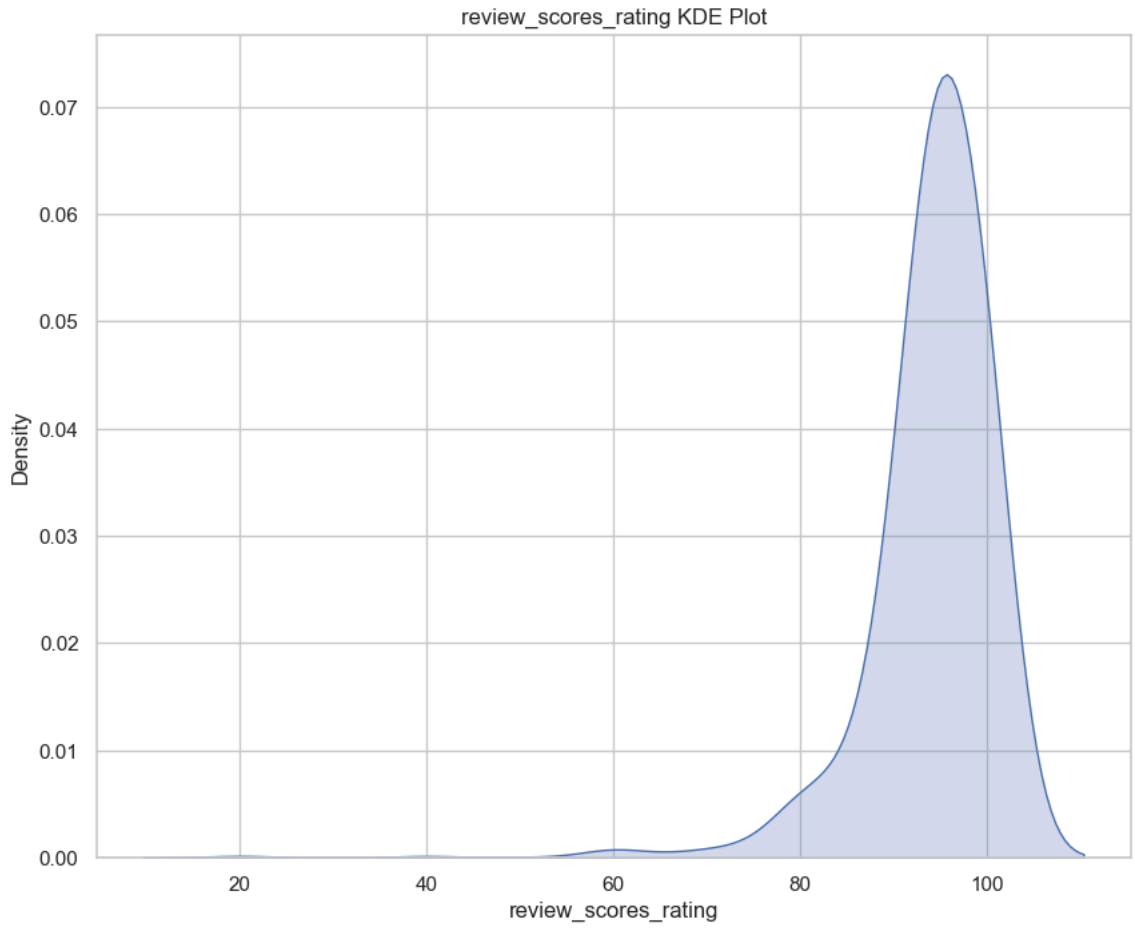
In [135…
```
plot_kde(df, 'host_response_rate')
```



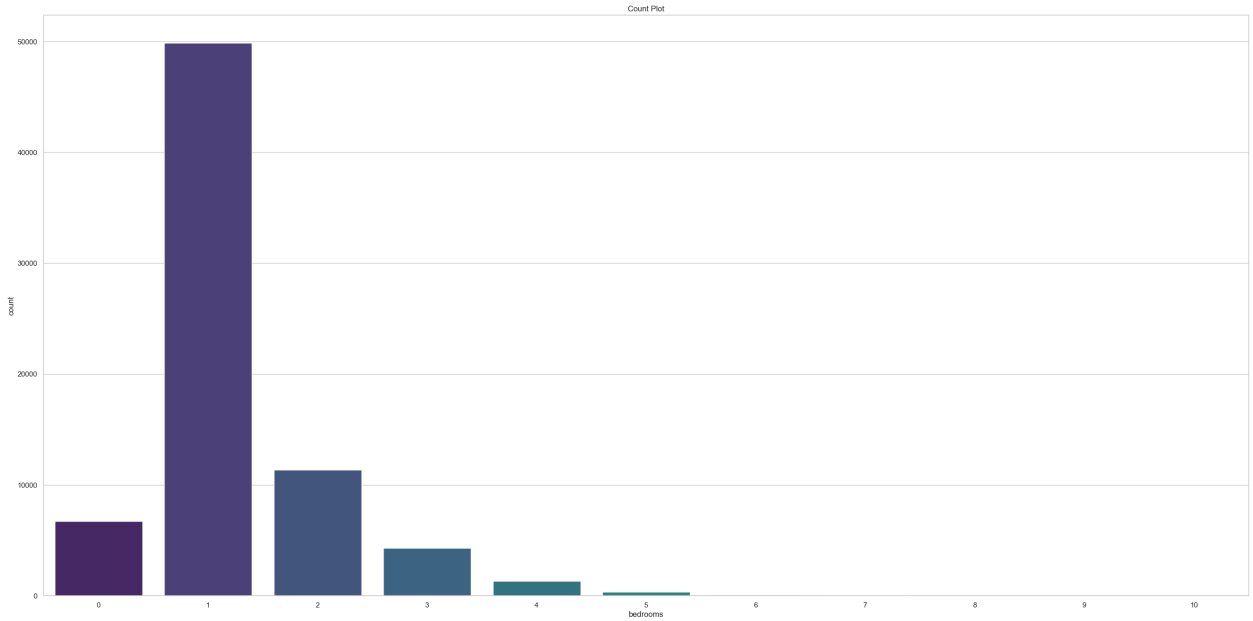
In [136…
```
plot_kde(df, 'number_of_reviews')
```



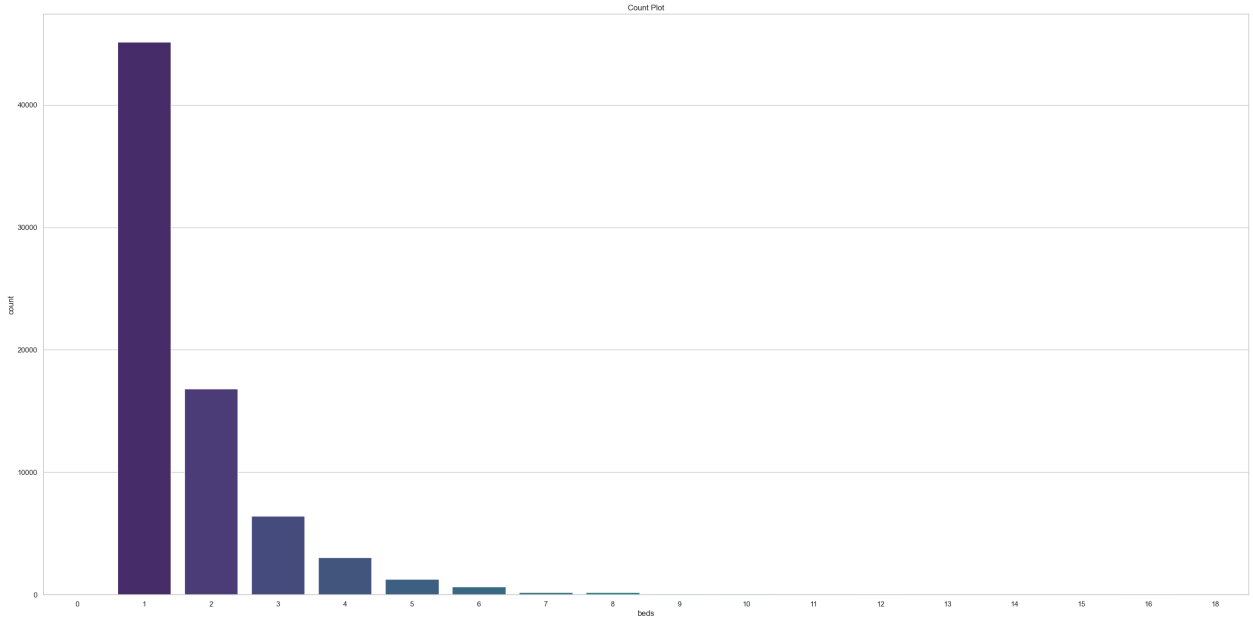
In [137…
```
plot_kde(df, 'review_scores_rating')
```



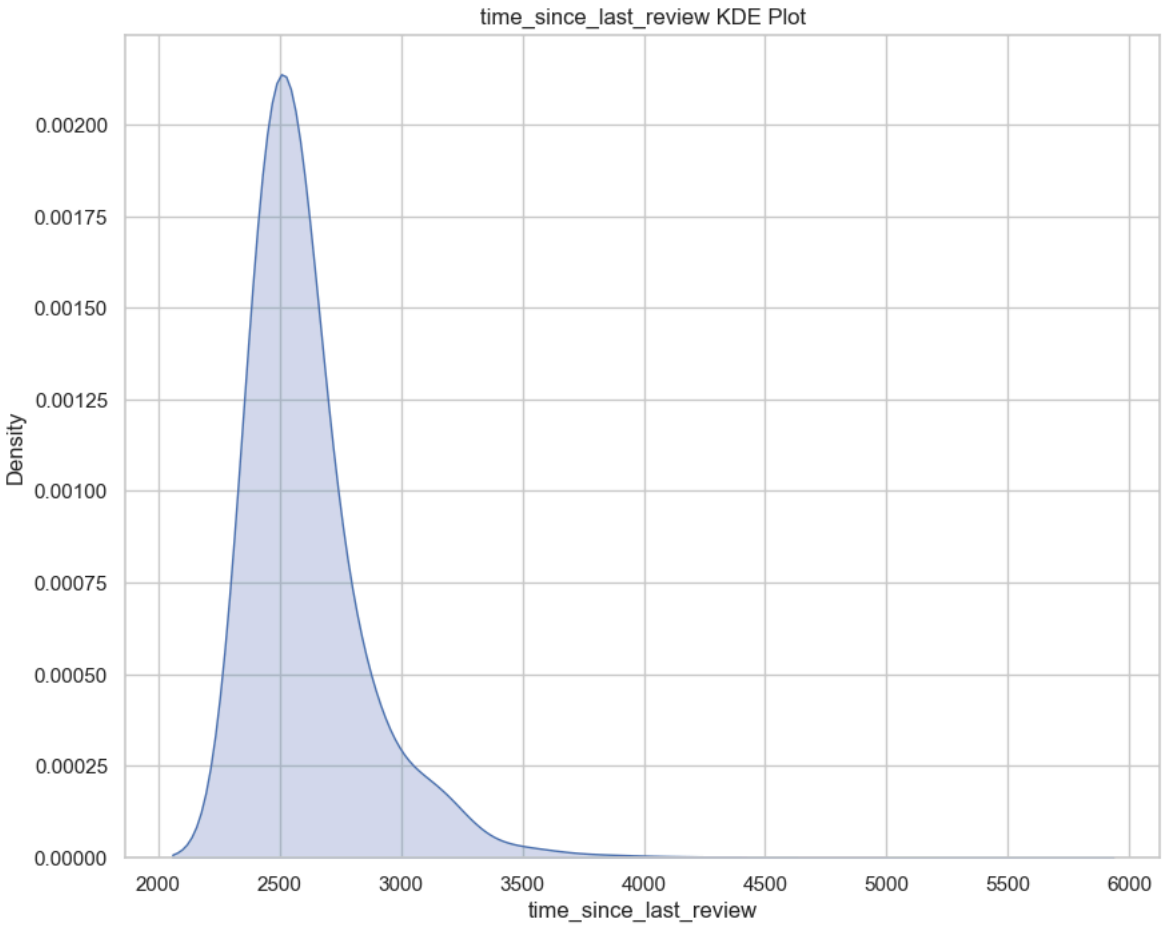
In [138…
```
create_seaborn_countplot(df, 'bedrooms')
```



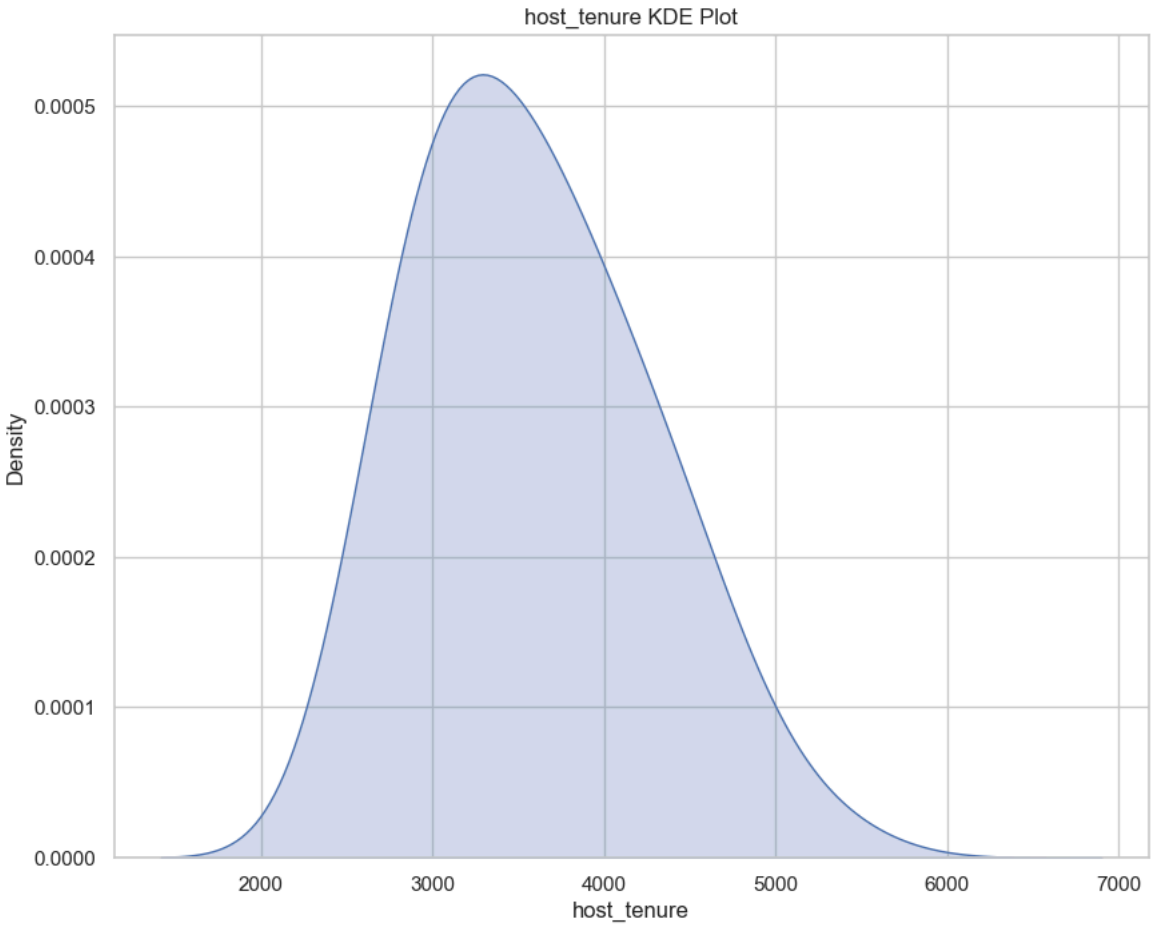
In [139…
```
create_seaborn_countplot(df, 'beds')
```



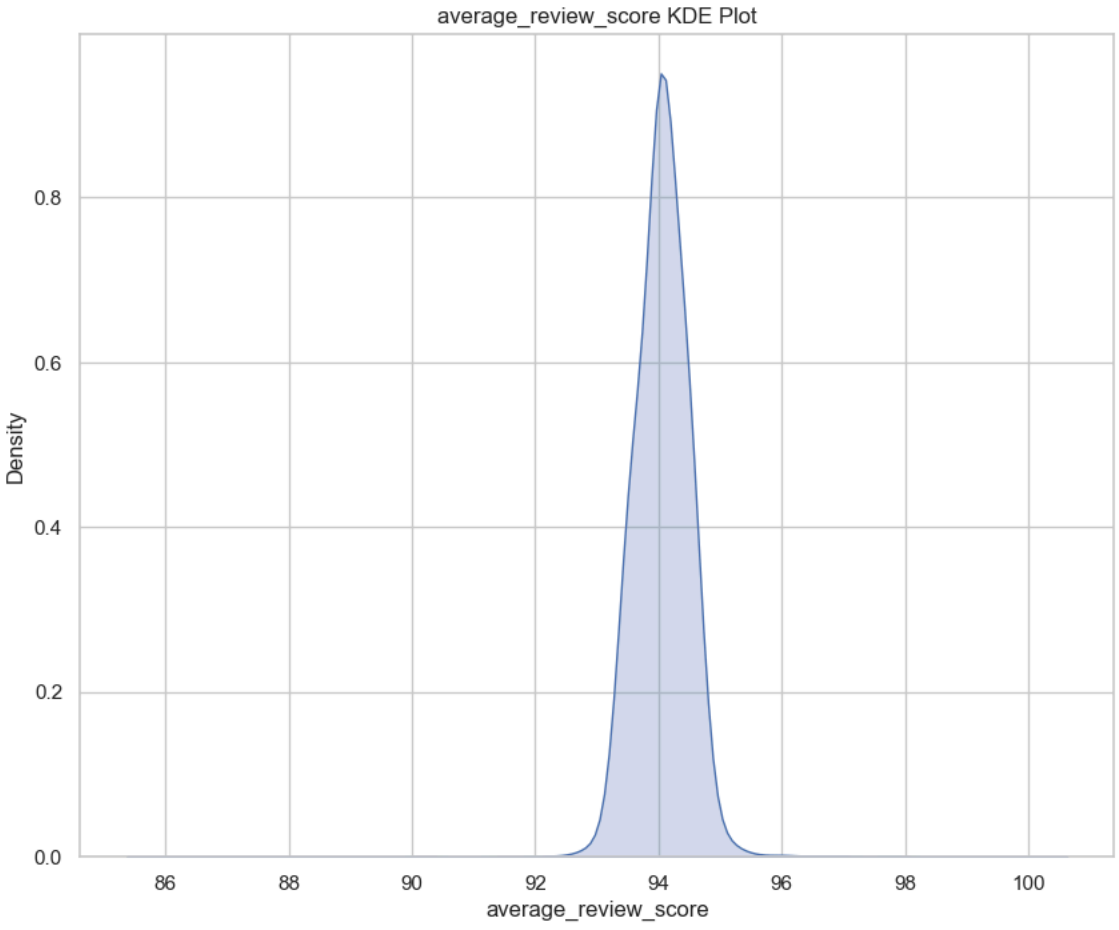
In [140…
```
plot_kde(df, 'time_since_last_review')
```



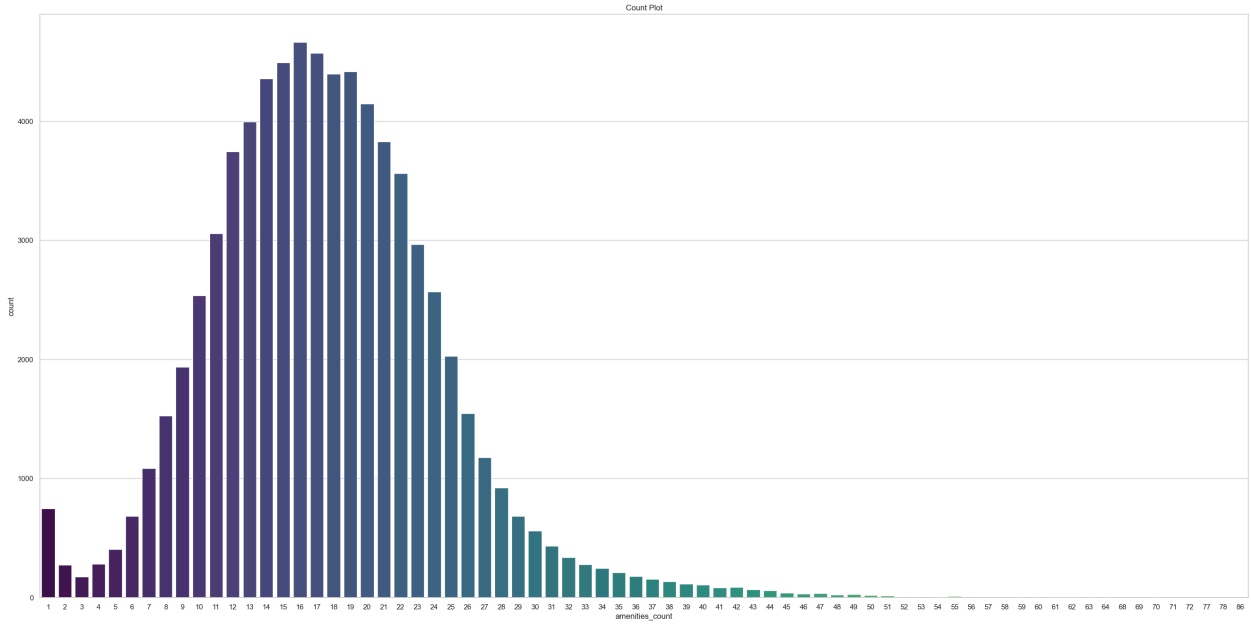
In [141…
```
plot_kde(df, 'host_tenure')
```



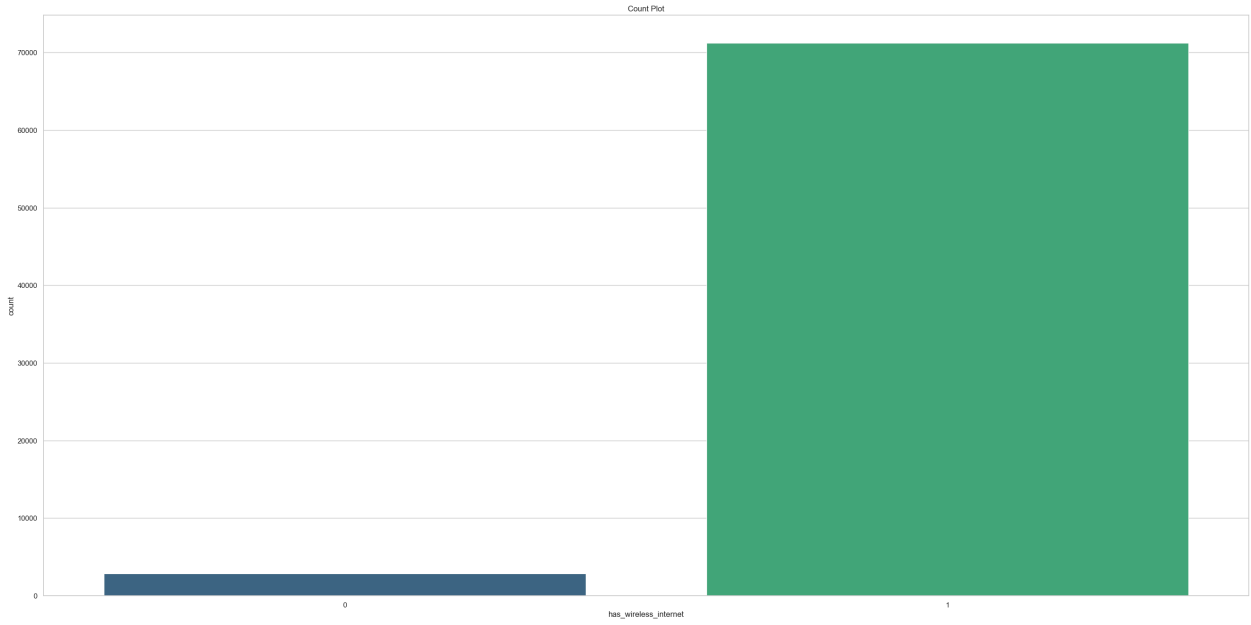
In [142…
```
plot_kde(df, 'average_review_score')
```



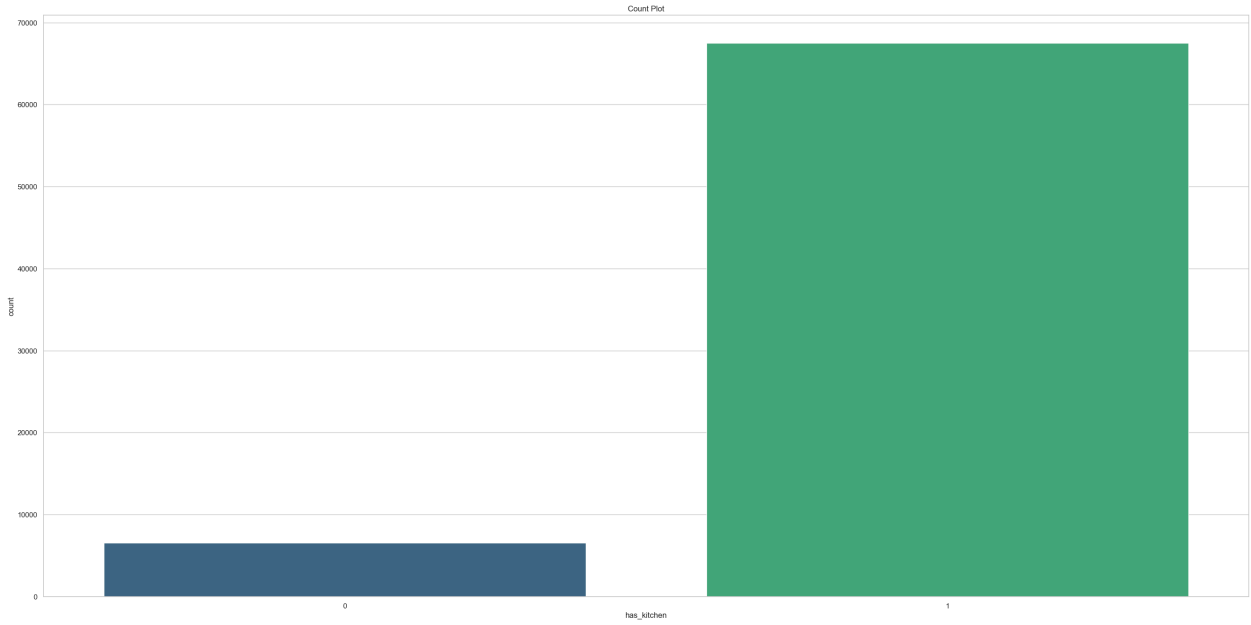
In [143…
```
create_seaborn_countplot(df,'amenities_count' )
```



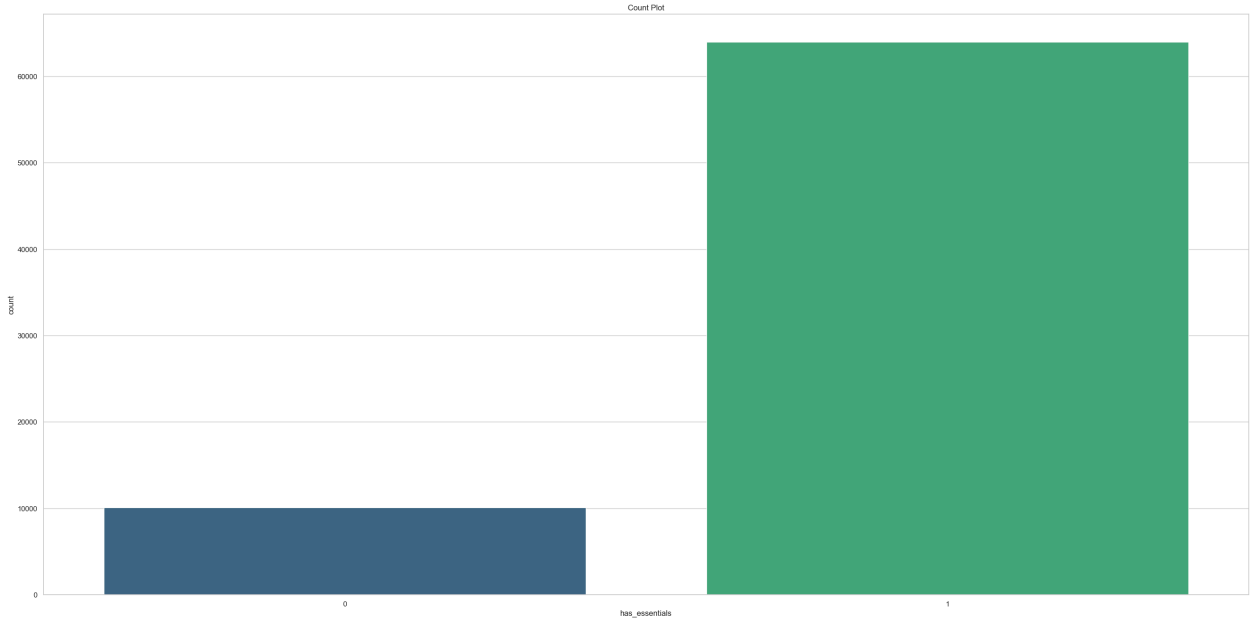
In [144…
```
create_seaborn_countplot(df, 'has_wireless_internet')
```



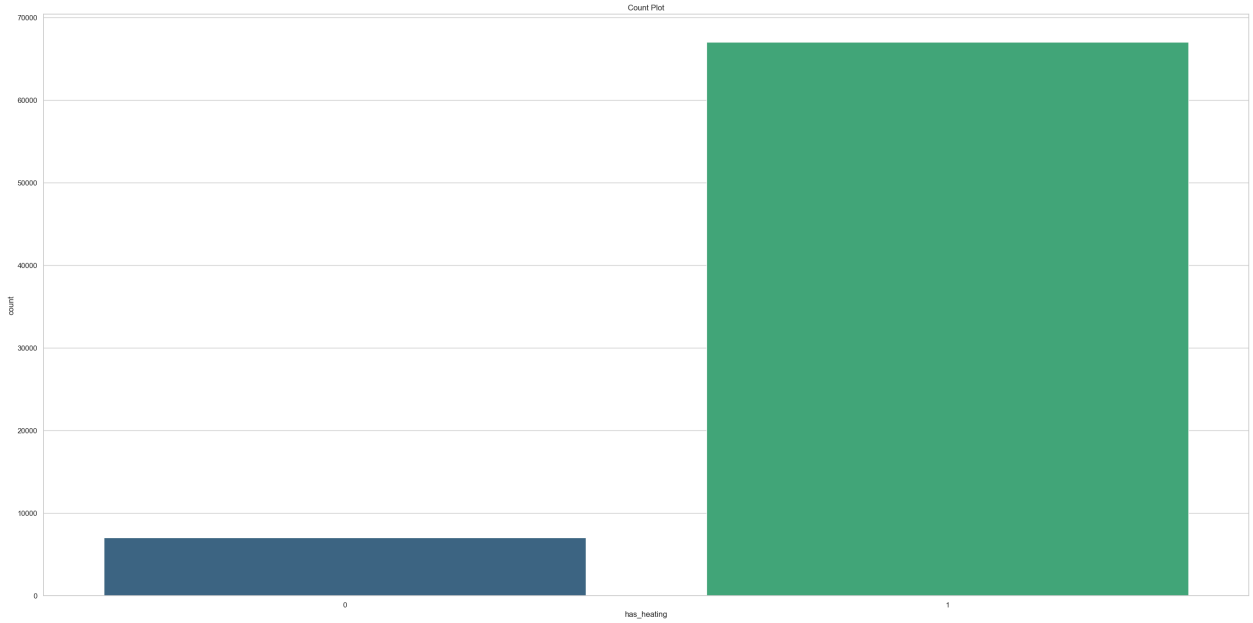
In [145…
```
create_seaborn_countplot(df, 'has_kitchen')
```



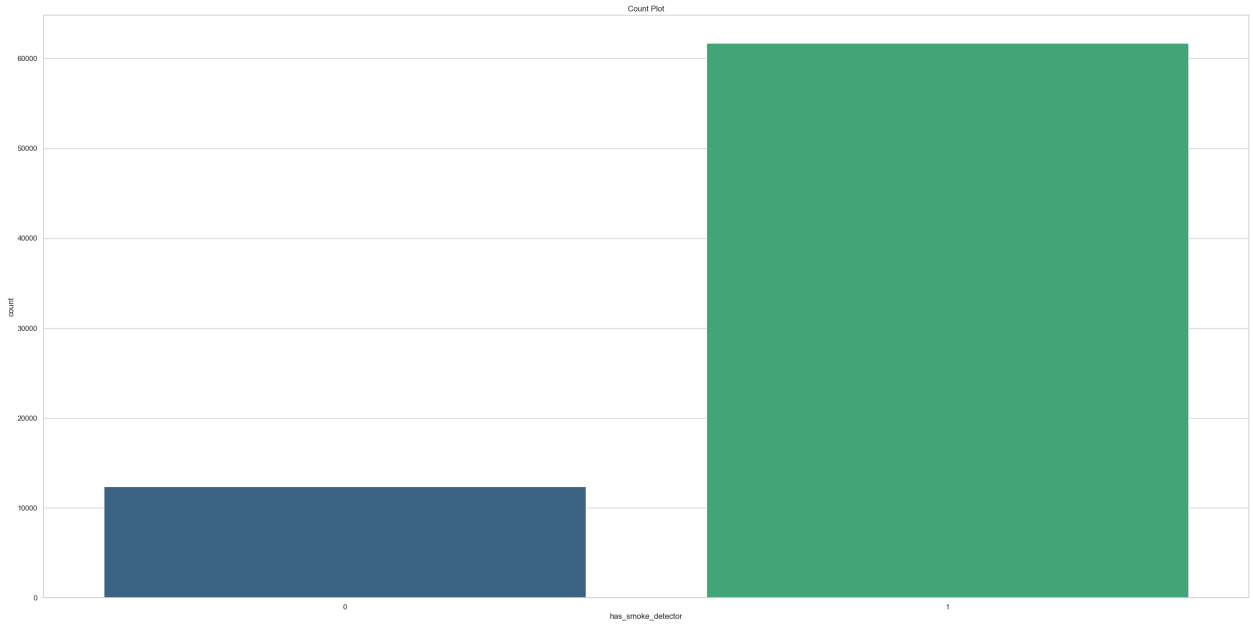
In [146…
```
create_seaborn_countplot(df, 'has_essentials')
```



In [147…
```
create_seaborn_countplot(df, 'has_heating')
```



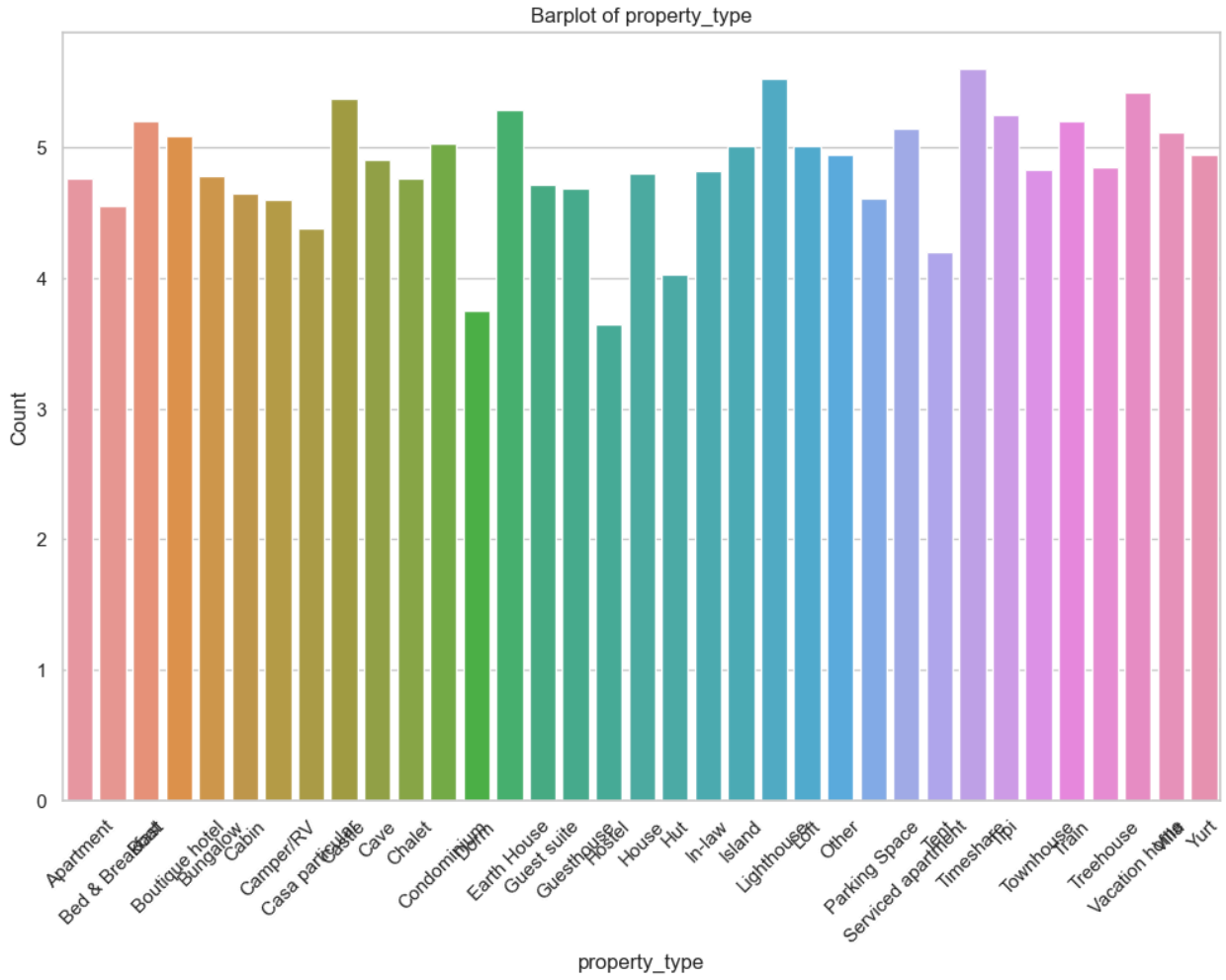
In [148…
```
create_seaborn_countplot(df, 'has_smoke_detector')
```



Bivarint Analysis

In [149…
```
# Calculating the mean log price for each property type
mean_log_price_wrt_pt = df.groupby(df['property_type'])['log_price'].mean().to_frame().reset_index() # return the dataframe of pr
```

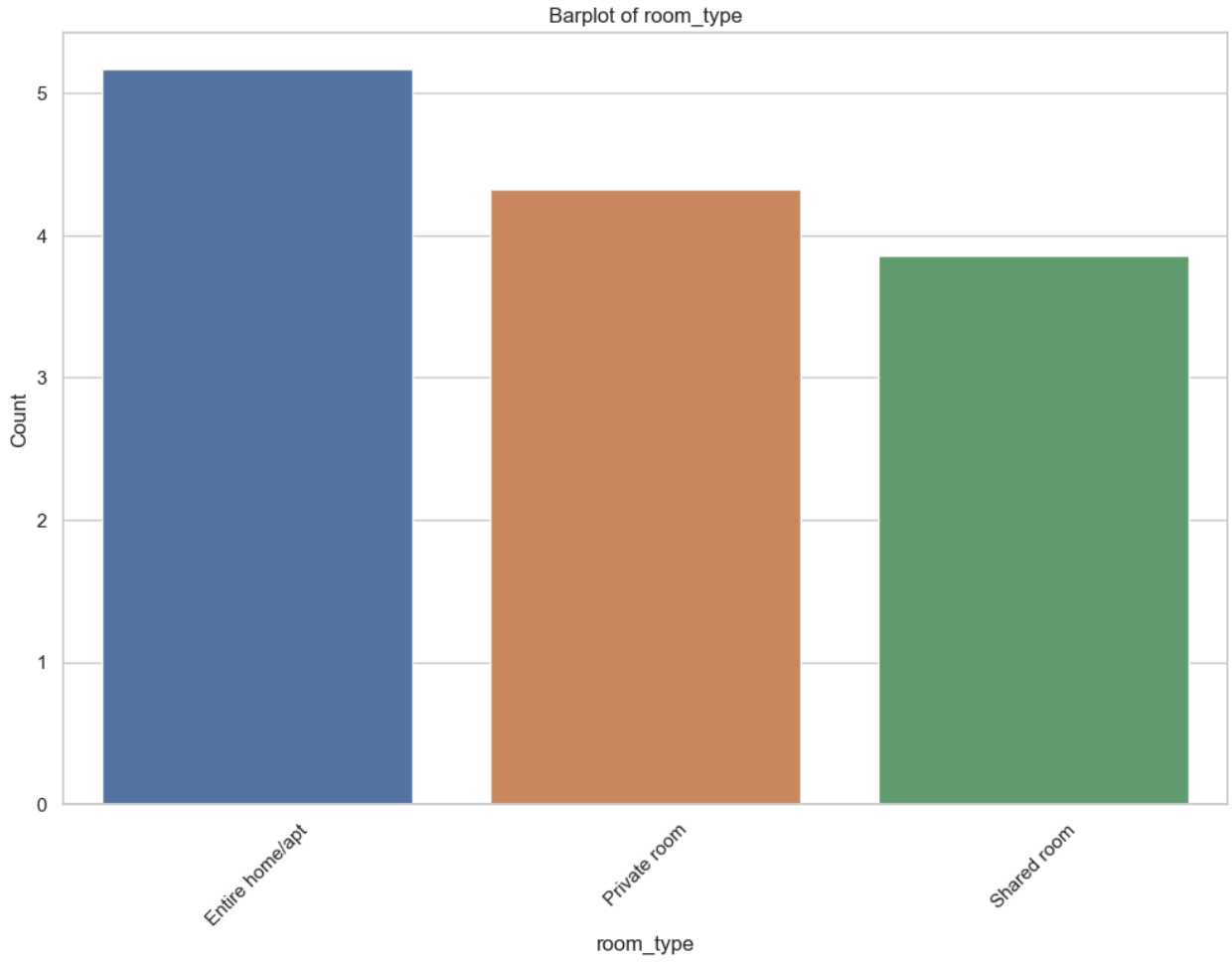
In [150…
```
plot_barplot(mean_log_price_wrt_pt, 'property_type', 'log_price')
```



Out[150]: <AxesSubplot: title={'center': 'Barplot of property_type'}, xlabel='property_type', ylabel='Count'>

In [151…
```
# Calculating the mean log price for each property type
mean_log_price_wrt_rt = df.groupby(df['room_type'])['log_price'].mean().to_frame().reset_index() # return the dataframe of proper
```

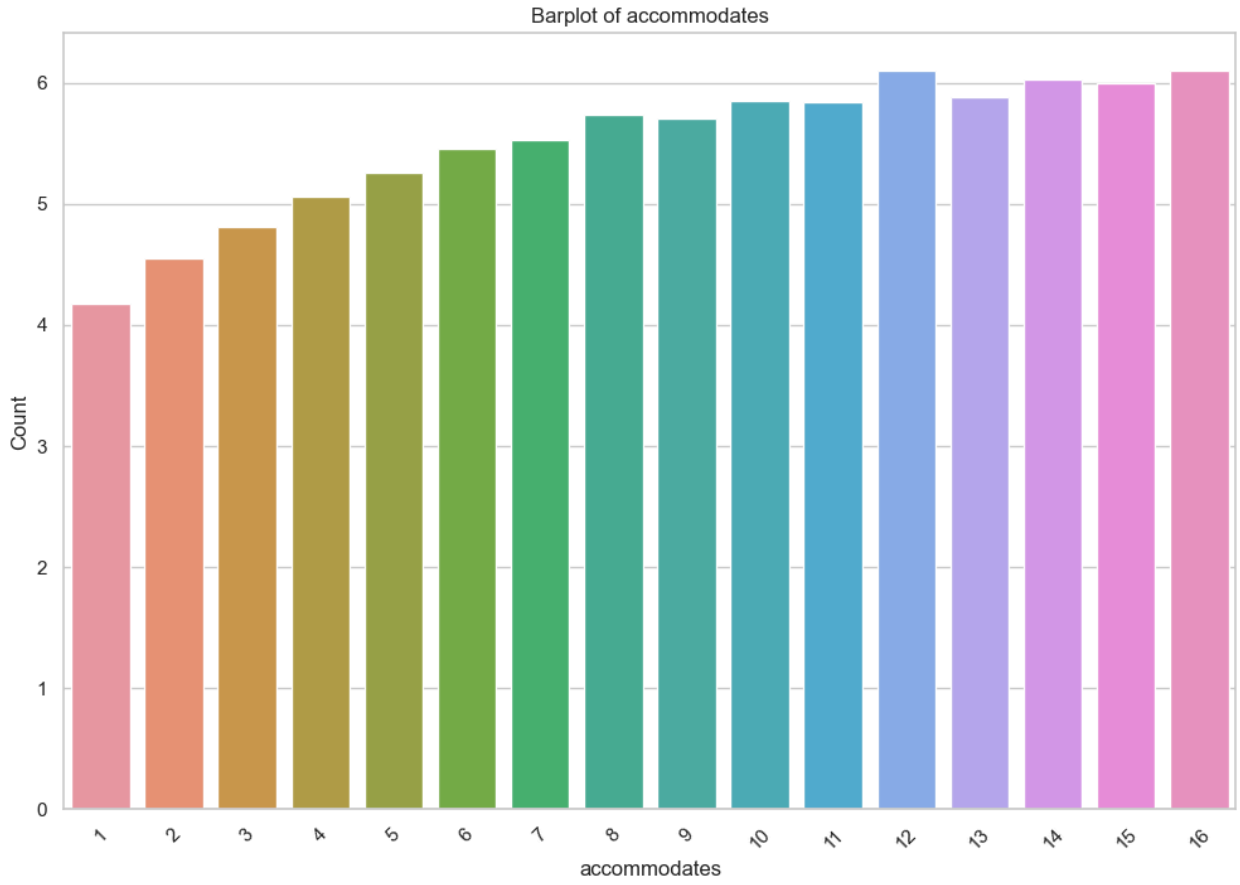
In [152…
```
plot_barplot(mean_log_price_wrt_rt, 'room_type', 'log_price')
```



Out[152]: <AxesSubplot: title={'center': 'Barplot of room_type'}, xlabel='room_type', ylabel='Count'>

In [153…
```
mean_log_price_wrt_acc = df.groupby(df['accommodates'])['log_price'].mean().to_frame().reset_index() # return the dataframe of ac
```

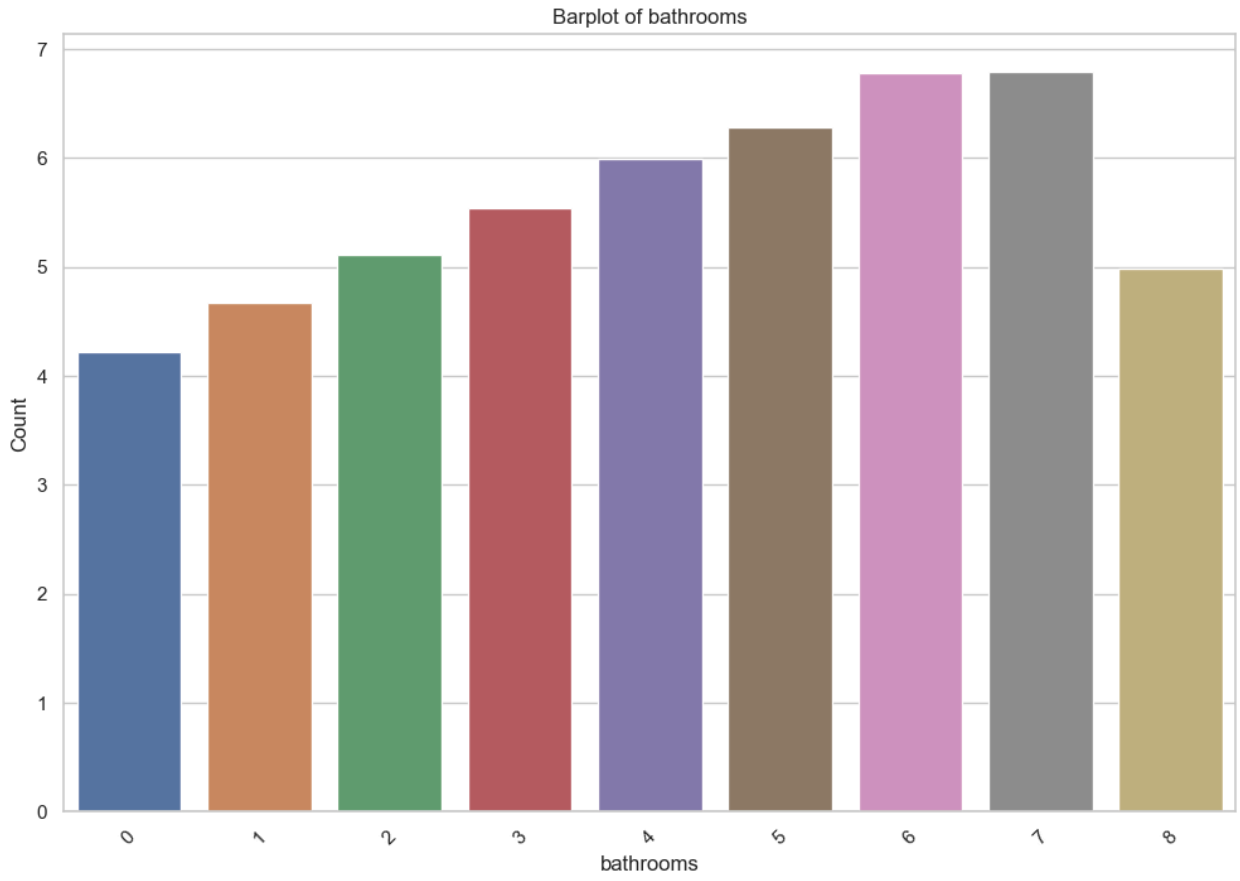
In [154…
```
plot_barplot(mean_log_price_wrt_acc, 'accommodates', 'log_price')
```



Out[154]: <AxesSubplot: title={'center': 'Barplot of accommodates'}, xlabel='accommodates', ylabel='Count'>

In [155…
```
mean_log_price_wrt_br = df.groupby(df['bathrooms'])['log_price'].mean().to_frame().reset_index() # return the dataframe of bathro
```

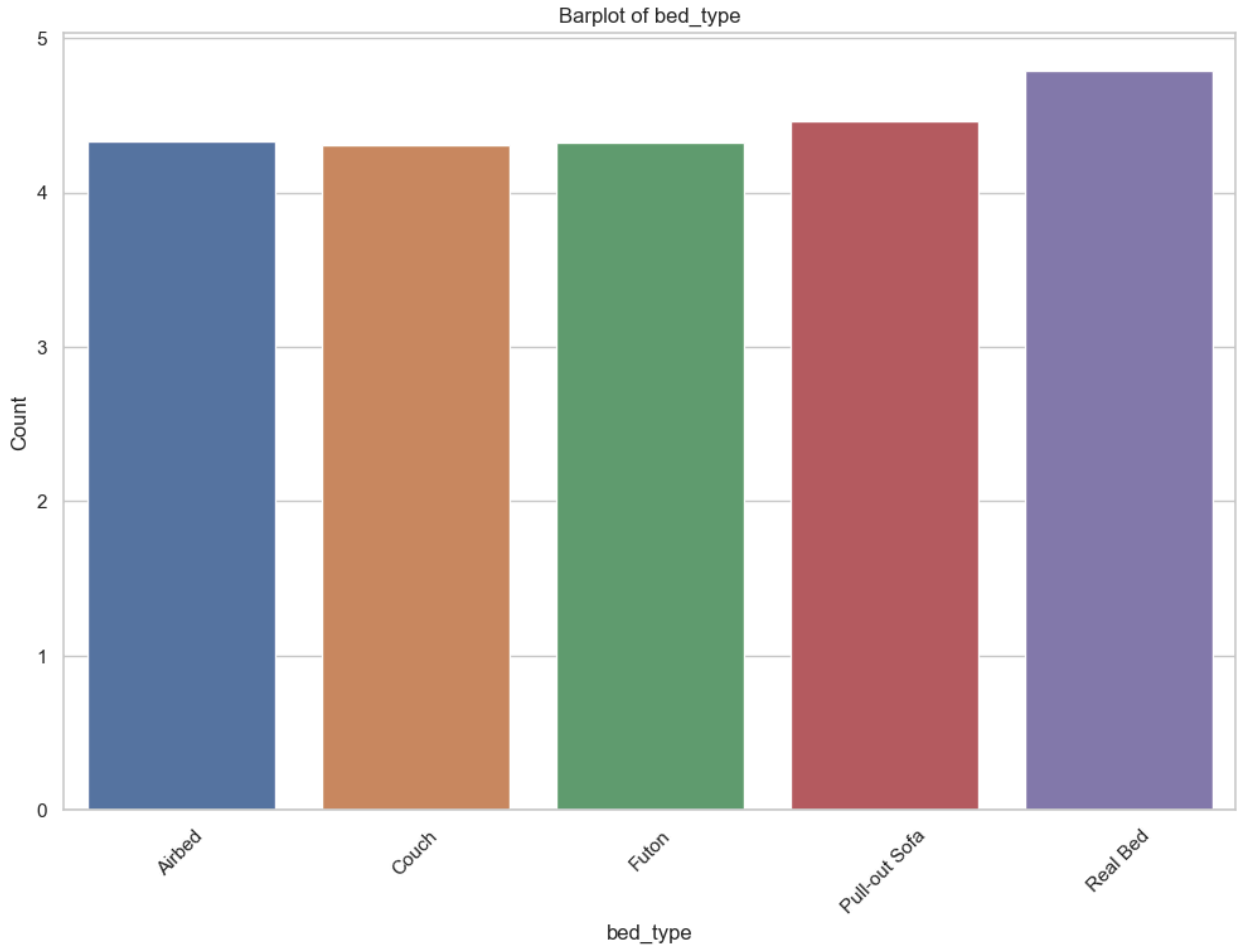
In [156…
```
plot_barplot(mean_log_price_wrt_br, 'bathrooms', 'log_price')
```



Out[156]: <AxesSubplot: title={'center': 'Barplot of bathrooms'}, xlabel='bathrooms', ylabel='Count'>

In [157…
```
mean_log_price_wrt_bt = df.groupby(df['bed_type'])['log_price'].mean().to_frame().reset_index() # return the dataframe of bathroo
```

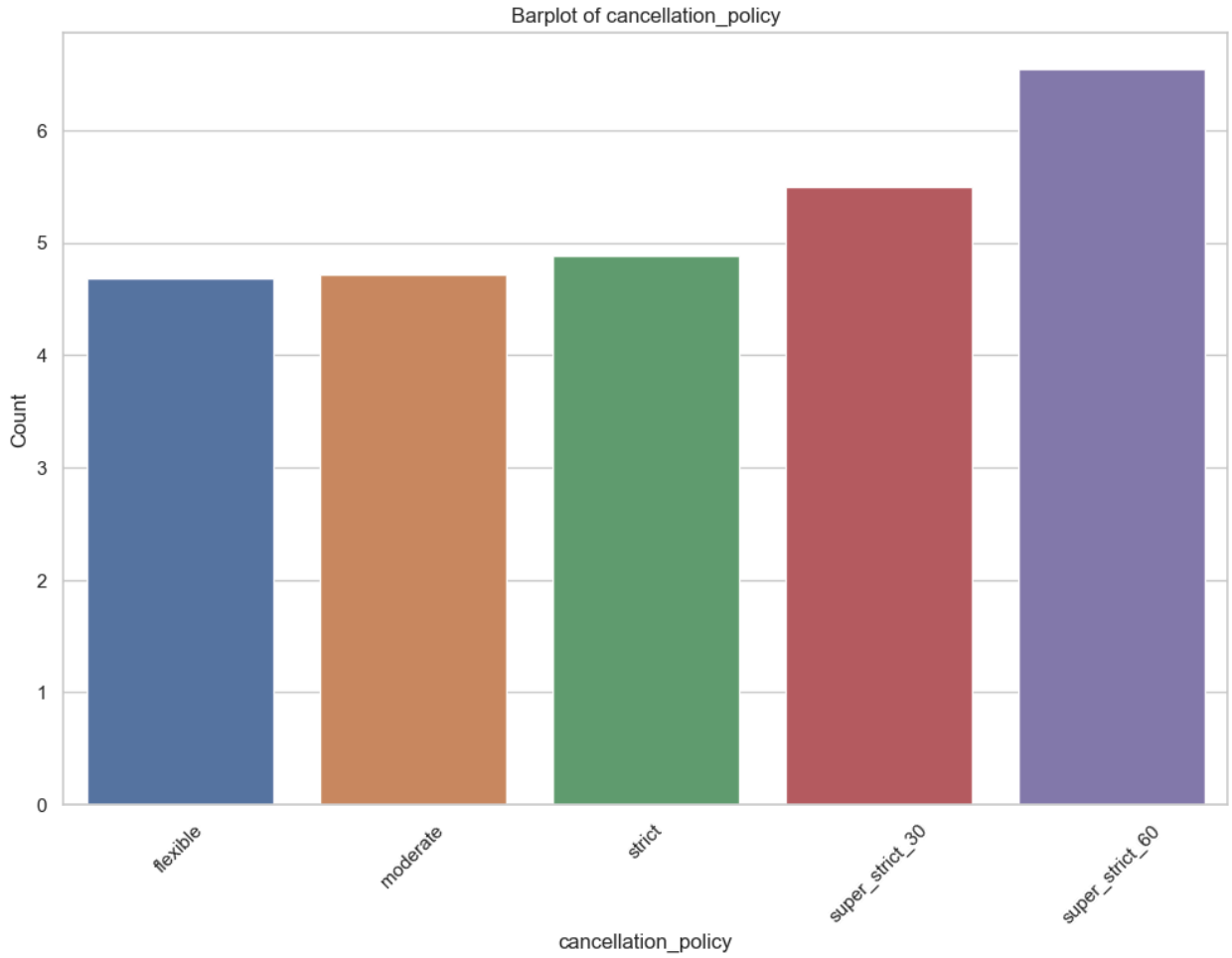
In [158…
```
plot_barplot(mean_log_price_wrt_bt, 'bed_type', 'log_price')
```



Out[158]: <AxesSubplot: title={'center': 'Barplot of bed_type'}, xlabel='bed_type', ylabel='Count'>

In [159…
```
mean_log_price_wrt_cp = df.groupby(df['cancellation_policy'])['log_price'].mean().to_frame().reset_index() # return the dataframe
```

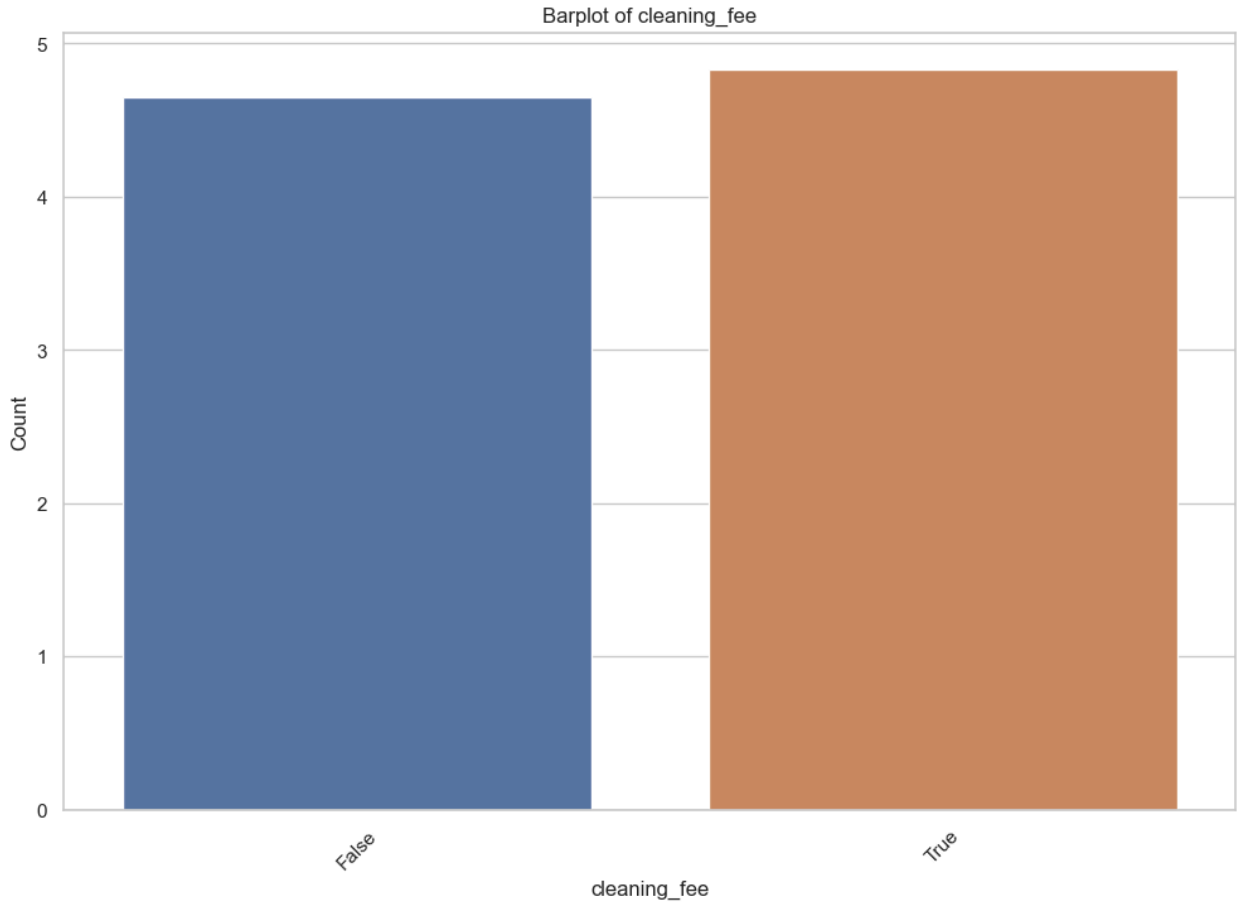
In [160…
```
plot_barplot(mean_log_price_wrt_cp, 'cancellation_policy', 'log_price')
```



Out[160]: <AxesSubplot: title={'center': 'Barplot of cancellation_policy'}, xlabel='cancellation_policy', ylabel='Count'>

In [161…
```
mean_log_price_wrt_cf = df.groupby(df['cleaning_fee'])['log_price'].mean().to_frame().reset_index() # return the dataframe of bat
```

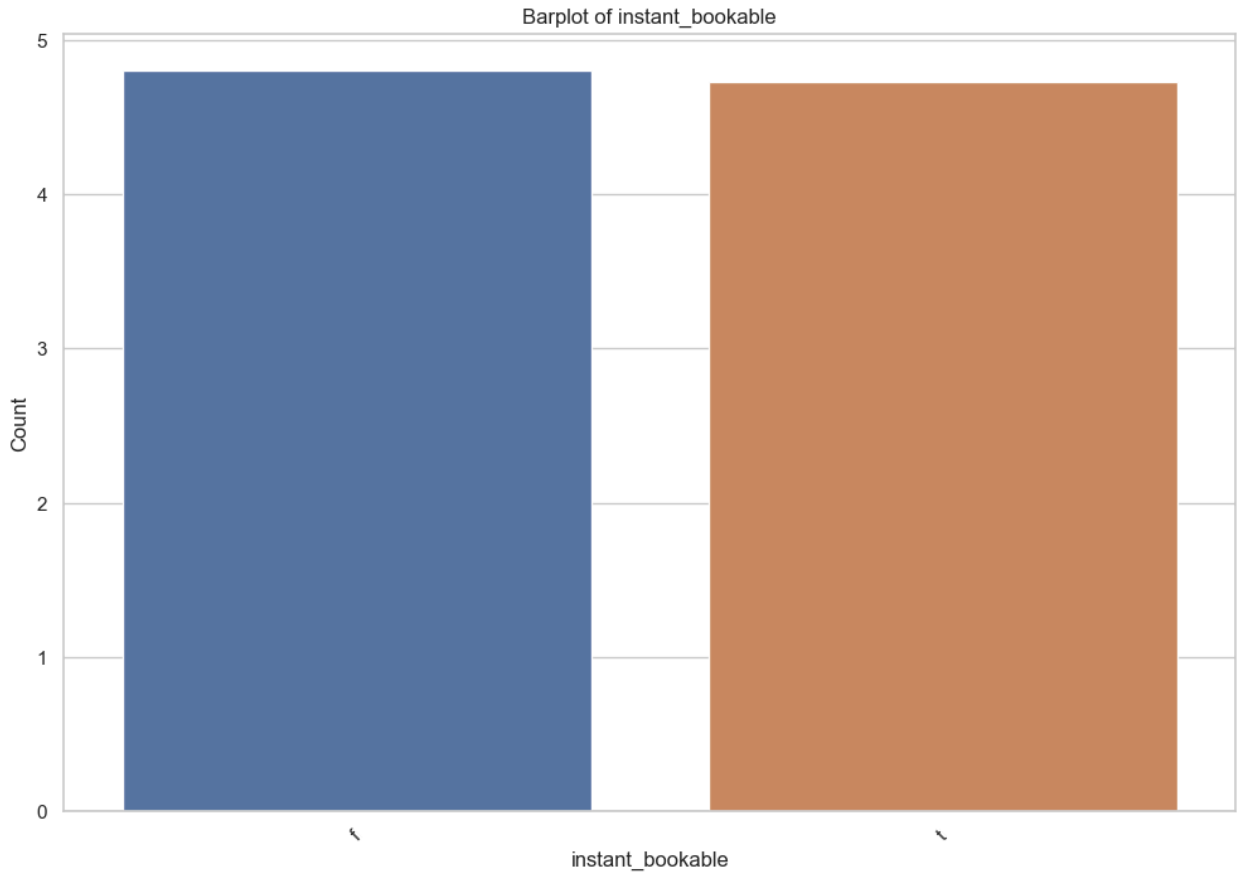
In [162…
```
plot_barplot(mean_log_price_wrt_cf, 'cleaning_fee', 'log_price')
```



Out[162]: <AxesSubplot: title={'center': 'Barplot of cleaning_fee'}, xlabel='cleaning_fee', ylabel='Count'>

In [163…
```
mean_log_price_wrt_ib = df.groupby(df['instant_bookable'])['log_price'].mean().to_frame().reset_index() # return the dataframe of
```

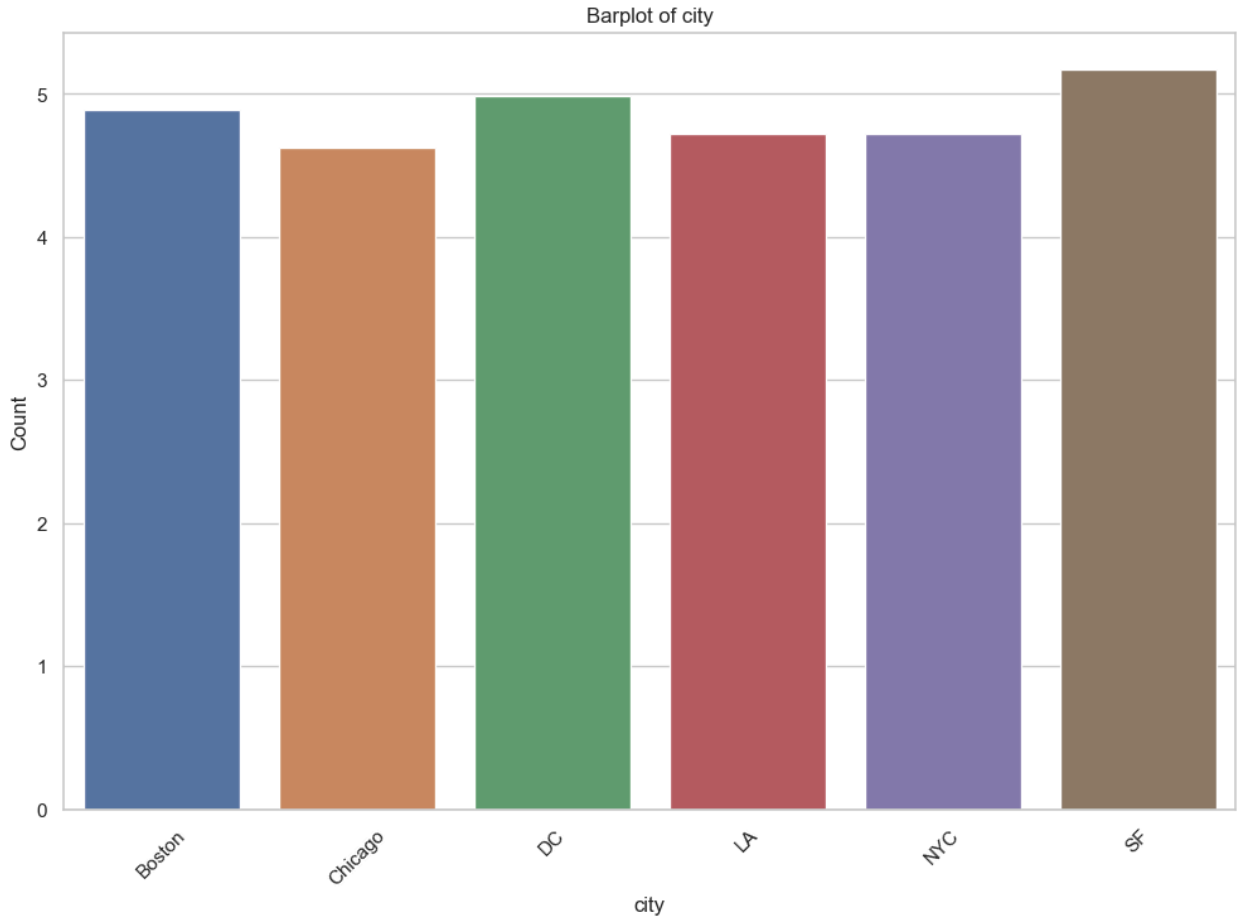
In [164…
```
plot_barplot(mean_log_price_wrt_ib, 'instant_bookable', 'log_price')
```



Out[164]: <AxesSubplot: title={'center': 'Barplot of instant_bookable'}, xlabel='instant_bookable', ylabel='Count'>

In [165…
```
mean_log_price_wrt_cty = df.groupby(df['city'])['log_price'].mean().to_frame().reset_index() # return the dataframe of bathrooms
```

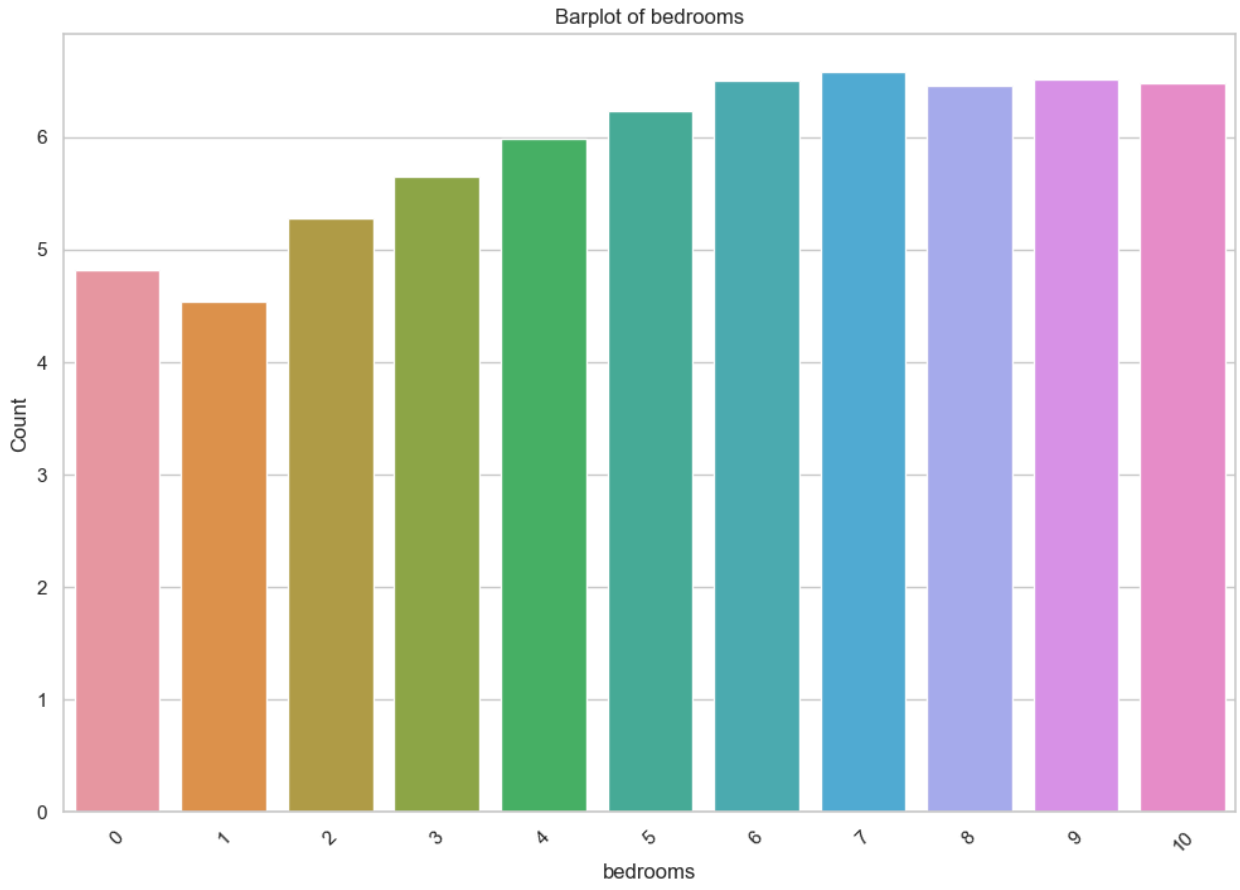
In [166…
```
plot_barplot(mean_log_price_wrt_cty, 'city', 'log_price')
```



Out[166]: <AxesSubplot: title={'center': 'Barplot of city'}, xlabel='city', ylabel='Count'>

In [167…
```
mean_log_price_wrt_bedr = df.groupby(df['bedrooms'])['log_price'].mean().to_frame().reset_index() # return the dataframe of bathr
```

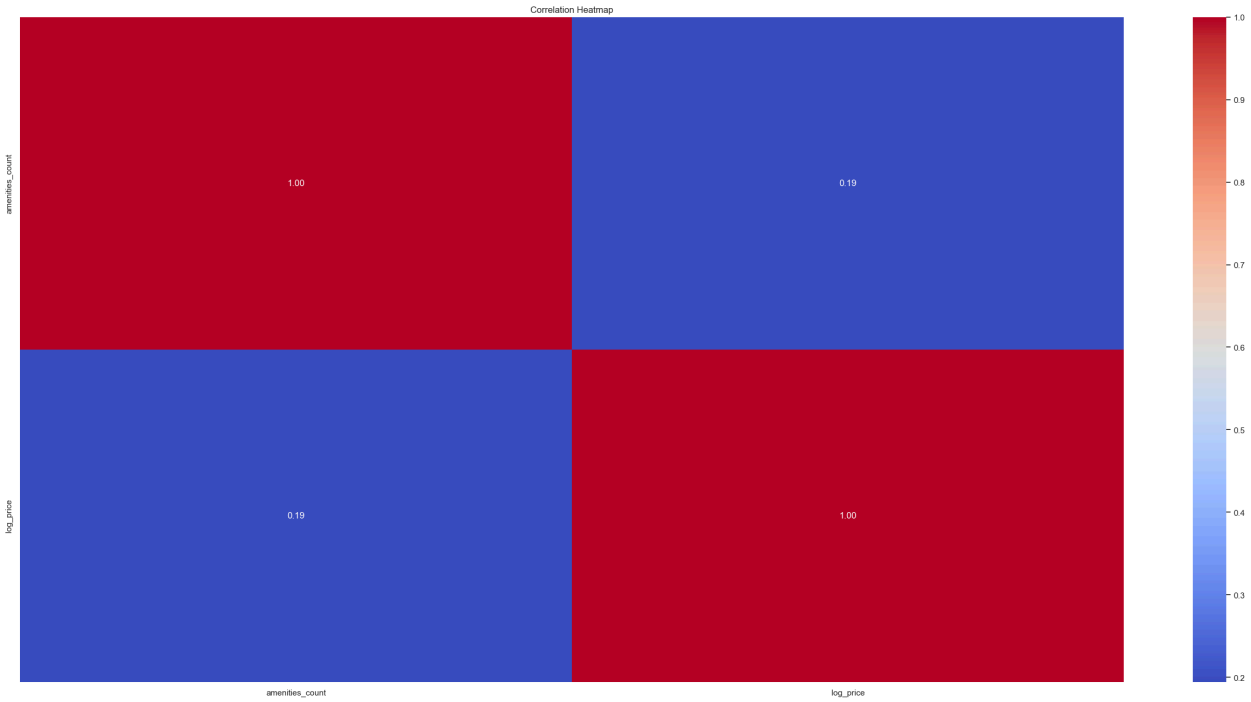
In [168…
```
plot_barplot(mean_log_price_wrt_bedr, 'bedrooms', 'log_price')
```



Out[168]: <AxesSubplot: title={'center': 'Barplot of bedrooms'}, xlabel='bedrooms', ylabel='Count'>

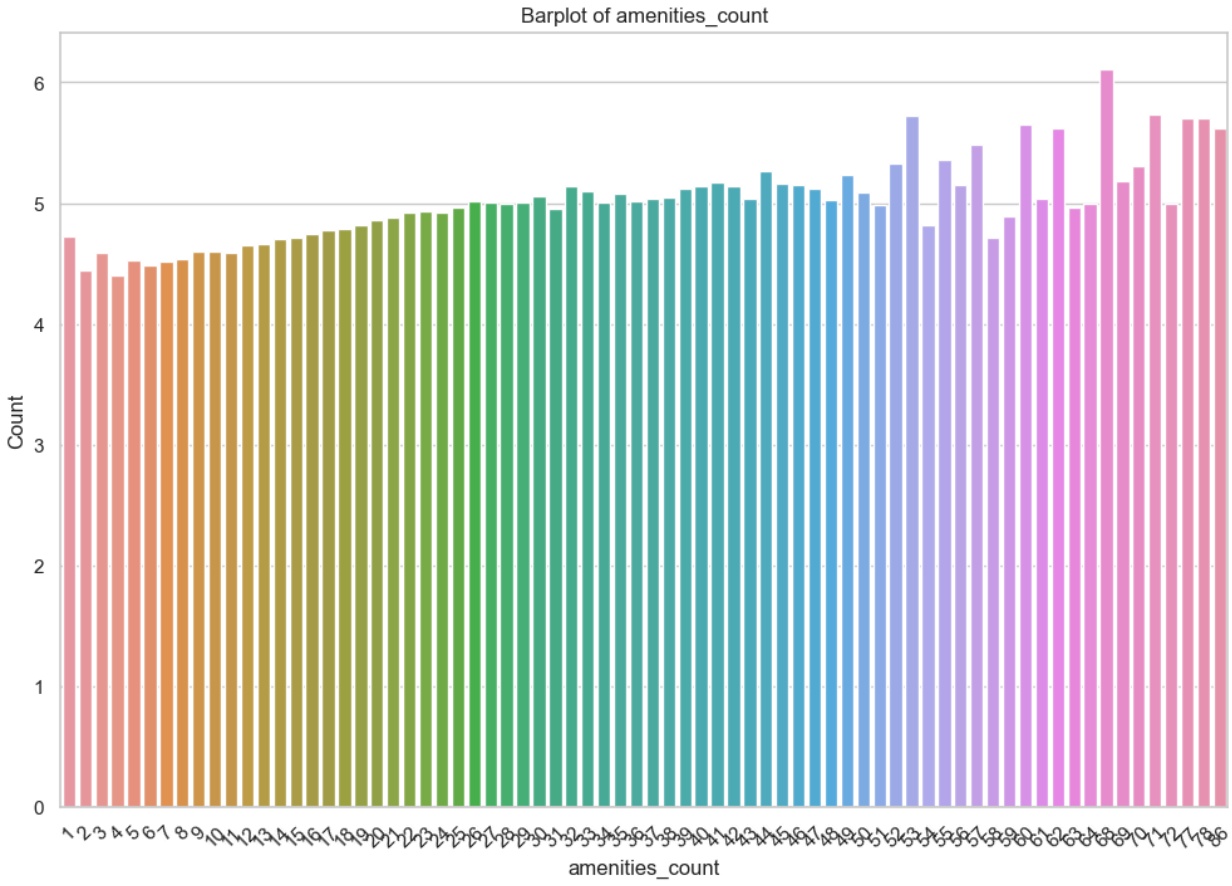
In [ ]:

In [169…
```
columns_to_include = ['amenities_count','log_price']
# Plot correlation heatmap
plot_correlation_heatmap(df, columns_to_include)
```



In [170…
```
mean_log_price_wrt_ac = df.groupby(df['amenities_count'])['log_price'].mean().to_frame().reset_index() # return the dataframe of
```

In [171…
```
plot_barplot(mean_log_price_wrt_ac, 'amenities_count', 'log_price')
```



Out[171]: <AxesSubplot: title={'center': 'Barplot of amenities_count'}, xlabel='amenities_count', ylabel='Count'>

In [172…
```
# Example usage:
aggregate_columns = ['property_type','room_type', 'accommodates','bathrooms', 'bedrooms','neighbourhood', 'cancellation_policy']
mode_data_city = calculate_mode(df, 'city', aggregate_columns)
```

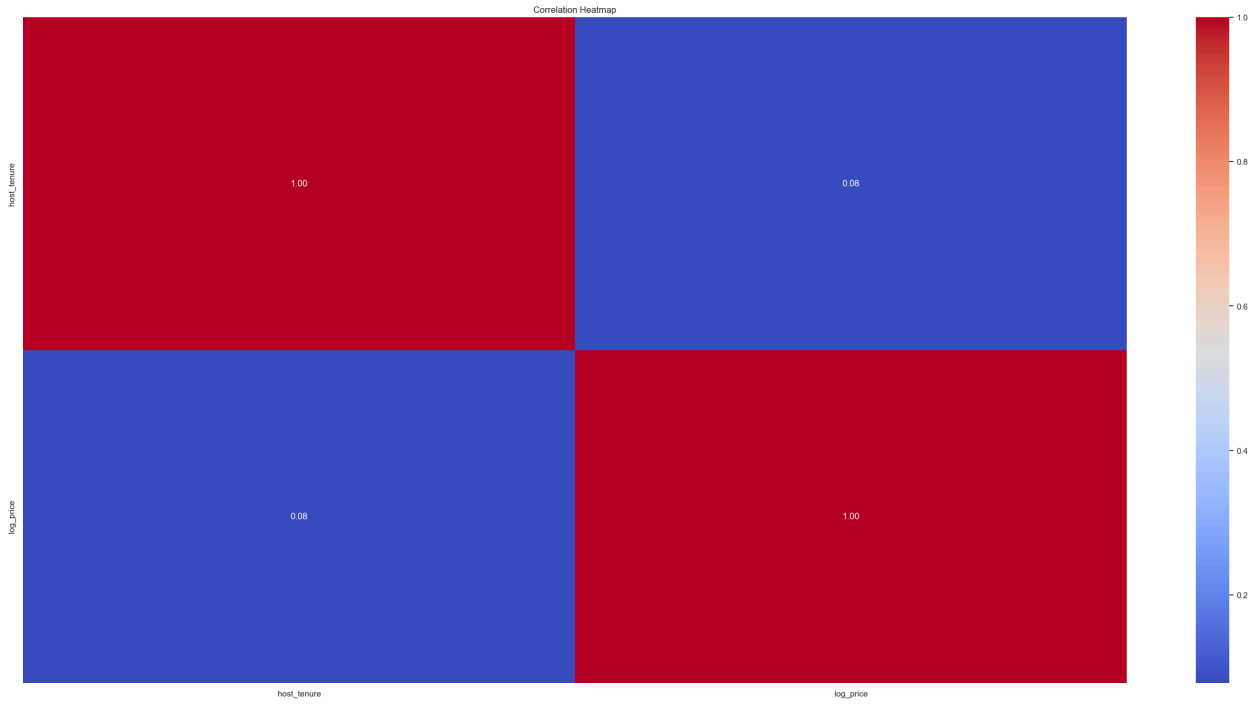
In [173…
```
mode_data_city
```

Out[173]:

city	mode_property_type	mode_room_type	mode_accommodates	mode_bathrooms	mode_bedrooms	mode_neighbourhood	mode_canc
Boston	Apartment	Entire home/apt	2	1	1	Allston-Brighton	
Chicago	Apartment	Entire home/apt	2	1	1	Lakeview	
DC	Apartment	Entire home/apt	2	1	1	Capitol Hill	
LA	Apartment	Entire home/apt	2	1	1	Mid-Wilshire	
NYC	Apartment	Entire home/apt	2	1	1	Williamsburg	
SF	Apartment	Entire home/apt	2	1	1	Mission District	

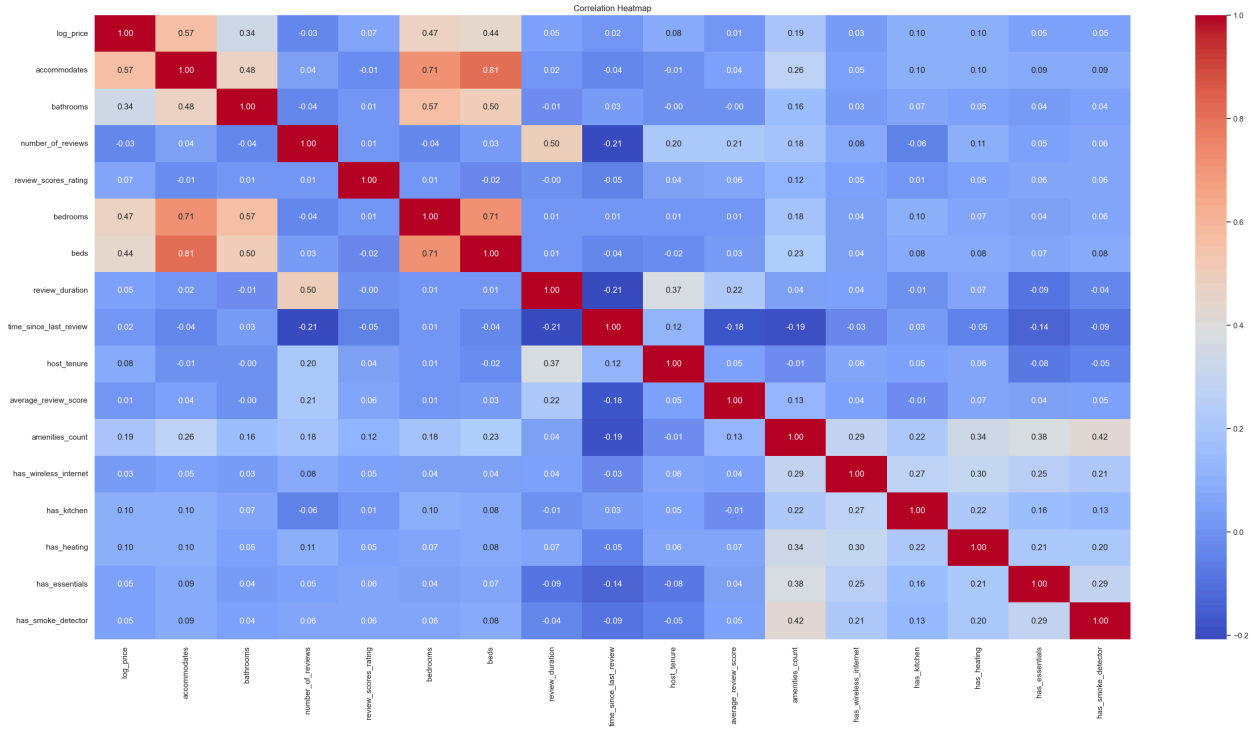
In [174…

```
columns_to_include = ['host_tenure','log_price']
# Plot correlation heatmap
plot_correlation_heatmap(df, columns_to_include)
```



In [175…

```
columns_to_include = ['log_price','accommodates', 'bathrooms','number_of_reviews',
       'review_scores_rating', 'bedrooms', 'beds',
       'review_duration', 'time_since_last_review', 'host_tenure',
       'average_review_score', 'amenities_count', 'has_wireless_internet',
       'has_kitchen', 'has_heating', 'has_essentials', 'has_smoke_detector']

corr_df = df[columns_to_include].corr()
# Plot correlation heatmap
plot_correlation_heatmap(df, columns_to_include)
```



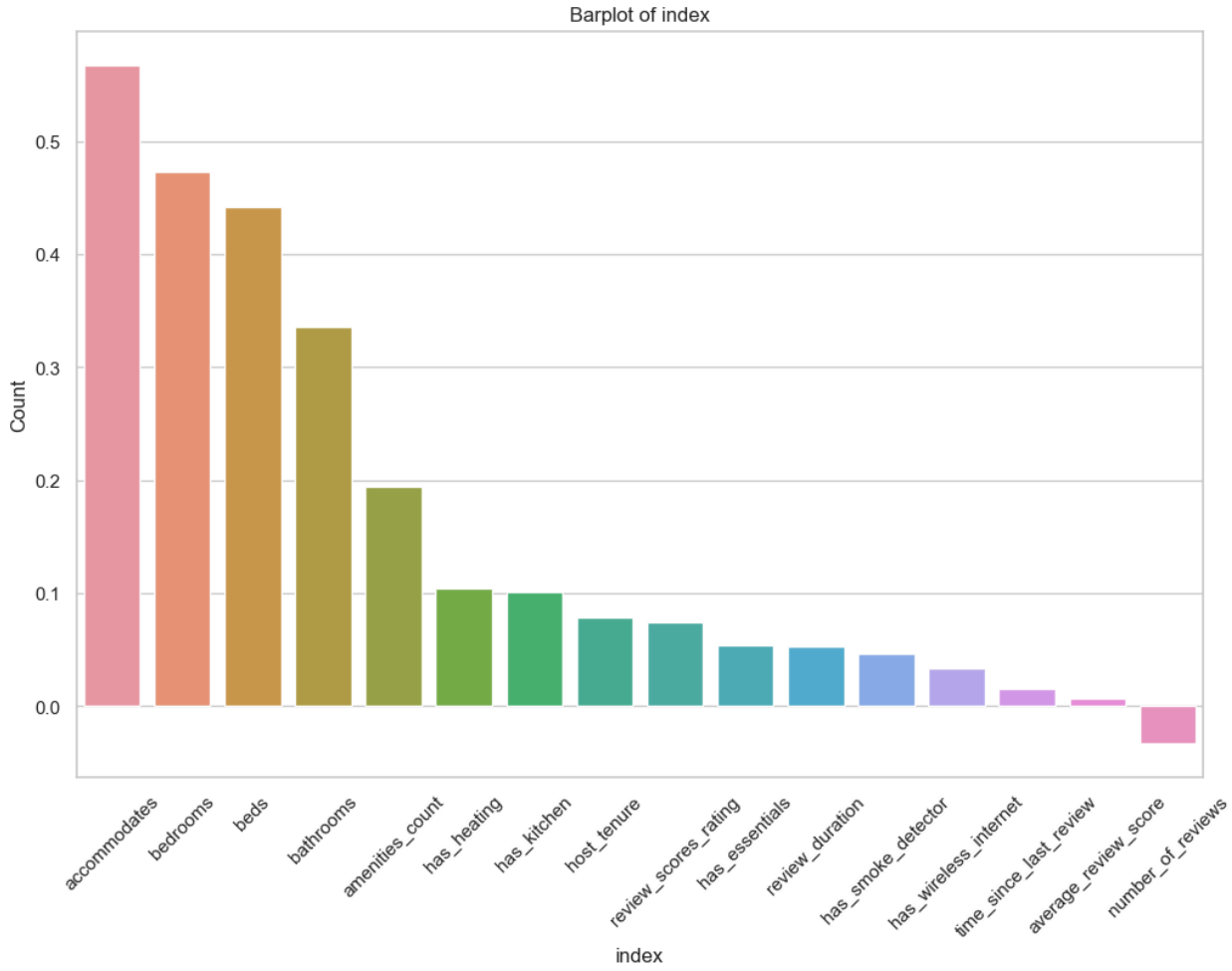
In [176…
```
correlatiomn_matrix = corr_df['log_price'].sort_values(ascending=False).to_frame().reset_index()
```

In [177…
```
correlatiomn_matrix
```

Out[177]:

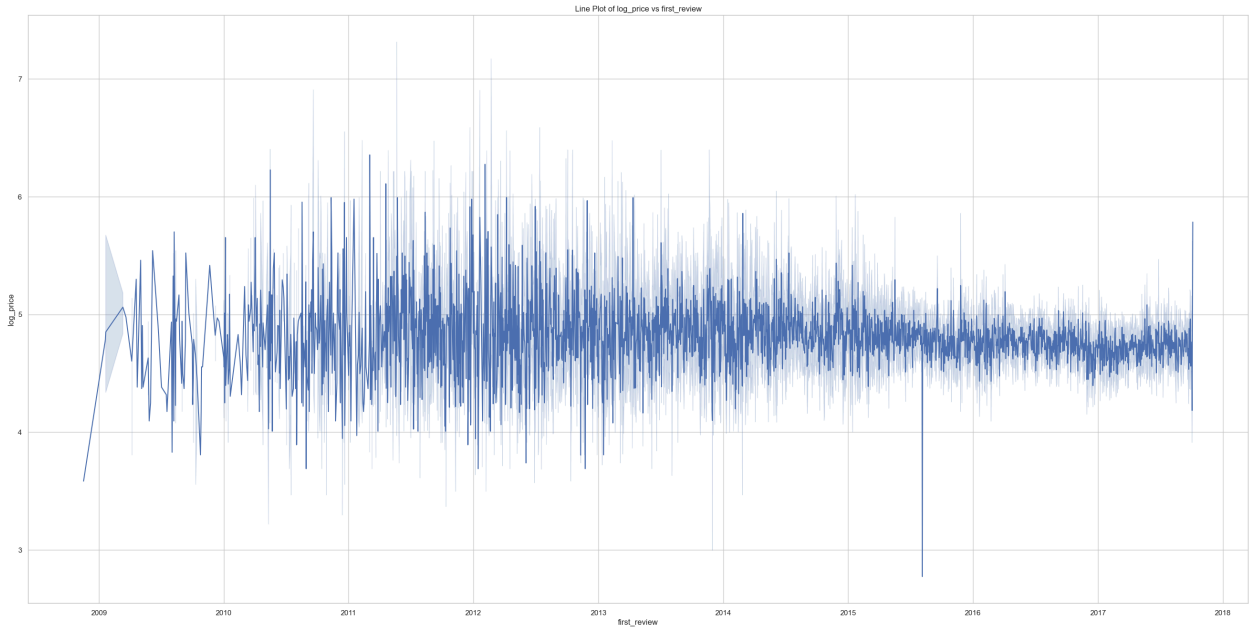
	index	log_price
0	log_price	1.000000
1	accommodates	0.567574
2	bedrooms	0.473028
3	beds	0.441619
4	bathrooms	0.335589
5	amenities_count	0.194528
6	has_heating	0.103929
7	has_kitchen	0.101046
8	host_tenure	0.078244
9	review_scores_rating	0.074475
10	has_essentials	0.053435
11	review_duration	0.052769
12	has_smoke_detector	0.046862
13	has_wireless_internet	0.034074
14	time_since_last_review	0.015599
15	average_review_score	0.007285
16	number_of_reviews	-0.032470

In [178…
```
plot_barplot(correlatiomn_matrix.iloc[1:], 'index', 'log_price') # Correlation matrix
```

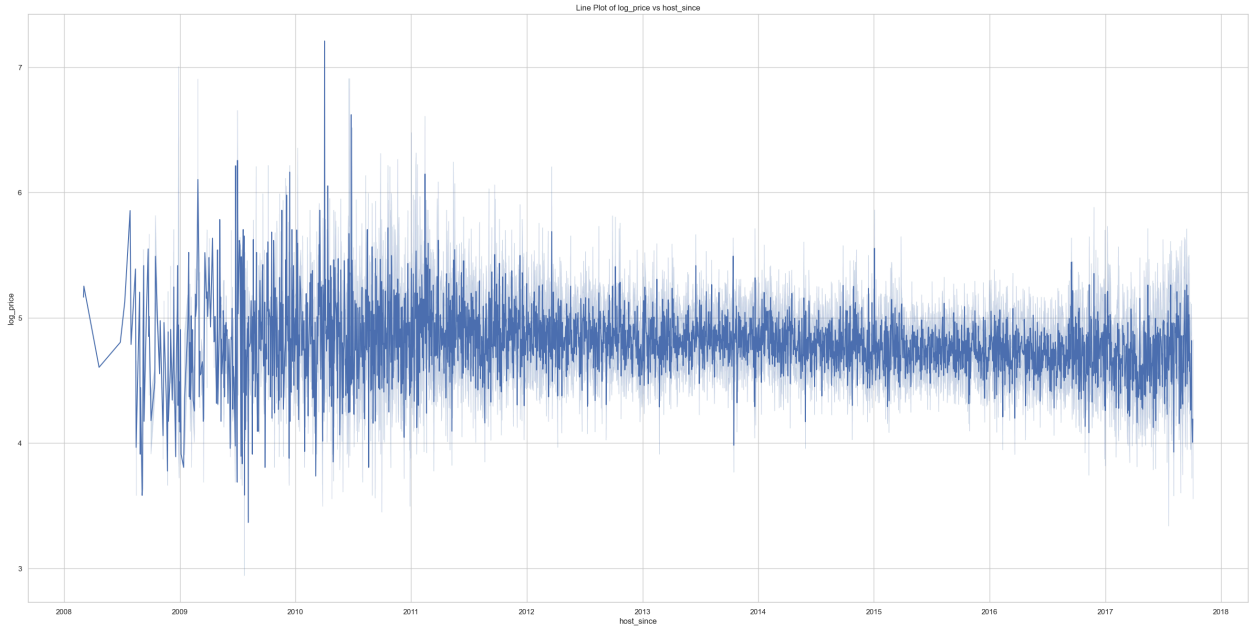


Out[178]: <AxesSubplot: title={'center': 'Barplot of index'}, xlabel='index', ylabel='Count'>

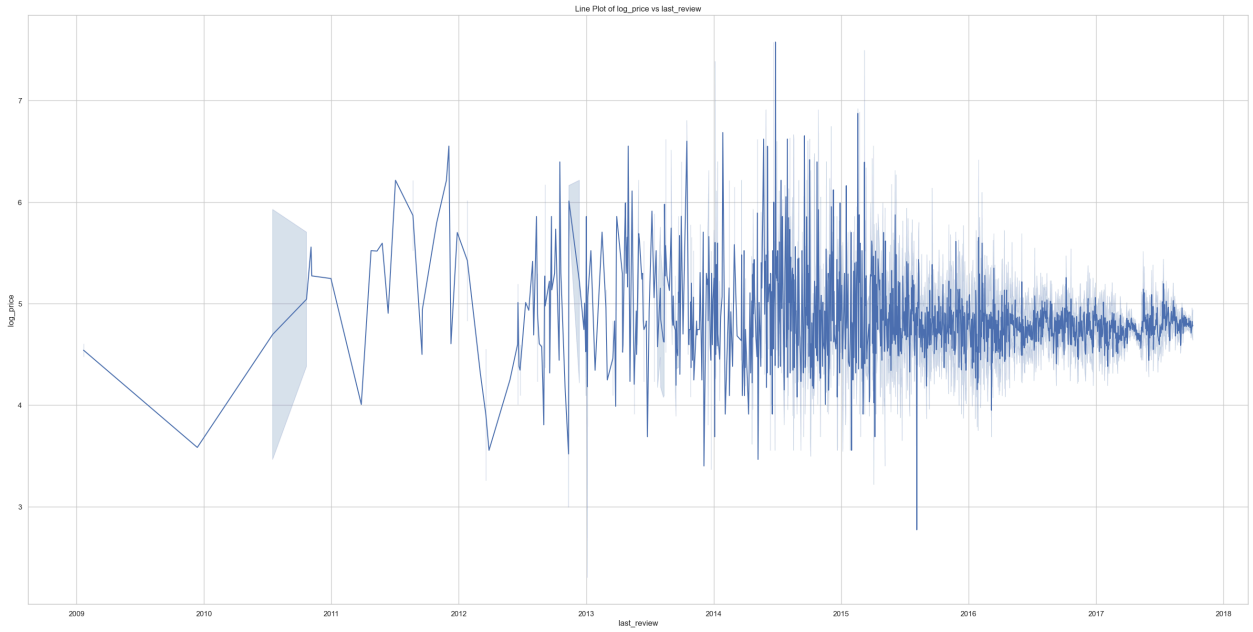
In [179…
```
# Example usage:
plot_lineplot(df, 'first_review','log_price' )
```



In [180…
```
plot_lineplot(df, 'host_since','log_price' )
```



In [181…
```
plot_lineplot(df, 'last_review','log_price' )
```



In [182…
```
df[['property_type','room_type']]
```

Out[182]:

	property_type	room_type
0	Apartment	Entire home/apt
1	Apartment	Entire home/apt
2	Apartment	Entire home/apt
3	House	Entire home/apt
4	Apartment	Entire home/apt
...	...	...
74106	Apartment	Private room
74107	Apartment	Entire home/apt
74108	Apartment	Entire home/apt
74109	Apartment	Entire home/apt
74110	Boat	Entire home/apt

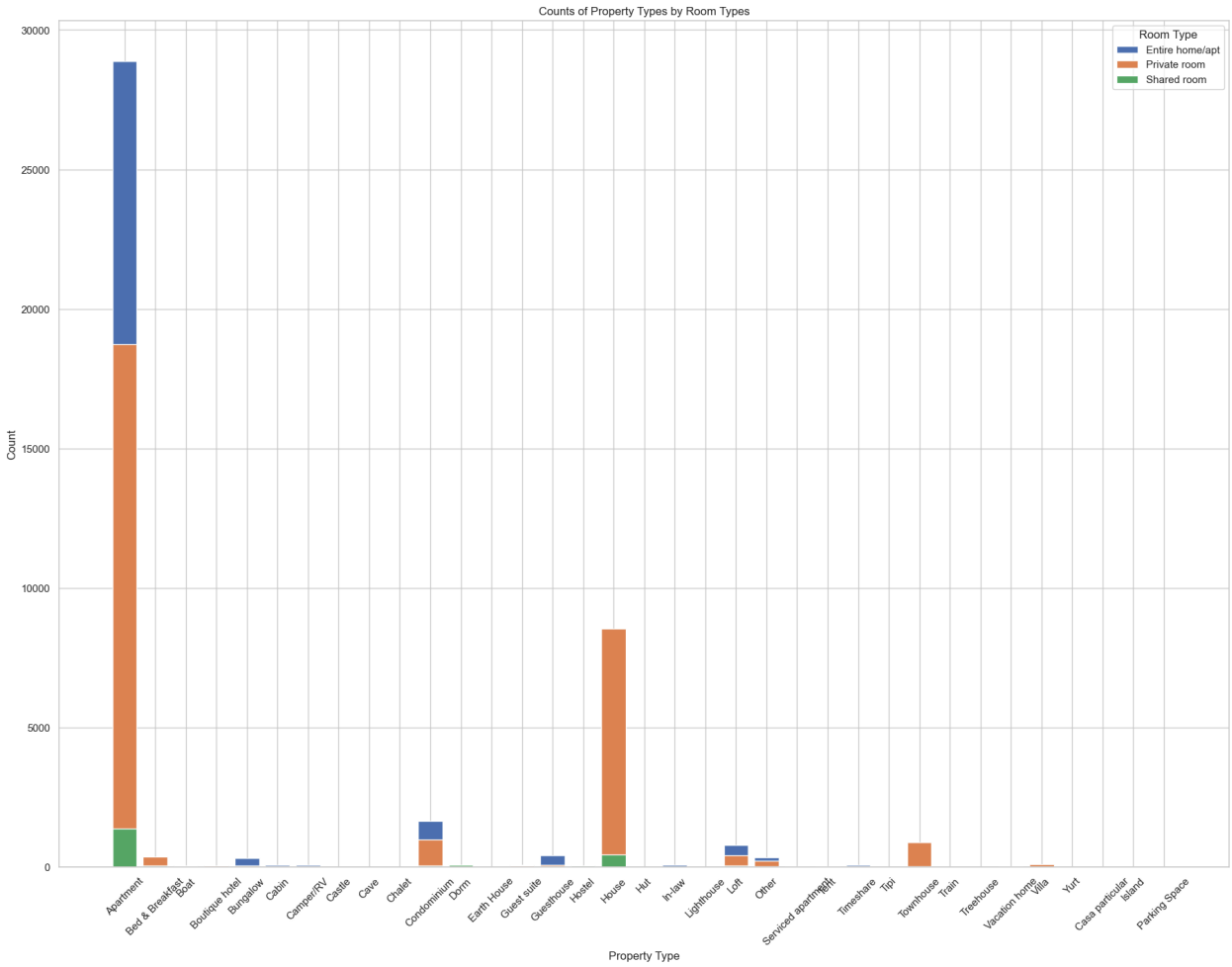
74111 rows × 2 columns

In [183…
```
pt_rt_df = df.groupby(['property_type','room_type'])['room_type'].count().to_frame()
```

```
pt_rt_df.columns = ['room_type_count']
pt_rt_df = pt_rt_df.reset_index()
```

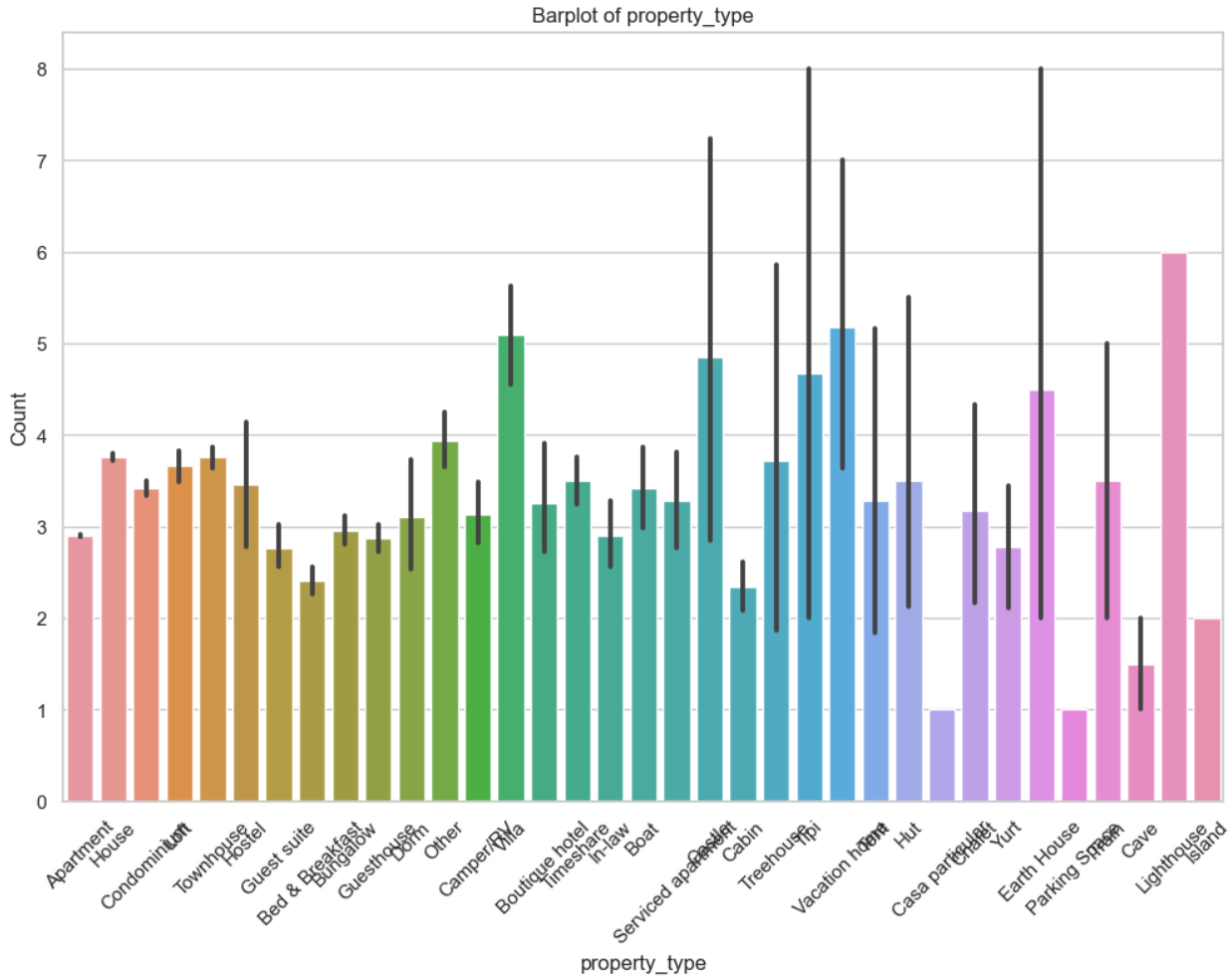
In [184…

```
# Set Seaborn style
sns.set_style("whitegrid")

# Plot unstacked bar chart
plt.figure(figsize=(22, 16))
for room_type in pt_rt_df['room_type'].unique():
    subset = pt_rt_df[pt_rt_df['room_type'] == room_type]
    plt.bar(subset['property_type'], subset['room_type_count'], label=room_type)

plt.title('Counts of Property Types by Room Types')
plt.xlabel('Property Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='Room Type')
plt.show()
```



In [185…

```
plot_barplot(df, 'property_type','accommodates')
```



Out[185]: <AxesSubplot: title={'center': 'Barplot of property_type'}, xlabel='property_type', ylabel='Count'>

In [186…
```
pt_cp_df = df.groupby(['property_type','cancellation_policy'])['cancellation_policy'].count().to_frame()

pt_cp_df.columns = ['cancellation_policy_count']
pt_cp_df = pt_cp_df.reset_index()
```

In [187…
```
pt_cp_df
```

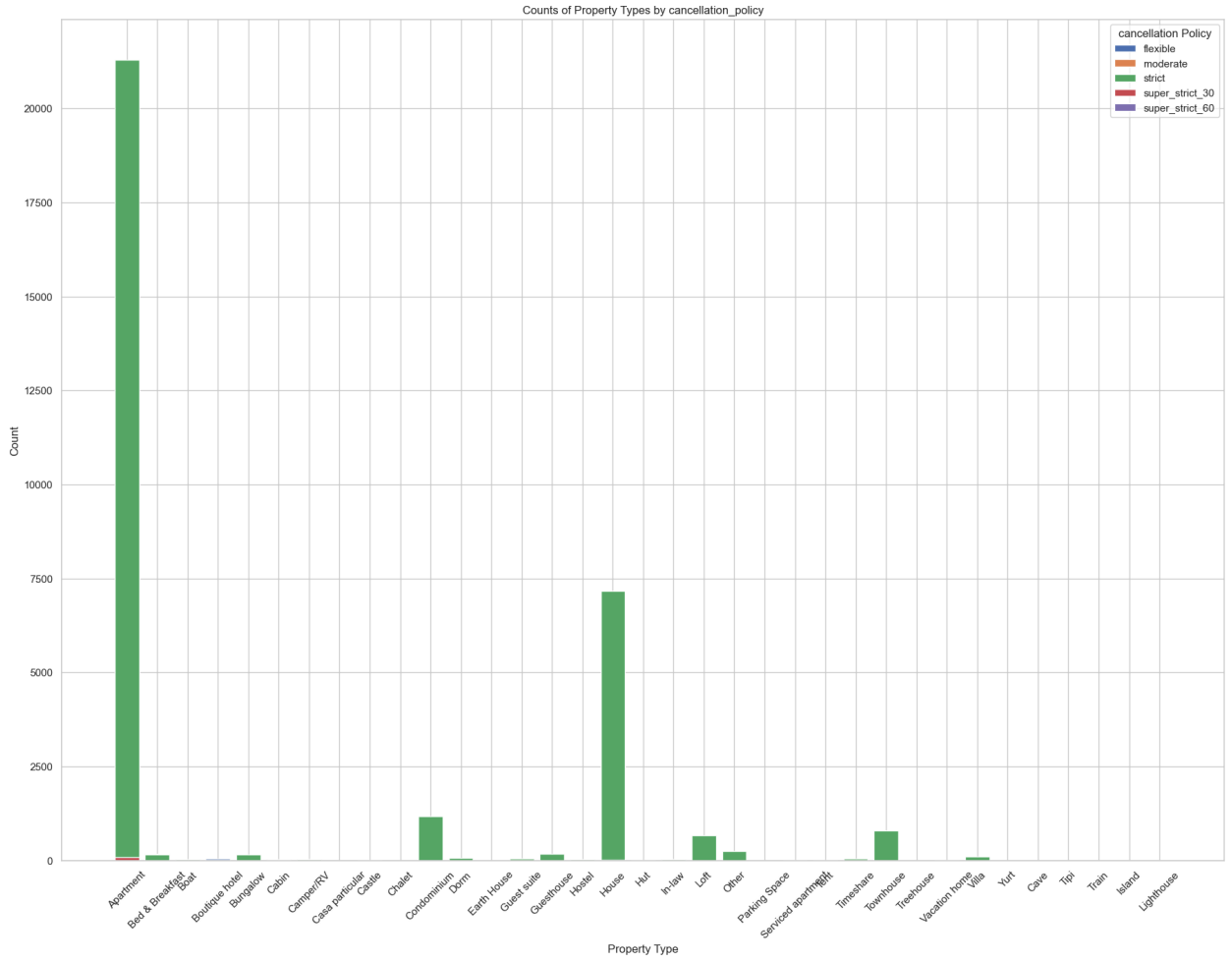
Out[187]:

	property_type	cancellation_policy	cancellation_policy_count
0	Apartment	flexible	15288
1	Apartment	moderate	12318
2	Apartment	strict	21301
3	Apartment	super_strict_30	96
4	Bed & Breakfast	flexible	187
...	...	...	...
92	Villa	strict	104
93	Villa	super_strict_60	5
94	Yurt	flexible	6
95	Yurt	moderate	2
96	Yurt	strict	1

97 rows × 3 columns

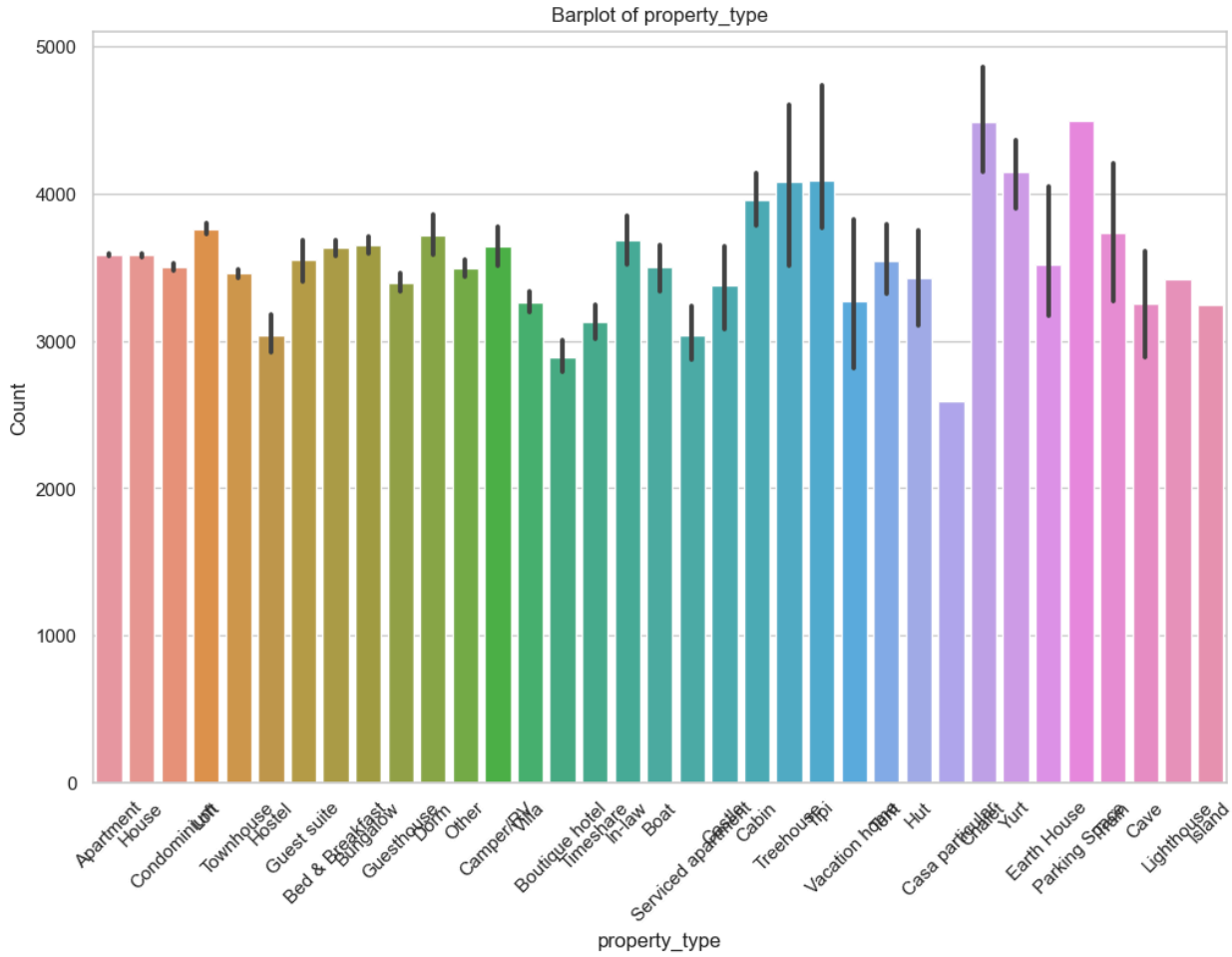
In [188…
```
# Set Seaborn style
sns.set_style("whitegrid")

# Plot unstacked bar chart
plt.figure(figsize=(22, 16))
for cancellation_policy in pt_cp_df['cancellation_policy'].unique():
    subset = pt_cp_df[pt_cp_df['cancellation_policy'] == cancellation_policy]
    plt.bar(subset['property_type'], subset['cancellation_policy_count'], label=cancellation_policy)

plt.title('Counts of Property Types by cancellation_policy')
plt.xlabel('Property Type')
```

```
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.legend(title='cancellation Policy')
plt.show()
```



In [189…

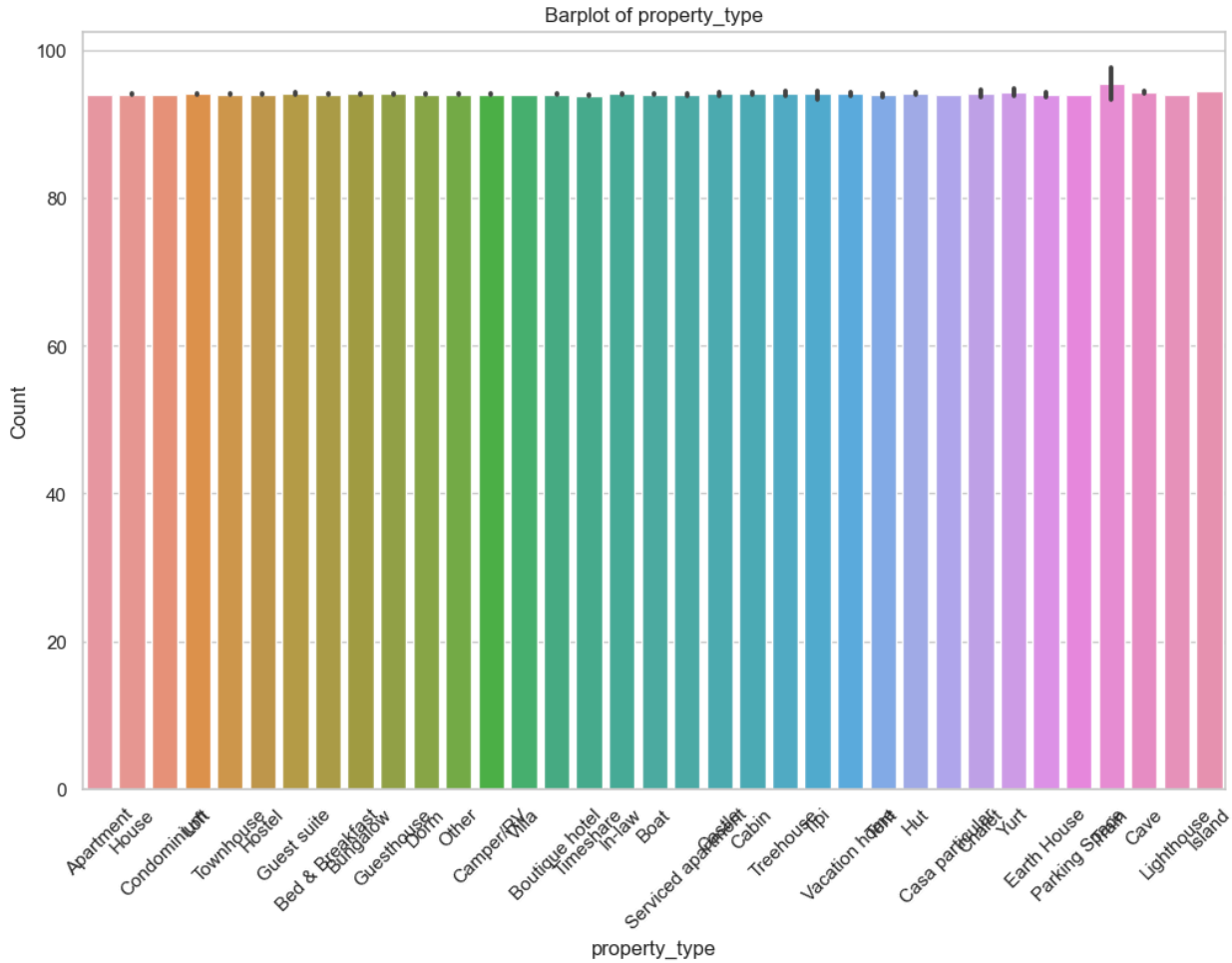
```
plot_barplot(df, 'property_type','host_tenure')
```



Out[189]: <AxesSubplot: title={'center': 'Barplot of property_type'}, xlabel='property_type', ylabel='Count'>

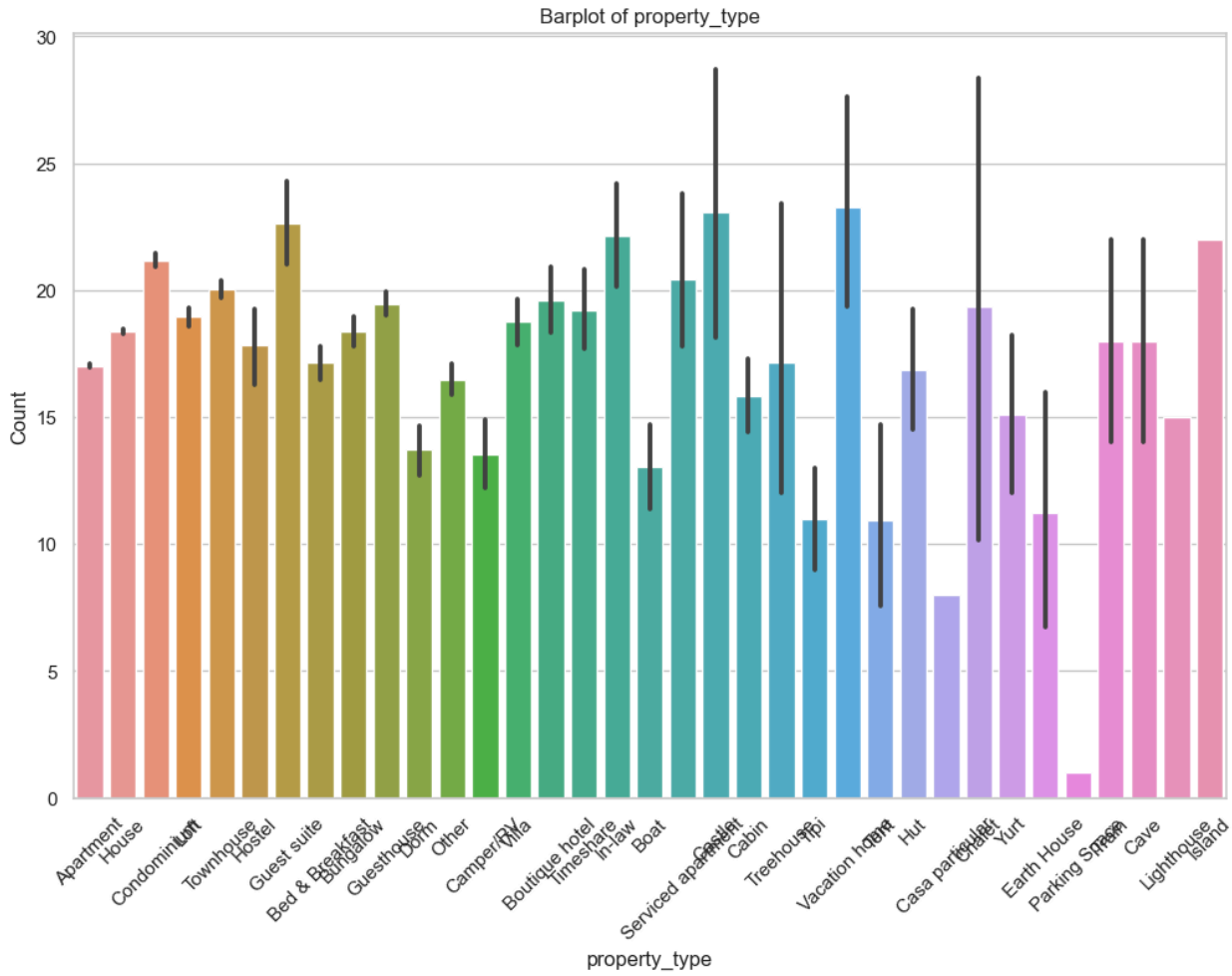
In [190…

```
plot_barplot(df, 'property_type','average_review_score')
```



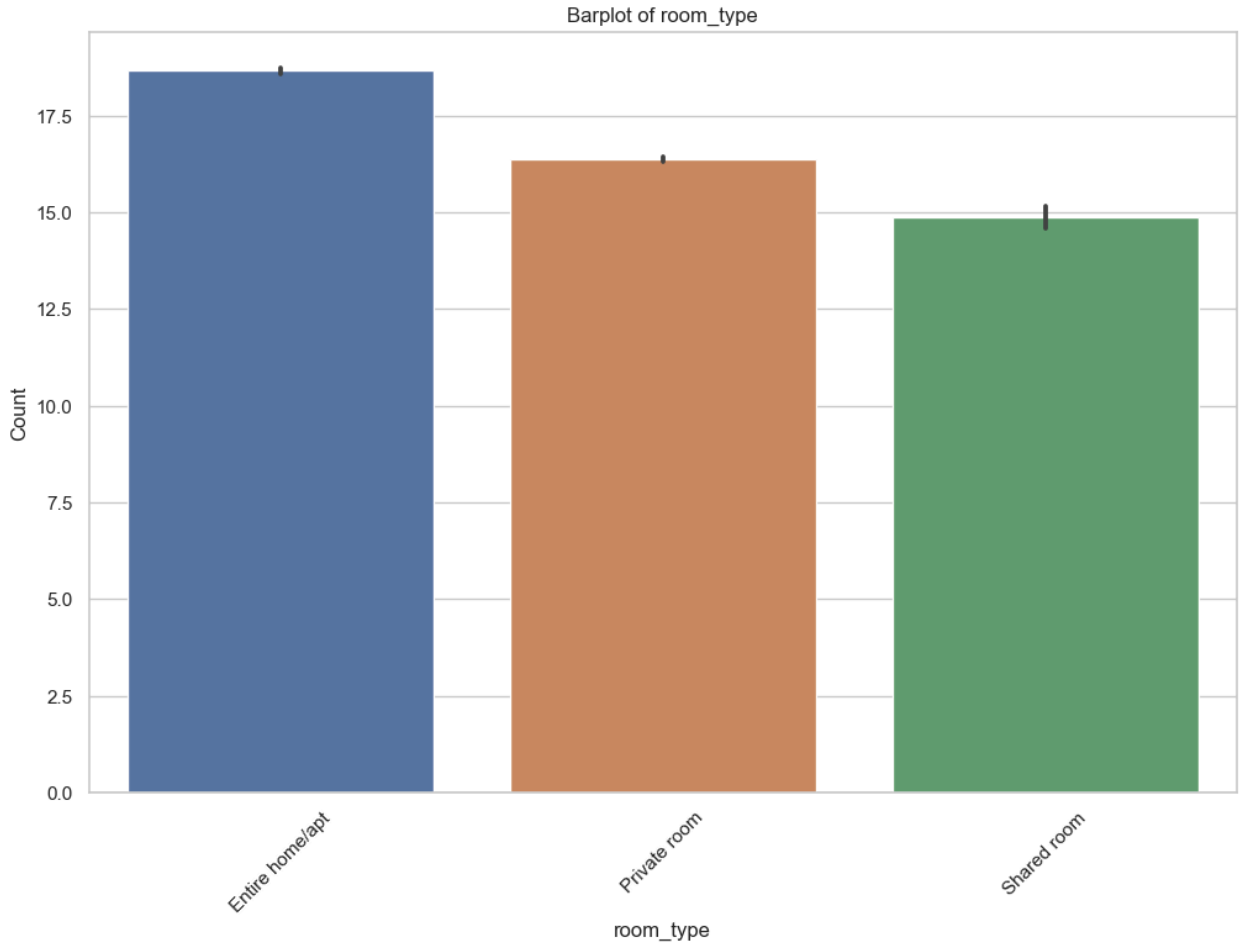
Out[190]: <AxesSubplot: title={'center': 'Barplot of property_type'}, xlabel='property_type', ylabel='Count'>

In [191…
```
plot_barplot(df, 'property_type','amenities_count')
```



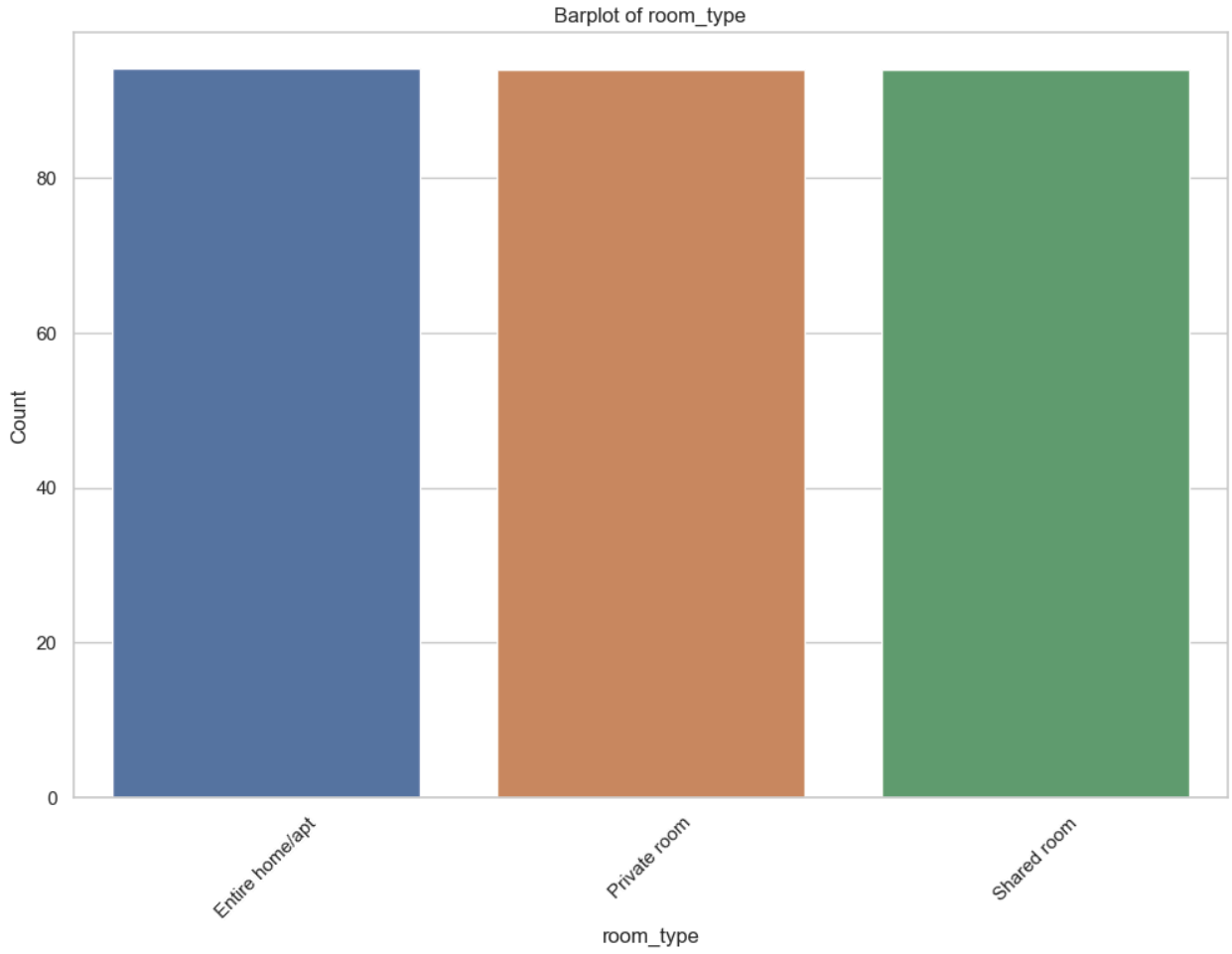
Out[191]: <AxesSubplot: title={'center': 'Barplot of property_type'}, xlabel='property_type', ylabel='Count'>

In [192…
```
plot_barplot(df, 'room_type','amenities_count')
```



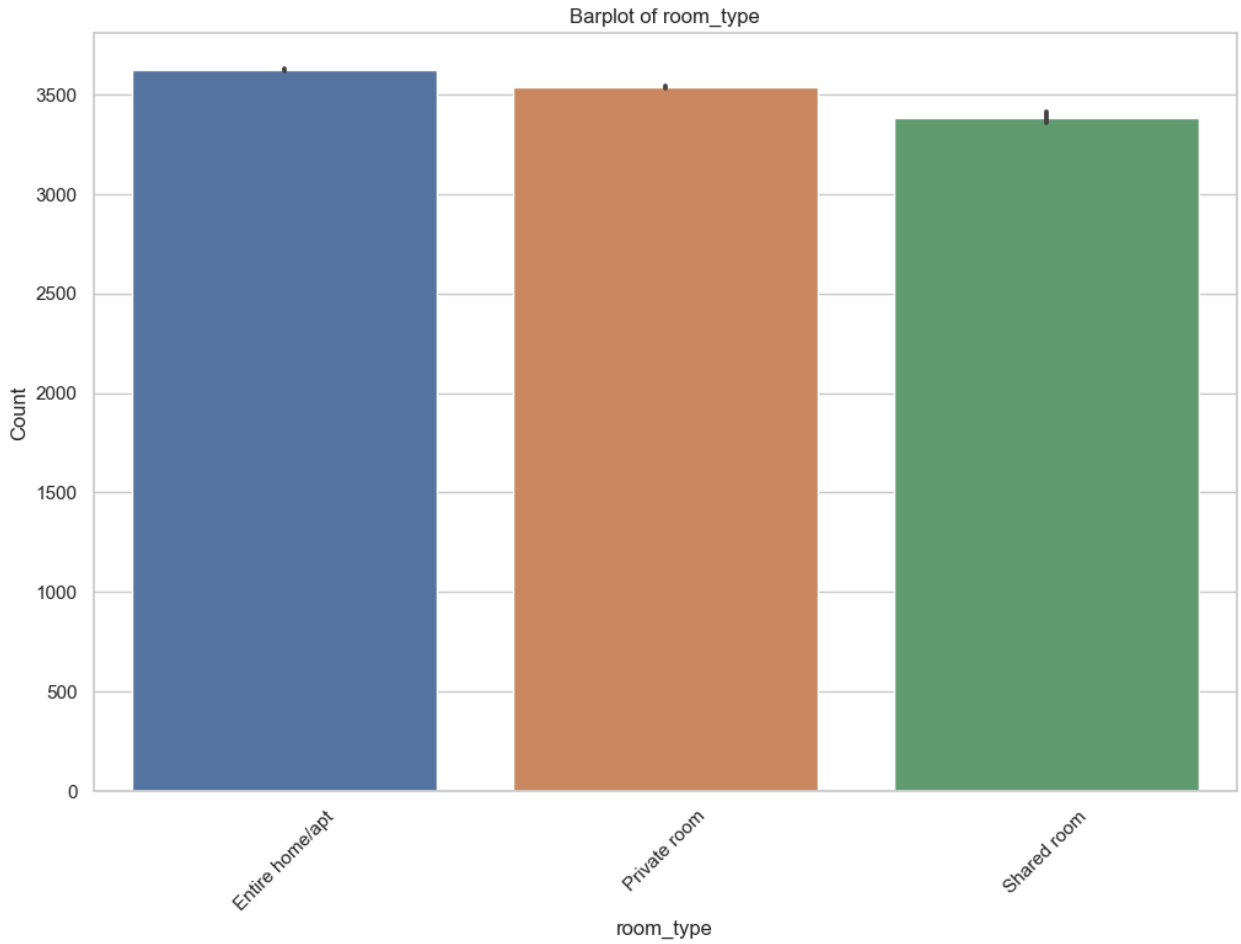
Out[192]: <AxesSubplot: title={'center': 'Barplot of room_type'}, xlabel='room_type', ylabel='Count'>

In [193…
```
plot_barplot(df, 'room_type','average_review_score')
```



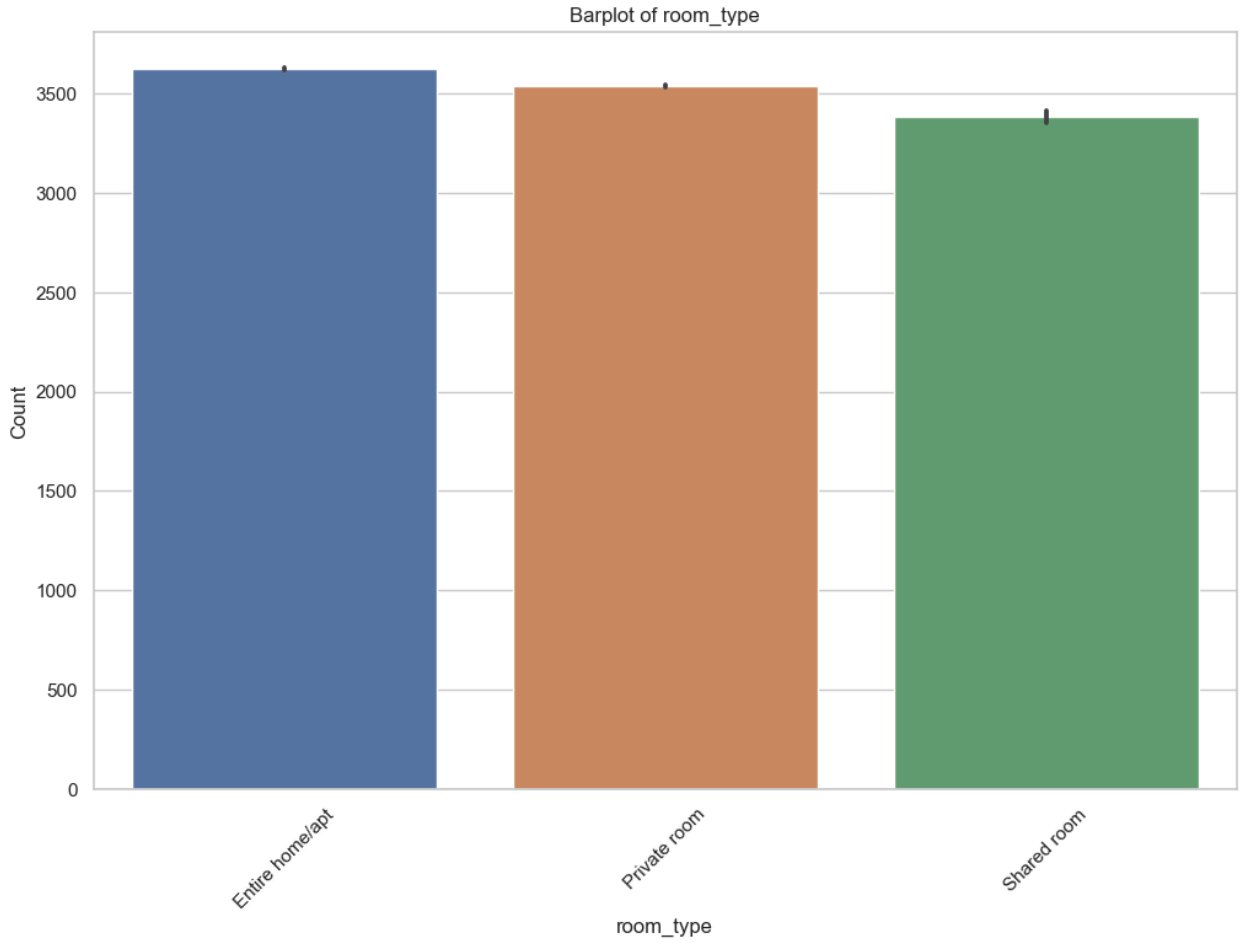
Out[193]: <AxesSubplot: title={'center': 'Barplot of room_type'}, xlabel='room_type', ylabel='Count'>

In [194…
```
plot_barplot(df, 'room_type','host_tenure')
```



Out[194]: <AxesSubplot: title={'center': 'Barplot of room_type'}, xlabel='room_type', ylabel='Count'>

In [195…
```
plot_barplot(df, 'room_type','host_tenure')
```



Out[195]: <AxesSubplot: title={'center': 'Barplot of room_type'}, xlabel='room_type', ylabel='Count'>

Sentiment Analysis

Here iam not performing any text preprocessing , because it is oneshot sentiment analysis if i done any stopwords remvoal or lemmatization or stemming it may be give less acurrcy hence iam skipping the part of preprocessing here

- we have covered ratings part here

In [196…

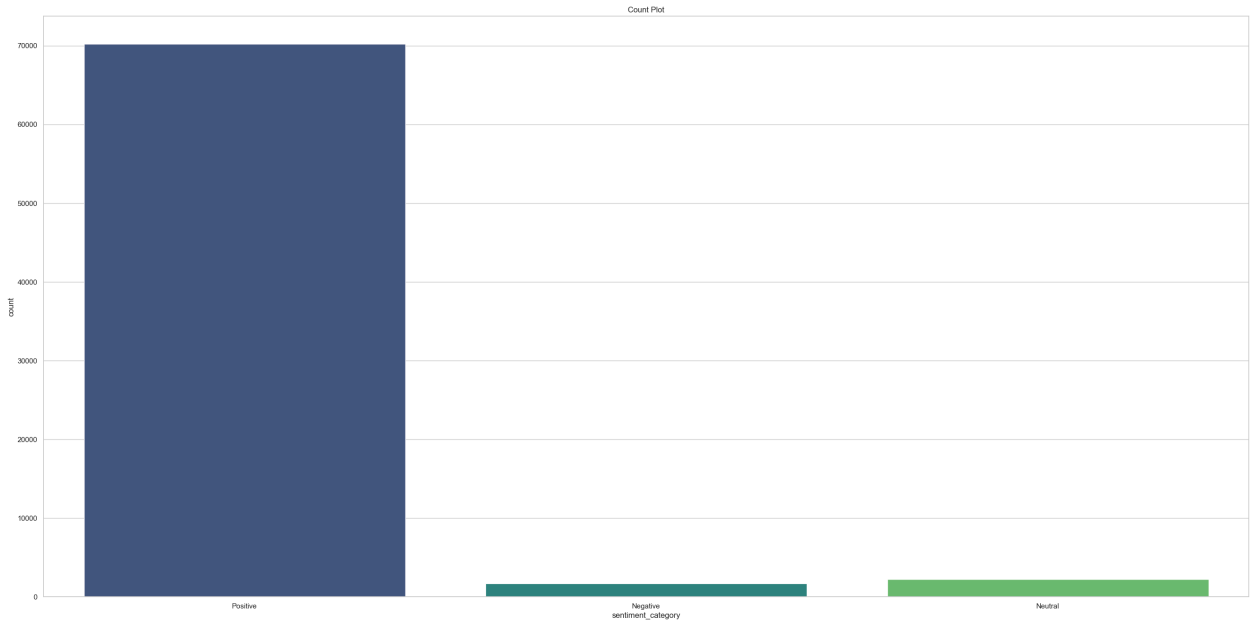
```
# Download VADER lexicon (if not already downloaded)
nltk.download('vader_lexicon')

def get_sentiment_category(text):
    """
    Analyzes the sentiment of the given text and returns the sentiment category.

    Parameters:
    - text (str): The text to analyze for sentiment.

    Returns:
    - str: The sentiment category ('Positive', 'Negative', or 'Neutral').
    """

    # Create sentiment analyzer
    analyzer = SentimentIntensityAnalyzer()

    # Get sentiment scores
    sentiment_scores = analyzer.polarity_scores(text)

    # Determine sentiment category based on the compound score
    compound_score = sentiment_scores['compound']
    if compound_score >= 0.05:
        sentiment_category = 'Positive'
    elif compound_score <= -0.05:
        sentiment_category = 'Negative'
    else:
        sentiment_category = 'Neutral'

    return sentiment_category
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\91845\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [201…
```
# df['sentiment_category'] = df['description'].apply(get_sentiment_category)

df_senti = pd.read_csv("intermediate.csv")

senti_dummy = pd.get_dummies(df['sentiment_category'],prefix='senti_',dtype=int)
```

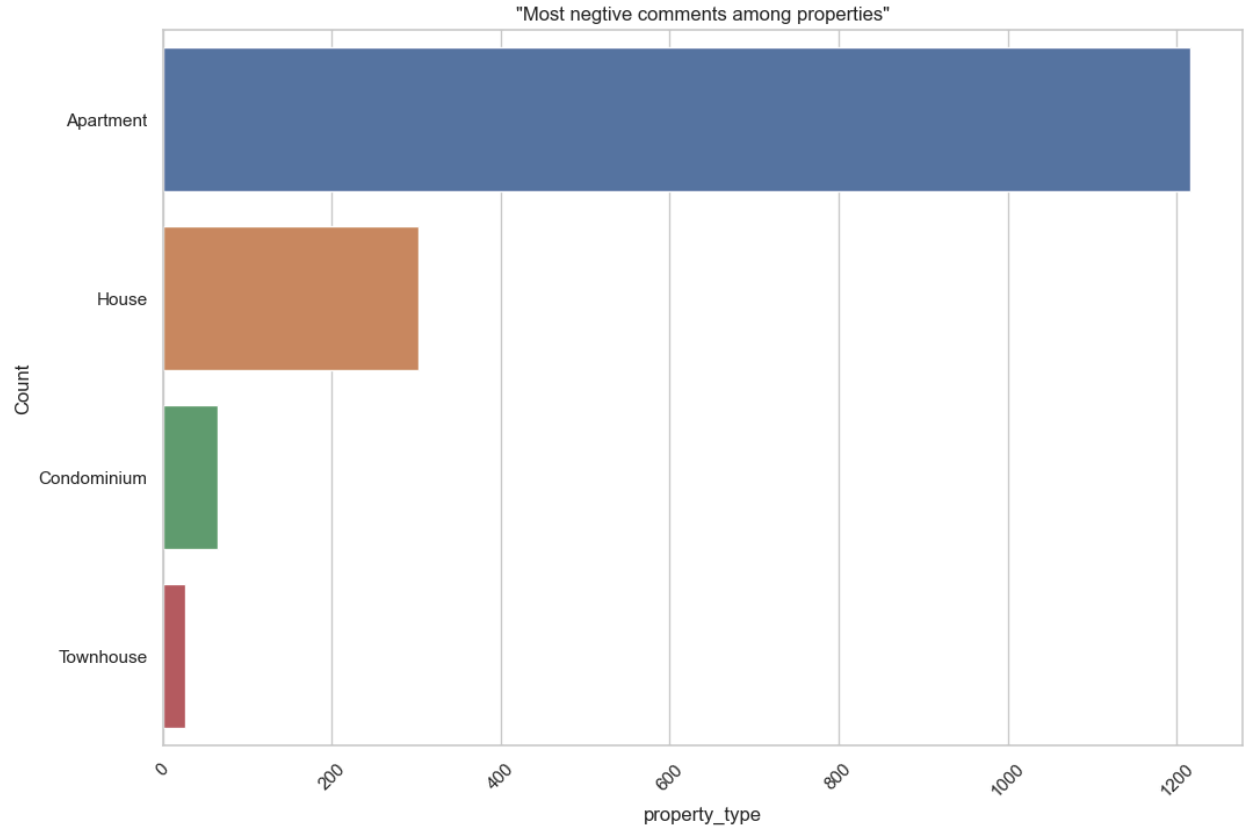
In [200…
```
# df.to_csv("intermediate.csv",index=False)
```

In [202…
```
create_seaborn_countplot(df_senti, 'sentiment_category')
```



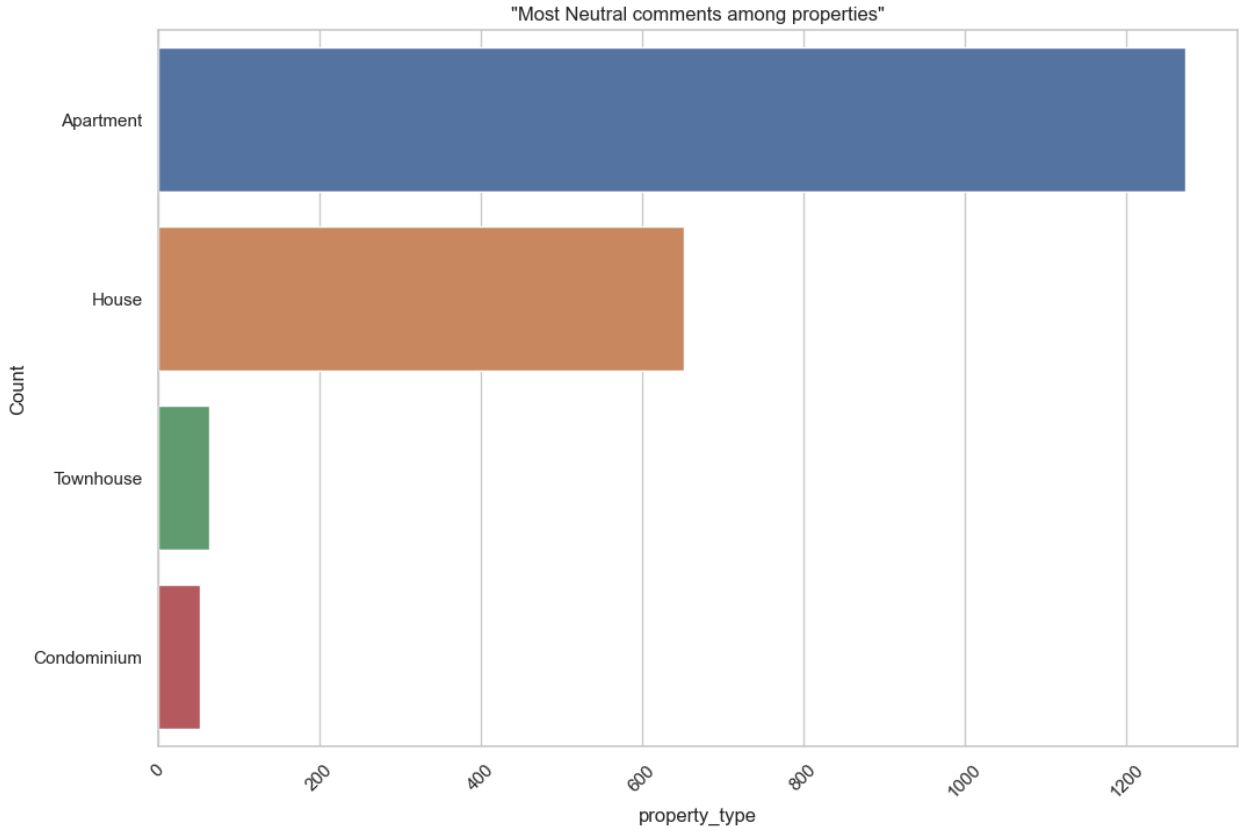
In [204…
```
pos_property_type = df_senti[df_senti['sentiment_category']=='Positive']['property_type'].value_counts()[:4].to_frame().reset_inc

plot_barplot(pos_property_type, 'property_type', 'count', title='"Most postive comments among properties"')
```

In [207…
```
neg_property_type = df_senti[df_senti['sentiment_category']=='Negative']['property_type'].value_counts()[:4].to_frame().reset_inc

plot_barplot(neg_property_type, 'property_type', 'index', title='"Most negtive comments among properties"')
```



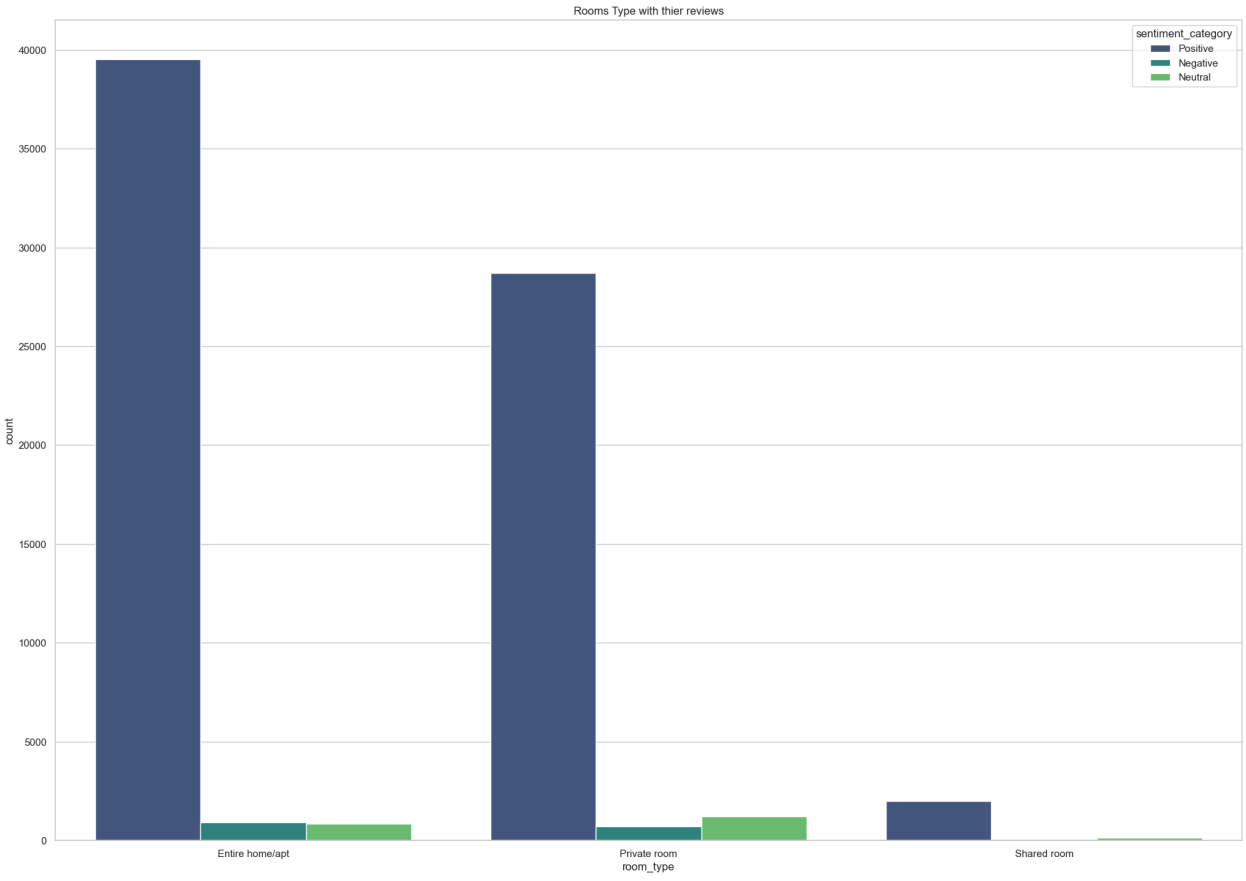
Out[207]: <AxesSubplot: title={'center': '"Most negtive comments among properties"'}, xlabel='property_type', ylabel='Count'>

In [208…
```
neu_property_type = df_senti[df_senti['sentiment_category']=='Neutral']['property_type'].value_counts()[:4].to_frame().reset_inde
plot_barplot(neu_property_type, 'property_type', 'index', title='"Most Neutral comments among properties"')
```



Out[208]: <AxesSubplot: title={'center': '"Most Neutral comments among properties"'}, xlabel='property_type', ylabel='Count'>

In [209…
```
plt.figure(figsize=(23, 16))
sns.countplot(x='room_type', data=df_senti, palette='viridis', hue='sentiment_category')
plt.title('Rooms Type with thier reviews')
plt.show()
```



In [ ]:
```
df_senti.columns
```

Geospatial Analysis

In [210…
```
df.columns
```

Out[210]:
```
Index(['id', 'log_price', 'property_type', 'room_type', 'amenities',
       'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy',
       'cleaning_fee', 'city', 'description', 'first_review',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rate',
       'host_since', 'instant_bookable', 'last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'number_of_reviews',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds',
       'amenities_count', 'has_wireless_internet', 'has_kitchen',
       'has_heating', 'has_essentials', 'has_smoke_detector',
       'review_duration', 'time_since_last_review', 'host_tenure',
       'average_review_score', 'sentiment_category'],
      dtype='object')
```

1. Price Distribution Map

In [211…
```
import plotly.express as px

fig = px.scatter_mapbox(df, lat="latitude", lon="longitude", color="log_price", size="log_price",
                        color_continuous_scale=px.colors.sequential.Viridis, size_max=25, zoom=10)
fig.update_layout(mapbox_style="carto-positron", title="Price Distribution Map")
fig.show()
```

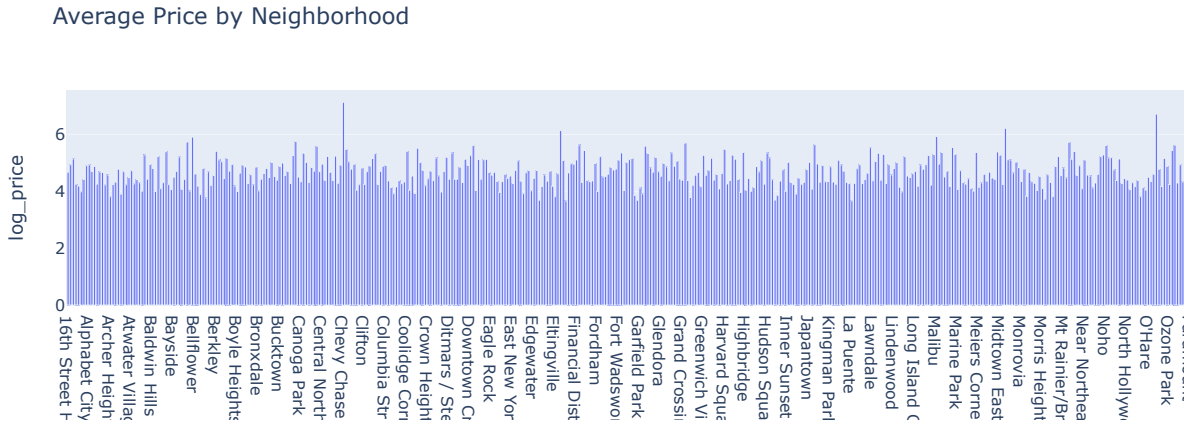
Price Distribution Map

Outcome:

This map will show the spatial distribution of listing prices. Areas with larger marker sizes indicate higher priced listings.

2. Neighborhood Price Comparison

In [212…
```
neighborhood_prices = df.groupby('neighbourhood')['log_price'].mean().reset_index()

fig = px.bar(neighborhood_prices, x='neighbourhood', y='log_price', title='Average Price by Neighborhood')
fig.show()
```



Outcome:

This bar plot shows the average listing prices for each neighborhood, allowing you to identify neighborhoods with higher or lower prices.

3. Spatial Clustering

In [213…

```
from sklearn.cluster import KMeans


kmeans = KMeans(n_clusters=5)  # Adjust number of clusters as needed
df['cluster'] = kmeans.fit_predict(df[['latitude', 'longitude']])

fig = px.scatter_mapbox(df, lat="latitude", lon="longitude", color="cluster",
                        color_continuous_scale=px.colors.sequential.Viridis, zoom=10)
fig.update_layout(mapbox_style="carto-positron", title="Spatial Clustering of Listings")
fig.show()
```

Outcome:

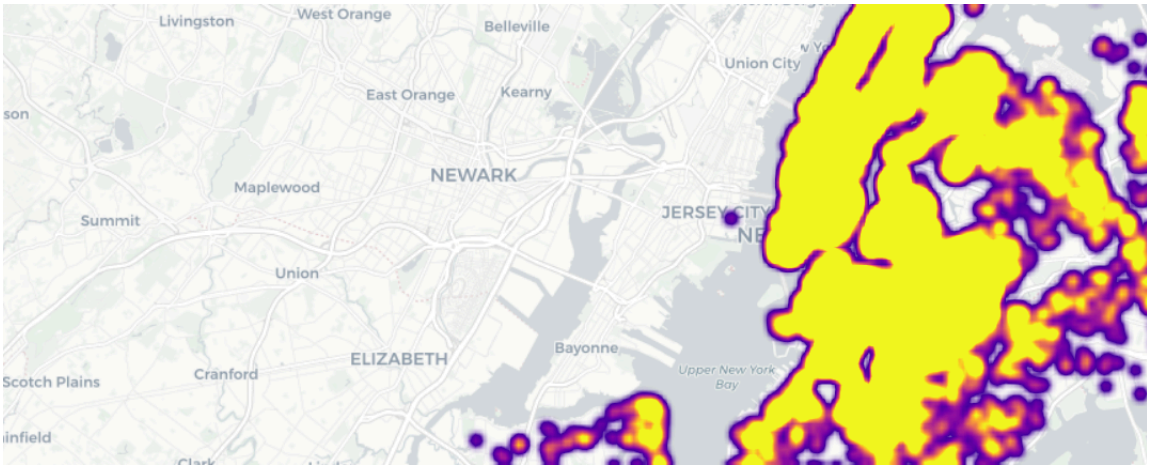
This map shows spatial clusters of listings with similar pricing patterns, helping identify areas with distinct pricing characteristics.

4. Heatmaps

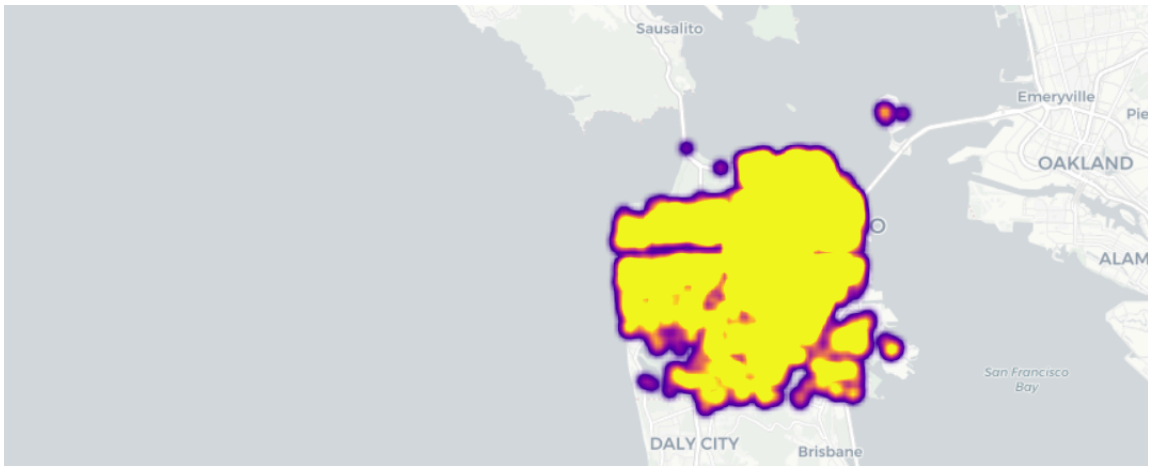
In [214…

```
import plotly.express as px

# Define city coordinates
city_coordinates = {
    'NYC': {'latitude': 40.7128, 'longitude': -74.0060},
    'SF': {'latitude': 37.7749, 'longitude': -122.4194},
    'DC': {'latitude': 38.9072, 'longitude': -77.0369},
    'LA': {'latitude': 34.0522, 'longitude': -118.2437},
    'Chicago': {'latitude': 41.8781, 'longitude': -87.6298},
    'Boston': {'latitude': 42.3601, 'longitude': -71.0589}
}

# Plot density map for each city
for city, coordinates in city_coordinates.items():
    fig = px.density_mapbox(df, lat='latitude', lon='longitude', z='log_price', radius=10,
                            center=dict(lat=coordinates['latitude'], lon=coordinates['longitude']), zoom=10,
                            mapbox_style="carto-positron", title=f"Density Map of Listing Prices in {city}")
    fig.show()
```

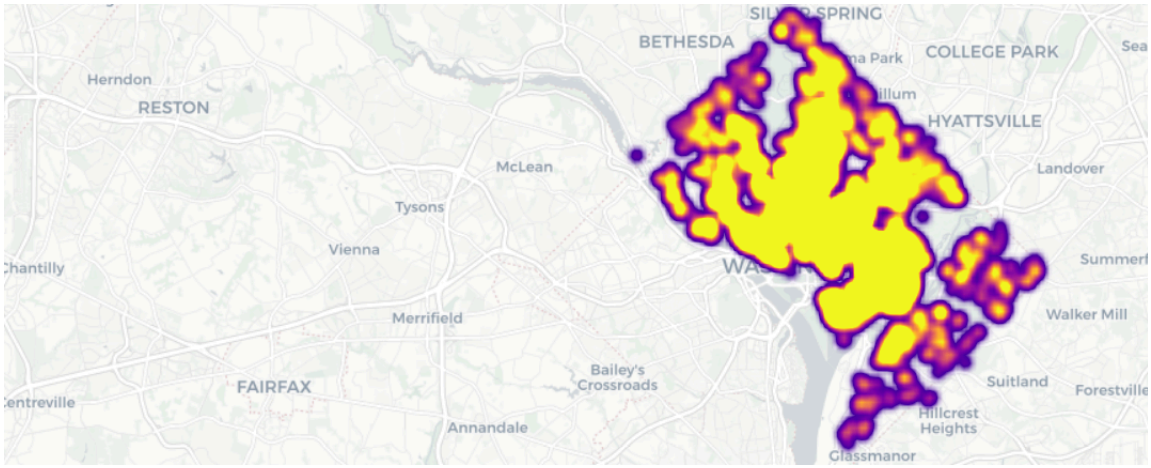
Density Map of Listing Prices in NYC



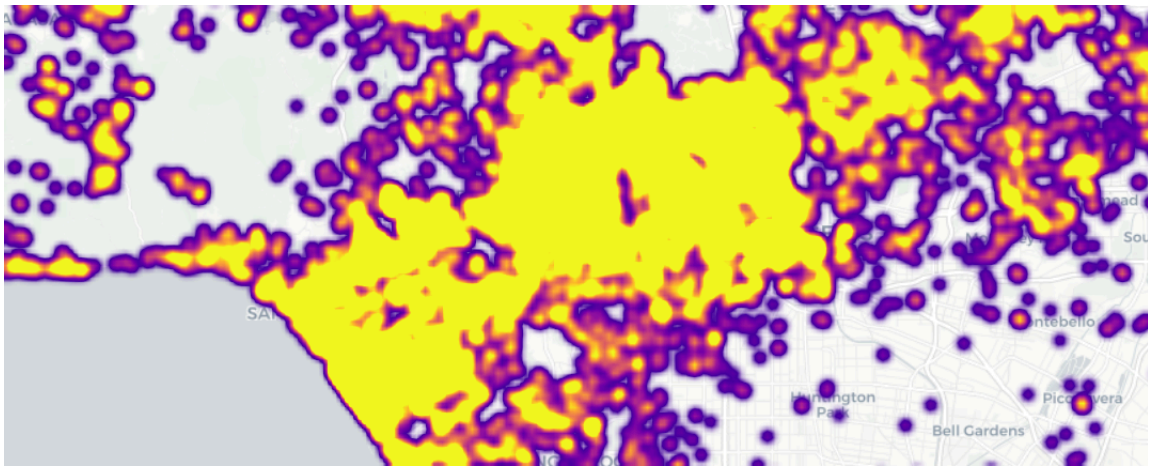
Density Map of Listing Prices in SF



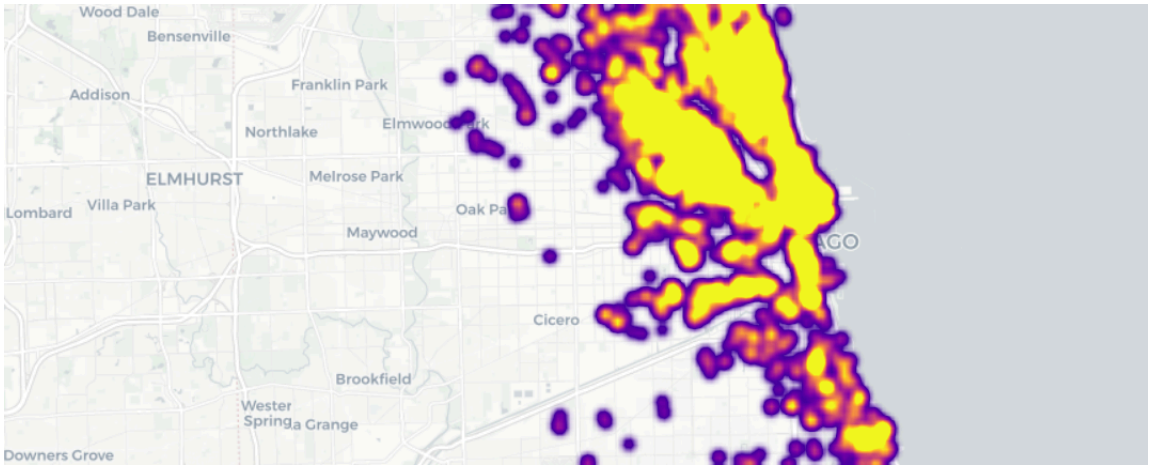
Density Map of Listing Prices in DC



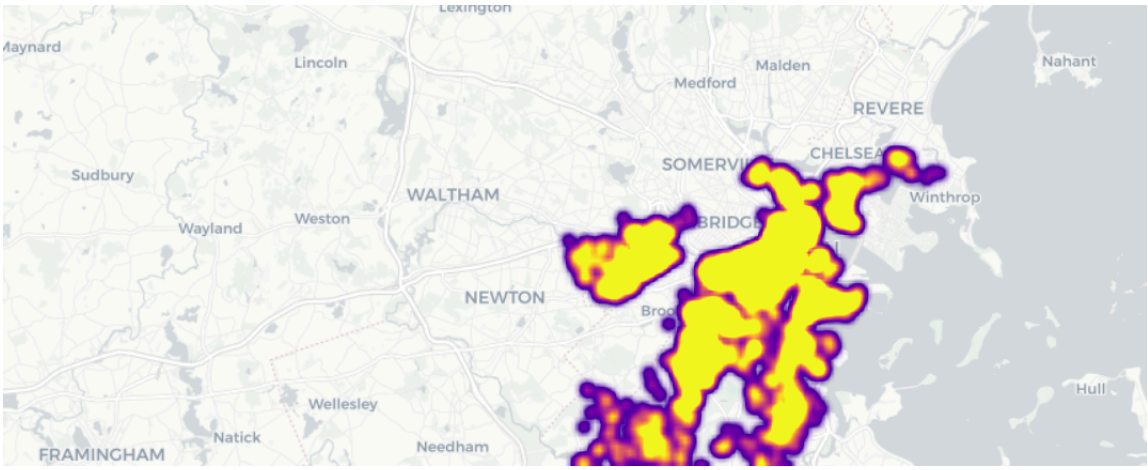
Density Map of Listing Prices in LA



Density Map of Listing Prices in Chicago



Density Map of Listing Prices in Boston



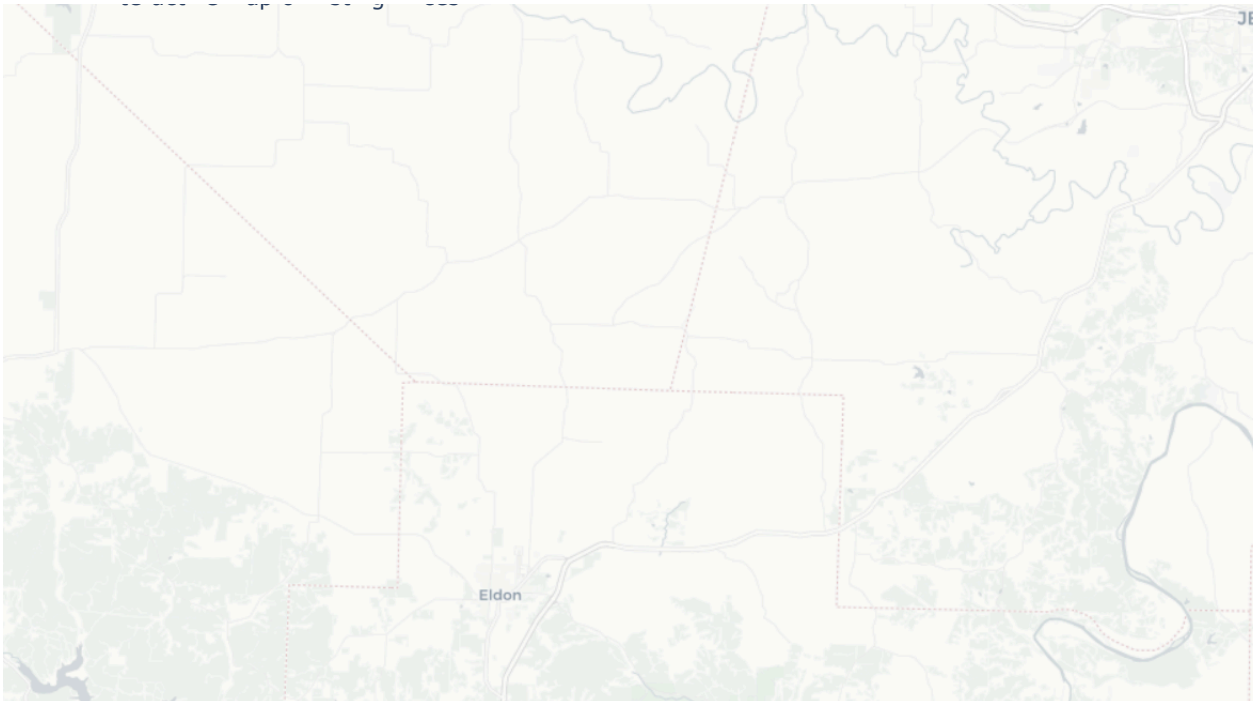
Outcome:

This heatmap visualizes the density of listing prices, with areas of higher density indicating regions with higher or lower prices.

5. Interactive Maps

In [215…

```
import plotly.express as px

# Create an interactive map
fig = px.scatter_mapbox(df, lat='latitude', lon='longitude', color='log_price', hover_name='neighbourhood',
                        hover_data=['log_price', 'property_type', 'room_type'],
                        color_continuous_scale=px.colors.sequential.Viridis, zoom=10)

# Update map layout
fig.update_layout(mapbox_style="carto-positron", title='Interactive Map of Listing Prices', margin={"r":0,"t":0,"l":0,"b":0})

# Show the interactive map
fig.show()
```



6. Proximity to City Centers

In [216…
```
from geopy.distance import great_circle

# Define city center coordinates
city_coordinates = {
    'NYC': {'latitude': 40.7128, 'longitude': -74.0060},
    'SF': {'latitude': 37.7749, 'longitude': -122.4194},
    'DC': {'latitude': 38.9072, 'longitude': -77.0369},
    'LA': {'latitude': 34.0522, 'longitude': -118.2437},
    'Chicago': {'latitude': 41.8781, 'longitude': -87.6298},
    'Boston': {'latitude': 42.3601, 'longitude': -71.0589}
}

# Function to calculate distance to city center
def calculate_distance(row):
    city = row['city']
    listing_coordinates = (row['latitude'], row['longitude'])
    city_center = (city_coordinates[city]['latitude'], city_coordinates[city]['longitude'])
    return great_circle(city_center, listing_coordinates).miles

# Apply the function to create the distance_to_city_center column
df['distance_to_city_center'] = df.apply(calculate_distance, axis=1)

# Plot scatter map
fig = px.scatter_mapbox(df, lat="latitude", lon="longitude", color="distance_to_city_center",
                        color_continuous_scale=px.colors.sequential.Viridis, zoom=10,
                        mapbox_style="carto-positron", title="Proximity to City Centers")
fig.show()
```

Proximity to City Centers

Outcome:

This map shows listings color-coded by their distance from the city center, helping you identify areas closer or farther from the city center.

In [1]:
```
!jupyter nbconvert --to html HomeStay_EDA.ipynb
```

In [ ]:
```

```

In [ ]:
```
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import shap

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# from evidently.report import Report
# from evidently.metrics import RegressionQualityMetric, RegressionErrorPlot, Re
# from evidently.metric_preset import DataDriftPreset, RegressionPreset
```

In [ ]:
```
df = pd.read_csv("/kaggle/input/training-data/training_data (2).csv")
sentiment_data = pd.read_csv("/kaggle/input/senti-data/senti_data.csv")
```

In [ ]:
```
# Initialize a dictionary to store label encoders
label_encoders = {}

# Create a copy of the original DataFrame to preserve the original data
encoded_df = df.copy()

# List of categorical columns to be label encoded
categorical_columns = ['property_type', 'room_type', 'bed_type',
                       'cancellation_policy', 'city',
                       'host_has_profile_pic', 'host_identity_verified',
                       'instant_bookable', 'cleaning_fee']

# Apply label encoding to each categorical column
for column in categorical_columns:
    # Initialize LabelEncoder
    label_encoder = LabelEncoder()

    # Apply label encoding to the column
    encoded_df[column] = label_encoder.fit_transform(encoded_df[column])

    # Store the label encoder object in the dictionary
    label_encoders[column] = label_encoder

# Save the label encoders to a file using pickle
with open('label_encoders.pkl', 'wb') as f:
    pickle.dump(label_encoders, f)
```

In [ ]:
```
# Extracting month from date columns and adding them as new columns
df['first_review_m'] = pd.to_datetime(df['first_review']).dt.month
df['host_since_m'] = pd.to_datetime(df['host_since']).dt.month
df['last_review_m'] = pd.to_datetime(df['last_review']).dt.month

# Selecting columns of interest
int_df = df[['id', 'log_price', 'accommodates', 'bathrooms', 'zipcode','neighbou
            'host_response_rate', 'number_of_reviews', 'review_scores_rating',
```

```
            'bedrooms', 'latitude', 'longitude','beds', 'review_duration', 'tim
            'host_tenure', 'average_review_score', 'amenities_count',
            'has_wireless_internet', 'has_kitchen', 'has_heating', 'has_essenti
            'has_smoke_detector','first_review_m', 'host_since_m', 'last_review
```

In [ ]:
```
label_encoder_senti = LabelEncoder()
sentiment_data['sentiment_category'] = label_encoder_senti.fit_transform(sentime

# Save this to labelencoders dict
label_encoders['sentiment_category'] = label_encoder_senti
```

In [ ]:
```
Model_training_df = pd.concat([int_df, encoded_df[categorical_columns],sentiment
```

In [ ]:
```
X = Model_training_df.drop("log_price", axis=1)

y = Model_training_df['log_price']
```

In [ ]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize regression models
models = {
    'linear_regression': LinearRegression(),
    'ridge_regression': Ridge(),
    'lasso_regression': Lasso(),
    'elastic_net_regression': ElasticNet(),
    'decision_tree_regression': DecisionTreeRegressor(),
    'random_forest_regression': RandomForestRegressor(),
    'gradient_boosting_regression': GradientBoostingRegressor(),
    'xgboost_regression': xgb.XGBRegressor(),
    'lightgbm_regression': lgb.LGBMRegressor(),
    'catboost_regression': CatBoostRegressor(silent=True)
}

# Train each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Calculate Mean Squared Error (MSE)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    # Print the evaluation metric
    print(f"{name} MSE: {mse}")
    print(f"{name} MSE: {mae}")

    print("-"*30)


# Save the models dictionary to a pickle file
with open('trained_models.pkl', 'wb') as f:
    pickle.dump(models, f)
```

```
linear_regression MSE: 0.2345703679069435
linear_regression MSE: 0.36350034347637267
------------------------------
ridge_regression MSE: 0.23457080577368813
ridge_regression MSE: 0.3635002848955916
------------------------------
lasso_regression MSE: 0.5059901469503172
lasso_regression MSE: 0.5550222372563071
------------------------------
elastic_net_regression MSE: 0.3999987876578443
elastic_net_regression MSE: 0.48878860969443066
------------------------------
decision_tree_regression MSE: 0.318172686448424
decision_tree_regression MSE: 0.40878492734756794
------------------------------
KeyboardInterrupt
```

In [ ]:
```python
# Load the trained models from the pickle file
with open('trained_models.pkl', 'rb') as f:
    models = pickle.load(f)

# Initialize dictionaries to store evaluation metrics for each model
evaluation_metrics = {
    'Model': [],
    'MSE': [],
    'MAE': [],
    'R2': []
}

# Evaluate the performance of each model
for name, model in models.items():
    # Predict on the test data
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store evaluation metrics in the dictionary
    evaluation_metrics['Model'].append(name)
    evaluation_metrics['MSE'].append(mse)
    evaluation_metrics['MAE'].append(mae)
    evaluation_metrics['R2'].append(r2)
```

From evaluvation we came to know XGBoost, LightBoost, Catboost performed well hence hyperparameter tuning those

In [ ]:
```python
# Initialize XGBoost Regression with hyperparameters
xgb_reg = xgb.XGBRegressor(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
```

```
    objective='reg:squarederror',
    random_state=42
)

# Initialize LightGBM Regression with hyperparameters
lgb_reg = lgb.LGBMRegressor(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='regression',
    random_state=42
)

# Initialize CatBoost Regression with hyperparameters
catboost_reg = CatBoostRegressor(
    n_estimators=100,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bylevel=0.8,
    objective='MAE',
    random_seed=42
)

# Train XGBoost Regression
xgb_reg.fit(X_train, y_train)

# Train LightGBM Regression
lgb_reg.fit(X_train, y_train)

# Train CatBoost Regression
catboost_reg.fit(X_train, y_train)

# Initialize a dictionary to store models, evaluation metrics, and feature impor
trained_models = {
    'XGBoost': {'model': xgb_reg, 'evaluation_metrics': {}, 'feature_importance'
    'LightGBM': {'model': lgb_reg, 'evaluation_metrics': {}, 'feature_importance
    'CatBoost': {'model': catboost_reg, 'evaluation_metrics': {}, 'feature_impor
}

# Evaluate each model and store evaluation metrics
for model_name, model_data in trained_models.items():
    model = model_data['model']
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    model_data['evaluation_metrics'] = {'MSE': mse, 'MAE': mae, 'R2': r2}

# Store feature importance for XGBoost and LightGBM models
trained_models['XGBoost']['feature_importance'] = xgb_reg.feature_importances_
trained_models['LightGBM']['feature_importance'] = lgb_reg.feature_importances_

# CatBoost provides feature importance as part of its model object
trained_models['CatBoost']['feature_importance'] = catboost_reg.get_feature_impo

# Save the trained models dictionary to a pickle file
```

```
with open('trained_models_with_metrics.pkl', 'wb') as f:
    pickle.dump(trained_models, f)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.057669 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2823
[LightGBM] [Info] Number of data points in the train set: 59288, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 4.780538
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
0:      learn: 0.5283770        total: 19.4ms   remaining: 1.92s
1:      learn: 0.5005708        total: 38.6ms   remaining: 1.89s
2:      learn: 0.4777701        total: 54.9ms   remaining: 1.77s
3:      learn: 0.4573508        total: 71ms     remaining: 1.71s
4:      learn: 0.4394896        total: 87.8ms   remaining: 1.67s
5:      learn: 0.4254715        total: 105ms    remaining: 1.64s
6:      learn: 0.4120376        total: 121ms    remaining: 1.61s
7:      learn: 0.4006176        total: 138ms    remaining: 1.58s
8:      learn: 0.3912509        total: 155ms    remaining: 1.57s
9:      learn: 0.3837547        total: 172ms    remaining: 1.54s
10:     learn: 0.3763976        total: 187ms    remaining: 1.51s
11:     learn: 0.3699939        total: 204ms    remaining: 1.49s
12:     learn: 0.3645251        total: 221ms    remaining: 1.48s
13:     learn: 0.3592366        total: 238ms    remaining: 1.46s
14:     learn: 0.3551481        total: 254ms    remaining: 1.44s
15:     learn: 0.3514148        total: 270ms    remaining: 1.42s
16:     learn: 0.3474932        total: 286ms    remaining: 1.4s
17:     learn: 0.3448925        total: 302ms    remaining: 1.37s
18:     learn: 0.3426493        total: 318ms    remaining: 1.35s
19:     learn: 0.3398243        total: 334ms    remaining: 1.34s
20:     learn: 0.3372497        total: 351ms    remaining: 1.32s
21:     learn: 0.3350543        total: 367ms    remaining: 1.3s
22:     learn: 0.3326743        total: 383ms    remaining: 1.28s
23:     learn: 0.3305303        total: 400ms    remaining: 1.26s
24:     learn: 0.3289850        total: 415ms    remaining: 1.25s
25:     learn: 0.3274024        total: 434ms    remaining: 1.23s
26:     learn: 0.3260330        total: 450ms    remaining: 1.22s
27:     learn: 0.3245224        total: 466ms    remaining: 1.2s
28:     learn: 0.3233772        total: 482ms    remaining: 1.18s
29:     learn: 0.3220635        total: 498ms    remaining: 1.16s
30:     learn: 0.3211448        total: 514ms    remaining: 1.14s
31:     learn: 0.3199108        total: 531ms    remaining: 1.13s
32:     learn: 0.3190772        total: 548ms    remaining: 1.11s
33:     learn: 0.3182923        total: 565ms    remaining: 1.1s
34:     learn: 0.3172507        total: 581ms    remaining: 1.08s
35:     learn: 0.3164421        total: 597ms    remaining: 1.06s
36:     learn: 0.3154776        total: 614ms    remaining: 1.04s
37:     learn: 0.3149131        total: 630ms    remaining: 1.03s
38:     learn: 0.3140155        total: 648ms    remaining: 1.01s
39:     learn: 0.3135432        total: 663ms    remaining: 995ms
40:     learn: 0.3131326        total: 679ms    remaining: 976ms
41:     learn: 0.3123354        total: 694ms    remaining: 959ms
42:     learn: 0.3117445        total: 710ms    remaining: 941ms
43:     learn: 0.3112021        total: 725ms    remaining: 922ms
44:     learn: 0.3103399        total: 742ms    remaining: 906ms
45:     learn: 0.3096059        total: 759ms    remaining: 891ms
46:     learn: 0.3091153        total: 774ms    remaining: 873ms
47:     learn: 0.3084194        total: 790ms    remaining: 856ms
48:     learn: 0.3080107        total: 806ms    remaining: 839ms
49:     learn: 0.3070531        total: 823ms    remaining: 823ms
```

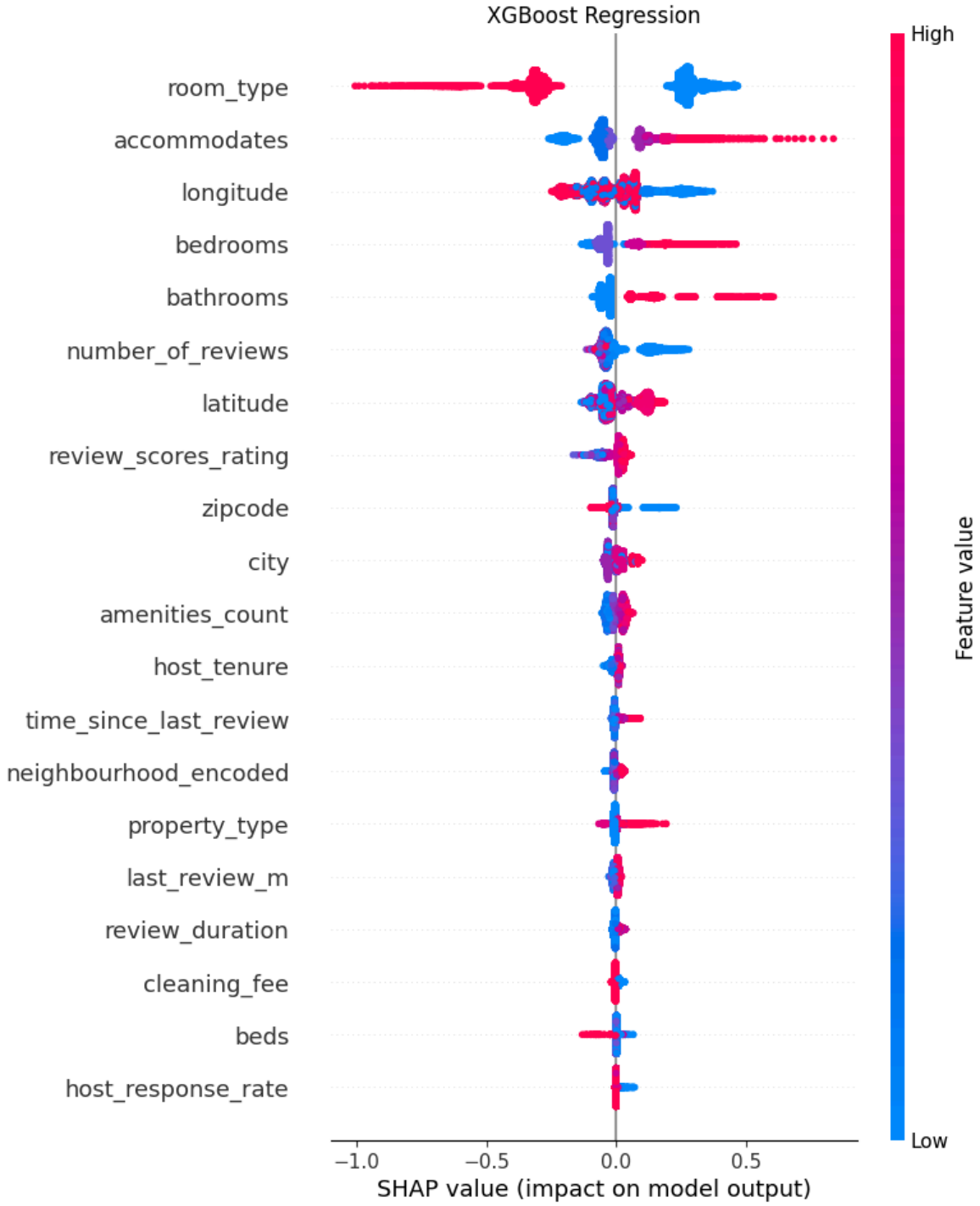
```
50:     learn: 0.3067287        total: 839ms    remaining: 806ms
51:     learn: 0.3061652        total: 859ms    remaining: 793ms
52:     learn: 0.3058116        total: 875ms    remaining: 776ms
53:     learn: 0.3054287        total: 891ms    remaining: 759ms
54:     learn: 0.3050366        total: 906ms    remaining: 742ms
55:     learn: 0.3046971        total: 923ms    remaining: 725ms
56:     learn: 0.3043352        total: 938ms    remaining: 708ms
57:     learn: 0.3039915        total: 954ms    remaining: 691ms
58:     learn: 0.3036730        total: 972ms    remaining: 675ms
59:     learn: 0.3029704        total: 994ms    remaining: 663ms
60:     learn: 0.3027043        total: 1.01s    remaining: 647ms
61:     learn: 0.3024723        total: 1.03s    remaining: 631ms
62:     learn: 0.3020133        total: 1.04s    remaining: 614ms
63:     learn: 0.3016240        total: 1.06s    remaining: 598ms
64:     learn: 0.3013868        total: 1.08s    remaining: 580ms
65:     learn: 0.3010828        total: 1.09s    remaining: 564ms
66:     learn: 0.3008134        total: 1.11s    remaining: 547ms
67:     learn: 0.3004791        total: 1.13s    remaining: 530ms
68:     learn: 0.3002619        total: 1.14s    remaining: 514ms
69:     learn: 0.3000789        total: 1.16s    remaining: 497ms
70:     learn: 0.2997407        total: 1.18s    remaining: 481ms
71:     learn: 0.2991211        total: 1.21s    remaining: 469ms
72:     learn: 0.2987850        total: 1.23s    remaining: 454ms
73:     learn: 0.2985691        total: 1.25s    remaining: 438ms
74:     learn: 0.2983258        total: 1.27s    remaining: 424ms
75:     learn: 0.2981386        total: 1.29s    remaining: 409ms
76:     learn: 0.2978486        total: 1.31s    remaining: 392ms
77:     learn: 0.2976093        total: 1.33s    remaining: 375ms
78:     learn: 0.2974317        total: 1.34s    remaining: 357ms
79:     learn: 0.2971941        total: 1.36s    remaining: 340ms
80:     learn: 0.2969903        total: 1.37s    remaining: 322ms
81:     learn: 0.2968263        total: 1.39s    remaining: 305ms
82:     learn: 0.2965747        total: 1.4s     remaining: 288ms
83:     learn: 0.2960388        total: 1.42s    remaining: 271ms
84:     learn: 0.2959037        total: 1.44s    remaining: 253ms
85:     learn: 0.2957173        total: 1.45s    remaining: 236ms
86:     learn: 0.2956088        total: 1.47s    remaining: 219ms
87:     learn: 0.2951233        total: 1.49s    remaining: 203ms
88:     learn: 0.2950080        total: 1.5s     remaining: 186ms
89:     learn: 0.2948812        total: 1.52s    remaining: 168ms
90:     learn: 0.2946939        total: 1.53s    remaining: 151ms
91:     learn: 0.2945682        total: 1.55s    remaining: 134ms
92:     learn: 0.2942810        total: 1.56s    remaining: 118ms
93:     learn: 0.2940411        total: 1.58s    remaining: 101ms
94:     learn: 0.2936845        total: 1.59s    remaining: 83.9ms
95:     learn: 0.2934826        total: 1.61s    remaining: 67.1ms
96:     learn: 0.2933020        total: 1.63s    remaining: 50.3ms
97:     learn: 0.2931379        total: 1.64s    remaining: 33.5ms
98:     learn: 0.2930214        total: 1.66s    remaining: 16.7ms
99:     learn: 0.2929389        total: 1.67s    remaining: 0us
```

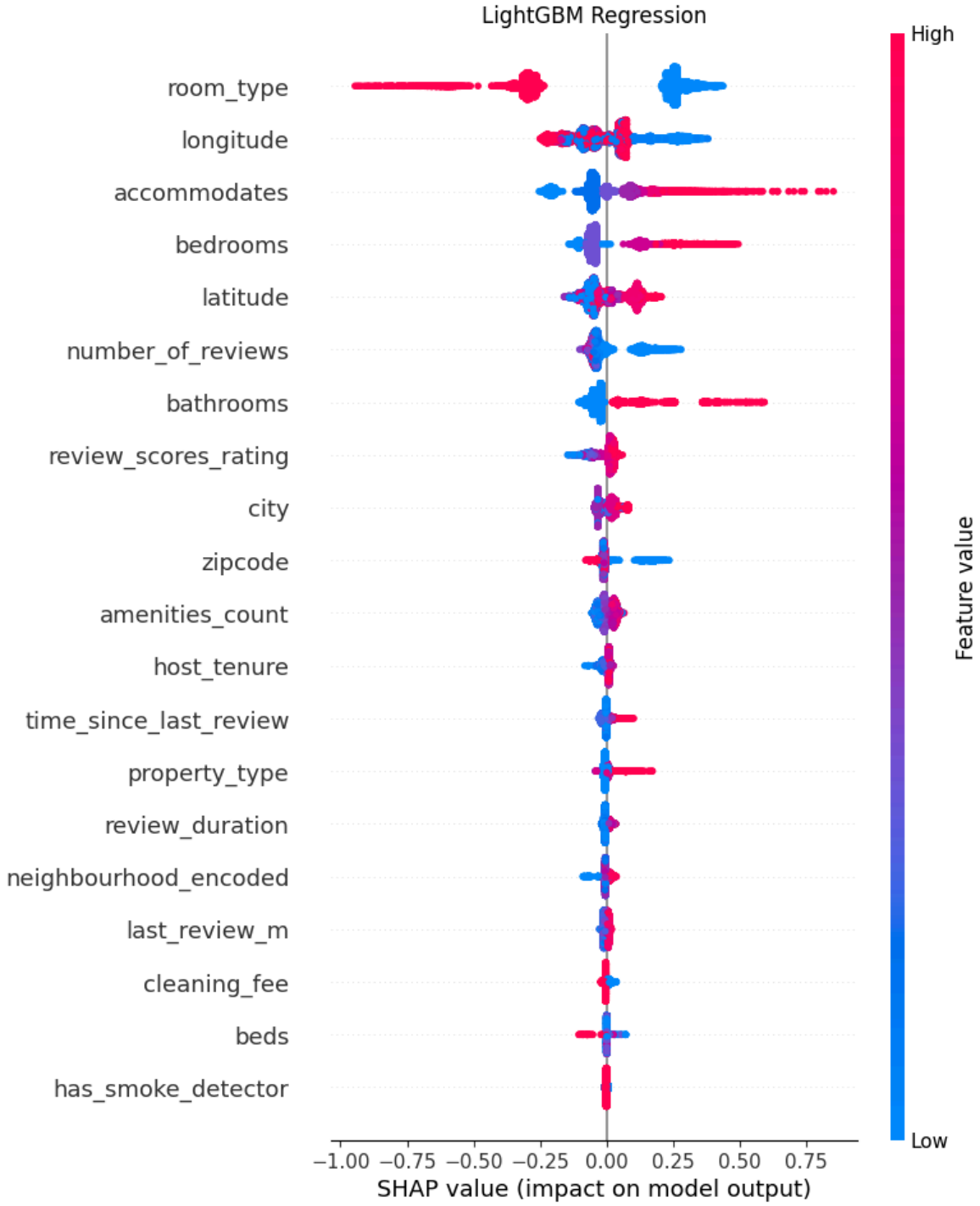
In [ ]:
```python
# Initialize the SHAP explainer for each model
xgb_explainer = shap.Explainer(xgb_reg)
lgb_explainer = shap.Explainer(lgb_reg)
catboost_explainer = shap.Explainer(catboost_reg)

# Compute SHAP values for each model
xgb_shap_values = xgb_explainer.shap_values(X_test)
lgb_shap_values = lgb_explainer.shap_values(X_test)
catboost_shap_values = catboost_explainer.shap_values(X_test)
```

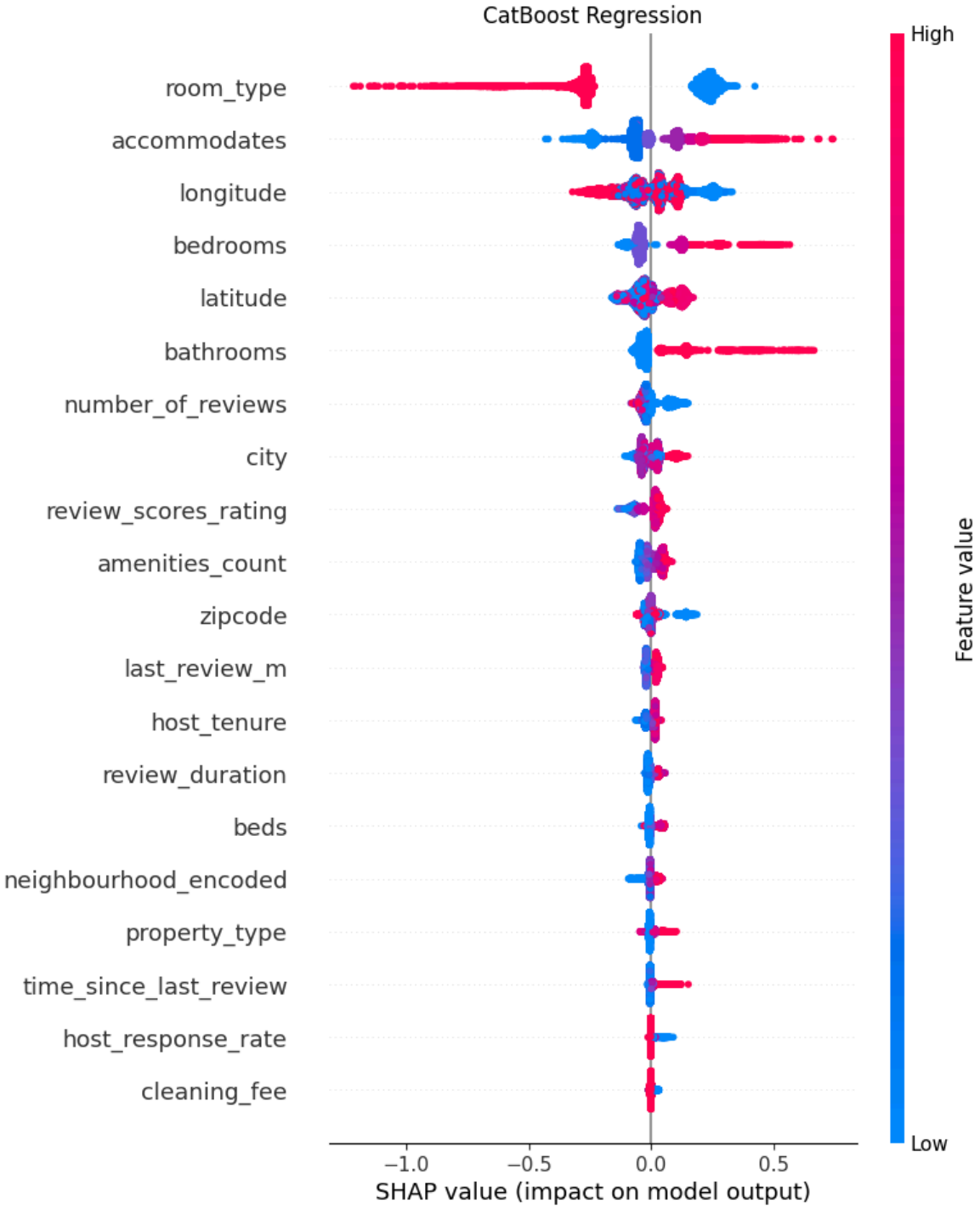
```
# Store SHAP values in the trained_models dictionary
trained_models['XGBoost']['shap_values'] = xgb_shap_values
trained_models['LightGBM']['shap_values'] = lgb_shap_values
trained_models['CatBoost']['shap_values'] = catboost_shap_values

# Save the trained models dictionary to a pickle file
with open('trained_models_with_metrics_and_shap.pkl', 'wb') as f:
    pickle.dump(trained_models, f)
```

In [ ]:

```
# Plot SHAP summary plots for each model
shap.summary_plot(xgb_shap_values, X_test, feature_names=X_test.columns, show=Fa
plt.title('XGBoost Regression')
plt.savefig('xgboost_shap_summary_plot.png')
plt.show()

shap.summary_plot(lgb_shap_values, X_test, feature_names=X_test.columns, show=Fa
plt.title('LightGBM Regression')
plt.savefig('lightgbm_shap_summary_plot.png')
plt.show()

shap.summary_plot(catboost_shap_values, X_test, feature_names=X_test.columns, sh
plt.title('CatBoost Regression')
plt.savefig('catboost_shap_summary_plot.png')
plt.show()
```

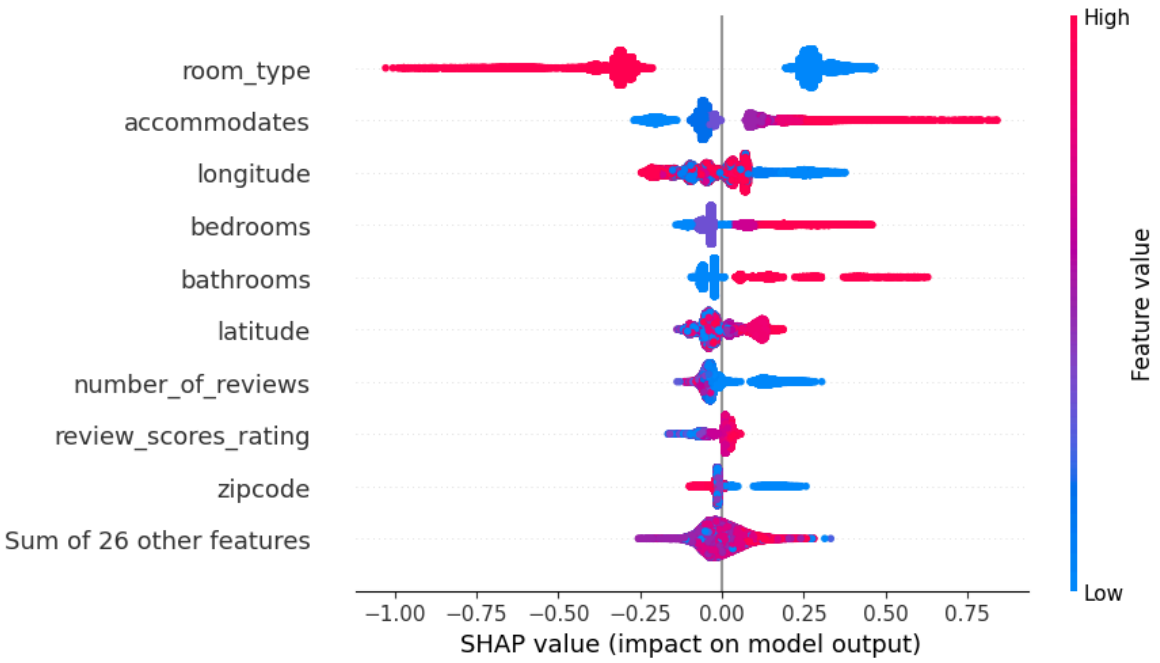

XGBoost Regression
room_type
accommodates
longitude
bedrooms
bathrooms
number_of_reviews
latitude
review_scores_rating
zipcode
city
amenities_count
host_tenure
time_since_last_review
neighbourhood_encoded
property_type
last_review_m
review_duration
cleaning_fee
beds
host_response_rate
−1.0
−0.5
0.0
0.5
SHAP value (impact on model output)
High
Feature value
Low


LightGBM Regression
room_type
longitude
accommodates
bedrooms
latitude
number_of_reviews
bathrooms
review_scores_rating
city
zipcode
amenities_count
host_tenure
time_since_last_review
property_type
review_duration
neighbourhood_encoded
last_review_m
cleaning_fee
beds
has_smoke_detector
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75
SHAP value (impact on model output)
High
Low
Feature value



XGBoost

In [ ]:
```
explainer = shap.TreeExplainer(trained_models["XGBoost"]["model"])
explanation = explainer(X=X_train,y=y_train)

shap.plots.beeswarm(explanation)
```



In [ ]:
```
shap_values = explainer(X_train,y_train)
```

In [ ]:
```
# visualize the first prediction's explanation
shap.plots.force(shap_values[0, ...])
```

Out[ ]:

Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

In [ ]:
```
shap.force_plot(
    explainer.expected_value, shap_values.values[:1000, :], X_train.iloc[:1000,
)
```

Out[ ]:

Visualization omitted, Javascript library not loaded!

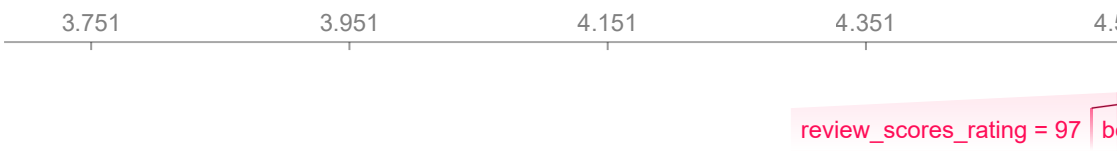
Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

CatBoost

In [ ]:
```
shap.initjs()
```

In [ ]:
```
explainer = shap.TreeExplainer(trained_models["CatBoost"]["model"])
shap_values = explainer(X_train,y_train)

# visualize the first prediction's explanation
shap.plots.force(shap_values[0, ...])
```

Out[ ]:



The above explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in blue.

If we take many explanations such as the one shown above, rotate them 90 degrees, and then stack them horizontally,

In [ ]:
```
shap.force_plot(
    explainer.expected_value, shap_values.values[:1000, :], X_train.iloc[:1000,
)
```

Out[ ]:

To understand how a single feature affects the output of the model, we can plot the SHAP value of that feature vs. the value of the feature for all the examples in a dataset. Since SHAP values represent a feature's responsibility for a change in the model output,

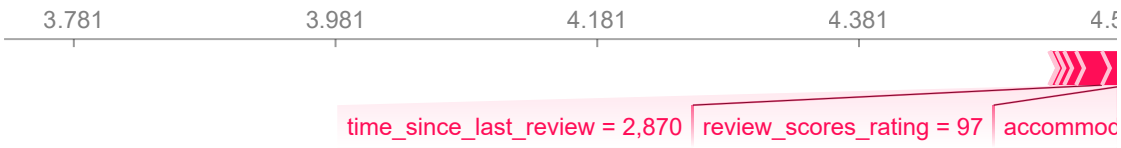
---

LightBoost

In [ ]:
```
explainer = shap.TreeExplainer(trained_models["LightGBM"]["model"])
shap_values = explainer(X_train, y_train)
```

```
# visualize the first prediction's explanation
shap.plots.force(shap_values[0, ...])
```

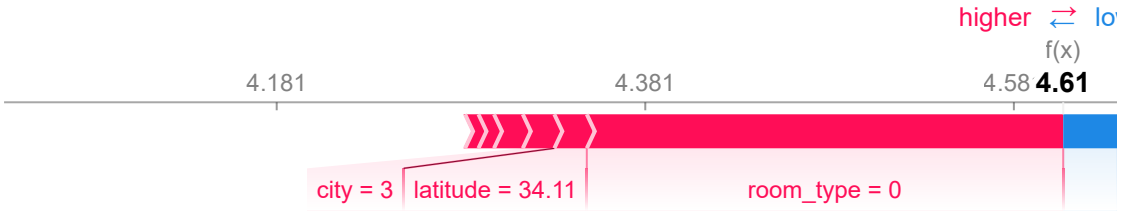
Out[ ]:



In [ ]:

```
shap.force_plot(
    explainer.expected_value, shap_values.values[1, :], X_train.iloc[0, :]
)
```

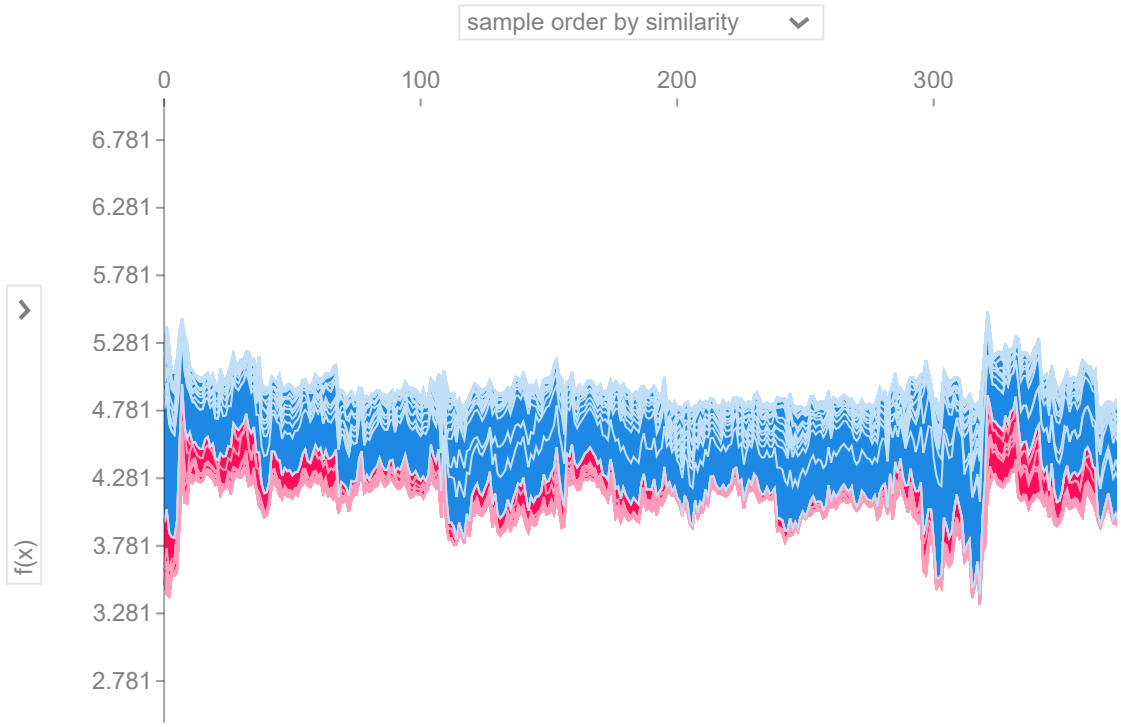
Out[ ]:



In [ ]:

```
shap.force_plot(
    explainer.expected_value, shap_values.values[:1000, :], X_train.iloc[:1000,
)
```

Out[ ]:



Saving the shap values and use them in the front page

Model Card

In [ ]:
```
model_details = """
  # Model Details

  ## Description
  * Model name: XgBoost
  * Model version
  * Model author: Darshan kumar
  * Model type: Regression Task
  * Model architecture: Include any relevant information about algorithms, param
  * Date
  * contact: Darshankumar@gmail.com

  ## Intended use
  * Primary use case: building a predective model for predicting 'log_price' of
"""
```

In [ ]:
```
training_dataset = """
  # Training dataset

  * Training dataset: The training dataset comprises data collected from a varie
  * Training period: The training dataset spans from a recent timeframe, capturi
  * Sub-groups: The dataset includes relevant sub-groups such as demographic inf
  * Limitations: While efforts were made to ensure data accuracy, there are know
  * Pre-processing: Prior to analysis, the data underwent various pre-processing

"""
```

In [ ]:
```
model_evaluation = """
  # Model evaluation

 * Evaluation process: The model was evaluated using standard regression evaluat
 * Evaluation dataset: The evaluation dataset was created by partitioning the or
 * Metrics: Key model quality metrics for regression tasks include mean squared
 * Decision threshold: In regression tasks, there isn't a decision threshold in

"""
```

In [ ]:
```
# !pip install evidently
```

In [ ]:
```
train_data['prediction'] = xgb_reg.predict(X)
```

In [ ]:
```
train_data.columns = cols = ['id',
 'target',
 'accommodates',
 'bathrooms',
 'zipcode',
 'neighbourhood_encoded',
 'host_response_rate',
 'number_of_reviews',
 'review_scores_rating',
 'bedrooms',
 'latitude',
 'longitude',
 'beds',
```

```
 'review_duration',
 'time_since_last_review',
 'host_tenure',
 'average_review_score',
 'amenities_count',
 'has_wireless_internet',
 'has_kitchen',
 'has_heating',
 'has_essentials',
 'has_smoke_detector',
 'first_review_m',
 'host_since_m',
 'last_review_m',
 'property_type',
 'room_type',
 'bed_type',
 'cancellation_policy',
 'city',
 'host_has_profile_pic',
 'host_identity_verified',
 'instant_bookable',
 'cleaning_fee',
 'sentiment_category',
 'prediction']
```

In [ ]:
```
model_card = Report(metrics=[
    Comment(model_details),
    Comment(training_dataset),
    DatasetSummaryMetric(),
    Comment(model_evaluation),
    RegressionQualityMetric(),
    RegressionErrorDistribution(),
])

model_card.run(current_data=train_data[:60000], reference_data=train_data[60000:
model_card
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_regression.py:918: Undef
inedMetricWarning:

R^2 score is not well-defined with less than two samples.

/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_regression.py:918: Undef
inedMetricWarning:

R^2 score is not well-defined with less than two samples.
```

Out[ ]:

Model Details

Description

- Model name: XgBoost
- Model version
- Model author: Darshan kumar
- Model type: Regression Task
- Model architecture: Include any relevant information about algorithms, parameters, etc.
- Date
- contact: Darshankumar@gmail.com

Intended use

- Primary use case: building a predective model for predicting 'log_price' of home

Training dataset

- Training dataset: The training dataset comprises data collected from a variety of sources, primarily focusing on listings from an online accommodation platform. These listings include details such as 'id', 'accommodates', 'bathrooms', 'zipcode', 'neighbourhood_encoded', 'host_response_rate', 'number_of_reviews', 'review_scores_rating', 'bedrooms', 'latitude', 'longitude', 'beds', 'review_duration', 'time_since_last_review', 'host_tenure', 'average_review_score', 'amenities_count', 'has_wireless_internet', 'has_kitchen', 'has_heating', 'has_essentials', 'has_smoke_detector', 'first_review_m', 'host_since_m', 'last_review_m', 'property_type', 'room_type', 'bed_type', 'cancellation_policy', 'city', 'host_has_profile_pic', 'host_identity_verified', 'instant_bookable', 'cleaning_fee', 'sentiment_category'.
- Training period: The training dataset spans from a recent timeframe, capturing data from the past few years up to the present.
- Sub-groups: The dataset includes relevant sub-groups such as demographic information, property characteristics, and host-related attributes.
- Limitations: While efforts were made to ensure data accuracy, there are known limitations inherent to the dataset. These may include missing values, inconsistencies, or biases in the data collection process.
- Pre-processing: Prior to analysis, the data underwent various pre-processing steps including cleaning, normalization, and feature engineering to ensure compatibility and optimize predictive modeling performance.

Dataset Summary

Metric	Current
id column	None
target column	target
prediction column	prediction
date column	None
number of columns	37
number of rows	60000
missing values	0
categorical columns	0
numeric columns	35
text columns	0
datetime columns	0
empty columns	0
constant columns	0
almost constant features	3
duplicated columns	0
almost duplicated features	1

Model evaluation

- Evaluation process: The model was evaluated using standard regression evaluation techniques, such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) score. These metrics provide insight into the model's ability to accurately predict numerical outcomes.
- Evaluation dataset: The evaluation dataset was created by

partitioning the original dataset into separate training and evaluation sets. The evaluation set consists of a subset of the data that was not used during model training, ensuring unbiased assessment of the model's predictive performance on unseen data.

- Metrics: Key model quality metrics for regression tasks include mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) score. These metrics quantify the difference between predicted and actual values, providing an indication of the model's accuracy and goodness of fit.
- Decision threshold: In regression tasks, there isn't a decision threshold in the same sense as classification tasks. Instead, the model's predictions are continuous numerical values, and evaluation metrics are used to assess the closeness of these predictions to the actual target values.

Regression Model Performance. Target: 'target'

Current: Model Quality (+/- std)

ME	MAE	MAPE
0.0 (0.41)	0.3 (0.28)	26361550053491.62 (64572346459877.41)

Reference: Model Quality (+/- std)

ME	MAE	MAPE
-0.01 (0.42)	0.3 (0.29)	6.39 (0.06)

In [ ]:
```
import pandas as pd
import pickle
import xgboost as xgb
import lightgbm as lgb
from catboost import CatBoostRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.feature_selection import RFE, SelectKBest, mutual_info_regression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

In [ ]:
```
df = pd.read_csv("/kaggle/input/model-training/model_training.csv")

senti_dummy = pd.read_csv('/kaggle/input/model-training/model_training_senti.csv
```

In [ ]:
```
def preprocess_text(text):
    """
    Preprocess the text by removing punctuation and stopwords, and converting to

    Parameters:
        text (str): Input text.

    Returns:
        str: Preprocessed text.
    """
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Convert to lowercase
    text = text.lower()
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # Join tokens back into a string
    preprocessed_text = ' '.join(filtered_tokens)
    return preprocessed_text


def clean_zipcode(zipcode):
    if pd.isnull(zipcode):
        return None  # Return None for NaN values
    zipcode_str = str(zipcode)  # Convert to string
    if len(zipcode_str) == 5:
        return int(zipcode_str)  # Convert to integer if length is 5
    else:
        return None  # Return None for other cases


def label_encode_column(df, column_name):
    """
    Perform label encoding on a column in a DataFrame.

    Parameters:
        df (DataFrame): The input DataFrame.
```

```
        column_name (str): The name of the column to be label encoded.

    Returns:
        DataFrame: The input DataFrame with the specified column label encoded.
    """
    # Instantiate LabelEncoder
    label_encoder = LabelEncoder()

    # Fit label encoder and transform the column
    df[column_name + '_encoded'] = label_encoder.fit_transform(df[column_name])

    return df , label_encoder
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [ ]:
```
# Assuming df is your DataFrame
df['zipcode'] = df['zipcode'].apply(clean_zipcode)

# Specify the columns with missing values
columns_with_missing = ['zipcode']

# Create a copy of the DataFrame with only the columns containing missing values
df_missing = df[columns_with_missing].copy()

# Instantiate the KNNImputer with the desired number of neighbors (n_neighbors)
imputer = KNNImputer(n_neighbors=5)

# Fit the imputer on the data with missing values and transform the data
imputed_data = imputer.fit_transform(df_missing)

# Convert the imputed data back to a DataFrame
imputed_df = pd.DataFrame(imputed_data, columns=columns_with_missing)

# Round and convert to integer
imputed_df['zipcode'] = imputed_df['zipcode'].round().astype(int)

# Replace the missing values in the original DataFrame with the imputed values
df[columns_with_missing] = imputed_df
```

In [ ]:
```
df,label_encoder = label_encode_column(df, 'neighbourhood')


null_indices = df[df['neighbourhood'].isnull()].index.tolist()


df.loc[null_indices, 'neighbourhood_encoded'] = float('nan')
```

In [ ]:
```
# Specify the columns with missing values
columns_with_missing = ['neighbourhood_encoded']

# Create a copy of the DataFrame with only the columns containing missing values
df_missing = df[columns_with_missing].copy()

# Instantiate the KNNImputer with the desired number of neighbors (n_neighbors)
imputer = KNNImputer(n_neighbors=5)

# Fit the imputer on the data with missing values and transform the data
```

```
imputed_data = imputer.fit_transform(df_missing)

# Convert the imputed data back to a DataFrame
imputed_df = pd.DataFrame(imputed_data, columns=columns_with_missing)

# Round and convert to integer
imputed_df['neighbourhood_encoded'] = imputed_df['neighbourhood_encoded'].round(

# Replace the missing values in the original DataFrame with the imputed values
df[columns_with_missing] = imputed_df

df['neighbourhood'] = label_encoder.inverse_transform(df['neighbourhood_encoded'
```

In [ ]:
```
# Getting dummy variablaggregatees for categorical columns
dummies_df = pd.get_dummies(df[['property_type', 'room_type', 'bed_type',
                                'cancellation_policy', 'city',
                                'host_has_profile_pic', 'host_identity_verified
                                'instant_bookable']], dtype=int)

# Getting dummy variables for 'cleaning_fee' column
cleaning_fee_df = pd.get_dummies(df['cleaning_fee'], dtype=int, prefix='cleaning
```

In [ ]:
```
# Extracting month from date columns and adding them as new columns
df['first_review_m'] = pd.to_datetime(df['first_review']).dt.month
df['host_since_m'] = pd.to_datetime(df['host_since']).dt.month
df['last_review_m'] = pd.to_datetime(df['last_review']).dt.month
```

In [ ]:
```
# Selecting columns of interest
int_df = df[['id', 'log_price', 'accommodates', 'bathrooms', 'zipcode','neighbou
             'host_response_rate', 'number_of_reviews', 'review_scores_rating',
             'bedrooms', 'latitude', 'longitude','beds', 'review_duration', 'tim
             'host_tenure', 'average_review_score', 'amenities_count',
             'has_wireless_internet', 'has_kitchen', 'has_heating', 'has_essenti
             'has_smoke_detector','first_review_m', 'host_since_m', 'last_review
```

In [ ]:
```
ModelTraining_df = pd.concat([int_df, dummies_df, cleaning_fee_df,senti_dummy],
```

Feature Selection

Report: Feature Selection Before Model Fitting for High-Dimensionality Data

Abstract:

When dealing with datasets containing a large number of features (high dimensionality), selecting the most relevant features before training a machine learning model is generally the recommended approach. This report explores the benefits of feature selection and the drawbacks of fitting all features, along with specific scenarios where fitting all features might be considered.

Benefits of Feature Selection First:

- **Reduced Training Time:** By eliminating irrelevant or redundant features, the model trains on a smaller dataset, leading to faster training and potentially lower computational cost.
- **Improved Model Performance:** Irrelevant features can introduce noise and hinder the model's ability to learn the true relationships between features and the target variable. Feature selection focuses the model on the most informative features, potentially leading to better accuracy and generalizability on unseen data.
- **Reduced Overfitting:** Overfitting occurs when the model learns patterns specific to the training data that don't generalize well. Irrelevant features can contribute to overfitting. Feature selection helps create a model that performs well on both training and testing data.
- **Improved Interpretability:** With fewer features, it's easier to understand how the model makes predictions. This is crucial for debugging issues, explaining the model's behavior, and gaining insights into the data.

Drawbacks of Fitting All Features First:

- **Increased Training Time:** Training on all features takes longer, especially for large datasets.
- **Potential for Overfitting:** Including irrelevant features can lead to overfitting, impacting the model's performance on unseen data.
- **Reduced Interpretability:** With a large number of features, it's difficult to understand which features contribute the most to the model's predictions.

When Might Fitting All Features Be Considered?

- **Limited Data:** If you have a very small dataset, feature selection might remove too much information, potentially harming model performance.
- **Domain Knowledge Lacking:** If you have limited knowledge about the features and their relationship to the target variable, feature selection methods might not be as effective.

Conclusion:

For high-dimensional data (many features), selecting the best features before fitting the model is generally preferred. It leads to faster training, potentially better performance, and often provides a more interpretable model. However, there might be situations where fitting all features is considered due to limited data or lack of domain knowledge.

- Start with filter methods: Due to their efficiency and interpretability, filter methods are a good first approach to identify potentially relevant features. Popular choices include correlation analysis, information gain, and L1 regularization.

- Refine with wrapper or embedded methods: If needed, refine the feature selection using wrapper methods (forward selection, RFE) or embedded methods (LASSO) for potentially better performance. These methods can be computationally expensive, so use them after filter methods have narrowed down the candidate features.

- Consider PCA: Especially for very high-dimensional data, PCA can be a valuable tool to reduce dimensionality while preserving the most informative features.

In [ ]:
```
X = ModelTraining_df.drop('log_price', axis=1)
y = ModelTraining_df['log_price']
```

In [ ]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Perform feature selection
k_best = SelectKBest(mutual_info_regression, k=20)  # Select top 20 features bas
X_train_selected = k_best.fit_transform(X_train, y_train)
X_test_selected = k_best.transform(X_test)

# Get selected feature indices
selected_feature_indices = k_best.get_support(indices=True)

# Get selected feature names
selected_features = X.columns[selected_feature_indices]

print("Selected features:", selected_features)
```

```
Selected features: Index(['accommodates', 'bathrooms', 'zipcode', 'neighbourhood_
encoded',
       'number_of_reviews', 'review_scores_rating', 'bedrooms', 'latitude',
       'longitude', 'beds', 'time_since_last_review', 'host_tenure',
       'average_review_score', 'amenities_count', 'property_type_Apartment',
       'room_type_Entire home/apt', 'room_type_Private room',
       'room_type_Shared room', 'cleaning_fee__False', 'cleaning_fee__True'],
      dtype='object')
```

In [ ]:
```
# # Standardize features
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X)

# # RFE
# estimator = LinearRegression()
# rfe_selector = RFE(estimator, n_features_to_select=15, step=1)  # Select top 5
# X_rfe_selected = rfe_selector.fit_transform(X_scaled, y)
# selected_rfe_features = X.columns[rfe_selector.get_support(indices=True)]

# print("Selected features using RFE:", selected_rfe_features)
```

Wrapper or Embedded Methods (e.g., RFR):

1. **Computationally Expensive:** Wrapper methods involve training the model multiple times with different subsets of features, which can be computationally expensive, especially for large datasets or complex models.

2. **Model Dependent:** Wrapper methods rely on the performance of a specific machine learning model. If the chosen model is not well-suited for the data or the task, the feature selection process may not yield optimal results.

3. **Overfitting Risk:** There's a risk of overfitting, especially with wrapper methods like forward selection, where features are added one by one. This can lead to selecting

features that are only relevant to the training set and do not generalize well to unseen data.

In [ ]:
```
# # Standardize features
# scaler = StandardScaler()
# X_scaled = scaler.fit_transform(X)

# # PCA
# pca = PCA(n_components=15)  # Reduce to 5 principal components
# X_pca = pca.fit_transform(X_scaled)

# print("PCA components:", pca.components_)
```

PCA (Principal Component Analysis):

1. **Loss of Interpretability:** PCA transforms the original features into a new set of orthogonal features (principal components), which may not be directly interpretable in terms of the original features. This loss of interpretability can make it challenging to understand the meaning of each principal component.

2. **Information Loss:** PCA aims to maximize variance in the data, but in doing so, it may discard some information that is not captured by the principal components with the highest variance. This can lead to a loss of information, especially if the lower-variance components contain important features.

3. **Nonlinear Relationships:** PCA assumes linear relationships between variables. If the underlying relationships in the data are nonlinear, PCA may not effectively capture these relationships, leading to suboptimal dimensionality reduction.

Model Training

In [ ]:
```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Initialize regression models
models = {
    'linear_regression': LinearRegression(),
    'ridge_regression': Ridge(),
    'lasso_regression': Lasso(),
    'elastic_net_regression': ElasticNet(),
    'decision_tree_regression': DecisionTreeRegressor(),
    'random_forest_regression': RandomForestRegressor(),
    'gradient_boosting_regression': GradientBoostingRegressor(),
    'xgboost_regression': xgb.XGBRegressor(),
    'lightgbm_regression': lgb.LGBMRegressor(),
    'catboost_regression': CatBoostRegressor(silent=True)
}

# Train each model
for name, model in models.items():
```

```
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Calculate Mean Squared Error (MSE)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    # Print the evaluation metric
    print(f"{name} MSE: {mse}")
    print(f"{name} MSE: {mae}")

    print("-"*30)


# Save the models dictionary to a pickle file
with open('trained_models.pkl', 'wb') as f:
    pickle.dump(models, f)
```

```
linear_regression MSE: 0.24105005154305392
linear_regression MSE: 0.36903727653140234
------------------------------
ridge_regression MSE: 0.24105137756307524
ridge_regression MSE: 0.369037250018861
------------------------------
lasso_regression MSE: 0.5072913482976185
lasso_regression MSE: 0.5559326285430588
------------------------------
elastic_net_regression MSE: 0.4017577169551864
elastic_net_regression MSE: 0.4900197170786799
------------------------------
decision_tree_regression MSE: 0.31182273901256996
decision_tree_regression MSE: 0.4024522341569183
------------------------------
random_forest_regression MSE: 0.15521153081387837
random_forest_regression MSE: 0.28350343755029006
------------------------------
gradient_boosting_regression MSE: 0.1727719373971861
gradient_boosting_regression MSE: 0.3036576074580185
------------------------------
xgboost_regression MSE: 0.15205490511774514
xgboost_regression MSE: 0.2824325192706379
------------------------------
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.008831 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 2146
[LightGBM] [Info] Number of data points in the train set: 59288, number of used f
eatures: 20
[LightGBM] [Info] Start training from score 4.780538
lightgbm_regression MSE: 0.1529626331375161
lightgbm_regression MSE: 0.28420173675727073
------------------------------
catboost_regression MSE: 0.1477884671345181
catboost_regression MSE: 0.2784356276589088
------------------------------
```

In [ ]:

```
# Load the trained models from the pickle file
with open('trained_models.pkl', 'rb') as f:
    models = pickle.load(f)
```

```
# Initialize dictionaries to store evaluation metrics for each model
evaluation_metrics = {
    'Model': [],
    'MSE': [],
    'MAE': [],
    'R2': []
}

# Evaluate the performance of each model
for name, model in models.items():
    # Predict on the test data
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Store evaluation metrics in the dictionary
    evaluation_metrics['Model'].append(name)
    evaluation_metrics['MSE'].append(mse)
    evaluation_metrics['MAE'].append(mae)
    evaluation_metrics['R2'].append(r2)
```

SHAP

In [ ]:
```
import shap

# Initialize the SHAP explainer for each model
xgb_explainer = shap.Explainer(xgb_reg)
lgb_explainer = shap.Explainer(lgb_reg)
catboost_explainer = shap.Explainer(catboost_reg)

# Compute SHAP values for each model
xgb_shap_values = xgb_explainer.shap_values(X_test)
lgb_shap_values = lgb_explainer.shap_values(X_test)
catboost_shap_values = catboost_explainer.shap_values(X_test)

# Store SHAP values in the trained_models dictionary
trained_models['XGBoost']['shap_values'] = xgb_shap_values
trained_models['LightGBM']['shap_values'] = lgb_shap_values
trained_models['CatBoost']['shap_values'] = catboost_shap_values

# Save the trained models dictionary to a pickle file
with open('trained_models_with_metrics_and_shap.pkl', 'wb') as f:
    pickle.dump(trained_models, f)
```

In [ ]:
```
# Initialize XGBoost Regression with hyperparameters
xgb_reg = xgb.XGBRegressor(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='reg:squarederror',
    random_state=42
```

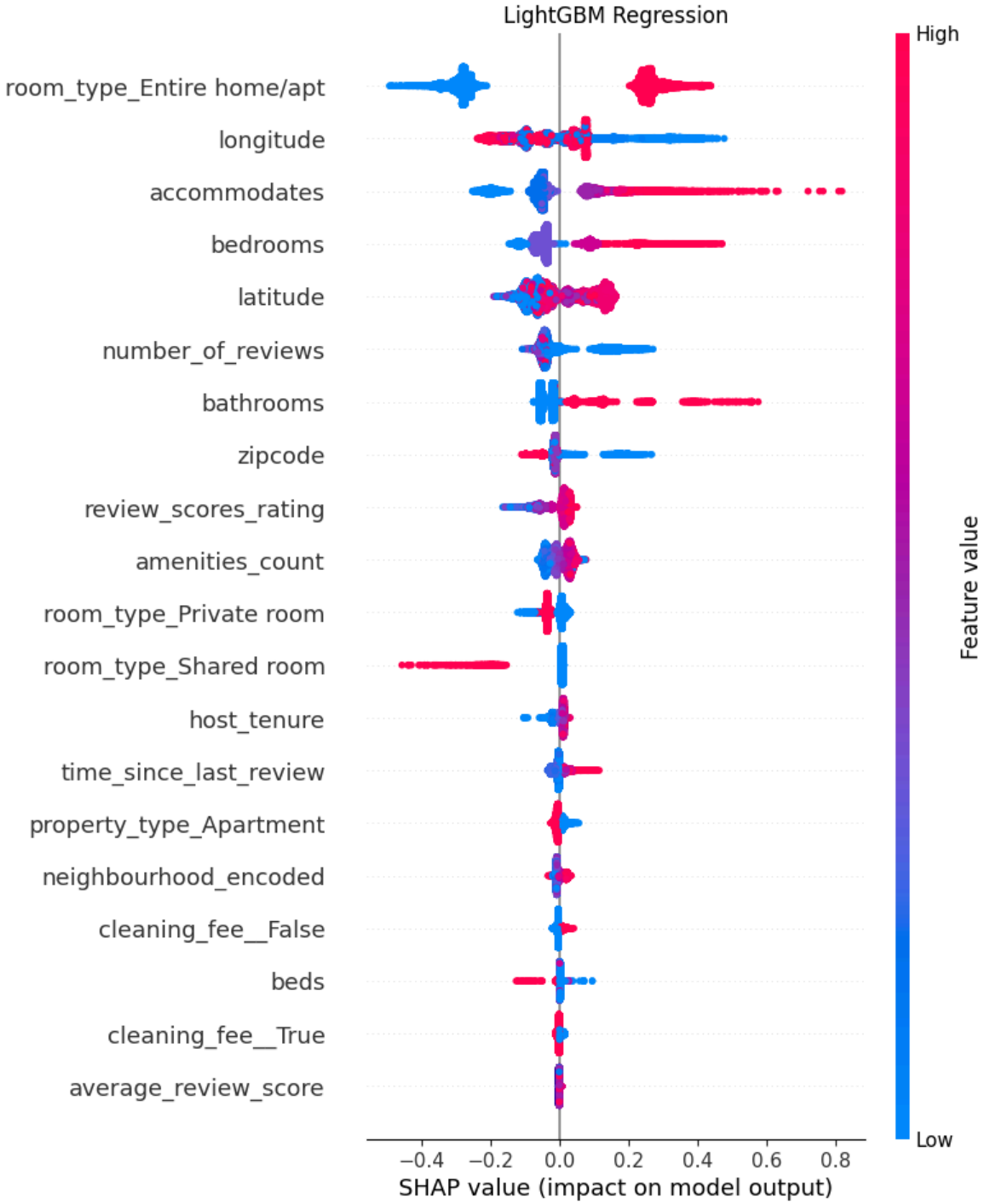
```
)

# Initialize LightGBM Regression with hyperparameters
lgb_reg = lgb.LGBMRegressor(
    n_estimators=100,
    max_depth=3,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    objective='regression',
    random_state=42
)

# Initialize CatBoost Regression with hyperparameters
catboost_reg = CatBoostRegressor(
    n_estimators=100,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bylevel=0.8,
    objective='MAE',
    random_seed=42
)

# Train XGBoost Regression
xgb_reg.fit(X_train, y_train)

# Train LightGBM Regression
lgb_reg.fit(X_train, y_train)

# Train CatBoost Regression
catboost_reg.fit(X_train, y_train)

# Initialize a dictionary to store models, evaluation metrics, and feature impor
trained_models = {
    'XGBoost': {'model': xgb_reg, 'evaluation_metrics': {}, 'feature_importance'
    'LightGBM': {'model': lgb_reg, 'evaluation_metrics': {}, 'feature_importance
    'CatBoost': {'model': catboost_reg, 'evaluation_metrics': {}, 'feature_impor
}

# Evaluate each model and store evaluation metrics
for model_name, model_data in trained_models.items():
    model = model_data['model']
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    model_data['evaluation_metrics'] = {'MSE': mse, 'MAE': mae, 'R2': r2}

# Store feature importance for XGBoost and LightGBM models
trained_models['XGBoost']['feature_importance'] = xgb_reg.feature_importances_
trained_models['LightGBM']['feature_importance'] = lgb_reg.feature_importances_

# CatBoost provides feature importance as part of its model object
trained_models['CatBoost']['feature_importance'] = catboost_reg.get_feature_impo

# Save the trained models dictionary to a pickle file
with open('trained_models_with_metrics.pkl', 'wb') as f:
    pickle.dump(trained_models, f)
```

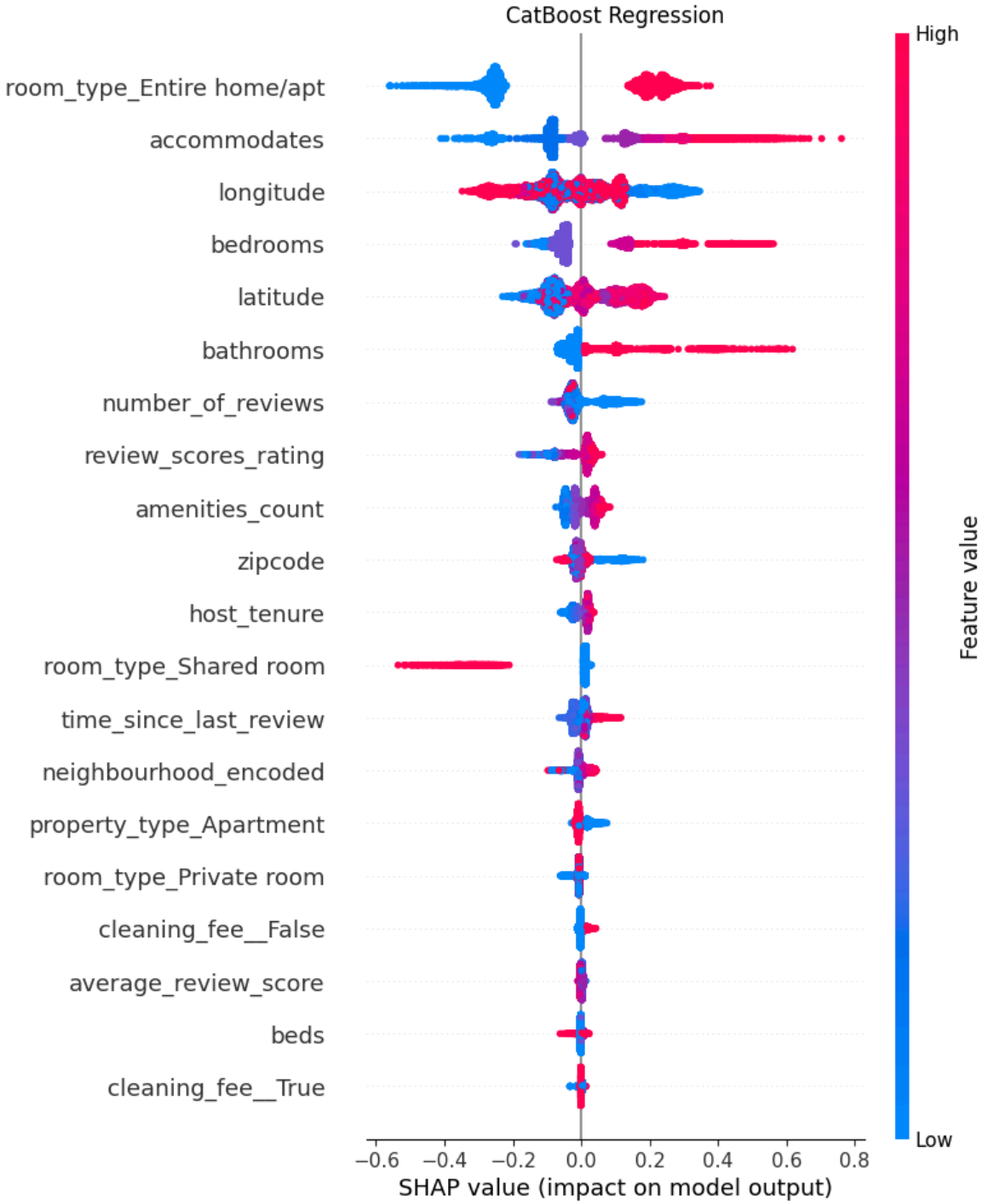
```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.015452 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2146
[LightGBM] [Info] Number of data points in the train set: 59288, number of used f
eatures: 20
[LightGBM] [Info] Start training from score 4.780538
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
0:      learn: 0.5292717        total: 13ms     remaining: 1.28s
1:      learn: 0.5020531        total: 25.1ms   remaining: 1.23s
2:      learn: 0.4780849        total: 37.3ms   remaining: 1.21s
3:      learn: 0.4580973        total: 49.2ms   remaining: 1.18s
4:      learn: 0.4407128        total: 61.4ms   remaining: 1.17s
5:      learn: 0.4262902        total: 73.5ms   remaining: 1.15s
6:      learn: 0.4133027        total: 84.7ms   remaining: 1.13s
7:      learn: 0.4029372        total: 96.6ms   remaining: 1.11s
8:      learn: 0.3933309        total: 108ms    remaining: 1.09s
9:      learn: 0.3844541        total: 121ms    remaining: 1.09s
10:     learn: 0.3775177        total: 133ms    remaining: 1.07s
11:     learn: 0.3705298        total: 145ms    remaining: 1.06s
12:     learn: 0.3654351        total: 157ms    remaining: 1.05s
13:     learn: 0.3605758        total: 169ms    remaining: 1.04s
14:     learn: 0.3565841        total: 181ms    remaining: 1.02s
15:     learn: 0.3528353        total: 193ms    remaining: 1.01s
16:     learn: 0.3492485        total: 203ms    remaining: 993ms
17:     learn: 0.3461501        total: 215ms    remaining: 978ms
18:     learn: 0.3434251        total: 226ms    remaining: 964ms
19:     learn: 0.3408800        total: 238ms    remaining: 951ms
20:     learn: 0.3388004        total: 249ms    remaining: 937ms
21:     learn: 0.3368661        total: 260ms    remaining: 923ms
22:     learn: 0.3348756        total: 272ms    remaining: 912ms
23:     learn: 0.3333416        total: 283ms    remaining: 897ms
24:     learn: 0.3311657        total: 294ms    remaining: 882ms
25:     learn: 0.3298528        total: 306ms    remaining: 872ms
26:     learn: 0.3286015        total: 319ms    remaining: 863ms
27:     learn: 0.3273916        total: 332ms    remaining: 855ms
28:     learn: 0.3262418        total: 343ms    remaining: 840ms
29:     learn: 0.3246378        total: 355ms    remaining: 828ms
30:     learn: 0.3235921        total: 366ms    remaining: 815ms
31:     learn: 0.3222863        total: 378ms    remaining: 803ms
32:     learn: 0.3211385        total: 389ms    remaining: 789ms
33:     learn: 0.3199293        total: 400ms    remaining: 777ms
34:     learn: 0.3190597        total: 411ms    remaining: 763ms
35:     learn: 0.3183955        total: 423ms    remaining: 751ms
36:     learn: 0.3176559        total: 434ms    remaining: 738ms
37:     learn: 0.3169993        total: 445ms    remaining: 725ms
38:     learn: 0.3164459        total: 456ms    remaining: 714ms
39:     learn: 0.3155340        total: 469ms    remaining: 704ms
40:     learn: 0.3145677        total: 481ms    remaining: 692ms
41:     learn: 0.3141420        total: 492ms    remaining: 680ms
42:     learn: 0.3134567        total: 503ms    remaining: 667ms
43:     learn: 0.3128501        total: 515ms    remaining: 655ms
44:     learn: 0.3121332        total: 528ms    remaining: 645ms
45:     learn: 0.3115547        total: 539ms    remaining: 633ms
46:     learn: 0.3106945        total: 551ms    remaining: 621ms
47:     learn: 0.3101771        total: 562ms    remaining: 609ms
48:     learn: 0.3097459        total: 574ms    remaining: 598ms
49:     learn: 0.3090876        total: 586ms    remaining: 586ms
```

```
50:     learn: 0.3088007        total: 597ms    remaining: 574ms
51:     learn: 0.3081422        total: 609ms    remaining: 562ms
52:     learn: 0.3076767        total: 620ms    remaining: 550ms
53:     learn: 0.3073437        total: 631ms    remaining: 538ms
54:     learn: 0.3069962        total: 642ms    remaining: 526ms
55:     learn: 0.3065169        total: 655ms    remaining: 514ms
56:     learn: 0.3062033        total: 667ms    remaining: 503ms
57:     learn: 0.3058495        total: 679ms    remaining: 492ms
58:     learn: 0.3054431        total: 691ms    remaining: 480ms
59:     learn: 0.3051142        total: 701ms    remaining: 467ms
60:     learn: 0.3046800        total: 713ms    remaining: 456ms
61:     learn: 0.3042588        total: 724ms    remaining: 444ms
62:     learn: 0.3040620        total: 737ms    remaining: 433ms
63:     learn: 0.3038499        total: 748ms    remaining: 421ms
64:     learn: 0.3035325        total: 760ms    remaining: 409ms
65:     learn: 0.3030499        total: 772ms    remaining: 398ms
66:     learn: 0.3028277        total: 784ms    remaining: 386ms
67:     learn: 0.3024201        total: 795ms    remaining: 374ms
68:     learn: 0.3020773        total: 806ms    remaining: 362ms
69:     learn: 0.3018288        total: 817ms    remaining: 350ms
70:     learn: 0.3016000        total: 829ms    remaining: 338ms
71:     learn: 0.3011825        total: 840ms    remaining: 327ms
72:     learn: 0.3008193        total: 854ms    remaining: 316ms
73:     learn: 0.3005548        total: 870ms    remaining: 306ms
74:     learn: 0.3002296        total: 882ms    remaining: 294ms
75:     learn: 0.3001088        total: 893ms    remaining: 282ms
76:     learn: 0.2997899        total: 904ms    remaining: 270ms
77:     learn: 0.2994804        total: 916ms    remaining: 258ms
78:     learn: 0.2992194        total: 927ms    remaining: 247ms
79:     learn: 0.2990569        total: 938ms    remaining: 235ms
80:     learn: 0.2988599        total: 950ms    remaining: 223ms
81:     learn: 0.2985505        total: 962ms    remaining: 211ms
82:     learn: 0.2983718        total: 978ms    remaining: 200ms
83:     learn: 0.2980636        total: 998ms    remaining: 190ms
84:     learn: 0.2978309        total: 1.01s    remaining: 179ms
85:     learn: 0.2974944        total: 1.03s    remaining: 167ms
86:     learn: 0.2973512        total: 1.04s    remaining: 156ms
87:     learn: 0.2971847        total: 1.06s    remaining: 144ms
88:     learn: 0.2970160        total: 1.07s    remaining: 132ms
89:     learn: 0.2968159        total: 1.08s    remaining: 120ms
90:     learn: 0.2965025        total: 1.09s    remaining: 108ms
91:     learn: 0.2963253        total: 1.1s     remaining: 95.7ms
92:     learn: 0.2958876        total: 1.11s    remaining: 83.8ms
93:     learn: 0.2957152        total: 1.12s    remaining: 71.8ms
94:     learn: 0.2954276        total: 1.14s    remaining: 59.9ms
95:     learn: 0.2952674        total: 1.15s    remaining: 47.9ms
96:     learn: 0.2949206        total: 1.16s    remaining: 35.9ms
97:     learn: 0.2947312        total: 1.17s    remaining: 23.9ms
98:     learn: 0.2946000        total: 1.18s    remaining: 12ms
99:     learn: 0.2943565        total: 1.19s    remaining: 0us
```

In [ ]:
```
# Plot SHAP summary plots for each model
shap.summary_plot(xgb_shap_values, X_test, feature_names=X_test.columns, show=Fa
plt.title('XGBoost Regression')
plt.savefig('xgboost_shap_summary_plot.png')
plt.show()

shap.summary_plot(lgb_shap_values, X_test, feature_names=X_test.columns, show=Fa
plt.title('LightGBM Regression')
plt.savefig('lightgbm_shap_summary_plot.png')
```

```
plt.show()

shap.summary_plot(catboost_shap_values, X_test, feature_names=X_test.columns, sh
plt.title('CatBoost Regression')
plt.savefig('catboost_shap_summary_plot.png')
plt.show()
```


LightGBM Regression
room_type_Entire home/apt
longitude
accommodates
bedrooms
latitude
number_of_reviews
bathrooms
zipcode
review_scores_rating
amenities_count
room_type_Private room
room_type_Shared room
host_tenure
time_since_last_review
property_type_Apartment
neighbourhood_encoded
cleaning_fee__False
beds
cleaning_fee__True
average_review_score
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
SHAP value (impact on model output)
High
Feature value
Low



In [ ]:
```
!jupyter nbconvert --to html HomeStay_OHE_MT.ipynb
```

In [ ]:
```
```