Rental Price Prediction Using Machine Learning: A Case Study of Airbnb Listings in New York City

Chang Liu

240147976

School of Business and Management

BUSM131 – Masterclass in Business Analytics

May 13, 2025

Abstract

Accurately predicting rental property prices is a crucial component of real estate investment strategy, directly influencing asset screening, acquisition, and pricing decisions. This study develops a machine learning–based rental price prediction system using Airbnb listing data from New York City. The project addresses the challenge of modelling complex, nonlinear relationships between rental prices and diverse features, including location, structural traits, service attributes, and user-generated quality indicators.

The dataset underwent comprehensive preprocessing, including geographic feature engineering, outlier removal, and variable transformation. Five widely used machine learning models—K-Nearest Neighbours, Random Forest, Gradient Boosting, LightGBM, and Artificial Neural Networks—were trained using cross-validation, with hyperparameters optimized through GridSearchCV or empirical design. Model performance was evaluated using R^2, MSE, RMSE, MAE, and MAPE, supported by residual and kernel density diagnostics.

The Gradient Boosting model achieved the best overall performance, combining high accuracy, low variance, and consistent residual distribution. The system provides investors and platform operators with a robust decision-support tool for rental yield estimation. Beyond technical effectiveness, the model is designed for practical integration into business workflows, enabling scalable and interpretable pricing insights. Limitations and future development directions, such as incorporating additional features or extending to other markets, are also discussed.

1. Introduction

Given the growing reliance on data-driven tools in property investment, predicting rental prices accurately is increasingly critical. For both institutional and individual investors, understanding expected rental income is fundamental to evaluating asset profitability, managing portfolio risk, and guiding location-based acquisition strategies. As housing markets grow more complex and data-rich, traditional valuation methods often struggle to capture nonlinear dynamics and interrelated features. There is thus increasing demand for scalable, data-driven tools that can support more informed and systematic investment decisions.

To address this need, the present study proposes a machine learning framework for rental price prediction based on structured listing data. The model draws on four key categories of features: location, basic property traits, facilities and services, and service quality. Machine learning models are particularly well suited to this task, as they can process high-dimensional inputs, identify latent patterns, and generate more accurate and consistent forecasts than traditional methods.

This study builds on recent academic developments in machine learning–based real estate modelling. Tchuente and Nyawa (2022) demonstrated the importance of geographic information in housing valuation by integrating geocoded features such as coordinates and distance to amenities into machine learning models, resulting in a more than 50% increase in prediction accuracy. Zhao et al. (2023) introduced a Transformer-based generative language model

that incorporates textual property descriptions. Their model outperformed traditional ML methods in both prediction quality and interpretability. Hernes et al. (2024) evaluated various algorithms, including linear regression, random forests, and artificial neural networks, and found that ANNs produced the lowest mean absolute percentage error (MAPE), reinforcing their capacity to model complex, nonlinear relationships. In a related study, Hernes, Tutak, and Siewiera (2024) applied ensemble learning to primary market data in Poland and achieved 98% accuracy, further demonstrating the real-world applicability of advanced ML techniques.

Inspired by these contributions, this project evaluates five widely used ML algorithms—K-Nearest Neighbours (KNN), Random Forest, Gradient Boosting, LightGBM, and Artificial Neural Networks (ANN). Each model is trained via cross-validation and assessed using multiple evaluation metrics to determine predictive effectiveness and business relevance.

To present the modelling and analysis process, the remainder of this report is organized as follows: Section 2 defines the business problems addressed. Section 3 details the dataset, preprocessing, feature engineering, and model development. Section 4 presents the results of model evaluation and performance comparison. Section 5 concludes with key findings and directions for future work.

2. Business Problem

Real estate investment has long been recognised as a key avenue for generating wealth, but poor investment decisions can result in substantial sunk costs and financial risk. For both institutional and individual investors, one of

the most pressing questions is: how much income can a property realistically generate? Accurately predicting rental yields is therefore essential for evaluating profitability, managing risk, and making informed acquisition and pricing decisions. However, this task is particularly challenging in dynamic real estate markets, where rent levels are influenced by a complex interplay of factors, including location, property characteristics, seasonal fluctuations, guest preferences, and broader market dynamics. Traditional pricing approaches such as manual benchmarking, simple regression models or intuition-based valuation often struggle to adapt to such multidimensional conditions, which limits their practical value in fast-paced and competitive markets.

To address these challenges, a growing number of real estate technology companies have turned to data-driven forecasting tools. Organisations like AirDNA provide granular rental profitability reports across different neighbourhoods, underscoring the need for location-sensitive modelling (AirDNA, 2023). CBRE, one of the largest commercial real estate firms, has adopted AI-driven appraisal systems to improve accuracy and speed, especially in volatile or data-rich markets (CBRE Research, 2023). Sonder, a technology-enabled property operator, employs machine learning to dynamically optimise pricing and occupancy across its property portfolio (Marr & Ward, 2019). Even Airbnb has integrated a Smart Pricing tool that automatically adjusts listing prices based on platform-level data and market signals, although this tool primarily serves individual hosts rather than professional investors.

These examples illustrate how rental price prediction models are evolving from research prototypes into operational tools that directly influence business

processes. In this context, interpretable and scalable machine learning models can support multiple phases across the property investment lifecycle. At the early stage, such models assist in property screening, where investors rely on rental forecasts to compare listings and prioritise acquisition decisions. For instance, Zillow's Zestimate tool helps users evaluate potential investments by offering automated value estimates during the property selection process (Zillow, 2024).

In the area of investment portfolio planning, price prediction models enable capital to be allocated based on forecasted yield potential across markets. The Anaplan Connected Planning platform provides a real-world example, integrating predictive pricing analytics into institutional portfolio configuration to align resources with expected returns (Anaplan, 2023).

At the operational level, machine learning models can help landlords and platforms determine optimal initial listing prices and adjust them in response to market dynamics. A practical example is ImmoScout24's use of predictive models via the WohnBarometer tool, which guides landlords in setting competitive prices at the listing stage to maximise occupancy and rental efficiency (ImmoScout24, 2024).

Moreover, such models can serve as back-end components in automated valuation systems, offering scalable solutions for mass appraisal or institutional-grade decision platforms. By embedding the predictions into front-end interfaces or API services, these tools can also empower non-technical users, such as landlords, agents, or junior analysts, to access consistent valuation estimates and interpret model outputs through intuitive visualisations. As a

result, predictive pricing becomes not only a forecasting function but also a foundation for improved communication, client advisory, and internal decision alignment.

The goal of this project is to contribute to this growing area by developing a reliable, interpretable, and practically useful machine learning model for rental price prediction. Rather than providing full investment analysis or return estimates, the system focuses on predicting rental values based on structured listing data. These predictions can then be used as inputs for investment screening, pricing, planning, and reporting. Ultimately, the model is intended to support more transparent and data-driven decision-making in the real estate sector.

3. Data, EDA, and Methods

This section provides an overview of the data source, exploratory data analysis, preprocessing workflow, and machine learning methods used to develop the rental price prediction models. The analysis is based on historical Airbnb booking data, focusing on listings located in New York City (NYC), though the original dataset includes properties from multiple U.S. states. The dataset contains structured variables such as pricing, property characteristics, amenities, geographic data, and user reviews.

To ensure data quality, listings with a target price of zero or missing values in key variables were excluded. Additionally, price outliers were removed using the interquartile range method, retaining only listings with prices within 1.5 × IQR from the 5th to 95th percentile range. This step helped reduce the impact of extreme values and enhanced model stability.

3.1 Preprocessing Stage

Before model development, a comprehensive preprocessing phase was conducted to transform raw data into structured inputs. This involved cleaning, engineering, and categorizing features to ensure consistency with machine learning models. Features were divided into four categories:

3.1.1 Location

Spatial information contained in the original dataset was carefully processed to generate geographic features that could help explain variations in property prices. See Appendix A.1.These features were constructed using each listing's latitude and longitude, incorporating administrative classification, distance calculations, and functional transformations to reflect location-based heterogeneity.

First, each listing was matched to one of New York City's five boroughs—Manhattan, Brooklyn, Queens, Bronx, or Staten Island—based on its geographic coordinates. This was achieved using spatial joins with official borough shapefiles via the GeoPandas library. A new categorical variable, *borough*, was created and subsequently encoded for modelling purposes.

Second, Haversine distances were calculated between each listing and the city's three major airports: John F. Kennedy International Airport (JFK), LaGuardia Airport (LGA), and Newark Liberty International Airport (EWR). These distances, named *distance_to_jfk*, *distance_to_lga*, and *distance_to_ewr*, capture differences in transportation accessibility.

To account for proximity to the urban core, a new variable,

distance_to_manhattan, was created based on the distance from each listing to a central point in Manhattan (defined as Central Park). To reflect the economic effect of urban centrality, a transformation was applied using the formula *manhattan_effect = exp(-0.1 * distance_to_manhattan)*, under the assumption that listings closer to the city centre tend to command higher prices.

Additional landmark-based features were created by calculating distances to well-known borough-specific locations, such as the Brooklyn Bridge and Prospect Park. These resulted in variables like manhattan_times_square_distance and brooklyn_prospect_park_distance—aim to reflect the localized influence of cultural and touristic proximity.

All location-related variables were grouped into a unified feature set named location_features. Prior to model training, these variables were imputed for missing values and standardized using RobustScaler to minimize the influence of outliers and maintain consistency across models.

A visualization of the borough classifications and landmark distributions is provided in Figure 1.

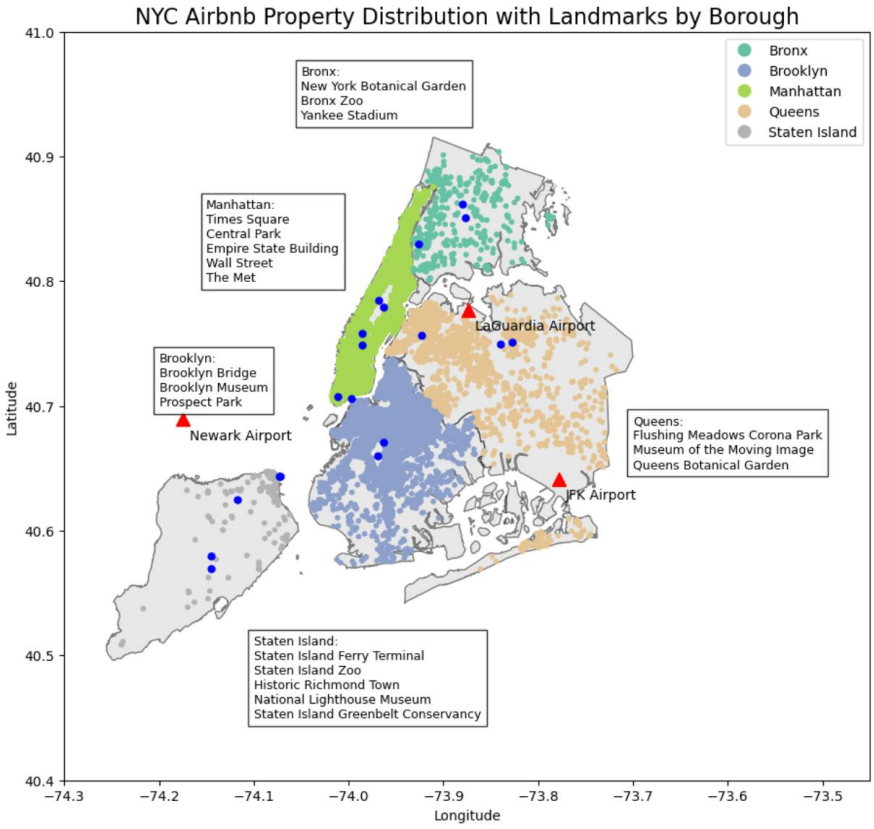


Figure 1. Geomap: Property Locations and Landmarks by Borough

3.1.2 Basic traits

Fundamental property characteristics contained in the original dataset were directly included as predictive variables, focusing on core attributes that reflect the size and capacity of each listing. These include the number of guests a listing can accommodate (*accommodates*), the number of bedrooms (*bedrooms*), beds (*beds*), and bathrooms (*bathrooms*).

Initial descriptive statistics and visual analysis revealed considerable variability across these features, all of which showed a generally positive association with property prices. See Appendix A.2. Boxplots of price by each trait (see Figure 2) suggest that listings with more space and amenities tend to have higher prices. For example, properties with more bedrooms or bathrooms usually

command higher median prices, although the increase in price tends to level off beyond certain thresholds.

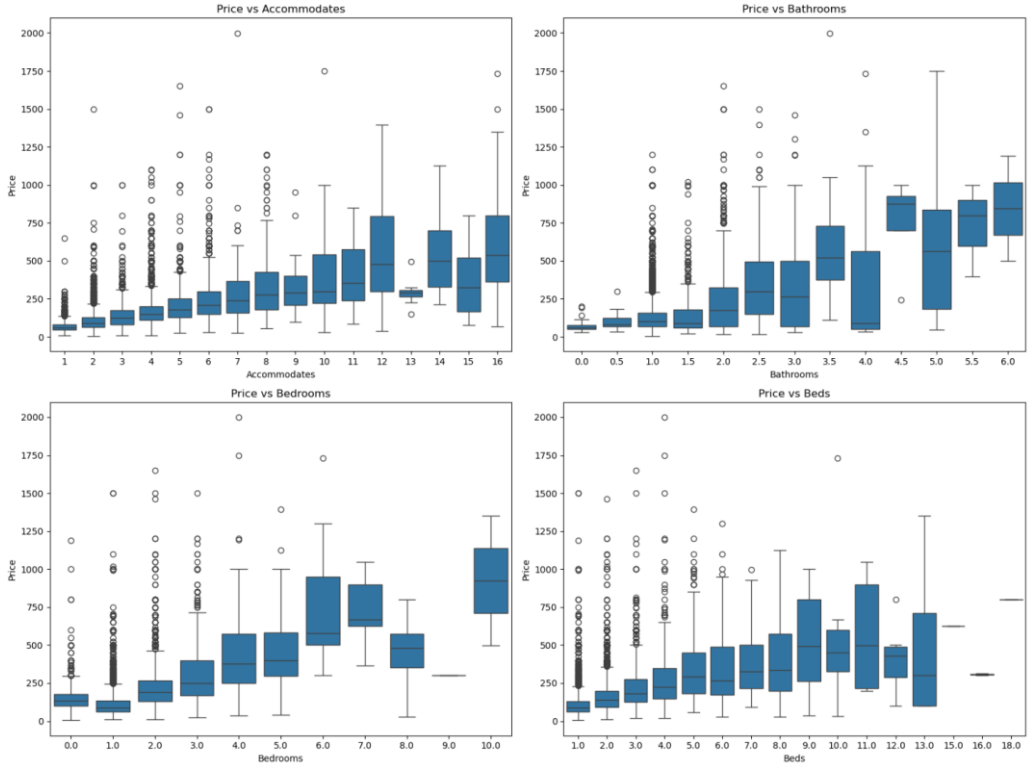


Figure 2. Boxplot: Basic Traits and Price

To further explore interrelationships between traits, a correlation matrix was computed (see Figure 3). The results indicated relatively high correlations between *accommodates* and *beds* (r = 0.84), and between *bedrooms* and *beds* (r = 0.70). While these correlations suggest possible redundancy, they also reflect the inherent structural linkage in property layouts.

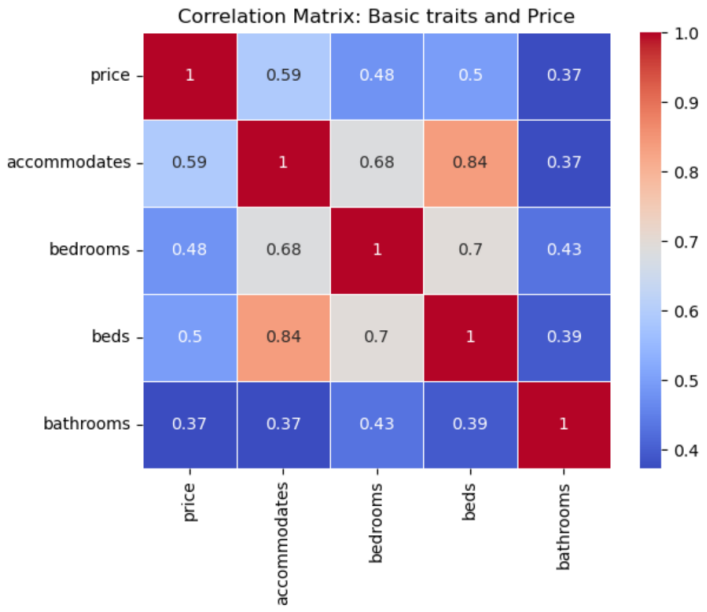


Figure 3. Correlation Matrix: Basic Traits and Price

To enhance the descriptive power of these basic traits, two engineered features were created. The first, *price_per_accommodates*, is the ratio of total price to the number of accommodates, capturing the average cost per guest. The second, *beds_per_bedroom*, measures bed density and reflects how intensively space is used for sleeping. These derived variables were added to the final feature set used for model training.

3.1.3 Facilities and Services

Facilities and services variables contained in the original dataset were systematically cleaned and transformed to enhance their usability in predictive modelling. See Appendix A.3.

Key variables in this category include *amenities*, *instant_bookable*, *cleaning_fee*, and *cancellation_policy*, each representing distinct aspects of service availability and booking flexibility.

First, the *amenities* field was originally stored as a single string containing a comma-separated list of features, often enclosed in curly braces and quotation marks, and occasionally including placeholder values such as "translation missing". To extract meaningful information, the string was parsed into a list format, with extraneous characters removed and invalid entries filtered out. Each amenity name was then standardized by trimming whitespace and applying consistent capitalization. Based on the cleaned list, a new numeric variable *amenities_count* was created to quantify the total number of available amenities per listing, serving as a proxy for service richness.

Second, both *instant_bookable* and *cleaning_fee* were treated as Boolean indicators, though they appeared in various formats including 't', 'f', True, False, 'TRUE', and 'FALSE'. These values were harmonized into binary form, where affirmative values were mapped to 1 and all others, including missing entries, were mapped to 0. The resulting features were cast as integers for compatibility with machine learning algorithms and included in the final feature set.

Third, for the categorical variable *cancellation_policy*, categories with extremely low frequency ('super_strict_30', 'super_strict_60') were removed to mitigate issues related to class imbalance. Remaining values were converted to string type, missing entries were imputed with 'Unknown', and the field was encoded numerically using label encoding to meet model input requirements.

This structured transformation ensured consistency, minimized noise, and enhanced the modelling readiness of service-related features.

3.2.4 Quality

Quality-related variables contained in the original dataset include

review_scores_rating and number_of_reviews, which serve as proxies for listing reputation, user satisfaction, and historical platform performance. These features are expected to offer insights into how perceived service quality influences housing prices.

Both variables are continuous and were preserved in their original numerical form for model development. See Appendix A.4.

To explore their relationship with price, hexbin plots were employed for visual inspection (see Figure 4). The distribution of review_scores_rating shows that highly rated listings are concentrated in the low- to mid-price range, with some extending into higher-price tiers, suggesting a potential positive correlation.

In contrast, *number_of_reviews* is heavily right-skewed. Listings with few reviews appear across all price levels, while those with exceptionally high counts cluster mainly in the low- to mid-price range, implying a possible nonlinear or indirect relationship with price.

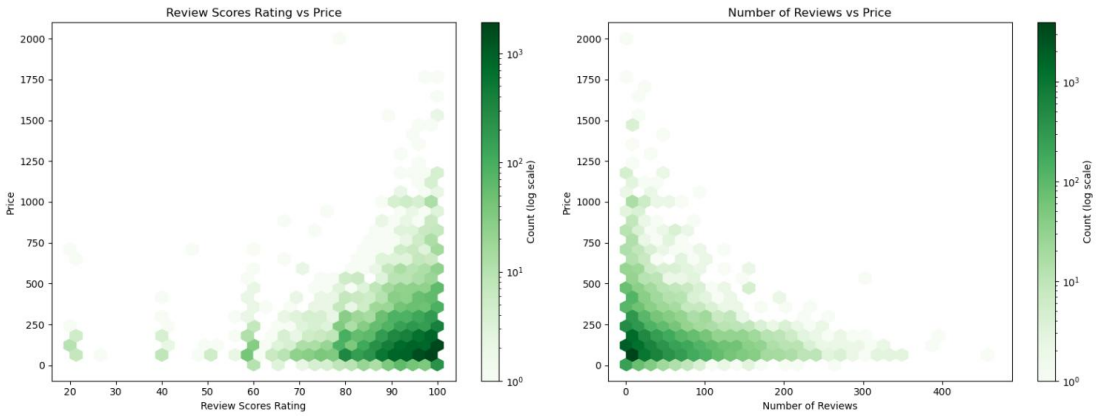


Figure 4. Hexbin Plots: Quality variables and Price

In summary, Table 1 presents the descriptive statistics of all related variables after preprocessing.

Table 1. Summary Statistics after Preprocessing

	count	mean	std	min	25%	50%	75%	max
accommodates	17764.0	3.01	1.94	1.00	2.00	2.00	4.00	16.00
bathrooms	17764.0	1.13	0.39	0.00	1.00	1.00	1.00	6.00
bedrooms	17764.0	1.19	0.74	0.00	1.00	1.00	1.00	10.00
beds	17764.0	1.65	1.14	1.00	1.00	1.00	2.00	18.00
price_per_accommodates	17764.0	49.91	29.37	2.50	30.00	43.61	61.82	650.00
beds_per_bedroom	17764.0	1.30	0.59	0.20	1.00	1.00	1.50	10.00
instant_bookable	17764.0	0.30	0.46	0.00	0.00	0.00	1.00	1.00
cleaning_fee	17764.0	0.80	0.40	0.00	1.00	1.00	1.00	1.00
amenities_count	17764.0	17.49	7.57	0.00	13.00	16.00	21.00	77.00
review_scores_rating	17764.0	93.75	7.29	20.00	91.00	95.00	99.00	100.00
number_of_reviews	17764.0	28.80	38.90	1.00	4.00	14.00	37.00	465.00
manhattan_effect	17764.0	0.49	0.20	0.02	0.33	0.47	0.63	1.00
distance_to_jfk	17764.0	18.59	4.30	2.58	15.54	18.93	21.75	41.69
distance_to_lga	17764.0	10.45	3.58	0.66	8.42	10.21	12.03	42.92
distance_to_ewr	17764.0	20.12	4.19	6.35	17.16	19.32	22.25	38.78
distance_to_manhattan	17764.0	8.18	4.68	0.03	4.60	7.58	11.20	38.01

3.2 Modelling Stage

After completing data preprocessing, the dataset was ready for model training and price prediction.

In the modelling stage, five mainstream machine learning models were employed: K-Nearest Neighbours (KNN), Random Forest, Gradient Boosting, LightGBM, and Artificial Neural Networks (ANN). These models were selected due to their distinct structures and algorithmic strengths in capturing nonlinear relationships and feature interactions. Specifically, KNN is simple to implement

and serves as a baseline model. Random Forest offers robustness and resistance to overfitting, particularly useful for high-dimensional data. Gradient Boosting is known for its strong performance in regression tasks. LightGBM provides an efficient implementation of boosting methods with faster computation and better scalability. ANN, through its layered architecture, enables complex nonlinear approximations and is effective when hidden feature interactions are expected.

Before training, the dataset was split into training and test sets in an 80:20 ratio to ensure fair evaluation and external validation. To reduce overfitting and enhance model generalization, five-fold cross-validation was applied to all models during training and hyperparameter tuning. In each fold, the training set was partitioned into five subsets, and four subsets were used for training and one for validation. This process was repeated five times, and the average performance across folds was used to guide hyperparameter selection.

All hyperparameter optimization for tree-based models (KNN, Random Forest, Gradient Boosting, and LightGBM) was conducted using GridSearchCV, which systematically searches the parameter grid and performs internal cross-validation to identify the best-performing configuration. For the ANN model, a fixed architecture was empirically selected based on common best practices in deep learning, due to the high computational cost and structural complexity involved in exhaustive hyperparameter search. After tuning, each model was re-trained using the optimal hyperparameters on the full training set and evaluated on the independent test set. The predicted prices were compared to the actual values, and five evaluation metrics were computed and reported: R^2, MSE, RMSE, MAE, and MAPE. The performance results of all models were

summarized in a comparison table to facilitate comprehensive performance analysis (see Table 3).

All implementations were conducted in Python 3.11 using libraries including Pandas, NumPy, Scikit-Learn, LightGBM, Keras, Seaborn, and Matplotlib.

4. Analysis and Results

This section provides a comprehensive evaluation of the five machine learning models, based on their training and validation results, developed to predict rental property prices. The analysis covers hyperparameter tuning, performance comparison based on multiple evaluation metrics, error distribution interpretation, and diagnostic visualisations such as residual plots. Together, these assessments identify the most effective model and highlight key insights for later business-oriented discussions. See Appendix A.5.

4.1 Hyperparameter Optimisation

Prior to performance evaluation, a rigorous hyperparameter tuning process was applied to optimise each model. For KNN, Random Forest, Gradient Boosting, and LightGBM, parameter optimisation was carried out using grid-based search strategies in conjunction with five-fold cross-validation. For the ANN model, a manually selected architecture was implemented based on empirical testing. A summary of the final parameter configurations for all models is presented in Table 2.

Table 2. Best Hyperparameters for the Machine Learning Algorithms

Algorithm	Hyperparameter	Best Value
K-Nearest Neighbours	number of neighbours	10
Random Forest	max depth	20
	min samples per leaf	1
	min samples to split	2
	number of trees	100
Gradient Boosting	learning rate	0.05
	max depth	5
	min samples per leaf	5
	min samples to split	5
	number of trees	300
	subsample ratio	0.8
LightGBM	column subsampling	0.8
	learning rate	0.05
	max depth	10
	number of trees	300
	number of leaves	31
	subsample ratio	0.8

For the K-Nearest Neighbours (KNN) model, optimal performance was achieved when the number of neighbours was set to 10. For the Random Forest model, peak performance was achieved with a tree depth limit of 20, a leaf size threshold of one, a node split minimum of two samples, and 100 decision trees. For the Gradient Boosting model, the optimal configuration

consisted of a learning rate set to 0.05, a tree depth cap of 5, 5 samples per leaf and per split, 300 trees, and a subsample ratio of 0.8. For the LightGBM model, the best performance was obtained with a column sampling rate of 0.8 , a learning rate of 0.05, tree depth constrained to 10, 300 boosting rounds, 31 leaves per tree , and a subsample ratio of 0.8— indicating strong generalisation and fitting ability.

Additionally, the ANN algorithm consisted of three fully connected layers comprising 128, 64, and 32 neurons, respectively. To enhance non-linearity, the ReLU activation was employed, supplemented by batch normalization and a dropout mechanism set at 0.2 after each dense layer to reduce the risk of overfitting. Model training was conducted using the Adam optimizer, with a learning rate of 0.001, over 50 epochs and a mini- batch size of 64. Due to the unique structure of neural networks, hyperparameters were not tuned via grid search but were instead empirically determined prior to training.

4.2 Performance Evaluation

Upon completion of training, model performance was assessed through five commonly adopted regression indicators: R-squared, mean squared error, root mean squared error, mean absolute error, and mean absolute percentage error, denoted respectively as R^2, MSE, RMSE, MAE, and MAPE. A detailed summary of the evaluation outcomes is provided in Table 3.

Table 3. Model Performance Metrics Comparison

	R^2	MSE	RMSE	MAE	MAPE
KNN	0.7867	2154.8336	46.4202	29.2919	24.0285
Random Forest	0.9901	99.5916	9.9796	1.4694	0.6540
Gradient Boosting	0.9964	36.1768	6.0147	2.1817	1.3979
LightGBM	0.9911	89.6400	9.4678	3.1712	2.1614
ANN	0.9774	227.9126	15.0968	9.1230	8.0537

The evaluation results highlight Gradient Boosting as the most effective model overall. It outperformed the others on three key evaluation metrics, achieving the highest coefficient of determination (R^2=0.9964), a minimum Mean Squared Error at 36.18, and the lowest RMSE at 6.01. These outcomes suggest that Gradient Boosting not only captures a high proportion of variance in rental prices but also delivers superior predictive accuracy.

The Random Forest algorithm also demonstrated notable strength in controlling absolute and relative errors. Specifically, it achieved the lowest MAE (1.47) and MAPE (0.65), suggesting that it is more robust in reducing average absolute error and relative percentage deviation. This is particularly valuable in applications requiring stability across different price ranges.

While the LightGBM and ANN models demonstrated reasonably good results, they were slightly outperformed by Gradient Boosting and Random Forest. KNN, in contrast, consistently underperformed across all five metrics, with the lowest R^2 and the highest error values, suggesting that it struggles to capture complex

patterns in high-dimensional data.

4.3 Kernel Density Estimation of Prediction Errors

To further validate the evaluation metrics, Figure 5 presents the kernel density estimation plots of the percentage prediction errors for all five models. These plots provide an intuitive visual comparison of predictive stability and bias.

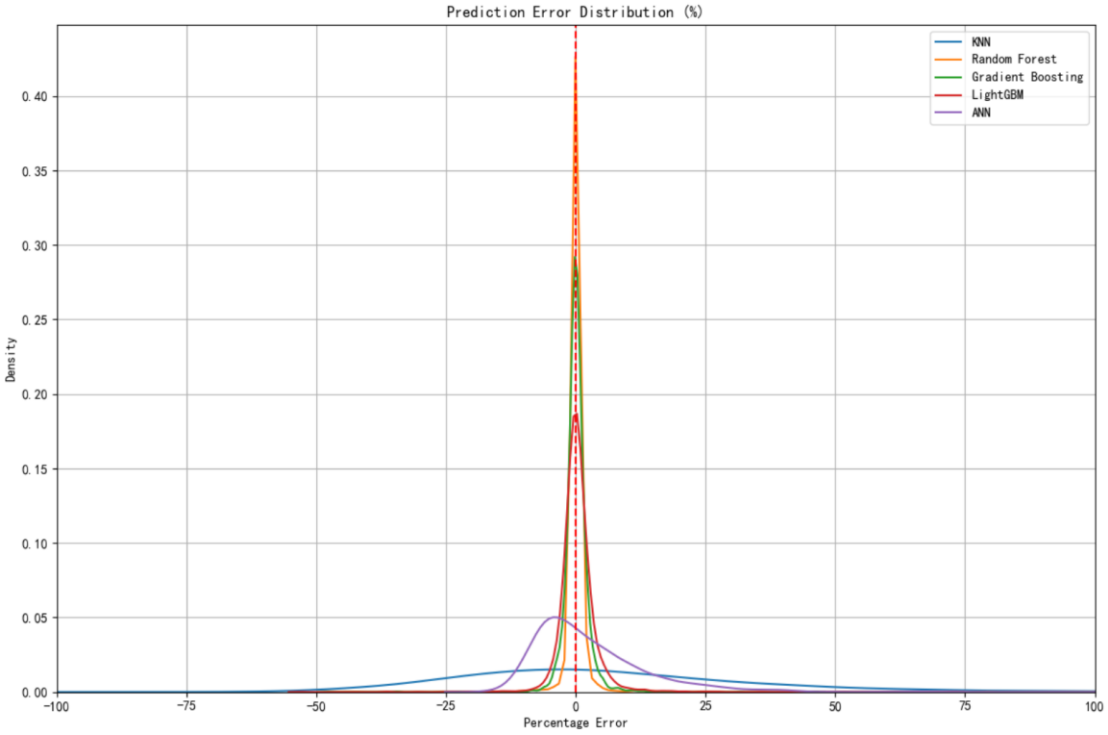


Figure 5. Kernel Density Estimation Plots of Prediction Errors

The KDE plots reveal that the error distributions of the Gradient Boosting and Random Forest models are sharply peaked and symmetrically centred around zero, indicating low variance and high predictive precision. In particular, the Random Forest model exhibits the narrowest and tallest distribution curve, which reflects exceptional consistency in its predictions.

By contrast, the K-Nearest Neighbours model displays the broadest and flattest error distribution, with prediction errors widely dispersed. This pattern aligns

with its poor performance across all evaluation metrics and highlights its limited capacity to capture complex, high-dimensional relationships.

Furthermore, the error curve of the Artificial Neural Network model appears moderately skewed, showing a noticeable deviation from zero. This asymmetry suggests the presence of systematic bias in its predictions, possibly resulting from empirically chosen hyperparameters rather than a rigorously tuned configuration via grid search.

4.4 Residual Analysis

To further understand model behaviour across different price ranges, Figure 6 presents the residual plots for all five models. Unlike the kernel density plots in Figure 5, which illustrate the overall concentration and spread of prediction errors, residual plots provide a diagnostic perspective by showing how prediction errors vary with the predicted price values. These plots are particularly useful in identifying systematic prediction bias and potential heteroskedasticity.

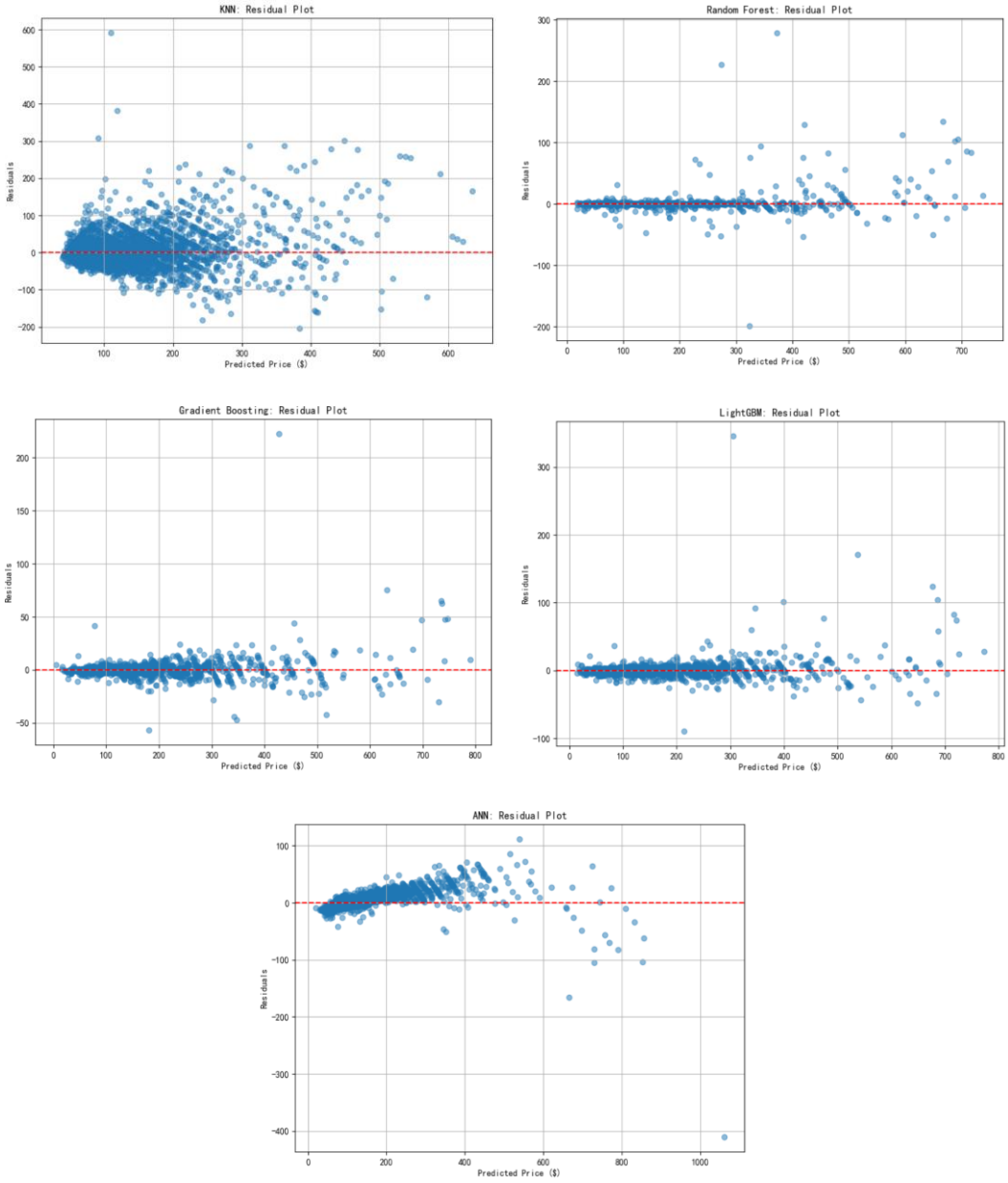


Figure 6. Residual Plots for All Models

As shown in the plots, the residuals of the Gradient Boosting and Random Forest models are randomly and symmetrically distributed around zero, without displaying any discernible trend or funnel-shaped dispersion. This indicates that both models produce stable and unbiased predictions across the entire price range, reinforcing their strong performance in terms of RMSE and MAE. Their prediction errors are not only small but also consistent, which confirms their robustness and reliability in the rental price prediction task. And the LightGBM

model demonstrates a similarly concentrated residual pattern, though a few large deviations are observed in the upper price segments. Despite the presence of these outliers, the overall residual distribution remains balanced, suggesting that LightGBM maintains solid generalisation capability while offering practical benefits in time- and resource-constrained scenarios.

In contrast, the residual distribution of the Artificial Neural Network model deviates significantly from the ideal pattern. A clear negative trend emerges as the predicted price increases, indicating a tendency to overestimate rental prices, particularly in higher price ranges. This pattern is accompanied by increasing variance in residuals, which suggests the presence of heteroskedasticity. These issues may stem from underfitting or insufficient hyperparameter tuning, especially considering that the ANN model was constructed based on empirical design choices rather than a systematic grid search approach. As such, its ability to capture complex nonlinear relationships appears limited.

Finally, the K-Nearest Neighbours model performs the worst among all models, as evidenced by a wide and irregular spread of residuals across all predicted price levels. The residuals exhibit strong heteroskedasticity and randomness, especially in the mid-to-high price range, indicating that the model lacks the capacity to effectively learn from high-dimensional or nonlinear data. This aligns with its consistently poor performance across all quantitative evaluation metrics.

4.5 Summary of Technical Evaluation

In conclusion, the analysis across both quantitative metrics and visual diagnostics consistently identifies Gradient Boosting as the top-performing

model, offering a compelling balance of high goodness-of-fit, low absolute and relative errors, and stable residual behaviour. The Random Forest model demonstrates a comparable ability to reduce prediction variance, making it well-suited for tasks where predictive precision is a priority. When fast execution and minimal resource usage are critical, LightGBM proves to be a resource-efficient and scalable modelling option.

In contrast, ANN and KNN demonstrate structural or algorithmic limitations under the current configuration and dataset. ANN's bias and variance issues could potentially be mitigated through more extensive hyperparameter tuning and data augmentation. KNN’s limitations, however, appear more fundamental given its non-parametric, distance-based nature, which does not scale well with feature complexity.

These comprehensive results form the empirical basis for discussing business implications, model deployment strategies, and future development directions in the next section. In addition to selecting the best model, these findings also illustrate the critical importance of robust model evaluation in practical applications. Numerical performance alone may not fully capture the strengths and weaknesses of each model unless supported by clear visual diagnostics. For instance, although ANN displayed a reasonable R^2 value, its residual and error distribution patterns revealed potential reliability issues under varying price levels. Similarly, while LightGBM did not outperform Gradient Boosting in any single metric, its consistent and balanced behaviour across multiple diagnostics makes it a strong candidate in scenarios where computational efficiency is valued. The alignment of metric-based insights with graphical patterns reinforces the credibility of the conclusions drawn. Ultimately, this multi-layered evaluation approach provides a solid analytical foundation for

integrating predictive modelling into rental investment decisions.

5. Discussion and Conclusion

This final section provides a comprehensive discussion of the technical findings, practical implications, and potential limitations of the rental price prediction system developed in this study. It also outlines directions for future research and deployment strategies.

5.1 Conclusion

This study developed and evaluated five machine learning models to predict rental property prices. Based on multiple evaluation metrics (R^2, MSE, RMSE, MAE, MAPE) and graphical error diagnostics (including kernel density estimation and residual plots), the Gradient Boosting model demonstrated the best overall performance in terms of accuracy, consistency, and generalisation ability, followed by the Random Forest model. In contrast, KNN and ANN models showed weaker performance, particularly in handling high-dimensional data. These technical findings not only demonstrate the feasibility of applying machine learning to rental price prediction but also provide investors with a quantifiable and practical decision-support tool.

While Gradient Boosting outperformed other models, predictive accuracy alone is not sufficient for real-world deployment. For practical use, stakeholders must also understand the model's decision logic. To this end, feature importance analysis was conducted using the trained Random Forest model, providing interpretable insights into variable influence.

The analysis identified room type, accommodates, number of bathrooms and

bedrooms, and distance to airports as the most influential predictors. These variables represent a property's structural scale and market value logic. Ensuring their availability enhances prediction accuracy and supports effective screening. Thus, both platforms and investors should prioritise collecting and standardising these features to ensure robust inputs and enable scalable decision-making workflows.

5.2 Practical Implications

Beyond technical validation, this study provides actionable guidance for both investors and platform operators. The proposed predictive system can support multiple stages of decision-making, from initial screening to operational integration, across two key dimensions: investment evaluation and organisational deployment.

Firstly, from a managerial perspective, the predictive system proves especially effective in supporting early-stage screening and evaluation of investment opportunities. The model’s output—the predicted rental price—can serve as an objective estimate of income potential for a given listing. Investors can compare this estimate against the acquisition cost to approximate rental yield and assess the economic viability of the investment. While the model does not directly compute return on investment, rental income estimation remains a core step in investment assessment. Among the five models evaluated, Gradient Boosting delivered the most dependable and precise estimates, making it particularly suitable for practical deployment. Compared to intuition or manual valuation, this system offers a scalable, consistent, and transparent alternative — especially helpful when analysing large portfolios across markets.

Secondly, from an organisational implementation perspective, operationalising the model within business workflows would require modest but essential structural changes. For instance, rental platforms should ensure that key predictive variables—such as room type, accommodates, and number of bathrooms—are embedded into the property submission forms to support structured and complete data collection. Meanwhile, data teams would need to develop automated pipelines for preprocessing, standardisation, and validation to ensure reliable model inputs across different use cases.

In addition, front-end development teams could design user-friendly interfaces that allow non-technical users, such as landlords or agents, to submit property information and instantly receive rental price predictions along with analytical feedback. To improve user trust and understanding, visual diagnostic tools—such as error distribution plots, residual charts, and “predicted vs. actual” comparison graphs—should be embedded directly into the interface. These visualisations serve not only as intuitive explanations of model behaviour but also as accessible evidence of performance, helping users grasp both the strengths and limitations of the system. Including such tools as part of the user experience can promote greater transparency, facilitate cross-team communication, and encourage broader model adoption.

5.3 Limitations and Future Directions

A number of constraints remain to be considered, even though the model has shown substantial effectiveness.

First, this study focuses solely on rental price prediction and does not incorporate holding costs, property taxes, or management fees, thus preventing

direct calculation of total ROI. Second, although the model includes a wide array of structural and geographic features, certain factors that may significantly affect rental prices—such as floor level, renovation quality, or local commercial amenities—were not available in the current dataset. Future improvements could incorporate both structured and unstructured data sources, such as listing images or user reviews, to further enhance model precision. Third, the ANN model underperformed in this study, likely due to limited tuning. More advanced neural architectures, extended training durations, or larger datasets may improve its performance in future experiments.

In summary, this project has developed a rental price prediction system that combines technical accuracy with managerial relevance. The model offers practical utility for real estate investors during the selection and pricing phases and can be deployed in organisational workflows with modest adaptation. Future work may extend this approach to other cities, time frames, or platforms to test its generalisability, and explore end-to-end investment advisory systems that incorporate rental predictions into broader financial evaluation frameworks.

References

1. AirDNA. (2023). *New York Short-Term Rental Market Report*. Available at: https://www.airdna.co
2. CBRE Research. (2023). *AI in Real Estate: How Data Science Is Reshaping Property Valuation*. Available at: https://www.cbre.com/insights
3. Zillow. (2024). *Zestimate: Automated Home Valuation Model*. Available at: https://www.zillow.com/z/zestimate
4. Anaplan. (2023). *Connected Planning in Real Estate*. Available at: https://www.anaplan.com/customers
5. ImmoScout24. (2024). *WohnBarometer: Market Price Prediction for Rentals*. Available at: https://www.immobilienscout24.de
6. Marr, B. and Ward, M. (2019). *Artificial Intelligence in Practice: How 50 Successful Companies Used AI and Machine Learning to Solve Problems*. John Wiley & Sons.
7. Tchuente, D. and Nyawa, S., 2022. *Real estate price estimation in French cities using geocoding and machine learning*. Annals of Operations Research. Available at: https://doi.org/10.1007/s10479-021-03932-5
8. Zhao, Y., Chetty, G. and Tran, D., 2023. *Real estate price prediction on generative language models*. In: 2023 International Conference on Computational Science and Data Engineering (CSDE). IEEE. Available at: https://doi.org/10.1109/CSDE59766.2023.10487658
9. Hernes, M., Tutak, P. and Siewiera, M. (2024). *Prediction of residential real estate price on primary market using machine learning*. Procedia

Computer Science, 246, pp.3142–3147. https://doi.org/10.1016/j.procs.2024.09.358

10. Hernes, M., Tutak, P., Nadolny, M. and Mazurek, A. (2024). *Real estate valuation using machine learning*. Procedia Computer Science, 246, pp.4592–4599. Available at: https://doi.org/10.1016/j.procs.2024.09.323

Appendix A.1

1. Preprocessing_Location

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import os
```

In [3]:
```
df = pd.read_csv('original data.csv')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74111 entries, 0 to 74110
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   id                      74111 non-null  int64  
 1   log_price               74111 non-null  float64
 2   property_type           74111 non-null  object 
 3   room_type               74111 non-null  object 
 4   amenities               74111 non-null  object 
 5   accommodates            74111 non-null  int64  
 6   bathrooms               73911 non-null  float64
 7   bed_type                74111 non-null  object 
 8   cancellation_policy     74111 non-null  object 
 9   cleaning_fee            74111 non-null  bool   
 10  city                    74111 non-null  object 
 11  description             74111 non-null  object 
 12  first_review            58247 non-null  object 
 13  host_has_profile_pic    73923 non-null  object 
 14  host_identity_verified  73923 non-null  object 
 15  host_response_rate      55812 non-null  object 
 16  host_since              73923 non-null  object 
 17  instant_bookable        74111 non-null  object 
 18  last_review             58284 non-null  object 
 19  latitude                74111 non-null  float64
 20  longitude               74111 non-null  float64
 21  name                    74111 non-null  object 
 22  neighbourhood           67239 non-null  object 
 23  number_of_reviews       74111 non-null  int64  
 24  review_scores_rating    57389 non-null  float64
 25  thumbnail_url           65895 non-null  object 
 26  zipcode                 73145 non-null  object 
 27  bedrooms                74020 non-null  float64
 28  beds                    73980 non-null  float64
dtypes: bool(1), float64(7), int64(3), object(18)
memory usage: 15.9+ MB
None
```

In [4]:
```
df_cleaned = df.dropna()
print(f"Total records after cleaning: {len(df_cleaned)} ")
```

```
Total records after cleaning: 38502
```

In [5]:
```
nyc_df = df_cleaned[df_cleaned['city'] == 'NYC']
print(f"Total records in NYC dataset after filtering: {len(nyc_df)} ")
```

```
Total records in NYC dataset after filtering: 17856
```

In [6]:
```
print(nyc_df.isnull().sum())
nyc_df.to_csv('cleaned_nyc_data.csv', index=False)
print("Cleaned data has been saved as 'cleaned_nyc_data.csv'")
```

```
id                        0
log_price                 0
property_type             0
room_type                 0
amenities                 0
accommodates              0
bathrooms                 0
bed_type                  0
cancellation_policy       0
cleaning_fee              0
city                      0
description               0
first_review              0
host_has_profile_pic      0
host_identity_verified    0
host_response_rate        0
host_since                0
instant_bookable          0
last_review               0
latitude                  0
longitude                 0
name                      0
neighbourhood             0
number_of_reviews         0
review_scores_rating      0
thumbnail_url             0
zipcode                   0
bedrooms                  0
beds                      0
dtype: int64
Cleaned data has been saved as 'cleaned_nyc_data.csv'
```

Location

1. import GeoJson File (downloaded from Github)

In [9]:
```
import geopandas as gpd
from shapely.geometry import Point
```

In [10]:

```
boroughs = gpd.read_file('boroughs.geojson')
print(boroughs.head())
df = pd.read_csv('cleaned_nyc_data.csv')
print(df.info())
```

```
   BoroCode       BoroName    Shape_Leng    Shape_Area  \
0         5  Staten Island  330385.03697  1.623853e+09
1         4         Queens  861038.47930  3.049947e+09
2         3       Brooklyn  726568.94634  1.959432e+09
3         1      Manhattan  358532.95642  6.364422e+08
4         2          Bronx  464517.89055  1.186804e+09

                                            geometry
0  MULTIPOLYGON (((-74.05051 40.56642, -74.05047 ...
1  MULTIPOLYGON (((-73.83668 40.59495, -73.83678 ...
2  MULTIPOLYGON (((-73.86706 40.58209, -73.86769 ...
3  MULTIPOLYGON (((-74.01093 40.68449, -74.01193 ...
4  MULTIPOLYGON (((-73.89681 40.79581, -73.89694 ...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17856 entries, 0 to 17855
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      17856 non-null  int64
 1   log_price               17856 non-null  float64
 2   property_type           17856 non-null  object
 3   room_type               17856 non-null  object
 4   amenities               17856 non-null  object
 5   accommodates            17856 non-null  int64
 6   bathrooms               17856 non-null  float64
 7   bed_type                17856 non-null  object
 8   cancellation_policy     17856 non-null  object
 9   cleaning_fee            17856 non-null  bool
 10  city                    17856 non-null  object
 11  description             17856 non-null  object
 12  first_review            17856 non-null  object
 13  host_has_profile_pic    17856 non-null  object
 14  host_identity_verified  17856 non-null  object
 15  host_response_rate      17856 non-null  object
 16  host_since              17856 non-null  object
 17  instant_bookable        17856 non-null  object
 18  last_review             17856 non-null  object
 19  latitude                17856 non-null  float64
 20  longitude               17856 non-null  float64
 21  name                    17856 non-null  object
 22  neighbourhood           17856 non-null  object
 23  number_of_reviews       17856 non-null  int64
 24  review_scores_rating    17856 non-null  float64
 25  thumbnail_url           17856 non-null  object
 26  zipcode                 17856 non-null  object
 27  bedrooms                17856 non-null  float64
```

```
 28  beds                    17856 non-null  float64
dtypes: bool(1), float64(7), int64(3), object(18)
memory usage: 3.8+ MB
None
```

2. classify data into five boroughs (according to lat/lon)

In [12]:
```python
def assign_borough(lat, lon, boroughs_gdf):
    point = Point(lon, lat)
    for _, borough in boroughs_gdf.iterrows():
        if borough['geometry'].contains(point):
            return borough['BoroName']
    return "Other"
```

In [13]:
```python
nyc_df= df[df['city'] == 'NYC'].copy()
nyc_df['geometry'] = nyc_df.apply(lambda row: Point(row['longitude'], row['latitude']), axis=1)
nyc_gdf = gpd.GeoDataFrame(nyc_df, geometry='geometry')

boroughs = boroughs.to_crs("EPSG:4326")
nyc_gdf.set_crs("EPSG:4326", inplace=True)

nyc_gdf = gpd.sjoin(nyc_gdf, boroughs, how='left', predicate='within')

nyc_gdf['borough'] = nyc_gdf['BoroName']

print(nyc_gdf['borough'].value_counts())

print(nyc_df['latitude'].min(), nyc_df['latitude'].max())
print(nyc_df['longitude'].min(), nyc_df['longitude'].max())
```

```
borough
Manhattan        8030
Brooklyn         7367
Queens           1946
Bronx             376
Staten Island     135
Name: count, dtype: int64
40.50868437 40.90389506
-74.23985905 -73.71737449
```

In [14]:
```python
missing_boroughs = nyc_gdf[nyc_gdf['borough'].isnull()]
print(f"Number of rows with missing borough: {missing_boroughs.shape[0]}")
print(missing_boroughs.head())
```

```
Number of rows with missing borough: 2
            id  log_price property_type        room_type  \
7414   7826323   3.871201         House     Private room
14933  4156913   5.703782         House  Entire home/apt

                                              amenities  accommodates  \
7414   {"Wireless Internet",Kitchen,"Free parking on ...             2
14933  {TV,"Wireless Internet","Air conditioning",Kit...             6

       bathrooms  bed_type cancellation_policy  cleaning_fee  ... zipcode  \
7414         1.0  Real Bed            flexible         False  ...   10550
14933        2.5  Real Bed            moderate          True  ...   11509

      bedrooms beds                     geometry index_right BoroCode BoroName  \
7414       1.0  1.0  POINT (-73.83585 40.89809)         NaN      NaN      NaN
14933      4.0  5.0  POINT (-73.73961 40.59057)         NaN      NaN      NaN

      Shape_Leng Shape_Area  borough
7414         NaN        NaN      NaN
14933        NaN        NaN      NaN

[2 rows x 36 columns]
```

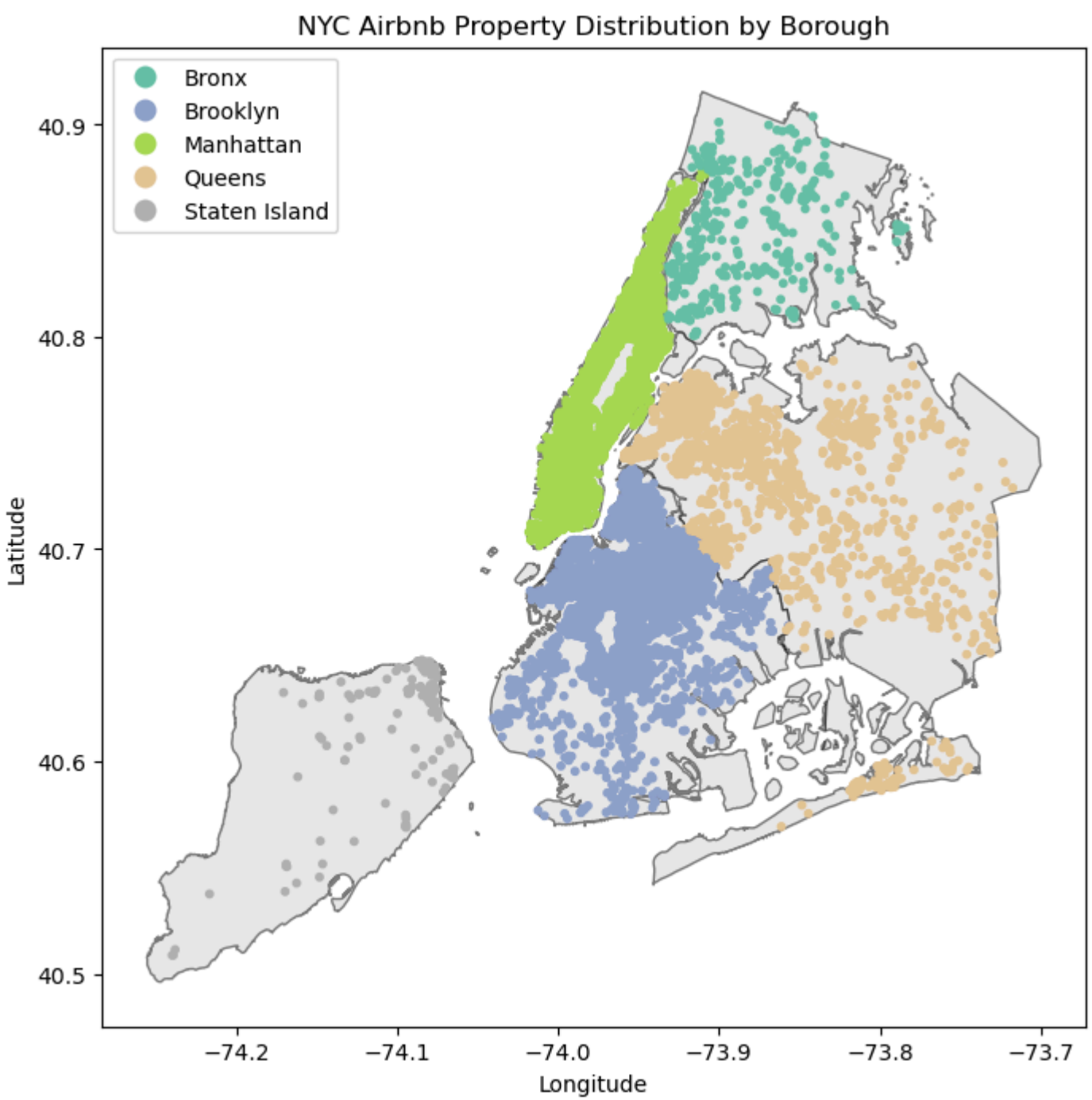
In [15]:
```python
nyc_gdf = nyc_gdf.dropna(subset=['borough'])
```

In [16]:
```python
nyc_gdf.drop(columns='geometry').to_csv('nyc_classified_data.csv', index=False)

print("Classified data has been successfully saved as 'nyc_classified_data.csv'")
```

```
Classified data has been successfully saved as 'nyc_classified_data.csv'
```

3. plot

In [18]:
```python
if not isinstance(nyc_gdf, gpd.GeoDataFrame):
    print("Please ensure the data is in GeoDataFrame format and contains the geometry column")
else:
    fig, ax = plt.subplots(figsize=(12, 8))
    boroughs.plot(ax=ax, color='lightgrey', edgecolor='black', alpha=0.5)
    nyc_gdf.plot(column='borough', ax=ax, markersize=10, legend=True, cmap='Set2')
    plt.title("NYC Airbnb Property Distribution by Borough")
    plt.xlabel("Longitude")
    plt.ylabel("Latitude")
    plt.show()
```



Distance to Airport

In [20]:
```
nyc_df = pd.read_csv('nyc_classified_data.csv')
```

```
print(nyc_df.columns)
```

```
Index(['id', 'log_price', 'property_type', 'room_type', 'amenities',
       'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy',
       'cleaning_fee', 'city', 'description', 'first_review',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rate',
       'host_since', 'instant_bookable', 'last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'number_of_reviews',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds',
       'index_right', 'BoroCode', 'BoroName', 'Shape_Leng', 'Shape_Area',
       'borough'],
      dtype='object')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_8048\3986478363.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on impor
t or set low_memory=False.
  nyc_df = pd.read_csv('nyc_classified_data.csv')
```

In [21]:
```
def haversine(lat1, lon1, lat2, lon2):
    R = 6371
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2) ** 2
    c = 2 * np.arcsin(np.sqrt(a))
    return R * c

jfk_airport = (40.6413, -73.7781)
lga_airport = (40.7769, -73.8740)
ewr_airport = (40.6895, -74.1745)
```

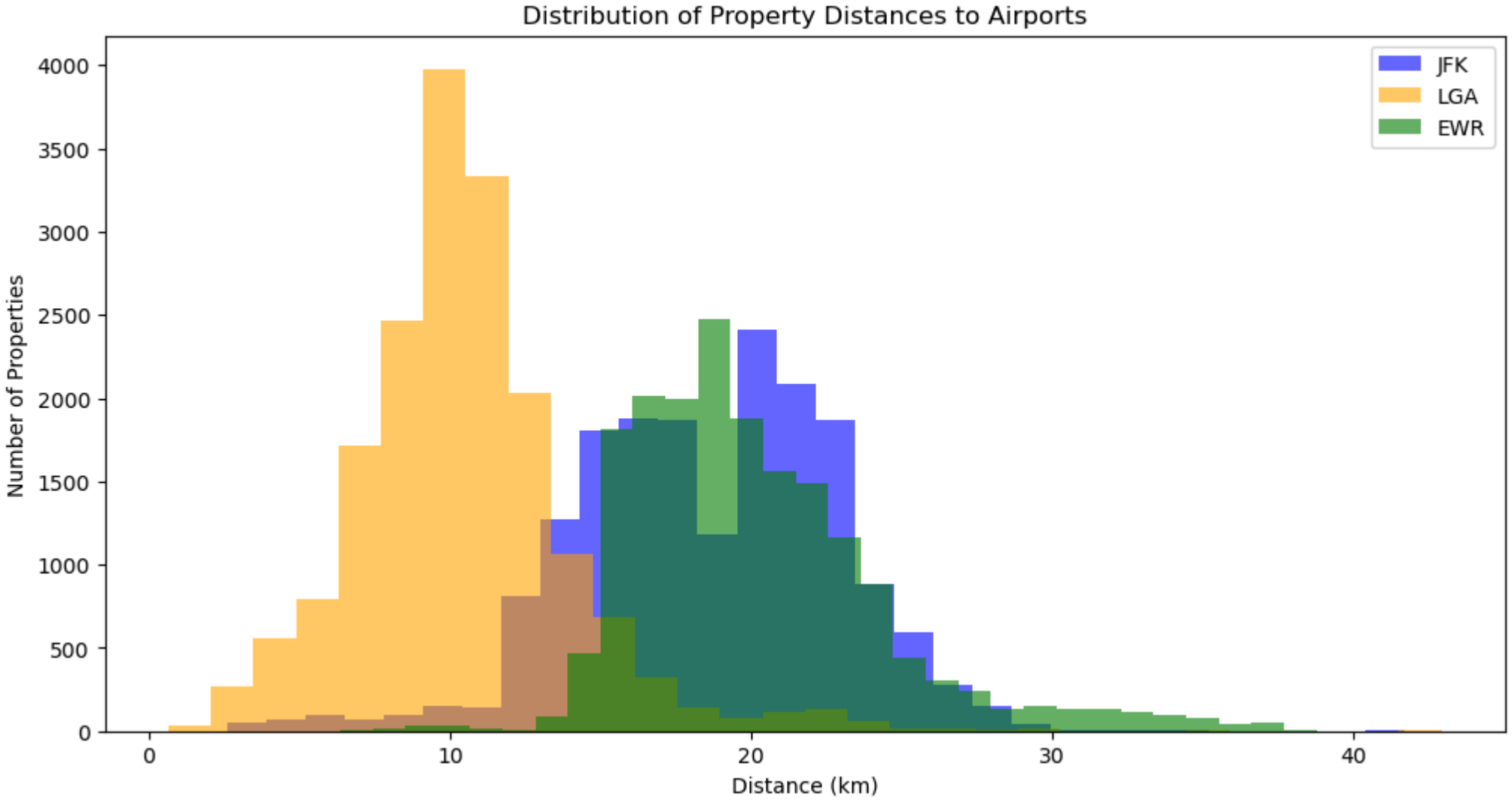
In [22]:
```
nyc_df['distance_to_jfk'] = nyc_df.apply(
    lambda row: haversine(row['latitude'], row['longitude'], jfk_airport[0], jfk_airport[1]),
    axis=1
)
nyc_df['distance_to_lga'] = nyc_df.apply(
    lambda row: haversine(row['latitude'], row['longitude'], lga_airport[0], lga_airport[1]),
    axis=1
)
nyc_df['distance_to_ewr'] = nyc_df.apply(
    lambda row: haversine(row['latitude'], row['longitude'], ewr_airport[0], ewr_airport[1]),
    axis=1
)
print(nyc_df[['id','distance_to_jfk', 'distance_to_lga', 'distance_to_ewr']].head())
```

	id	distance_to_jfk	distance_to_lga	distance_to_ewr
0	6304928	22.556586	9.761466	17.799104
1	7919400	23.214594	6.821102	23.489703
2	5578513	19.631988	10.973902	16.516343
3	18224863	14.578484	16.158419	19.536180
4	16679342	12.307846	14.025187	21.558997

In [23]:
```
nyc_df.to_csv('nyc_with_distances.csv', index=False)
print("Computation completed, data saved as 'nyc_with_distances.csv'")
```

```
Computation completed, data saved as 'nyc_with_distances.csv'
```

In [24]:
```
plt.figure(figsize=(12, 6))

plt.hist(nyc_df['distance_to_jfk'], bins=30, alpha=0.6, label='JFK', color='blue')
plt.hist(nyc_df['distance_to_lga'], bins=30, alpha=0.6, label='LGA', color='orange')
plt.hist(nyc_df['distance_to_ewr'], bins=30, alpha=0.6, label='EWR', color='green')

plt.title("Distribution of Property Distances to Airports")
plt.xlabel("Distance (km)")
plt.ylabel("Number of Properties")
plt.legend()
plt.show()
```



Distance to tourist attractions

In [26]:
```
nyc_df = pd.read_csv('nyc_with_distances.csv')
print(nyc_df.columns)
```

```
Index(['id', 'log_price', 'property_type', 'room_type', 'amenities',
       'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy',
       'cleaning_fee', 'city', 'description', 'first_review',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rate',
       'host_since', 'instant_bookable', 'last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'number_of_reviews',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds',
       'index_right', 'BoroCode', 'BoroName', 'Shape_Leng', 'Shape_Area',
       'borough', 'distance_to_jfk', 'distance_to_lga', 'distance_to_ewr'],
      dtype='object')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_8048\2791484958.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on impor
t or set low_memory=False.
  nyc_df = pd.read_csv('nyc_with_distances.csv')
```

1. Bronx

In [28]:
```
bronx_df = nyc_df[nyc_df['borough'] == 'Bronx'].copy()
```

In [29]:
```
bronx_landmarks = {
    "New York Botanical Garden": (40.8617, -73.8801),
    "Bronx Zoo": (40.8506, -73.8763),
    "Yankee Stadium": (40.8296, -73.9262)
}
```

In [30]:
```
prefix = 'bronx'

for landmark, coords in bronx_landmarks.items():
    bronx_df[f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'] = bronx_df.apply(
        lambda row: haversine(row['latitude'], row['longitude'], coords[0], coords[1]),
        axis=1
    )

print(bronx_df[[col for col in bronx_df.columns if f'{prefix}_' in col]].head())
```

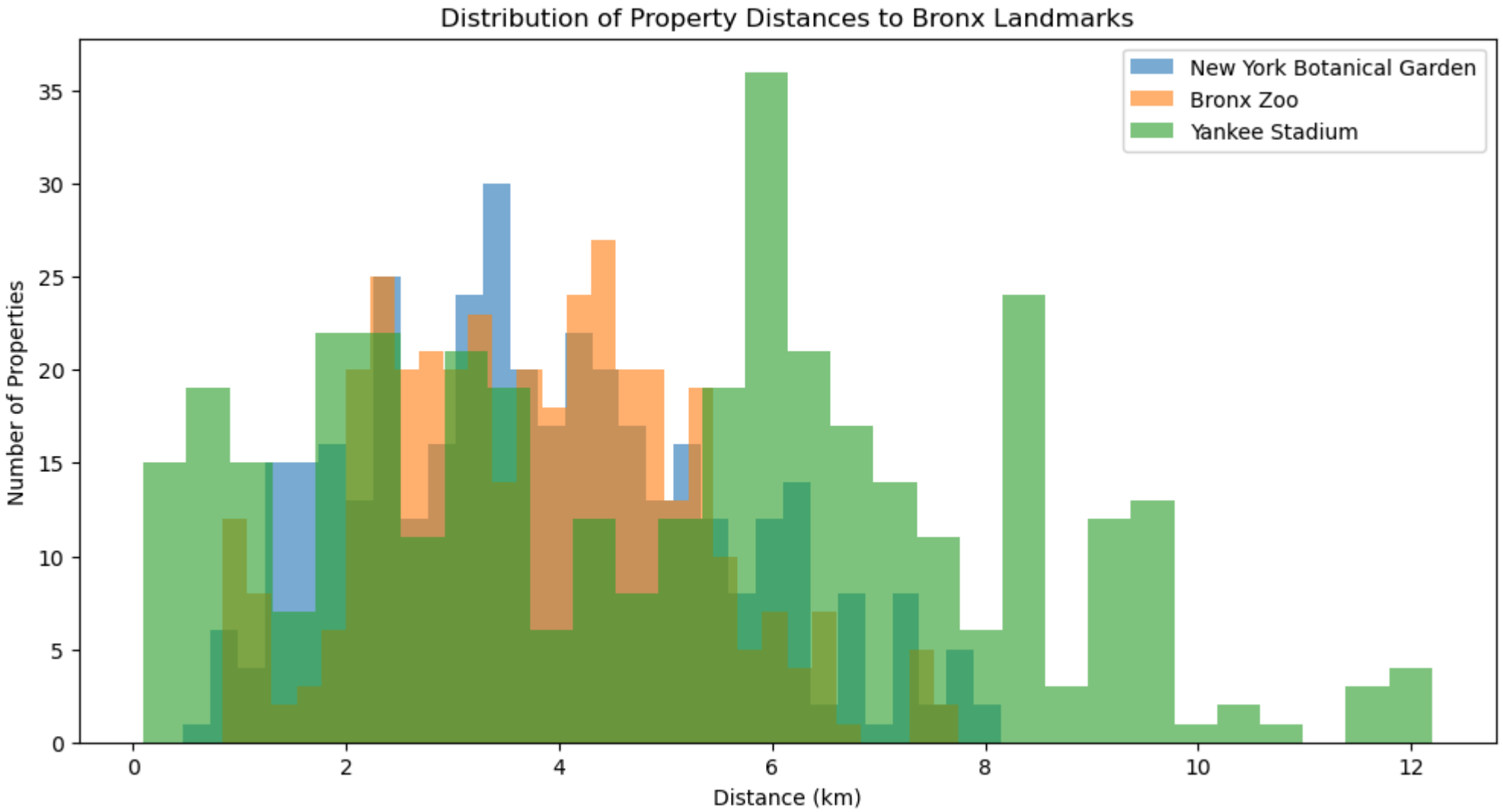
```
     bronx_new_york_botanical_garden_distance  bronx_bronx_zoo_distance  \
38                                   6.286574                 5.021895
114                                  4.281037                 4.769779
135                                  5.635136                 4.746414
178                                  3.315623                 4.357397
199                                  5.319659                 4.455665

     bronx_yankee_stadium_distance
38                        6.551017
114                       9.488413
135                       2.014750
178                       8.452701
199                       1.825599
```

In [31]:
```
new_columns = ['bronx_new_york_botanical_garden_distance',
               'bronx_bronx_zoo_distance',
               'bronx_yankee_stadium_distance']

print(bronx_df[new_columns].head())

for col in new_columns:
    nyc_df.loc[nyc_df['borough'] == 'Bronx', col] = bronx_df[col]


output_path = 'nyc_with_distances.csv'
nyc_df.to_csv(os.path.expanduser(output_path), index=False)

print(f"The new columns have been successfully added and saved to {output_path}")
```

```
     bronx_new_york_botanical_garden_distance  bronx_bronx_zoo_distance  \
38                                   6.286574                 5.021895
114                                  4.281037                 4.769779
135                                  5.635136                 4.746414
178                                  3.315623                 4.357397
199                                  5.319659                 4.455665

     bronx_yankee_stadium_distance
38                        6.551017
114                       9.488413
135                       2.014750
178                       8.452701
199                       1.825599
The new columns have been successfully added and saved to nyc_with_distances.csv
```

In [32]:
```
plt.figure(figsize=(12, 6))
for landmark in bronx_landmarks.keys():
    column_name = f'bronx_{landmark.replace(" ", "_").lower()}_distance'
```

```
    plt.hist(bronx_df[column_name], bins=30, alpha=0.6, label=landmark)

plt.title("Distribution of Property Distances to Bronx Landmarks")
plt.xlabel("Distance (km)")
plt.ylabel("Number of Properties")
plt.legend()
plt.show()
```



2. Brooklyn

In [34]:
```
brooklyn_df = nyc_df[nyc_df['borough'] == 'Brooklyn'].copy()
```

In [35]:
```
brooklyn_landmarks = {
    "Brooklyn Bridge": (40.7061, -73.9969),
    "Brooklyn Museum": (40.6712, -73.9636),
    "Prospect Park": (40.6602, -73.9690),
}
```

In [36]:
```
prefix = 'brooklyn'
for landmark, coords in brooklyn_landmarks.items():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    brooklyn_df[column_name] = brooklyn_df.apply(
        lambda row: haversine(row['latitude'], row['longitude'], coords[0], coords[1]),
        axis=1
    )
print(brooklyn_df[[col for col in brooklyn_df.columns if f'{prefix}_' in col]].head())
```

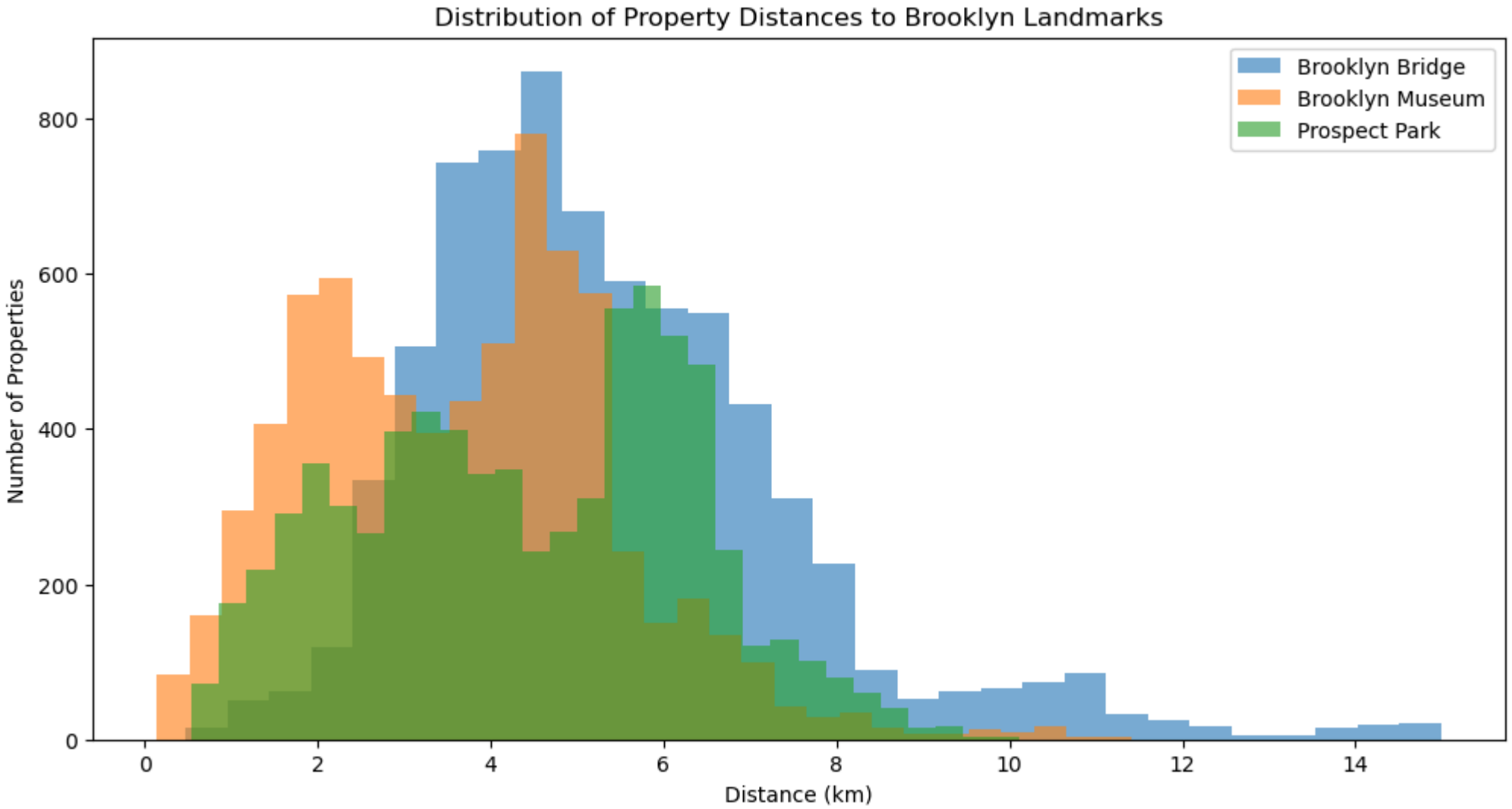
```
    brooklyn_brooklyn_bridge_distance  brooklyn_brooklyn_museum_distance  \
3                            7.944195                           3.233742
4                            8.360470                           3.832219
8                            6.437757                           1.678598
9                            8.753014                           4.429630
10                           4.473129                           5.130823

    brooklyn_prospect_park_distance
3                          2.383247
4                          3.934416
8                          1.718880
9                          4.681652
10                         6.434742
```

In [37]:
```
new_columns = ['brooklyn_brooklyn_bridge_distance',
               'brooklyn_brooklyn_museum_distance',
               'brooklyn_prospect_park_distance']

for col in new_columns:
    nyc_df.loc[nyc_df['borough'] == 'Brooklyn', col] = brooklyn_df[col]


output_path = 'nyc_with_distances.csv'
nyc_df.to_csv(os.path.expanduser(output_path), index=False)

print(f"The new columns have been successfully added and saved to {output_path}")
```

```
The new columns have been successfully added and saved to nyc_with_distances.csv
```

In [38]:
```
plt.figure(figsize=(12, 6))
for landmark in brooklyn_landmarks.keys():
```

```
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    plt.hist(brooklyn_df[column_name], bins=30, alpha=0.6, label=landmark)

plt.title("Distribution of Property Distances to Brooklyn Landmarks")
plt.xlabel("Distance (km)")
plt.ylabel("Number of Properties")
plt.legend()
plt.show()
```



3. Manhattan

In [40]:
```
manhattan_df = nyc_df[nyc_df['borough'] == 'Manhattan'].copy()
```

In [41]:
```
manhattan_landmarks = {
    "Times Square": (40.7580, -73.9855),
    "Central Park": (40.7851, -73.9683),
    "Empire State Building": (40.7488, -73.9854),
    "Wall Street": (40.7074, -74.0113),
    "The Met": (40.7794, -73.9632)
}
```

In [42]:
```
prefix = 'manhattan'
for landmark, coords in manhattan_landmarks.items():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    manhattan_df[column_name] = manhattan_df.apply(
        lambda row: haversine(row['latitude'], row['longitude'], coords[0], coords[1]),
        axis=1
    )

print(manhattan_df[[col for col in manhattan_df.columns if f'{prefix}_' in col]].head())
```

```
    manhattan_times_square_distance  manhattan_central_park_distance  \
0                          0.950369                         2.739744
1                          6.587855                         3.288634
2                          3.796060                         6.932345
7                          0.936819                         4.004979
13                         2.853119                         6.192325

    manhattan_empire_state_building_distance  manhattan_wall_street_distance  \
0                                   1.949643                       6.792900
1                                   7.469212                      12.560679
2                                   2.773571                       2.949548
7                                   0.422174                       5.422775
13                                  1.941820                       3.178777

    manhattan_the_met_distance
0                     2.629977
1                     3.587578
2                     6.414264
7                     3.554574
13                    5.821100
```

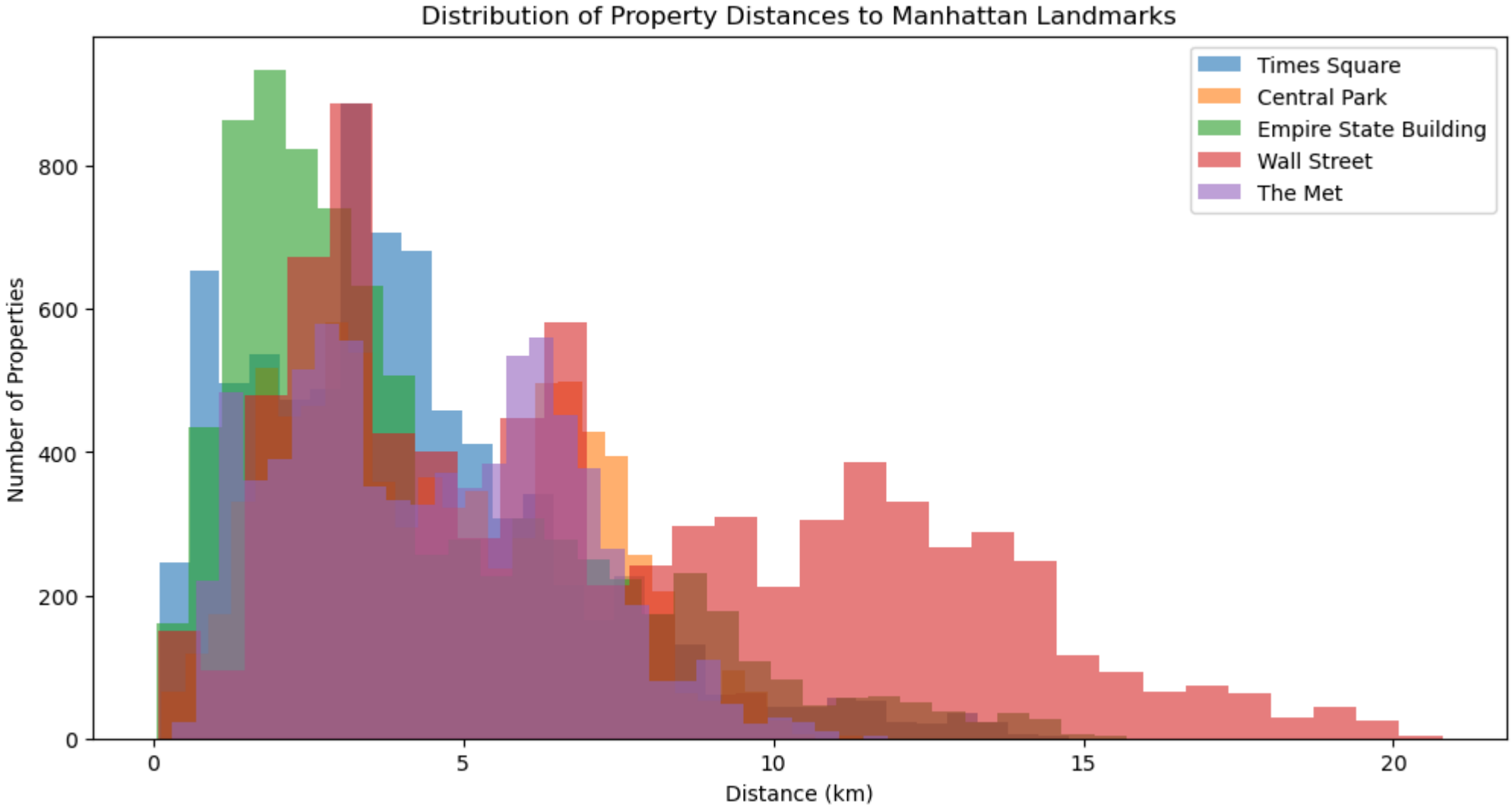
In [43]:
```
new_columns = ['manhattan_times_square_distance',
               'manhattan_central_park_distance',
               'manhattan_empire_state_building_distance',
              'manhattan_wall_street_distance',
              'manhattan_the_met_distance']

for col in new_columns:
```

```
    nyc_df.loc[nyc_df['borough'] == 'Manhattan', col] = manhattan_df[col]


output_path = 'nyc_with_distances.csv'
nyc_df.to_csv(os.path.expanduser(output_path), index=False)

print(f"The new columns have been successfully added and saved to {output_path}")
```

```
The new columns have been successfully added and saved to nyc_with_distances.csv
```

In [44]:

```
plt.figure(figsize=(12, 6))
for landmark in manhattan_landmarks.keys():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    plt.hist(manhattan_df[column_name], bins=30, alpha=0.6, label=landmark)

plt.title("Distribution of Property Distances to Manhattan Landmarks")
plt.xlabel("Distance (km)")
plt.ylabel("Number of Properties")
plt.legend()
plt.show()
```



4. Queens

In [46]:
```
queens_df = nyc_df[nyc_df['borough'] == 'Queens'].copy()
```

In [47]:
```
queens_landmarks = {
    "Flushing Meadows Corona Park": (40.7498, -73.8401),
    "Museum of the Moving Image": (40.7563, -73.9235),
    "Queens Botanical Garden": (40.7516, -73.8276)
}
```

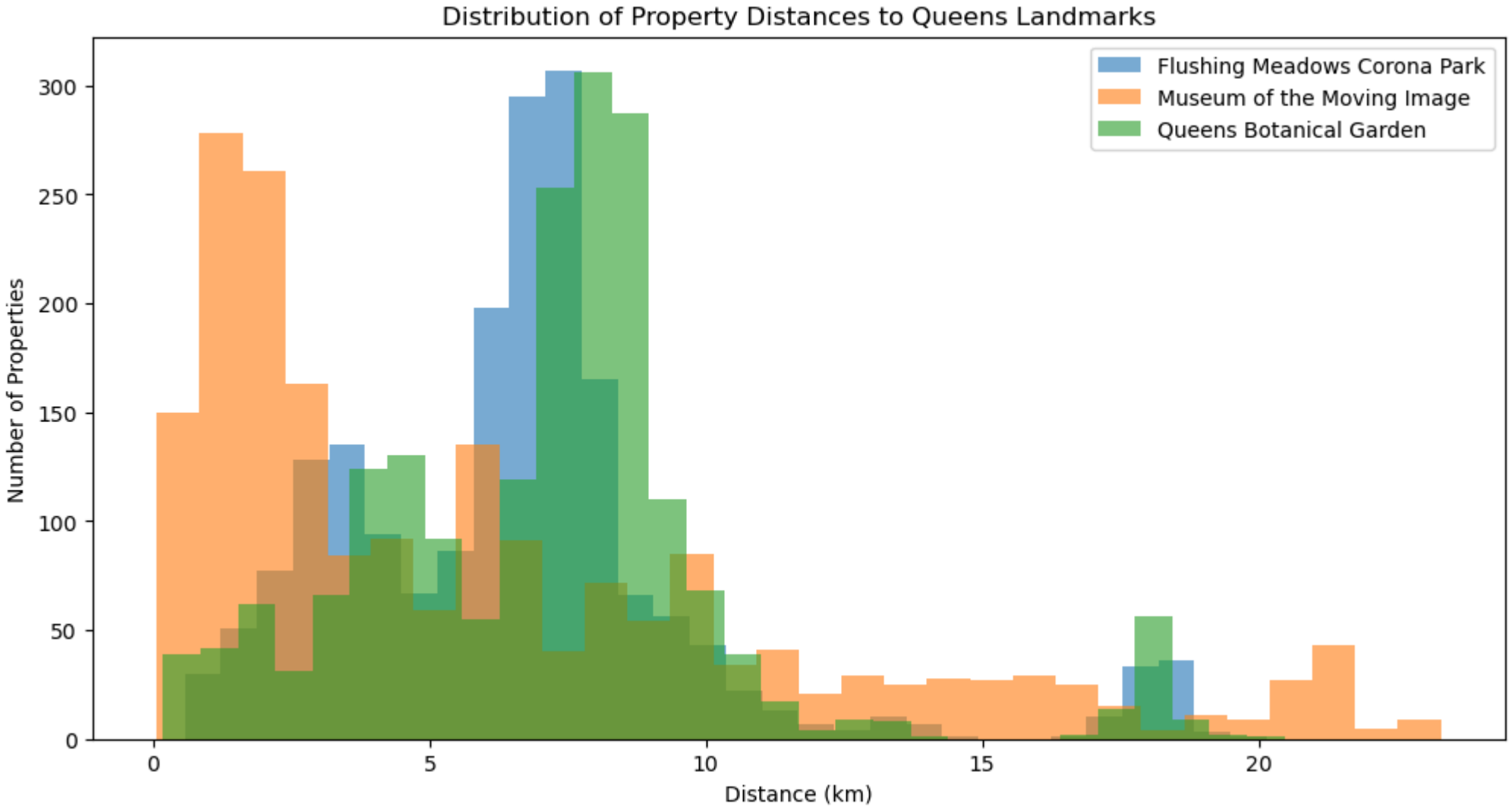
In [48]:
```
prefix = 'queens'
for landmark, coords in queens_landmarks.items():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    queens_df[column_name] = queens_df.apply(
        lambda row: haversine(row['latitude'], row['longitude'], coords[0], coords[1]),
        axis=1
    )

print(queens_df[[col for col in queens_df.columns if f'{prefix}_' in col]].head())
```

```
    queens_flushing_meadows_corona_park_distance  \
5                                      10.238762
6                                       2.388706
30                                      2.507286
66                                      7.079696
68                                     18.154708

    queens_museum_of_the_moving_image_distance  \
5                                    16.616944
6                                     6.723704
30                                    9.367813
66                                    4.725221
68                                   21.486637

    queens_queens_botanical_garden_distance
5                                 9.578515
6                                 3.124137
30                                1.464671
66                                8.081144
68                               18.148362
```

In [49]:
```
new_columns = ['queens_flushing_meadows_corona_park_distance',
               'queens_museum_of_the_moving_image_distance',
               'queens_queens_botanical_garden_distance']

for col in new_columns:
    nyc_df.loc[nyc_df['borough'] == 'Queens', col] = queens_df[col]


output_path = 'nyc_with_distances.csv'
nyc_df.to_csv(os.path.expanduser(output_path), index=False)

print(f"The new columns have been successfully added and saved to {output_path}")
```

```
The new columns have been successfully added and saved to nyc_with_distances.csv
```

In [50]:
```
plt.figure(figsize=(12, 6))
for landmark in queens_landmarks.keys():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    plt.hist(queens_df[column_name], bins=30, alpha=0.6, label=landmark)

plt.title("Distribution of Property Distances to Queens Landmarks")
plt.xlabel("Distance (km)")
plt.ylabel("Number of Properties")
plt.legend()
plt.show()
```



5. Staten Island

In [52]:
```
staten_island_df = nyc_df[nyc_df['borough'] == 'Staten Island'].copy()
```

In [53]:
```
staten_island_landmarks = {
    "Staten Island Ferry Terminal": (40.6437, -74.0732),
    "Staten Island Zoo": (40.6251, -74.1170),
    "Historic Richmond Town": (40.5696, -74.1455),
    "National Lighthouse Museum": (40.6439, -74.0718),
    "Staten Island Greenbelt Conservancy": (40.5795, -74.1452)
}
```

In [54]:
```
prefix = 'staten_island'
for landmark, coords in staten_island_landmarks.items():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    staten_island_df[column_name] = staten_island_df.apply(
        lambda row: haversine(row['latitude'], row['longitude'], coords[0], coords[1]),
        axis=1
    )

print(staten_island_df[[col for col in staten_island_df.columns if f'{prefix}_' in col]].head())
```

```
     staten_island_staten_island_ferry_terminal_distance  \
177                                           5.585215
224                                           0.863594
226                                           1.870417
336                                          10.114065
359                                           4.422318

     staten_island_staten_island_zoo_distance  \
177                                  1.703506
224                                  3.470383
226                                  3.145357
336                                  7.028271
359                                  0.897007

     staten_island_historic_richmond_town_distance  \
177                                       4.928042
224                                       9.709676
226                                       8.507243
336                                       1.779959
359                                       7.155613

     staten_island_national_lighthouse_museum_distance  \
177                                           5.688772
224                                           0.983162
226                                           1.930148
336                                          10.187218
359                                           4.541661

     staten_island_staten_island_greenbelt_conservancy_distance
177                                           3.925201
224                                           8.787854
226                                           7.686757
336                                           2.465900
359                                           6.092802
```

In [55]:

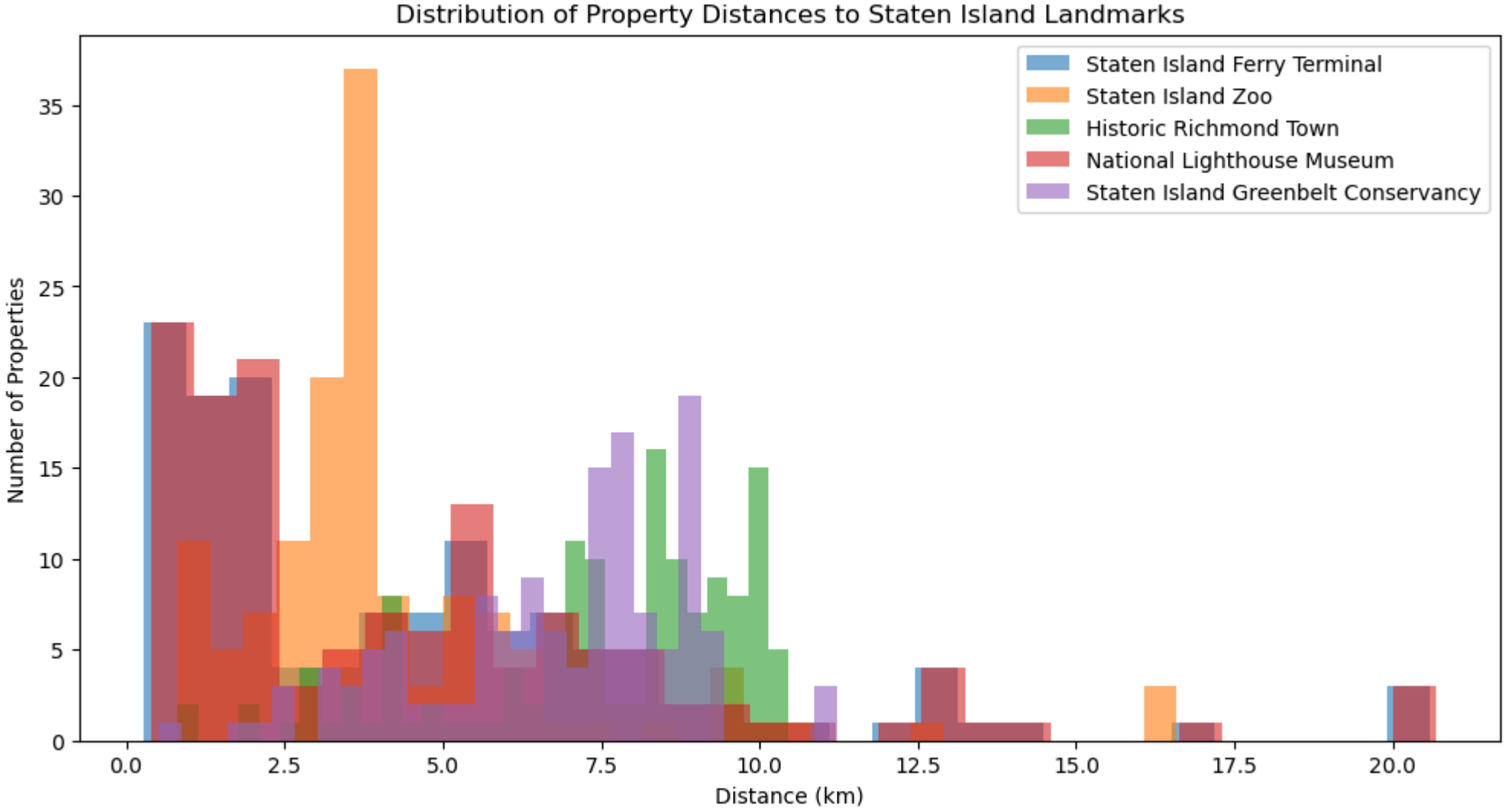
```
new_columns = ['staten_island_staten_island_ferry_terminal_distance',
               'staten_island_staten_island_zoo_distance',
               'staten_island_historic_richmond_town_distance',
              'staten_island_national_lighthouse_museum_distance',
              'staten_island_staten_island_greenbelt_conservancy_distance']

for col in new_columns:
    nyc_df.loc[nyc_df['borough'] == 'Staten Island', col] = staten_island_df[col]


output_path = 'nyc_with_distances.csv'
```

```
nyc_df.to_csv(os.path.expanduser(output_path), index=False)

print(f"The new columns have been successfully added and saved to {output_path}")
```

```
The new columns have been successfully added and saved to nyc_with_distances.csv
```

In [56]:

```
plt.figure(figsize=(12, 6))
for landmark in staten_island_landmarks.keys():
    column_name = f'{prefix}_{landmark.replace(" ", "_").lower()}_distance'
    plt.hist(staten_island_df[column_name], bins=30, alpha=0.6, label=landmark)

plt.title("Distribution of Property Distances to Staten Island Landmarks")
plt.xlabel("Distance (km)")
plt.ylabel("Number of Properties")
plt.legend()
plt.show()
```



plot

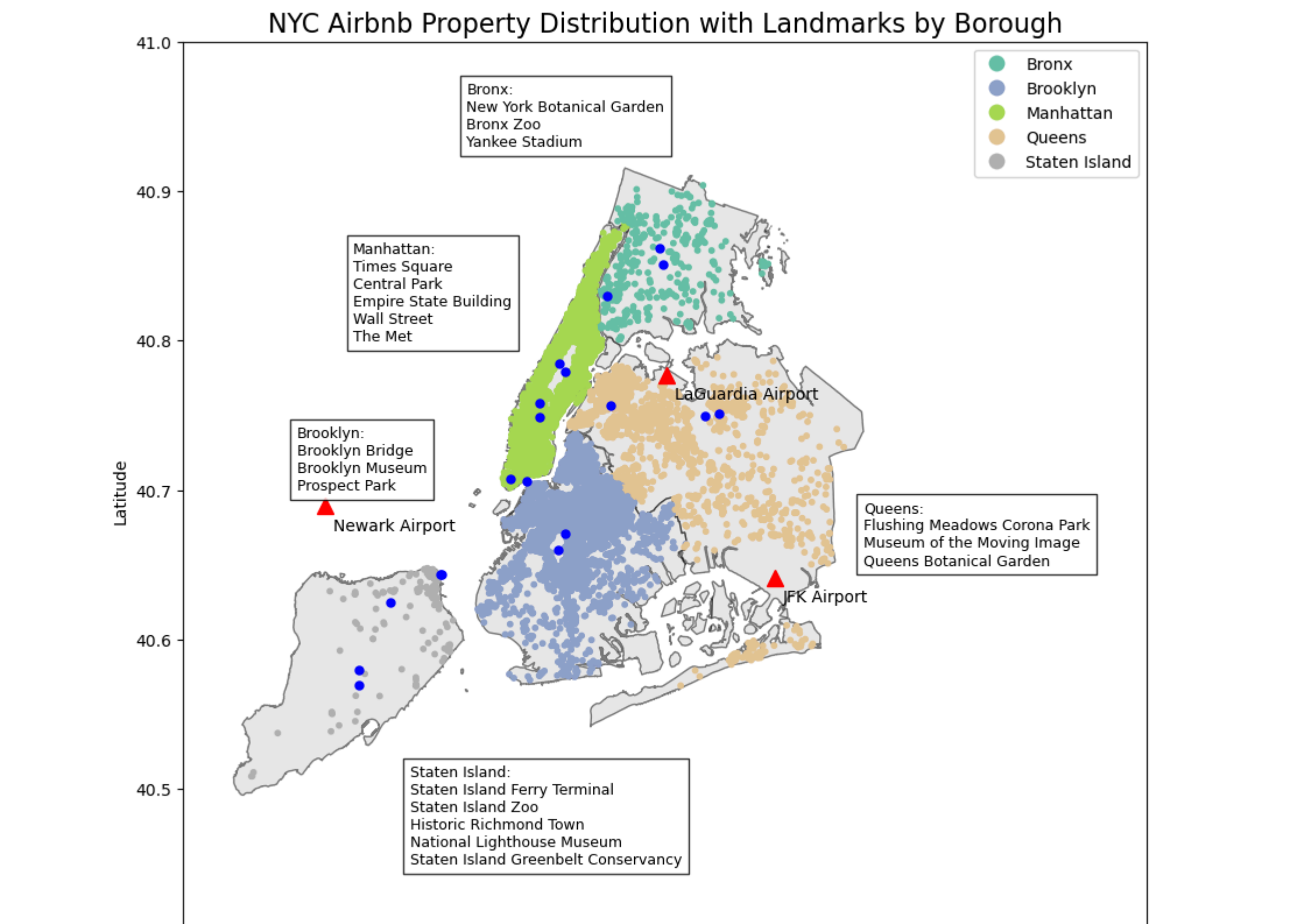
In [58]:
```
airports = {
    "JFK Airport": (40.6413, -73.7781) ,
    "LaGuardia Airport": (40.7769, -73.8740) ,
    "Newark Airport": (40.6895, -74.1745)
}

landmarks = {
    "Bronx": {
        "New York Botanical Garden": (40.8617, -73.8801),
        "Bronx Zoo": (40.8506, -73.8763),
        "Yankee Stadium": (40.8296, -73.9262)
```

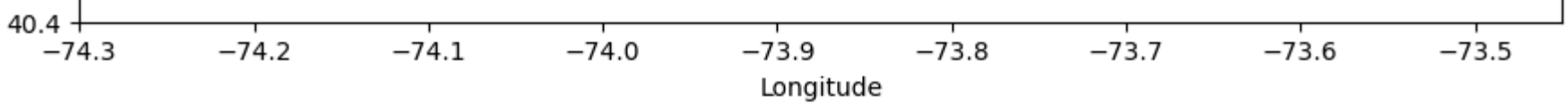
```
    },
    "Brooklyn": {
          "Brooklyn Bridge": (40.7061, -73.9969),
          "Brooklyn Museum": (40.6712, -73.9636),
          "Prospect Park": (40.6602, -73.9690),
    },
    "Manhattan": {
        "Times Square": (40.7580, -73.9855),
        "Central Park": (40.7851, -73.9683),
        "Empire State Building": (40.7488, -73.9854),
        "Wall Street": (40.7074, -74.0113),
        "The Met": (40.7794, -73.9632)
    },
    "Queens": {
         "Flushing Meadows Corona Park": (40.7498, -73.8401),
         "Museum of the Moving Image": (40.7563, -73.9235),
         "Queens Botanical Garden": (40.7516, -73.8276)
    },
    "Staten Island":{
        "Staten Island Ferry Terminal": (40.6437, -74.0732),
        "Staten Island Zoo": (40.6251, -74.1170),
        "Historic Richmond Town": (40.5696, -74.1455),
        "National Lighthouse Museum": (40.6439, -74.0718),
        "Staten Island Greenbelt Conservancy": (40.5795, -74.1452)
    }
}
```

In [59]:

```
fig, ax = plt.subplots(figsize=(12, 10))

boroughs.plot(ax=ax, color='lightgrey', edgecolor='black', alpha=0.5)
nyc_gdf.plot(column='borough', ax=ax, markersize=10, legend=True, cmap='Set2')

labels = {
    "Bronx": (-74.05, 40.93),
    "Brooklyn": (-74.20, 40.70),
    "Manhattan": (-74.15, 40.80),
    "Queens": (-73.70, 40.65),
    "Staten Island": (-74.1, 40.45)
}
for borough, points in landmarks.items():
    for name, coord in points.items():
        ax.plot(
            coord[1], coord[0],
            marker='o', color='blue', markersize=5, label=name if ax.get_legend() is None else ""
        )
for airport, coord in airports.items():
```

```
    ax.plot(
        coord[1], coord[0],
        marker='^', color='red', markersize=10, label=airport if ax.get_legend() is None else ""
    )
    ax.annotate(
    airport,
    xy=(coord[1], coord[0]),
    xytext=(5, -15),
    textcoords='offset points',
    fontsize=10,
    color='black'
    )

for borough, points in landmarks.items():
    landmark_list = "\n".join(points.keys())
    ax.annotate(
        f"{borough}:\n{landmark_list}",
        xy=labels[borough],
        xytext=(0, 0),
        textcoords="offset points",
        fontsize=9,
        color="black",
        bbox=dict(facecolor='white', alpha=0.8, edgecolor='black')
    )

plt.title("NYC Airbnb Property Distribution with Landmarks by Borough", fontsize=16)
plt.xlim([-74.3, -73.45])
plt.ylim([40.4, 41.0])
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```


NYC Airbnb Property Distribution with Landmarks by Borough
Latitude
41.0
40.9
40.8
40.7
40.6
40.5
Bronx
Brooklyn
Manhattan
Queens
Staten Island
Bronx:
New York Botanical Garden
Bronx Zoo
Yankee Stadium
Manhattan:
Times Square
Central Park
Empire State Building
Wall Street
The Met
Brooklyn:
Brooklyn Bridge
Brooklyn Museum
Prospect Park
Queens:
Flushing Meadows Corona Park
Museum of the Moving Image
Queens Botanical Garden
Staten Island:
Staten Island Ferry Terminal
Staten Island Zoo
Historic Richmond Town
National Lighthouse Museum
Staten Island Greenbelt Conservancy
LaGuardia Airport
Newark Airport
JFK Airport



Distance to the Center of Manhattan

In [61]:
```
nyc_df = pd.read_csv('nyc_with_distances.csv')
print(nyc_df.columns)
```

```
Index(['id', 'log_price', 'property_type', 'room_type', 'amenities',
       'accommodates', 'bathrooms', 'bed_type', 'cancellation_policy',
       'cleaning_fee', 'city', 'description', 'first_review',
       'host_has_profile_pic', 'host_identity_verified', 'host_response_rate',
       'host_since', 'instant_bookable', 'last_review', 'latitude',
       'longitude', 'name', 'neighbourhood', 'number_of_reviews',
       'review_scores_rating', 'thumbnail_url', 'zipcode', 'bedrooms', 'beds',
       'index_right', 'BoroCode', 'BoroName', 'Shape_Leng', 'Shape_Area',
       'borough', 'distance_to_jfk', 'distance_to_lga', 'distance_to_ewr',
       'bronx_new_york_botanical_garden_distance', 'bronx_bronx_zoo_distance',
       'bronx_yankee_stadium_distance', 'brooklyn_brooklyn_bridge_distance',
       'brooklyn_brooklyn_museum_distance', 'brooklyn_prospect_park_distance',
       'manhattan_times_square_distance', 'manhattan_central_park_distance',
       'manhattan_empire_state_building_distance',
       'manhattan_wall_street_distance', 'manhattan_the_met_distance',
       'queens_flushing_meadows_corona_park_distance',
       'queens_museum_of_the_moving_image_distance',
       'queens_queens_botanical_garden_distance',
       'staten_island_staten_island_ferry_terminal_distance',
       'staten_island_staten_island_zoo_distance',
       'staten_island_historic_richmond_town_distance',
       'staten_island_national_lighthouse_museum_distance',
       'staten_island_staten_island_greenbelt_conservancy_distance'],
      dtype='object')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_8048\2791484958.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on impor
t or set low_memory=False.
  nyc_df = pd.read_csv('nyc_with_distances.csv')
```

In [62]:
```
manhattan_center = (40.7831, -73.9712)  # Central Park

nyc_df['distance_to_manhattan'] = nyc_df.apply(
    lambda row: haversine(row['latitude'], row['longitude'], manhattan_center[0], manhattan_center[1]),
    axis=1
)

print(nyc_df[['borough', 'distance_to_manhattan']].head())
```

	borough	distance_to_manhattan
0	Manhattan	2.413194
1	Manhattan	3.615394
2	Manhattan	6.670668
3	Brooklyn	15.587109
4	Brooklyn	14.683675

In [63]:
```
output_path = 'nyc_with_distances.csv'

nyc_df.to_csv(output_path, index=False)

print(f"Data has been successfully saved to {output_path}")
```

```
Data has been successfully saved to nyc_with_distances.csv
```

Appendix A.2

2. Preprocessing_Basic Traits

In [2]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
df_nyc = pd.read_csv('nyc_with_distances.csv')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_22228\3094127854.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on impo
rt or set low_memory=False.
  df_nyc = pd.read_csv('nyc_with_distances.csv')
```

In [4]:
```
print(df_nyc.isnull().sum())
```

```
id                                                0
log_price                                         0
property_type                                     0
room_type                                         0
amenities                                         0
accommodates                                      0
bathrooms                                         0
bed_type                                          0
cancellation_policy                               0
cleaning_fee                                      0
city                                              0
description                                       0
first_review                                      0
host_has_profile_pic                              0
host_identity_verified                            0
host_response_rate                                0
host_since                                        0
instant_bookable                                  0
last_review                                       0
latitude                                          0
longitude                                         0
name                                              0
neighbourhood                                     0
number_of_reviews                                 0
review_scores_rating                              0
thumbnail_url                                     0
zipcode                                           0
bedrooms                                          0
beds                                              0
index_right                                       0
BoroCode                                          0
BoroName                                          0
Shape_Leng                                        0
Shape_Area                                        0
borough                                           0
distance_to_jfk                                   0
distance_to_lga                                   0
distance_to_ewr                                   0
bronx_new_york_botanical_garden_distance      17478
bronx_bronx_zoo_distance                      17478
bronx_yankee_stadium_distance                 17478
brooklyn_brooklyn_bridge_distance             10486
brooklyn_brooklyn_museum_distance             10486
brooklyn_prospect_park_distance               10486
manhattan_times_square_distance                9823
manhattan_central_park_distance                9823
```

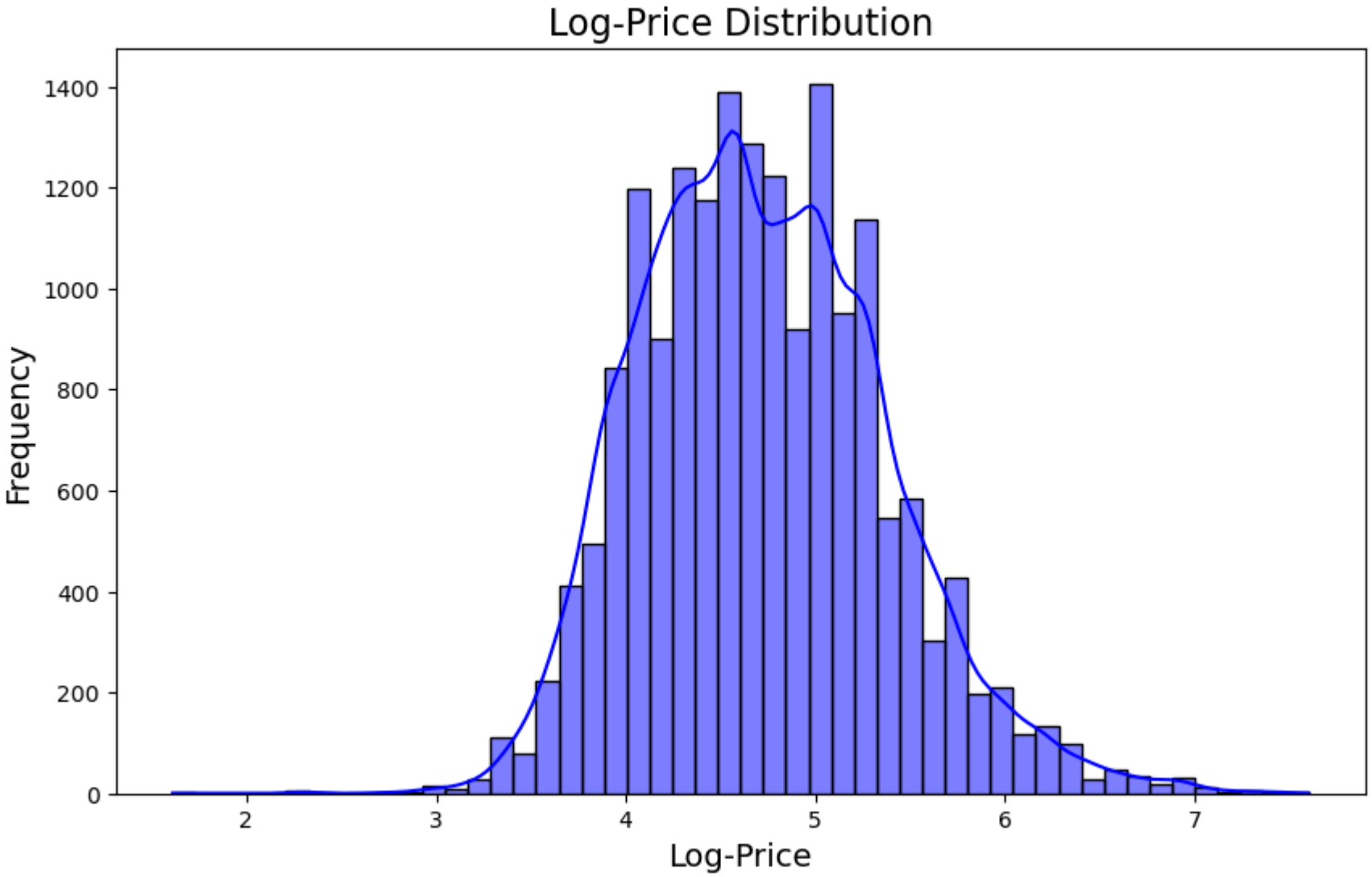
```
manhattan_empire_state_building_distance                    9823
manhattan_wall_street_distance                              9823
manhattan_the_met_distance                                  9823
queens_flushing_meadows_corona_park_distance               15907
queens_museum_of_the_moving_image_distance                 15907
queens_queens_botanical_garden_distance                    15907
staten_island_staten_island_ferry_terminal_distance        17718
staten_island_staten_island_zoo_distance                   17718
staten_island_historic_richmond_town_distance              17718
staten_island_national_lighthouse_museum_distance          17718
staten_island_staten_island_greenbelt_conservancy_distance 17718
distance_to_manhattan                                          0
price                                                          0
dtype: int64
```

Price

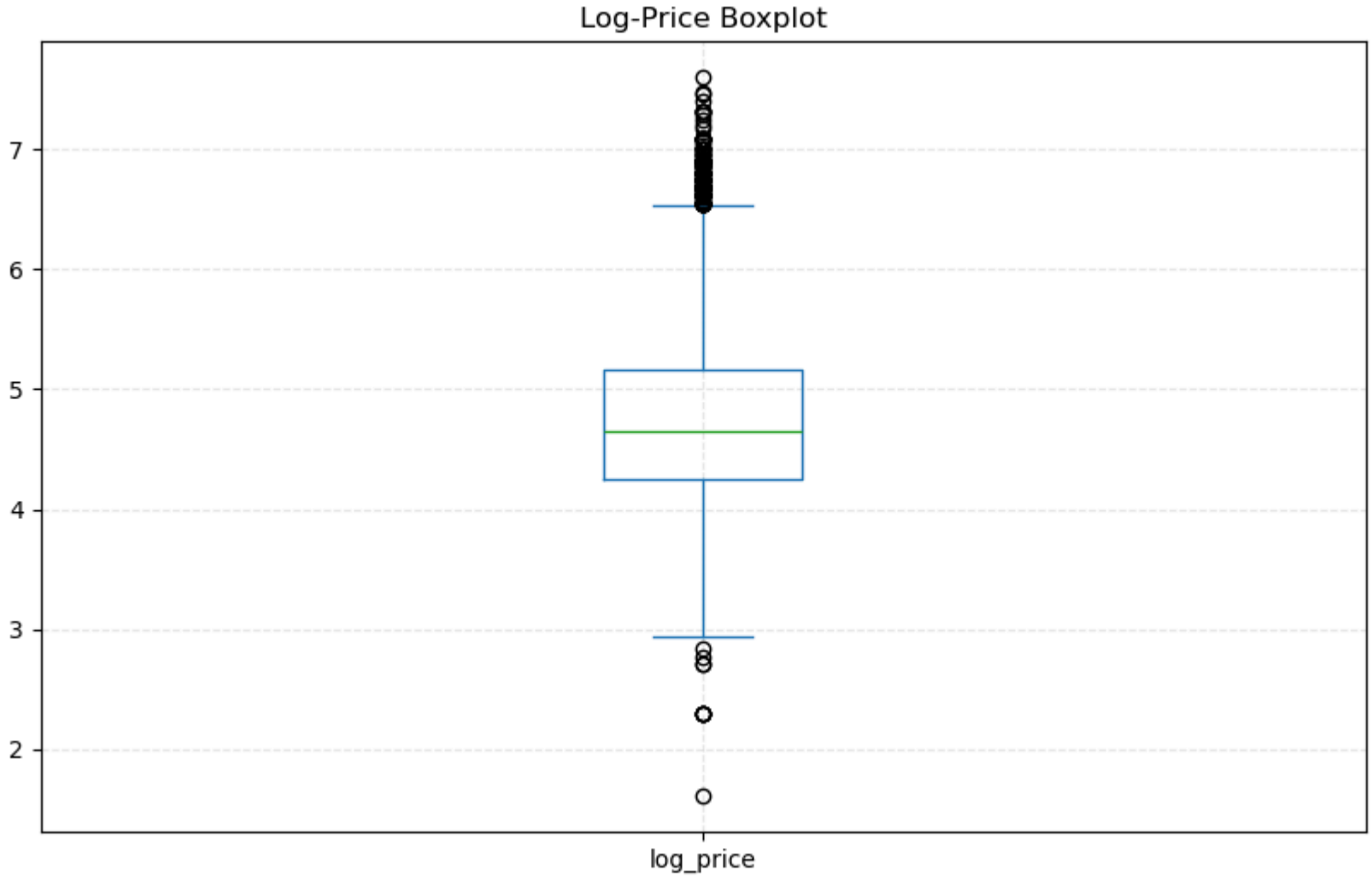
In [6]:
```
log_price_stats = df_nyc['log_price'].describe()
print("Basic description--log_price: ")
print(log_price_stats)
```

```
Basic description--log_price:
count    17853.000000
mean         4.719665
std          0.647487
min          1.609438
25%          4.248495
50%          4.653960
75%          5.164786
max          7.600402
Name: log_price, dtype: float64
```

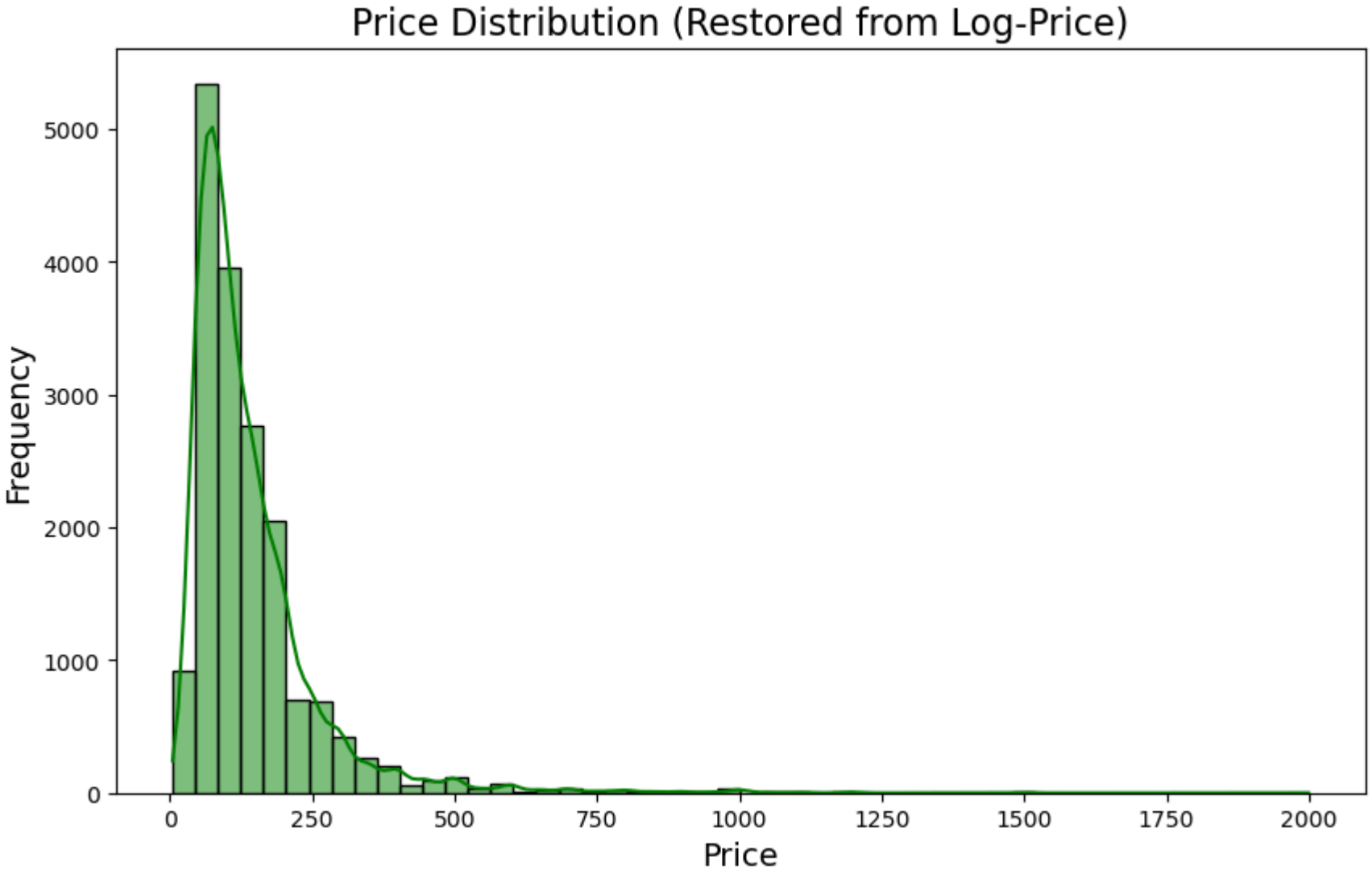
In [7]:
```
plt.figure(figsize=(10, 6))
sns.histplot(df_nyc['log_price'], bins=50, kde=True, color='blue')
plt.title("Log-Price Distribution", fontsize=16)
plt.xlabel("Log-Price", fontsize=14)
plt.ylabel("Frequency", fontsize=14)
plt.show()
```



In [8]:
```
plt.figure(figsize=(10, 6))
df_nyc['log_price'].plot.box(title='Log-Price Boxplot')
plt.grid(linestyle="--", alpha=0.3)
plt.show()
```



In [9]:
```
df_nyc['price'] = np.exp(df_nyc['log_price'])
plt.figure(figsize=(10, 6))
sns.histplot(df_nyc['price'], bins=50, kde=True, color='green')
plt.title("Price Distribution (Restored from Log-Price)", fontsize=16)
plt.xlabel("Price", fontsize=14)
plt.ylabel("Frequency", fontsize=14)
plt.show()
```

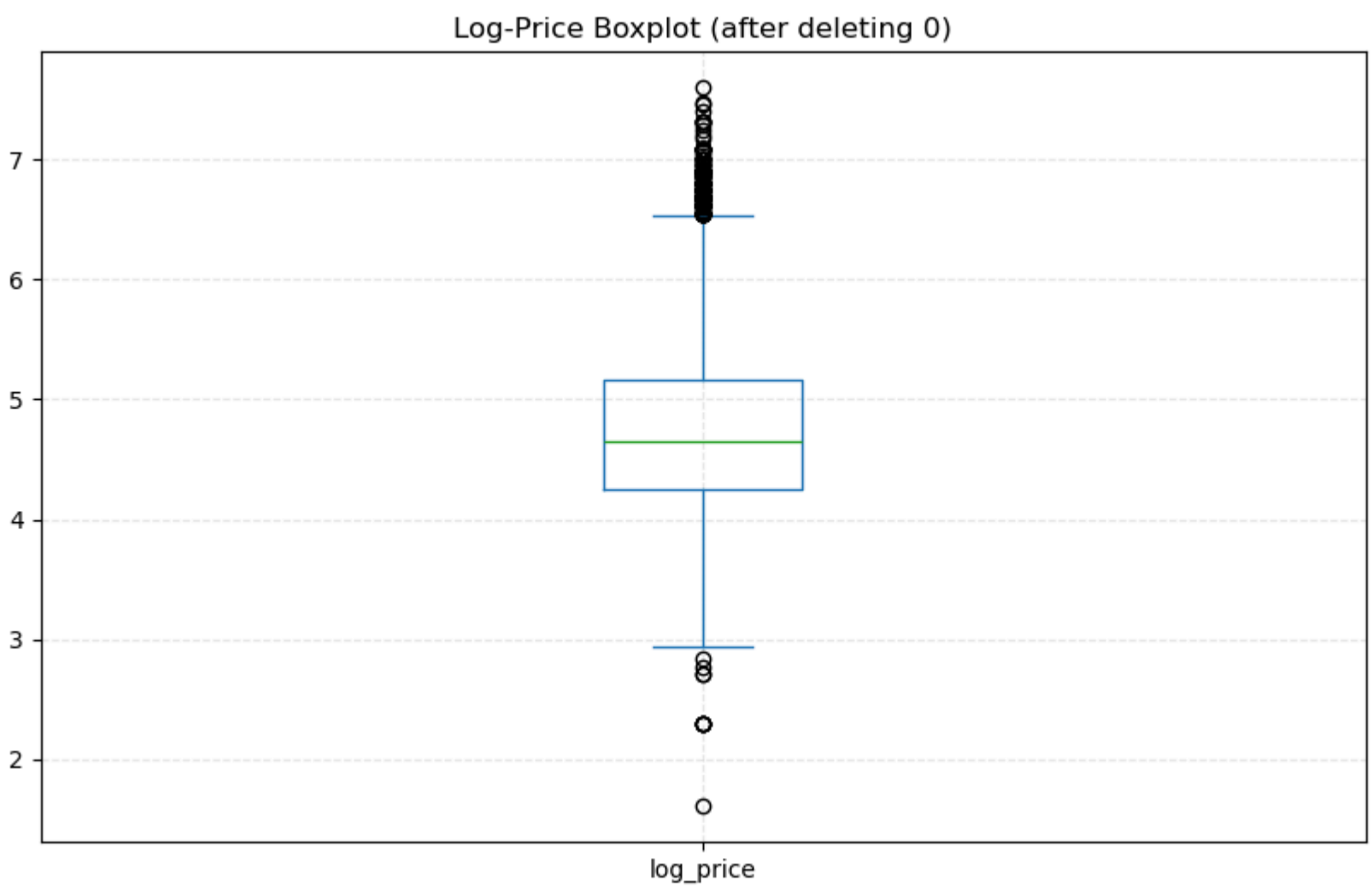


In [10]:
```
price_stats = df_nyc['price'].describe()
print("Basic statistics of the restored price:")
print(price_stats)
```

```
Basic statistics of the restored price:
count    17853.000000
mean       141.163558
std        121.157093
min          5.000000
25%         70.000000
50%        105.000000
75%        175.000000
max       1999.000001
Name: price, dtype: float64
```

In [11]:

```python
df_nyc = df_nyc[(df_nyc['log_price'] != 0)]   #delete 0
plt.figure(figsize=(10, 6))
df_nyc['log_price'].plot.box(title='Log-Price Boxplot (after deleting 0)')
plt.grid(linestyle="--", alpha=0.3)
plt.show()
```



In [12]:
```
df_nyc.to_csv('nyc_with_distances.csv', index=False)

print(f"The cleaned data has been saved and overwritten to the original file: {'nyc_with_distances.csv'}")
```

```
The cleaned data has been saved and overwritten to the original file: nyc_with_distances.csv
```

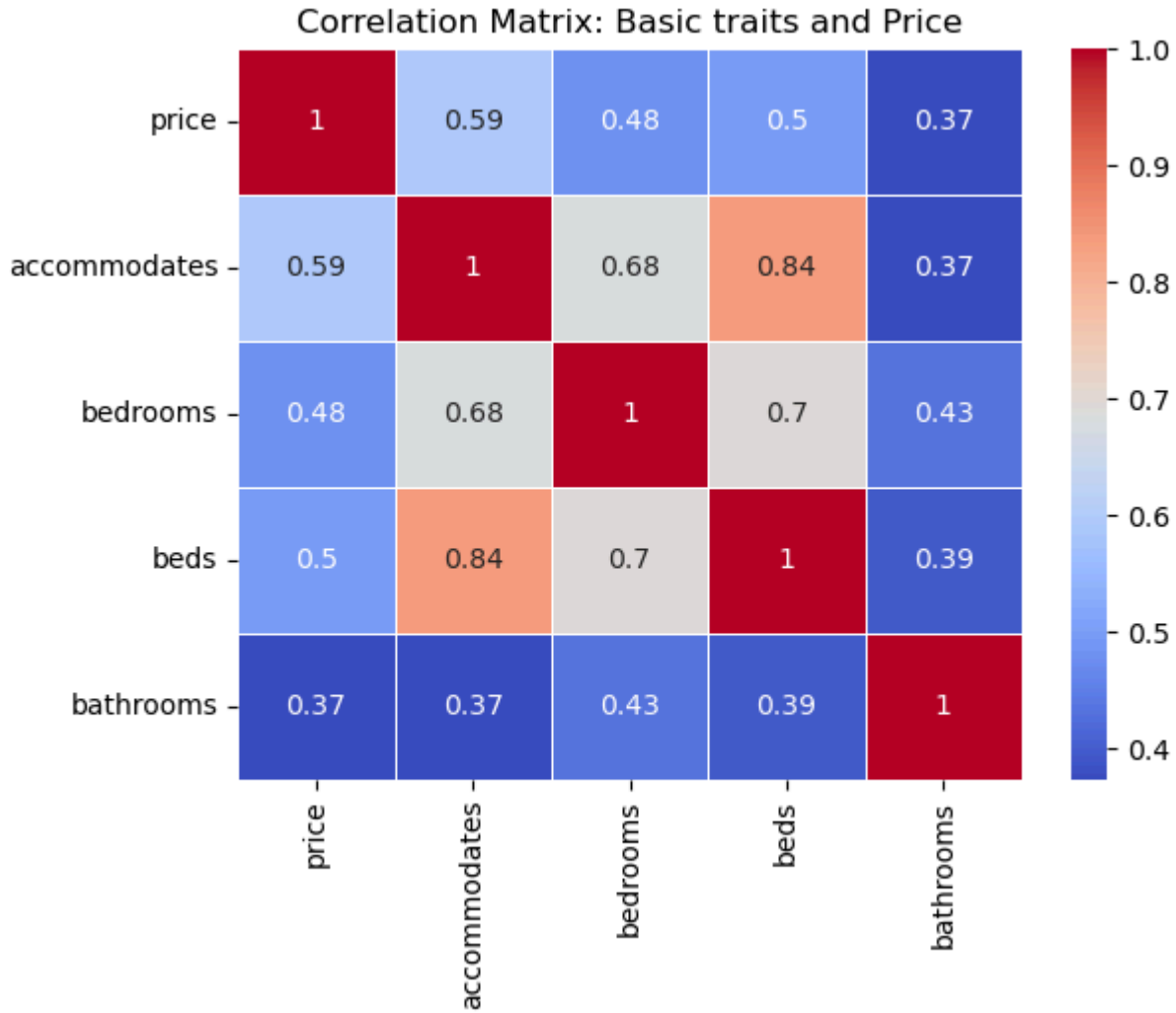
In [13]:
```
df_nyc = pd.read_csv('nyc_with_distances.csv')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_992\3094127854.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on import or set low_memory=False.
  df_nyc = pd.read_csv('nyc_with_distances.csv')
```

In [14]:
```
correlation_matrix = df_nyc[['price', 'accommodates', 'bedrooms','beds','bathrooms']].corr()
print(correlation_matrix)
```

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix: Basic traits and Price')
plt.show()
```

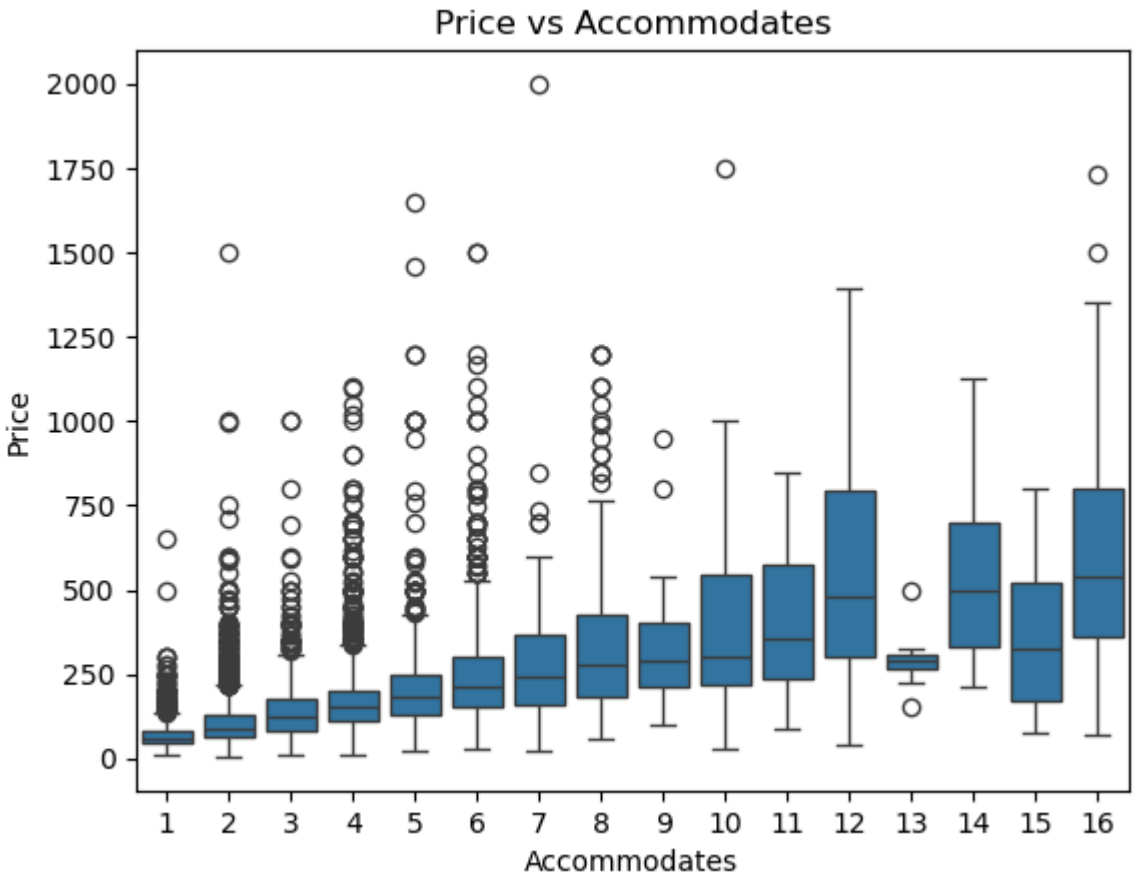
	price	accommodates	bedrooms	beds	bathrooms
price	1.000000	0.592593	0.480019	0.497628	0.373356
accommodates	0.592593	1.000000	0.681815	0.838367	0.374912
bedrooms	0.480019	0.681815	1.000000	0.696092	0.431561
beds	0.497628	0.838367	0.696092	1.000000	0.393282
bathrooms	0.373356	0.374912	0.431561	0.393282	1.000000



Accomodates

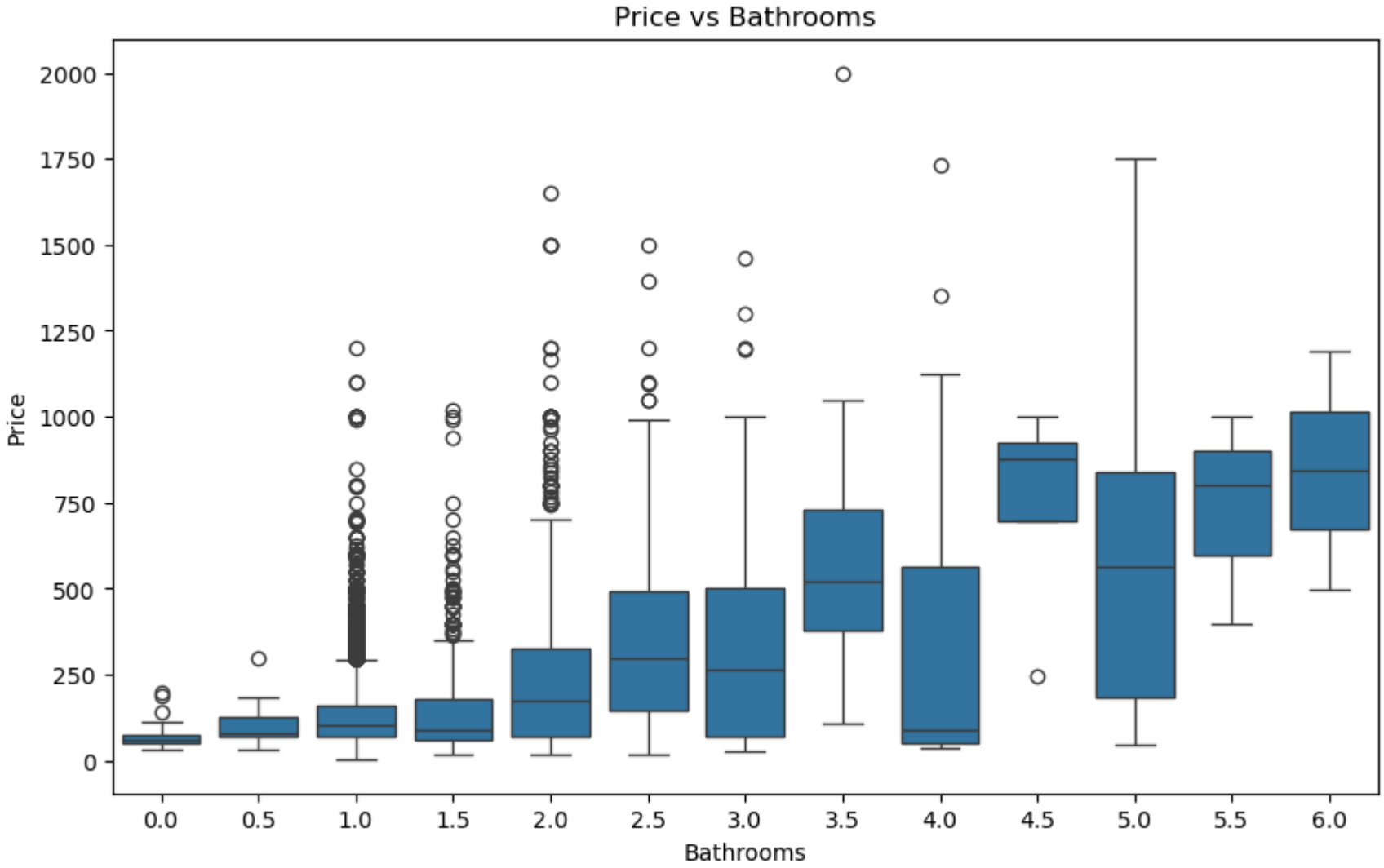
In [16]:
```
sns.boxplot(data=df_nyc, x='accommodates', y='price')
plt.title("Price vs Accommodates")
```

```
plt.xlabel("Accommodates")
plt.ylabel("Price")
plt.show()
```



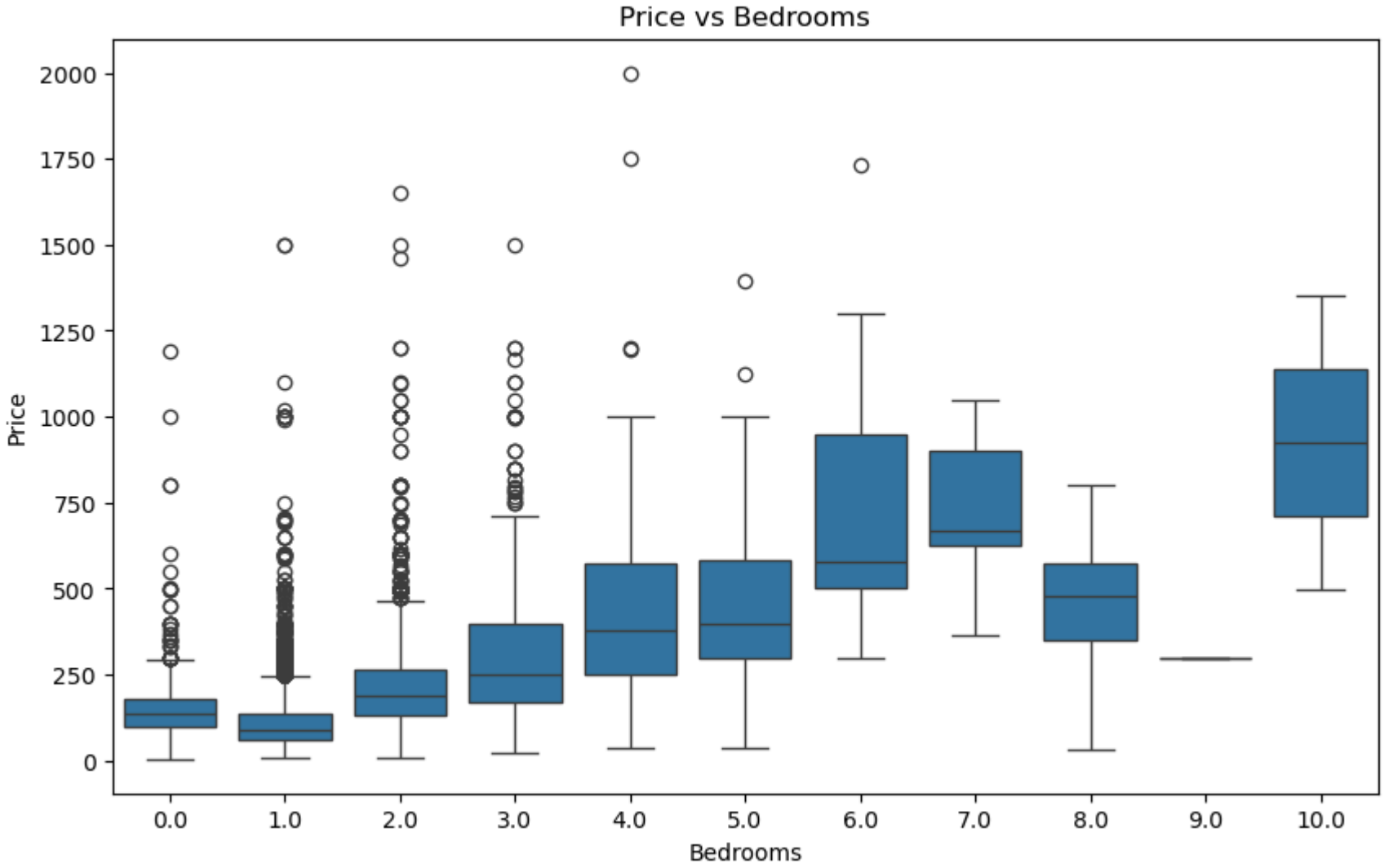
Bathrooms

In [18]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_nyc, x='bathrooms', y='price')
plt.title('Price vs Bathrooms')
plt.xlabel('Bathrooms')
plt.ylabel('Price')
plt.show()
```



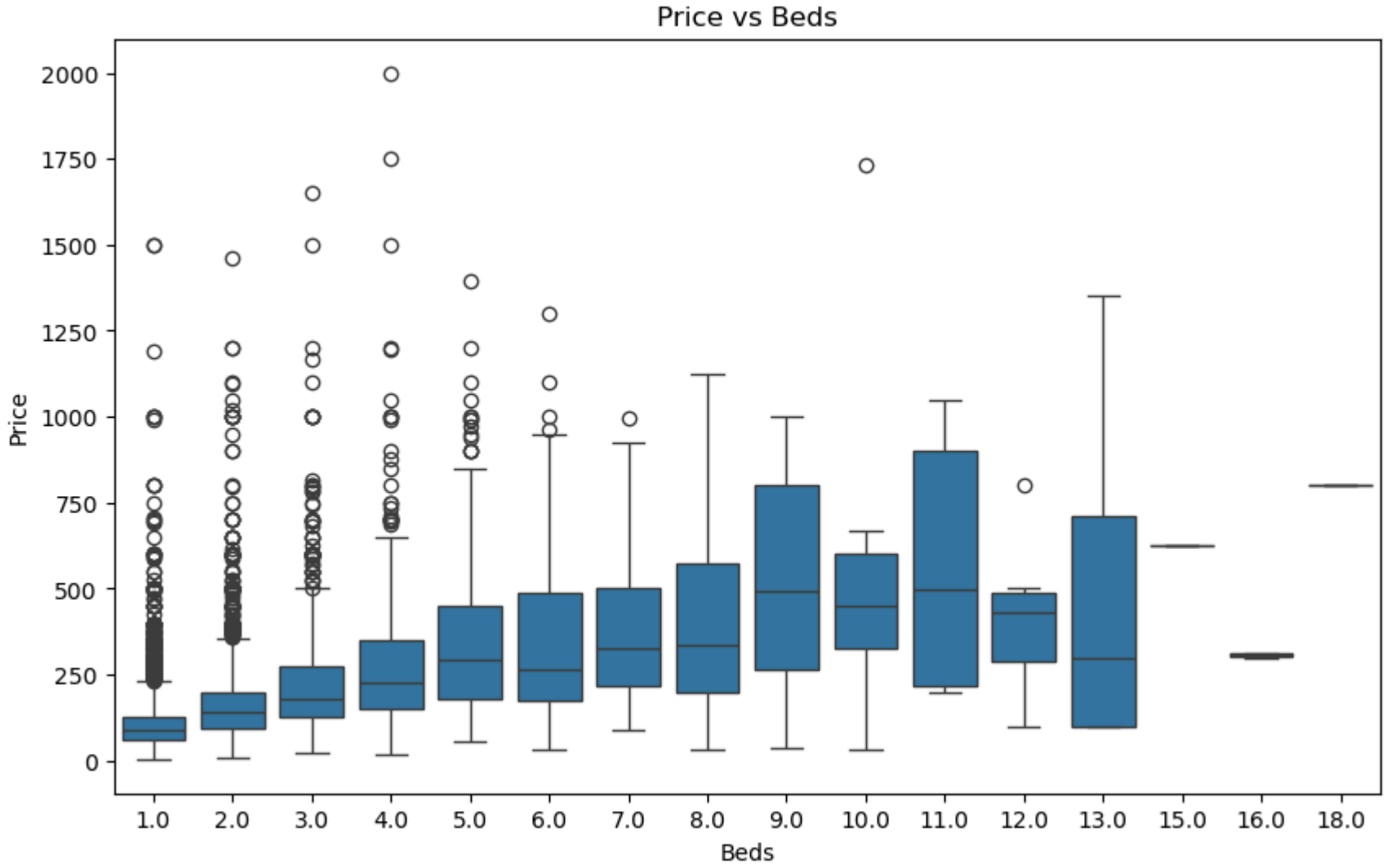
bedrooms

In [20]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_nyc, x='bedrooms', y='price')
plt.title('Price vs Bedrooms')
plt.xlabel('Bedrooms')
plt.ylabel('Price')
plt.show()
```



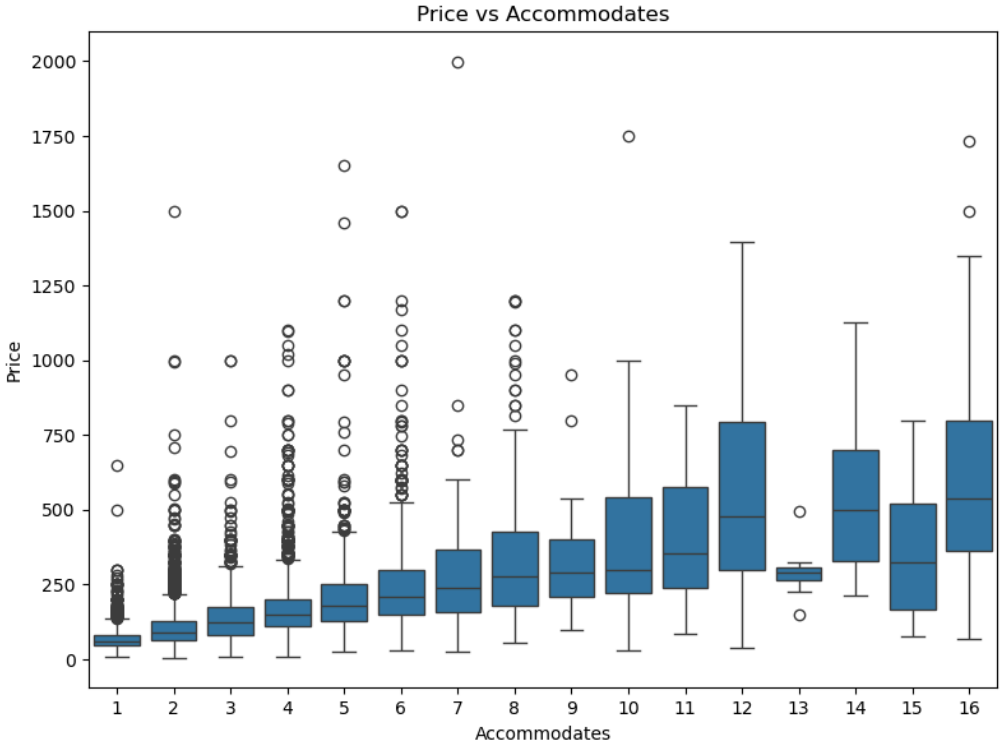
beds

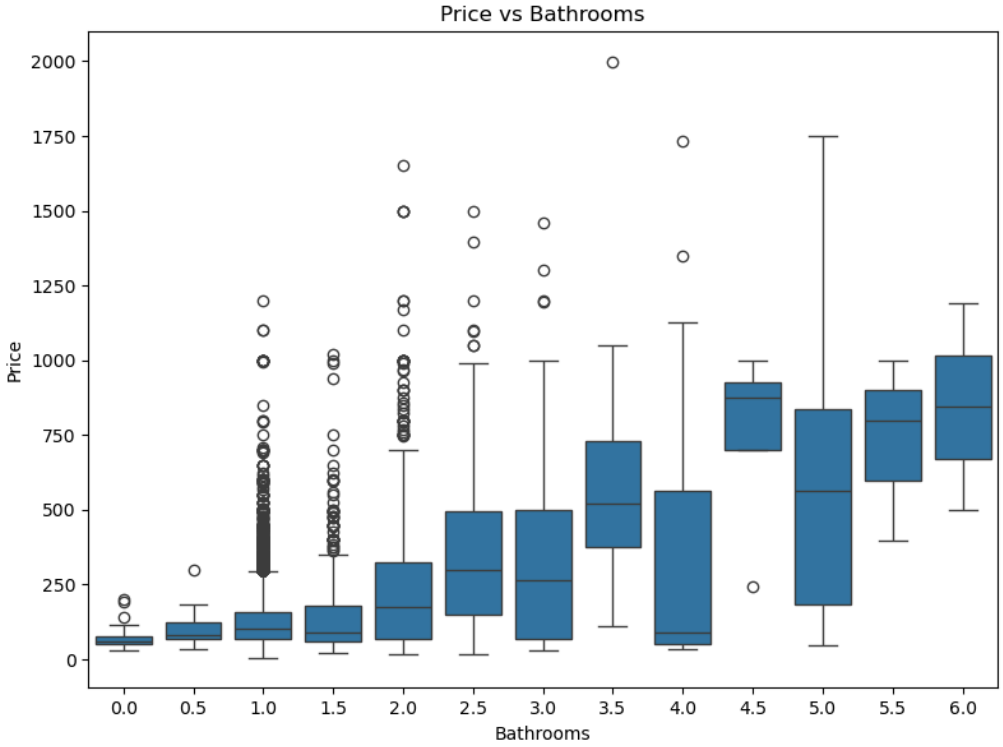
In [22]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_nyc, x='beds', y='price')
plt.title('Price vs Beds')
plt.xlabel('Beds')
plt.ylabel('Price')
plt.show()
```

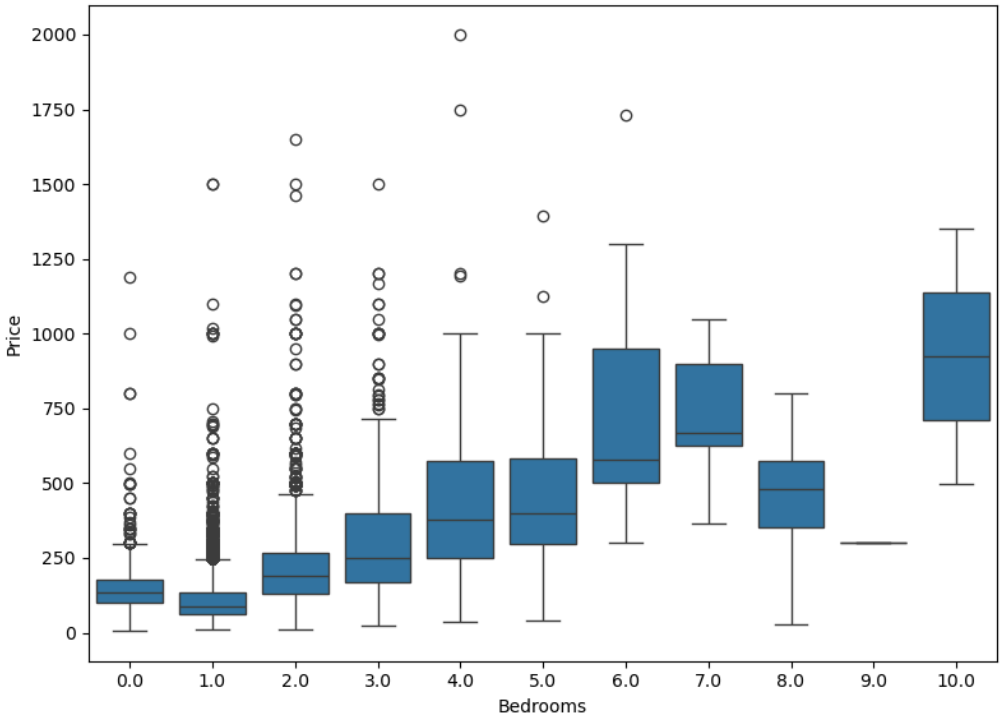


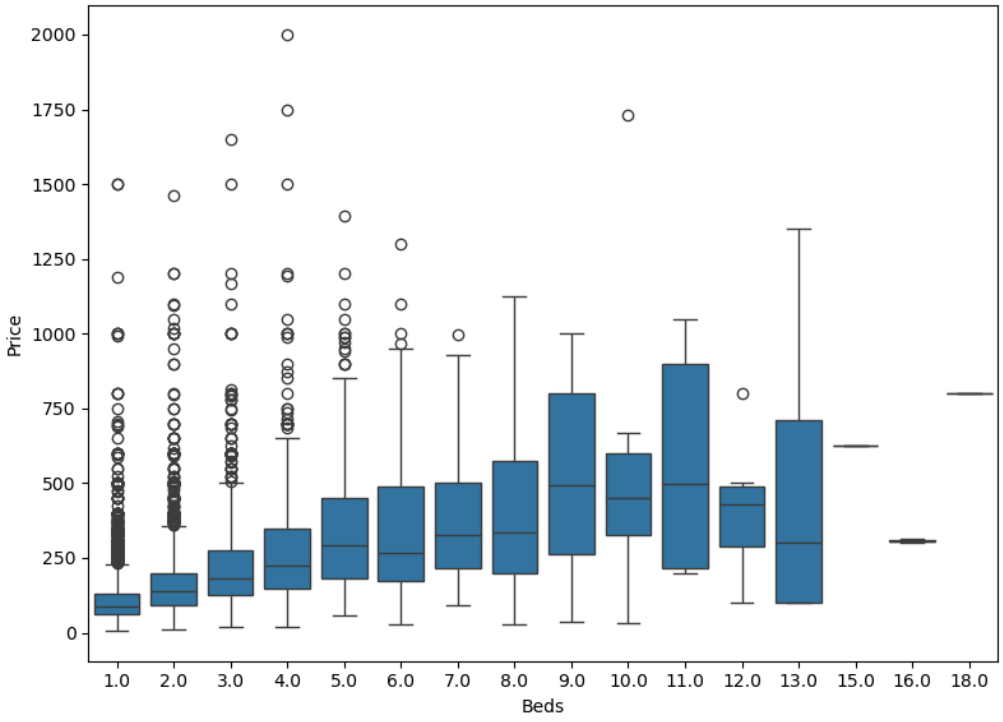
In [23]:
```
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# 1: Price vs Accommodates
sns.boxplot(data=df_nyc, x='accommodates', y='price', ax=axes[0, 0])
axes[0, 0].set_title("Price vs Accommodates")
axes[0, 0].set_xlabel("Accommodates")
axes[0, 0].set_ylabel("Price")

# 2: Price vs Bathrooms
sns.boxplot(data=df_nyc, x='bathrooms', y='price', ax=axes[0, 1])
axes[0, 1].set_title("Price vs Bathrooms")
axes[0, 1].set_xlabel("Bathrooms")
axes[0, 1].set_ylabel("Price")
```

```
# 3: Price vs Bedrooms
sns.boxplot(data=df_nyc, x='bedrooms', y='price', ax=axes[1, 0])
axes[1, 0].set_title("Price vs Bedrooms")
axes[1, 0].set_xlabel("Bedrooms")
axes[1, 0].set_ylabel("Price")

# 4: Price vs Beds
sns.boxplot(data=df_nyc, x='beds', y='price', ax=axes[1, 1])
axes[1, 1].set_title("Price vs Beds")
axes[1, 1].set_xlabel("Beds")
axes[1, 1].set_ylabel("Price")

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


Price vs Accommodates
Price
Accommodates

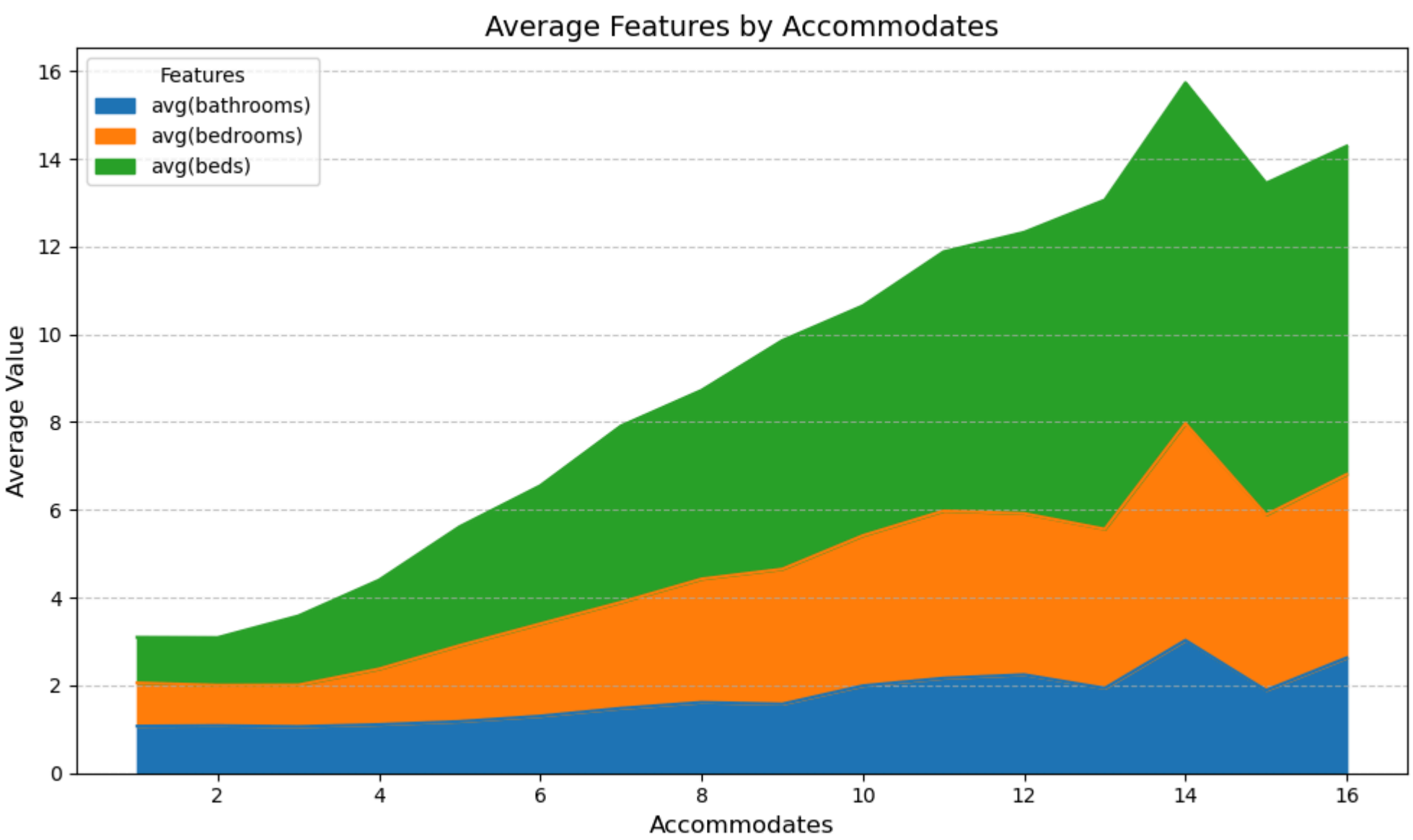

Price vs Bathrooms
Price
Bathrooms


Price vs Bedrooms
Price
Bedrooms


Price vs Beds
Price
Beds

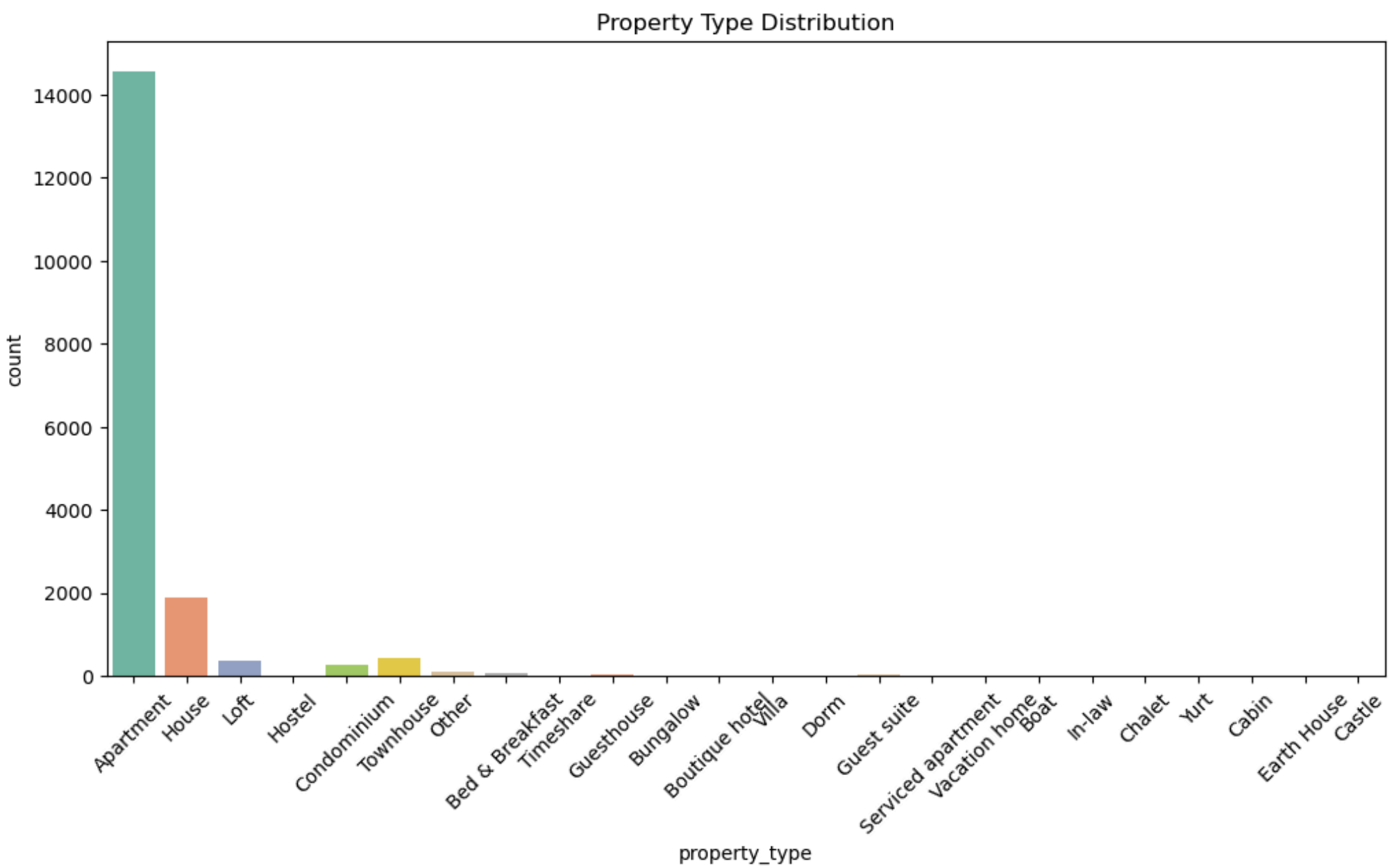
Accomodates & bedrooms/bathrooms/beds

In [25]:
```
df_by_acc = df_nyc.groupby('accommodates').agg({
    'bathrooms': 'mean',
    'bedrooms': 'mean',
    'beds': 'mean'
}).reset_index()
df_by_acc.columns = ['accommodates', 'avg(bathrooms)', 'avg(bedrooms)', 'avg(beds)']

df_by_acc.plot(
    kind='area',
    x='accommodates',
    y=df_by_acc.columns[1:],
    stacked=True,
    figsize=(10, 6)
)

plt.title('Average Features by Accommodates', fontsize=14)
plt.xlabel('Accommodates', fontsize=12)
plt.ylabel('Average Value', fontsize=12)
plt.legend(title='Features', loc='upper left')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Property type

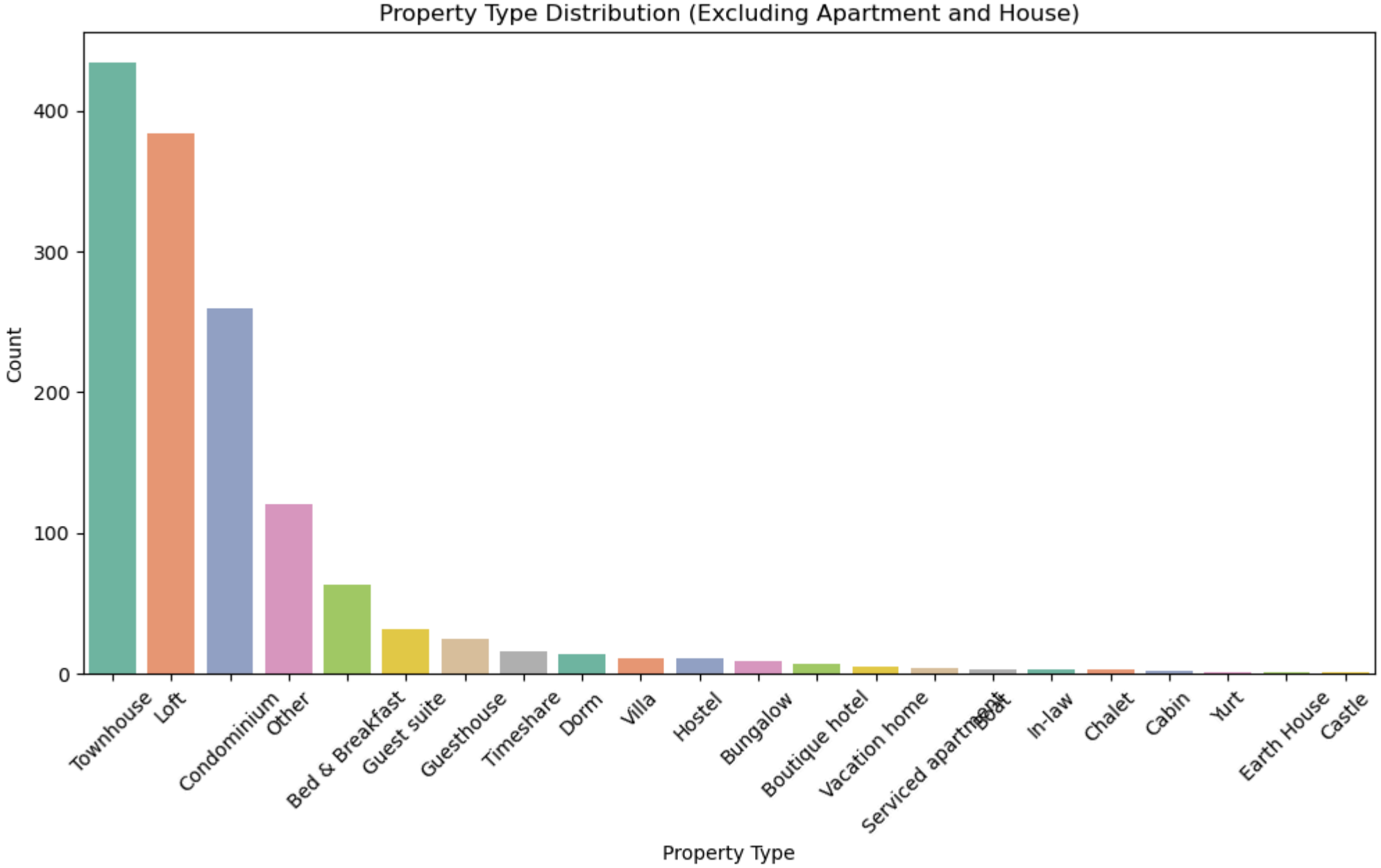
In [27]:
```
plt.figure(figsize=(12, 6))
sns.countplot(x='property_type', data=df_nyc, hue='property_type', palette='Set2', legend=False)
plt.title('Property Type Distribution')
plt.xticks(rotation=45)
plt.show()
```



In [28]:
```
df_other = df_nyc[(df_nyc['property_type'] != 'Apartment') & (df_nyc['property_type'] != 'House')]

property_counts = df_other['property_type'].value_counts().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(x=property_counts.index, y=property_counts.values, palette='Set2')

plt.title('Property Type Distribution (Excluding Apartment and House)')
```

```
plt.xticks(rotation=45)
plt.ylabel('Count')
plt.xlabel('Property Type')
plt.show()
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_992\4243015488.py:6: FutureWarning:

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

  sns.barplot(x=property_counts.index, y=property_counts.values, palette='Set2')
```



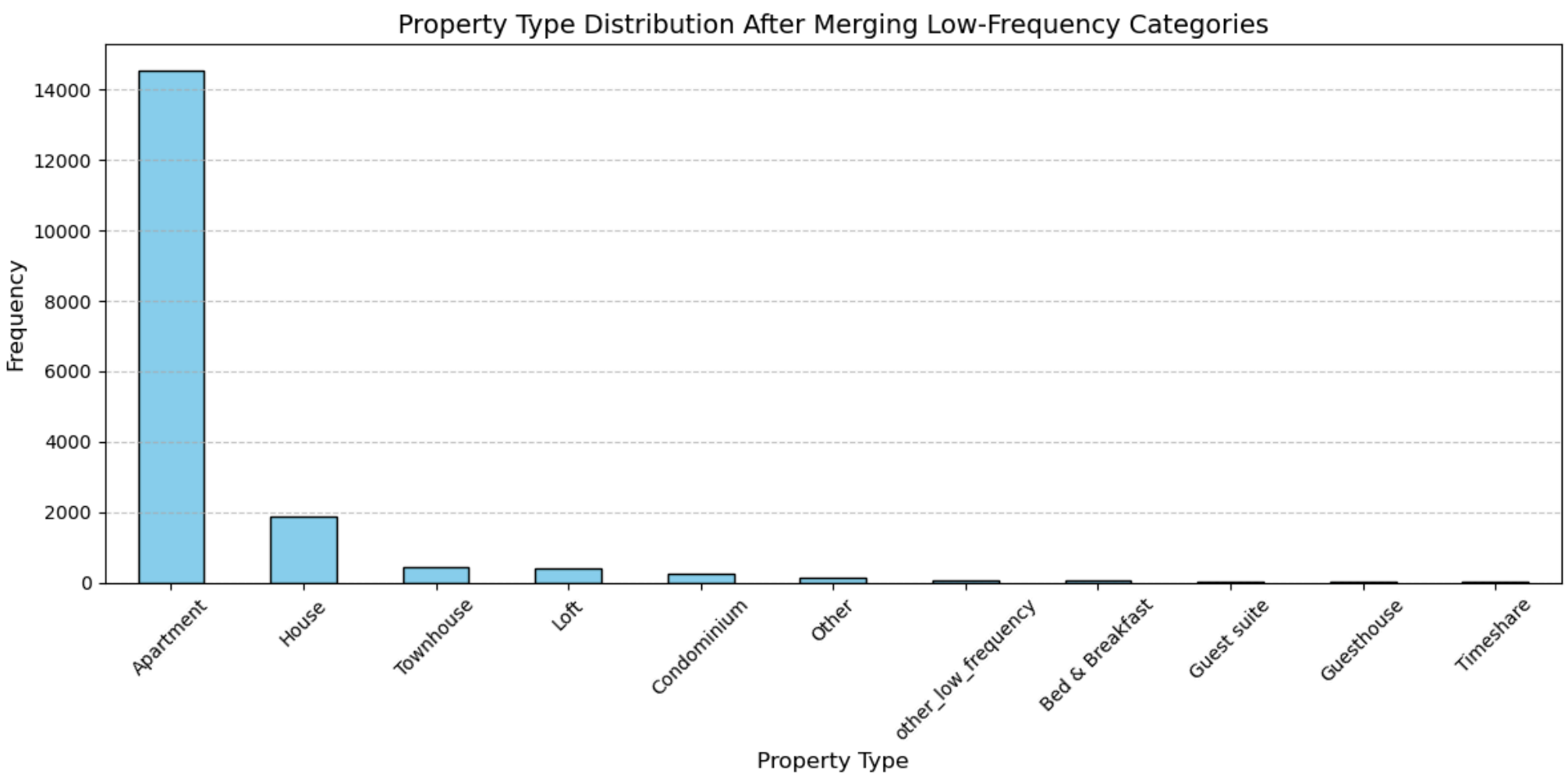
In [29]:
```
property_type_counts = df_nyc['property_type'].value_counts()

top_5_property_types = property_type_counts.head(5)

print(top_5_property_types)
```

```
property_type
Apartment      14559
House           1885
Townhouse        434
Loft             384
Condominium      260
Name: count, dtype: int64
```

In [30]:

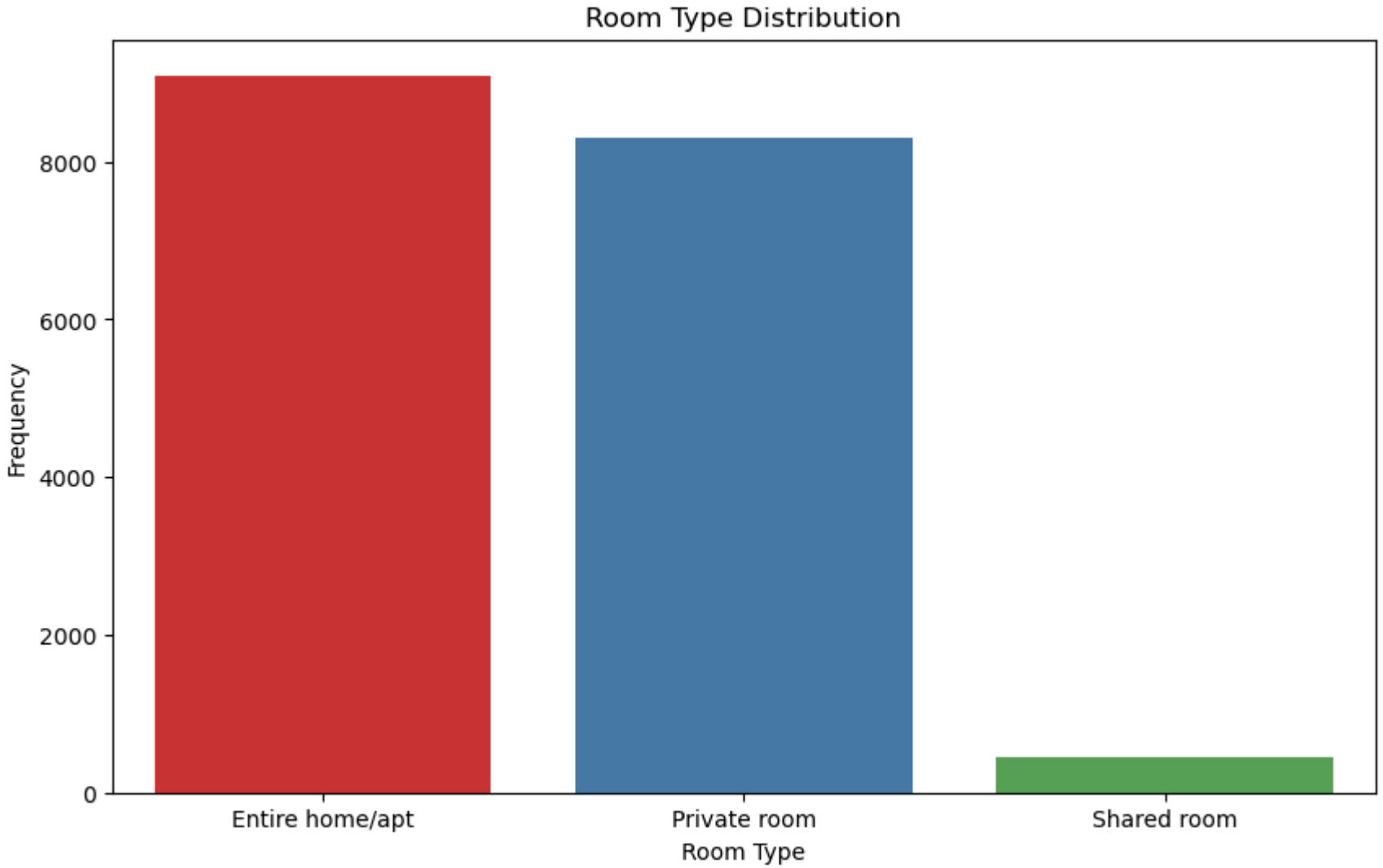
```python
top_10_property_types = property_type_counts.head(10)

low_frequency_types = property_type_counts[~property_type_counts.index.isin(top_10_property_types.index)].index
for property_type in low_frequency_types:
    count = property_type_counts[property_type]
    print(f"{property_type}: {count}")

df_nyc['property_type'] = df_nyc['property_type'].apply(
    lambda x: x if x in top_10_property_types.index else 'other_low_frequency'
)

property_type_counts_after_merge = df_nyc['property_type'].value_counts()

print("\nProperty type distribution after merging low-frequency categories:")
print(property_type_counts_after_merge)

plt.figure(figsize=(12, 6))
property_type_counts_after_merge.sort_values(ascending=False).plot(
    kind='bar', color='skyblue', edgecolor='black'
)
plt.title('Property Type Distribution After Merging Low-Frequency Categories', fontsize=14)
plt.xlabel('Property Type', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
Dorm: 14
Villa: 11
Hostel: 11
Bungalow: 9
Boutique hotel: 7
Vacation home: 5
Serviced apartment: 4
Boat: 3
In-law: 3
Chalet: 3
Cabin: 2
Yurt: 1
Earth House: 1
Castle: 1

Property type distribution after merging low-frequency categories:
property_type
Apartment              14559
House                   1885
Townhouse                434
Loft                     384
Condominium              260
Other                    120
other_low_frequency       75
Bed & Breakfast           63
Guest suite               32
Guesthouse                25
Timeshare                 16
Name: count, dtype: int64
```



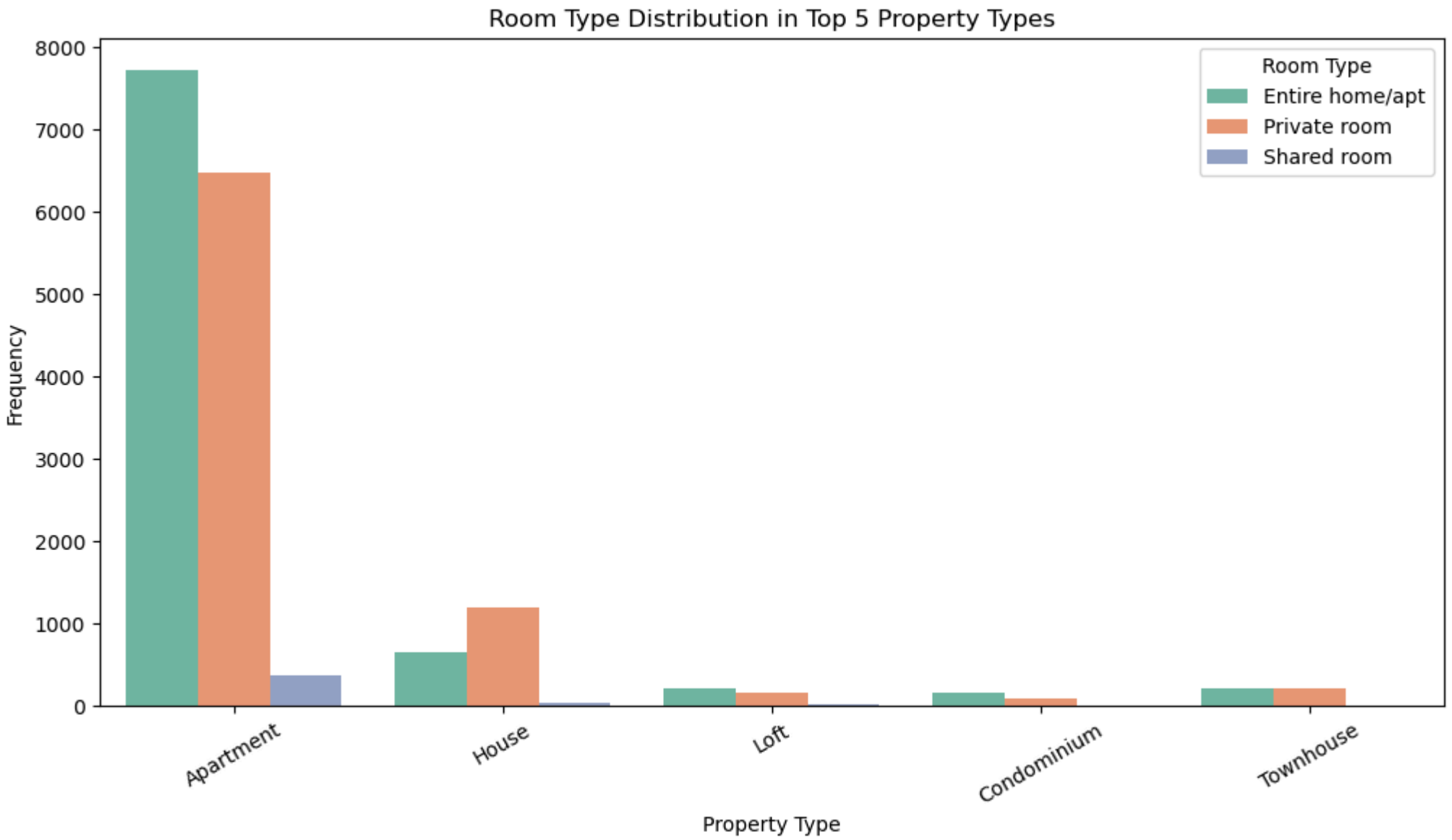
Room-type

In [32]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='room_type', data=df_nyc, hue='room_type', palette='Set1', legend=False)
plt.title('Room Type Distribution')
plt.xlabel('Room Type')
plt.ylabel('Frequency')
plt.show()
```



In [33]:
```
top_5_property_types = ['Apartment', 'House', 'Townhouse', 'Loft', 'Condominium']
df_top5 = df_nyc[df_nyc['property_type'].isin(top_5_property_types)]

plt.figure(figsize=(12, 6))
sns.countplot(
    x='property_type',
    hue='room_type',
    data=df_top5,
    palette='Set2'
)

plt.title('Room Type Distribution in Top 5 Property Types')
plt.xlabel('Property Type')
```

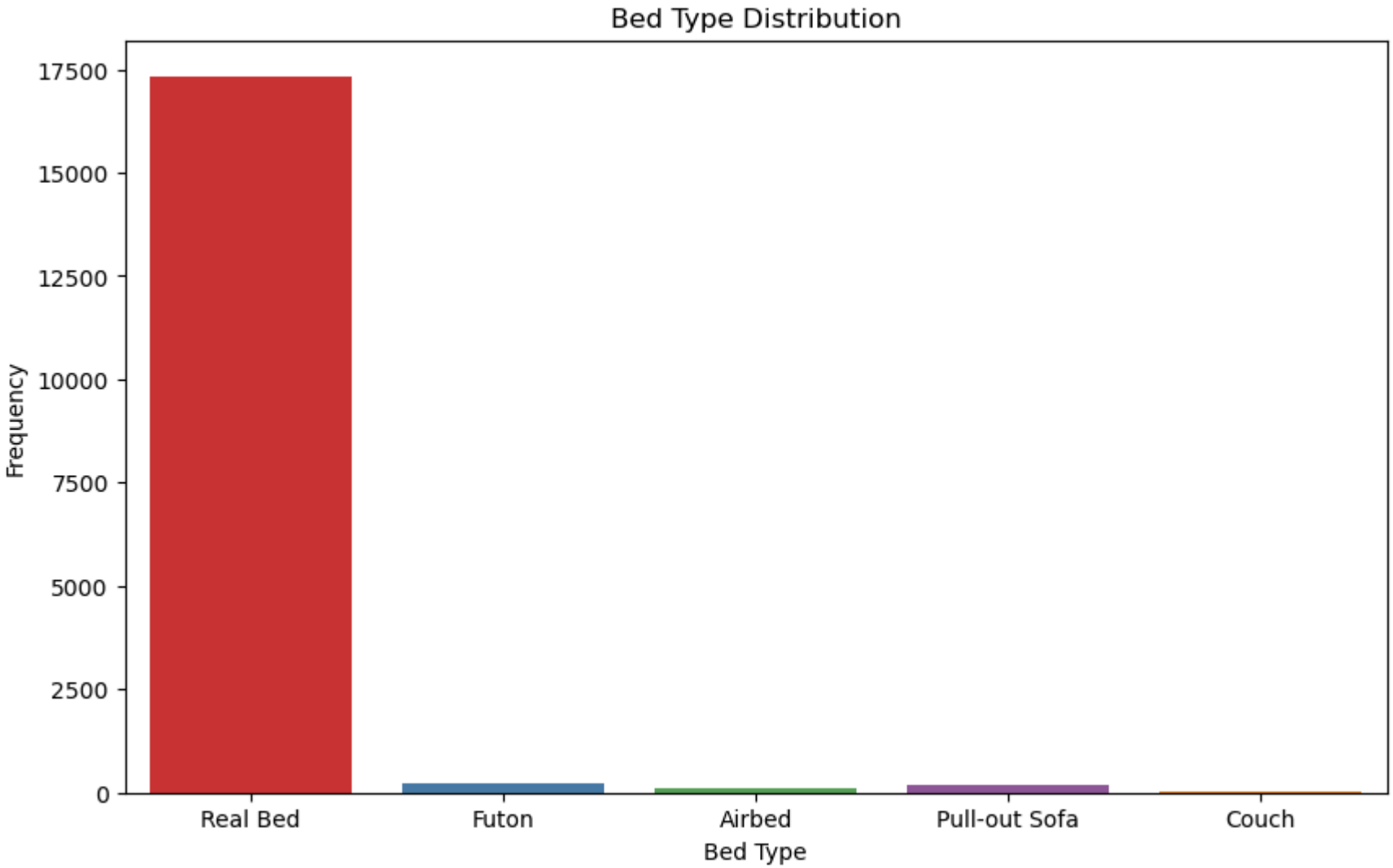
```
plt.ylabel('Frequency')
plt.legend(title='Room Type')
plt.xticks(rotation=30)
plt.show()
```



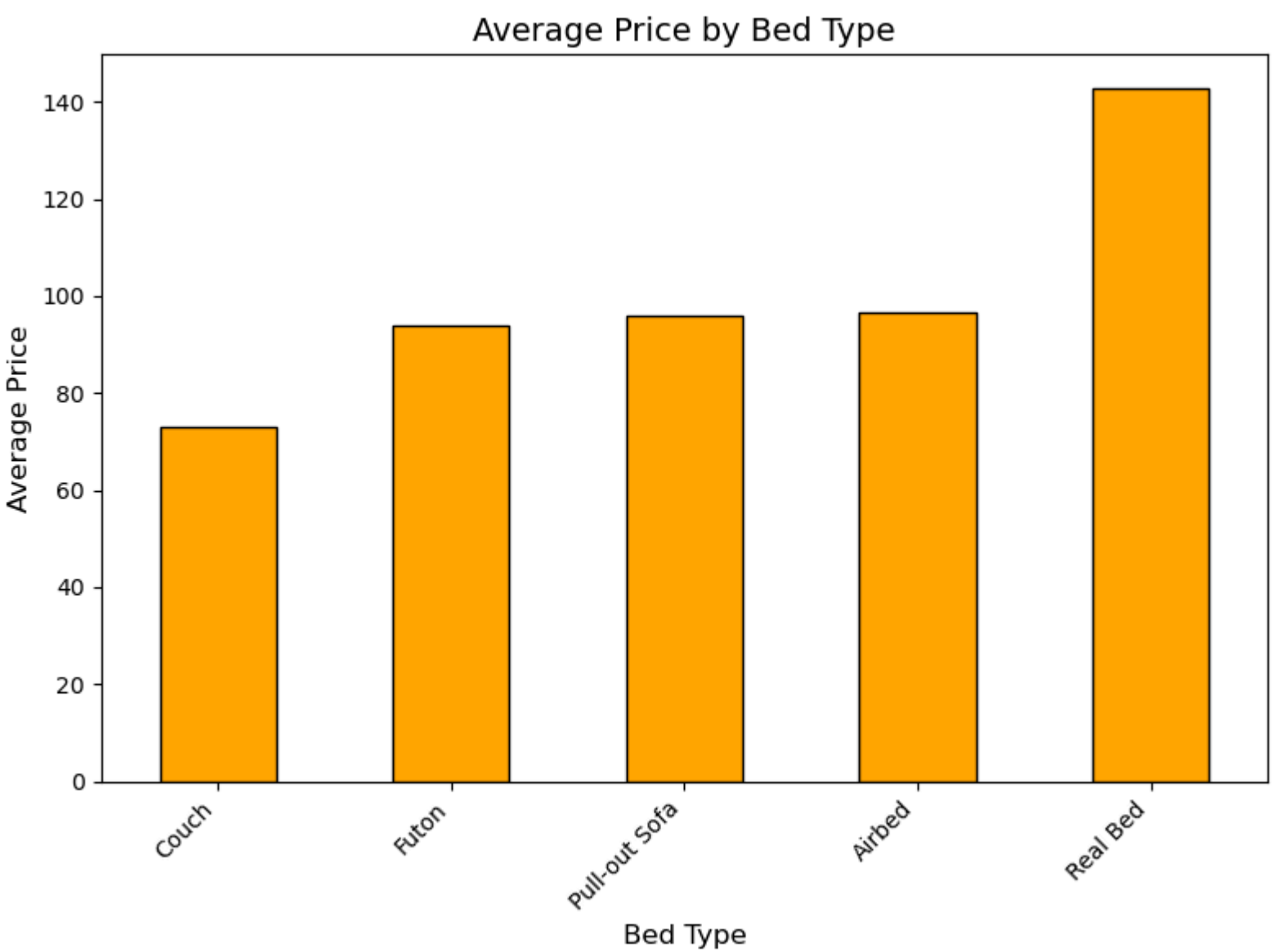
bed type

In [35]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='bed_type', data=df_nyc, hue='bed_type', palette='Set1', legend=False)
plt.title('Bed Type Distribution')
```

```
plt.xlabel('Bed Type')
plt.ylabel('Frequency')
plt.show()
```



In [64]:
```
bed_type_price = df_nyc.groupby('bed_type')['price'].mean().sort_values()

bed_type_price.plot(kind='bar', figsize=(8, 6), color='orange', edgecolor='black')
plt.title("Average Price by Bed Type", fontsize=14)
plt.xlabel("Bed Type", fontsize=12)
plt.ylabel("Average Price", fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```


Average Price by Bed Type
Average Price
0
20
40
60
80
100
120
140
Couch
Futon
Pull-out Sofa
Airbed
Real Bed
Bed Type

Appendix A.3

3. Preprocessing_Facilities and Services

In [2]:
```
import pandas as pd
from itertools import chain
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df_nyc = pd.read_csv('nyc_with_distances.csv')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_13736\3094127854.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on impo
rt or set low_memory=False.
  df_nyc = pd.read_csv('nyc_with_distances.csv')
```

Amenities

In [5]:
```
print(f"Number of missing values in 'amenities': {df_nyc['amenities'].isnull().sum()}")
```

```
Number of missing values in 'amenities': 0
```

1. choose Top50 Amenities（after deleting 'translation missing' items)

In [7]:
```
df_nyc['amenities_list'] = df_nyc['amenities'].apply(lambda x: [item.strip('"') for item in x.strip('{}').split(',')])
df_nyc['amenities_list'] = df_nyc['amenities_list'].apply(lambda x: [item.strip().capitalize() for item in x])
```

In [8]:
```
exploded_df = df_nyc.explode('amenities_list')

amenities_counts = exploded_df['amenities_list'].value_counts()
```

In [9]:
```
print(amenities_counts.head(50))
```

amenities_list	
Wireless internet	17510
Heating	16838
Kitchen	16505
Essentials	15891
Air conditioning	15757
Smoke detector	14935
Hangers	12910
Carbon monoxide detector	11994
Tv	11830
Shampoo	11690
Hair dryer	11561
Iron	11029
Laptop friendly workspace	10926
Internet	9863
Family/kid friendly	8336
Translation missing: en.hosting_amenity_50	7084
First aid kit	6636
Fire extinguisher	6558
Washer	6548
Dryer	6534
Buzzer/wireless intercom	6524
Translation missing: en.hosting_amenity_49	5606
Cable tv	5198
Lock on bedroom door	4907
Elevator	4455
24-hour check-in	3949
Self check-in	3026
Safety card	2596
Refrigerator	2480
Dishes and silverware	2279
Hot water	2277
Bed linens	2225
Stove	2213
Cooking basics	2164
Oven	2131
Pets allowed	2036
Microwave	2024
Free parking on premises	2009
Pets live on this property	1969
Breakfast	1948
Coffee maker	1824
Lockbox	1773
Private entrance	1749
Extra pillows and blankets	1625
Bathtub	1438

```
Doorman                          1430
Step-free access                 1232
Gym                              1069
Wheelchair accessible            1050
Dishwasher                       1023
Name: count, dtype: int64
```

In [10]:
```python
df_nyc['amenities_list'] = df_nyc['amenities_list'].apply(lambda x: [item.strip() for item in x if not item.strip().lower().startswith('tra
df_nyc['amenities_list'] = df_nyc['amenities_list'].apply(lambda x: [item.strip().capitalize() for item in x])

translation_missing_items = [item for sublist in df_nyc['amenities_list'] for item in sublist if 'translation missing' in item.lower()]
print(f"Remaining 'translation_missing' items: {set(translation_missing_items)}")
```

```
Remaining 'translation_missing' items: set()
```

In [11]:
```python
exploded_df = df_nyc.explode('amenities_list')
amenities_counts = exploded_df['amenities_list'].value_counts()
print(amenities_counts.head(50))
```

```
amenities_list
Wireless internet             17510
Heating                       16838
Kitchen                       16505
Essentials                    15891
Air conditioning              15757
Smoke detector                14935
Hangers                       12910
Carbon monoxide detector      11994
Tv                            11830
Shampoo                       11690
Hair dryer                    11561
Iron                          11029
Laptop friendly workspace     10926
Internet                       9863
Family/kid friendly            8336
First aid kit                  6636
Fire extinguisher              6558
Washer                         6548
Dryer                          6534
Buzzer/wireless intercom       6524
Cable tv                       5198
Lock on bedroom door           4907
Elevator                       4455
24-hour check-in               3949
Self check-in                  3026
Safety card                    2596
Refrigerator                   2480
Dishes and silverware          2279
Hot water                      2277
Bed linens                     2225
Stove                          2213
Cooking basics                 2164
Oven                           2131
Pets allowed                   2036
Microwave                      2024
Free parking on premises       2009
Pets live on this property     1969
Breakfast                      1948
Coffee maker                   1824
Lockbox                        1773
Private entrance               1749
Extra pillows and blankets     1625
Bathtub                        1438
Doorman                        1430
Step-free access               1232
```

```
Gym                            1069
Wheelchair accessible          1050
Dishwasher                     1023
Luggage dropoff allowed         911
Private living room             899
Name: count, dtype: int64
```

2. change data into Boolean type and save the data into a new file

In [13]:
```python
top_50_amenities = amenities_counts.head(50).index

for amenity in top_50_amenities:
    column_name = amenity.strip().replace(' ', '_')
    df_nyc[column_name] = df_nyc['amenities_list'].apply(lambda x: amenity in x)

df_nyc.drop(columns=['amenities_list'], inplace=True)

df_nyc.to_csv('nyc_with_top_50_amenities.csv', index=False)

print("Dataset with top 50 amenities saved as 'nyc_with_top_50_amenities.csv'")
```

```
Dataset with top 50 amenities saved as 'nyc_with_top_50_amenities.csv'
```

3. Visualize （use TOP5 as example)

In [15]:
```python
top_5_amenities = amenities_counts.head(5).index
top_5_columns = [amenity.strip().replace(' ', '_') for amenity in top_5_amenities]

top_5_with_ids = df_nyc[['id'] + top_5_columns].head(10).set_index('id')
top_5_str_df = top_5_with_ids.replace({True: 'True', False: 'False'})

fig, ax = plt.subplots(figsize=(12, 6))
ax.axis('tight')
ax.axis('off')

table = ax.table(
    cellText=top_5_str_df.reset_index().values,
    colLabels=['ID'] + list(top_5_str_df.columns),
    cellLoc='center',
    loc='center',
)

table.auto_set_font_size(False)
table.set_fontsize(10)
```

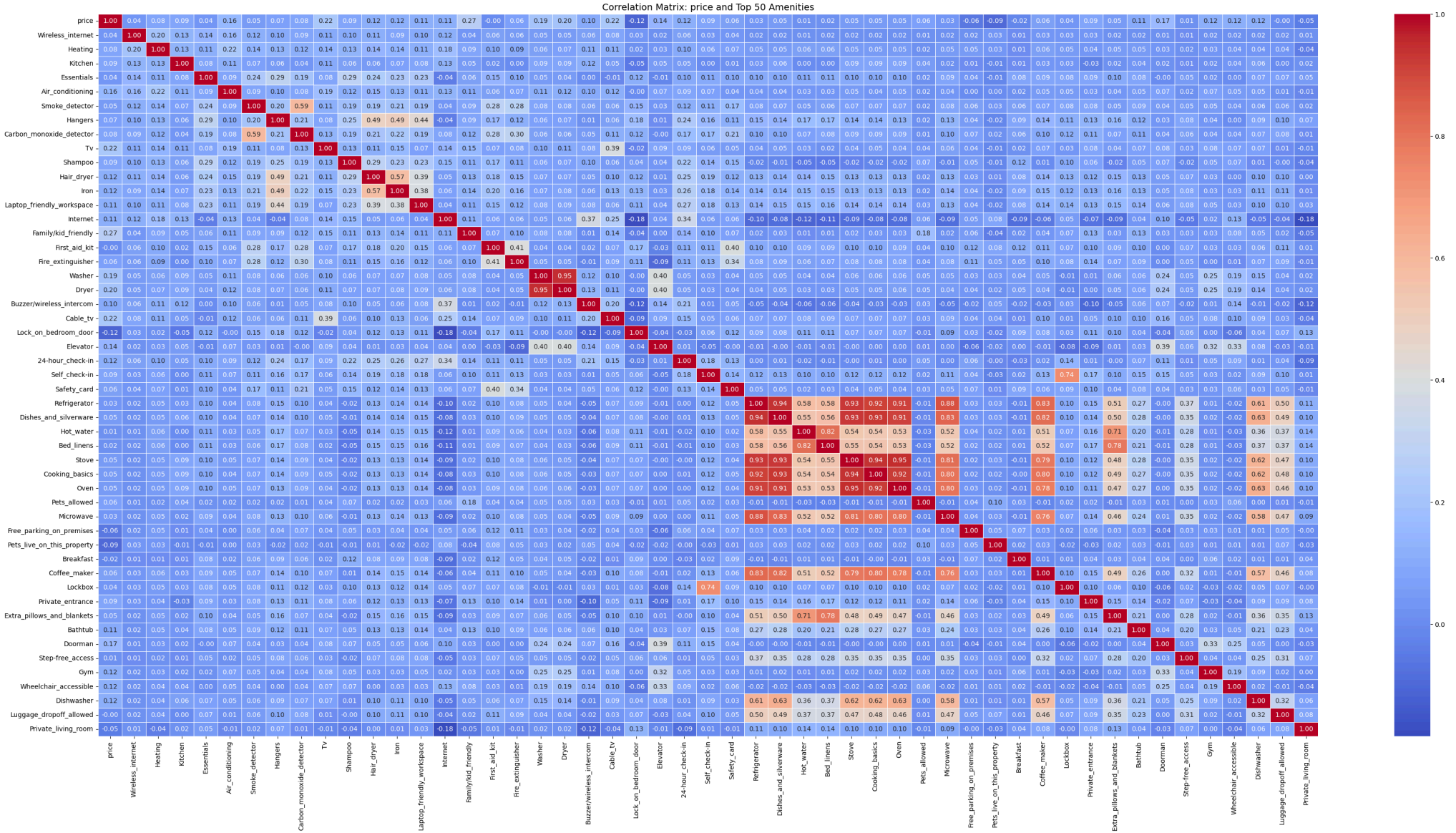
```
table.auto_set_column_width(col=list(range(len(top_5_str_df.columns) + 1)))

for key, cell in table.get_celld().items():
    cell.set_edgecolor('black')
    cell.set_linewidth(0.5)

plt.show()
```

ID	Wireless_internet	Heating	Kitchen	Essentials	Air_conditioning
6304928	True	True	True	True	True
7919400	True	True	True	True	True
5578513	True	True	True	True	True
18224863	True	True	True	True	True
16679342	False	True	True	False	True
14122244	True	True	True	True	True
14490881	True	True	True	True	True
4734170	True	True	True	True	True
421056	True	True	True	True	True
15705862	True	True	True	True	True

4. correlation (Top50/20)

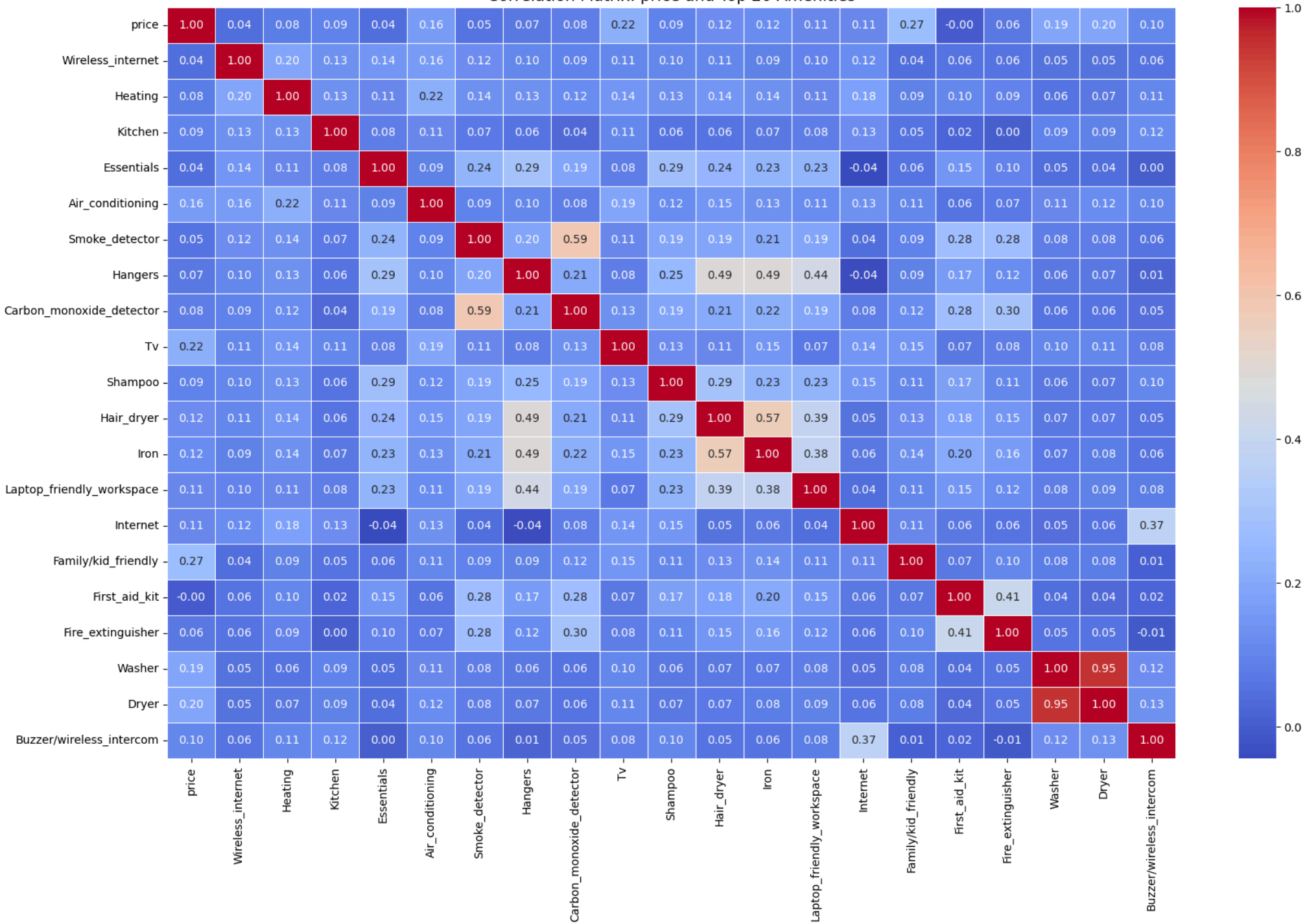
In [17]:
```
top_50_df = df_nyc[['price'] + [amenity.strip().replace(' ', '_') for amenity in top_50_amenities]]

correlation_matrix = top_50_df.corr()

plt.figure(figsize=(42, 20))
sns.heatmap(
```

```
    correlation_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    cbar=True,
    linewidths=0.5
)
plt.title("Correlation Matrix: price and Top 50 Amenities", fontsize=14)
plt.show()
```



In [18]:

```
top_20_amenities = amenities_counts.head(20).index

top_20_df = df_nyc[['price'] + [amenity.strip().replace(' ', '_') for amenity in top_20_amenities]]

correlation_matrix = top_20_df.corr()

plt.figure(figsize=(20, 12))
sns.heatmap(
    correlation_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    cbar=True,
    linewidths=0.5
)
plt.title("Correlation Matrix: price and Top 20 Amenities", fontsize=14)
plt.show()
```


Correlation Matrix: price and Top 20 Amenities
price 1.00 0.04 0.08 0.09 0.04 0.16 0.05 0.07 0.08 0.22 0.09 0.12 0.12 0.11 0.11 0.27 -0.00 0.06 0.19 0.20 0.10
Wireless_internet 0.04 1.00 0.20 0.13 0.14 0.16 0.12 0.10 0.09 0.11 0.10 0.11 0.09 0.10 0.12 0.04 0.06 0.06 0.05 0.05 0.06
Heating 0.08 0.20 1.00 0.13 0.11 0.22 0.14 0.13 0.12 0.14 0.13 0.14 0.14 0.11 0.18 0.09 0.10 0.09 0.06 0.07 0.11
Kitchen 0.09 0.13 0.13 1.00 0.08 0.11 0.07 0.06 0.04 0.11 0.06 0.06 0.07 0.08 0.13 0.05 0.02 0.00 0.09 0.09 0.12
Essentials 0.04 0.14 0.11 0.08 1.00 0.09 0.24 0.29 0.19 0.08 0.29 0.24 0.23 0.23 -0.04 0.06 0.15 0.10 0.05 0.04 0.00
Air_conditioning 0.16 0.16 0.22 0.11 0.09 1.00 0.09 0.10 0.08 0.19 0.12 0.15 0.13 0.11 0.13 0.11 0.06 0.07 0.11 0.12 0.10
Smoke_detector 0.05 0.12 0.14 0.07 0.24 0.09 1.00 0.20 0.59 0.11 0.19 0.19 0.21 0.19 0.04 0.09 0.28 0.28 0.08 0.08 0.06
Hangers 0.07 0.10 0.13 0.06 0.29 0.10 0.20 1.00 0.21 0.08 0.25 0.49 0.49 0.44 -0.04 0.09 0.17 0.12 0.06 0.07 0.01
Carbon_monoxide_detector 0.08 0.09 0.12 0.04 0.19 0.08 0.59 0.21 1.00 0.13 0.19 0.21 0.22 0.19 0.08 0.12 0.28 0.30 0.06 0.06 0.05
Tv 0.22 0.11 0.14 0.11 0.08 0.19 0.11 0.08 0.13 1.00 0.13 0.11 0.15 0.07 0.14 0.15 0.07 0.08 0.10 0.11 0.08
Shampoo 0.09 0.10 0.13 0.06 0.29 0.12 0.19 0.25 0.19 0.13 1.00 0.29 0.23 0.23 0.15 0.11 0.17 0.11 0.06 0.07 0.10
Hair_dryer 0.12 0.11 0.14 0.06 0.24 0.15 0.19 0.49 0.21 0.11 0.29 1.00 0.57 0.39 0.05 0.13 0.18 0.15 0.07 0.07 0.05
Iron 0.12 0.09 0.14 0.07 0.23 0.13 0.21 0.49 0.22 0.15 0.23 0.57 1.00 0.38 0.06 0.14 0.20 0.16 0.07 0.08 0.06
Laptop_friendly_workspace 0.11 0.10 0.11 0.08 0.23 0.11 0.19 0.44 0.19 0.07 0.23 0.39 0.38 1.00 0.04 0.11 0.15 0.12 0.08 0.09 0.08
Internet 0.11 0.12 0.18 0.13 -0.04 0.13 0.04 -0.04 0.08 0.14 0.15 0.05 0.06 0.04 1.00 0.11 0.06 0.06 0.05 0.06 0.37
Family/kid_friendly 0.27 0.04 0.09 0.05 0.06 0.11 0.09 0.09 0.12 0.15 0.11 0.13 0.14 0.11 0.11 1.00 0.07 0.10 0.08 0.08 0.01
First_aid_kit -0.00 0.06 0.10 0.02 0.15 0.06 0.28 0.17 0.28 0.07 0.17 0.18 0.20 0.15 0.06 0.07 1.00 0.41 0.04 0.04 0.02
Fire_extinguisher 0.06 0.06 0.09 0.00 0.10 0.07 0.28 0.12 0.30 0.08 0.11 0.15 0.16 0.12 0.06 0.10 0.41 1.00 0.05 0.05 -0.01
Washer 0.19 0.05 0.06 0.09 0.05 0.11 0.08 0.06 0.06 0.10 0.06 0.07 0.07 0.08 0.05 0.08 0.04 0.05 1.00 0.95 0.12
Dryer 0.20 0.05 0.07 0.09 0.04 0.12 0.08 0.07 0.06 0.11 0.07 0.07 0.08 0.09 0.06 0.08 0.04 0.05 0.95 1.00 0.13
Buzzer/wireless_intercom 0.10 0.06 0.11 0.12 0.00 0.10 0.06 0.01 0.05 0.08 0.10 0.05 0.06 0.08 0.37 0.01 0.02 -0.01 0.12 0.13 1.00
price
Wireless_internet
Heating
Kitchen
Essentials
Air_conditioning
Smoke_detector
Hangers
Carbon_monoxide_detector
Tv
Shampoo
Hair_dryer
Iron
Laptop_friendly_workspace
Internet
Family/kid_friendly
First_aid_kit
Fire_extinguisher
Washer
Dryer
Buzzer/wireless_intercom
1.0
0.8
0.6
0.4
0.2
0.0

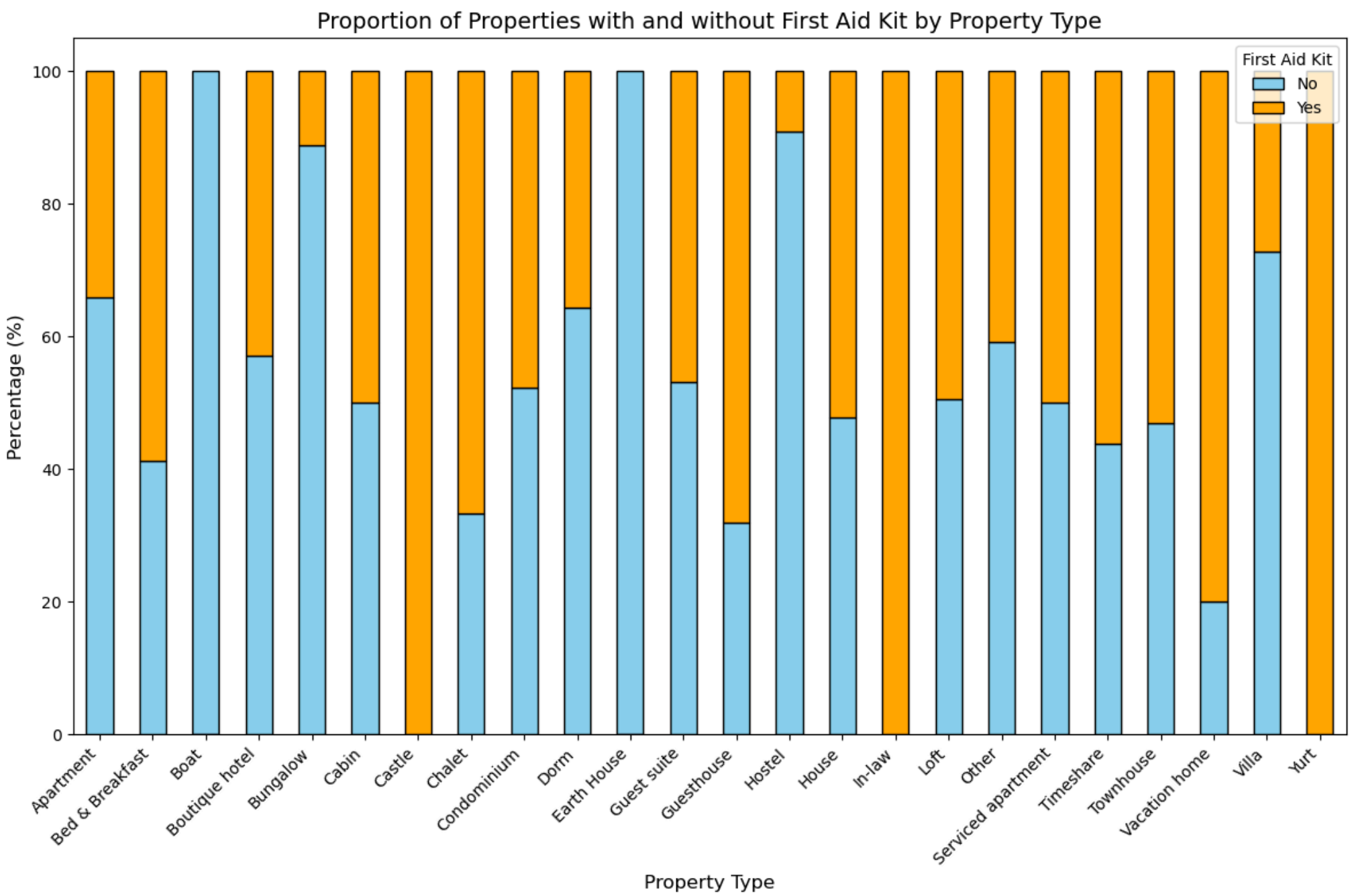
5. Analyze 'First Aid Kit'(negative correlation)

In [20]:
```
grouped_data = df_nyc.groupby(['property_type', 'First_aid_kit'])['price'].agg(['count', 'mean']).reset_index()

total_count_per_type = grouped_data.groupby('property_type')['count'].transform('sum')
grouped_data['percentage'] = grouped_data['count'] / total_count_per_type * 100

print(grouped_data)
```

	property_type	First_aid_kit	count	mean	percentage
0	Apartment	False	9606	140.692588	65.979806
1	Apartment	True	4953	136.769635	34.020194
2	Bed & Breakfast	False	26	114.269231	41.269841
3	Bed & Breakfast	True	37	153.702703	58.730159
4	Boat	False	3	142.000000	100.000000
5	Boutique hotel	False	4	113.500000	57.142857
6	Boutique hotel	True	3	139.000000	42.857143
7	Bungalow	False	8	161.250000	88.888889
8	Bungalow	True	1	95.000000	11.111111
9	Cabin	False	1	80.000000	50.000000
10	Cabin	True	1	250.000000	50.000000
11	Castle	True	1	175.000000	100.000000
12	Chalet	False	1	99.000000	33.333333
13	Chalet	True	2	175.500000	66.666667
14	Condominium	False	136	179.073529	52.307692
15	Condominium	True	124	247.241935	47.692308
16	Dorm	False	9	75.555556	64.285714
17	Dorm	True	5	51.600000	35.714286
18	Earth House	False	1	275.000000	100.000000
19	Guest suite	False	17	130.117647	53.125000
20	Guest suite	True	15	113.133333	46.875000
21	Guesthouse	False	8	82.500000	32.000000
22	Guesthouse	True	17	61.823529	68.000000
23	Hostel	False	10	67.000000	90.909091
24	Hostel	True	1	280.000000	9.090909
25	House	False	900	119.572222	47.745358
26	House	True	985	122.416244	52.254642
27	In-law	True	3	110.000000	100.000000
28	Loft	False	194	222.273196	50.520833
29	Loft	True	190	217.905263	49.479167
30	Other	False	71	133.521127	59.166667
31	Other	True	49	144.755102	40.833333
32	Serviced apartment	False	2	160.000000	50.000000
33	Serviced apartment	True	2	90.000000	50.000000
34	Timeshare	False	7	286.285714	43.750000
35	Timeshare	True	9	400.666667	56.250000
36	Townhouse	False	204	176.774510	47.004608
37	Townhouse	True	230	182.447826	52.995392
38	Vacation home	False	1	155.000000	20.000000
39	Vacation home	True	4	181.000000	80.000000
40	Villa	False	8	146.750000	72.727273
41	Villa	True	3	95.333333	27.272727
42	Yurt	True	1	95.000000	100.000000

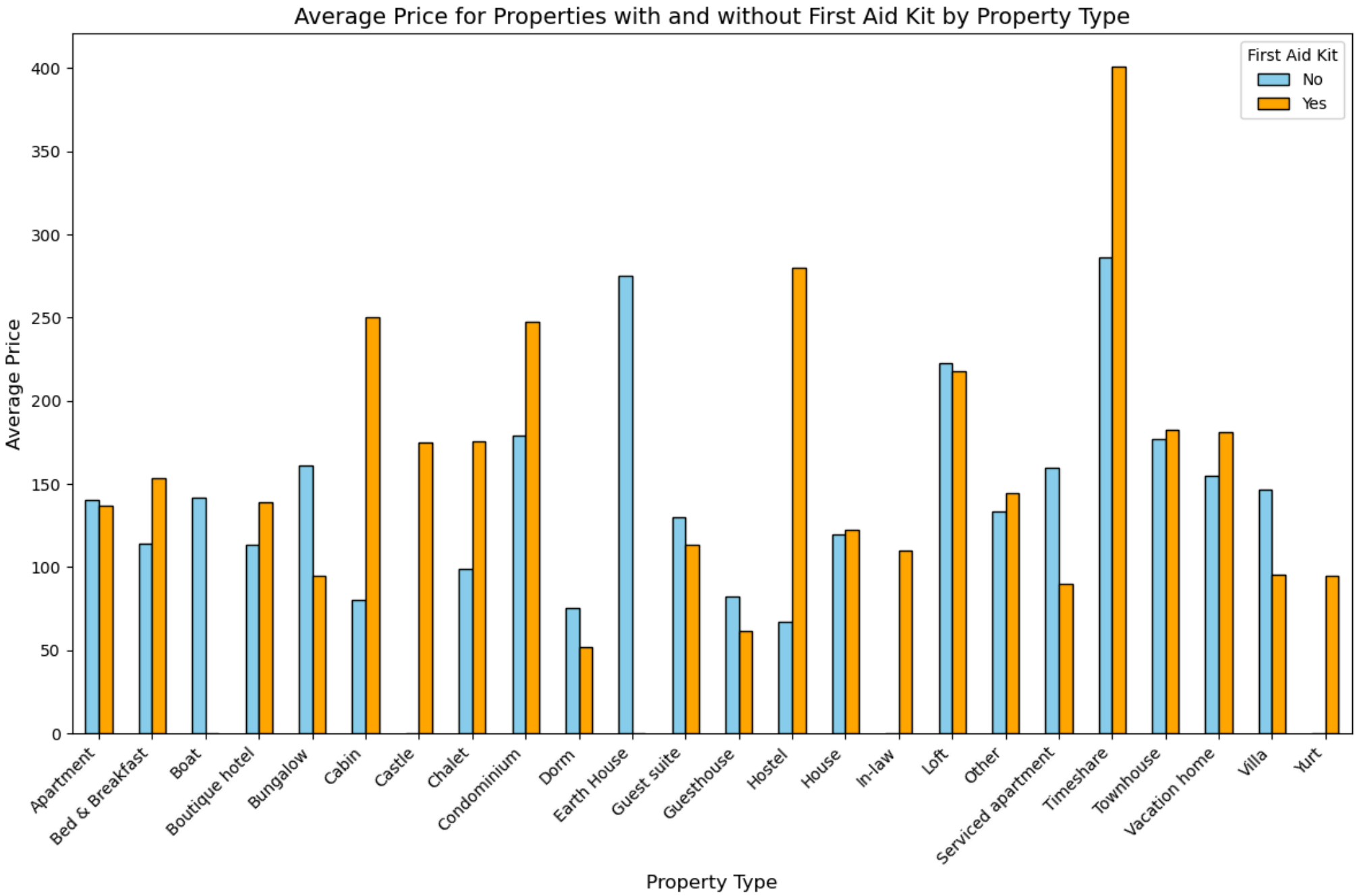
In [21]:

```
pivot_percentage = grouped_data.pivot(index='property_type', columns='First_aid_kit', values='percentage').fillna(0)

pivot_percentage.plot(kind='bar', stacked=True, figsize=(12, 8), color=['skyblue', 'orange'], edgecolor='black')
plt.title('Proportion of Properties with and without First Aid Kit by Property Type', fontsize=14)
plt.xlabel('Property Type', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.legend(title='First Aid Kit', labels=['No', 'Yes'])
plt.tight_layout()
plt.show()
```



In [22]:
```
pivot_mean = grouped_data.pivot(index='property_type', columns='First_aid_kit', values='mean').fillna(0)

pivot_mean.plot(kind='bar', figsize=(12, 8), color=['skyblue', 'orange'], edgecolor='black')
plt.title('Average Price for Properties with and without First Aid Kit by Property Type', fontsize=14)
plt.xlabel('Property Type', fontsize=12)
```

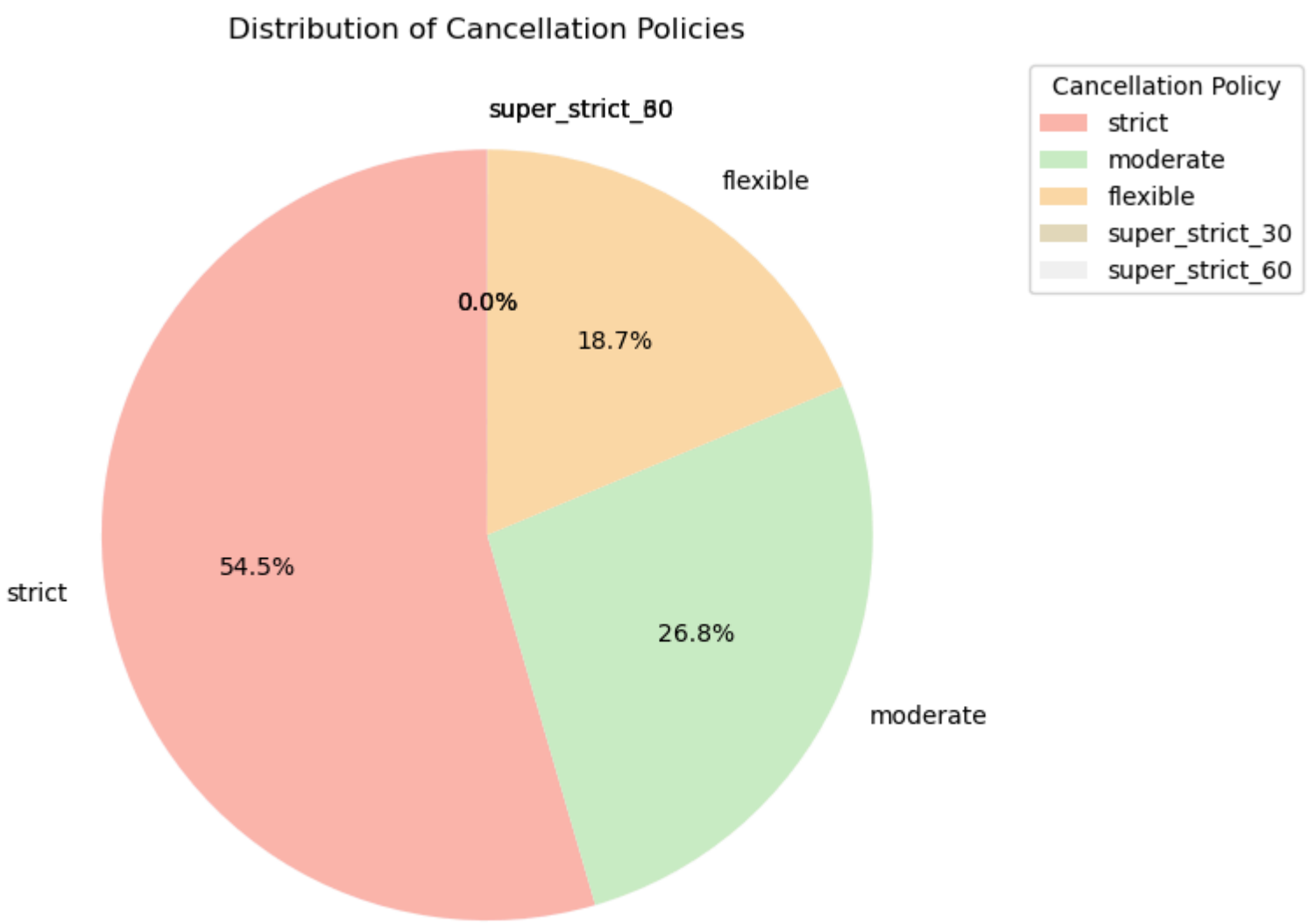
```
plt.ylabel('Average Price', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.legend(title='First Aid Kit', labels=['No', 'Yes'])
plt.tight_layout()
plt.show()
```



Cancellation policy

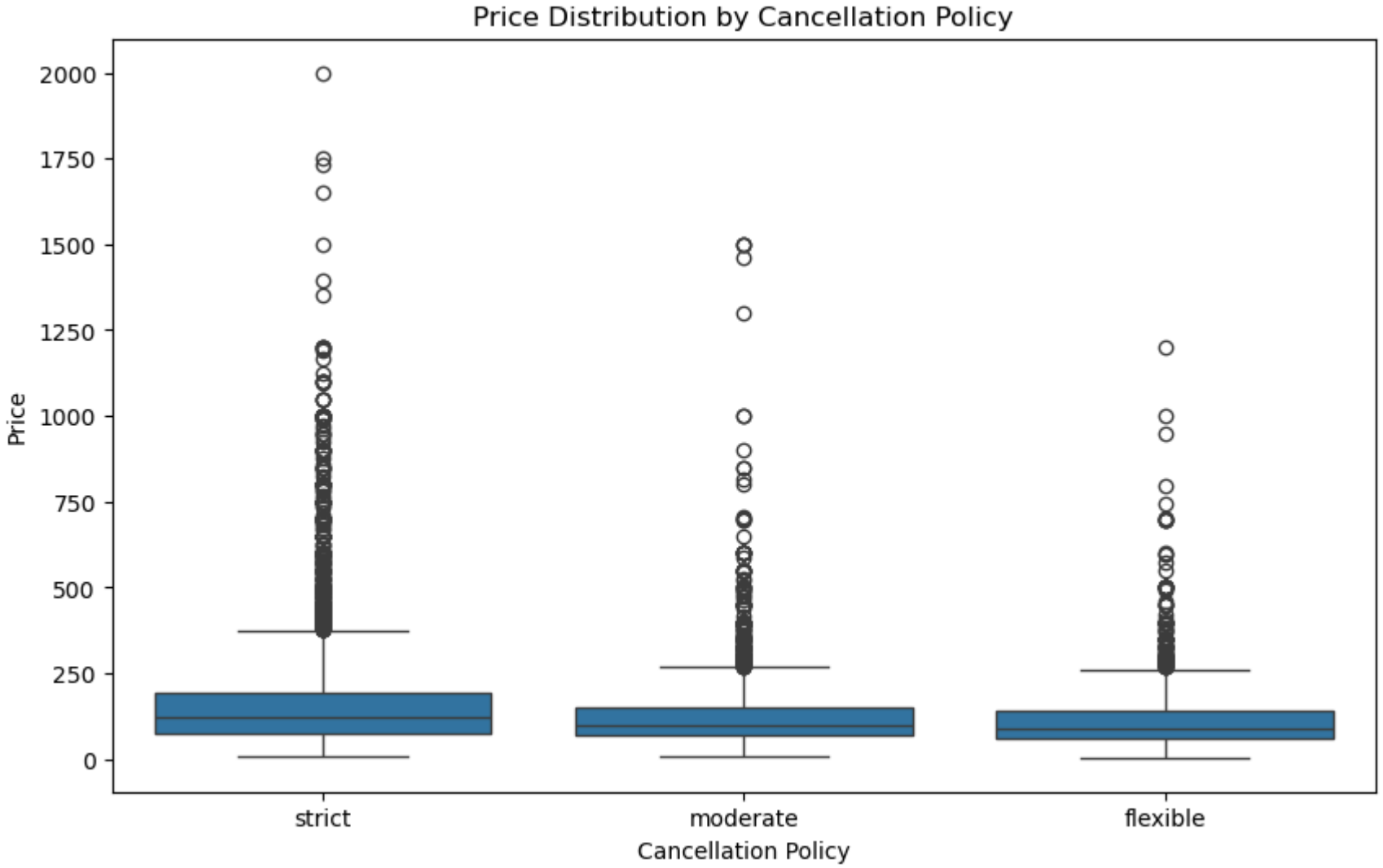
In [24]:
```
cancellation_counts = df_nyc['cancellation_policy'].value_counts()
print(cancellation_counts)

plt.figure(figsize=(8, 6))
cancellation_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, cmap='Pastel1')
plt.title('Distribution of Cancellation Policies')
plt.ylabel('')
plt.legend(title='Cancellation Policy', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

```
cancellation_policy
strict             9734
moderate           4783
flexible           3332
super_strict_30       3
super_strict_60       1
Name: count, dtype: int64
```



In [25]:
```
df_nyc = df_nyc[~df_nyc['cancellation_policy'].isin(['super_strict_30', 'super_strict_60'])]
```

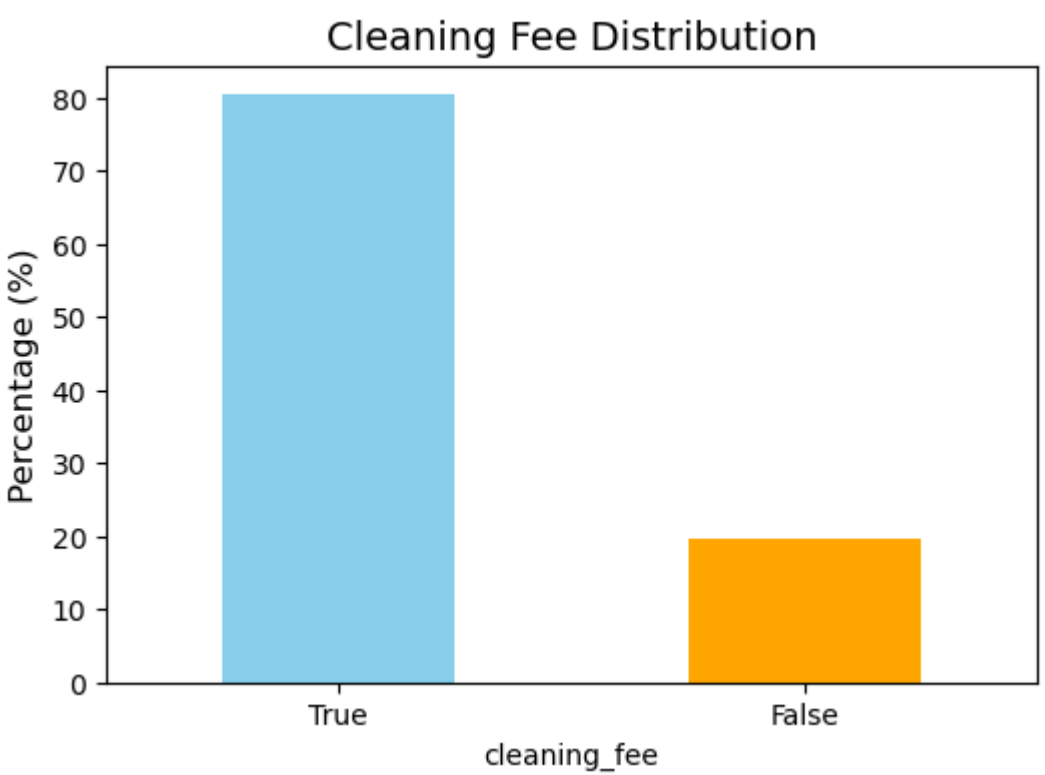
In [26]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_nyc, x='cancellation_policy', y='price')
plt.title('Price Distribution by Cancellation Policy')
plt.xlabel('Cancellation Policy')
plt.ylabel('Price')
plt.show()
```



Cleaning_fee

In [28]:
```
cleaning_fee_distribution = df_nyc['cleaning_fee'].value_counts(normalize=True) * 100
```

In [29]:
```
cleaning_fee_distribution.plot(kind='bar', color=['skyblue', 'orange'], figsize=(6, 4))
plt.title("Cleaning Fee Distribution", fontsize=14)
plt.ylabel("Percentage (%)", fontsize=12)
plt.xticks(rotation=0)
plt.show()
```



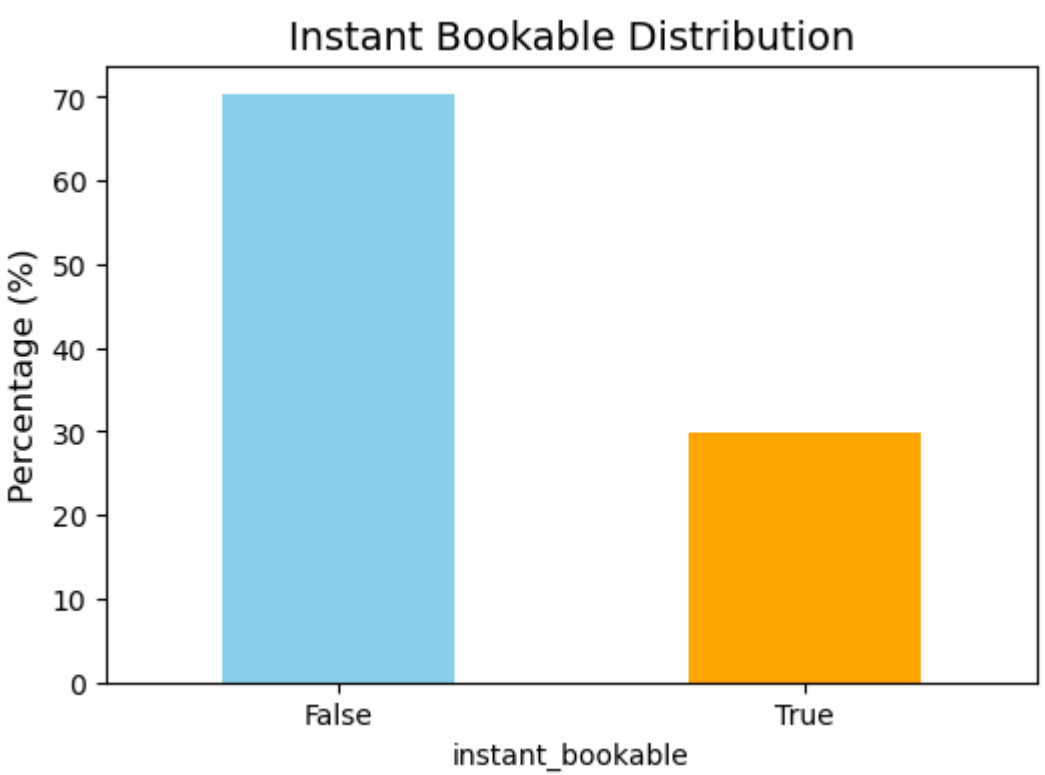
In [30]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_nyc, x='cleaning_fee', y='price')
plt.title("Average Price by Cleaning Fee", fontsize=14)
plt.ylabel("Average Price", fontsize=12)
plt.xticks(rotation=0)
plt.show()
```

Instant_bookable

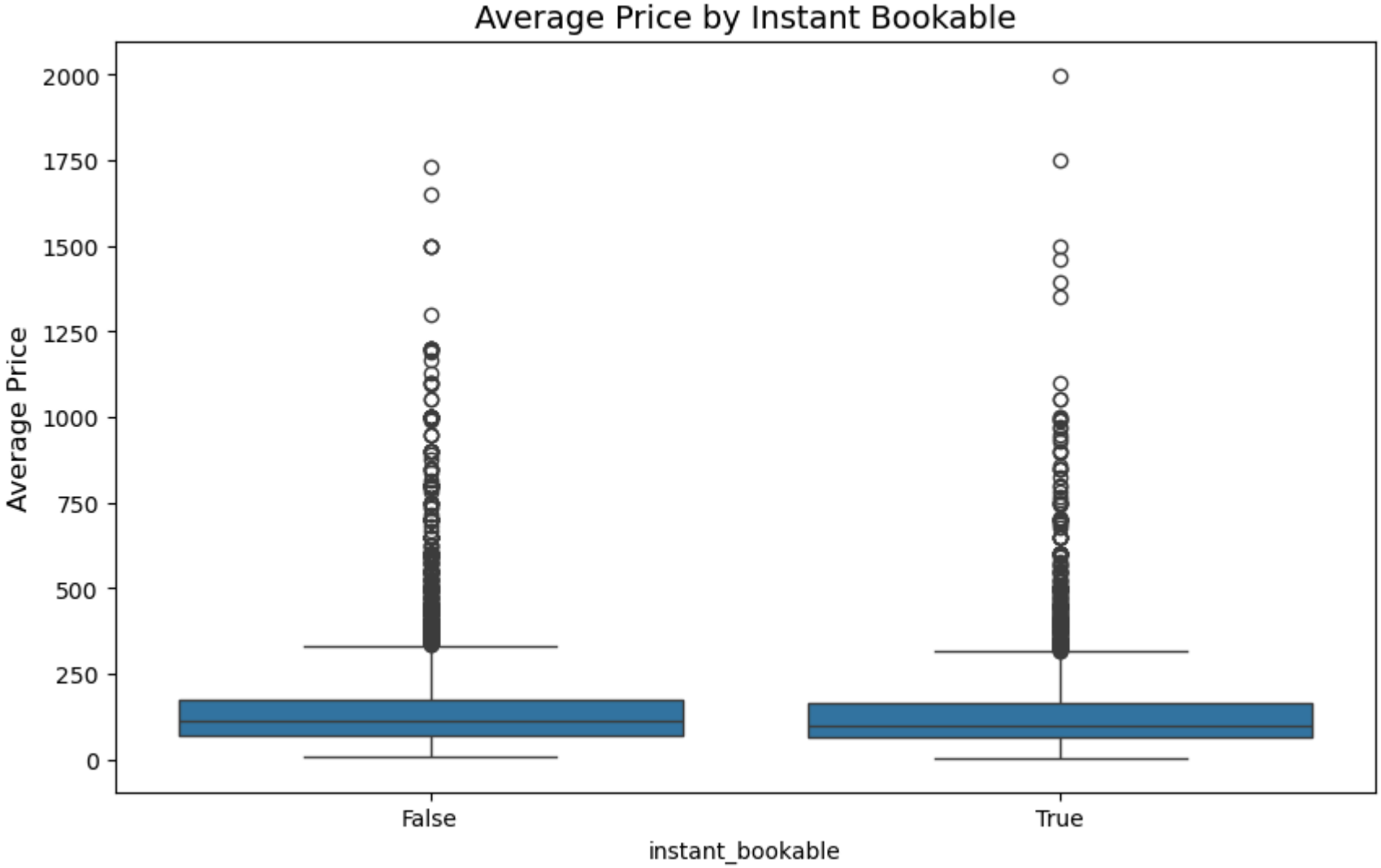
In [32]:
```
df_nyc['instant_bookable'] = df_nyc['instant_bookable'].map({'t': True, 'f': False})
```

In [33]:
```
instant_bookable_distribution = df_nyc['instant_bookable'].value_counts(normalize=True) * 100
```

In [34]:
```
instant_bookable_distribution.plot(kind='bar', color=['skyblue', 'orange'], figsize=(6, 4))
plt.title("Instant Bookable Distribution", fontsize=14)
plt.ylabel("Percentage (%)", fontsize=12)
plt.xticks(rotation=0)
plt.show()
```



In [35]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_nyc, x='instant_bookable', y='price')
plt.title("Average Price by Instant Bookable", fontsize=14)
plt.ylabel("Average Price", fontsize=12)
plt.xticks(rotation=0)
plt.show()
```



In [36]:
```
df_nyc['instant_bookable_numeric'] = df_nyc['instant_bookable'].map({True: 1, False: 0})

correlation = df_nyc['instant_bookable_numeric'].corr(df_nyc['price'])

print(f"Correlation between instant_bookable and price: {correlation:.2f}")
```

```
Correlation between instant_bookable and price: -0.03
```

Appendix A.4

4. Preprocessing_Quality

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df_nyc = pd.read_csv('nyc_with_distances.csv')
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_18868\3094127854.py:1: DtypeWarning: Columns (26) have mixed types. Specify dtype option on impo
rt or set low_memory=False.
  df_nyc = pd.read_csv('nyc_with_distances.csv')
```

In [4]:
```
print(df_nyc['review_scores_rating'].describe())
print(df_nyc['number_of_reviews'].describe())
```

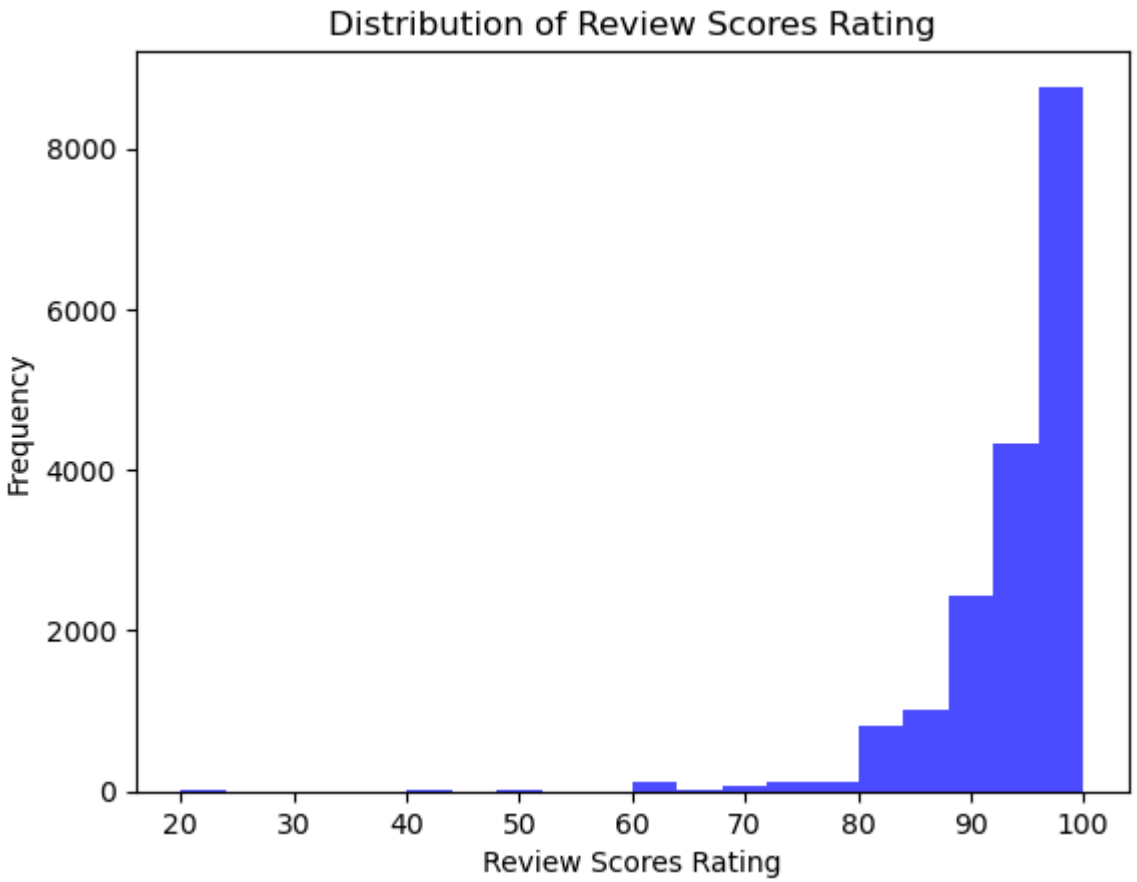
```
count    17853.000000
mean        93.758976
std          7.289859
min         20.000000
25%         91.000000
50%         95.000000
75%         99.000000
max        100.000000
Name: review_scores_rating, dtype: float64
count    17853.000000
mean        28.760432
std         38.861156
min          1.000000
25%          4.000000
50%         14.000000
75%         37.000000
max        465.000000
Name: number_of_reviews, dtype: float64
```

In [5]:
```
correlation_matrix = df_nyc[['price', 'review_scores_rating', 'number_of_reviews']].corr()
print(correlation_matrix)

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix: Quality Features and Price')
plt.show()
```

```
                        price  review_scores_rating  number_of_reviews
price                1.000000              0.032804          -0.010646
review_scores_rating  0.032804              1.000000          -0.043788
number_of_reviews    -0.010646             -0.043788           1.000000
```

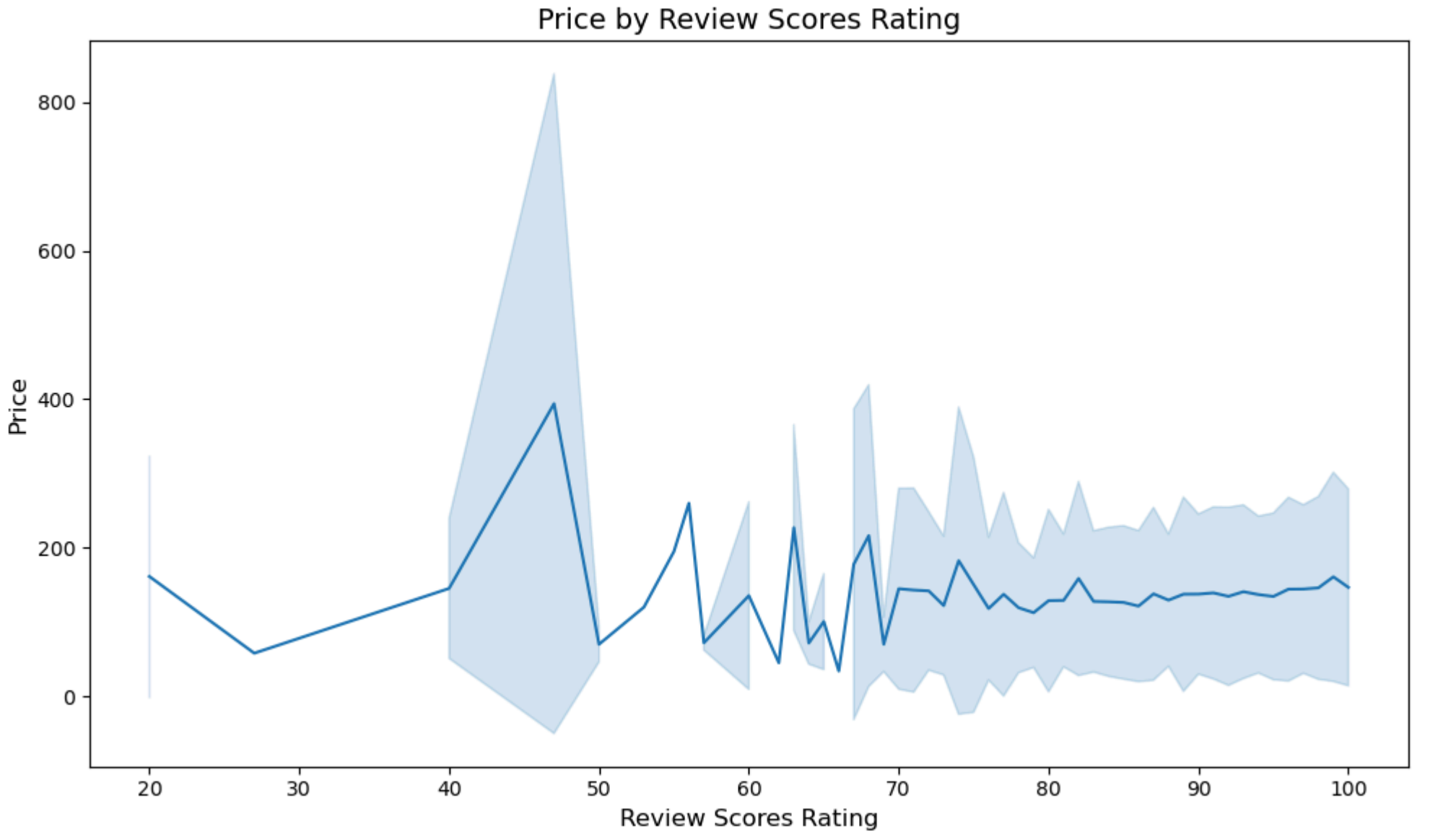
Review Scores Rating

In [7]:
```
plt.hist(df_nyc['review_scores_rating'], bins=20, color='blue', alpha=0.7)
plt.title('Distribution of Review Scores Rating')
plt.xlabel('Review Scores Rating')
plt.ylabel('Frequency')
plt.show()
```



In [8]:
```
plt.figure(figsize=(10, 6))
sns.lineplot(
    data=df_nyc,
    x='review_scores_rating',
    y='price',
    ci='sd',
    estimator='mean',
)

plt.title("Price by Review Scores Rating", fontsize=14)
plt.xlabel("Review Scores Rating", fontsize=12)
plt.ylabel("Price", fontsize=12)
plt.tight_layout()
plt.show()
```

```
C:\Users\sylvi\AppData\Local\Temp\ipykernel_18868\631567302.py:2: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  sns.lineplot(
```

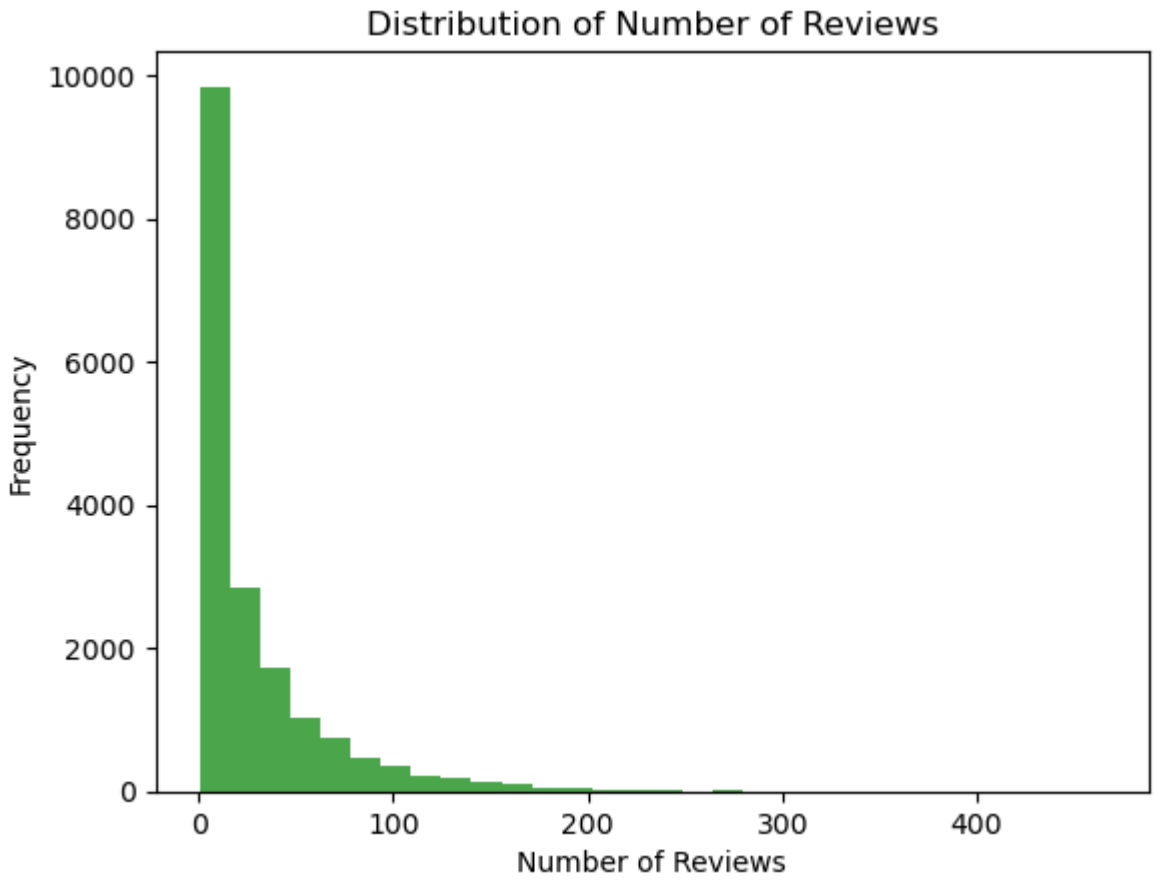


In [9]:
```
x = df_nyc['review_scores_rating']
y = df_nyc['price']

plt.figure(figsize=(8, 6))
plt.hexbin(x, y, gridsize=30, cmap='Greens', bins='log')
```

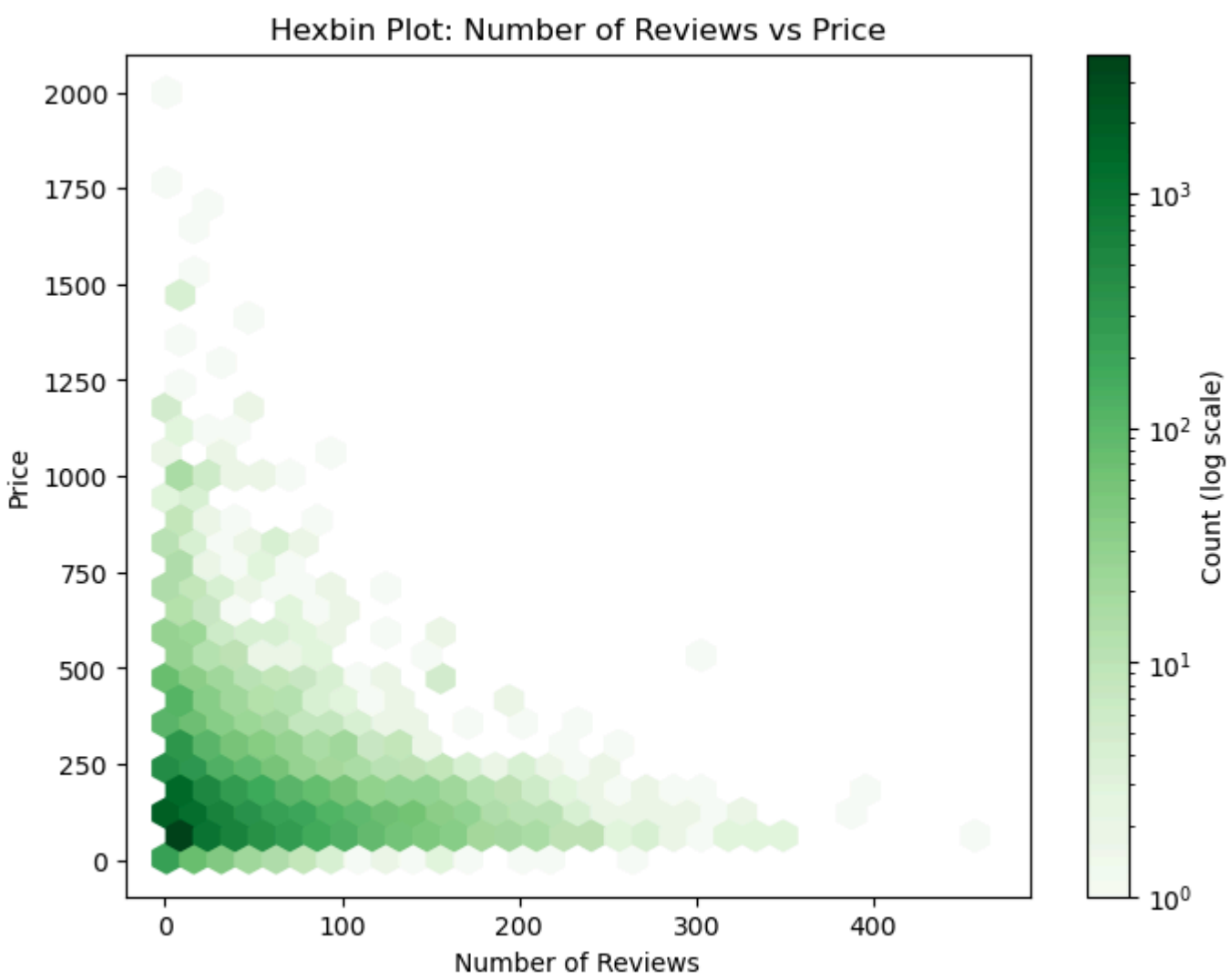
```
plt.colorbar(label='Count (log scale)')
plt.xlabel('Review Scores Rating')
plt.ylabel('Price')
plt.title('Hexbin Plot: Review Scores Rating vs Price')
plt.show()
```

Number of reviews

In [11]:
```
plt.hist(df_nyc['number_of_reviews'], bins=30, color='green', alpha=0.7)
plt.title('Distribution of Number of Reviews')
plt.xlabel('Number of Reviews')
plt.ylabel('Frequency')
plt.show()
```



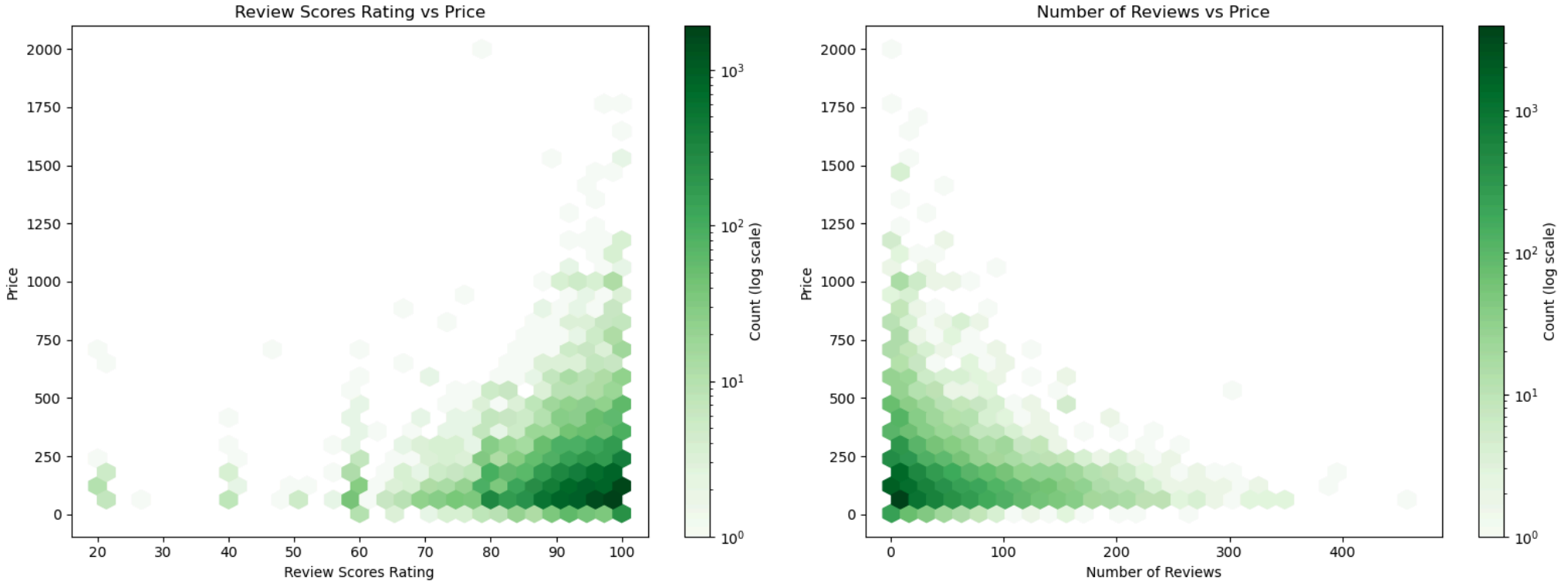
In [12]:
```
x = df_nyc['number_of_reviews']
y = df_nyc['price']

plt.figure(figsize=(8, 6))
plt.hexbin(x, y, gridsize=30, cmap='Greens', bins='log')
plt.colorbar(label='Count (log scale)')
plt.xlabel('Number of Reviews')
plt.ylabel('Price')
plt.title('Hexbin Plot: Number of Reviews vs Price')
plt.show()
```



In [24]:

```
# Merge the plots
x1 = df_nyc['review_scores_rating']
y1 = df_nyc['price']

x2 = df_nyc['number_of_reviews']
y2 = df_nyc['price']

fig, axs = plt.subplots(1, 2, figsize=(16, 6))

hb1 = axs[0].hexbin(x1, y1, gridsize=30, cmap='Greens', bins='log')
axs[0].set_xlabel('Review Scores Rating')
axs[0].set_ylabel('Price')
axs[0].set_title('Review Scores Rating vs Price')
```

```
cb1 = fig.colorbar(hb1, ax=axs[0])
cb1.set_label('Count (log scale)')

hb2 = axs[1].hexbin(x2, y2, gridsize=30, cmap='Greens', bins='log')
axs[1].set_xlabel('Number of Reviews')
axs[1].set_ylabel('Price')
axs[1].set_title('Number of Reviews vs Price')
cb2 = fig.colorbar(hb2, ax=axs[1])
cb2.set_label('Count (log scale)')

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



In [13]:

```
filtered_data = df_nyc[(df_nyc['number_of_reviews'] < 5) | (df_nyc['number_of_reviews']>300)]

grouped_data = filtered_data.groupby(['room_type', 'number_of_reviews'])['price'].mean().reset_index()


plt.figure(figsize=(12, 6))
sns.barplot(data=grouped_data, x='number_of_reviews', y='price', hue='room_type')
plt.title('Average Price by Room Type and Review Count Range', fontsize=14)
plt.xlabel('Review Count Range', fontsize=12)
plt.ylabel('Average Price', fontsize=12)
```

```
plt.legend(title='Room Type', fontsize=10)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Appendix A.5

5. Model

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder, RobustScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.pipeline import Pipeline
from lightgbm import LGBMRegressor
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from scikeras.wrappers import KerasRegressor
from tqdm import tqdm
import warnings
import matplotlib

matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # Set the default font
matplotlib.rcParams['axes.unicode_minus'] = False # Fix the issue where minus signs ('-') are displayed as boxes in saved images

warnings.filterwarnings('ignore')
```

In [3]:
```
class HousingPricePredictor:
    def __init__(self, file_path):
        """Initialize the housing price predictor"""
        self.file_path = file_path
        self.df = None
        self.X = None
        self.y = None
        self.models = {}
        self.scaler = RobustScaler()

    def load_data(self):
        """Load data and display basic information"""
        print("\n" + "=" * 50)
        print("Loading and Exploring Data")
        print("=" * 50)

        self.df = pd.read_csv(self.file_path)
```

```
        print(f"\nDataset shape: {self.df.shape}")
        print(f"Total records: {self.df.shape[0]}")
        print(f"Total features: {self.df.shape[1]}")

        sample_cols = ['id', 'log_price', 'property_type', 'room_type',
                       'accommodates', 'bathrooms', 'bedrooms', 'beds',
                       'latitude', 'longitude', 'review_scores_rating']
        sample_cols = [col for col in sample_cols if col in self.df.columns]
        print("\nSample data (first 2 rows of key columns):")
        print("-" * 50)
        print(self.df[sample_cols].head(2))

        return self

    def prepare_features(self):
        """Feature preparation and preprocessing"""
        print("\n" + "=" * 50)
        print("Feature Engineering and Preprocessing")
        print("=" * 50)

        # Ensure copy is used to avoid SettingWithCopyWarning
        df_processed = self.df.copy()

        # 0. Target variable transformation
        df_processed['actual_price'] = np.exp(df_processed['log_price'])

        # 1. Location features
        location_features = [
            'distance_to_jfk', 'distance_to_lga', 'distance_to_ewr',
            'distance_to_manhattan', 'manhattan_times_square_distance',
            'manhattan_central_park_distance', 'manhattan_empire_state_building_distance',
            'brooklyn_brooklyn_bridge_distance', 'brooklyn_prospect_park_distance',
            'latitude', 'longitude'
        ]

        # Check if the dataset contains other regional features
        queens_features = [col for col in df_processed.columns if 'queens_' in col.lower()]
        bronx_features = [col for col in df_processed.columns if 'bronx_' in col.lower()]
        staten_island_features = [col for col in df_processed.columns if 'staten_island_' in col.lower()]

        # If other regional features are found, add them to the location features list
        if queens_features:
            location_features.extend(queens_features)
            print(f"\nFound Queens features: {queens_features}")

        if bronx_features:
```

```
        location_features.extend(bronx_features)
        print(f"\nFound Bronx features: {bronx_features}")

    if staten_island_features:
        location_features.extend(staten_island_features)
        print(f"\nFound Staten Island features: {staten_island_features}")

    # Keep only columns that exist in the dataset
    available_location_features = [f for f in location_features if f in df_processed.columns]
    if len(available_location_features) != len(location_features):
        missing_features = set(location_features) - set(available_location_features)
        print(f"\nNote: The following location features are missing from the dataset: {missing_features}")

    location_features = available_location_features
    print(f"\nLocation features being used: {location_features}")

    # 2. Basic features
    basic_features = ['accommodates', 'bathrooms', 'bedrooms', 'beds']

    # 3. Facility and service features
    service_features = []

    # Process boolean features
    if 'instant_bookable' in df_processed.columns:
        df_processed['instant_bookable'] = df_processed['instant_bookable'].map({
            't': 1, 'f': 0, True: 1, False: 0, 'TRUE': 1, 'FALSE': 0
        }).fillna(0).astype(int)
        service_features.append('instant_bookable')

    if 'cleaning_fee' in df_processed.columns:
        df_processed['cleaning_fee'] = df_processed['cleaning_fee'].map({
            't': 1, 'f': 0, True: 1, False: 0, 'TRUE': 1, 'FALSE': 0
        }).fillna(0).astype(int)
        service_features.append('cleaning_fee')

    # Extract number of amenities
    if 'amenities' in df_processed.columns:
        df_processed['amenities_list'] = df_processed['amenities'].apply(
            lambda x: [item.strip().strip('"') for item in x.strip('{}').split(',')] if pd.notnull(x) else []
        )
        df_processed['amenities_list'] = df_processed['amenities_list'].apply(
            lambda x: [item for item in x if not item.lower().startswith('translation missing')]
        )
        df_processed['amenities_list'] = df_processed['amenities_list'].apply(
            lambda x: [item.strip().capitalize() for item in x]
        )
```

```
    df_processed['amenities_count'] = df_processed['amenities_list'].apply(len)
    service_features.append('amenities_count')

# 4. Quality features
quality_features = []
if 'review_scores_rating' in df_processed.columns:
    quality_features.append('review_scores_rating')
if 'number_of_reviews' in df_processed.columns:
    quality_features.append('number_of_reviews')

# Create advanced features
print("\nCreating advanced features...")

# Price per accommodate feature
df_processed['price_per_accommodates'] = df_processed['actual_price'] / df_processed['accommodates'].clip(
    lower=1)
advanced_features = ['price_per_accommodates']

# Bed density
if 'bedrooms' in df_processed.columns and 'beds' in df_processed.columns:
    df_processed['beds_per_bedroom'] = df_processed['beds'] / df_processed['bedrooms'].clip(lower=1)
    advanced_features.append('beds_per_bedroom')

# Manhattan distance effect
if 'distance_to_manhattan' in df_processed.columns:
    df_processed['manhattan_effect'] = np.exp(-0.1 * df_processed['distance_to_manhattan'])
    advanced_features.append('manhattan_effect')

# Handle 'cancellation policy'
if 'cancellation_policy' in df_processed.columns:
    df_processed = df_processed[
        ~df_processed['cancellation_policy'].isin(['super_strict_30', 'super_strict_60'])
    ]
    print(f"\nFiltered cancellation policies, keeping only strict, moderate, and flexible:")
    print(df_processed['cancellation_policy'].value_counts())

# Handle categorical features
categorical_features = ['property_type', 'room_type', 'bed_type', 'cancellation_policy']
encoded_cat_features = []

print("\nEncoding categorical features...")
for col in categorical_features:
    if col in df_processed.columns:
        df_processed[col] = df_processed[col].astype(str)
        df_processed[col] = df_processed[col].replace('nan', 'Unknown').fillna('Unknown')
        le = LabelEncoder()
```

```
            df_processed[f'{col}_encoded'] = le.fit_transform(df_processed[col])
            encoded_cat_features.append(f'{col}_encoded')

    # Combine all features
    numeric_features = (location_features + basic_features +
                        service_features + quality_features +
                        advanced_features)
    numeric_features = [f for f in numeric_features if f in df_processed.columns]

    # Handle missing values in numerical features
    print("\nHandling missing values in numeric features...")
    for col in numeric_features:
        df_processed[col] = pd.to_numeric(df_processed[col], errors='coerce')
        if df_processed[col].isnull().sum() > 0:
            median_val = df_processed[col].median()
            df_processed[col].fillna(median_val, inplace=True)

    # Prepare final feature set
    all_features = numeric_features + encoded_cat_features

    # Remove outliers
    print("\nRemoving outliers...")
    Q1 = df_processed['actual_price'].quantile(0.05)
    Q3 = df_processed['actual_price'].quantile(0.95)
    IQR = Q3 - Q1
    price_mask = (df_processed['actual_price'] >= Q1 - 1.5 * IQR) & (df_processed['actual_price'] <= Q3 + 1.5 * IQR)
    df_processed = df_processed[price_mask]

    # Save the processed DataFrame
    self.df_processed = df_processed

    # Create final feature matrix and target variable
    self.X = df_processed[all_features]
    self.y = df_processed['actual_price']

    print(f"\nFinal feature matrix shape: {self.X.shape}")
    print(f"Target variable shape: {self.y.shape}")
    print(f"\nSelected feature list:")
    for f in all_features:
        print(f"  - {f}")

    # Display summary table
    self._display_summary_table(df_processed)

    return self
```

```python
def _display_summary_table(self, df):
    print("\n" + "=" * 50)
    print("Summary Statistics")
    print("=" * 50)

    quality_vars = [
        'accommodates', 'bathrooms', 'bedrooms', 'beds',
        'property_type', 'room_type', 'bed_type',
        'price_per_accommodates', 'beds_per_bedroom',
        'instant_bookable', 'cleaning_fee', 'amenities_count',
        'review_scores_rating', 'number_of_reviews',
        'manhattan_effect',
        'distance_to_jfk', 'distance_to_lga', 'distance_to_ewr', 'distance_to_manhattan'
    ]

    quality_vars = [var for var in quality_vars if var in df.columns]
    numerical_vars = [var for var in quality_vars
                      if df[var].dtype != 'object' and var not in ['property_type', 'room_type', 'bed_type']]

    summary_stats = df[numerical_vars].describe().T.round(2)
    print("\nSummary of numerical variables:")
    print(summary_stats)

    categorical_vars = [var for var in quality_vars if var in ['property_type', 'room_type', 'bed_type']]
    if categorical_vars:
        print("\nTop categories of categorical variables:")
        for var in categorical_vars:
            if var in df.columns:
                print(f"\n{var} (Top 5 categories):")
                print(df[var].value_counts().head(5))


def train_and_evaluate(self):
    """Train and evaluate the optimized model"""
    print("\n" + "=" * 50)
    print("Model Training and Evaluation")
    print("=" * 50)

    # Split the dataset
    X_train, X_test, y_train, y_test = train_test_split(
        self.X, self.y, test_size=0.2, random_state=42
    )

    # Feature standardization
    X_train_scaled = self.scaler.fit_transform(X_train)
    X_test_scaled = self.scaler.transform(X_test)
```

```python
# 1. Train KNN model
    print("\nTraining KNN model...")
    knn_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('knn', KNeighborsRegressor())
    ])

    # Define parameter grid
    knn_param_grid = {'knn__n_neighbors': [3, 5, 7, 10, 15, 20]}

    # Run grid search
    print("Starting grid search...")
    grid_knn = GridSearchCV(
        estimator=knn_pipeline,
        param_grid=knn_param_grid,
        scoring='neg_root_mean_squared_error',
        cv=5,
        n_jobs=-1,
        verbose=1
    )

    # Fit and store best model
    grid_knn.fit(X_train, y_train)
    self.models['KNN'] = grid_knn.best_estimator_

    # Print best parameters
    print(f"  - Best k: {grid_knn.best_params_['knn__n_neighbors']}")


# 2. Train Random Forest Model
    print("\nTraining Random Forest Model...")

    # Define the parameter grid
    rf_param_grid = {
        'n_estimators': [100, 300, 500],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5],
        'min_samples_leaf': [1, 2]
    }

    # Run grid search
    print("Starting grid search...")
    grid_rf = GridSearchCV(
        estimator=RandomForestRegressor(random_state=42, n_jobs=-1),
        param_grid=rf_param_grid,
```

```
            scoring='neg_root_mean_squared_error',
            cv=5,
            n_jobs=-1,
            verbose=1
        )

        # Fit and store best model
        grid_rf.fit(X_train, y_train)
        self.models['Random Forest'] = grid_rf.best_estimator_

        # Print best parameters
        for param, value in grid_rf.best_params_.items():
            print(f"  - {param}: {value}")


    # 3. Train Gradient Boosting model
        print("\nTraining Gradient Boosting model...")

        # Define parameter grid
        gb_param_grid = {
            'n_estimators': [100, 300],
            'learning_rate': [0.05, 0.1],
            'max_depth': [3, 5],
            'subsample': [0.8],
            'min_samples_split': [5],
            'min_samples_leaf': [5]
        }

        # Run grid search
        print("Starting grid search...")
        grid_gb = GridSearchCV(
            estimator=GradientBoostingRegressor(random_state=42),
            param_grid=gb_param_grid,
            scoring='neg_root_mean_squared_error',
            cv=5,
            n_jobs=-1,
            verbose=1
        )

        # Fit and store best model
        grid_gb.fit(X_train, y_train)
        self.models['Gradient Boosting'] = grid_gb.best_estimator_

        # Print best parameters
        for param, val in grid_gb.best_params_.items():
            print(f"  - {param}: {val}")
```

```python
# 4. Train LightGBM Model
    print("\nTraining LightGBM Model...")

    # Define parameter grid
    lgbm_param_grid = {
        'n_estimators': [100, 300],
        'learning_rate': [0.03, 0.05],
        'max_depth': [5, 10],
        'num_leaves': [31],
        'subsample': [0.8],
        'colsample_bytree': [0.8]
    }

    # Run grid search
    print("Starting grid search...")
    grid_lgbm = GridSearchCV(
        estimator=LGBMRegressor(random_state=42, verbose=-1, reg_alpha=0.0, reg_lambda=0.0),
        param_grid=lgbm_param_grid,
        scoring='neg_root_mean_squared_error',
        cv=5,
        n_jobs=-1,
        verbose=1
    )

    # Fit and store best model
    grid_lgbm.fit(X_train, y_train)
    self.models['LightGBM'] = grid_lgbm.best_estimator_

    # Print best parameters
    for param, val in grid_lgbm.best_params_.items():
        print(f"  - {param}: {val}")


 # 5. Train neural network model
    print("\nTraining neural network model...")

    # Define a function to build the ANN model
    def build_ann():
        model = Sequential([
            Dense(128, activation='relu', input_dim=X_train_scaled.shape[1]),
            BatchNormalization(),
            Dropout(0.2),
            Dense(64, activation='relu'),
            BatchNormalization(),
```

```
            Dropout(0.2),
            Dense(32, activation='relu'),
            BatchNormalization(),
            Dropout(0.2),
            Dense(1)
        ])
        model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
        return model

    # Create KerasRegressor and specify hyperparameters
    ann_regressor = KerasRegressor(
        model=build_ann,
        epochs=50,
        batch_size=64,
        verbose=1
    )

    # Train the model
    print("Training the neural network...")
    ann_regressor.fit(X_train_scaled, y_train)
    self.models['ANN'] = ann_regressor

# Evaluate all models
    print("\n" + "=" * 50)
    print("Model Evaluation Metrics")
    print("=" * 50)

    results = {}
    model_names = list(self.models.keys())

    for model_name in tqdm(model_names, desc="Evaluating models"):
        model = self.models[model_name]
        if model_name == 'ANN':
            y_pred = model.predict(X_test_scaled).flatten()
        else:
            y_pred = model.predict(X_test)

        if not hasattr(self, 'predictions'):
            self.predictions = {}
        self.predictions[model_name] = y_pred

        metrics = self._calculate_metrics(y_test, y_pred)
        results[model_name] = metrics

        print(f"\n{model_name} Model Evaluation Metrics:")
        print(f"  R² (R-squared): {metrics['R²']:.4f}")
```

```
            print(f"  MSE (Mean Squared Error): {metrics['MSE']:.4f}")
            print(f"  RMSE (Root Mean Squared Error): {metrics['RMSE']:.4f}")
            print(f"  MAE (Mean Absolute Error): {metrics['MAE']:.4f}")
            print(f"  MAPE (Mean Absolute Percentage Error): {metrics['MAPE']:.4f}%")

        comparison_df = pd.DataFrame(results).T
        print("\n" + "=" * 50)
        print("Model Performance Comparison")
        print("=" * 50)
        print("\nComparison results of the five evaluation metrics:")
        print(comparison_df.to_string(float_format='%.4f'))

        print("\nBest Model Analysis:")
        print("-" * 40)
        for metric in comparison_df.columns:
            if metric == 'R²':
                best_model = comparison_df[metric].idxmax()
                best_value = comparison_df[metric].max()
                print(f"{metric}: {best_model} ({best_value:.4f})")
            else:
                best_model = comparison_df[metric].idxmin()
                best_value = comparison_df[metric].min()
                print(f"{metric}: {best_model} ({best_value:.4f})")

        if 'Random Forest' in self.models:
            self._analyze_feature_importance(self.models['Random Forest'])

        self._create_visualizations(X_test, X_test_scaled, y_test)
        comparison_df.to_csv('Model_Performance_Comparison.csv', encoding='utf-8-sig')
        print("\nModel performance comparison results saved to 'Model_Performance_Comparison.csv'")

        return self

    def _calculate_metrics(self, y_true, y_pred):
        """Compute evaluation metrics"""
        metrics = {}
        metrics['R²'] = r2_score(y_true, y_pred)
        metrics['MSE'] = mean_squared_error(y_true, y_pred)
        metrics['RMSE'] = np.sqrt(metrics['MSE'])
        metrics['MAE'] = mean_absolute_error(y_true, y_pred)
        metrics['MAPE'] = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
        return metrics

    def _analyze_feature_importance(self, model):
        """Analyze feature importance"""
        print("\n" + "=" * 50)
```

```
        print("Feature Importance Analysis")
        print("=" * 50)

        # Get and sort feature importances
        feature_importance = pd.DataFrame({
            'Feature Name': self.X.columns,
            'Importance Score': model.feature_importances_
        }).sort_values('Importance Score', ascending=False)

        # Display top 15 important features
        print("\nTop 15 Most Important Features: ")
        print("-" * 60)
        top_features = feature_importance.head(15)
        for idx, (feature, importance) in enumerate(zip(top_features['Feature Name'], top_features['Importance Score']), 1):
            print(f"{idx:2d}. {feature:35} {importance:.4f}")

        plt.figure(figsize=(12, 8))
        sns.barplot(x='Importance Score', y='Feature Name', data=top_features)
        plt.title('Top 15 Most Important Features')
        plt.tight_layout()
        plt.savefig('feature_importance.png')
        plt.close()

        feature_importance.to_csv('feature_importance.csv', encoding='utf-8-sig', index=False)
        print("\nFeature importance has been saved to 'feature_importance.csv'")
        print("Feature importance plot has been saved to 'feature_importance.png'")

    def _create_visualizations(self, X_test, X_test_scaled, y_test):
        print("\n" + "=" * 50)
        print("Creating Visualizations")
        print("=" * 50)

        if not hasattr(self, 'predictions'):
            print("No prediction results available for visualization.")
            return

        visualization_steps =  ["Actual vs Predicted Plot","Residual Plot","Price Distribution","Feature Relationships","Prediction Error D
        for step in tqdm(visualization_steps, desc="Generating visualizations"):
            if step == "Actual vs Predicted Plot":
                self._plot_actual_vs_predicted(y_test)
            elif step == "Residual Plot":
                self._plot_residuals(y_test)
            elif step == "Price Distribution":
                self._plot_price_distribution()
            elif step == "Feature Relationships":
                self._plot_feature_relationships()
```

```
        elif step == "Prediction Error Distribution":
            self._plot_error_distribution(y_test)

    print("\nAll visualizations have been saved as PNG files.")

def _plot_actual_vs_predicted(self, y_test):
    for model_name, y_pred in self.predictions.items():
        plt.figure(figsize=(8, 6))
        plt.scatter(y_test, y_pred, alpha=0.5)
        plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
        plt.title(f"{model_name}: Actual vs Predicted Prices")
        plt.xlabel("Actual Price ($)")
        plt.ylabel("Predicted Price ($)")
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(f'{model_name}_actual_vs_predicted.png')
        plt.close()

def _plot_residuals(self, y_test):
    for model_name, y_pred in self.predictions.items():
        residuals = y_test - y_pred
        plt.figure(figsize=(8, 6))
        plt.scatter(y_pred, residuals, alpha=0.5)
        plt.axhline(y=0, color='r', linestyle='--')
        plt.title(f"{model_name}: Residual Plot")
        plt.xlabel("Predicted Price ($)")
        plt.ylabel("Residuals")
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(f'{model_name}_residuals.png')
        plt.close()

def _plot_price_distribution(self):
    plt.figure(figsize=(10, 6))
    sns.histplot(self.y, kde=True)
    plt.title("Price Distribution")
    plt.xlabel("Price ($)")
    plt.ylabel("Density")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('price_distribution.png')
    plt.close()

def _plot_feature_relationships(self):
    key_features = ['accommodates', 'bedrooms', 'bathrooms', 'distance_to_manhattan', 'review_scores_rating']
    key_features = [f for f in key_features if f in self.df_processed.columns]
```

```
        if len(key_features) > 0:
            fig, axes = plt.subplots(len(key_features), 1, figsize=(10, 4 * len(key_features)))
            if len(key_features) == 1:
                axes = [axes]

            for i, feature in enumerate(key_features):
                sns.scatterplot(x=self.df_processed[feature], y=self.df_processed['actual_price'], alpha=0.5,
                                ax=axes[i])
                axes[i].set_title(f"Relationship between {feature} and price")
                axes[i].set_ylabel("Price ($)")
                axes[i].grid(True)

            plt.tight_layout()
            plt.savefig('feature_relationships.png')
            plt.close()

    def _plot_error_distribution(self, y_test):
        plt.figure(figsize=(12, 8))

        for model_name, y_pred in self.predictions.items():
            error_pct = 100 * (y_pred - y_test) / y_test
            sns.kdeplot(error_pct, label=model_name)

        plt.title("Prediction Error Distribution (%)")
        plt.xlabel("Percentage Error")
        plt.ylabel("Density")
        plt.legend()
        plt.grid(True)
        plt.axvline(x=0, color='r', linestyle='--')
        plt.xlim(-100, 100)
        plt.tight_layout()
        plt.savefig('error_distribution.png')
        plt.close()
```

In [4]:

```
def main():
    try:
        predictor = HousingPricePredictor('nyc_with_distances.csv')
        predictor.load_data().prepare_features().train_and_evaluate()
        print("\n" + "=" * 50)
        print("Task completed successfully!")
        print("=" * 50)
    except Exception as e:
        print(f"\nError: {str(e)}")
        import traceback
        traceback.print_exc()
```

```
if __name__ == "__main__":
    main()
```

```
==================================================
Loading and Exploring Data
==================================================

Dataset shape: (17853, 59)
Total records: 17853
Total features: 59

Sample data (first 2 rows of key columns):
--------------------------------------------------
        id  log_price property_type        room_type  accommodates  bathrooms  \
0  6304928   5.129899     Apartment  Entire home/apt             7        1.0
1  7919400   4.976734     Apartment  Entire home/apt             5        1.0

   bedrooms  beds   latitude  longitude  review_scores_rating
0       3.0   3.0  40.766115 -73.989040                  93.0
1       1.0   3.0  40.808110 -73.943756                  92.0

==================================================
Feature Engineering and Preprocessing
==================================================

Found Queens features: ['queens_flushing_meadows_corona_park_distance', 'queens_museum_of_the_moving_image_distance', 'queens_queens_botanic
al_garden_distance']

Found Bronx features: ['bronx_new_york_botanical_garden_distance', 'bronx_bronx_zoo_distance', 'bronx_yankee_stadium_distance']

Found Staten Island features: ['staten_island_staten_island_ferry_terminal_distance', 'staten_island_staten_island_zoo_distance', 'staten_is
land_historic_richmond_town_distance', 'staten_island_national_lighthouse_museum_distance', 'staten_island_staten_island_greenbelt_conservan
cy_distance']

Location features being used: ['distance_to_jfk', 'distance_to_lga', 'distance_to_ewr', 'distance_to_manhattan', 'manhattan_times_square_dis
tance', 'manhattan_central_park_distance', 'manhattan_empire_state_building_distance', 'brooklyn_brooklyn_bridge_distance', 'brooklyn_prospe
ct_park_distance', 'latitude', 'longitude', 'queens_flushing_meadows_corona_park_distance', 'queens_museum_of_the_moving_image_distance', 'q
ueens_queens_botanical_garden_distance', 'bronx_new_york_botanical_garden_distance', 'bronx_bronx_zoo_distance', 'bronx_yankee_stadium_dista
nce', 'staten_island_staten_island_ferry_terminal_distance', 'staten_island_staten_island_zoo_distance', 'staten_island_historic_richmond_to
wn_distance', 'staten_island_national_lighthouse_museum_distance', 'staten_island_staten_island_greenbelt_conservancy_distance']

Creating advanced features...

Filtered cancellation policies, keeping only strict, moderate, and flexible:
cancellation_policy
strict      9734
moderate    4783
flexible    3332
Name: count, dtype: int64
```

```
Encoding categorical features...

Handling missing values in numeric features...

Removing outliers...

Final feature matrix shape: (17764, 38)
Target variable shape: (17764,)

Selected feature list:
  - distance_to_jfk
  - distance_to_lga
  - distance_to_ewr
  - distance_to_manhattan
  - manhattan_times_square_distance
  - manhattan_central_park_distance
  - manhattan_empire_state_building_distance
  - brooklyn_brooklyn_bridge_distance
  - brooklyn_prospect_park_distance
  - latitude
  - longitude
  - queens_flushing_meadows_corona_park_distance
  - queens_museum_of_the_moving_image_distance
  - queens_queens_botanical_garden_distance
  - bronx_new_york_botanical_garden_distance
  - bronx_bronx_zoo_distance
  - bronx_yankee_stadium_distance
  - staten_island_staten_island_ferry_terminal_distance
  - staten_island_staten_island_zoo_distance
  - staten_island_historic_richmond_town_distance
  - staten_island_national_lighthouse_museum_distance
  - staten_island_staten_island_greenbelt_conservancy_distance
  - accommodates
  - bathrooms
  - bedrooms
  - beds
  - instant_bookable
  - cleaning_fee
  - amenities_count
  - review_scores_rating
  - number_of_reviews
  - price_per_accommodates
  - beds_per_bedroom
  - manhattan_effect
  - property_type_encoded
```

```
  - room_type_encoded
  - bed_type_encoded
  - cancellation_policy_encoded

==================================================
Summary Statistics
==================================================

Summary of numerical variables:
                         count   mean    std    min    25%    50%    75%  \
accommodates           17764.0   3.01   1.94   1.00   2.00   2.00   4.00
bathrooms              17764.0   1.13   0.39   0.00   1.00   1.00   1.00
bedrooms               17764.0   1.19   0.74   0.00   1.00   1.00   1.00
beds                   17764.0   1.65   1.14   1.00   1.00   1.00   2.00
price_per_accommodates 17764.0  49.91  29.37   2.50  30.00  43.61  61.82
beds_per_bedroom       17764.0   1.30   0.59   0.20   1.00   1.00   1.50
instant_bookable       17764.0   0.30   0.46   0.00   0.00   0.00   1.00
cleaning_fee           17764.0   0.80   0.40   0.00   1.00   1.00   1.00
amenities_count        17764.0  17.49   7.57   0.00  13.00  16.00  21.00
review_scores_rating   17764.0  93.75   7.29  20.00  91.00  95.00  99.00
number_of_reviews      17764.0  28.80  38.90   1.00   4.00  14.00  37.00
manhattan_effect       17764.0   0.49   0.20   0.02   0.33   0.47   0.63
distance_to_jfk        17764.0  18.59   4.30   2.58  15.54  18.93  21.75
distance_to_lga        17764.0  10.45   3.58   0.66   8.42  10.21  12.03
distance_to_ewr        17764.0  20.12   4.19   6.35  17.16  19.32  22.25
distance_to_manhattan  17764.0   8.18   4.68   0.03   4.60   7.58  11.20

                          max
accommodates            16.00
bathrooms                6.00
bedrooms                10.00
beds                    18.00
price_per_accommodates 650.00
beds_per_bedroom        10.00
instant_bookable         1.00
cleaning_fee             1.00
amenities_count         77.00
review_scores_rating   100.00
number_of_reviews      465.00
manhattan_effect         1.00
distance_to_jfk         41.69
distance_to_lga         42.92
distance_to_ewr         38.78
distance_to_manhattan   38.01

Top categories of categorical variables:
```

```
property_type (Top 5 categories):
property_type
Apartment      14513
House           1867
Townhouse        424
Loft             376
Condominium      253
Name: count, dtype: int64

room_type (Top 5 categories):
room_type
Entire home/apt    9011
Private room       8301
Shared room         452
Name: count, dtype: int64

bed_type (Top 5 categories):
bed_type
Real Bed         17249
Futon              209
Pull-out Sofa      177
Airbed              88
Couch               41
Name: count, dtype: int64

==================================================
Model Training and Evaluation
==================================================

Training KNN model...
Starting grid search...
Fitting 5 folds for each of 6 candidates, totalling 30 fits
  - Best k: 10

Training Random Forest Model...
Starting grid search...
Fitting 5 folds for each of 36 candidates, totalling 180 fits
  - max_depth: 20
  - min_samples_leaf: 1
  - min_samples_split: 2
  - n_estimators: 100

Training Gradient Boosting model...
Starting grid search...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
```

```
  - learning_rate: 0.05
  - max_depth: 5
  - min_samples_leaf: 5
  - min_samples_split: 5
  - n_estimators: 300
  - subsample: 0.8

Training LightGBM Model...
Starting grid search...
Fitting 5 folds for each of 8 candidates, totalling 40 fits
  - colsample_bytree: 0.8
  - learning_rate: 0.05
  - max_depth: 10
  - n_estimators: 300
  - num_leaves: 31
  - subsample: 0.8

Training neural network model...
Training the neural network...
Epoch 1/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 25s 17ms/step - loss: 27947.0918
Epoch 2/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 23918.2559
Epoch 3/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 18380.5176
Epoch 4/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 13868.8994
Epoch 5/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 8730.3613
Epoch 6/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 5734.5127
Epoch 7/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 3868.9265
Epoch 8/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 2387.6121
Epoch 9/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 1566.7460
Epoch 10/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 1218.1461
Epoch 11/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 1006.0692
Epoch 12/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 17ms/step - loss: 1094.6256
Epoch 13/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 988.8381
Epoch 14/50
```

```
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 964.4660
Epoch 15/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 994.1875
Epoch 16/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 952.5213
Epoch 17/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 949.3553
Epoch 18/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - loss: 954.6258
Epoch 19/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 997.5087
Epoch 20/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 842.7967
Epoch 21/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 910.0065
Epoch 22/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 918.8831
Epoch 23/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 914.2990
Epoch 24/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 801.5556
Epoch 25/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 920.5190
Epoch 26/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 916.5526
Epoch 27/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 797.3889
Epoch 28/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 824.7533
Epoch 29/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 832.4340
Epoch 30/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 802.9287
Epoch 31/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 823.2221
Epoch 32/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 889.6133
Epoch 33/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - loss: 887.4394
Epoch 34/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 849.0937
Epoch 35/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 854.5332
Epoch 36/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 859.1699
Epoch 37/50
```

```
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 774.1105
Epoch 38/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 818.3398
Epoch 39/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 812.3327
Epoch 40/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 809.2545
Epoch 41/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 780.4910
Epoch 42/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 854.8329
Epoch 43/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 761.1912
Epoch 44/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 893.4928
Epoch 45/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 777.4446
Epoch 46/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 770.8685
Epoch 47/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 751.6914
Epoch 48/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 886.7595
Epoch 49/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - loss: 796.1525
Epoch 50/50
223/223 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 880.3394

==================================================
Model Evaluation Metrics
==================================================
Evaluating models:  20%|████████████                                            | 1/5 [00:01<00:06,  1.52s/it]
KNN Model Evaluation Metrics:
  R² (R-squared): 0.7867
  MSE (Mean Squared Error): 2154.8336
  RMSE (Root Mean Squared Error): 46.4202
  MAE (Mean Absolute Error): 29.2919
  MAPE (Mean Absolute Percentage Error): 24.0285%
Evaluating models:  60%|████████████████████████████████████                    | 3/5 [00:01<00:00,  2.14it/s]
```

```
Random Forest Model Evaluation Metrics:
  R² (R-squared): 0.9901
  MSE (Mean Squared Error): 99.5916
  RMSE (Root Mean Squared Error): 9.9796
  MAE (Mean Absolute Error): 1.4694
  MAPE (Mean Absolute Percentage Error): 0.6540%

Gradient Boosting Model Evaluation Metrics:
  R² (R-squared): 0.9964
  MSE (Mean Squared Error): 36.1768
  RMSE (Root Mean Squared Error): 6.0147
  MAE (Mean Absolute Error): 2.1817
  MAPE (Mean Absolute Percentage Error): 1.3979%

LightGBM Model Evaluation Metrics:
  R² (R-squared): 0.9911
  MSE (Mean Squared Error): 89.6400
  RMSE (Root Mean Squared Error): 9.4678
  MAE (Mean Absolute Error): 3.1712
  MAPE (Mean Absolute Percentage Error): 2.1614%
56/56 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step
Evaluating models: 100%|████████████████████████████████████████████████| 5/5 [00:04<00:00,  1.12it/s]
```

```
ANN Model Evaluation Metrics:
  R² (R-squared): 0.9774
  MSE (Mean Squared Error): 227.9126
  RMSE (Root Mean Squared Error): 15.0968
  MAE (Mean Absolute Error): 9.1230
  MAPE (Mean Absolute Percentage Error): 8.0537%

==================================================
Model Performance Comparison
==================================================

Comparison results of the five evaluation metrics:
                      R²       MSE    RMSE     MAE    MAPE
KNN               0.7867 2154.8336 46.4202 29.2919 24.0285
Random Forest     0.9901   99.5916  9.9796  1.4694  0.6540
Gradient Boosting 0.9964   36.1768  6.0147  2.1817  1.3979
LightGBM          0.9911   89.6400  9.4678  3.1712  2.1614
ANN               0.9774  227.9126 15.0968  9.1230  8.0537

Best Model Analysis:
----------------------------------------
R²: Gradient Boosting (0.9964)
MSE: Gradient Boosting (36.1768)
RMSE: Gradient Boosting (6.0147)
MAE: Random Forest (1.4694)
MAPE: Random Forest (0.6540)

==================================================
Feature Importance Analysis
==================================================

Top 15 Most Important Features:
------------------------------------------------------------
 1. room_type_encoded                   0.3179
 2. price_per_accommodates              0.2961
 3. accommodates                        0.2313
 4. bathrooms                           0.1412
 5. bedrooms                            0.0033
 6. manhattan_empire_state_building_distance 0.0022
 7. longitude                           0.0010
 8. amenities_count                     0.0008
 9. beds                                0.0007
10. distance_to_jfk                     0.0006
11. number_of_reviews                   0.0005
12. distance_to_lga                     0.0005
13. distance_to_ewr                     0.0005
```

```
14. manhattan_times_square_distance     0.0004
15. review_scores_rating                0.0004
Feature importance has been saved to 'feature_importance.csv'
Feature importance plot has been saved to 'feature_importance.png'

==================================================
Creating Visualizations
==================================================
Generating visualizations: 100%|██████████████████████████████████████████████████| 5/5 [00:26<00:00,  5.26s/it]
All visualizations have been saved as PNG files.

Model performance comparison results saved to 'Model_Performance_Comparison.csv'

==================================================
Task completed successfully!
==================================================
```