Queen Mary
University of London

BUSM131 – Masterclass in Business Analytics - 2023/24

Dr. Guven Demirel

Airbnb Price Prediction

Project Report

Name: Mr. Vatsal Vipul Doshi

Student ID: 230504415

Table of Contents

Abstract

The goal of this project is to help Airbnb hosts precisely price their listings for optimum profit without the need for manual research, by using machine learning models trained on existing listing prices to provide precise pricing forecasts. The dataset contains 74,111 rows and 29 columns with attributes and pricing information for US Airbnb listings.

We focused on a subset of New York City listings to train and test our machine learning models. We used several techniques, which incorporates Linear Regression, Ridge Regression, Lasso Regression, Random Forest, XG Boost, Support Vector Regression, and Artificial Neural Networks. Furthermore, we've utilized feature technique that involve Principal Component Analysis, imputation, categorical encoding, feature splitting, variable transformation, as well as feature creation. For example, we used the Open-Source Routing Machine to determine the average and minimum distances of listings from three New York City airports, as well as categorise each listing by borough.

Our study proved that Support Vector Regression and XG-Boost outperformed other machine learning models. Interestingly, although being the most complicated, Artificial Neural Networks (ANN) failed to perform better than the others, implying a decrease in degree of freedom, which could lead to overfitting and compromise the model's robustness.

Introduction

Within the contemporary sharing economy, websites such as Airbnb have completely transformed the travel and hospitality sector by providing both homeowners and tourists with a way to make money off of their properties. One of Airbnb's key techniques is dynamic pricing, which uses data to modify listing costs depending on to market demand, seasonality, and other factors. Understanding the effectiveness and significance of Airbnb's Dynamic Price Prediction Model is vital for both hosts and guests, as it directly effects pricing strategies, revenue creation, and consumer behaviour on the platform.

Literature Review

1. *J. Sims, N. Ameen & R. Bauer (2019)* demonstrates how applied revenue management and benchmarking techniques can improve a host's booking performance. The motivation and price experience of professional and non-professional hosts differ. Benchmarking specialists recognise that market research is an essential component of running a successful business. The study used machine learning strategies to analyse price dynamics and booking patterns, revealing information on host behaviour and consumer response.
2. *Y. Wang (2023)* examines dynamic pricing on Airbnb utilising machine learning techniques such as linear regression, random forest, K-nearest neighbour, AdaBoost Classifier, and Naïve Bayes. They use data from Airbnb rentals in New York from 2022 to compare the performance of various models using R-squared (R2) and root mean square error (RMSE). The results show that the random forest model surpasses the others, with an R2 of 0.997 and an RMSE of 0.038, indicating that it is suitable for establishing ideal Airbnb

prices. The Naïve Bayes classifier outperforms the random forest model with a lower RMSE.

3. *L. Cao (2023)* shows a thorough method to data pretreatment, exploration, and machine learning modelling for predicting Airbnb prices. They effectively deal with missing values, outliers, and categorical variables while also conducting extensive data exploration using statistical analysis and visualisation approaches. By training and evaluating three different regression algorithms and optimising their hyperparameters, the author determines the best performing model for predicting Airbnb prices using metrics such as mean squared error and R-squared, thereby providing a solid framework for predictive modelling in the hospitality industry.

Summary of Report Structure

This research examines Airbnb's Dynamic Price Prediction Model, attempting to analyse its performance and significance for hosts and guests. Following this introduction, the following sections will look at the business problem, data sources, exploratory data analysis (EDA), methodologies, results analysis, and concluding discussions. This research attempts to provide a thorough knowledge of the changing pricing structure inside the Airbnb ecosystem by combining academic insights with empirical analysis.

Business Problem

Within the constantly changing hospitality sector, Airbnb has become a disruptive force that is completely changing how people travel and stay. The key to Airbnb's success is its implementation of dynamic pricing, which modifies listing prices in real-time in response to a range of variables like seasonality, market demand, and property features. While dynamic pricing has the potential to maximise revenue for hosts while providing competitive prices for visitors, its efficacy is dependent on the preciseness and efficacy of Airbnb's Dynamic Price Prediction Model.

The foundation of Airbnb's pricing approach is its Dynamic Price Prediction Model, which affects both the affordability of lodging for visitors and the profitability of hosts. The programme attempts to forecast ideal listing prices which satisfy supply and demand dynamics by examining a tonne of data, including previous booking patterns, rival pricing, and market trends. But for hosts and Airbnb as a platform, the model's ability to predict pricing properly and maximise revenue is still a major source of concern.

The main issue facing the company is how useful and successful Airbnb's Dynamic Price Prediction Model is. Using this technique, hosts can maximise their earnings by setting competitive prices that draw in guests. On the other hand, inaccurate pricing projections may result in lost income for hosts or turn off potential customers with too high prices. Additionally, visitors might think that the platform's dynamic pricing changes are unfair or inconsistent, which would erode their faith in and devotion to it.

Decisions supported by Analytics

Analytics is critical for directing strategic decisions regarding Airbnb's pricing strategy and competitive positioning in the hotel industry. To begin, it is critical to assess

Airbnb's Dynamic Price Prediction Model's ability to accurately estimate listing prices and maximise revenue for hosts. Airbnb may evaluate the model's performance parameters, including accuracy, precision, and income production potential, by utilising advanced analytics approaches such as predictive modelling and statistical analysis. Furthermore, understanding the influence of dynamic pricing changes on booking patterns and customer satisfaction requires researching consumer behaviour in response to them. Airbnb can gain a better understanding of how guests perceive and respond to pricing changes by using data-driven insights from consumer segmentation, sentiment analysis, and behavioural analytics, which can then be used to inform pricing strategies to improve user experience and satisfaction levels. Furthermore, competitive positioning analysis allows Airbnb to compare its pricing tactics to competitors and alternative lodging options on the market. By evaluating price dynamics, market share trends, and value propositions, Airbnb may find areas for differentiation and strategic positioning in order to maintain a competitive edge and effectively gain market share. In summary, analytics is a strategic enabler for Airbnb, allowing data-driven decision-making to optimise pricing strategies, improve user satisfaction, and increase competitive positioning in the dynamic hospitality sector environment.

Organizations Facing Similar Problems

Comparable issues with dynamic pricing and revenue optimisation arise for companies like Uber, Amazon, and Booking.com in the sharing economy and e-commerce space. With algorithms that modify charges in response to changing factors like time of day, weather, and traffic congestion, Uber's use of surge pricing is an example of how dynamic pricing is applied to balance supply and demand for rides. Likewise, Amazon uses dynamic pricing algorithms to quickly adjust product prices, maintaining competitiveness in the face of shifting demand, rival pricing schemes, and stock levels.

Adoption of Price Prediction in Market

Additionally, in order to maximise income production, Booking.com uses machine learning algorithms to constantly adjust hotel prices in response to demand estimates, market trends, and consumer preferences. These businesses demonstrate how sophisticated analytics and machine learning are combined to manage the challenges of dynamic pricing. This provides Airbnb with insightful information that it can use to improve its price policies and efforts to maximise revenue in the sharing economy. These companies make investments in cutting-edge analytics and machine learning to create strong pricing plans that improve revenue generation and customer satisfaction. Their insights are invaluable for tackling related issues within Airbnb's ecosystem.

Data, EDA, and Methods

Dataset

The "Airbnb Price Prediction" dataset by 'STEVEZHENG' from Kaggle.com has 29 variables and 74,112 entries and 67,122 unique values of diverse data kinds, including text, date/time, numeric, and category features. It covers important parameters such as accommodation capacity and property qualities, as well as host-specific characteristics and cancellation procedures. Furthermore, textual data provides information about property amenities and descriptions. The URL and postal code variables provide more information on property photos and location. With this wide range of data, the dataset allows for a full examination of the factors that influence Airbnb listing prices, as well as predictive modelling to optimise pricing strategies and improve user experiences on the platform.

Target Variables

Our target variables are Room type, Cleaning_fee, Bedrooms, Bathrooms, Accommodates, etc.

Missing Values

These are the following features in our dataset along with the count of missing values for each feature:

- Bathrooms: 200
- First_reviews: 15,864
- Host_has_profile_pic: 188
- Host_identity_verified: 188
- Host_response_rate: 18,299
- Host_since: 188
- Name: 23
- Neighbourhood: 6,872
- Review_scores_rating: 16,722
- Thumbnail_url: 8,216
- Zipcode: 966
- Bedroom: 91
- Beds: 31

These missing variables can have an impact on the quality and reliability of our study, thus appropriate procedures like imputation or elimination may be required to deal with them effectively. We removed variables like first_review, host_response_rate, review_scores_rating, last_review, and thumbnail_url because they are not used in our model. To fill the missing numbers, we calculated median values using the .median() method and filled them with the .fillna() method.

Exploratory Data Analysis (EDA)

1. Top 10 Most Expensive & Cheapest Neighbourhood in NYC:

The charts compare hotel prices in several neighbourhoods of New York City, with the x-axis representing the logarithm of prices and the y-axis representing the neighbourhoods. The ten lowest and most costly neighbourhoods are shown separately. The costliest neighbourhood, Mill Basin, has a logarithmic room price of around 6.2, while the cheapest, Morris Park, has a value of about 3.7. This implies that a room in Mill Basin costs nearly twice as much as one in Morris Park. The wide variance in costs between neighbourhoods emphasises the importance of neighbourhood as a factor in price prediction, which may cause projections to bias higher in more expensive locations.

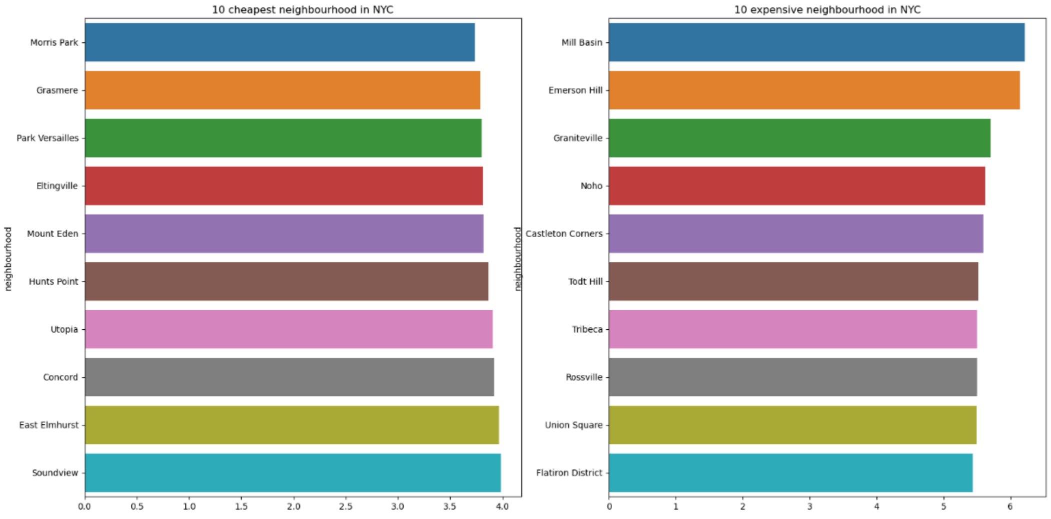


Fig.1 Top 10 cheapest & expensive neighbourhoods in NYC

2. Top 10 Most Expensive & Cheapest Boroughs in NYC:

The accompanying bar charts compare accommodation pricing across different boroughs in New York City, revealing significant patterns. In the "10 cheapest Borough in NYC" ranking, the Bronx is the most affordable borough, followed by Queens and Staten Island. In contrast, the "10 Expensive Boroughs in NYC" list shows Manhattan as the costliest borough, followed by Brooklyn and New Jersey. These data highlight significant pricing variations between NYC boroughs, with Manhattan being the most expensive and the Bronx being the most affordable. Such insights are critical for predictive modelling, especially in rental price estimates, because neighbourhoods with higher prices might impact upward expectations.

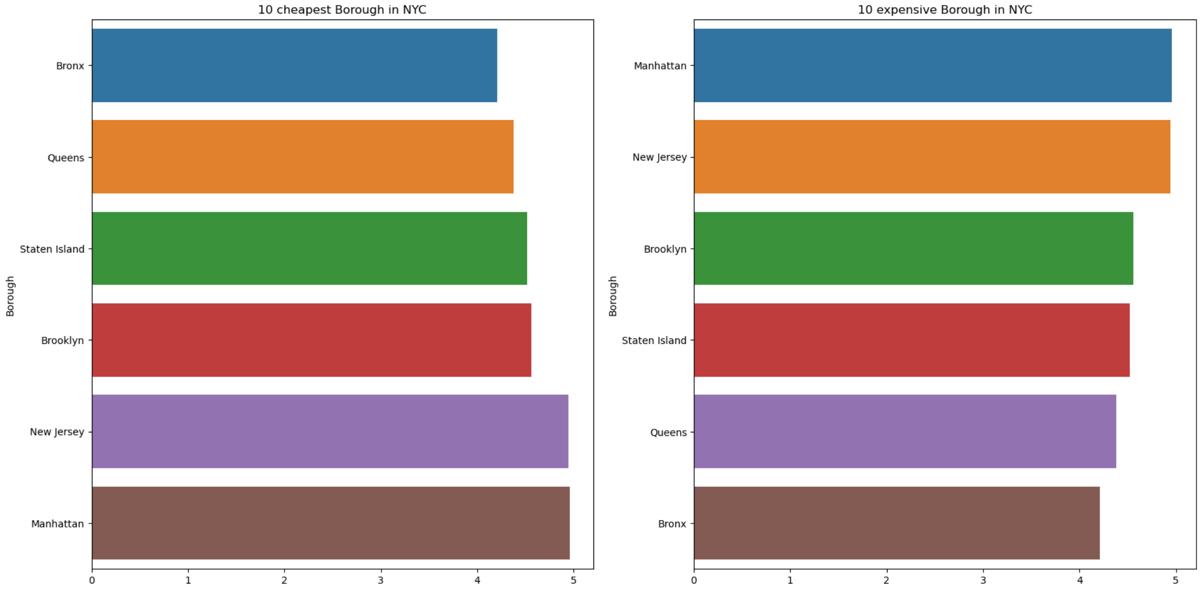


Fig. 2 Top 10 cheapest & expensive boroughs in NYC

3. Types of rooms and its count:

The graph shows the distribution of room kinds in an unidentified area. The most common type is entire homes/apartments, which number around 16,000. Private rooms are in the middle, with an overall number of around 15,000, however shared rooms are the least prominent, with about 1,000.

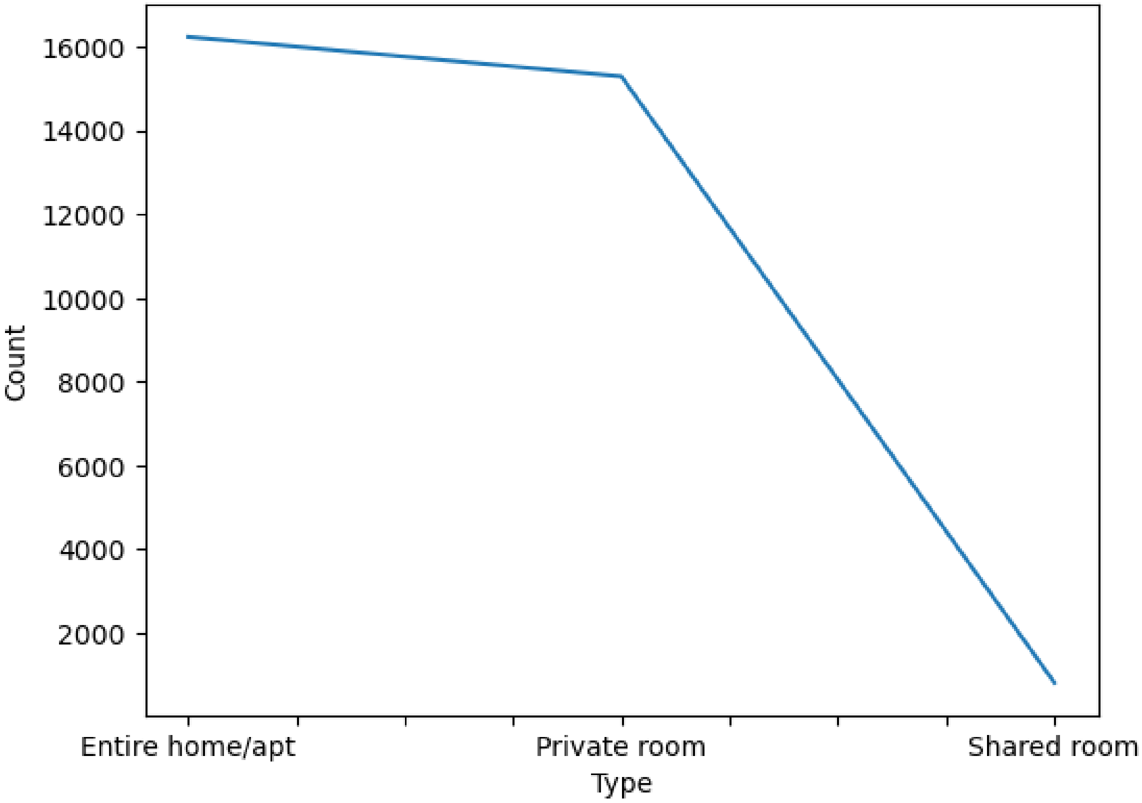


Fig. 3 Types of rooms and its count

4. Density and distribution of prices for room type:

The box plot depicts the price distribution across various room types on a logarithmic scale. It demonstrates that full homes/apartments are typically the most expensive, followed by private and shared rooms. Furthermore, prices vary the most for full homes/apartments, showing a wider variety of pricing than for other room kinds. Shared rooms, on the other hand, have the least variability, indicating a narrower pricing range. This means that, while complete homes/apartments may have both very expensive and more economical options, shared rooms typically have a more constant pricing structure.

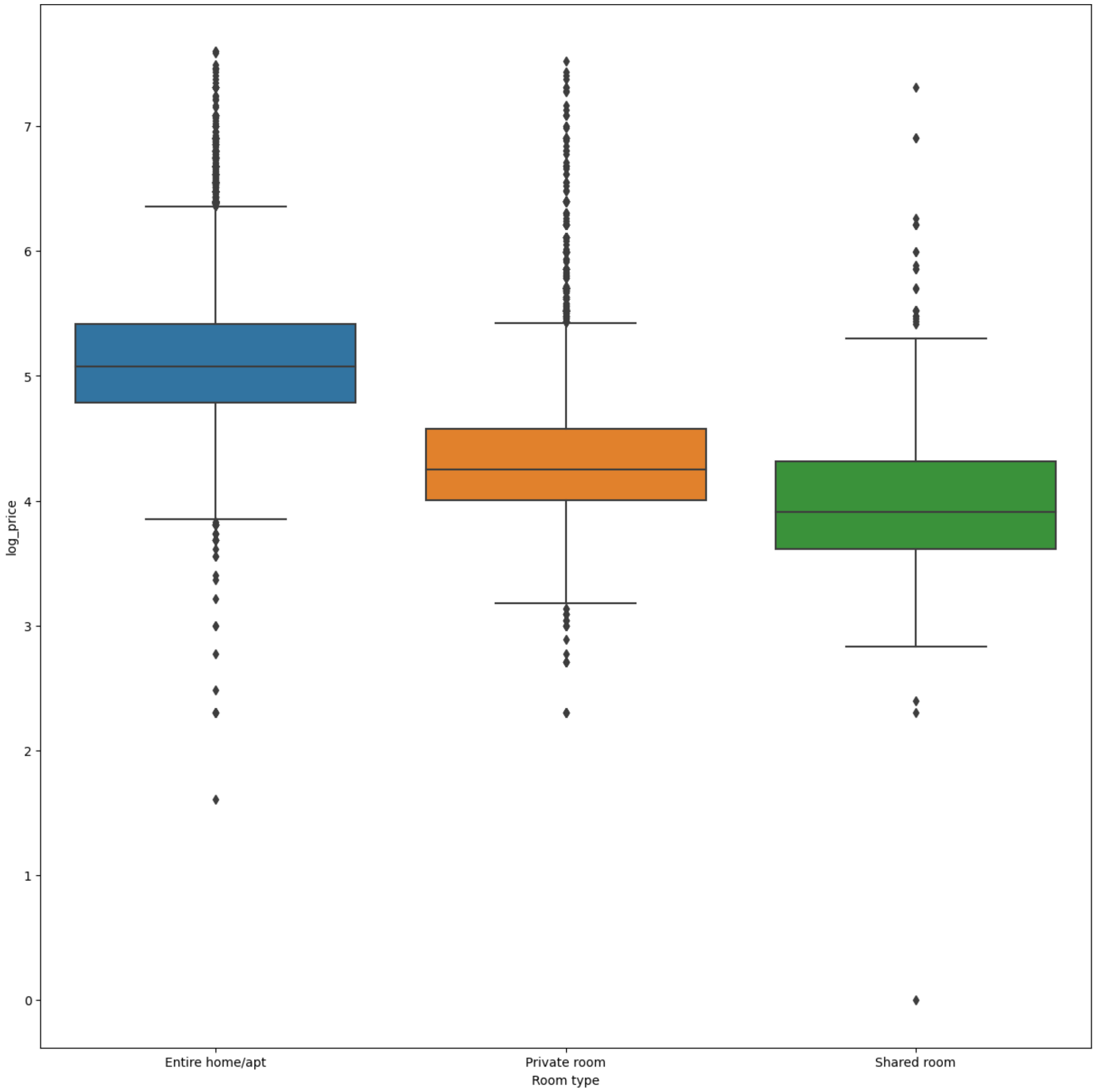


Fig. 4 Density and distribution of prices for room type

5. Instant Bookable Rooms v/s Price:

We have constructed a boxplot to demonstrate the relationship between room type, log price, and the instant bookable feature. This box plot compares the distribution of log prices for various room types based on whether they are instantaneously bookable or not. It can assist in identifying any potential pricing disparities based on the quick bookable functionality across different types of rooms.

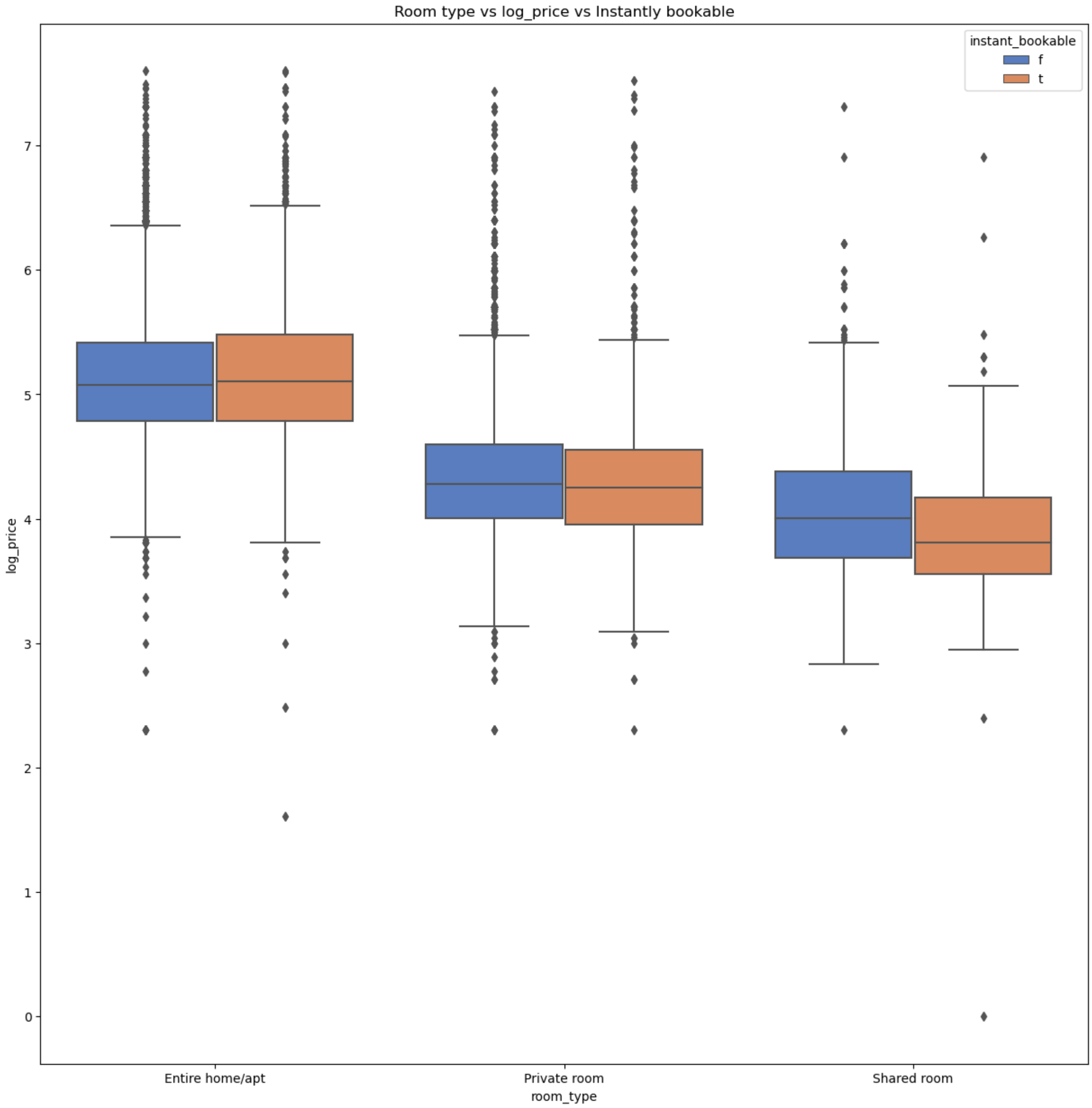


Fig. 5 Instant Bookable Rooms v/s Price

6. Plots for the count of bedrooms, beds, bathrooms, and bed_type:

The four bar charts show the distribution of the number of bedrooms, beds, bathrooms, and bed types in Airbnb listings in New York City.

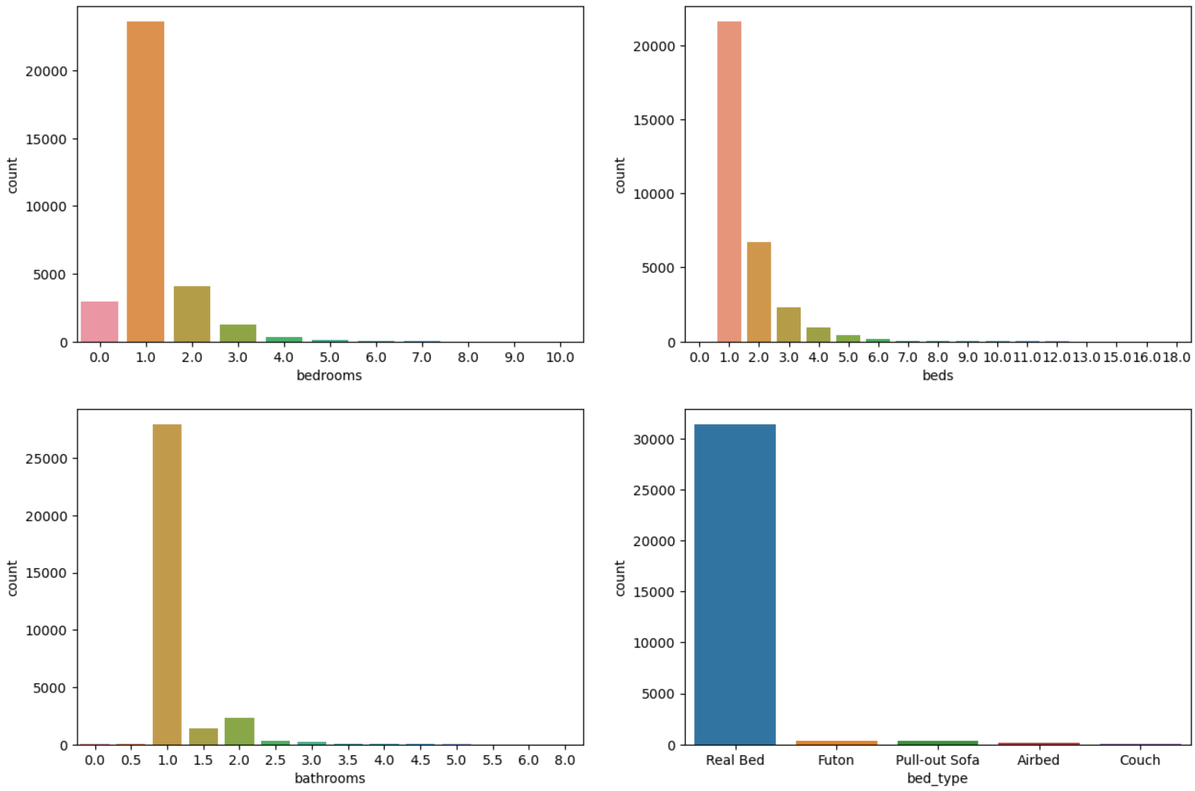


Fig. 6 Plots for the count of bedrooms, beds, bathrooms, and bed_type

- The first bar chart shows the number of bedrooms in a listing. There are very few listings with more than 4 bedrooms, and the most common number of bedrooms is 1.
- The second bar chart shows the number of beds in a listing. There are many more listings with 1 or 2 beds than listings with more beds. This suggests that most Airbnb listings in NYC are small apartments.
- The third bar chart shows the number of bathrooms in a listing. Most listings have 1 bathroom, and there are very few listings with more than 2 bathrooms.
- The distribution of various bed kinds is displayed in the fourth bar chart. The most common sort of bed is the real bed, whereas air beds, futons, and pull-out sofas are less common. This illustrates how the couch bed type is close to zero.

Overall, the bar charts suggest that most Airbnb listings in NYC are small apartments with one or two bedrooms, one bathroom, and one or two beds. These listings are likely targeted towards tourists or short-term visitors.

7. Number of accommodated v/s mean (average) price:

This bar plot shows the relationship between the number of people a property can accommodate and the average price of that property in NYC. The bar graph depicts a rough linear relationship between log_price and the property's capacity to accommodate people. It indicates that the price of the property grows as its capacity to accommodate people increases.

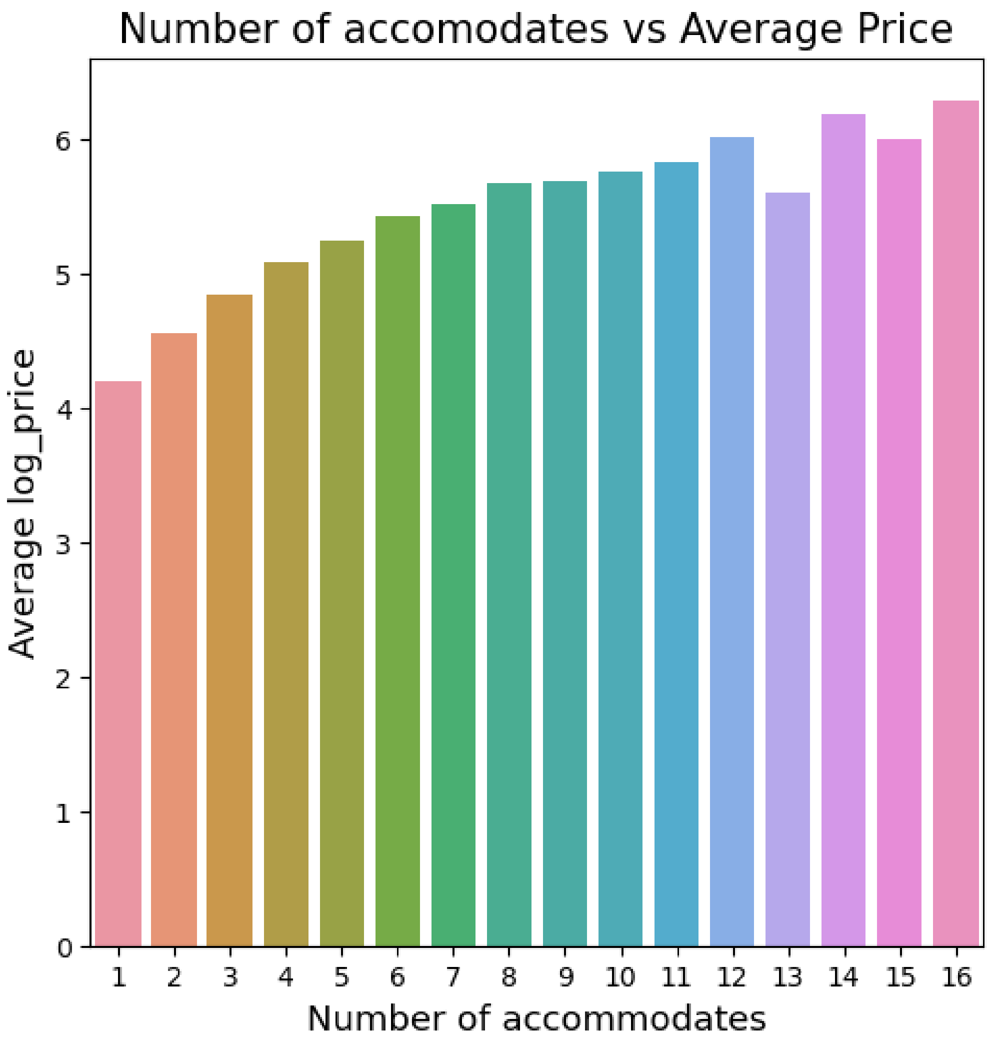


Fig. 7 Number of accommodated v/s mean (average) price

Principal Component Analysis (PCA)

A dataset denoted by 'dummy_df' is subjected to Principal Component Analysis, or PCA. Using 'StandardScaler()', the data is first standardised. Following that, two PCA objects are constructed, one with 116 components ('pca') and the other with 6 components ('pca1'). Principal components are then created by fitting the PCA models to the standardised data. The 'pca_fit1' contains the altered data. To help determine the ideal number of components to keep for dimensionality reduction, the algorithm also generates a scree plot to show the variation explained by each principal component.

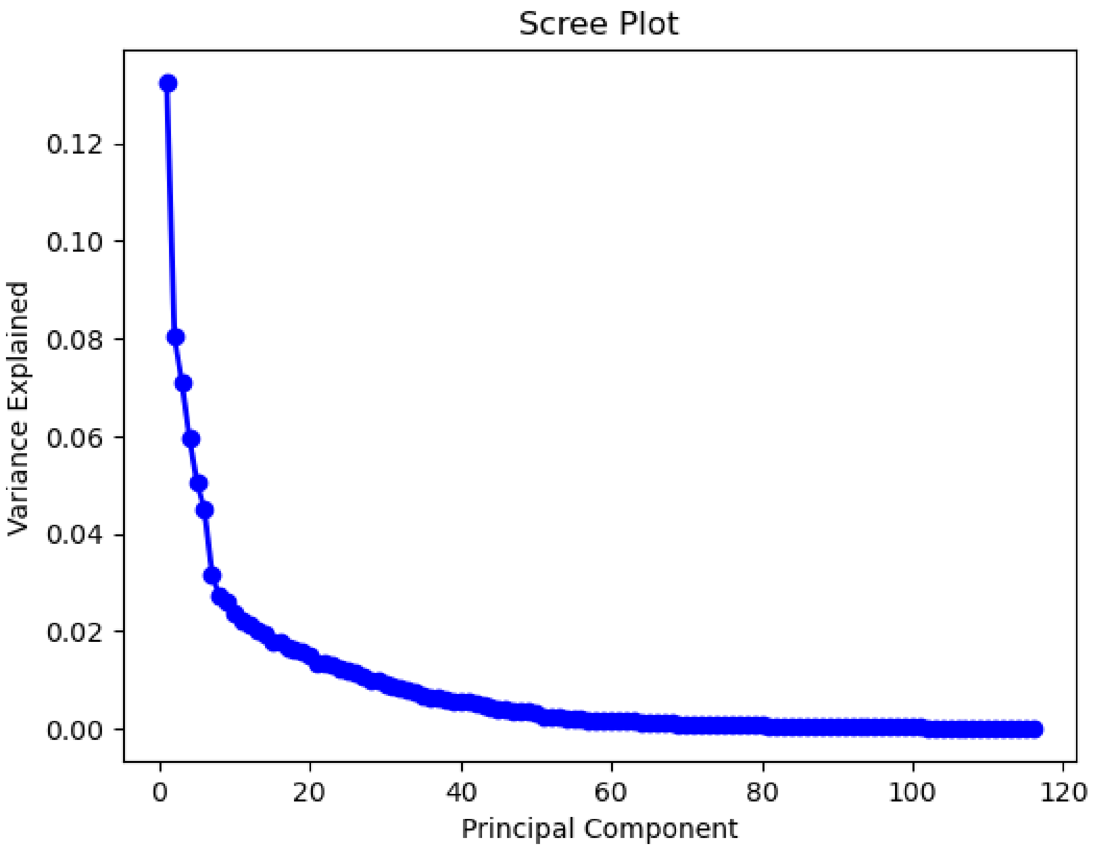


Fig. The variance explained by 6 components is comparable to that of 116 components.

Train-Validation-Test Split

We have used a train-validation-test split of 60-20-20 in our model assessment procedure, which splits the dataset into three subsets: a test set for the final model evaluation, a validation set for hyperparameter tuning and model selection, and a training set for training the model. In our model, 'GridSearchCV' is used to fine-tune hyperparameters to improve machine learning model performance. By systematically searching a predetermined grid of hyperparameters and analysing each combination using cross-validation, 'GridSearchCV' assists us in selecting the ideal set of hyperparameters that maximises model performance. Cross-validation assures that the model's performance estimations are resilient and less prone to overfitting by training and evaluating the model on diverse subsets of the data. As a result, 'GridSearchCV' not only streamlines the hyperparameter tuning process, but it also improves our model's dependability and generalizability by employing cross validation.

Pipeline of Pre-processing

A pipeline method is used to apply one-hot encoding to categorical columns. Initially, two lists are defined: categorical_columns, which contains the names of categorical columns, and numerical_columns, which contains the names of numerical columns. A two-step sequential categorical pipeline is defined for one-hot encoding. The most frequent value in each column is used to fill in any missing values in the first stage, SimpleImputer. OneHotEncoder, the second phase, converts categorical data into binary vectors, including a binary feature for each category. Encoding of the test data ignores any unseen categories, thanks to the handle_unknown='ignore' argument.

Subsequently, parameters for model evaluation are established in order to evaluate regression models' performance. Given the true and projected values, the evaluate_model function computes the coefficient of determination (R2) and the root mean squared error (RMSE).

Finally, ColumnTransformer is used to build a preprocessing pipeline. The preparation procedures for both numerical and categorical columns are combined in this pipeline. While a distinct numerical pipeline is built to handle numerical columns, the previously specified categorical pipeline is applied to categorical columns. By coordinating these pipelines and making sure the proper preparation measures are taken for each type of column, the ColumnTransformer builds a thorough preprocessing workflow for the dataset.

ML Models

We have used a number of machine learning methods, such as Support Vector Regression (SVR), XGBoost, Random Forest, Ridge Regression, Lasso Regression, Linear Regression, and Artificial Neural Networks (ANN).

- **Linear regression** is a basic but effective method for predicting a continuous target variable that involves fitting a linear connection between the input data and the goal.
- **Ridge regression** and **Lasso regression** are linear regression variations that use regularisation to reduce overfitting by penalising big coefficients.
- **Random forest** is an ensemble learning method that generates many decision trees during training and provides the average forecast of the individual trees for regression tasks.
- **XGBoost (Extreme Gradient Boosting)** is a boosting method that successively creates numerous weak learners and combines them to form a strong learner with excellent predicted accuracy.

- **Artificial Neural Networks (ANN)** are computational models based on the biological neural networks of the human brain, with interconnected nodes organised in layers.
- **Support Vector Regression (SVR)** is a regression approach that uses the Support Vector Machine (SVM) framework to discover the hyperplane that best separates data points, with the goal of minimising error within a given margin.

Each method has advantages and disadvantages, making it appropriate for various types of datasets and prediction tasks.

For each model in the validation set, we calculated the root mean squared error (RMSE) and coefficient of determination (R^2). These measures show how well the models fit the data and how precise their forecasts are. We compare the R^2 and RMSE values of each model to determine which algorithm performs best for our particular dataset and prediction task. We can select the best model for use in practical applications by using this methodical comparison to help us make well-informed decisions on model selection.

Python ML Libraries Installed

- ***pandas (pd)***: used for data processing and analysis, including reading, writing, cleaning, and altering data.
- ***numpy (np)***: A set of mathematical functions as well as being necessary for numerical computing.
- ***matplotlib.pyplot (plt)***: Used for data visualisation.
- ***seaborn (sns)***: Provides a high-level interface for creating visually appealing and useful statistical graphs.

- ***folium***: A Python library for building interactive maps and visualisations.
- ***tensorflow.keras***: TensorFlow provides a high-level API for developing and training deep learning models, which includes neural networks.
- ***sklearn (scikit-learn)***: A complete machine learning library that offers simple and effective tools for data mining and analysis.
- ***XGBRegressor (XGBoost)***: An improved gradient boosting library for regression projects.
- ***keras***: A high-level neural networks API built on top of TensorFlow or Theano. (that comprises Sequential, Dense, History, and Adam).

Analysis & Results

The intended objective of this study is to forecast Airbnb rental pricing using a variety of machine learning models such as Linear Regression, Ridge, Lasso, Random Forest, XGBoost, ANN, and SVR. The table below contains information on the model specifications, hyperparameters, results interpretation, and logic for model selection and progression.

The evaluation measures utilised to quantify model performance were RMSE and R^2. Models were validated using a normal train-test split, followed by cross-validation in grid search parameters to verify robustness and generalisability. PCA was used to mitigate dimensionality and perhaps improve model performance by emphasising the most informative characteristics.

Performance comparison of Models

The following table displays the resulting RMSE and R^2 values for all the simple and complex Machine Learning models used:

Model	RMSE (Train)	R^2 (Train)	RMSE (Train)	R^2 (Train)	RMSE (Train)	R^2 (Train)
Linear Regression	0.3865	0.6586	0.3774	0.6737		
Ridge Regression	0.3865	0.6586	0.3893	0.6548		
Lasso Regression	0.4704	0.4943	0.3865	0.6586		
Random Forest	0.4066	0.6222	0.4035	0.6292		
XG Boost	0.3375	0.7397	0.3096	0.7817	0.3307	0.7494
ANN	0.3781	0.5054	0.3679	0.5492		
SVR	0.3401	0.7356	0.3336	0.7465		

Analysis of the models

1. XGBoost:

XGBoost used gradient boosting with the following hyperparameters: n_estimators [50, 100, 150], learning_rate [0.05, 0.1, 0.2], and max_depth [3, 5, 7]. This setup sought to balance model complexity, learning speed, and depth in order to improve forecast accuracy. XGBoost had the best performance, with the lowest RMSE and the highest R^2 across training, validation, and test sets. Its capacity to capture complicated patterns makes it perfect for accurate price projections and strategic pricing optimisation, demonstrating its resilience and dependability for Airbnb pricing analysis.

2. SVR:

To reconcile the trade-off between margin width and fit, SVR was set up with C values [0.1, 1.0, 10.0], and tested with 'linear' and 'rbf' kernels. The 'rbf' kernel was effective in capturing non-linear interactions. SVR demonstrated strong performance, with competitive RMSE and R^2 values similar to those of XGBoost, suggesting its capacity to handle complicated data patterns. This makes SVR a credible alternative for accurate pricing projections, allowing it to be used in the planning process for Airbnb rental pricing.

3. ANN:

To deal with complexity and overfitting, the ANN model evaluated single and two-layer topologies [(50, 100), (50, 50)]. The activation functions 'relu', 'tanh', and 'logistic' were assessed, with 'relu' outperforming the others in terms of efficiency and vanishing gradient mitigation. To control overfitting, alpha values ranging from 0.0001 to 0.01 were used. The model improved moderately in identifying non-linear relationships, as seen by low RMSE and high R^2 metrics. ANN is appropriate for investigating deeper patterns in data, although it failed to surpass XGBoost or SVR in this investigation.

Fine-tuning using cross-validation

Utilising ensemble learning, hyperparameter tuning, and cross-validation. The following actions were made in order to maximise model selection and performance:

Cross-Validation for Initial Model Selection

Evaluating the generality and stability of the model was the goal.

1. K-Fold Cross-Validation was used with five folds to ensure that the performance of every model was evaluated uniformly across various data subsets.
2. The previously listed models were evaluated.
3. Root Mean Squared Error (RMSE) and coefficient of determination (R2) were the evaluation metrics.

Fine-Tuning Hyperparameters

Improving generality and accuracy of the model is the goal. The Grid Search Cross-Validation technique was utilised to methodically investigate and pinpoint the ideal hyperparameters. Tables 1.1 and 1.2 on pages 26 & 27 provide the hyperparameters that have been tuned.

Ensemble Predictions

Utilising each model's unique capabilities to increase overall forecast accuracy was the goal.

Method:

Ensemble Strategy: predictions from the top-performing models—XGBoost and SVR in particular, which displayed better performance metrics—were weighted averaged. Based on model performance (inverse of RMSE), weights were computed.

In order to create a final ensemble prediction that balanced the advantages of support vector regression and gradient boosting, predictions from XGBoost and SVR were combined.

Summary

Cross-validation was essential for assessing early models, which resulted in well-informed grid search hyperparameter tuning. The final model was made more accurate and resilient by the ensemble technique, which included SVR and XGBoost predictions. This made the model ideal for forecasting Airbnb rental prices. Robust model performance is ensured by this thorough process, making it appropriate for tactical price decisions.

Reasoning for Model Progression

1. To create a baseline, comprehend the data, and determine the significance of each feature, we began with simple models (Linear, Ridge, and Lasso).
2. To enhance prediction capability and examine non-linearity and feature interactions, ensemble methods (Random Forest and XG Boost) were upgraded to.
3. Updated to Complex Models (SVR and ANNs) to record more detailed patterns and optimise performance; SVR displays competitive outcomes, while ANN offers a neural network viewpoint.

Table 2.1 Comparative Table of Various Hyperparameter Selections for Ensemble Regression and Non-Linear Models

Hyperparameter	XG Boost	SVR	ANN
Model Type	Gradient Boosting	Support Vector Regression	Multi-layer Perception
Hidden Layers	-	-	hidden_layer_sizes
Values Tested	-	-	[(50,),(100,),(50, 50)]
Activation Function	-	-	Activation
Values Tested	-	-	[‘relu’, ‘tanh’, ‘logistic']
Regularisation	reg_alpha, reg_lambda	C	alpha
Values Tested	Typical default, not tested here	[0.1, 1, 10]	[0.0001, 0.001, 0.01]
Learning Rate	learning_rate	-	-
Values Tested	[0.05, 0.1, 0.2]	-	-
Number of Trees	n_estimators	-	-
Values Tested	[50, 100, 150]	-	-
Tree Depth	max_depth	-	-
Values Tested	[3, 5, 7]	-	-

Table 2.1 Comparative Table of Various Hyperparameter Selections for Ensemble Regression and Non-Linear Models

Hyper-parameter	Linear Regression	Ridge Regression	Lasso Regression	Random Forest
Model type	Linear Regression	Linear Regression with L2	Linear Regression with L1	Ensemble method
Framework	Linear Regression	Ridge	Lasso	RandomForestRegressor
Regularisation	-	alpha	alpha	max_features, min_samples_split, min_sample_leaf
Values Tested	-	[0.1, 1, 10]	[0.1, 1, 10]	-
Number of trees	-	-	-	n_estimators
Values Tested	-	-	-	[50, 100, 150]
Tree Depth	-	-	-	max_depth
Values Tested	-	-	-	[1, 2, 3, 4]

Conclusion

According to the analysis, XGBoost has the best RMSE and R2 scores across training, validation, and test datasets, making it the best model for forecasting Airbnb rental costs. SVR is a potent substitute since it demonstrates competitive performance as well.

Strong predictive modelling is ensured by this all-encompassing method, which supports Airbnb hosts' strategic decisions about price, investments, and market analysis.

Discussion and Conclusion

Explaining results to the owner

We employed a variety of regression analyses and techniques throughout the project, including ANN, XGBoost, Lasso, Random Forest, Ridge, Linear, and SVR. SVR was the next best performer, with XGBoost showing the greatest improvement in predicted accuracy. On the validation set, the XGBoost model yields an RMSE value of 0.3096, which is comparatively better than typical linear models.

In simple terms, we can say that we were able to forecast the rental price with high accuracy and low error using one of the machine learning models, XGBoost, based on information around the properties that we fed in.

Improvement in business

Precise estimation of costs can aid hosts in attracting more reservations, and seasonal and volatile price adjustments can guarantee year-round booking consistency. The hosts' profits may increase as a result. According to one study, the hosts' annual losses as a result of imprecise pricing keeping approached forty thousand dollars. The hosts have a good chance of making $46,000 in addition to their regular sources of income and profits to make up for those losses.

Organisational Changes Required

1. ***Implementation of the fresh model***: From the perspective of tech implementation, there will be a great deal of change. This indicates that organisational activities would be required to link or replace the current price

prediction model on the out-of-service tech infrastructure and execute the modification in the price forecasting model. A significant amount of business will would be required for this, ideally from the CPO and CTO, who can then supervise and oversee the IT and product teams as they work to accomplish the change.

2. ***Training and Support***: To execute introduction, knowledge transfer, and training of the hosts to start using and pricing prediction model for appropriate rental prices, a large-scale deployment of training and support teams, both online and offline, would be required.

3. ***Data infrastructure***: To continuously add and update new data from various sources and subject them to rigorous testing in accordance with the model, an infrastructure upgrade would be required. This would allow the price prediction model's accuracy to continue to increase while lowering error.

Affected business processes

1. ***Tech and Product teams***: The introduction of a new service and the need for maintenance, testing, and training to support and collect massive volumes of data as well as to manage servers would have an impact on the technical, engineering, and product teams.

2. ***Customer Success and Support***: Customers would be very excited about the new pricing prediction model, and they would also have many questions and concerns about its adoption and the hosts, who are the final consumers, would want to ensure a smooth and easy implementation. The hosts would have many questions about how to utilise it, the financial rewards they may receive, and the training needed to implement the recommendations for improving certain aspects of their properties. This would entail the success and customer support teams being prepared and upskilled.

3. ***Accounts and Finance Team***: The price prediction model's introduction would call for more precise accounting and record keeping, which would necessitate the finance and account teams' training and upskilling.

Convincing Stakeholders

1. ***Demonstrate benefits (financial and technical)***: We would need to outline the advantages of using the price prediction model. We must demonstrate with numerical data how profitable the model's implementation would be, requiring minimal short-term work but yielding long-term benefits for Airbnb and, consequently, for the teams and individuals who work there. Additionally, we would have to demonstrate how we would make the lives of hosts easier by streamlining price prediction and dynamic pricing changes, which would boost their earnings and enhance their Airbnb hosting experience.
2. ***Run pilot***: Although the model appears promising on paper, business stakeholders won't be persuaded to invest in it and upgrade their current infrastructure until it has been implemented and the benefits have been demonstrated in real time.

Limitations and Further Development

1. ***Data Limitations***: Existing models depend on features that are currently accessible, which may not fully represent market dynamics and certain models may not be developed to address today's complicated issues.
2. ***Universal applicability***: Depending on the local characteristics, the model might not be universally applicable and might need to be modified.
3. ***Upcoming Improvements:***
    a. *Extra Information*: Include extra specific information, including regional events or seasonal patterns.

b. *Real-time Data*: To increase the accuracy of dynamic pricing, develop models using real-time data streams.

c. *User Feedback Loop*: Consider suggestions from both hosts and visitors to make the model better over time.

4. ***Data Collection***:

a. New Sources: Look into joint ventures to find more comprehensive data sources.

b. User Interaction: To improve the model's prediction power, collect information on user behaviour and preferences.

References

1. "*Dynamic pricing and benchmarking in AirBnB" by J. Sims, N. Ameen & R. Bauer (2019).*
2. *"Airbnb Dynamic Pricing Using Machine Learning" by Y. Wang (2023).*
3. *"A Data-Driven Look at Airbnb in NYC: Market Trends, Insights and Best Practices" by L. Cao (2023).*

Appendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import folium
import tensorflow.keras as keras
from folium.plugins import HeatMap
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, Lasso
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_classification
from sklearn.preprocessing import FunctionTransformer
from sklearn.neural_network import MLPRegressor
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import History
from keras.optimizers import Adam

data =
pd.read_csv("C:/Users/navab/OneDrive/Documents/masterclass/train_data.
csv")
#C:\Users\navab\OneDrive\Documents\masterclass

data_cleaned = data[data['city'].str.contains('NYC', case=False,
na=False)]

# Save the filtered data to a new CSV file called data_cleaned.csv
data_cleaned.to_csv('data_cleaned.csv', index=False)

#data_cleaned['Price'] = np.exp(data_cleaned['log_price'])

data_cleaned=data_cleaned.drop("first_review",axis='columns')

data_cleaned=data_cleaned.drop("host_response_rate",axis='columns')

data_cleaned=data_cleaned.drop("review_scores_rating",axis='columns')
```

```
data_cleaned=data_cleaned.drop("last_review",axis='columns')

data_cleaned=data_cleaned.drop("thumbnail_url",axis='columns')
```

For travel time columns : 'avg_lead_time','min_lead_time','min_airport_time', 'avg_airport_time','man_time','bro_time','que_time','bnx_time', 'sta_time','nj_time'

Read data from CSV file

data = pd.read_csv("C:/Users/navab/Downloads/train.csv") data_cleaned = data[data['city'].str.contains('NYC', case=False, na=False)]

Save the filtered data to a new CSV file called data_cleaned.csv

#data_cleaned.to_csv('data_cleaned.csv', index=False)

lat_airbnb = data_cleaned['latitude'].tolist() long_airbnb = data_cleaned['longitude'].tolist()

def traveltime(lat, long): url = f"http://127.0.0.1:5000/route/v1/driving/{long},{lat};-73.8718,40.7733?steps=true" #40.7733, -73.8718
response = requests.get(url) if response.status_code == 200: duration = response.json()['routes'][0]['duration'] return duration else: print("Error fetching travel time:", response.text) return None

Save results to Excel file

file_path = "C:/Users/navab/OneDrive/Documents/masterclass/airports.xlsx"

Check if the Excel file exists, if not create it with the first row for headers

if not os.path.exists(file_path): with pd.ExcelWriter(file_path, engine='openpyxl', mode='w') as writer: pd.DataFrame(columns=['start_latitude', 'start_longitude', 'end_latitude', 'end_longitude', 'avg_travel_time', 'min_travel_time']).to_excel(writer, sheet_name='Sheet1', index=False)

Loop through each Airbnb location, calculate travel times, and append to the excel sheet in Excel file

with pd.ExcelWriter(file_path, engine='openpyxl', mode='a') as writer: # Load the workbook workbook = writer.book # Check if the sheet already exists sheet_name = 'LaGuardia Airport' if sheet_name in workbook.sheetnames: worksheet = workbook[sheet_name] else: worksheet = workbook.create_sheet(sheet_name) # Write column headers if the sheet is newly created headers = ['start_latitude', 'start_longitude', 'end_latitude', 'end_longitude', 'avg_travel_time', 'min_travel_time'] worksheet.append(headers)

```
for lat, long in zip(lat_airbnb, long_airbnb):
    # Calculate travel times
    avg_time = traveltime(lat, long)
    min_time = traveltime(lat, long)

    # Append data to the sheet
    data_row = [lat, long, 40.5031, -74.2533 , avg_time, min_time]
    worksheet.append(data_row)
```

Print confirmation

print("Travel times saved to Excel file.")

For a new column Borough

#longlat = pd.read_csv("longlat.csv",encoding = 'utf-8', sep = '\t') geolocator = geopy.Nominatim(user_agent="check_1")

def get_zip_code(x): location = geolocator.reverse("{}, {}".format(x['latitude'],x['longitude'])) return location.raw['address']['postcode'] appended_df['zipcode_new'] = appended_df.apply(lambda x: get_zip_code(x), axis = 1) print(appended_df.head())

```
# Replace null values with median in specific columns
columns_to_fill = ['bathrooms', 'bedrooms', 'beds']  # Specify columns you want to fill null values for
for column in columns_to_fill:
    median_value = data_cleaned[column].median()  # Calculate median for the column
    data_cleaned[column].fillna(median_value, inplace=True)  # Replace null values with median
```

```
#Appending the amenities to list l
l=list(data_cleaned['amenities'])
# splitting the list of amenities in a individual rows into list of
lists
l=[[word.strip('[" ]') for word in row[1:-1].split(',')] for row in l]
#creating a new column 'amenities_lists' and then assiging the
previous l to amenities_lists.
data_cleaned['amenities_lists'] = l
all_amenities = [item.strip('[" ]') for sublist in l for item in
sublist]
# Count the frequency of each unique amenity
amenity_counts = pd.Series(all_amenities).value_counts()
# Calculate the percentage of occurrence of each amenity
amenity_percentages = (amenity_counts / len(data_cleaned)) * 100
# Output the frequency and percentage of each amenity
print(amenity_counts)
print(len(amenity_percentages))
```

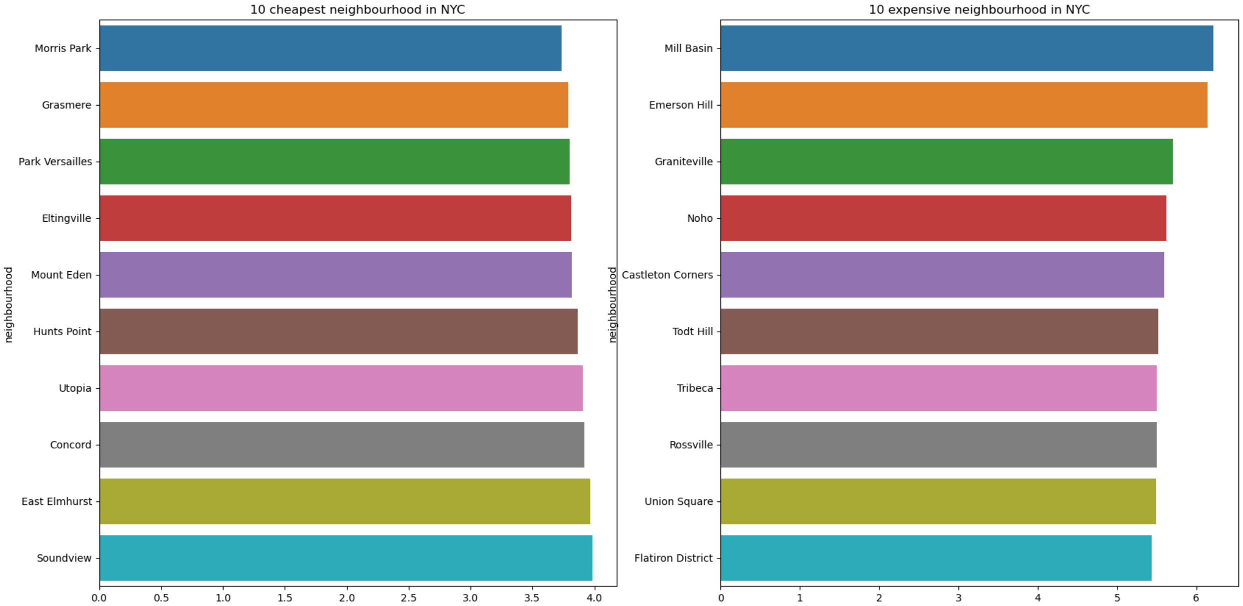
```
Wireless Internet            31304
Kitchen                      30068
Heating                      29523
Essentials                   27561
Air conditioning             27159
                             ...
Air purifier                    12
Washer / Dryer                  11
Lake access                     11
Ski in/Ski out                   6
Roll-in shower with chair        1
Name: count, Length: 117, dtype: int64
117
```

```
# Convert list of amenities to string
data_cleaned['amenities_lists'] =
data_cleaned['amenities_lists'].apply(lambda x: ','.join(x))

# Split the string into separate columns
amenities_df =
data_cleaned['amenities_lists'].str.get_dummies(sep=',')

# Concatenate the new columns with the original DataFrame
data_cleaned = pd.concat([data_cleaned, amenities_df], axis=1)

# Drop the original 'amenities_lists' column
data_cleaned.drop(columns=['amenities_lists'], inplace=True)

dummy_df=data_cleaned.copy()

columns_to_drop = ['id',
'accommodates','log_price','room_type','amenities','property_type','ba
throoms','bed_type',

'cancellation_policy','cleaning_fee','city','description','host_has_pr
ofile_pic','host_identity_verified',

'host_since','instant_bookable','latitude','longitude','name','neighbo
urhood','number_of_reviews',
                   'beds','Borough','bedrooms','zipcode']

# Drop the specified columns
dummy_df.drop(columns=columns_to_drop, inplace=True)

new_data =
pd.read_excel('C:/Users/navab/OneDrive/Documents/masterclass/append.xl
sx')
```

EDA

```
#Top 10 expensive & Cheapest neighbourhood in NYC
a = data_cleaned.groupby('neighbourhood')
['log_price'].mean().sort_values(ascending=True).head(10)
b = data_cleaned.groupby('neighbourhood')
['log_price'].mean().sort_values(ascending=False).head(10)
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(121)
sns.barplot(y=a.index, x=a.values, ax=ax1)
ax1.set_title('10 cheapest neighbourhood in NYC')
ax2 = fig.add_subplot(122)
sns.barplot(y=b.index, x=b.values, ax=ax2)
ax2.set_title('10 expensive neighbourhood in NYC')
plt.show()

C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

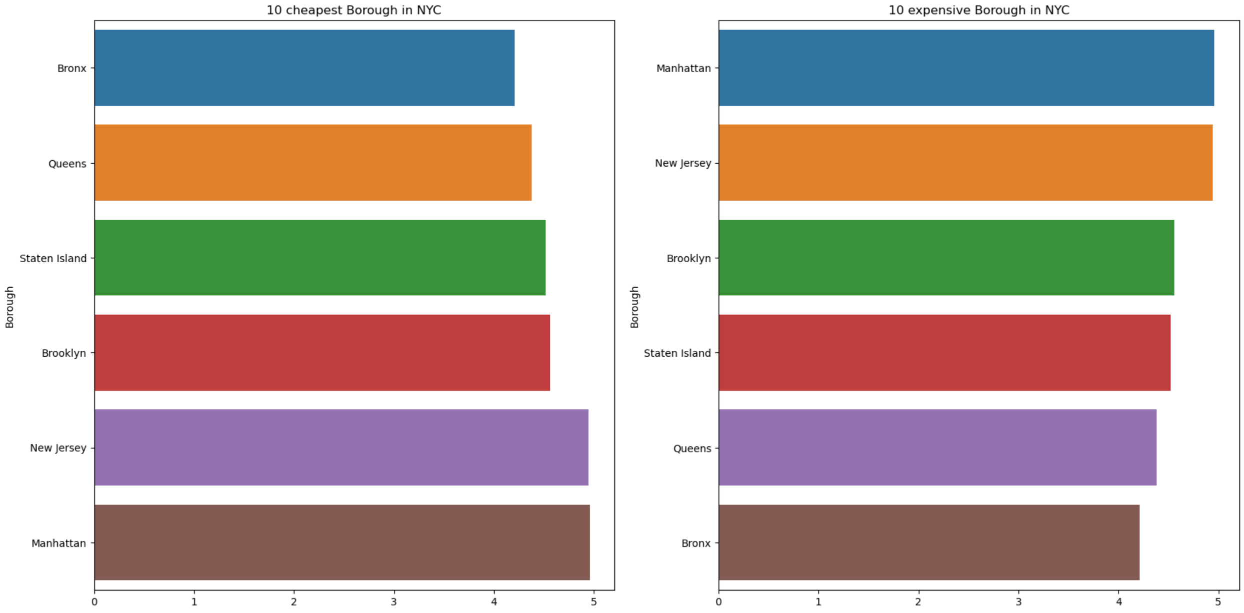
```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



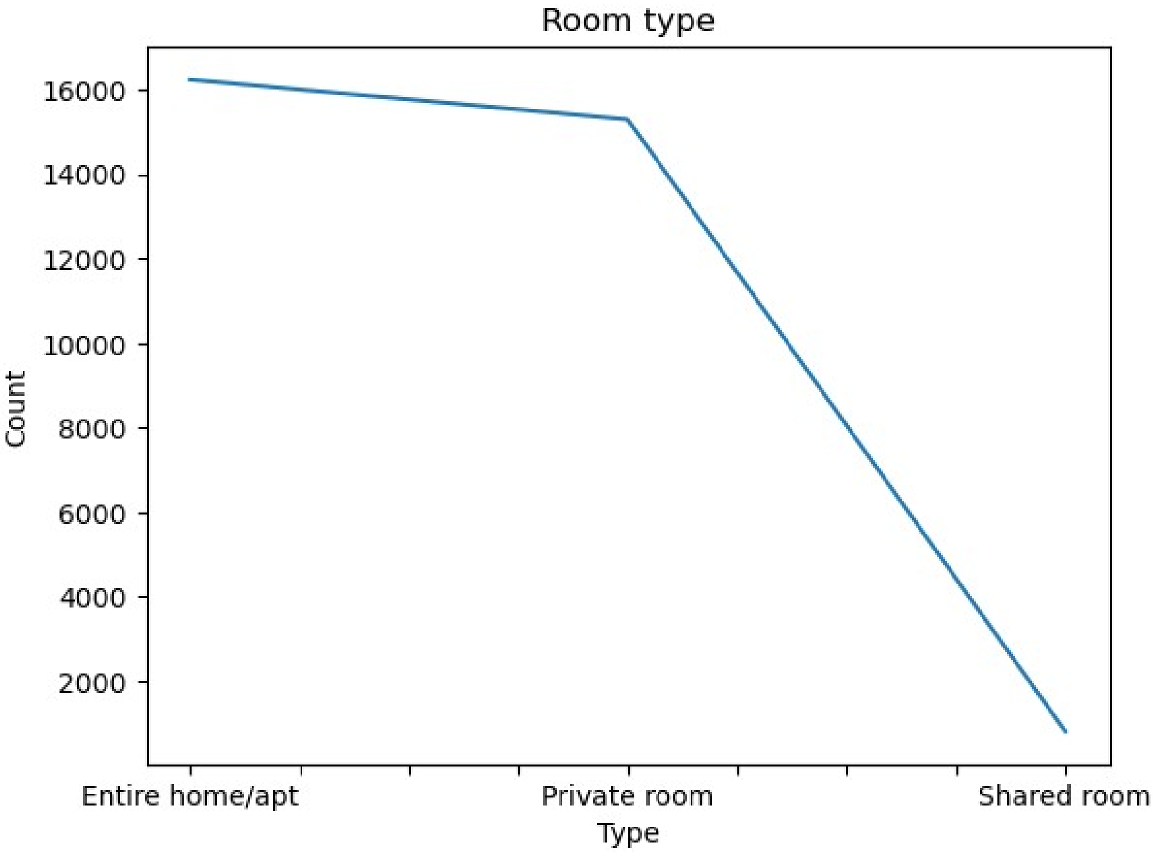
```python
#Top 10 expensive & Cheapest Borough in NYC
a = data_cleaned.groupby('Borough')
['log_price'].mean().sort_values(ascending=True).head(10)
b = data_cleaned.groupby('Borough')
['log_price'].mean().sort_values(ascending=False).head(10)
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(121)
sns.barplot(y=a.index, x=a.values, ax=ax1)
ax1.set_title('10 cheapest Borough in NYC')
ax2 = fig.add_subplot(122)
sns.barplot(y=b.index, x=b.values, ax=ax2)
ax2.set_title('10 expensive Borough in NYC')
plt.show()
```

```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
```

```
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

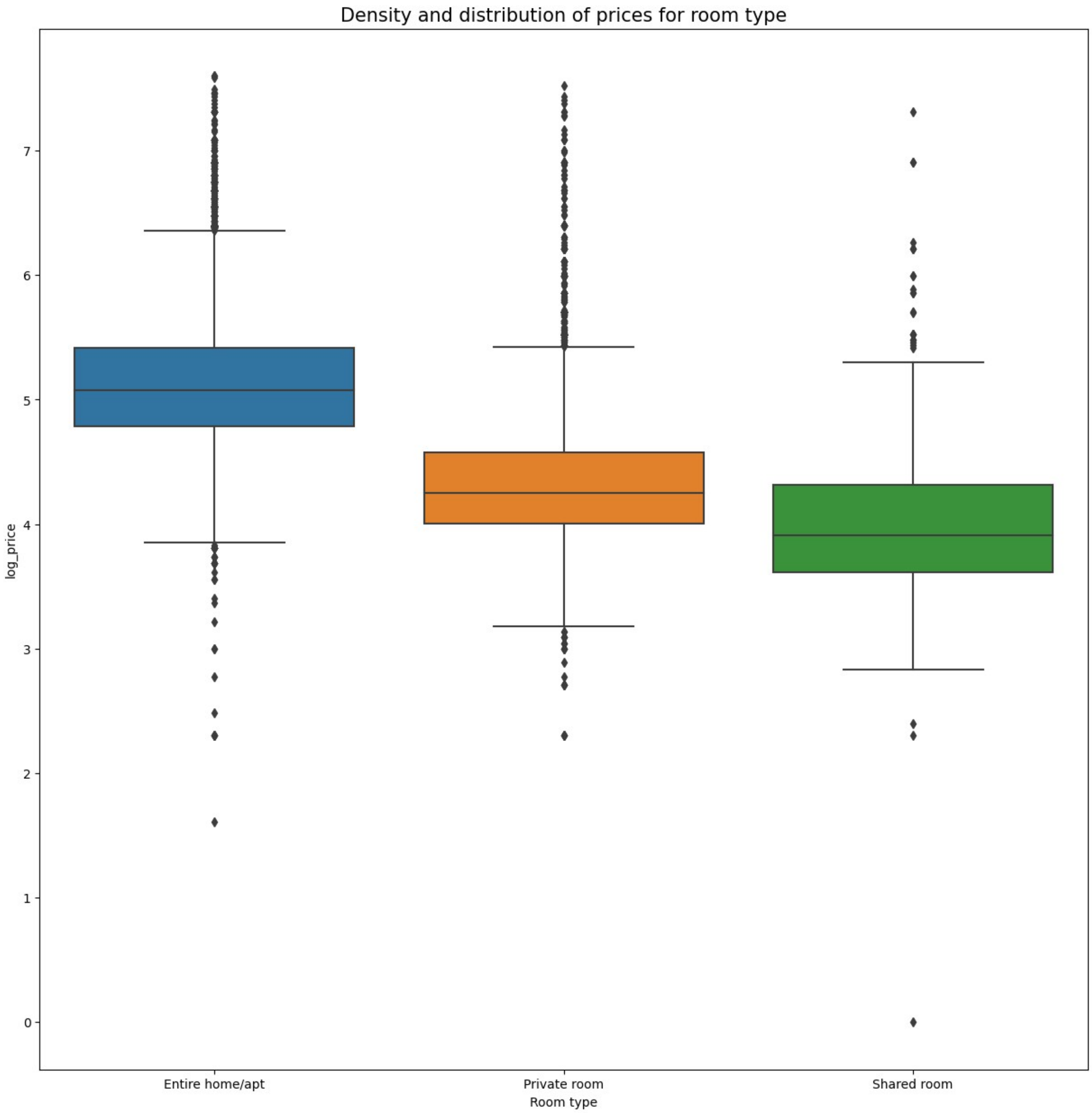


```
#Ploting room count
data_cleaned['room_type'].value_counts().plot(kind='line')
plt.title('Room type')
plt.ylabel('Count')
plt.xlabel('Type')

Text(0.5, 0, 'Type')
```



```
#Box Plot to view outliers
plt.figure(figsize=(15,15))
sns.boxplot(data=data_cleaned, x='room_type', y='log_price')
plt.title('Density and distribution of prices for room type',
fontsize=15)
plt.xlabel('Room type')
plt.ylabel("log_price")
```

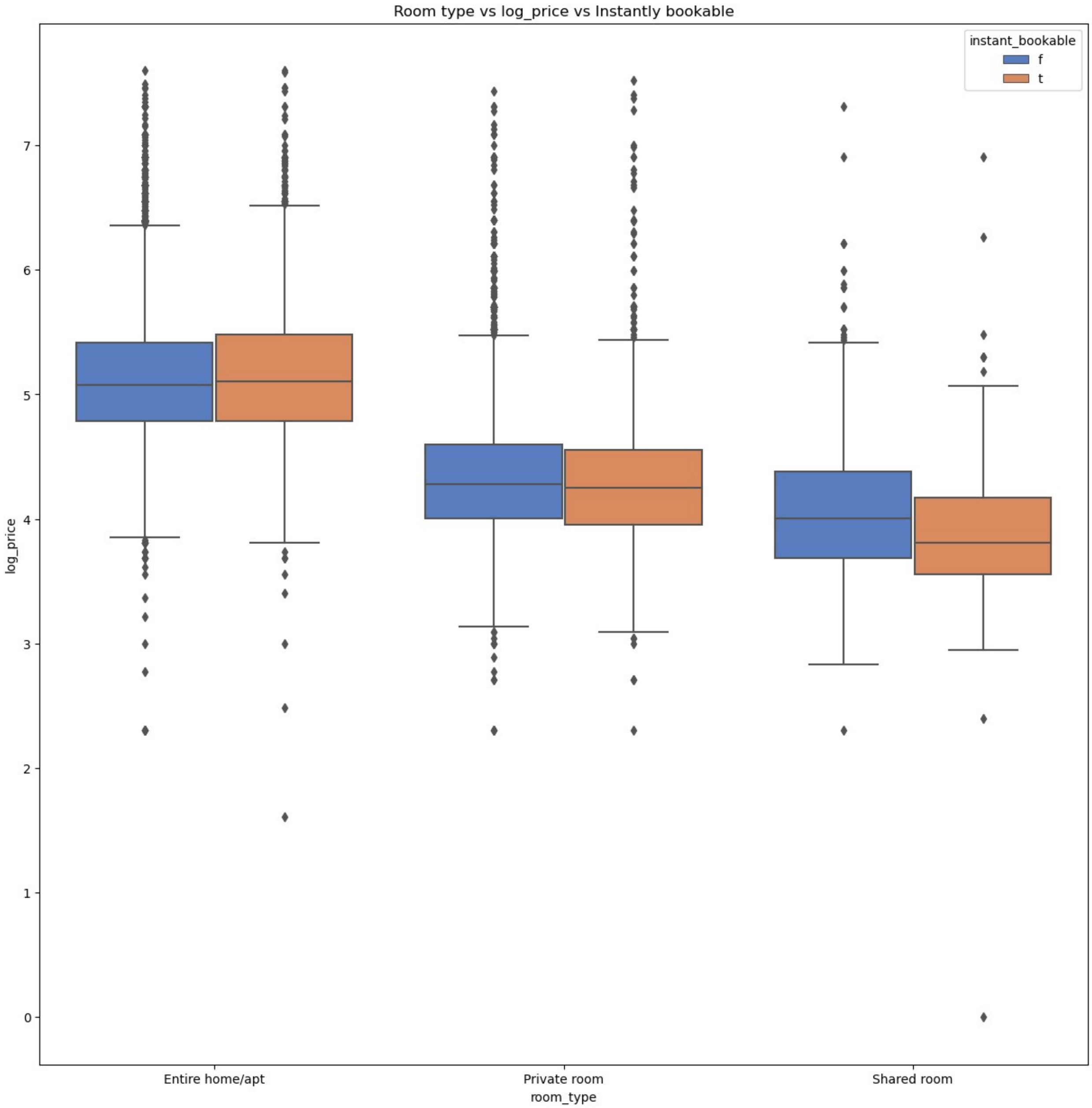
```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):

Text(0, 0.5, 'log_price')
```



```
#correlating instant bookable rooms with price
plt.figure(figsize=(15,15))
sns.boxplot(x='room_type',y='log_price', hue="instant_bookable",
data=data_cleaned, palette='muted')
plt.title("Room type vs log_price vs Instantly bookable")
```

```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
```

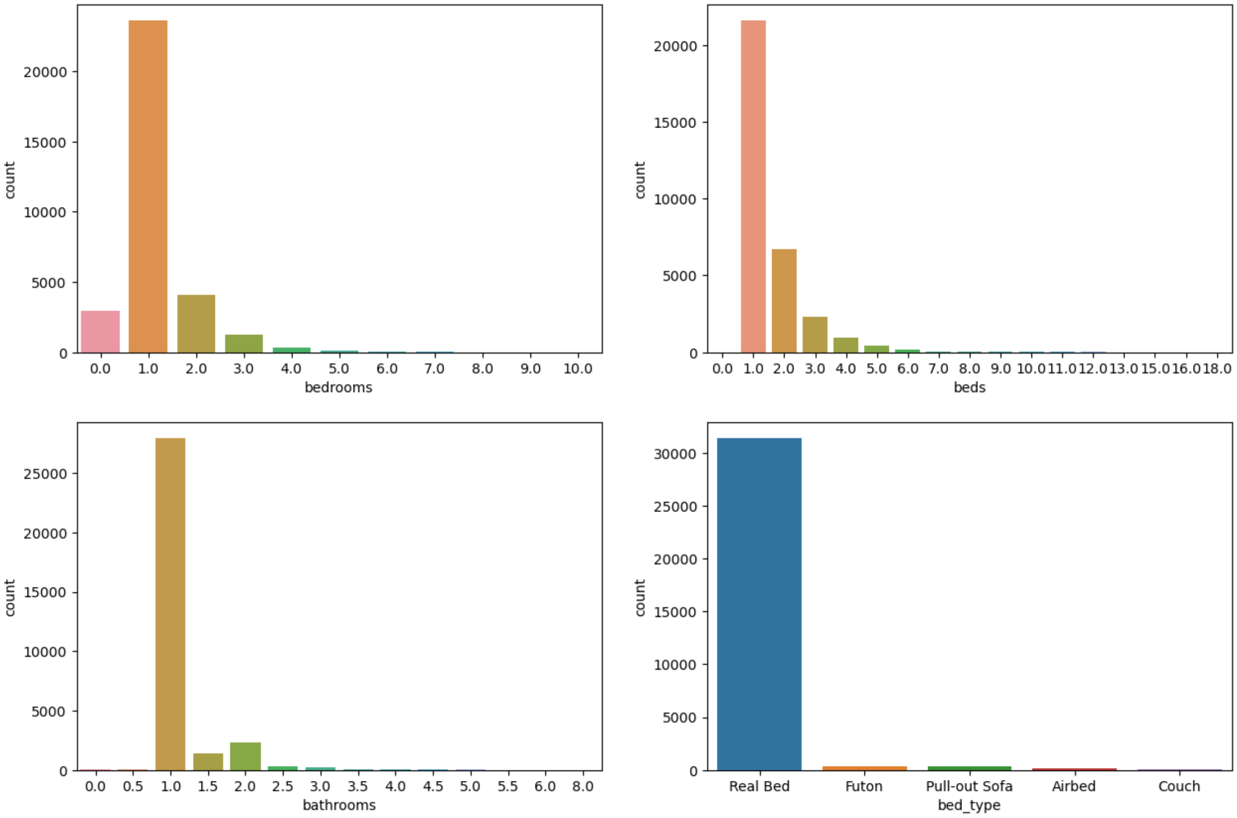
```
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):

Text(0.5, 1.0, 'Room type vs log_price vs Instantly bookable')
```



```python
#plotting the count of bedrooms, beds and bathrooms.
a=['bedrooms','beds','bathrooms','bed_type']
fig, axes = plt.subplots(2,2, figsize=(15,10))
axes = [ax for axes_row in axes for ax in axes_row]
for i, c in enumerate(a):
    f = sns.countplot(x=data_cleaned[c], data=data_cleaned,
ax=axes[i])
```

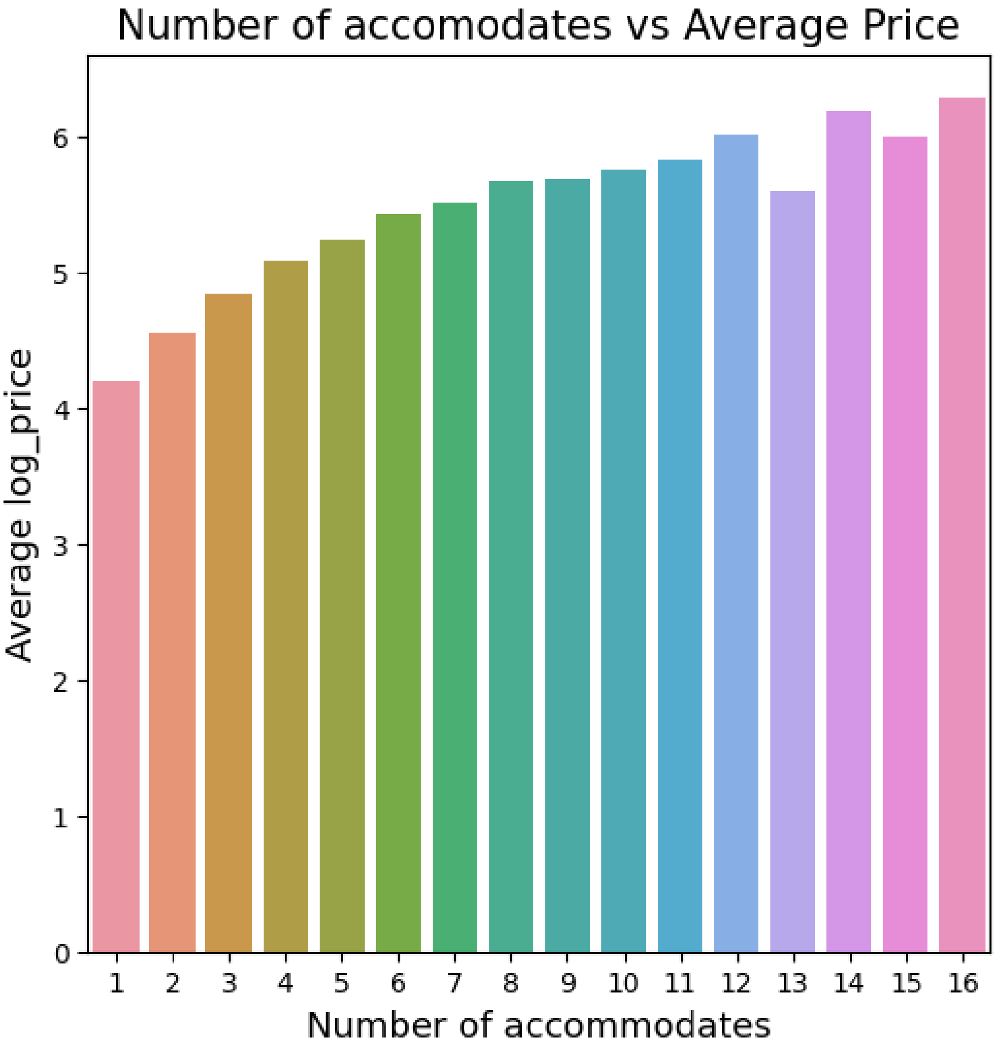
```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
```

```
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



```
#plotting the number of accomodates vs mean price
a = data_cleaned.groupby('accommodates')['log_price'].mean()
fig = plt.figure(figsize=(6,6))
sns.barplot(y=a.values, x=a.index)
plt.xlabel("Number of accommodates", size=13)
plt.ylabel("Average log_price", size=13)
plt.title("Number of accomodates vs Average Price",size=15)

C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
```

```
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):

Text(0.5, 1.0, 'Number of accomodates vs Average Price')
```



PCA

```
#need to perform pca on dummy_df
scaler = StandardScaler()
```

```
scaled_df = dummy_df.copy()

pca = PCA(n_components=116)

pca1=PCA(n_components = 6)

pca_fit = pca.fit(scaled_df)

pca_fit12 = pca1.fit(scaled_df)

pca_fit1=pca1.transform(scaled_df)

pca1.transform(scaled_df[:10])
```

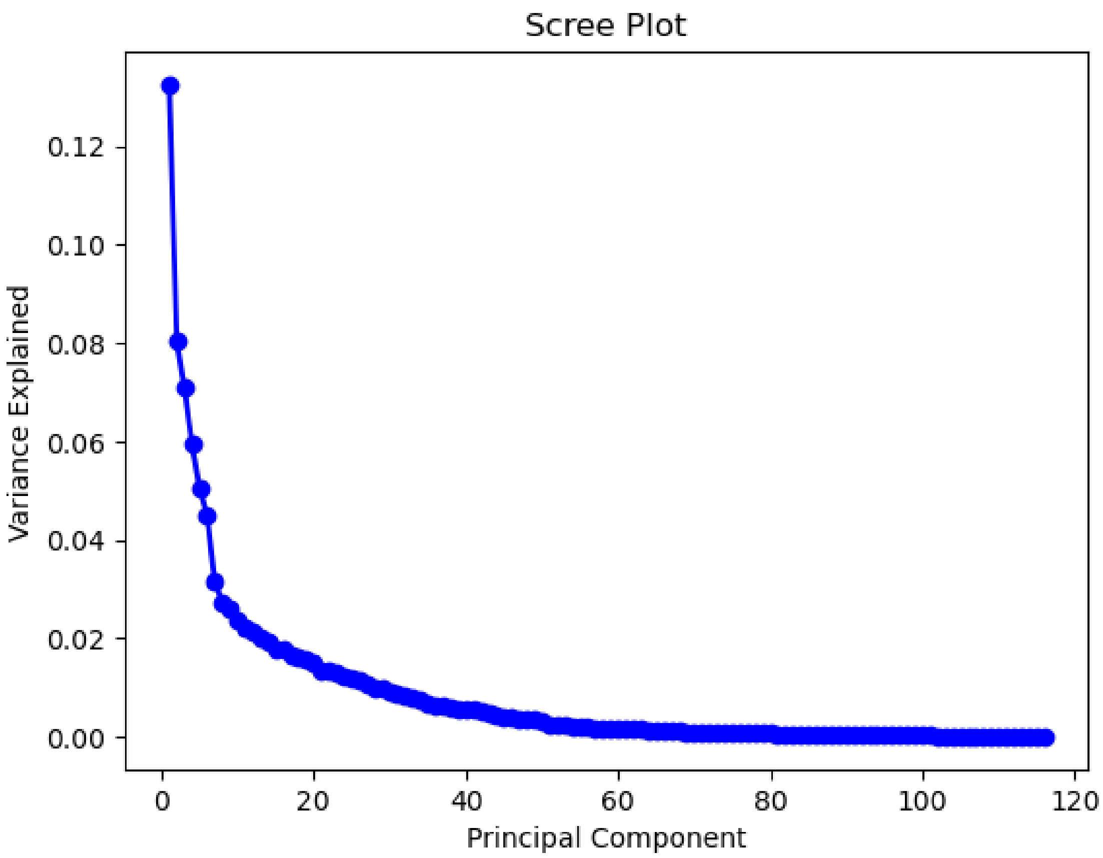
```
array([[-0.90916489,  0.67098529, -0.11362837, -0.143054  , -
0.47203793,
        -0.70811846],
       [ 0.22671311, -0.18588384, -0.12756037,  0.47851232, -0.9858864
,
        -0.06805372],
       [ 0.84820567, -0.08654152, -0.83604227, -0.54867924,
0.09451501,
        -0.23088921],
       [-0.24722777, -0.4382555 , -0.02444861,  0.90540099,
0.08704122,
        -0.52673114],
       [ 0.55000783, -0.67199802, -0.45413866, -0.33247472,  0.5509238
,
        -0.79010181],
       [ 1.38034558,  2.06977363,  1.27228   , -0.19027759, -
0.23631966,
        -0.3181557 ],
       [-1.46252884,  0.63757736, -0.2360672 ,  0.39472172,
0.86736554,
         0.6486099 ],
       [ 2.65433137,  1.40603338,  1.58679579, -1.18163796,
0.83496523,
         0.25163726],
       [ 0.72370277, -0.96664181,  0.01645475,  0.60814722, -
0.41227077,
        -0.09560317],
       [ 0.28946433, -0.71472352, -0.33852669,  1.53261092,
0.36807572,
        -1.15948001]])
```

```
pca_df=pd.DataFrame(pca_fit1)

pd.DataFrame(pca_fit.components_)
```

```
          0              1              2              3              4
\
0    0.137382  1.424372e-02  1.159859e-02  1.046229e-01  7.175293e-04

1   -0.102945  1.650898e-02  1.363790e-02 -9.730801e-02  6.828271e-04

2   -0.099197  2.259258e-02  1.815250e-02  5.003380e-03  9.998758e-04

3   -0.052632  4.916447e-04  7.661006e-04 -1.746111e-02  1.407422e-04

4    0.123162  7.038752e-03  5.689725e-03  5.699134e-02  2.520919e-04

..        ...           ...           ...           ...           ...

111  0.000131  7.821588e-03  2.630726e-02  3.825468e-04  9.196554e-01

112 -0.000034  1.725062e-03 -9.274587e-03 -1.710427e-04 -1.093741e-01

113  0.000297  3.367874e-03  1.540443e-03 -1.342787e-04  5.670926e-03

114  0.000064 -1.906471e-03  1.450977e-03 -2.803750e-05 -4.258075e-04

115 -0.000000  1.157900e-16  2.552362e-17  6.238417e-17 -3.143749e-16


              5             6             7             8
9    \
0    1.895863e-02  5.949161e-03  2.976313e-03  1.602122e-02
6.906973e-02
1    1.902324e-02  1.580759e-03 -2.544760e-04  5.085759e-03
2.520115e-02
2    3.168135e-02  6.524673e-03  2.827173e-03  1.626134e-02
5.870920e-02
3   -5.131347e-04 -3.639441e-03 -1.615183e-03 -1.072232e-02 -
2.185101e-02
4    9.466329e-03  5.153235e-04  1.788714e-04  2.730256e-03
2.373407e-02
..            ...           ...           ...           ...
...
111 -9.811466e-04 -2.531750e-03 -1.752887e-02  1.839599e-03 -
1.278494e-04
112  2.085890e-03 -1.352332e-02  1.052703e-02 -3.981587e-04 -
9.865497e-04
113  7.543240e-04 -9.143931e-03  6.557999e-03  1.647344e-03 -
6.064726e-04
114 -2.290550e-03  4.774788e-04  2.508663e-03 -1.454260e-03
6.720426e-04
115 -4.606637e-17 -5.584792e-17  2.246062e-16  3.175656e-16 -
3.345563e-17

     ...           106           107           108           109  \
0    ...  1.771138e-02  5.455106e-03  2.363949e-02  1.335250e-02
```

```
1    ...  1.804377e-02  5.893972e-03  2.374305e-02  1.388778e-02
2    ...  2.865894e-02  8.480806e-03  3.722469e-02  2.188382e-02
3    ...  1.139428e-03  6.738696e-04  1.922881e-03  1.255536e-03
4    ...  7.848097e-03  2.933133e-03  1.074480e-02  5.431693e-03
..   ...           ...           ...           ...           ...
111  ... -5.192839e-03 -2.199085e-02  2.939154e-03  9.005955e-03
112  ...  6.663182e-03  1.885612e-02 -1.332500e-02  1.287227e-03
113  ... -5.351768e-03  7.339905e-04 -2.359373e-04  4.352717e-03
114  ... -9.103961e-04 -8.870715e-03  5.177105e-04 -2.788709e-04
115  ... -1.775178e-16  8.271708e-17  1.483348e-16  1.154193e-16

              110           111           112       113           114
\
0    2.072628e-02  2.174453e-02  3.797818e-02  0.017441  7.706972e-03

1    2.029850e-02  7.788470e-03 -2.587156e-02  0.015851  1.698880e-01

2    3.298240e-02  2.074247e-02 -1.593448e-02  0.028776 -2.376671e-01

3    1.869812e-03 -9.527494e-03  2.224336e-03  0.002910  4.561041e-01

4    9.286294e-03  4.130888e-03  2.920407e-02  0.007779  2.590437e-01

..            ...           ...           ...       ...           ...

111 -6.153782e-03 -2.779847e-03  4.588464e-05  0.004112 -2.303411e-04

112 -2.918762e-03 -4.683143e-03 -3.177697e-04  0.001570 -2.231496e-04

113 -2.643777e-03 -2.526279e-03  9.641558e-04  0.001525 -5.307676e-04

114 -6.812465e-04  1.047745e-03 -3.259363e-05 -0.001107 -3.784001e-05

115  6.068535e-17 -3.209703e-17  1.808669e-17 -0.707107  2.086496e-17


              115
0    3.726726e-02
1    1.599543e-01
2   -2.576117e-01
3    4.312527e-01
4    2.584021e-01
..            ...
111  1.209085e-04
112  3.785547e-04
113  3.430654e-04
114  3.014726e-05
115 -1.636275e-17

[116 rows x 116 columns]
```

```
pd.DataFrame(pca_fit12.components_)

        0         1         2         3         4         5         6
\
0  0.137382  0.014244  0.011599  0.104623  0.000718  0.018959
0.005949
1 -0.102944  0.016509  0.013638 -0.097307  0.000683  0.019023
0.001581
2 -0.099205  0.022592  0.018152  0.005005  0.001000  0.031681
0.006525
3 -0.052665  0.000491  0.000766 -0.017471  0.000141 -0.000513 -
0.003639
4  0.122919  0.007036  0.005686  0.056929  0.000252  0.009470
0.000516
5 -0.136664  0.001830  0.001688 -0.020510  0.000169  0.002213
0.000467

        7         8         9    ...       106       107       108
109  \
0  0.002976  0.016021  0.069070  ...  0.017711  0.005455  0.023639
0.013352
1 -0.000254  0.005086  0.025201  ...  0.018044  0.005894  0.023743
0.013888
2  0.002827  0.016262  0.058710  ...  0.028659  0.008481  0.037225
0.021884
3 -0.001615 -0.010722 -0.021849  ...  0.001139  0.000674  0.001922
0.001255
4  0.000180  0.002736  0.023750  ...  0.007844  0.002931  0.010739
0.005429
5  0.000361  0.001447 -0.006237  ...  0.001346  0.000832  0.002468
0.001819

        110       111       112       113       114       115
0  0.020726  0.021745  0.037978  0.017441  0.007707  0.037267
1  0.020299  0.007788 -0.025872  0.015851  0.169888  0.159955
2  0.032982  0.020743 -0.015935  0.028776 -0.237667 -0.257610
3  0.001869 -0.009527  0.002224  0.002910  0.456103  0.431250
4  0.009281  0.004137  0.029201  0.007774  0.259038  0.258394
5  0.002896  0.003354 -0.001297  0.000320  0.069879  0.046836

[6 rows x 116 columns]

PC_values = np.arange(pca.n_components_) + 1
plt.plot(PC_values, pca.explained_variance_ratio_, 'o-', linewidth=2,
color='blue')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```



```
explained_variance_ratio = pca.explained_variance_ratio_

# Calculate cumulative explained variance ratio
cumulative_explained_variance_ratio =
explained_variance_ratio.cumsum()

print("Explained variance ratio for all component:",
sum(explained_variance_ratio))

# Print the explained variance ratio for each component
print("Explained variance ratio for each component:",
explained_variance_ratio)


# Print the cumulative explained variance ratio
print("Cumulative explained variance ratio:",
cumulative_explained_variance_ratio)
```

```
Explained variance ratio for all component: 1.0
Explained variance ratio for each component: [1.32567945e-01
8.04266572e-02 7.09118291e-02 5.95683487e-02
 5.04949937e-02 4.48684633e-02 3.14822833e-02 2.70797524e-02
 2.58612304e-02 2.37266904e-02 2.20166038e-02 2.14638717e-02
 2.02358341e-02 1.92712268e-02 1.77969374e-02 1.76495225e-02
 1.65787059e-02 1.61544953e-02 1.55819850e-02 1.50309018e-02
 1.35214999e-02 1.33990324e-02 1.29194003e-02 1.23611886e-02
 1.18621019e-02 1.13184802e-02 1.06696761e-02 9.99504872e-03
 9.65919278e-03 9.14702013e-03 8.77009747e-03 8.31529305e-03
 7.80500741e-03 7.41208754e-03 6.65461468e-03 6.29293220e-03
 6.10790544e-03 5.70003091e-03 5.53618886e-03 5.44233839e-03
 5.32057462e-03 4.90637714e-03 4.75299327e-03 4.46496400e-03
 4.07187966e-03 3.89717835e-03 3.71721368e-03 3.40863287e-03
 3.34388597e-03 3.12734654e-03 2.53253514e-03 2.42952885e-03
 2.34485795e-03 2.10073677e-03 1.98889028e-03 1.89456200e-03
 1.67739014e-03 1.66350047e-03 1.62960906e-03 1.56903486e-03
 1.46916516e-03 1.45067624e-03 1.42219991e-03 1.29617977e-03
 1.21937924e-03 1.12989249e-03 1.07721978e-03 9.82696869e-04
 8.86095705e-04 8.61193698e-04 8.33862084e-04 8.11355979e-04
 7.73693537e-04 7.70719504e-04 7.35539592e-04 7.23053288e-04
 7.03521370e-04 6.64797290e-04 6.05127347e-04 5.83977601e-04
 5.57091185e-04 4.98468846e-04 4.74844612e-04 4.68522323e-04
 4.58732084e-04 4.42850867e-04 4.34472031e-04 4.19173817e-04
 3.81803962e-04 3.77800395e-04 3.62673208e-04 3.43833932e-04
 3.38711072e-04 3.12994513e-04 3.03907445e-04 2.83124100e-04
 2.38349200e-04 2.33190461e-04 2.06720909e-04 1.99406529e-04
 1.68772924e-04 1.57694360e-04 1.30502065e-04 1.18391121e-04
 9.54648943e-05 8.65940602e-05 7.38379700e-05 6.63379838e-05
 5.65870686e-05 5.10781680e-05 4.47194769e-05 4.45461736e-05
 4.25059196e-05 2.31848801e-05 3.65854745e-06 1.09182134e-33]
Cumulative explained variance ratio: [0.13256794 0.2129946  0.28390643
0.34347478 0.39396977 0.43883824
 0.47032052 0.49740027 0.5232615  0.54698819 0.5690048  0.59046867
 0.6107045  0.62997573 0.64777267 0.66542219 0.6820009  0.69815539
 0.71373738 0.72876828 0.74228978 0.75568881 0.76860821 0.7809694
 0.7928315  0.80414998 0.81481966 0.82481471 0.8344739  0.84362092
 0.85239102 0.86070631 0.86851132 0.8759234  0.88257802 0.88887095
 0.89497886 0.90067889 0.90621508 0.91165741 0.91697799 0.92188437
 0.92663736 0.93110232 0.9351742  0.93907138 0.9427886  0.94619723
 0.94954111 0.95266846 0.955201   0.95763052 0.95997538 0.96207612
 0.96406501 0.96595957 0.96763696 0.96930046 0.97093007 0.97249911
 0.97396827 0.97541895 0.97684115 0.97813733 0.97935671 0.9804866
 0.98156382 0.98254652 0.98343261 0.98429381 0.98512767 0.98593902
 0.98671272 0.98748344 0.98821898 0.98894203 0.98964555 0.99031035
 0.99091548 0.99149945 0.99205654 0.99255501 0.99302986 0.99349838
 0.99395711 0.99439996 0.99483443 0.99525361 0.99563541 0.99601321
 0.99637589 0.99671972 0.99705843 0.99737143 0.99767533 0.99795846
 0.99819681 0.99843    0.99863672 0.99883612 0.9990049  0.99916259
 0.99929309 0.99941148 0.99950695 0.99959354 0.99966738 0.99973372
 0.99979031 0.99984139 0.9998861  0.99993065 0.99997316 0.99999634
 1.         1.        ]

# Assuming you have already performed PCA and have your pca_df
DataFrame
# And you have your original_variable_names available from your
```

```python
dummy_df

# Get the names of the original variables
original_variable_names = dummy_df.columns.tolist()

# Assuming you have your PCA models stored in pca_fit and pca_fit12
# pca_fit.components_ and pca_fit12.components_ contain the loadings (coefficients) of the original variables on each principal component
# You can print the top variables contributing to each principal component

# Now, let's print for pca_fit12 (with 6 components)
print("\nPrinting top variables for pca_fit12 (with 6 components):")
for i in range(len(pca_fit12.components_)):
    component_loadings = pca_fit12.components_[i]  # Loadings of the i-th principal component
    top_variable_indices = component_loadings.argsort()[::-1][:6]  # Get the indices of top 6 variables
    top_variables = [original_variable_names[idx] for idx in top_variable_indices]  # Get variable names
print(f"Principal Component {i+1} Top Variables:", top_variables)
```

```
Printing top variables for pca_fit12 (with 6 components):
Principal Component 6 Top Variables: ['Carbon monoxide detector',
'Fire extinguisher', 'First aid kit', 'Smoke detector', 'Safety card',
'Lock on bedroom door']
```

```python
# Assuming data_cleaned is your DataFrame containing the cleaned data
df_cleaned = data_cleaned.copy()  # Make a copy to avoid modifying the original DataFrame

# Reset the index to start from 1 to 32,349
df_cleaned.reset_index(drop=True, inplace=True)
df_cleaned.index += 0
df_cleaned.index.name = 'id'

# Create a new DataFrame to append the modified data
new_df = pd.DataFrame()

# Append the modified DataFrame to the new DataFrame
new_df = pd.concat([new_df, df_cleaned])

# Now new_df contains the modified data with IDs starting from 1 to 32,349
data_cleaned=new_df.copy()

# Load PCA results
pca_results = pd.DataFrame(pca_fit12.components_)
```

```
# Merge PCA results with original data
result = pd.concat([new_df, pca_df], axis=1)

data_cleaned1=pd.read_csv("C:/Users/navab/Downloads/data_cleaned.csv")

draft_result = pd.concat([data_cleaned1, pca_df], axis=1)

#change pca values when rerunnning the code

np.random.seed(42)

X = result[[0,1,2,3,4,5,
'room_type','bathrooms','bedrooms','accommodates','cleaning_fee','Boro
ugh']] # Dropping irrelevant columns
y = result['log_price']

# Load the new Excel file
new_data =
pd.read_excel('C:/Users/navab/OneDrive/Documents/masterclass/append.xl
sx')
#C:/Users/navab/OneDrive/Documents/masterclass/append


# Assuming the new columns you want to append are named 'column1' and
'column2'
new_columns =
new_data[['avg_lead_time','min_lead_time','min_airport_time',
'avg_airport_time','man_time',

'bro_time','que_time','bnx_time','sta_time','nj_time']]

# Concatenate the new columns with the existing features matrix X
X = pd.concat([X, new_columns], axis=1)

# Now X contains the additional columns from the new dataset
eda = pd.concat([X, y], axis=1)
```

Test Train Split

```
# Split the data into train and test sets (60/20/20)
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

X_train.dtypes
```

```
0                    float64
1                    float64
2                    float64
```

```
3                    float64
4                    float64
5                    float64
room_type             object
bathrooms            float64
bedrooms             float64
accommodates           int64
cleaning_fee            bool
Borough               object
avg_lead_time        float64
min_lead_time        float64
min_airport_time     float64
avg_airport_time     float64
man_time             float64
bro_time             float64
que_time             float64
bnx_time             float64
sta_time             float64
nj_time              float64
dtype: object
```

One Hot Encoding

```
#identifying numerical and categorial columns for one-hot encoding
numerical_columns = ['0','1','2','3','4','5','bathrooms', 'bedrooms',
'accommodates', 'cleaning_fee',
                  'avg_lead_time', 'min_lead_time',
'min_airport_time',
                  'avg_airport_time', 'man_time', 'bro_time',
'que_time',
                  'bnx_time', 'sta_time', 'nj_time']
categorical_columns = ['room_type','Borough']
```

Defining Model Evaluation Parameters

```
# Calculate RMSE and R2 for each model
def evaluate_model(y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)
    return rmse, r2
```

Preprocessing Pipeline

```
# Define preprocessing steps for numerical and categorical columns
categorical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Combine preprocessing pipelines using ColumnTransformer
preprocessor = ColumnTransformer([
    ('cat', categorical_pipeline, categorical_columns),
    ('num', numerical_pipeline, numerical_columns)
])
```

1. Linear Regression

```
# Define the Linear model
linear_model = LinearRegression()

X_train.columns = X_train.columns.astype(str)

preprocessor.fit(X_train)
```

```
ColumnTransformer(transformers=[('cat',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['room_type', 'Borough']),
                                ('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['0', '1', '2', '3', '4', '5',
'bathrooms',
                                  'bedrooms', 'accommodates',
'cleaning_fee',
                                  'avg_lead_time', 'min_lead_time',
                                  'min_airport_time',
'avg_airport_time',
                                  'man_time', 'bro_time', 'que_time',
                                  'bnx_time', 'sta_time',
'nj_time'])])
```

```
# Transform the training data using the fitted preprocessor
X_train_transformed = preprocessor.transform(X_train)

# Fit the models
linear_model.fit(X_train_transformed, y_train)
```

```
LinearRegression()
```

Predictions - Train

```
# 1. Training dataset -
linear_pred_train = linear_model.predict(X_train_transformed)

linear_train_rmse, linear_train_r2 = evaluate_model(y_train,
linear_pred_train)

# Evaluating Performance
print("Linear Regression: \nRMSE =", linear_train_rmse, "\nR2 =",
linear_train_r2)
```

```
Linear Regression:
RMSE = 0.3864768127901715
R2 = 0.6586358526376852
```

```
X_val.columns = X_val.columns.astype(str)

preprocessor.fit(X_val)
```

```
ColumnTransformer(transformers=[('cat',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['room_type', 'Borough']),
                                ('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['0', '1', '2', '3', '4', '5',
'bathrooms',
                                  'bedrooms', 'accommodates',
'cleaning_fee',
                                  'avg_lead_time', 'min_lead_time',
                                  'min_airport_time',
'avg_airport_time',
                                  'man_time', 'bro_time', 'que_time',
                                  'bnx_time', 'sta_time',
'nj_time'])])
```

```
# Transform the training data using the fitted preprocessor
X_val_transformed = preprocessor.transform(X_val)

# Fit the models
linear_model.fit(X_val_transformed, y_val)
```

```
LinearRegression()
```

```
# 2. Validation dataset -
linear_pred_val = linear_model.predict(X_val_transformed)

linear_val_rmse, linear_val_r2 = evaluate_model(y_val,
linear_pred_val)

# Evaluating Performance
print("Linear Regression: \nRMSE =", linear_val_rmse, "\nR2 =",
linear_val_r2)
```

```
Linear Regression:
RMSE = 0.38930113695805324
R2 = 0.6548522412933646
```

Prediction Test

```
X_test.columns = X_test.columns.astype(str)

preprocessor.fit(X_test)
```

```
ColumnTransformer(transformers=[('cat',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['room_type', 'Borough']),
                                ('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['0', '1', '2', '3', '4', '5',
'bathrooms',
                                  'bedrooms', 'accommodates',
'cleaning_fee',
                                  'avg_lead_time', 'min_lead_time',
                                  'min_airport_time',
'avg_airport_time',
                                  'man_time', 'bro_time', 'que_time',
                                  'bnx_time', 'sta_time',
'nj_time'])])
```

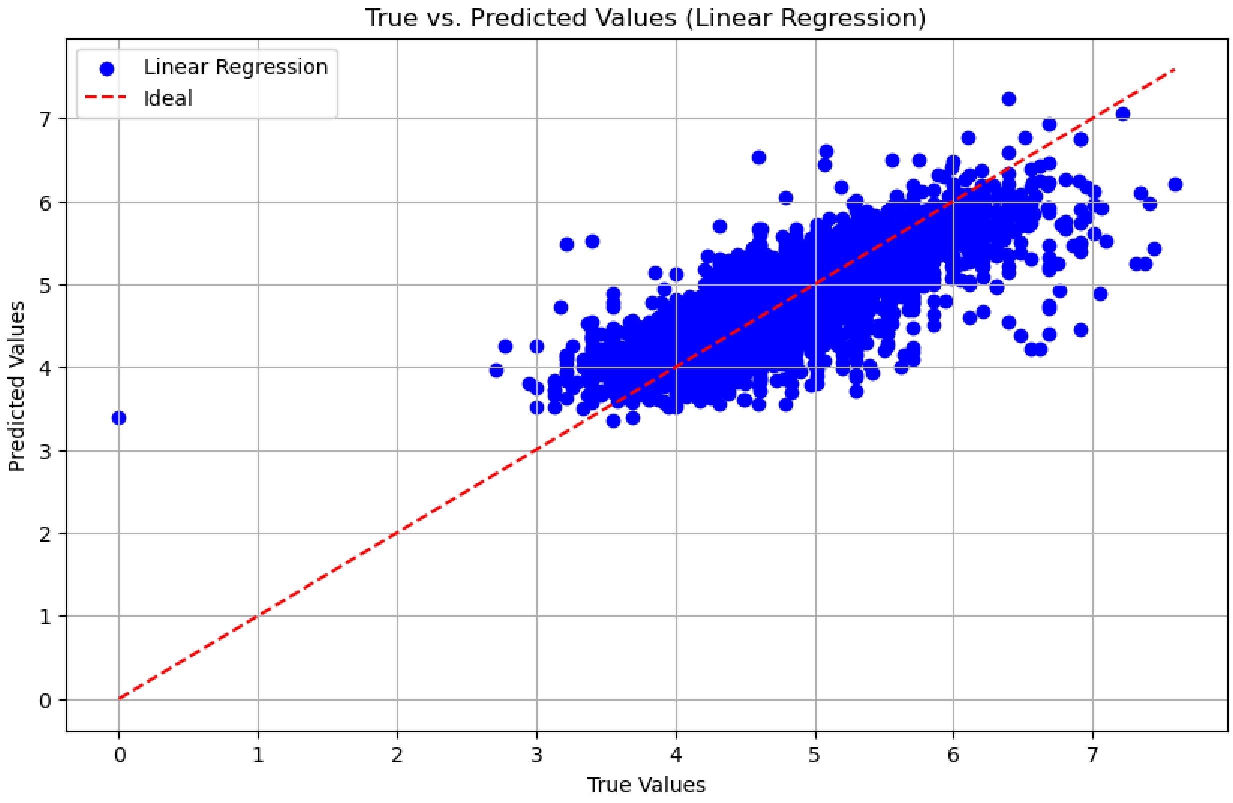
```
# Transform the training data using the fitted preprocessor
X_test_transformed = preprocessor.transform(X_test)

# Fit the models
linear_model.fit(X_test_transformed, y_test)
```

```
LinearRegression()
```

```
# 1. Training dataset -
linear_pred_test = linear_model.predict(X_test_transformed)

linear_test_rmse, linear_test_r2 = evaluate_model(y_test,
linear_pred_test)

# Evaluating Performance
print("Linear Regression: \nRMSE =", linear_test_rmse, "\nR2 =",
linear_test_r2)
```

```
Linear Regression:
RMSE = 0.3774101110088497
R2 = 0.6736818769679825
```

```
# Plot true vs. predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, linear_pred_test, color='blue', label='Linear
Regression')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label='Ideal')
plt.title('True vs. Predicted Values (Linear Regression)')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



2. Ridge Regression

```
param_grid_ridge = {'ridge__alpha': [0.1, 1.0, 10.0]}

pipeline_ridge = Pipeline([
    ('preprocessor', preprocessor),
    ('ridge', Ridge())
])

# Define grid search for each model
grid_search_ridge = GridSearchCV(pipeline_ridge, param_grid_ridge,
cv=5, scoring='neg_mean_squared_error')
```

Predictions : Train

```
# Convert column names to strings
X_train.columns = X_train.columns.astype(str)

grid_search_ridge.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',
'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ridge', Ridge())]),
             param_grid={'ridge__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')
```

```
#get the best estimators
best_ridge = grid_search_ridge.best_estimator_

# Predictions for Ridge and Lasso
ridge_pred = best_ridge.predict(X_train)

# Calculate RMSE and R2 for Ridge and Lasso
ridge_train_rmse, ridge_train_r2 = evaluate_model(y_train, ridge_pred)

print("Ridge: \nRMSE =", ridge_train_rmse, "R2 =", ridge_train_r2)
```

```
Ridge: 
RMSE = 0.3864746582212765 R2 = 0.6586396587678295
```

Prediction Val

```
# Convert column names to strings
X_val.columns = X_val.columns.astype(str)

grid_search_ridge.fit(X_val, y_val)
```

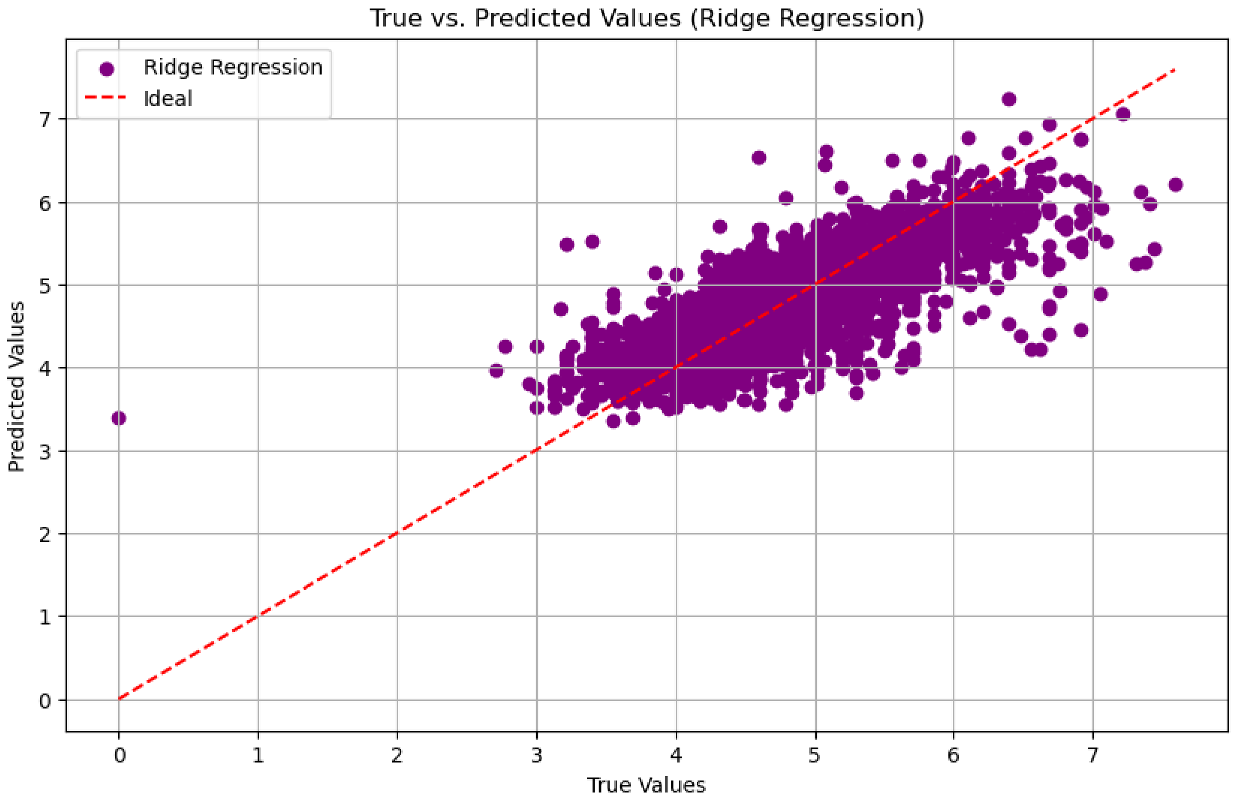
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
['room_type',
'Borough']),
('num',
Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
('scaler',
StandardScaler())]),
['0',
'1',
'2',
'3',
'4',
'5',
'bathrooms',
'bedrooms',
'accommodates',
'cleaning_fee',
'avg_lead_time',
'min_lead_time',
'min_airport_time',
'avg_airport_time',
'man_time',
'bro_time',
'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ridge', Ridge())]),
             param_grid={'ridge__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

#get the best estimators
best_ridge = grid_search_ridge.best_estimator_

# Predictions for Ridge and Lasso
ridge_pred = best_ridge.predict(X_val)

# Calculate RMSE and R2 for Ridge and Lasso
ridge_val_rmse, ridge_val_r2 = evaluate_model(y_val, ridge_pred)

print("Ridge: \nRMSE =", ridge_val_rmse, "R2 =", ridge_val_r2)

Ridge:
RMSE = 0.38930444603028147 R2 = 0.6548463737342511
```

Prediction Test

```
# Convert column names to strings
X_test.columns = X_test.columns.astype(str)

grid_search_ridge.fit(X_test, y_test)

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),
('num',
Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
('scaler',
StandardScaler())]),
['0',
'1',
'2',
'3',
'4',
'5',
'bathrooms',
'bedrooms',
'accommodates',
'cleaning_fee',
'avg_lead_time',
'min_lead_time',
'min_airport_time',
'avg_airport_time',
'man_time',
'bro_time',
'que_time',
'bnx_time',
'sta_time',
'nj_time'])])),
                                       ('ridge', Ridge())]),
             param_grid={'ridge__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

#get the best estimators
best_ridge = grid_search_ridge.best_estimator_

# Predictions for Ridge and Lasso
ridge_pred = best_ridge.predict(X_test)

# Calculate RMSE and R2 for Ridge and Lasso
ridge_test_rmse, ridge_test_r2 = evaluate_model(y_test, ridge_pred)

print("Ridge: \nRMSE =", ridge_test_rmse, "R2 =", ridge_test_r2)
```

```
Ridge:
RMSE = 0.37741497743634494 R2 = 0.6736734616464113
```

```
# Plot true vs. predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, ridge_pred, color='purple', label='Ridge
Regression')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label='Ideal')
plt.title('True vs. Predicted Values (Ridge Regression)')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



3. Lasso

```
param_grid_lasso = {'lasso__alpha': [0.1, 1.0, 10.0]}

pipeline_lasso = Pipeline([
    ('preprocessor', preprocessor),
    ('lasso', Lasso())
])

grid_search_lasso = GridSearchCV(pipeline_lasso, param_grid_lasso,
cv=5, scoring='neg_mean_squared_error')
```

Prediction Train

```
# Convert column names to strings
X_train.columns = X_train.columns.astype(str)

grid_search_lasso.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',
'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('lasso', Lasso())]),
             param_grid={'lasso__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

best_lasso = grid_search_lasso.best_estimator_

lasso_pred = best_lasso.predict(X_train)

lasso_train_rmse, lasso_train_r2 = evaluate_model(y_train, lasso_pred)

print("Lasso: \nRMSE =", lasso_train_rmse, "R2 =", lasso_train_r2)

Lasso:
RMSE = 0.4703885821723464 R2 = 0.49430976965207496
```

Prediction val

```
# Convert column names to strings
X_train.columns = X_train.columns.astype(str)

grid_search_lasso.fit(X_train, y_train)

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
['room_type',
'Borough']),
('num',
Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
('scaler',
StandardScaler())]),
['0',
'1',
'2',
'3',
'4',
'5',
'bathrooms',
'bedrooms',
'accommodates',
'cleaning_fee',
'avg_lead_time',
'min_lead_time',
'min_airport_time',
'avg_airport_time',
'man_time',
'bro_time',
'que_time',
'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('lasso', Lasso())]),
             param_grid={'lasso__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')
```

```
best_lasso = grid_search_lasso.best_estimator_

lasso_pred = best_lasso.predict(X_train)

lasso_train_rmse, lasso_train_r2 = evaluate_model(y_train, lasso_pred)

print("Ridge: \nRMSE =", ridge_train_rmse, "R2 =", ridge_train_r2)
```

```
Ridge:
RMSE = 0.3864746582212765 R2 = 0.6586396587678295
```

Prediction Test

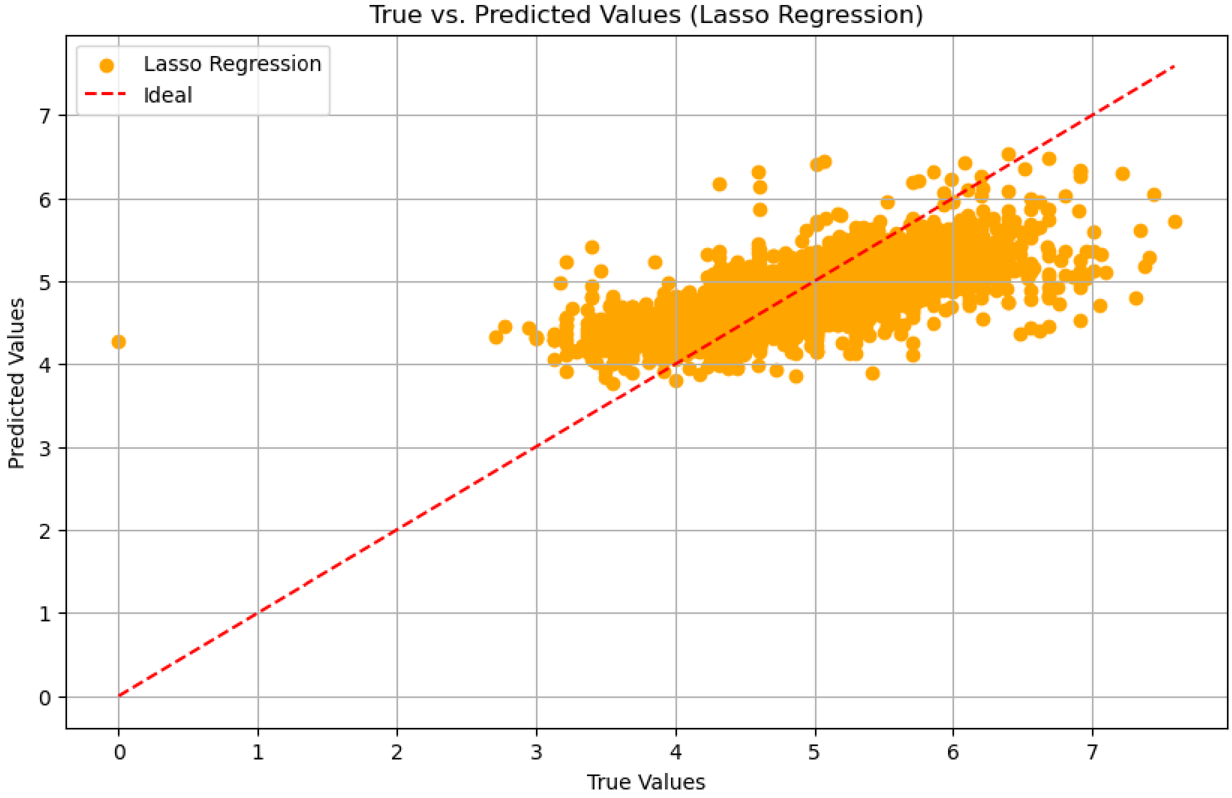
```
# Convert column names to strings
X_test.columns = X_test.columns.astype(str)

grid_search_lasso.fit(X_test, y_test)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('lasso', Lasso())]),
             param_grid={'lasso__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

best_lasso = grid_search_lasso.best_estimator_
```

```python
lasso_pred = best_lasso.predict(X_test)

lasso_test_rmse, lasso_test_r2 = evaluate_model(y_test, lasso_pred)

print("Ridge: \nRMSE =", lasso_test_rmse, "R2 =", lasso_test_r2)
```

```
Ridge:
RMSE = 0.4677051547103826 R2 = 0.4988607194124248
```

```python
# Plot true vs. predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, lasso_pred, color='orange', label='Lasso
Regression')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label='Ideal')
plt.title('True vs. Predicted Values (Lasso Regression)')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



4.Random Forest

```
param_grid_random_forest = {'random_forest__n_estimators': [50, 100, 
150],
                            'random_forest__max_depth': [1,2,3,4]}

pipeline_random_forest = Pipeline([
    ('preprocessor', preprocessor),
    ('random_forest', RandomForestRegressor())
])

grid_search_random_forest = GridSearchCV(pipeline_random_forest, 
param_grid_random_forest, cv=5, scoring='neg_mean_squared_error')
```

Prediction Train

```
X_train.columns = X_train.columns.astype(str)

grid_search_random_forest.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',
'1'...

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                         ('random_forest',
                                          RandomForestRegressor())]),
             param_grid={'random_forest__max_depth': [1, 2, 3, 4],
                         'random_forest__n_estimators': [50, 100,
150]},
             scoring='neg_mean_squared_error')
```

```
best_random_forest = grid_search_random_forest.best_estimator_

rf_pred = best_random_forest.predict(X_train)

rf_train_rmse, rf_train_r2 = evaluate_model(y_train, rf_pred)

print("Random Forest: \nRMSE =", rf_train_rmse, "R2 =", rf_train_r2)
```

```
Random Forest:
RMSE = 0.4065722567614497 R2 = 0.6222134420451775
```

Prediction val

```
X_val.columns = X_val.columns.astype(str)
```

```
grid_search_random_forest.fit(X_val, y_val)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('cat',
Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
['room_type',
'Borough']),
('num',
Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
('scaler',
StandardScaler())]),
['0',
'1'...
'bedrooms',
'accommodates',
'cleaning_fee',
'avg_lead_time',
'min_lead_time',
'min_airport_time',
'avg_airport_time',
'man_time',
'bro_time',
'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('random_forest',
                                        RandomForestRegressor())]),
             param_grid={'random_forest__max_depth': [1, 2, 3, 4],
                         'random_forest__n_estimators': [50, 100,
150]},
             scoring='neg_mean_squared_error')

best_random_forest = grid_search_random_forest.best_estimator_

rf_pred = best_random_forest.predict(X_val)

rf_val_rmse, rf_val_r2 = evaluate_model(y_val, rf_pred)

print("Random Forest: \nRMSE =", rf_val_rmse, "R2 =", rf_val_r2)
```

```
Random Forest:
RMSE = 0.403494942162454 R2 = 0.6292254608748855
```

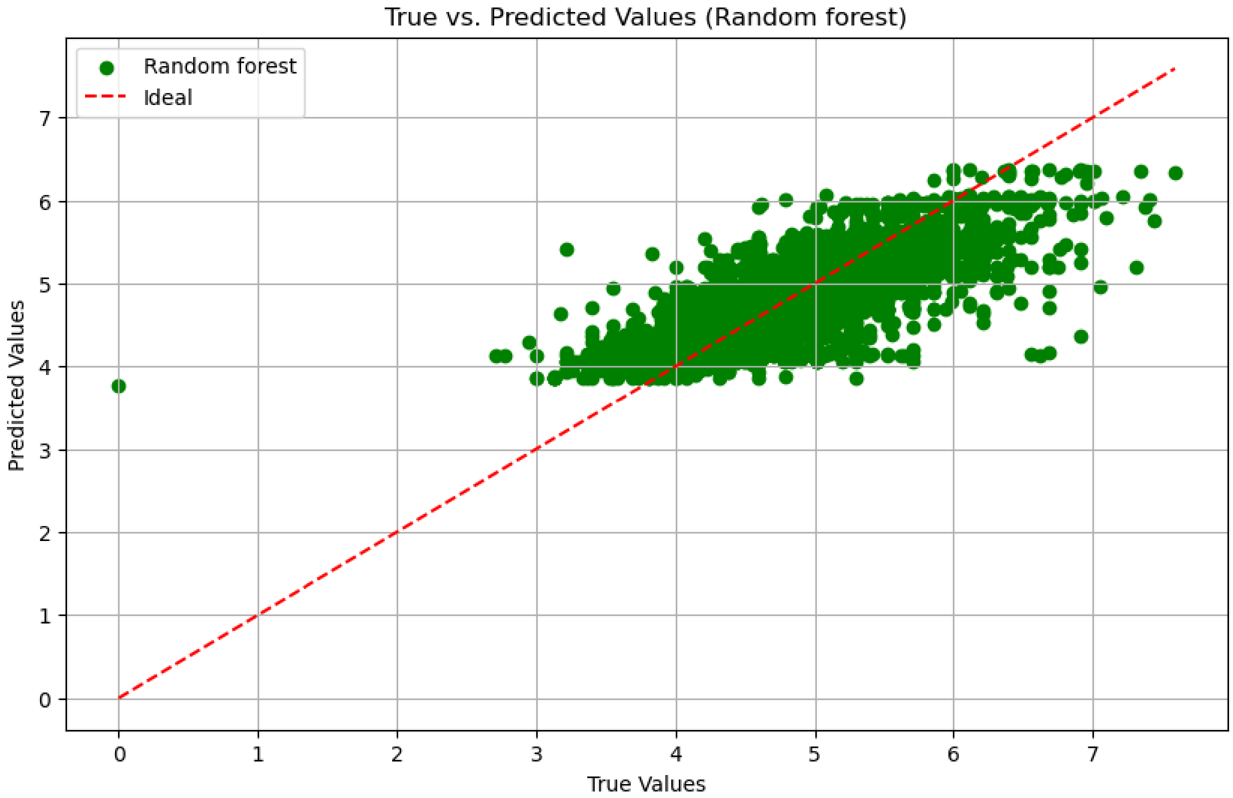
Prediction Test

```
X_test.columns = X_test.columns.astype(str)

grid_search_random_forest.fit(X_test, y_test)

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),
('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('random_forest',
                                        RandomForestRegressor())]),
             param_grid={'random_forest__max_depth': [1, 2, 3, 4],
                         'random_forest__n_estimators': [50, 100,
150]},
             scoring='neg_mean_squared_error')

best_random_forest = grid_search_random_forest.best_estimator_

rf_pred = best_random_forest.predict(X_test)
```

```
rf_test_rmse, rf_test_r2 = evaluate_model(y_test, rf_pred)

print("Random Forest: \nRMSE =", rf_test_rmse, "R2 =", rf_test_r2)
```

```
Random Forest:
RMSE = 0.3915295827792412 R2 = 0.6488090664043298
```

```
# Plot true vs. predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, rf_pred, color='green', label='Random forest')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label='Ideal')
plt.title('True vs. Predicted Values (Random forest)')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



5. XGradient Boosting

```
param_grid_xgb = {'xgb__n_estimators': [50, 100, 150],
                  'xgb__learning_rate': [0.05, 0.1, 0.2],
                 'xgb__max_depth': [3, 5, 7]}
```

```
pipeline_xgb = Pipeline([
    ('preprocessor', preprocessor),
    ('xgb', XGBRegressor())
])

grid_search_xgb = GridSearchCV(pipeline_xgb, param_grid_xgb, cv=5,
scoring='neg_mean_squared_error')
```

Prediction Train

```
# Convert column names to strings
X_train.columns = X_train.columns.astype(str)

grid_search_xgb.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

max_delta_step=None,
                                                     max_depth=None,
                                                     max_leaves=None,
min_child_weight=None,
                                                     missing=nan,
monotone_constraints=None,
multi_strategy=None,
n_estimators=None,
                                                     n_jobs=None,
num_parallel_tree=None,
random_state=None, ...))]),
             param_grid={'xgb__learning_rate': [0.05, 0.1, 0.2],
                         'xgb__max_depth': [3, 5, 7],
                         'xgb__n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')

best_xgb = grid_search_xgb.best_estimator_

xgb_pred = best_xgb.predict(X_train)

xgb_train_rmse, xgb_train_r2 = evaluate_model(y_train, xgb_pred)

print("XGBoost: \nRMSE =", xgb_train_rmse, "R2 =", xgb_train_r2)

XGBoost:
RMSE = 0.33748424812503847 R2 = 0.7396976824241108
```

Prediction val

```
# Convert column names to strings
X_val.columns = X_val.columns.astype(str)

grid_search_xgb.fit(X_val, y_val)

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

max_delta_step=None,
                                                     max_depth=None,
                                                     max_leaves=None,

min_child_weight=None,
                                                     missing=nan,

monotone_constraints=None,

multi_strategy=None,

n_estimators=None,
                                                     n_jobs=None,

num_parallel_tree=None,

random_state=None, ...))]),
             param_grid={'xgb__learning_rate': [0.05, 0.1, 0.2],
                         'xgb__max_depth': [3, 5, 7],
                         'xgb__n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')
```

```
best_xgb = grid_search_xgb.best_estimator_

xgb_pred = best_xgb.predict(X_val)

xgb_val_rmse, xgb_val_r2 = evaluate_model(y_val, xgb_pred)

print("XGBoost: \nRMSE =", xgb_val_rmse, "R2 =", xgb_val_r2)
```

```
XGBoost: 
RMSE = 0.30957222186827077 R2 = 0.7817482765706563
```

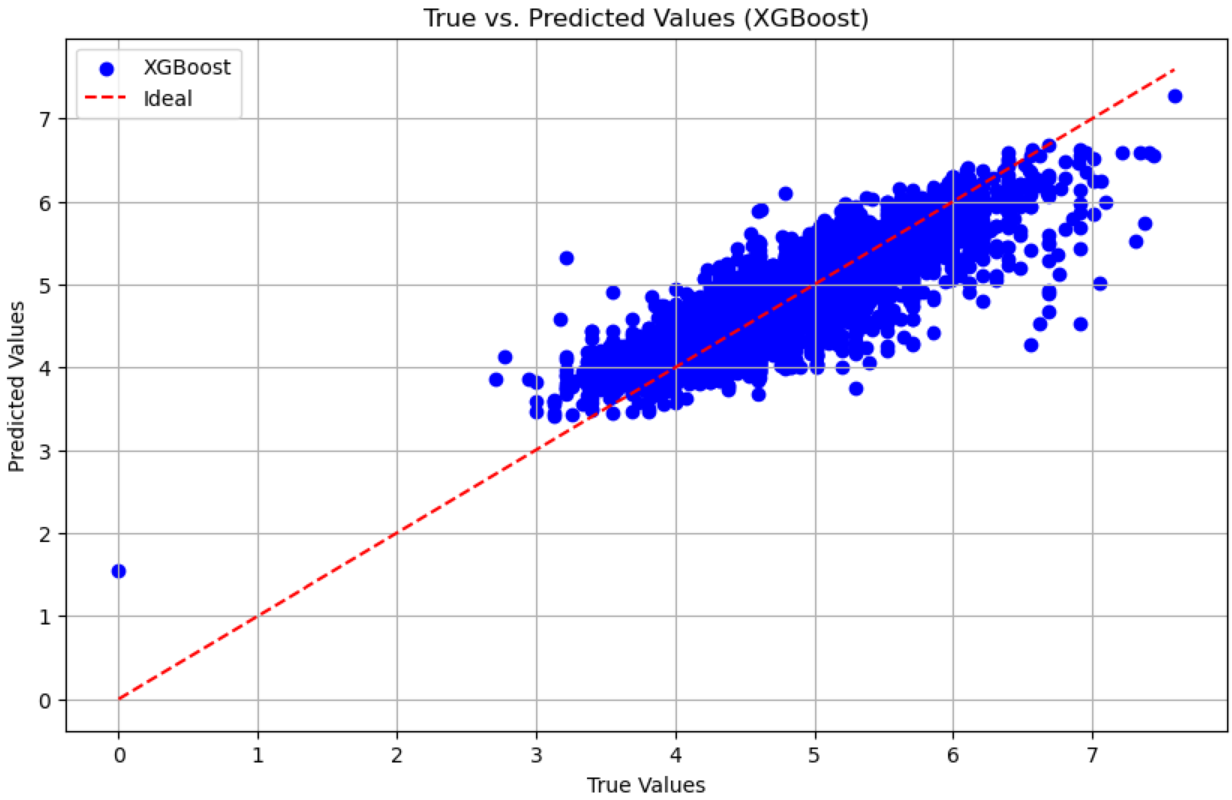
Prediction Test

```
# Convert column names to strings
X_test.columns = X_test.columns.astype(str)

grid_search_xgb.fit(X_test, y_test)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

max_delta_step=None,
                                                     max_depth=None,
                                                     max_leaves=None,

min_child_weight=None,
                                                     missing=nan,

monotone_constraints=None,

multi_strategy=None,

n_estimators=None,
                                                     n_jobs=None,

num_parallel_tree=None,

random_state=None, ...))]),
             param_grid={'xgb__learning_rate': [0.05, 0.1, 0.2],
                         'xgb__max_depth': [3, 5, 7],
                         'xgb__n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')

best_xgb = grid_search_xgb.best_estimator_

xgb_pred = best_xgb.predict(X_test)

xgb_test_rmse, xgb_test_r2 = evaluate_model(y_test, xgb_pred)

print("XGBoost: \nRMSE =", xgb_test_rmse, "R2 =", xgb_test_r2)
```

```
XGBoost:
RMSE = 0.3307196079035813 R2 = 0.7494271451006356
```

```
# Plot true vs. predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, xgb_pred, color='blue', label='XGBoost')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label='Ideal')
plt.title('True vs. Predicted Values (XGBoost)')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



ANN

```
# Define hyperparameter grid for ANN
param_grid_ann = {
    'ann__hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'ann__activation': ['relu', 'tanh', 'logistic'],
    'ann__alpha': [0.0001, 0.001, 0.01],
}

# Define ANN pipeline
pipeline_ann = Pipeline([
    ('preprocessor', preprocessor),
    ('ann', MLPRegressor(max_iter=500))
])

# Grid search for ANN
grid_search_ann = GridSearchCV(pipeline_ann, param_grid_ann, cv=5,
scoring='neg_mean_squared_error')
```

Prediction Train

```python
# Convert column names to strings
X_train.columns = X_train.columns.astype(str)

# Fit the grid search object
grid_search_ann.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',
'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ann',
MLPRegressor(max_iter=500))]),
             param_grid={'ann__activation': ['relu', 'tanh',
'logistic'],
                         'ann__alpha': [0.0001, 0.001, 0.01],
                         'ann__hidden_layer_sizes': [(50,), (100,),
(50, 50)]},
             scoring='neg_mean_squared_error')
```

```
# Get the best estimator
best_ann = grid_search_ann.best_estimator_

# Evaluate the model
ann_train_rmse, ann_train_r2 =
evaluate_model(best_ann.predict(X_train), y_train)

# Print RMSE and R2 score for ANN
print("ANN: \nRMSE =", ann_train_rmse, "R2 =", ann_train_r2)
```

```
ANN:
RMSE = 0.3780810691942245 R2 = 0.5053932308349539
```

Prediction val

```
# Convert column names to strings
X_val.columns = X_val.columns.astype(str)

# Fit the grid search object
grid_search_ann.fit(X_val, y_val)
```

```
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',
Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ann',
MLPRegressor(max_iter=500))]),
             param_grid={'ann__activation': ['relu', 'tanh',
'logistic'],
                         'ann__alpha': [0.0001, 0.001, 0.01],
                         'ann__hidden_layer_sizes': [(50,), (100,),
(50, 50)]},
             scoring='neg_mean_squared_error')

# Get the best estimator
best_ann = grid_search_ann.best_estimator_

# Evaluate the model
ann_val_rmse, ann_val_r2 = evaluate_model(best_ann.predict(X_val),
y_val)

# Print RMSE and R2 score for ANN
print("ANN: \nRMSE =", ann_val_rmse, "R2 =", ann_val_r2)
```

```
ANN:
RMSE = 0.36788029629931757 R2 = 0.5492192341727702
```

Prediction Test

```
# Convert column names to strings
X_test.columns = X_test.columns.astype(str)

# Fit the grid search object
grid_search_ann.fit(X_test, y_test)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
('scaler',

StandardScaler())]),

['0',

'1'...

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ann',
MLPRegressor(max_iter=500))]),
             param_grid={'ann__activation': ['relu', 'tanh',
'logistic'],
                         'ann__alpha': [0.0001, 0.001, 0.01],
                         'ann__hidden_layer_sizes': [(50,), (100,),
(50, 50)]},
             scoring='neg_mean_squared_error')

# Get the best estimator
best_ann = grid_search_ann.best_estimator_

# Evaluate the model
ann_test_rmse, ann_test_r2 = evaluate_model(best_ann.predict(X_test),
y_test)

# Print RMSE and R2 score for ANN
print("ANN: \nRMSE =", ann_test_rmse, "R2 =", ann_test_r2)

ANN:
RMSE = 0.3645379432371982 R2 = 0.5668729192849231
```

SVR

```
param_grid_svr = {'svr__C': [0.1, 1.0, 10.0],
                  'svr__kernel': ['linear', 'rbf']}

pipeline_svr = Pipeline([
    ('preprocessor', preprocessor),
    ('svr', SVR())
])

grid_search_svr = GridSearchCV(pipeline_svr, param_grid_svr, cv=5,
scoring='neg_mean_squared_error')
```

Prediction Train

```
# Convert column names to strings
X_train.columns = X_train.columns.astype(str)

grid_search_svr.fit(X_train, y_train)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',
'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('svr', SVR())]),
             param_grid={'svr__C': [0.1, 1.0, 10.0],
                         'svr__kernel': ['linear', 'rbf']},
             scoring='neg_mean_squared_error')

best_svr = grid_search_svr.best_estimator_

svr_pred = best_svr.predict(X_train)

svr_train_rmse, svr_train_r2 = evaluate_model(y_train, svr_pred)

print("SVR: \nRMSE =", svr_train_rmse, "R2 =", svr_train_r2)
```

```
SVR:
RMSE = 0.34010209839101985 R2 = 0.7356437129969223
```

Prediction val

```
# Convert column names to strings
X_val.columns = X_val.columns.astype(str)

grid_search_svr.fit(X_val, y_val)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',
'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('svr', SVR())]),
             param_grid={'svr__C': [0.1, 1.0, 10.0],
                         'svr__kernel': ['linear', 'rbf']},
             scoring='neg_mean_squared_error')

best_svr = grid_search_svr.best_estimator_

svr_pred = best_svr.predict(X_val)

svr_val_rmse, svr_val_r2 = evaluate_model(y_val, svr_pred)

print("SVR: \nRMSE =", svr_val_rmse, "R2 =", svr_val_r2)
```

```
SVR:
RMSE = 0.3336428977796782 R2 = 0.7464886065606153
```

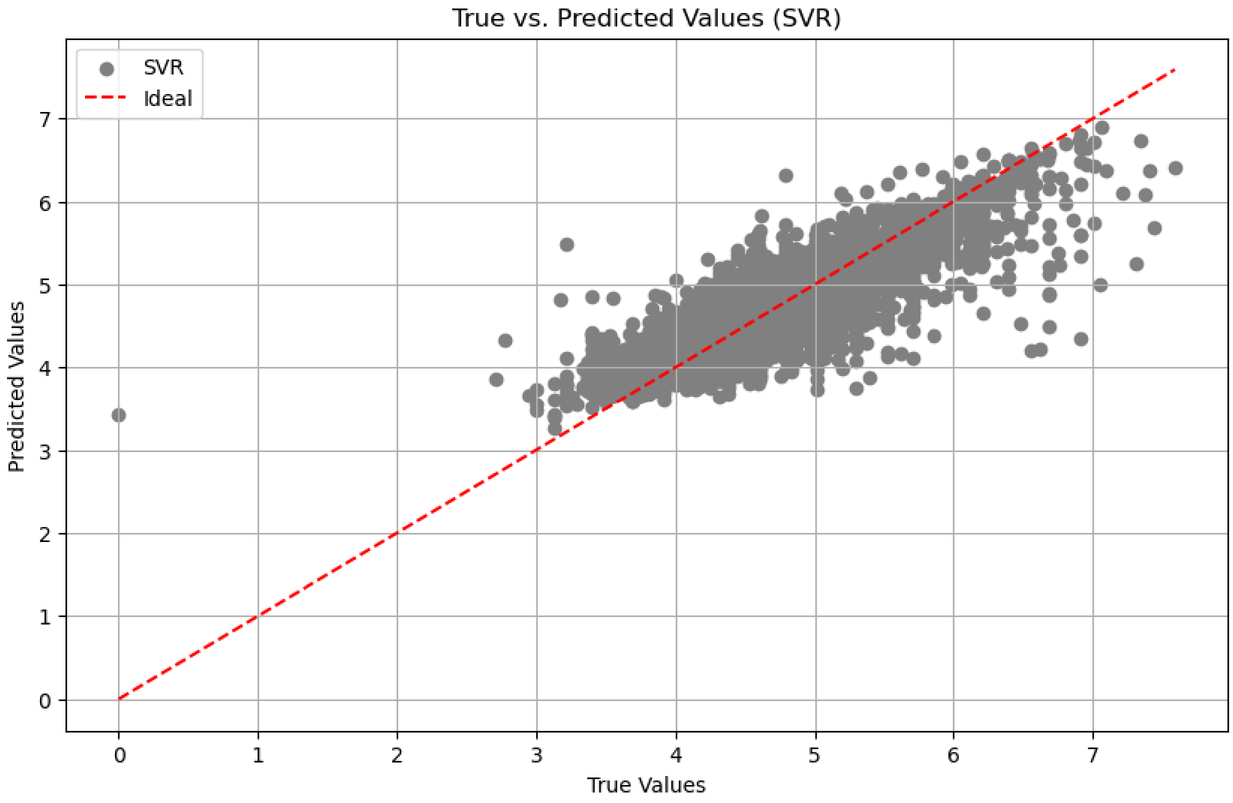
Prediction test (log price)

```
# Convert column names to strings
X_test.columns = X_test.columns.astype(str)
```

```
grid_search_svr.fit(X_test, y_test)

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',
'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('svr', SVR())]),
             param_grid={'svr__C': [0.1, 1.0, 10.0],
                         'svr__kernel': ['linear', 'rbf']},
             scoring='neg_mean_squared_error')

best_svr = grid_search_svr.best_estimator_

svr_pred = best_svr.predict(X_test)

svr_test_rmse, svr_test_r2 = evaluate_model(y_test, svr_pred)

print("SVR: \nRMSE =", svr_test_rmse, "R2 =", svr_test_r2)
```

```
SVR:
RMSE = 0.3189918346908839 R2 = 0.7668833651162165
```

```
# Plot true vs. predicted values for Linear Regression
plt.figure(figsize=(10, 6))
plt.scatter(y_test, svr_pred, color='grey', label='SVR')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--',
color='red', label='Ideal')
plt.title('True vs. Predicted Values (SVR)')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```



For normal price

1. Linear regression

Train

```
#data_cleaned['Price'] = np.exp(data_cleaned['log_price'])



# Define the Linear model
linear_model1 = LinearRegression()

X_train.columns = X_train.columns.astype(str)

preprocessor.fit(X_train)

ColumnTransformer(transformers=[('cat',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['room_type', 'Borough']),
                                ('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['0', '1', '2', '3', '4', '5',
'bathrooms',
                                  'bedrooms', 'accommodates',
'cleaning_fee',
                                  'avg_lead_time', 'min_lead_time',
                                  'min_airport_time',
'avg_airport_time',
                                  'man_time', 'bro_time', 'que_time',
                                  'bnx_time', 'sta_time',
'nj_time'])])
```

```
# Transform the training data using the fitted preprocessor
X_train_transformed = preprocessor.transform(X_train)

linear_model1.fit(X_train_transformed, np.exp(y_train))
```

```
LinearRegression()
```

```
# 1. Training dataset -
linear_pred_train1 = linear_model1.predict(X_train_transformed)

linear_train_np_rmse, linear_train_np_r2 =
evaluate_model(np.exp(y_train), linear_pred_train1)

# Evaluating Performance
print("Linear Regression: \nRMSE =", linear_train_np_rmse, "\nR2 =",
linear_train_np_r2)
```

```
Linear Regression:
RMSE = 95.8973531889311
R2 = 0.46342135123325756
```

val

```
X_val.columns = X_val.columns.astype(str)

preprocessor.fit(X_val)
```

```
ColumnTransformer(transformers=[('cat',
                                 Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['room_type', 'Borough']),
                                ('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['0', '1', '2', '3', '4', '5',
'bathrooms',
                                  'bedrooms', 'accommodates',
'cleaning_fee',
                                  'avg_lead_time', 'min_lead_time',
                                  'min_airport_time',
'avg_airport_time',
                                  'man_time', 'bro_time', 'que_time',
                                  'bnx_time', 'sta_time',
'nj_time'])])
```

```
# Transform the training data using the fitted preprocessor
X_val_transformed = preprocessor.transform(X_val)
```

```
linear_model1.fit(X_val_transformed, np.exp(y_val))
```

```
LinearRegression()
```

```
linear_pred_val1 = linear_model1.predict(X_val_transformed)
```

```
linear_val_np_rmse, linear_val_np_r2 = evaluate_model(np.exp(y_val),
linear_pred_val1)
```

```
# Evaluating Performance
print("Linear Regression: \nRMSE =", linear_val_np_rmse, "\nR2 =",
linear_val_np_r2)
```

```
Linear Regression:
RMSE = 92.93664566042034
R2 = 0.5108800972728177
```

Test

```
X_test.columns = X_test.columns.astype(str)
```

```
preprocessor.fit(X_test)
```

```
ColumnTransformer(transformers=[('cat',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
                                                 ('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
                                 ['room_type', 'Borough']),
                                ('num',
                                 Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
                                                 ('scaler',
StandardScaler())]),
                                 ['0', '1', '2', '3', '4', '5',
'bathrooms',
                                  'bedrooms', 'accommodates',
'cleaning_fee',
                                  'avg_lead_time', 'min_lead_time',
                                  'min_airport_time',
'avg_airport_time',
                                  'man_time', 'bro_time', 'que_time',
                                  'bnx_time', 'sta_time',
'nj_time'])])
```

```
# Transform the training data using the fitted preprocessor
X_test_transformed = preprocessor.transform(X_test)
```

```
linear_model1.fit(X_test_transformed, np.exp(y_test))
```

```
LinearRegression()
```

```
linear_pred_test1 = linear_model1.predict(X_test_transformed)
```

```
linear_test_np_rmse, linear_test_np_r2 =
evaluate_model(np.exp(y_test), linear_pred_test1)
```

```
# Evaluating Performance
print("Linear Regression: \nRMSE =", linear_test_np_rmse, "\nR2 =",
linear_test_np_r2)
```

```
Linear Regression:
RMSE = 90.32534158262608
R2 = 0.49789248352777615
```

2. Ridge

Prediction Train

```
grid_search_ridge.fit(X_train, np.exp(y_train))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',
'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ridge', Ridge())]),
             param_grid={'ridge__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

#get the best estimators
best_ridge = grid_search_ridge.best_estimator_

# Predictions for Ridge and Lasso
ridge_pred = best_ridge.predict(X_train)

# Calculate RMSE and R2 for Ridge and Lasso
ridge_train_np_rmse, ridge_train_np_r2 =
evaluate_model(np.exp(y_train), ridge_pred)

print("Ridge: \nRMSE =", ridge_train_np_rmse, "R2 =",
ridge_train_np_r2)
```

```
Ridge:
RMSE = 95.89678590663135 R2 = 0.4634276994933384
```

Prediction val

```
grid_search_ridge.fit(X_val, np.exp(y_val))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',
'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ridge', Ridge())]),
             param_grid={'ridge__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

#get the best estimators
best_ridge = grid_search_ridge.best_estimator_

# Predictions for Ridge and Lasso
ridge_pred = best_ridge.predict(X_val)

# Calculate RMSE and R2 for Ridge and Lasso
ridge_val_np_rmse, ridge_val_np_r2 = evaluate_model(np.exp(y_val),
ridge_pred)

print("Ridge: \nRMSE =", ridge_val_np_rmse, "R2 =", ridge_val_np_r2)

Ridge:
RMSE = 92.93782345662746 R2 = 0.5108676998549305
```

Predicted test

```
grid_search_ridge.fit(X_test, np.exp(y_test))

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',
OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',
'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ridge', Ridge())]),
             param_grid={'ridge__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')
```

```
#get the best estimators
best_ridge = grid_search_ridge.best_estimator_

# Predictions for Ridge and Lasso
ridge_pred = best_ridge.predict(X_test)

# Calculate RMSE and R2 for Ridge and Lasso
ridge_test_np_rmse, ridge_test_np_r2 = evaluate_model(np.exp(y_test),
ridge_pred)

print("Ridge: \nRMSE =", ridge_val_np_rmse, "R2 =", ridge_val_np_r2)
```

```
Ridge:
RMSE = 92.93782345662746 R2 = 0.5108676998549305
```

3. Lasso

Prediction Train

```
grid_search_lasso.fit(X_train, np.exp(y_train))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',
'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',
'nj_time'])])),
                                       ('lasso', Lasso())]),
             param_grid={'lasso__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

best_lasso = grid_search_lasso.best_estimator_

lasso_pred = best_lasso.predict(X_train)

lasso_train_np_rmse, lasso_train_np_r2 =
evaluate_model(np.exp(y_train), lasso_pred)

print("Lasso: \nRMSE =", lasso_train_np_rmse, "R2 =",
lasso_train_np_r2)

Lasso:
RMSE = 95.99532700346555 R2 = 0.4623243968387518
```

Prediction val

```
grid_search_lasso.fit(X_val, np.exp(y_val))

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),
['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('lasso', Lasso())]),
             param_grid={'lasso__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

best_lasso = grid_search_lasso.best_estimator_

lasso_pred = best_lasso.predict(X_val)

lasso_val_np_rmse, lasso_val_np_r2 = evaluate_model(np.exp(y_val),
lasso_pred)
```

```
print("Lasso: \nRMSE =", lasso_val_np_rmse, "R2 =", lasso_val_np_r2)
```

```
Lasso:
RMSE = 93.05160254965764 R2 = 0.5096693267058593
```

Predicted test

```
grid_search_lasso.fit(X_test, np.exp(y_test))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',
'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('lasso', Lasso())]),
             param_grid={'lasso__alpha': [0.1, 1.0, 10.0]},
             scoring='neg_mean_squared_error')

best_lasso = grid_search_lasso.best_estimator_

lasso_pred = best_lasso.predict(X_test)

lasso_test_np_rmse, lasso_test_np_r2 = evaluate_model(np.exp(y_test),
lasso_pred)

print("Lasso: \nRMSE =", lasso_test_np_rmse, "R2 =", lasso_test_np_r2)

Lasso:
RMSE = 90.42635684655166 R2 = 0.49676879256725714
```

4. Random Forest

Prediction Train

```
grid_search_random_forest.fit(X_train, np.exp(y_train))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',
'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('random_forest',
                                        RandomForestRegressor())]),
             param_grid={'random_forest__max_depth': [1, 2, 3, 4],
                         'random_forest__n_estimators': [50, 100,
150]},
             scoring='neg_mean_squared_error')
```

```
best_random_forest = grid_search_random_forest.best_estimator_

rf_pred = best_random_forest.predict(X_train)

rf_train_np_rmse, rf_train_np_r2 = evaluate_model(np.exp(y_train),
rf_pred)

print("Random Forest: \nRMSE =", rf_train_np_rmse, "R2 =",
rf_train_np_r2)
```

```
Random Forest:
RMSE = 93.41767444878269 R2 = 0.49081189499943776
```

Prediction val

```
grid_search_random_forest.fit(X_val, np.exp(y_val))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),
('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                        ('random_forest',
                                         RandomForestRegressor())]),
             param_grid={'random_forest__max_depth': [1, 2, 3, 4],
                         'random_forest__n_estimators': [50, 100,
150]},
             scoring='neg_mean_squared_error')
```

```
best_random_forest = grid_search_random_forest.best_estimator_

rf_pred = best_random_forest.predict(X_val)

rf_val_np_rmse, rf_val_np_r2 = evaluate_model(np.exp(y_val), rf_pred)

print("Random Forest: \nRMSE =", rf_val_np_rmse, "R2 =", rf_val_np_r2)
```

```
Random Forest:
RMSE = 87.01298763279082 R2 = 0.5712446924795399
```

Prediction Test

```
grid_search_random_forest.fit(X_test, np.exp(y_test))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',
'1'...

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('random_forest',
                                        RandomForestRegressor())]),
             param_grid={'random_forest__max_depth': [1, 2, 3, 4],
                         'random_forest__n_estimators': [50, 100,
150]},
             scoring='neg_mean_squared_error')

best_random_forest = grid_search_random_forest.best_estimator_

rf_pred = best_random_forest.predict(X_test)

rf_test_np_rmse, rf_test_np_r2 = evaluate_model(np.exp(y_test),
rf_pred)

print("Random Forest: \nRMSE =", rf_test_np_rmse, "R2 =",
rf_test_np_r2)

Random Forest:
RMSE = 83.77703527702738 R2 = 0.5680559736065185
```

5. X Gradient Boosting

Prediction Train

```
grid_search_xgb.fit(X_train, np.exp(y_train))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

max_delta_step=None,
                                                     max_depth=None,
                                                     max_leaves=None,

min_child_weight=None,
                                                     missing=nan,

monotone_constraints=None,

multi_strategy=None,
n_estimators=None,
                                                     n_jobs=None,

num_parallel_tree=None,

random_state=None, ...))]),
             param_grid={'xgb__learning_rate': [0.05, 0.1, 0.2],
                         'xgb__max_depth': [3, 5, 7],
                         'xgb__n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')

best_xgb = grid_search_xgb.best_estimator_

xgb_pred = best_xgb.predict(X_train)

xgb_train_np_rmse, xgb_train_np_r2 = evaluate_model(np.exp(y_train),
xgb_pred)

print("XGBoost: \nRMSE =", xgb_train_np_rmse, "R2 =", xgb_train_np_r2)

XGBoost:
RMSE = 80.83304934889647 R2 = 0.6187603421063224
```

Prediction val

```
grid_search_xgb.fit(X_val, np.exp(y_val))

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
('scaler',

StandardScaler())]),

['0',

'1'...

max_delta_step=None,
                                                     max_depth=None,
                                                     max_leaves=None,

min_child_weight=None,
                                                     missing=nan,

monotone_constraints=None,

multi_strategy=None,

n_estimators=None,
                                                     n_jobs=None,

num_parallel_tree=None,

random_state=None, ...))]),
             param_grid={'xgb__learning_rate': [0.05, 0.1, 0.2],
                         'xgb__max_depth': [3, 5, 7],
                         'xgb__n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')
```

```
best_xgb = grid_search_xgb.best_estimator_

xgb_pred = best_xgb.predict(X_val)

xgb_val_np_rmse, xgb_val_np_r2 = evaluate_model(np.exp(y_val),
xgb_pred)

print("XGBoost: \nRMSE =", xgb_val_np_rmse, "R2 =", xgb_val_np_r2)
```

```
XGBoost:
RMSE = 74.16607786550288 R2 = 0.6885043591115085
```

Prediction Test

```
grid_search_xgb.fit(X_test, np.exp(y_test))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

max_delta_step=None,
                                                     max_depth=None,
                                                     max_leaves=None,

min_child_weight=None,
                                                     missing=nan,

monotone_constraints=None,

multi_strategy=None,

n_estimators=None,
                                                     n_jobs=None,

num_parallel_tree=None,

random_state=None, ...))]),
             param_grid={'xgb__learning_rate': [0.05, 0.1, 0.2],
                         'xgb__max_depth': [3, 5, 7],
                         'xgb__n_estimators': [50, 100, 150]},
             scoring='neg_mean_squared_error')
```

```
best_xgb = grid_search_xgb.best_estimator_

xgb_pred = best_xgb.predict(X_test)

xgb_test_np_rmse, xgb_test_np_r2 = evaluate_model(np.exp(y_test),
xgb_pred)

print("XGBoost: \nRMSE =", xgb_test_np_rmse, "R2 =", xgb_test_np_r2)
```

```
XGBoost:
RMSE = 68.18253681651818 R2 = 0.713896079607504
```

6. ANN

Prediction train

```
# Fit the grid search object
grid_search_ann.fit(X_train, np.exp(y_train))
```

```
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'avg_lead_time',

'min_lead_time',
'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ann',
MLPRegressor(max_iter=500))]),
             param_grid={'ann__activation': ['relu', 'tanh',
'logistic'],
                         'ann__alpha': [0.0001, 0.001, 0.01],
                         'ann__hidden_layer_sizes': [(50,), (100,),
(50, 50)]},
             scoring='neg_mean_squared_error')

# Get the best estimator
best_ann = grid_search_ann.best_estimator_

# Evaluate the model
ann_train_np_rmse, ann_train_np_r2 =
evaluate_model(best_ann.predict(X_train), np.exp(y_train))

# Print RMSE and R2 score for ANN
print("ANN: \nRMSE =", ann_train_np_rmse, "R2 =", ann_train_np_r2)

ANN:
RMSE = 86.34240728472753 R2 = 0.19471599863258782
```

Prediction val

```
# Fit the grid search object
grid_search_ann.fit(X_val, np.exp(y_val))

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),
['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ann',
MLPRegressor(max_iter=500))]),
             param_grid={'ann__activation': ['relu', 'tanh',
'logistic'],
                         'ann__alpha': [0.0001, 0.001, 0.01],
                         'ann__hidden_layer_sizes': [(50,), (100,),
(50, 50)]},
             scoring='neg_mean_squared_error')

# Get the best estimator
best_ann = grid_search_ann.best_estimator_
```

```
# Evaluate the model
ann_val_np_rmse, ann_val_np_r2 =
evaluate_model(best_ann.predict(X_val), np.exp(y_val))

# Print RMSE and R2 score for ANN
print("ANN: \nRMSE =", ann_val_np_rmse, "R2 =", ann_val_np_r2)

ANN:
RMSE = 84.50659066985257 R2 = 0.29774076481103273
```

Prediction Test

```
# Fit the grid search object
grid_search_ann.fit(X_test, np.exp(y_test))

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
```

```
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
```

```
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
```

```
hasn't converged yet.
  warnings.warn(
C:\Users\navab\anaconda3\Lib\site-packages\sklearn\neural_network\
_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (500) reached and the optimization
hasn't converged yet.
  warnings.warn(

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1'...

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',
'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('ann',
MLPRegressor(max_iter=500))]),
             param_grid={'ann__activation': ['relu', 'tanh',
'logistic'],
                         'ann__alpha': [0.0001, 0.001, 0.01],
                         'ann__hidden_layer_sizes': [(50,), (100,),
(50, 50)]},
             scoring='neg_mean_squared_error')
```

```
# Get the best estimator
best_ann = grid_search_ann.best_estimator_

# Evaluate the model
ann_test_np_rmse, ann_test_np_r2 =
evaluate_model(best_ann.predict(X_test), np.exp(y_test))

# Print RMSE and R2 score for ANN
print("ANN: \nRMSE =", ann_test_np_rmse, "R2 =", ann_test_np_r2)
```

```
ANN:
RMSE = 80.37391119787735 R2 = 0.3157751624717988
```

SVR

Prediction train

```
grid_search_svr.fit(X_train, np.exp(y_train))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',
OneHotEncoder(handle_unknown='ignore'))]),
['room_type',
'Borough']),
('num',
Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
('scaler',
StandardScaler())]),
['0',
'1',
'2',
'3',
'4',
'5',
'bathrooms',
'bedrooms',
'accommodates',
'cleaning_fee',
'avg_lead_time',
'min_lead_time',
'min_airport_time',
'avg_airport_time',
'man_time',
'bro_time',
'que_time',
'bnx_time',

'sta_time',

'nj_time'])])),
                                        ('svr', SVR())]),
             param_grid={'svr__C': [0.1, 1.0, 10.0],
                         'svr__kernel': ['linear', 'rbf']},
             scoring='neg_mean_squared_error')

best_svr = grid_search_svr.best_estimator_

svr_pred = best_svr.predict(X_train)

svr_train_np_rmse, svr_train_np_r2 = evaluate_model(np.exp(y_train), 
svr_pred)

print("SVR: \nRMSE =", svr_train_np_rmse, "R2 =", svr_train_np_r2)

SVR: 
RMSE = 96.53280661559309 R2 = 0.45628662994005953
```

Prediction val

```
grid_search_svr.fit(X_val, np.exp(y_val))

GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),
('scaler',

StandardScaler())]),

['0',

'1',

'2',

'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('svr', SVR())]),
             param_grid={'svr__C': [0.1, 1.0, 10.0],
                         'svr__kernel': ['linear', 'rbf']},
             scoring='neg_mean_squared_error')

best_svr = grid_search_svr.best_estimator_
```

```
svr_pred = best_svr.predict(X_val)

svr_val_np_rmse, svr_val_np_r2 = evaluate_model(np. exp(y_val),
svr_pred)

print("SVR: \nRMSE =", svr_val_np_rmse, "R2 =", svr_val_np_r2)
```

```
SVR:
RMSE = 99.10864116699966 R2 = 0.4437572141666458
```

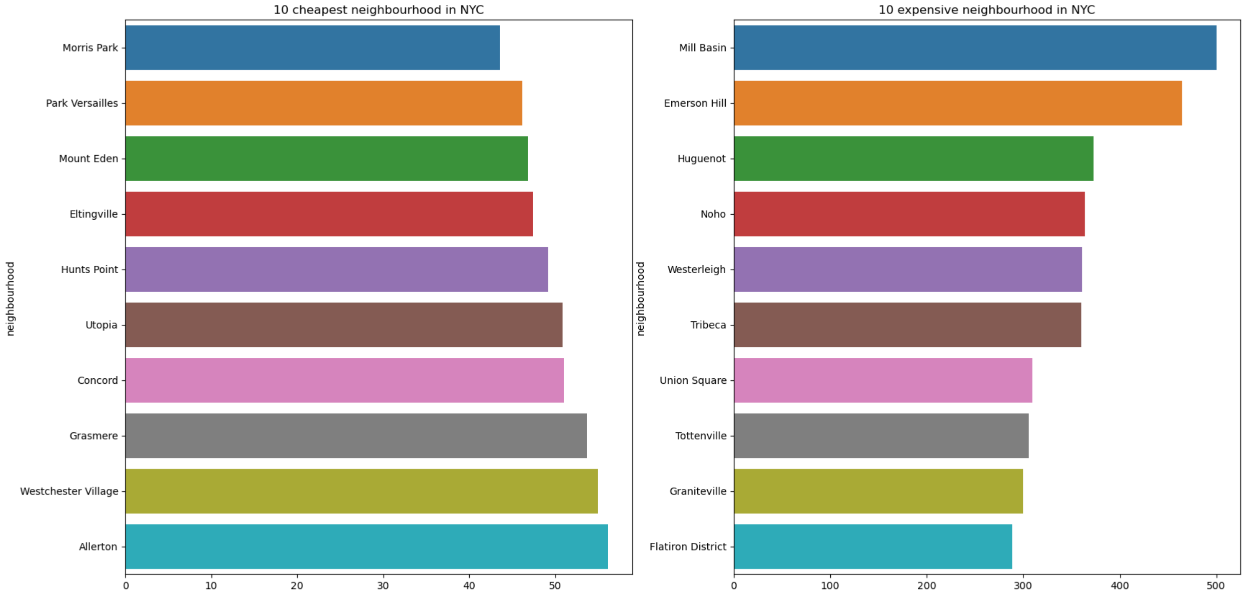
Prediction Test

```
grid_search_svr.fit(X_test, np.exp(y_test))
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('preprocessor',

ColumnTransformer(transformers=[('cat',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='most_frequent')),

('onehot',

OneHotEncoder(handle_unknown='ignore'))]),

['room_type',

'Borough']),

('num',

Pipeline(steps=[('imputer',

SimpleImputer(strategy='median')),

('scaler',

StandardScaler())]),

['0',

'1',

'2',
'3',

'4',

'5',

'bathrooms',

'bedrooms',

'accommodates',

'cleaning_fee',

'avg_lead_time',

'min_lead_time',

'min_airport_time',

'avg_airport_time',

'man_time',

'bro_time',

'que_time',

'bnx_time',

'sta_time',

'nj_time'])])),
                                       ('svr', SVR())]),
             param_grid={'svr__C': [0.1, 1.0, 10.0],
                         'svr__kernel': ['linear', 'rbf']},
             scoring='neg_mean_squared_error')

best_svr = grid_search_svr.best_estimator_

svr_pred = best_svr.predict(X_test)

svr_test_np_rmse, svr_test_np_r2 = evaluate_model(np. exp(y_test),
svr_pred)

print("SVR: \nRMSE =", svr_test_np_rmse, "R2 =", svr_test_np_r2)

SVR:
RMSE = 93.36339308481574 R2 = 0.4635481453398915

data_cleaned['Price'] = np.exp(data_cleaned['log_price'])
```

```python
#Top 10 expensive & Cheapest neighbourhood in NYC
a = data_cleaned.groupby('neighbourhood')
['Price'].mean().sort_values(ascending=True).head(10)
b = data_cleaned.groupby('neighbourhood')
['Price'].mean().sort_values(ascending=False).head(10)
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(121)
sns.barplot(y=a.index, x=a.values, ax=ax1)
ax1.set_title('10 cheapest neighbourhood in NYC')
ax2 = fig.add_subplot(122)
sns.barplot(y=b.index, x=b.values, ax=ax2)
ax2.set_title('10 expensive neighbourhood in NYC')
plt.show()
```

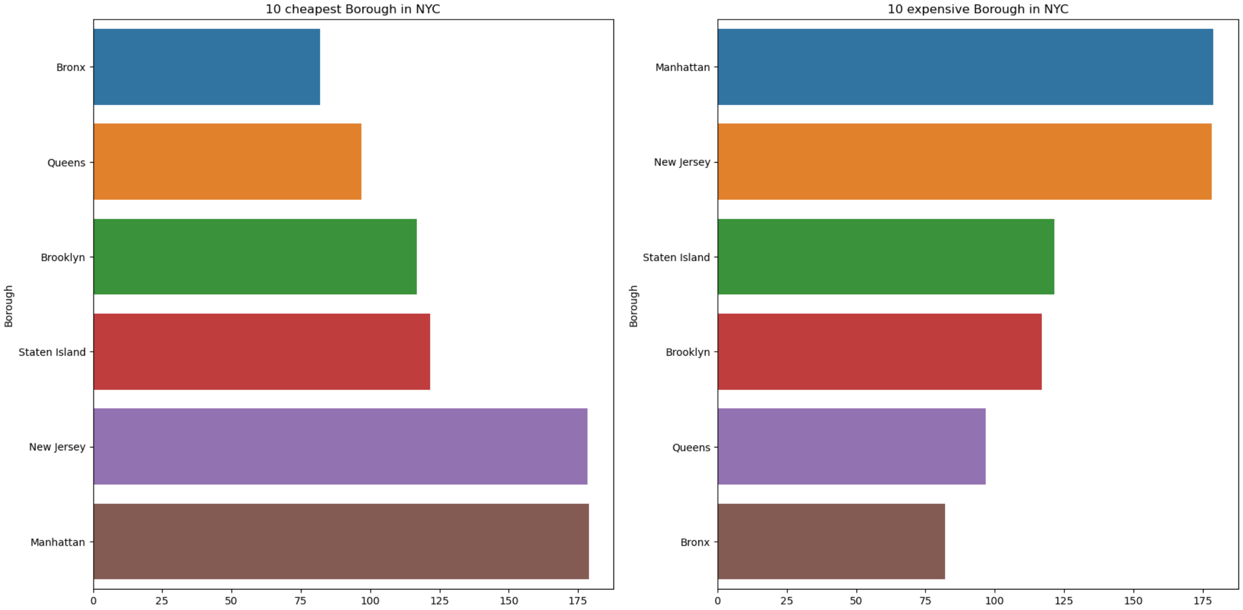
```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



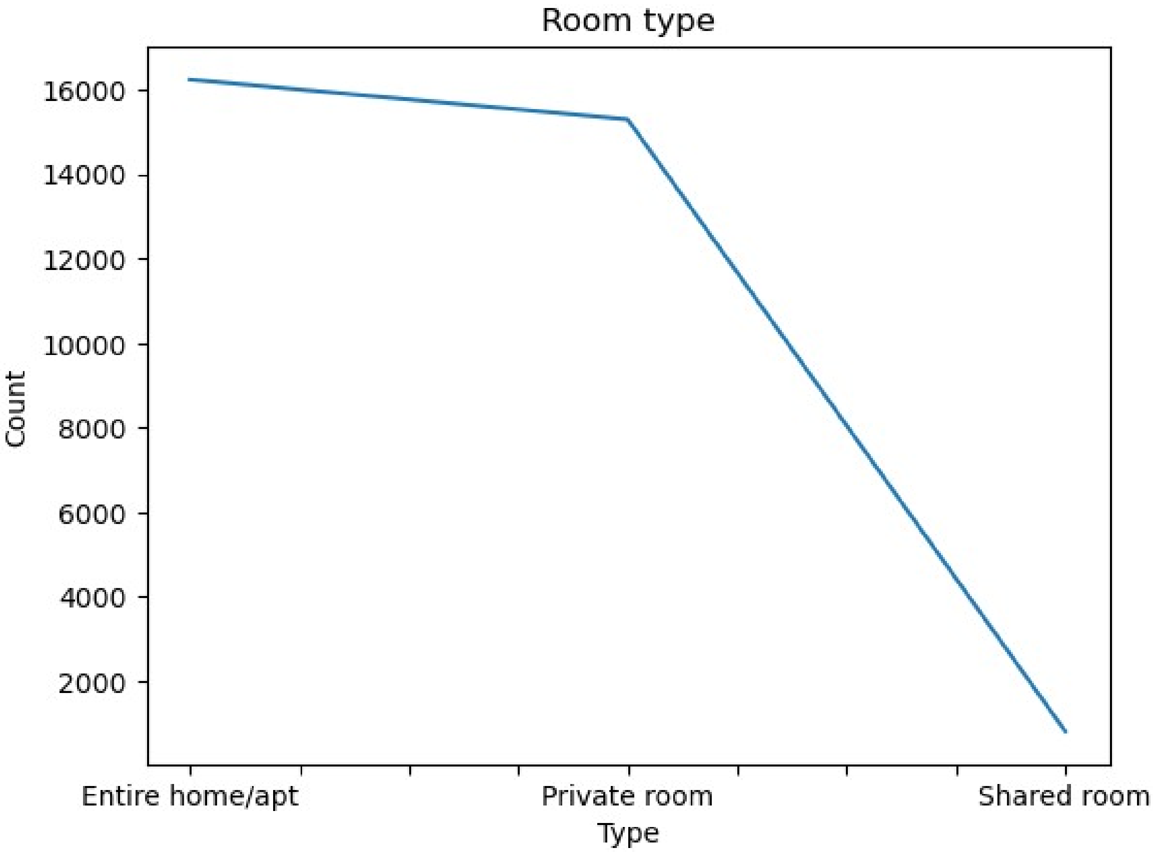
```
#Top 10 expensive & Cheapest Borough in NYC
a = data_cleaned.groupby('Borough')
['Price'].mean().sort_values(ascending=True).head(10)
b = data_cleaned.groupby('Borough')
['Price'].mean().sort_values(ascending=False).head(10)
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(121)
sns.barplot(y=a.index, x=a.values, ax=ax1)
ax1.set_title('10 cheapest Borough in NYC')
ax2 = fig.add_subplot(122)
sns.barplot(y=b.index, x=b.values, ax=ax2)
ax2.set_title('10 expensive Borough in NYC')
plt.show()
```

```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
```

```
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```

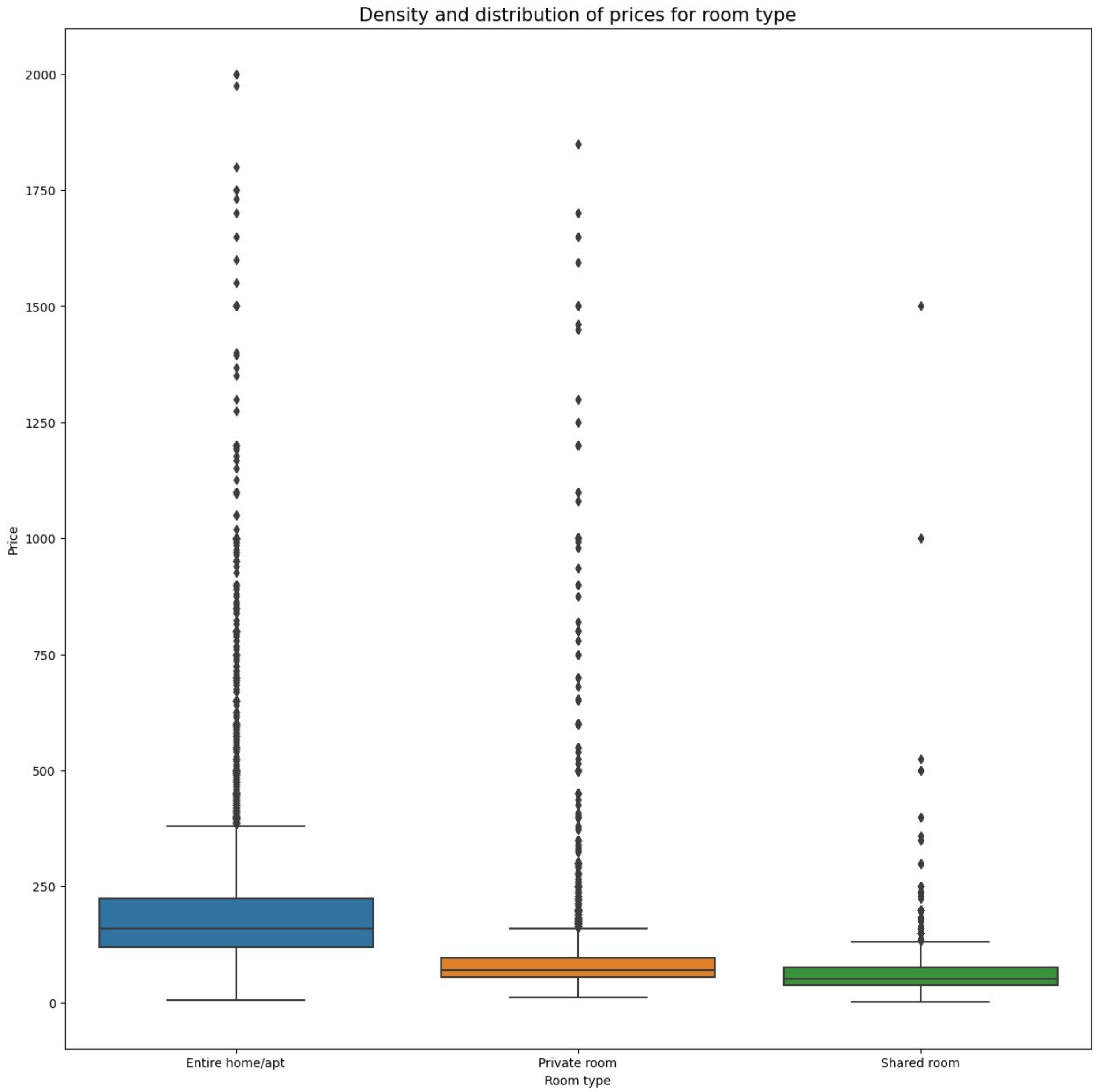


```
#Ploting room count
data_cleaned['room_type'].value_counts().plot(kind='line')
plt.title('Room type')
plt.ylabel('Count')
plt.xlabel('Type')

Text(0.5, 0, 'Type')
```



```
#Box Plot to view outliers
plt.figure(figsize=(15,15))
sns.boxplot(data=data_cleaned, x='room_type', y='Price')
plt.title('Density and distribution of prices for room type',
fontsize=15)
plt.xlabel('Room type')
plt.ylabel("Price")
```

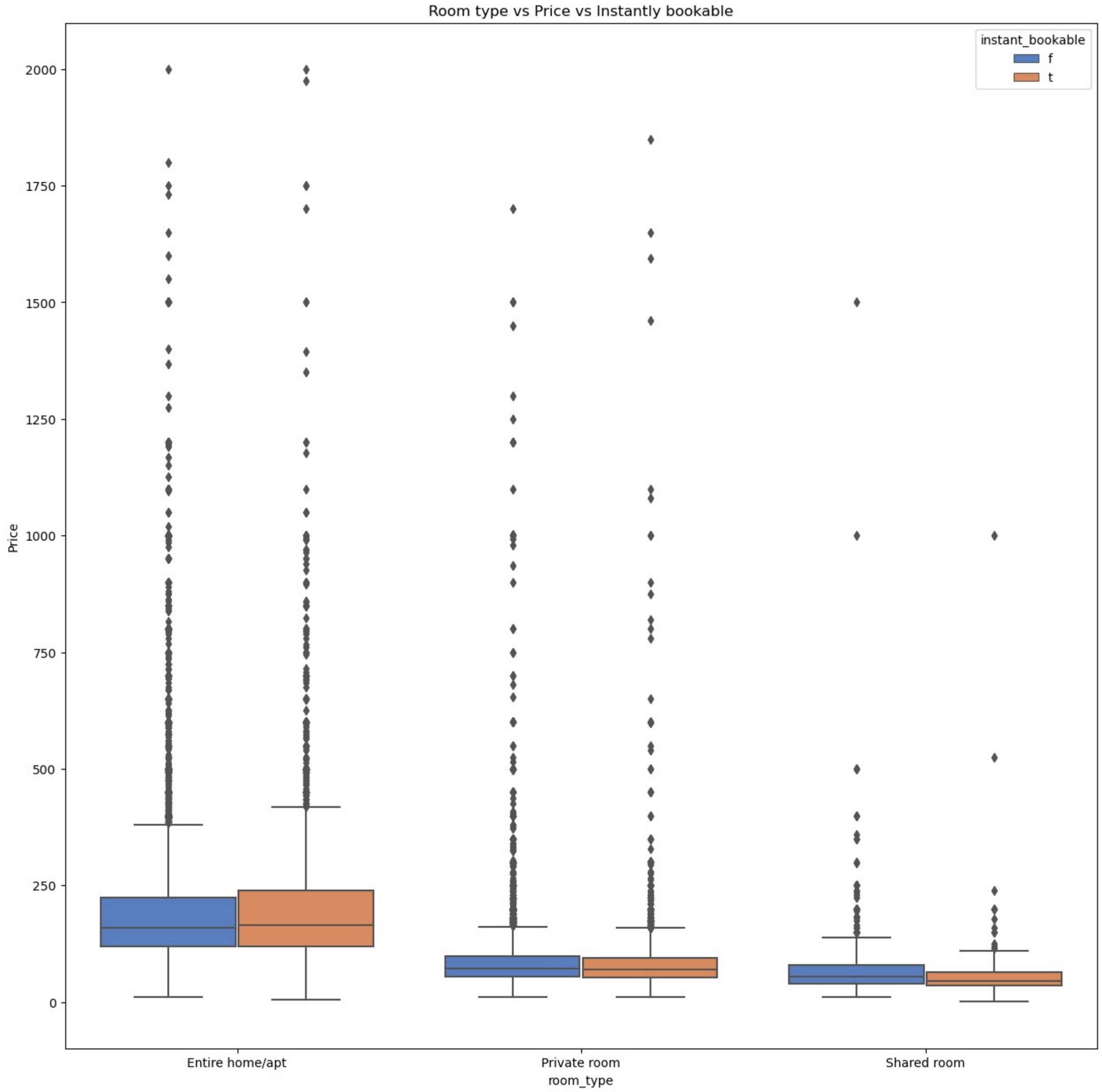
```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):

Text(0, 0.5, 'Price')
```



```
#correlating instant bookable rooms with price
plt.figure(figsize=(15,15))
sns.boxplot(x='room_type',y='Price', hue="instant_bookable",
data=data_cleaned, palette='muted')
plt.title("Room type vs Price vs Instantly bookable")
```

```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
```

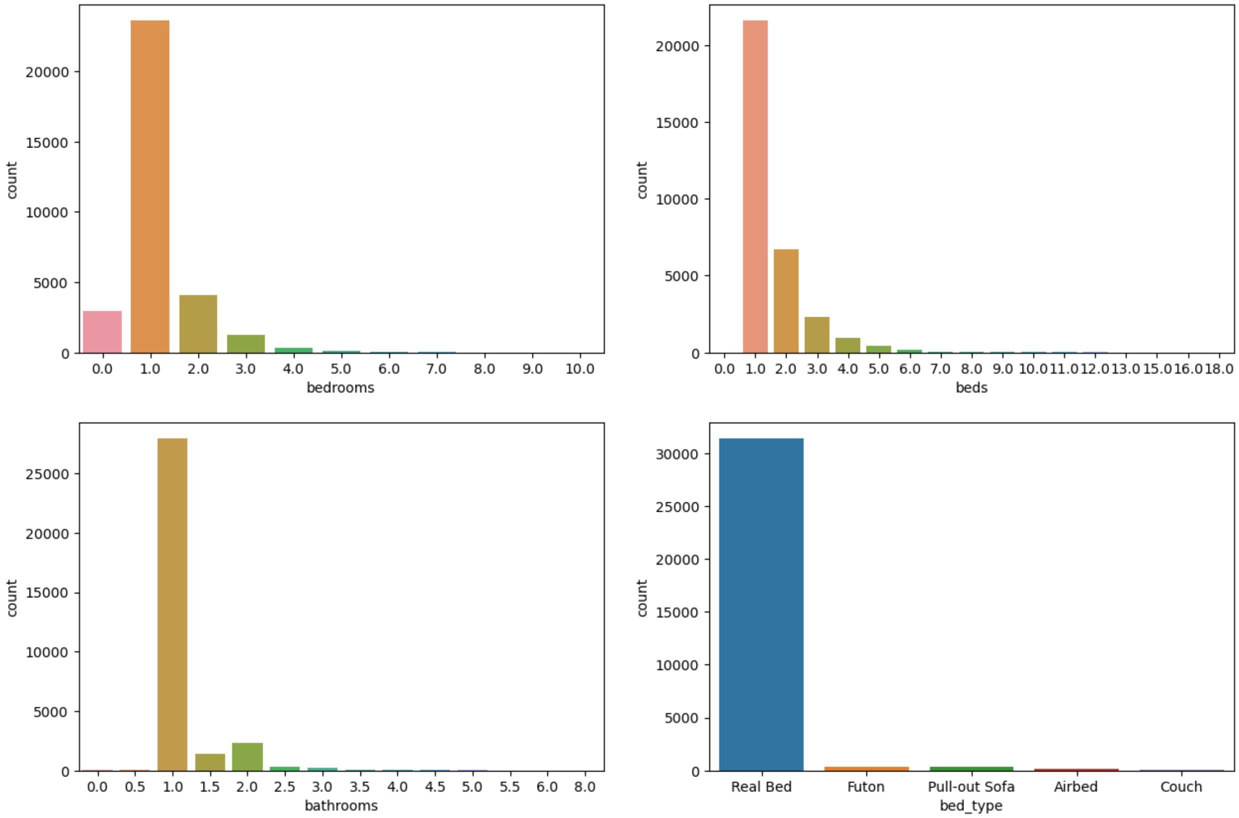
```
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):

Text(0.5, 1.0, 'Room type vs Price vs Instantly bookable')
```



```
#plotting the count of bedrooms, beds and bathrooms.
a=['bedrooms','beds','bathrooms','bed_type']
fig, axes = plt.subplots(2,2, figsize=(15,10))
axes = [ax for axes_row in axes for ax in axes_row]
for i, c in enumerate(a):
    f = sns.countplot(x=data_cleaned[c], data=data_cleaned,
ax=axes[i])
```

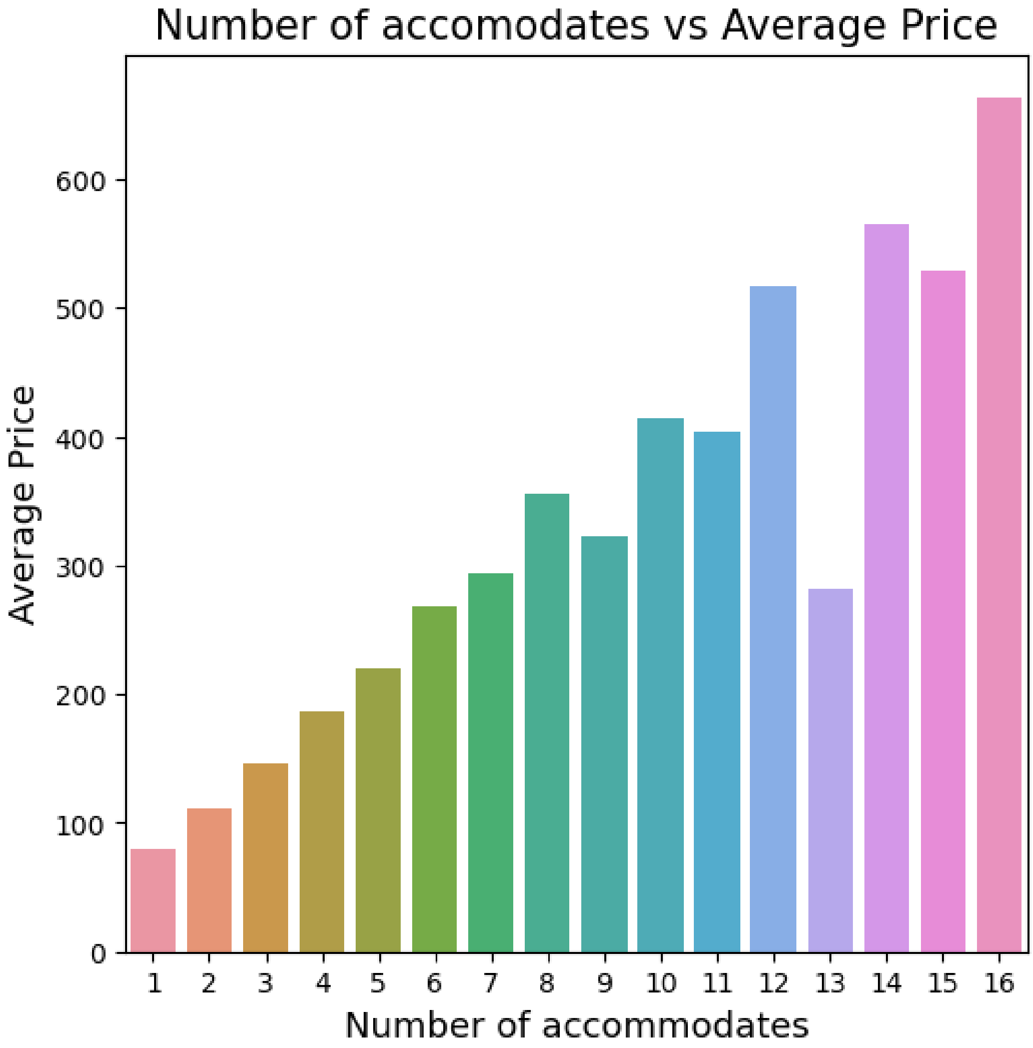
```
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
```

```
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



```
#plotting the number of accomodates vs mean price
a = data_cleaned.groupby('accommodates')['Price'].mean()
fig = plt.figure(figsize=(6,6))
sns.barplot(y=a.values, x=a.index)
plt.xlabel("Number of accommodates", size=13)
plt.ylabel("Average Price", size=13)
plt.title("Number of accomodates vs Average Price",size=15)

C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\navab\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1498:
```

```
FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):

Text(0.5, 1.0, 'Number of accomodates vs Average Price')
```


Number of accomodates vs Average Price
Average Price
Number of accommodates