Data X project assignment

Airbnb nightly price prediction based on dataset for Prague

Team 3

Vojtěch Neumann, Viktor Sívek, Richard Kalousek, Juraj Šlauka, Dominik Nováček

Praha, Duben 2025

1 Data understanding

This chapter aims to provide an initial understanding of the dataset used to predict nightly Airbnb rental prices in Prague. Exploring the structure, types of variables, and the quality of the data is essential before performing any preprocessing or modelling. We begin by performing a general inspection of the dataset.

The exploratory data analysis was conducted in Python using common data science libraries such as pandas, numpy, seaborn, and missingno for data inspection and visualization.

1.1 Initial Inspection

Dataset Overview

- **Number of rows (listings):** 10,108
- **Number of columns (features):** 79
- The dataset includes various types of information such as listing identifiers, host details, property descriptions, availability calendars, reviews, location coordinates, and pricing.

Data Types Summary

- **Integer (int64):** 24 columns
- **Float (float64):** 21 columns
- **Object (string-like):** 34 columns

The presence of 34 object columns indicates that the dataset includes a substantial amount of categorical or textual data (e.g., room_type, property_type, host_name, description), which may need conversion or encoding before modeling.

Missing Data Overview

Several columns contain missing values. Examples include:

- description, neighborhood_overview, host_location, bathrooms, beds, review_scores_rating, and other review-related fields.
- Columns such as calendar_updated, license, and neighbourhood_group_cleansed are completely empty and will be removed in preprocessing.

Initial Observations

- Variables such as price are stored as strings and require conversion to numerical format for analysis.
- Review scores and other derived features are frequently missing for listings with no reviews.

- The dataset contains features with a variety of scales, units, and formats, highlighting the need for normalization or transformation.

This initial inspection helps highlight critical data quality issues and sets the stage for further data cleaning, feature engineering, and modeling.

1.2 Target Variable Analysis – price

The price column is the target variable in our predictive modeling task. Understanding its structure, cleaning it appropriately, and evaluating its distribution are critical for model reliability and accuracy.

Data Type and Cleaning

Initially, the price column was stored as an object, representing monetary values as strings with currency symbols and commas (e.g., "$1,542.00"). This required cleaning before numerical analysis.

- All price entries were successfully converted to numeric format.
- Since all non-null values were whole numbers, the data was safely stored as a nullable Int64 type in a new column price_cleaned.

Missing Values

Out of 10,108 total listings, **1,300 entries (12.86%)** had missing price information. These were either dropped or handled appropriately to prevent bias or data leakage during model training.

Descriptive Statistics

The cleaned price_cleaned column had the following characteristics:

- **Count:** 8,808
- **Mean:** 2,730 CZK
- **Median:** 1,762 CZK
- **Standard Deviation:** 8,349 CZK
- **Minimum:** 223 CZK
- **Maximum:** 251,025 CZK

The large standard deviation and high maximum value already suggest substantial skewness and presence of extreme outliers.

Price Distribution (Cleaned)

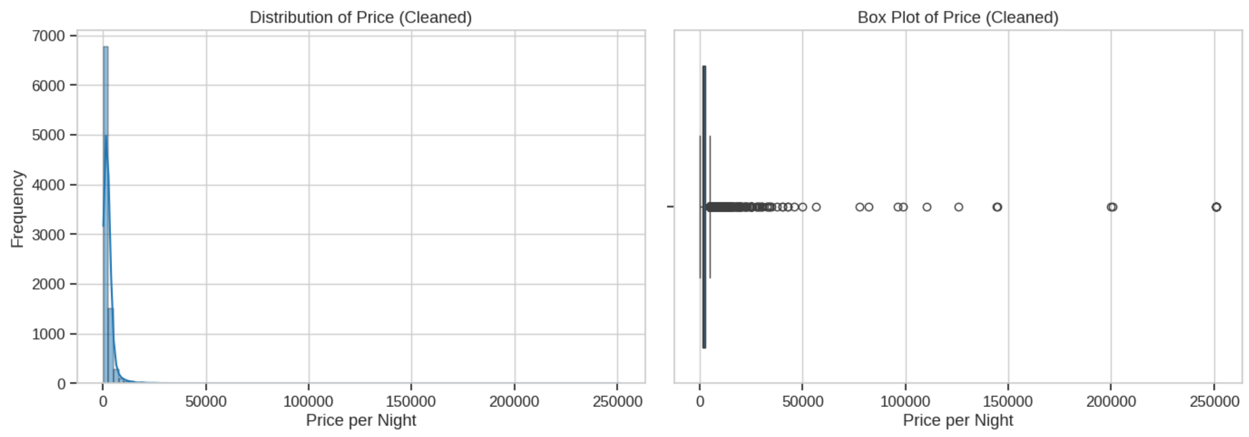


Figure 1: Boxplot and distribution of price

The histogram and box plot confirm a **heavily right-skewed** distribution. Most listings cluster around lower prices, while a small number of listings are priced extremely high (up to ~250,000 CZK), obscuring the majority in visualizations.

- **Skewness (raw price):** 23.42 → highly skewed
- Indicates potential problems for modeling algorithms sensitive to distribution shape.

Log-Transformed Price

To reduce skewness and improve model behavior, we applied a log transformation using log1p (log(1 + price)), which is robust to zero or missing values.

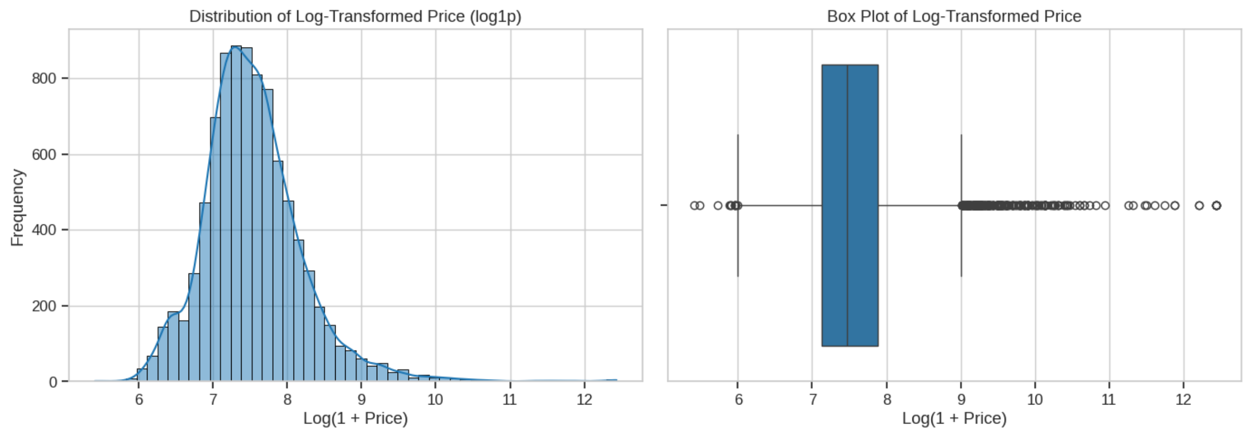


Figure 2: Boxplot and distribution of price (log)

- **Skewness (log-transformed):** 1.21
- The transformation significantly improved symmetry and spread.
- Outliers remain visible on the log scale, but the distribution is now more suitable for regression modeling.

Conclusion

Due to its better distributional properties, the log-transformed log_price will be used as the target variable in the upcoming modeling phase.

1.3 Missing Values Analysis

Understanding the extent and structure of missing values is critical before beginning any data preparation or modeling. This section examines both the quantity and the patterns of missing data across the dataset.

Overview of Missingness

Out of 79 total columns, **41 columns** contain missing values. Key statistics include:

- **100% missing:**
    - neighbourhood_group_cleansed, calendar_updated, license
      These columns provide no usable information and will be dropped.
- **High missingness (>50%):**
    - neighbourhood, neighborhood_overview
      These text-heavy columns will be considered for removal due to redundancy and low utility.
- **Moderate missingness:**
    - host_about (~39%), host_location (~23%)
    - beds (~12.4%), bathrooms (~12.3%), price / log_price (~12.9%)
      These will require imputation or exclusion based on further analysis.
- **Grouped missingness (~9.65%):**
  Many review-related fields are consistently missing together, suggesting they belong to listings with no reviews.

Missing Value Correlation Heatmap

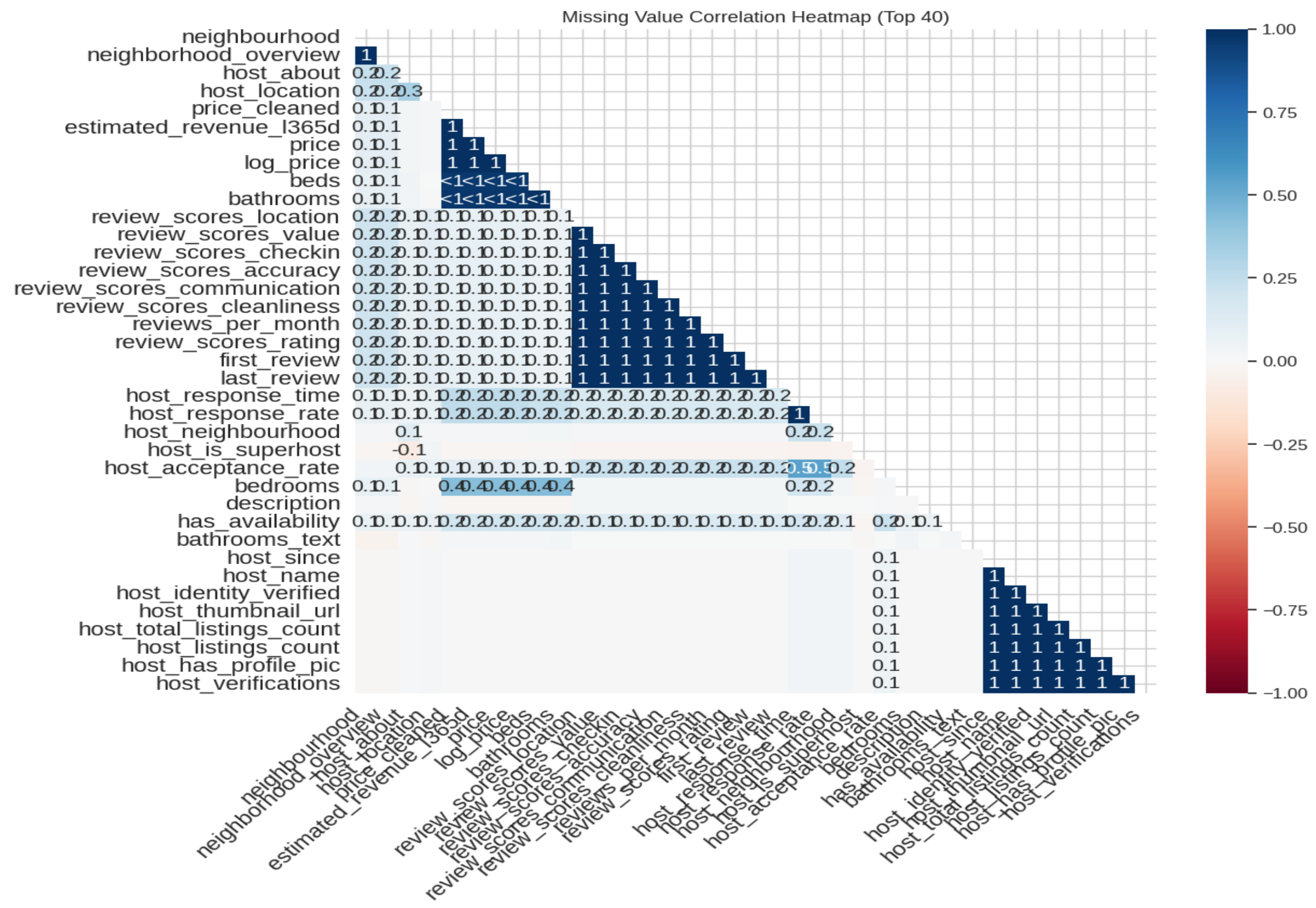


Figure 3: Missing value heatmap

The heatmap shows high correlations of missingness among the following groups:

- **Review block:**
  review_scores_*, first_review, last_review, reviews_per_month
  → Strong co-occurrence suggests missingness is structurally meaningful (i.e., no reviews exist).

- **Host-related attributes:**
  host_response_time, host_response_rate, host_acceptance_rate
  → Correlated but with weaker links.

- **Bedroom/bathroom information:**
  Moderate association, especially between bathrooms and beds.

Planned Actions for Data Preparation

Based on the above analysis, the following actions are proposed for the preprocessing phase:

1. **Drop Columns**
   - 100% missing: neighbourhood_group_cleansed, calendar_updated, license
   - High text-based missingness: neighbourhood, neighborhood_overview
   - Consider: host_about, host_location, bathrooms (will use bathrooms_text instead)
2. **Drop Rows**
   - Rows with missing target variable (price_cleaned / log_price)
     → Cannot be used for training or evaluation (~12.9% of listings)
3. **Impute Strategically**
   - **Reviews block:** Impute reviews_per_month with 0; consider mean/median for scores or add a has_reviews flag
   - **Beds, bedrooms:** Median imputation, possibly grouped by property_type
   - **Host response:** Use 'Missing' category for response time; mean/median for response rate
   - **Other host info:** Use mode or median
   - **Low missingness text (e.g., description)**: Impute with empty string
   - **Single missing values**: Drop or impute with mode

This structured approach allows us to retain as much useful information as possible while preparing a clean and model-ready dataset.

1.4 Data Type Examination

While the initial .info() summary helped us understand the general structure of the dataset, a deeper inspection of object-type columns reveals that many contain structured data stored as text (e.g., dates, booleans, lists, percentages). Identifying and converting these is crucial for proper modeling and analysis.

Identified Conversions and Parsing Needs

Using a combination of heuristics and known column semantics, we identified the following categories:

Detected Type	***Columns***
Dates	calendar_last_scraped, first_review, host_since, last_review, last_scraped
Booleans (t/f)	has_availability, host_has_profile_pic, host_identity_verified, host_is_superhost, instant_bookable

Percentages	host_acceptance_rate, host_response_rate
List-like strings	amenities, host_verifications
Specific text to parse	bathrooms_text (to extract number of bathrooms and types, e.g., shared/private)

Figure 4: feature table with datatypes

These columns will be **parsed, converted, or transformed** into structured formats in the Data Preparation phase.

Columns to Drop

Based on missingness, redundancy, or lack of modeling value, the following object-type columns are flagged for removal:

- **URLs** (metadata only):
  listing_url, host_url, host_thumbnail_url, host_picture_url, picture_url
- **Redundant or uninformative text fields:**
  name, host_name, description, host_about, host_location, neighborhood_overview, neighbourhood, host_neighbourhood, source
- **Other identifiers or non-predictive columns:**
  id, scrape_id, host_id, bathrooms_text

Object Columns to Treat as Categorical

The following columns are retained for modeling but will be encoded (e.g., one-hot or target encoding):

- host_response_time (low cardinality, e.g., "within an hour", "a few days")
- neighbourhood_cleansed (potentially high cardinality; grouping or target encoding may be needed)
- property_type (moderate/high cardinality; may be grouped)
- room_type (low cardinality; suitable for one-hot encoding)

This systematic review ensures that the data types are correctly interpreted and appropriately transformed during the **Data Preparation** phase. Converting these hidden formats improves model interpretability and ensures that downstream algorithms can process them effectively.

1.5 Unique Values and Cardinality Analysis

Analyzing the number of unique values per column helps determine which variables are constant, which are suitable for categorical encoding, and which may require special handling due to high cardinality.

Constant Columns

We identified **6 columns** that have only one unique non-null value or are entirely missing:

- calendar_last_scraped, calendar_updated, last_scraped, license, neighbourhood_group_cleansed, scrape_id

These columns carry **no predictive value** and will be **dropped** during the Data Preparation phase.

Low-Cardinality Columns

Twelve columns have **2 to 20 unique values** and are suitable for **One-Hot Encoding**, including:

- **Booleans and flags:**
  instant_bookable, host_is_superhost, has_availability, host_identity_verified, host_has_profile_pic
- **Categoricals:**
  room_type, host_response_time, host_verifications
- **Small numeric categories:**
  accommodates, bedrooms, calculated_host_listings_count_shared_rooms

Although accommodates and bedrooms are technically numerical, their low number of unique values (16 each) could allow for grouping or binning in future modeling.

High-Cardinality Columns

A total of **56 columns** have more than 50 unique values. These include:

- **Identifiers / Metadata:**
  id, listing_url, name, description, picture_url, host_id, host_url, host_name, host_about, host_thumbnail_url, host_picture_url
  → These will be **dropped** due to high cardinality and lack of predictive relevance.
- **Free-text and semi-structured columns:**
  amenities, host_verifications, neighborhood_overview
  → These require **parsing or feature extraction**, not direct encoding.
- **High cardinality categoricals:**
  - neighbourhood_cleansed, property_type, host_neighbourhood, host_location
    → These will be handled using **target encoding, frequency encoding, grouping**, or **CatBoost-style embeddings**.
- **Quantitative or date-based variables with many unique values:**
  - estimated_revenue_l365d, reviews_per_month, number_of_reviews, first_review, last_review, etc.
    → These will be treated as **continuous features** or engineered into time-based features (e.g., time since last review).

Summary of Planned Actions

ACTION	AFFECTED COLUMNS (EXAMPLES)
DROP CONSTANT COLUMNS	calendar_last_scraped, license, scrape_id, etc.
DROP IDS/URLS/TEXT	id, listing_url, host_id, description, name, etc.
ONE-HOT ENCODE	instant_bookable, room_type, host_is_superhost, host_response_time
TARGET/FREQUENCY ENCODE	neighbourhood_cleansed, property_type, host_neighbourhood
PARSE LIST-LIKE STRINGS	amenities, host_verifications → Create new features (e.g., num_amenities, flags)
TREAT AS CONTINUOUS	number_of_reviews, estimated_revenue_l365d, reviews_per_month, log_price, etc.

This cardinality-based review ensures that each column is treated according to its structure and statistical properties, optimizing both model performance and computational efficiency.

1.6 Outlier Detection

Outliers are extreme values that deviate significantly from the rest of the data and may distort model performance, especially for linear models. This section provides a preliminary exploration of outliers in key numeric columns through both descriptive statistics and visualizations.

Descriptive Statistics

Descriptive summaries of numeric columns revealed several features with suspiciously large maximum values compared to their 99th percentiles. Key findings include:

- **price_cleaned**:
    - Mean: 2,731 CZK
    - Median: 1,762 CZK
    - Max: 251,025 CZK
      → Extreme high-end values strongly skew the distribution.
      → A **log transformation** (see log_price) reduces skew but does not completely remove outliers.
- **minimum_nights** and **maximum_nights**:
    - Max values (730 and 3,333 respectively) far exceed typical ranges.
    - 99th percentile for minimum_nights: 30
      → Likely candidates for **capping or thresholding**.

- **accommodates, bedrooms, beds**:
  - High max values (up to 34 bedrooms or 32 beds) may reflect rare large properties.
  - Min values of 0 could represent data quality issues (e.g., studios missing proper entries).
    → May require **imputation or filtering**.
- **number_of_reviews, host_listings_count**:
  - Max values up to 1,922 and 3,315 respectively.
    → Could represent **popular listings** or **professional hosts** rather than errors.
    → Consider **log transformation or quantile-based binning**.

Box Plot Visualizations

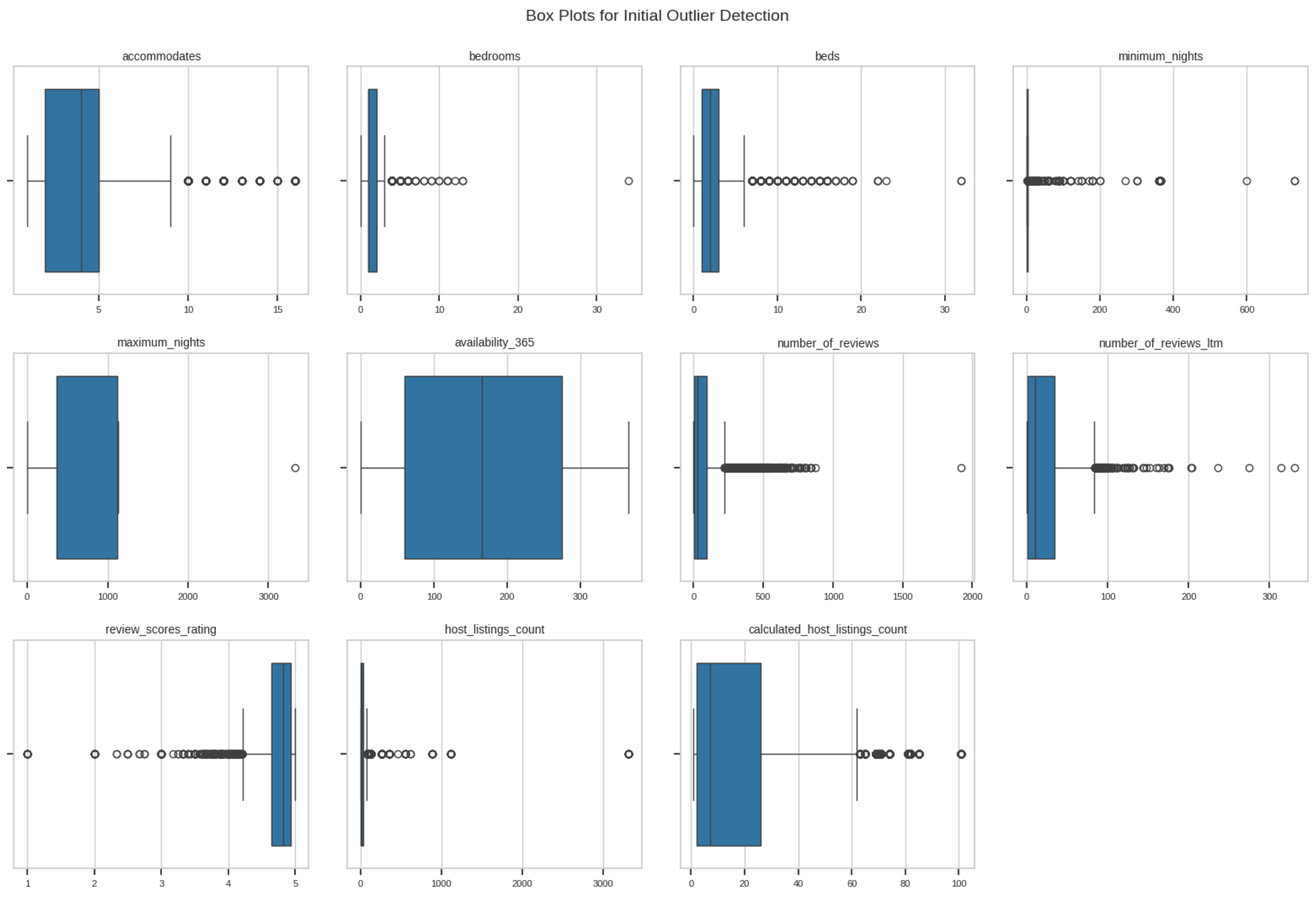


Figure 5: outlier boxplot

Box plots for 12 selected numeric columns confirm the presence of outliers, particularly:

- minimum_nights, maximum_nights: Strong right-side outliers.

- bedrooms, beds: Some extreme values (e.g., >10 bedrooms), as well as possible under-reporting (0 values).
- number_of_reviews, host_listings_count: Skewed distributions with long right tails.
- review_scores_rating: A few very low values, though likely real and reflecting poor guest experiences.

Planned Actions for Data Preparation

Feature	***Planned Handling***
price_cleaned	Use log-transformed version (log_price) for modeling; review remaining extreme outliers if needed
minimum_nights	Cap values at a reasonable threshold (e.g., 90 or 99th percentile)
maximum_nights	Cap or bin values above practical limits (e.g., >1125)
bedrooms, beds	Investigate zero values; impute based on similar listings or property_type
host_listings_count	Cap at 99th percentile or use log transform; flag extreme super-hosts
number_of_reviews	Consider log transformation; retain extreme but valid values
review_scores_rating	No action unless values are clearly erroneous

In the next stage, these findings will guide outlier handling during Data Preparation. The approach will prioritize preserving valid but rare observations, particularly those representing luxury listings or experienced hosts, while mitigating extreme distortions through capping or transformations.

1.7 Initial Relationship Visualization

To gain a better understanding of the relationship between the target variable (log_price) and key features, we explored several visualizations. This helps identify **predictive strength**, **variable importance**, and **data structure**.

Price vs. Room Type

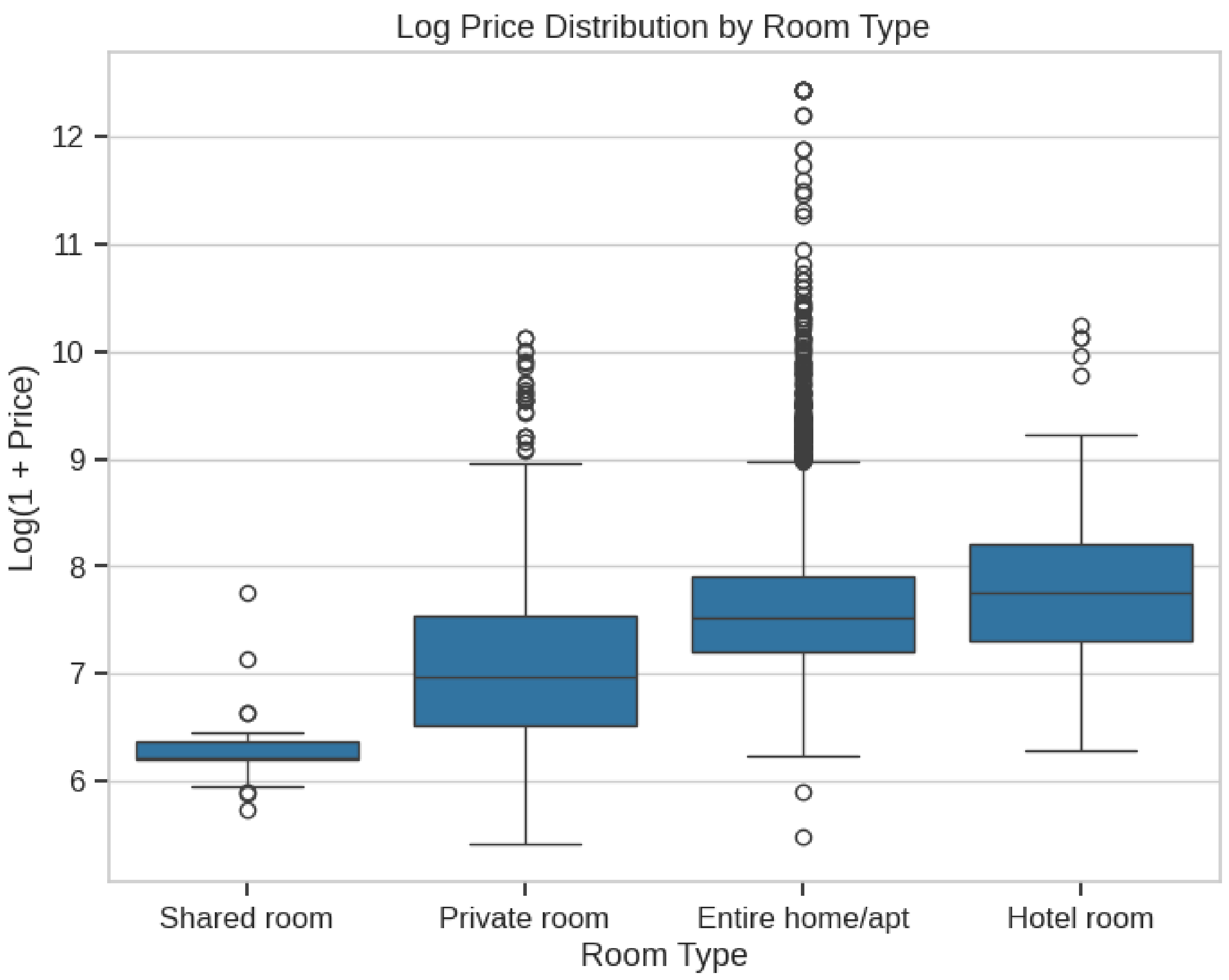


Figure 6: price distributiuon by room type

The box plot shows distinct price distributions by room type:

- **Entire home/apt** has the highest **median price** and the widest price spread.
- **Hotel room** also shows relatively high prices.
- **Private room** has a lower median price and less variability.
- **Shared room** is the least expensive, with a narrow, consistent distribution.

Interpretation:
room_type is clearly a **strong categorical predictor** of price. It should be retained and one-hot encoded for modeling.

Price vs. Neighbourhood

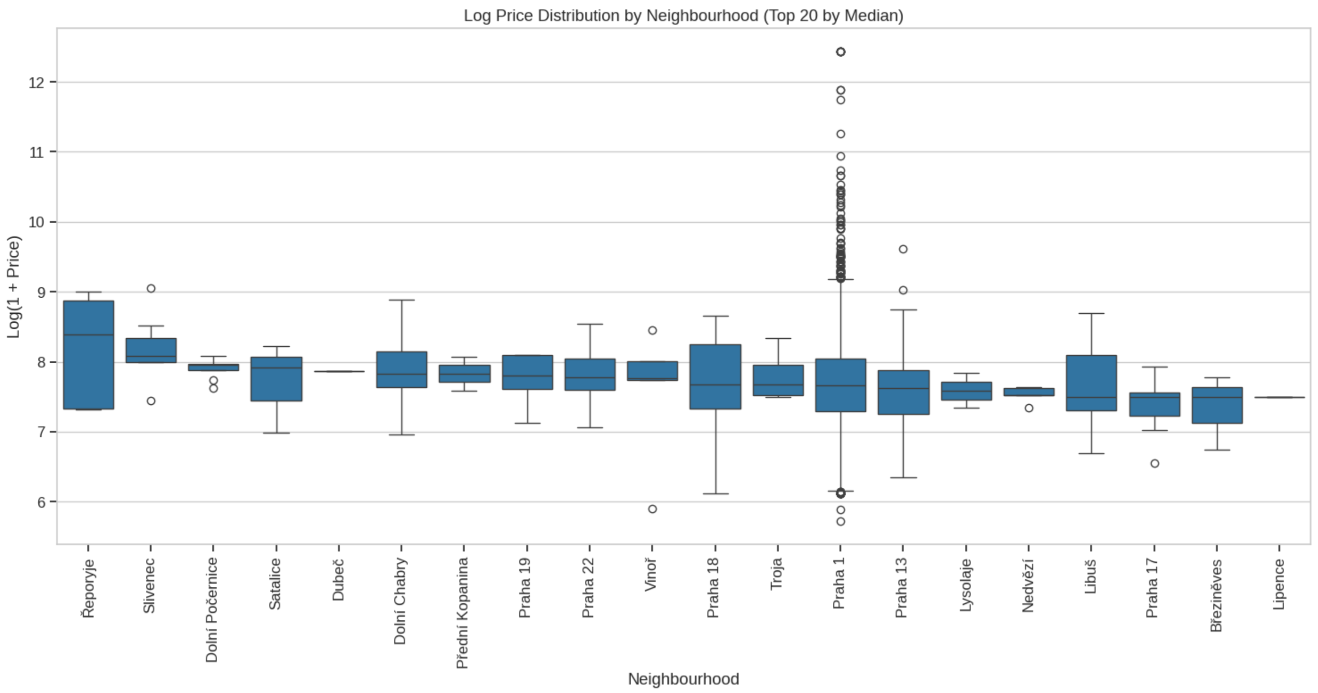


Figure 7: price distribution by neighbourhood

This box plot displays log_price distributions across the **top 20 Prague neighbourhoods** by median price:

- Neighbourhoods like **Řeporyje**, **Slivenec**, and **Praha 1** have notably higher median prices.
- Some areas (e.g., **Praha 1**) show high price variance and numerous luxury listings.
- Others (e.g., **Březiněves**, **Lipence**) have tighter distributions and lower prices.

Interpretation:
neighbourhood_cleansed has strong **location-specific price differences**, confirming its importance. However, its **high cardinality** requires careful handling (e.g., **target encoding**, **grouping**, or tree-based models).

Price vs. Size Metrics

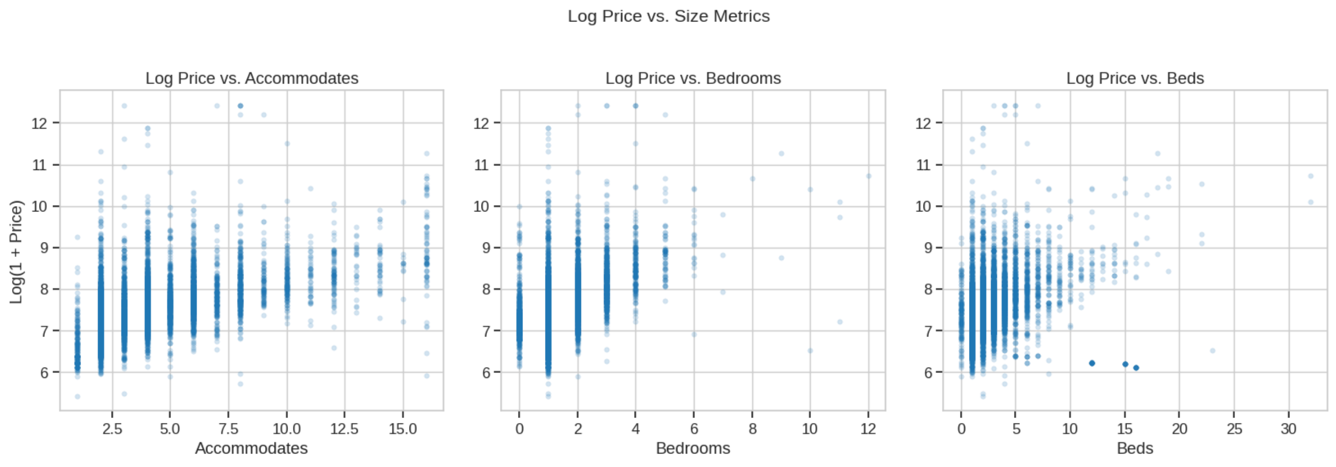


Figure 8: prize distribution by size metrics

These scatter plots compare log_price to **accommodates**, **bedrooms**, and **beds**:

- A **positive trend** is observed across all three metrics: larger listings tend to cost more.
- The relationship is not strictly linear; vertical spread at each x-value suggests **price is influenced by other features** beyond size.
- The trend is most visible for **bedrooms**, slightly less so for **beds** and **accommodates**.
- Notably, listings with bedrooms = 0 still span a wide range of prices, likely reflecting **studio apartments**.

Interpretation:
These size-related features are important predictors, but the presence of outliers and variance suggests **non-linear modeling techniques** (e.g., Random Forests, XGBoost) may be more appropriate.

Summary of Insights

FEATURE	**INSIGHT**
ROOM_TYPE	Strong categorical predictor; significant differences in price distribution
NEIGHBOURHOOD_CLEANSED	Strong location-based variation; high cardinality requires thoughtful encoding

ACCOMMODATES, BEDROOMS, BEDS	Positively correlated with price; non-linear patterns with variability

Next Steps in Data Preparation

- Ensure these features are **cleaned and imputed** where necessary.
- Encode categorical features:
    - room_type: One-hot encoding
    - neighbourhood_cleansed: Target encoding or grouped categories
- Consider **feature interactions** (e.g., room type × size or neighbourhood × size).
- Explore **non-linear models** or models robust to interactions and outliers.

1.8 Data Understanding Summary

This initial exploration of the Prague Airbnb dataset (snapshot from March 16, 2025) provides a solid foundation for subsequent modeling. Key findings and corresponding implications for the upcoming **Data Preparation** phase are summarized below:

1. 8.1 Dataset Overview

- The dataset contains **10,108 listings** described by **79 features**.
- Features span **multiple data types** (int64, float64, object) and will require extensive cleaning, type conversion, and transformation.

1.8.2 Target Variable: price

- Originally stored as **text (object)**, the price column was cleaned and successfully converted to a **nullable integer** (Int64) as price_cleaned.
- The distribution of price is **extremely right-skewed** (skewness ≈ 23.4).
- A **log transformation** was applied: log_price = log1p(price_cleaned) resulting in a more **symmetrical distribution** (skewness ≈ 1.21), suitable for modeling.
- **~12.9% of listings** are missing price data and will need to be **dropped** during Data Preparation.

1.8.3 Missing Data

- **Fully missing columns** (neighbourhood_group_cleansed, calendar_updated, license) will be dropped.
- **High-missing text columns** like neighbourhood, neighborhood_overview, host_about, and host_location (>20–50%) will be considered for removal to reduce noise.

- **Review-related columns** (e.g., review_scores_*, first_review, last_review, reviews_per_month, ~9.65%) and **host response fields** (e.g., host_response_time, ~9.2%) show **correlated missingness**, likely tied to listings without reviews.
- Features like beds (~12.4%) and bedrooms (~2.7%) also contain missing data.
- **Action**: Imputation strategies will include:
    - **Zeros** for reviews_per_month
    - **Medians** for beds and bedrooms
    - **"Missing" category** for host_response_time
    - Optionally adding a **has_reviews** indicator feature

1.8.4 Data Types & Formatting

- Several object columns require conversion or parsing:
    - **Date columns** (last_scraped, host_since, etc.) → datetime
    - **Boolean fields** (t/f) → 1/0
    - **Percentage strings** → cleaned and converted to float (e.g., host_response_rate)
    - **List-like strings** (amenities, host_verifications) → parsed to extract counts/flags
    - **Text fields** like bathrooms_text → parsed into numeric indicators

1.8.5 Cardinality & Feature Characteristics

- **Constant columns** (calendar_last_scraped, scrape_id, etc.) and **IDs/URLs** will be **dropped**.
- **High-cardinality identifiers/text** (e.g., id, host_name, description) provide no predictive value or will be optionally engineered later.
- **Low-cardinality categoricals** (room_type, instant_bookable) will be **one-hot encoded**.
- **High-cardinality categoricals** (neighbourhood_cleansed, property_type) will require **target encoding**, **grouping**, or model-specific handling.
- **Numeric features** (coordinates, counts, review scores) will undergo **imputation and scaling** as needed.

1.8.6 Outliers

- **Extreme outliers** found in:
    - minimum_nights (up to 730)
    - maximum_nights (up to 3333)
- **Action**: Apply **capping** at reasonable percentiles (e.g., 99th) or domain-specific limits.

- Other features (e.g., price, number_of_reviews, beds, host_listings_count) exhibit **right-skewed outliers**.
    - For price, the **log transformation** will be relied upon.
    - For others, consider **capping** or leveraging **robust tree-based models**.

1.8.7 Initial Feature Relationships

- Strong categorical differences in log_price by room_type and neighbourhood_cleansed.
- Positive, non-linear trends with listing **size metrics** (accommodates, bedrooms, beds).
- Variance within categories suggests the presence of **interactions** (e.g., room type × size, location × size).

2. Data Preparation

2.1 Setup and Target Definition

The dataset `listings.csv`, containing 10,108 Airbnb listings in Prague and 79 features, was successfully loaded. For the data preparation phase, the dataset was stored in a new DataFrame (`df_prep`) to separate it from the exploration phase.

The original `price` column was stored as a string and required cleaning. Currency symbols and commas were removed, and the column was converted to a numeric format (`price_cleaned`). All non-null values were found to be integers and were stored as nullable integers (Int64).

To address the strong right skew in the price distribution, a log transformation was applied: `log_price = log1p(price_cleaned)`

This transformed variable, `log_price`, was defined as the target for the regression model. The transformation improved the distribution's symmetry and reduced the impact of outliers.

Approximately 1,300 listings (12.9%) had missing prices and thus missing values in `log_price`. These rows will be dropped before modeling.

2.2 Initial Cleaning and Filtering

Removal of rows with missing target variable:
All 1,300 listings with missing log_price were removed, reducing the dataset from 10,108 to **8,808 rows**. This ensures all remaining data points have a valid target for supervised learning.

Column filtering based on EDA findings:
A total of **48 columns** were dropped. These included:

- Constant or empty columns (e.g., license, calendar_updated)
- Redundant columns or replaced variants (e.g., price, bathrooms_text)
- Identifiers and URLs (e.g., id, listing_url)

- High-missing or unused text fields (e.g., host_about, neighbourhood)
- Derived or leaky features (e.g., estimated_revenue_l365d)

The resulting dataset was simplified to **33 columns**.

Removal of rows with missing values in key features:
To avoid imputation for certain essential features, rows missing any of the following were removed: host_since, host_verifications, host_identity_verified, bathrooms, bedrooms, beds, has_availability.

This step removed an additional **40 rows**, resulting in a final dataset size of **8,768 listings**. These columns are now complete.

2.3 Data Type Conversion and Cleaning

To prepare the data for analysis and modeling, key columns were converted from raw string/object format into appropriate numeric or datetime types:

Boolean conversion ('t'/'f' → 1/0):
The columns host_identity_verified, has_availability, and instant_bookable were originally stored as strings 't' and 'f'. They were converted to integer values (1 = true, 0 = false).
No missing values remained in these columns after conversion.

Percentage conversion:
The column host_acceptance_rate, originally in string format with a % symbol (e.g., "100%"), was cleaned and converted to float values between 0.0 and 1.0.
The column host_response_rate was dropped earlier during initial cleaning.

Datetime conversion:
The columns host_since, first_review, and last_review were converted to datetime format. Missing values were preserved as NaT (Not a Time). These features will be used for further date-based engineering.

As a result, the dataset now has properly formatted data types across key features, ready for feature engineering and modeling.

2.4 Feature Engineering & Parsing

In this phase, several raw features were transformed into more meaningful, structured variables:

Amenities:
The amenities column (string-formatted list) was parsed to extract:

- num_amenities: total number of listed amenities.
- Binary indicators for key amenities such as: wifi, kitchen, air_conditioning, heating, washer, dryer, tv, parking, pool, pets_allowed, and long_term_stays_allowed.

The original column was dropped after feature extraction.

Host Verifications:
The host_verifications column was parsed to create a numeric feature:

- num_host_verifications: number of verification methods listed for the host.
  The original column was dropped.

Date Features:
New features were engineered from host_since and last_review using the approximate data scrape date (March 16, 2025):

- host_duration_days: number of days since the host joined the platform.
- days_since_last_review: number of days since the most recent review. Missing values (listings with no reviews) remain and will be imputed later.

The original date columns used for these features were dropped.

These transformations converted unstructured or raw data into clean, model-friendly numerical variables, increasing the predictive value of the dataset.

2.5 Categorical Feature Refinement

To prepare categorical variables for encoding, several refinements were applied:

host_response_time:
Mapped original strings to shorter, standardized values:

- 'within an hour' → within_hour, 'within a few hours' → within_hours, etc.
  Missing values were replaced with 'Unknown'.

room_type:
Standardized formatting by replacing spaces with underscores (e.g., 'Private room' → Private_room). No grouping was needed.

property_type:
Originally contained over 50 unique values. Rare categories with fewer than 50 listings were grouped into a new 'Other_Property' category.
This reduced cardinality to 12 categories and simplified encoding.

neighbourhood_cleansed:
Originally included 51 unique districts. These were grouped based on geographic proximity and housing similarity into:

- Old_Town_Center, New_Town_Vinohrady, Near_Center_East, Near_Center_West_South, North_West_Districts, and Outer_Districts.

A new column neighbourhood_group was created, and the original neighbourhood_cleansed column was dropped to reduce complexity.

These refinements ensure that categorical variables are clean, consistent, and ready for one-hot encoding or alternative encoding techniques in the next phase.

2.6 Handling Missing Values

In this final step of feature engineering, all remaining missing values were addressed systematically, depending on their origin and feature type. The primary sources of missingness were review-related columns and a few metadata fields.

Review-related columns:

- A new binary feature has_reviews was introduced, indicating whether a listing has received any reviews (number_of_reviews > 0).
- Missing values in reviews_per_month were imputed with 0, representing the absence of reviews.
- Review score columns (review_scores_rating, review_scores_location, review_scores_value) were imputed with the median values calculated only from listings that had received reviews. This approach ensures imputations reflect realistic values within the reviewed population.
- The days_since_last_review column was imputed with a large constant (9999) to distinguish listings with no review history or extremely old reviews.

Other columns:

- The only remaining missing values after review-specific imputation were in the host_acceptance_rate column. These were filled with the median value (1.0).
- No other columns (categorical or boolean) required imputation at this stage, as they either had no missing values or had already been handled in earlier steps.

After applying these strategies, a final check confirmed that **no missing values** remained in the dataset. The dataset was then saved in its current form before continuing with outlier handling and encoding.

2.6 Handling Outliers & Skewness

This phase focused on reducing the influence of extreme values and high skewness in numeric features. These transformations aimed to stabilize model learning and improve predictive accuracy, especially for algorithms sensitive to distributional irregularities.

Identification of Skewness and Outliers:

- Numeric features were evaluated for skewness (threshold > 1.0) and for extreme outliers (defined as values exceeding 5× the 99th percentile).
- Thirteen features were found to be highly skewed, including variables related to listing size (e.g. bathrooms, beds, accommodates), host activity (number_of_reviews, reviews_per_month, host_acceptance_rate), and availability.

- Among these, three features (minimum_nights, calculated_host_listings_count_shared_rooms, and reviews_per_month) exhibited extreme maximum values and were flagged for transformation.

Capping of Stay Duration Variables:

- The minimum_nights and maximum_nights variables were capped at their 99th percentiles (30 and 1125 nights respectively).
- This reduced the influence of unrealistic or rare values (e.g., 730 nights or more) while preserving the structure of typical listings.

Log Transformations for Skewed Features:

- A log1p transformation was applied to twelve features with high skewness and non-negative values.
- New columns were created with a _log suffix (e.g., reviews_per_month_log, number_of_reviews_log).
- These transformed features exhibited compressed ranges and improved distributional symmetry.

Verification of Skewness Reduction:

- Skewness was recalculated after capping and log transformation.
- Most transformed features showed substantial improvements, with skewness values closer to 0.
- However, a few features (e.g., calculated_host_listings_count_shared_rooms_log, minimum_nights, host_acceptance_rate_log) remained highly skewed due to structural properties or zero-inflation.

No additional transformations were applied to the remaining skewed features. Since tree-based models (e.g., Random Forest, XGBoost) are robust to monotonic transformations and skewed data, the feature set was considered sufficiently prepared for modeling. The dataset was saved in its transformed state, ready for encoding and model training.

2.7 Encoding Categorical Features

This phase focused on transforming all categorical variables into a numeric format suitable for machine learning algorithms. Two main strategies were applied based on the cardinality (number of unique values) of the categorical features: **One-Hot Encoding** for low-cardinality features and **Frequency Encoding** for high-cardinality features.

Identification of Categorical Features:

- Four columns were identified as categorical: host_response_time, room_type, neighbourhood_group, and property_type.

- The first three columns had low cardinality (≤6 unique values), while property_type had 12 unique categories and was considered moderately high-cardinality.

One-Hot Encoding (Low Cardinality):

- One-hot encoding was applied to host_response_time, room_type, and neighbourhood_group.
- This process created 15 new binary columns (dummy variables) representing each category.
- The original categorical columns were replaced by their encoded counterparts, increasing the total feature count from 56 to 68.
- This encoding is effective for tree-based models, as it allows clear representation of distinct categories without implying any ordinal relationship.

Frequency Encoding (High Cardinality):

- The property_type column was encoded using frequency encoding.
- A new column property_type_freq was created, containing the relative frequency (proportion) of each category in the dataset.
- This approach avoids the dimensionality explosion that would result from one-hot encoding a feature with many unique values.
- The original property_type column was retained for potential comparison during modeling or feature selection.

Final Output:

- All categorical variables are now represented numerically.
- The dataset is fully numeric and ready for scaling or direct model training.
- The encoding strategies were chosen with respect to model compatibility (especially tree-based models) and dimensionality constraints.

2.8 Final Review & Splitting

Before model training, a final review of the processed dataset was conducted to ensure all features are numeric and well-structured. This phase included checking for residual non-numeric columns, removing constant features, unifying data types, saving the pre-split dataset, and performing a reproducible train-test split.

First, the target variable log_price was separated from the feature matrix X. One residual object-type column (property_type) was still present and was removed, as its frequency-encoded version had already been created and retained. One constant column (has_availability) with zero variance was also dropped.

After these removals, the final feature set contained **66 numeric predictors**, all of which were verified to be of standard int64 or float64 types. Integer-like columns (including binary dummies) were unified to int64 after confirming no missing values remained. All float-like columns, including the target, were verified or converted to float64.

The complete cleaned dataset (X + y) was saved in both. parquet and .csv formats for reuse and external inspection.

Next, the dataset was split into training and testing subsets using an 80/20 split with random_state=42 to ensure reproducibility. This yielded:

- **Training set:** 7,014 observations
- **Test set:** 1,754 observations

The split datasets (X_train, X_test, y_train, y_test) maintain the same feature structure and are ready for feature scaling and modeling.

2.9 Feature Scaling & Saving

To ensure that all features are on a similar scale, especially important for models sensitive to magnitude (e.g., linear regression, SVM, KNN, neural networks), we applied standardization using StandardScaler. The scaler was **fit exclusively on the training data (X_train)** to avoid information leakage and then used to transform both X_train and X_test.

As expected, the transformed datasets X_train_scaled and X_test_scaled retained their original shape and feature structure. A descriptive statistics check confirmed that all standardized features now have approximately zero mean and unit variance, indicating successful scaling.

To support consistent inference and potential deployment, the fitted scaler object was saved to the /model directory using joblib. Additionally, the final list of feature names was stored to ensure alignment between training and inference time.

Finally, all four core datasets for modeling were saved as Parquet files in the /data/processed/ directory:

- X_train_scaled.parquet
- X_test_scaled.parquet
- y_train.parquet
- y_test.parquet

These files represent the fully prepared and standardized data, ready for training and evaluating predictive models.

3. Data visualization

This chapter provides visual exploration of key features in relation to the target variable (log_price). The goal is to better understand individual feature distributions and their effect on pricing behavior.

3.1 Feature Visualization

Log Price vs. Accommodates

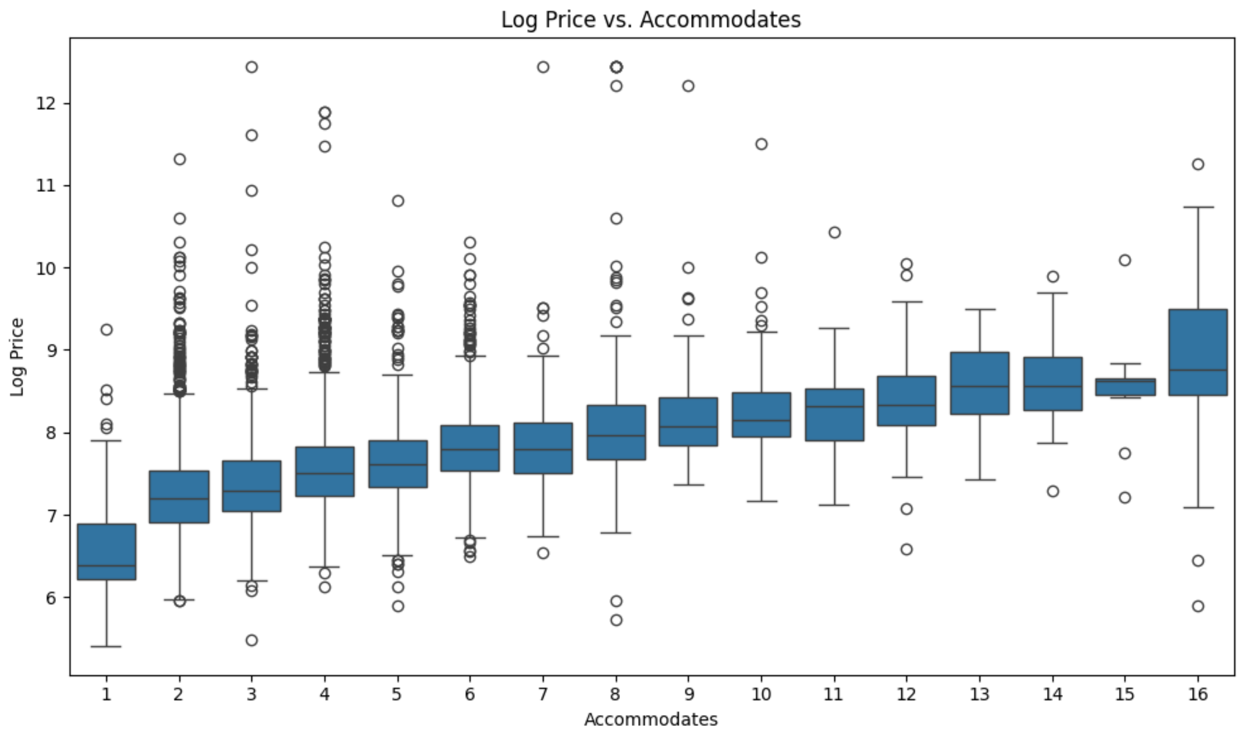


Figure 9: boxplot log price vs accommodates

The first boxplot illustrates the relationship between the number of guests a listing can accommodate (accommodates) and the predicted log-transformed price (log_price). As expected, listings with a higher capacity tend to be more expensive. However, for values greater than 8 guests, we observe a significant increase in price variability. This suggests that beyond a certain capacity threshold, other factors—such as property type, amenities, or location—begin to play a more dominant role in price determination.

Log Price vs. Number of Bathrooms

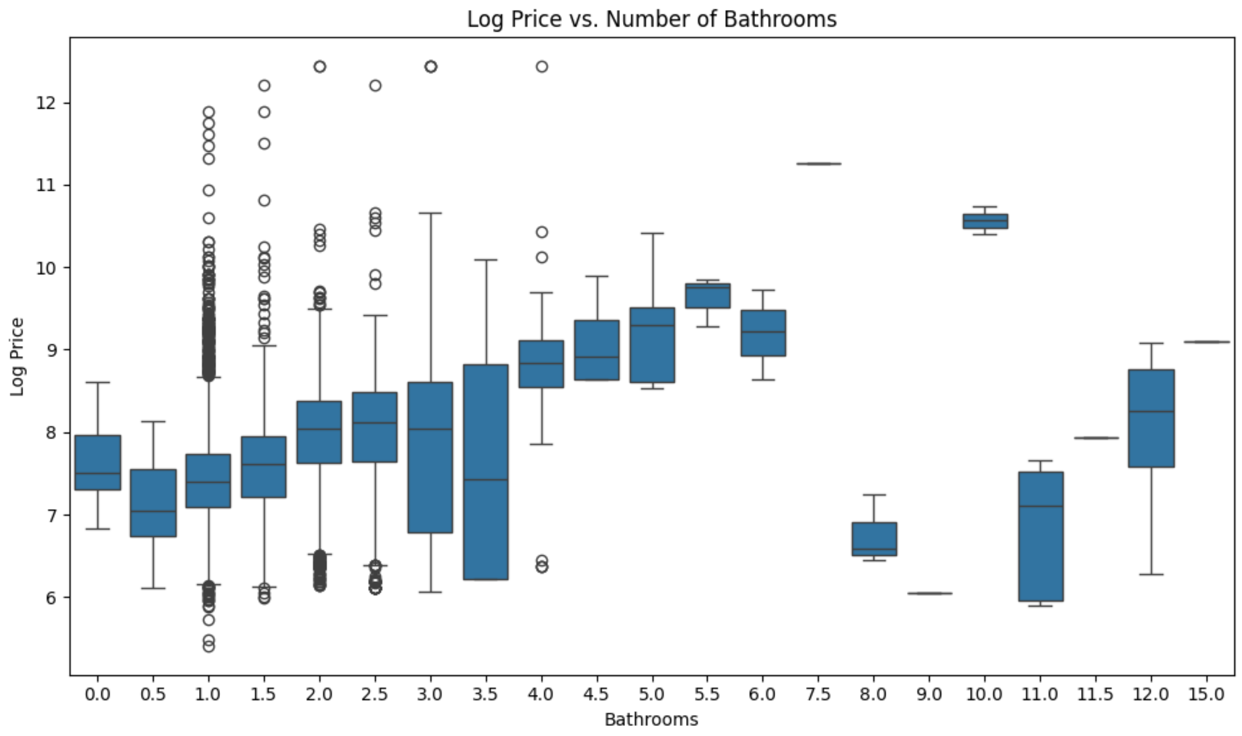


Figure 10: boxplot log price vs bathrooms

This boxplot depicts how the number of bathrooms in a listing relates to the log price. There is a clear upward trend: more bathrooms are associated with higher prices. Listings with 1–2 bathrooms are most common and show a wide range of prices, suggesting variation due to other influential features. For listings with more than four bathrooms, the sample size becomes too small for statistically robust conclusions, though they tend to show very high prices indicative of premium or luxury accommodations.

Log Price vs. Number of Bedrooms

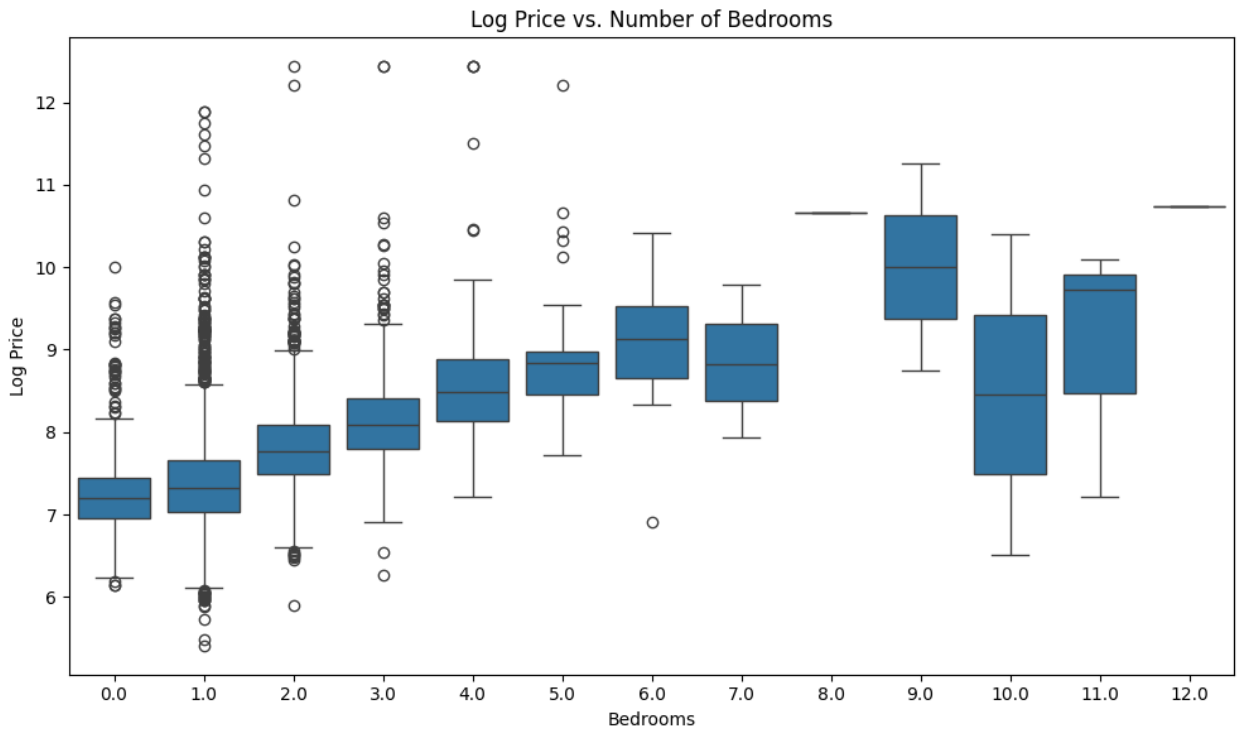


Figure 11: boxplot log price vs bedrooms

Similar to bathrooms and accommodates, listings with more bedrooms generally command higher prices. However, some one-bedroom properties still show high prices, suggesting the influence of premium locations or luxury features. Variability in pricing also grows with the number of bedrooms.

3.2 Relationship Exploration

Average Number of Bathrooms and Beds by Bedroom Count

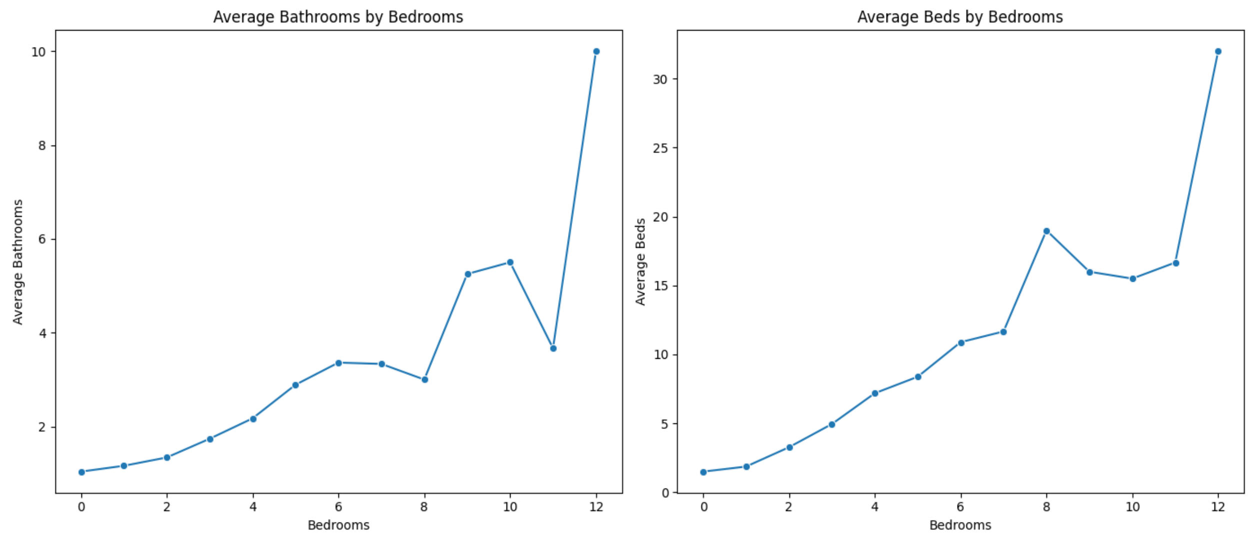


Figure 12: lineplot bathrooms by bedrooms/ beds by bedrooms

This comparison confirms the assumption that larger properties provide more amenities. The number of bathrooms increases roughly proportionally with bedrooms, although many multi-bedroom listings still have fewer bathrooms than bedrooms. On the other hand, the number of beds scales more steeply with bedroom count — likely due to multi-bed setups such as bunk beds or sofa beds. This supports the view that listings scale not only in space but also in guest capacity.

Log Price vs. Review Scores Rating

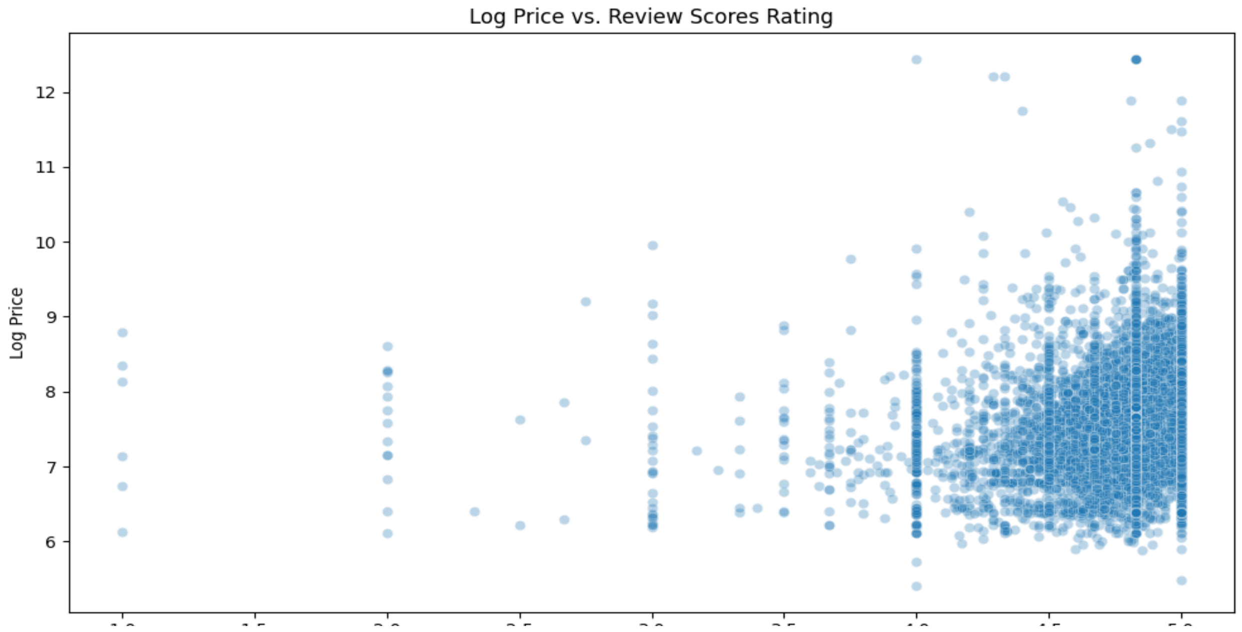


Figure 13: scatterplot log price vs review scores

The scatterplot reveals that the vast majority of listings cluster within the 4.5–5.0 review rating range, which is common in systems where highly dissatisfied users are less likely to leave feedback. Listings with extremely low ratings (below ~4.0) are associated with lower prices, likely reflecting decreased demand.

Interestingly, many review scores are exact integers (e.g., 1.0, 2.0, 3.0), suggesting rating rounding behavior by users.

Overall, although a weak positive trend is visible, the relationship between rating and price is not strictly linear. Other features (such as size and location) play stronger roles in price formation.

Distribution of Reviews per Month

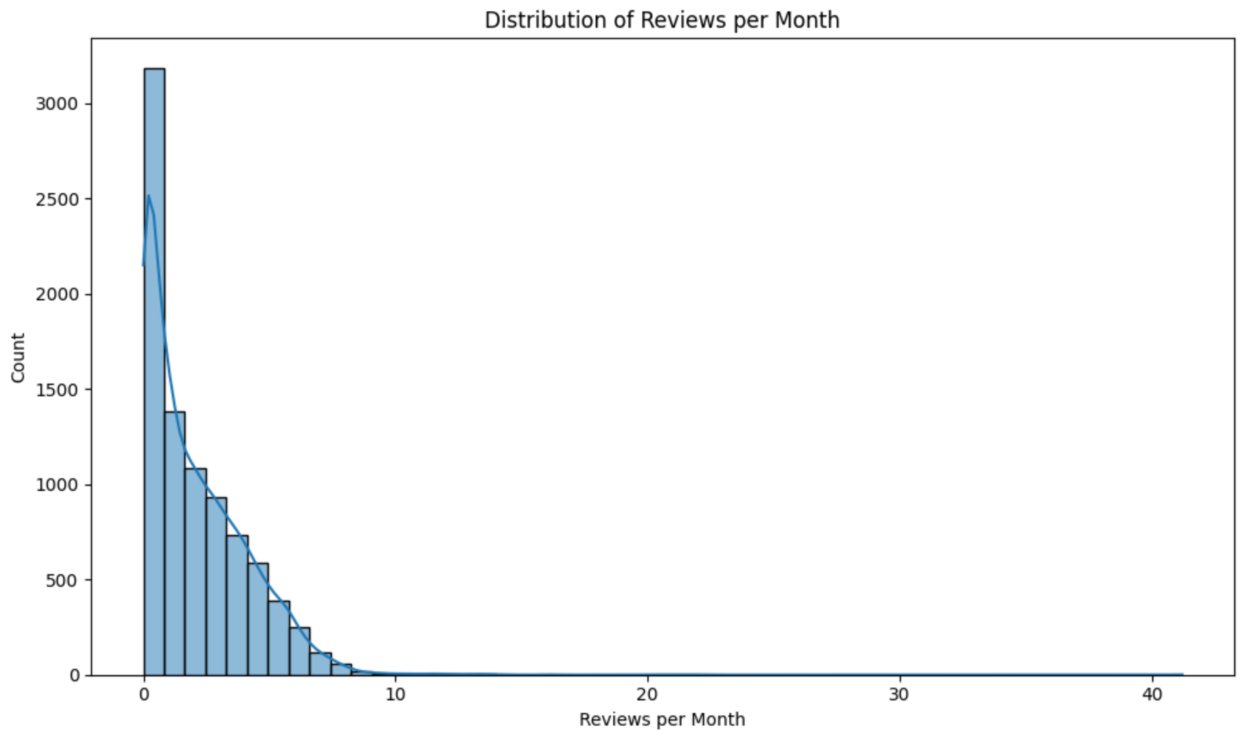


Figure 14: review per month distribution

The histogram of reviews_per_month is strongly right skewed. Most listings receive fewer than 2 reviews per month, with a substantial portion getting close to zero — indicating limited recent activity.

A small number of listings receive more than 4–5 reviews monthly, likely representing highly booked or popular accommodations. These may reflect high-visibility properties or those in high-demand areas.

Number of Amenities

To understand how the number of amenities offered by a listing relates to its price, we visualized both the distribution of amenities and the average predicted price (log-transformed) per amenity range:

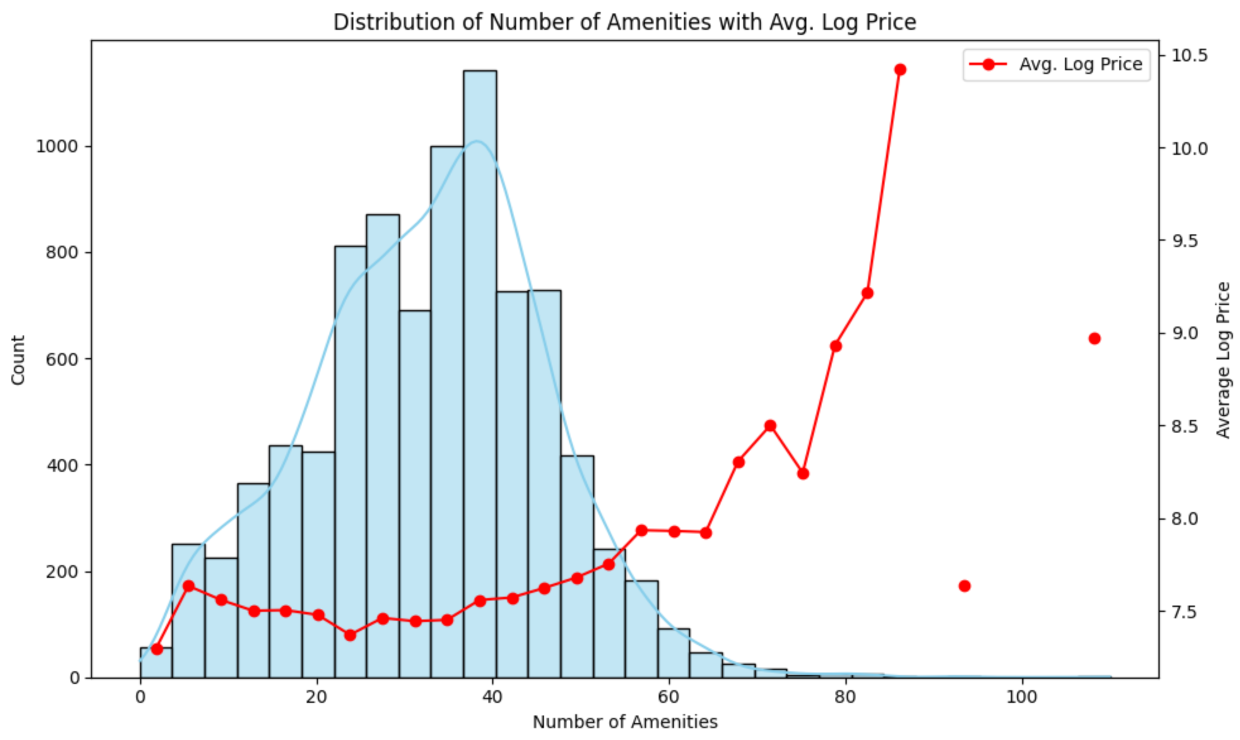


Figure 15: plot amenities distribution

The histogram reveals that the most common listings offer between 30–45 amenities. The red line shows the corresponding average log_price, which gradually increases with the number of amenities up to a certain point. Beyond ~60 amenities, prices may still rise but become highly variable due to the low number of listings in these bins. This suggests diminishing returns: while more amenities can justify higher prices, only a limited number of listings actually offer very high amenity counts.

Property Type

Next, we examine the average price by property_type. The chart below confirms an intuitive trend — entire homes, lofts, and aparthotels command the highest prices, while private or shared rooms are priced significantly lower:

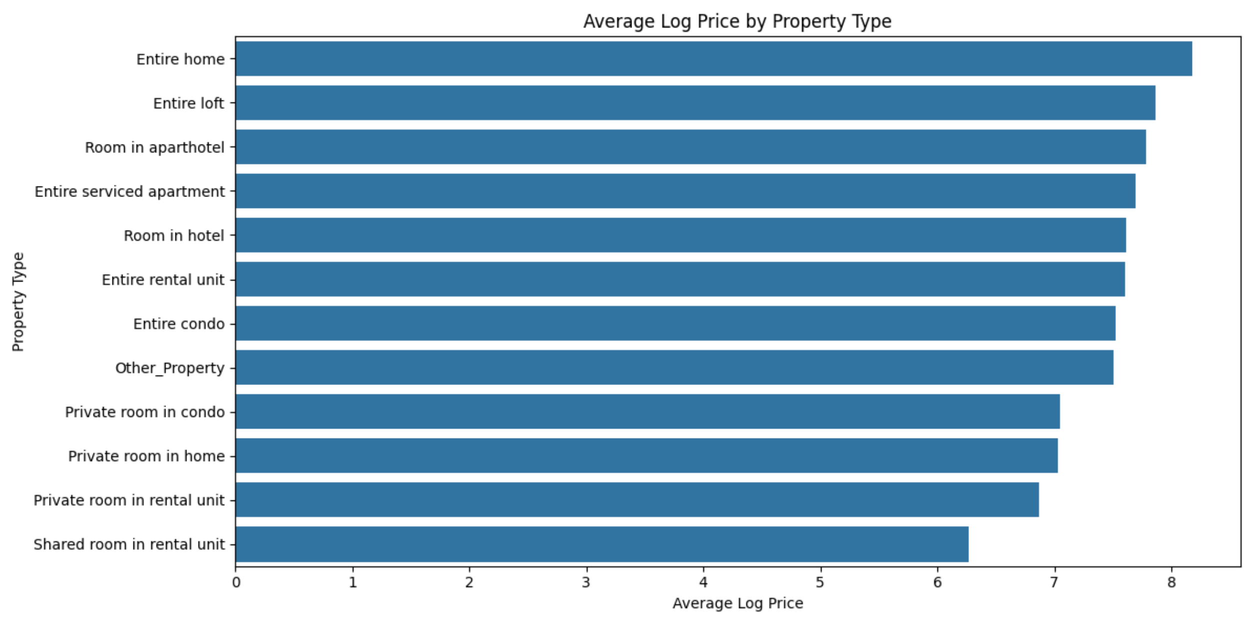


Figure 16: plot average log price by property type

Listings like *"Entire home"*, *"Entire loft"*, and *"Room in aparthotel"* top the list in terms of average log_price, likely reflecting both property size and intended use. On the opposite end, *"Shared room in rental unit"* and *"Private room in rental unit"* show the lowest average prices.

This analysis reinforces the importance of encoded property types in price prediction models (as validated by SHAP and built-in importance), as they encapsulate both physical features and market segment.

4. Modeling

4.1 Setup and Baselines

The goal of this phase is to train, evaluate, and tune machine learning models to accurately predict the log-transformed price (log_price) of Airbnb listings in Prague. We begin by setting up the modeling environment, loading prepared data, defining evaluation metrics, and establishing baseline performances.

4.1.1 Data and Libraries

We imported standard libraries for data handling (pandas, numpy), modeling (e.g., LinearRegression, RandomForestRegressor, XGBRegressor, LGBMRegressor), and model evaluation (mean_squared_error, r2_score, etc.). We also loaded the processed and scaled datasets (X_train_scaled, X_test_scaled, y_train, y_test) along with the fitted StandardScaler and the list of final features used for modeling.

The training and test data shapes were:

- X_train_scaled: 6979 rows × 54 features
- X_test_scaled: 1745 rows × 54 features
- Target: log_price, continuous, already log-transformed

All feature names were verified to match the saved feature list from the data preparation phase.

4.1.2 Evaluation Metrics

To evaluate model performance, we used two sets of metrics:

- **Cross-validation metric (log scale):**
  Root Mean Squared Error (RMSE), reported as negative (neg_root_mean_squared_error) to be compatible with scikit-learn scoring (where higher is better).
- **Final evaluation on the original price scale:**
  We defined a helper function to convert predictions and true values from log scale to original scale using np. expm1(). This function computes:
  - **RMSE** (Root Mean Squared Error)
  - **MAE** (Mean Absolute Error)
  - **R^2** (Coefficient of Determination)

These metrics allow us to compare models both in terms of log-transformed target consistency and real-world interpretability.

4.1.3 Baseline Models

To establish reference points for model performance, we implemented two baseline approaches:

1. **Mean Predictor (Dummy Regressor):**
   A naïve baseline that always predicts the mean value of the target (log_price) from the training set. This results in the following performance on the original price scale:
   - RMSE: 1998.72
   - MAE: 1048.06
   - R^2: –0.036 (below 0, worse than naive mean)
2. **Linear Regression (5-Fold Cross-Validation):**
   We trained a basic linear regression model and evaluated its performance using 5-fold cross-validation. The average RMSE on the log-transformed scale was:
   - **Mean CV RMSE (log scale):** 0.4186 ± 0.0162

This model provides a useful lower bound for comparison with more complex algorithms (e.g., tree ensembles, gradient boosting). Any improvement beyond this will demonstrate added value from feature engineering and algorithmic complexity.

4.2 Candidate Model Training & Evaluation (Initial Cross-Validation)

In this phase, we trained and evaluated several different types of regression models using 5-Fold Cross-Validation on the scaled training dataset. The goal was to establish a performance benchmark for each candidate model using default hyperparameters and to identify promising candidates for further tuning in the next phase.

4.2.1 Selected Candidate Models

We selected a diverse set of regression algorithms, including both regularized linear models and state-of-the-art ensemble-based methods. All models were initialized with default hyperparameters, and random states were set where applicable for reproducibility. The models evaluated were:

- **Ridge Regression** (linear model with L2 regularization)
- **Lasso Regression** (linear model with L1 regularization)
- **Random Forest Regressor**
- **XGBoost Regressor**
- **LightGBM Regressor**

This set ensures a comparison between simple linear baselines and powerful non-linear learners capable of capturing complex interactions.

4.2.2 Evaluation Protocol

Each model was evaluated using **5-Fold Cross-Validation** on the training data (X_train_scaled, y_train). The evaluation metric was **Root Mean Squared Error (RMSE)** on the **log-transformed**

target (log_price), which we denoted as neg_root_mean_squared_error for compatibility with scikit-learn.

4.2.3 Results

The table below shows the mean and standard deviation of cross-validated RMSE scores (on log scale) for each model:

MODEL	MEAN CV RMSE	STD DEV
LIGHTGBM	**0.3769**	0.0143
XGBOOST	0.3883	0.0155
RANDOM FOREST	0.3996	0.0083
RIDGE	0.4466	0.0202
LASSO	0.6663	0.0152

These results highlight that **gradient boosting methods (LightGBM, XGBoost)** significantly outperform both linear models and random forests under default settings. Notably, **LightGBM** achieved the lowest cross-validated RMSE, followed closely by **XGBoost**. Both are strong candidates for hyperparameter tuning and final model selection.

4.2.4 Observations

- The **Lasso regression** underperformed significantly, likely due to overly aggressive regularization that nullified many coefficients.
- **Ridge regression** performed slightly better but still behind non-linear models.
- **Tree-based ensemble models** consistently outperformed linear methods, indicating non-linear patterns in the data.
- **LightGBM** was the most promising model at this stage and will be further tuned in the next phase.

4.3 Hyperparameter Tuning

Following the initial evaluation of candidate models, we selected the top-performing tree-based regressors—**LightGBM**, **XGBoost**, **RandomForest**, and **CatBoost**—for hyperparameter tuning. The aim was to further improve predictive performance by optimizing key model parameters using **RandomizedSearchCV**.

4.3.1 Tuning Approach

Each model was tuned using 5-fold cross-validation and neg_root_mean_squared_error as the primary scoring metric (lower is better). We defined search spaces for the most impactful hyperparameters of each model, balancing complexity and training time. The search was run with n_iter=50 for all models, using CPU or GPU acceleration where available.

Hyperparameter ranges were chosen based on typical values in the literature and the specific characteristics of each model:

- **XGBoost**: tree depth, learning rate, subsample ratio, column sampling, regularization.
- **RandomForest**: tree count, split criteria, feature selection, tree depth.
- **LightGBM**: number of estimators, learning rate.
- **CatBoost**: iterations, depth, learning rate, L2 regularization, split bin count.

4.3.2 Results Summary

The table below summarizes the best cross-validated RMSE (log scale) achieved by each model after tuning:

MODEL	BEST CV RMSE (LOG)	BEST PARAMETERS (SUMMARY)
CATBOOST	**0.3331**	iterations=563, depth=9, learning_rate=0.0587, l2_leaf_reg=1.4, border_count=128
XGBOOST	0.3353	n_estimators=541, max_depth=5, learning_rate=0.0662, subsample=0.87, colsample_bytree=0.85
RANDOMFOREST	0.3623	n_estimators=246, max_depth=None, min_samples_leaf=2, max_features=0.6
LIGHTGBM	*(not tuned in final run)*	—

Observation: CatBoost emerged as the top-performing model with the lowest RMSE on the log-price scale, slightly outperforming XGBoost. Both models significantly improved over their default counterparts. RandomForest also benefited notably from tuning.

4.3.3 Remarks

- **XGBoost and CatBoost** both demonstrated strong predictive capacity, indicating the usefulness of gradient boosting in this domain.
- **RandomForest**, while simpler, still provided solid performance after tuning.
- **LightGBM** was excluded from the final tuning due to resource constraints but remains a strong alternative for future testing.

The tuned models will now be evaluated on the hold-out test set to confirm generalization and guide final model selection.

4.4 Final Model Selection and Training

Based on the cross-validation results from hyperparameter tuning, the **CatBoostRegressor** achieved the best performance, followed very closely by **XGBoost**. Both models showed strong generalization ability, with CatBoost achieving the lowest RMSE. However, due to CatBoost's slightly better cross-validated RMSE and its robust performance without requiring additional preprocessing for categorical features (although we used only numeric features here), we selected **CatBoost** as the final model.

4.4.1 Final Model Configuration

The best hyperparameters for CatBoost were manually retrieved from the RandomizedSearchCV process:

- border_count: 128,
- depth: 9,
- iterations: 563,
- l2_leaf_reg: 1.4078,
- learning_rate: 0.0587

These values reflect a balanced trade-off between model complexity and regularization. Notably, a relatively deep tree structure (depth=9) and moderate learning rate allowed the model to capture nonlinear interactions effectively.

4.4.2 Training the Final Model

The selected CatBoost model was instantiated with the optimal hyperparameters and trained on the entire scaled training dataset (X_train_scaled, y_train) to maximize learning from available data. The training completed successfully in under a minute, and the model confirmed to be properly fitted based on internal attributes (feature_importances_).

This trained CatBoost model will now be evaluated on the hold-out test set in the next phase to measure generalization performance and interpret its predictions.

4.5 Test Set Evaluation & Analysis

After finalizing the model selection and training the tuned XGBoost model on the complete training set, its generalization performance was evaluated using the untouched test set. This phase aimed to estimate how well the model predicts Airbnb listing prices in Prague on new, unseen data.

4.5.1Model Predictions and Metrics

The model predicted prices on the log-transformed scale, which were then inverse-transformed using np. expm1() to obtain prices in the original currency scale (CZK per night). The evaluation on the original scale yielded the following results:

- RMSE: 1205.06
- MAE: 558.41
- R^2: 0.624
- RMSE (log scale): 0.3109

These metrics demonstrate strong predictive performance and represent a significant improvement over the initial baselines. The test R^2 score indicates that approximately 62% of the variance in listing prices is explained by the model.

4.5.2 Overfitting Check

To assess overfitting, the test set RMSE (log scale) was compared with the cross-validation score from hyperparameter tuning:

- Best CV RMSE: 0.3353
- Test RMSE: 0.3109
- Difference: -0.0244

Since the test RMSE is slightly lower than the cross-validated RMSE, we can conclude that the model generalizes well and does not overfit the training data.

4.5.3 Prediction Analysis

The scatter plot below illustrates the relationship between actual and predicted prices. Most points are clustered around the red dashed line (perfect prediction), though some deviation is observed for high-price outliers.

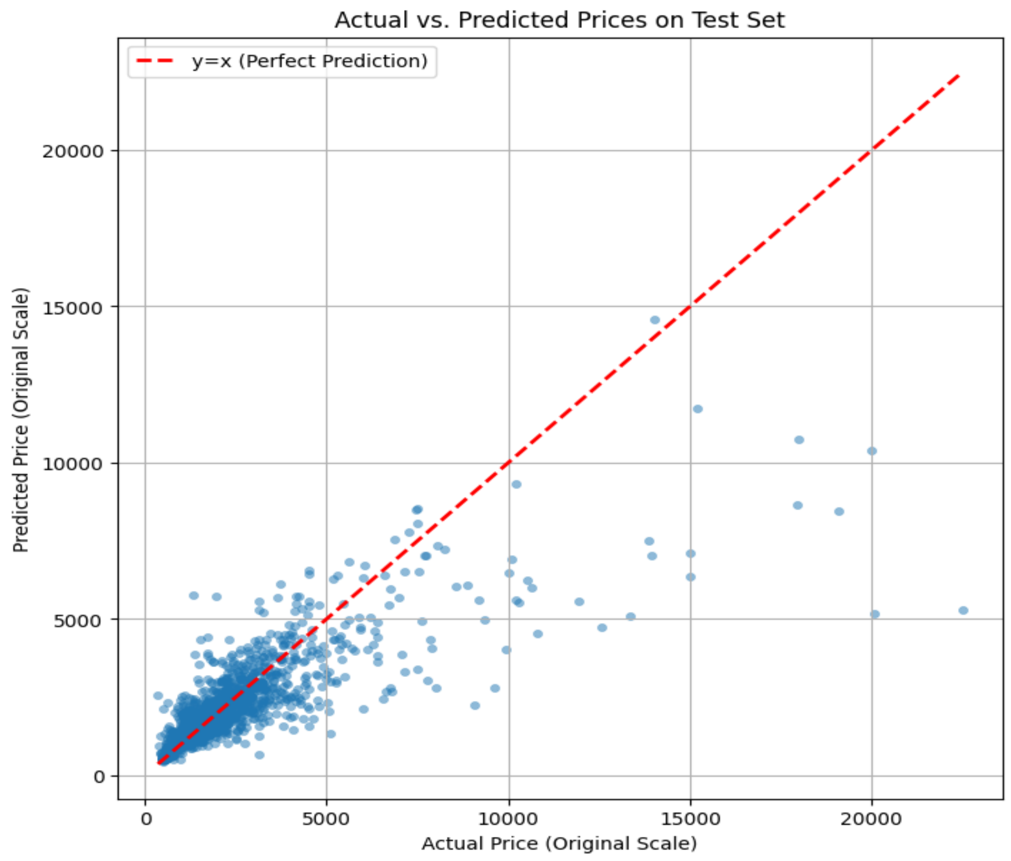


Figure 17: Scatter plot predicted vs actual price

Residual analysis further supports the model's robustness. Residuals (difference between actual and predicted values) are centered around zero, with no strong evidence of systematic bias.

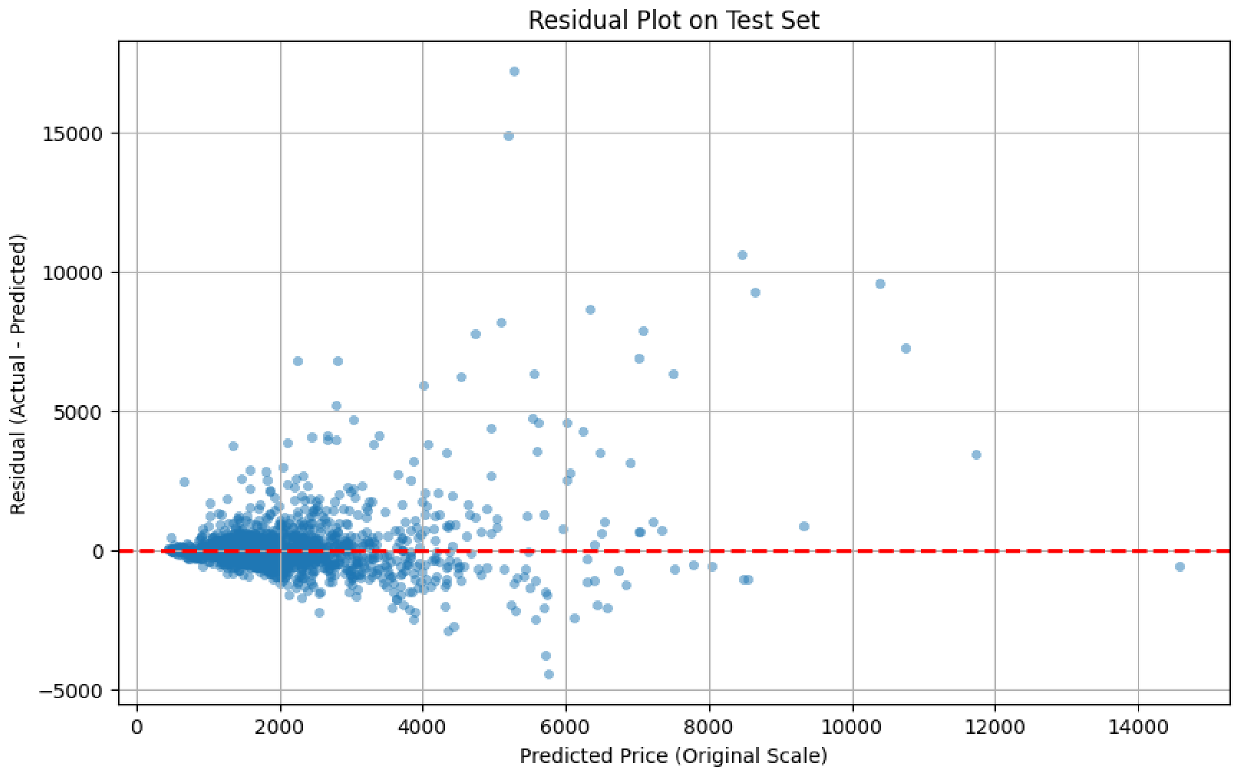


Figure 18: Scatterplot of residual

4.6 Model Documentation & Saving

In this final phase, we documented the key characteristics of the selected model (XGBoost) and saved the necessary artifacts for future reuse. This ensures that the trained model can be easily interpreted, evaluated further, or deployed in a production environment.

4.6.1 Model Limitations and Considerations

Despite its solid performance ($R^2 \approx 0.624$ on the test set), several limitations were observed:

- **High-Price Listings**: The model tends to underpredict very expensive listings, as seen in the residual and scatter plots.
- **Feature Set Limitations**: The dataset lacks dynamic features such as time-based demand, listing images, or micro-location details.
- **Data Freshness**: All predictions are based on a March 2025 snapshot, which might lead to outdated results if used later without retraining.
- **Encoding and Bias**: Categorical encoding (e.g., for property_type) may oversimplify important distinctions, and the model may replicate existing market biases.

4.6.2 Ideas for Model Improvement

To further enhance performance, several improvements are recommended:

- **Calendar Features**: Incorporate booking trends and price changes from calendar.csv.
- **Location Enhancements**: Add distance-to-POI or transport accessibility metrics.
- **Text Feature Engineering**: Use NLP on listing descriptions or amenity details.
- **Interaction Features**: Model key interactions (e.g., room_type × neighbourhood_group).
- **Advanced Models**: Explore CatBoost, deep learning, or ensemble techniques.
- **External Data**: Integrate public transport maps, tourism statistics, or event calendars.

4.6.3 Hyperparameter Tuning Summary

The XGBoost model was optimized using RandomizedSearchCV with 5-fold cross-validation and the RMSE (on the log scale) as the primary metric. The final configuration included:

- max_depth: 5
- learning_rate: 0.066
- n_estimators: 541
- subsample: 0.873
- colsample_bytree: 0.854
- gamma: 0.022
- reg_alpha: 0.501
- reg_lambda: 0.539

These settings produced the best average CV RMSE ≈ 0.3353 on the log-transformed scale.

4.6.4 Saving the Final Model

The trained model, scaler, and final feature list were saved using joblib:

- xgb_price_predictor.joblib: Final trained model
- standard_scaler.joblib: Fitted StandardScaler for preprocessing
- final_feature_list.joblib: List of final features used for training

These artifacts are essential for consistent future inference and are ready for loading in the accompanying Streamlit application or other analysis pipelines.

4.7 Model Interpretation

The goal of this phase was to understand how the final XGBoost model makes predictions and which features most influence its output. To achieve this, we used both global and local interpretation techniques.

4.7.1 Setup and Loading

We imported essential libraries for model inspection, including shap for SHAP value analysis and sklearn.inspection for permutation importance. The following artifacts were loaded from disk:

- Trained XGBoost model (xgb_price_predictor.joblib)
- Scaled training and test datasets
- List of final feature names used during training
- Test target variable (log_price)

All artifacts were loaded successfully and verified for consistency.

4.7.2. SHAP Analysis (Planned in Following Sections)

With SHAP (SHapley Additive exPlanations), we will:

- Calculate SHAP values on the test set.
- Visualize global feature importance based on average absolute SHAP values.
- Explore dependence plots and interaction effects for top features.

This analysis will allow us to not only validate the model's learning behavior but also explain individual predictions and understand how specific features (e.g., number of bedrooms, room type, neighborhood) affect the predicted Airbnb price in Prague.

4.6 Feature Importance Analysis

To better understand how the final XGBoost model makes its predictions, we analyzed feature importance using two complementary methods:

- **Built-in importance** based on the model's internal use of features (typically average gain).
- **Permutation importance** based on the decrease in performance when a feature’s values are randomly shuffled on the test set.

These methods help identify which features were most influential in determining listing prices.

4.6.1 Built-in Importance (XGBoost Gain-Based)

The XGBoost model provides a built-in feature importance metric, where the most important features are those that contributed most to reducing the loss during training (via "gain"). The top three features were:

- bedrooms_log
- accommodates_log
- bathrooms_log

This suggests that the overall **size and capacity** of the listing were the dominant predictors. Other influential features included the **property type**, room type (Entire_home/apt), and location categories such as Old_Town_Center.

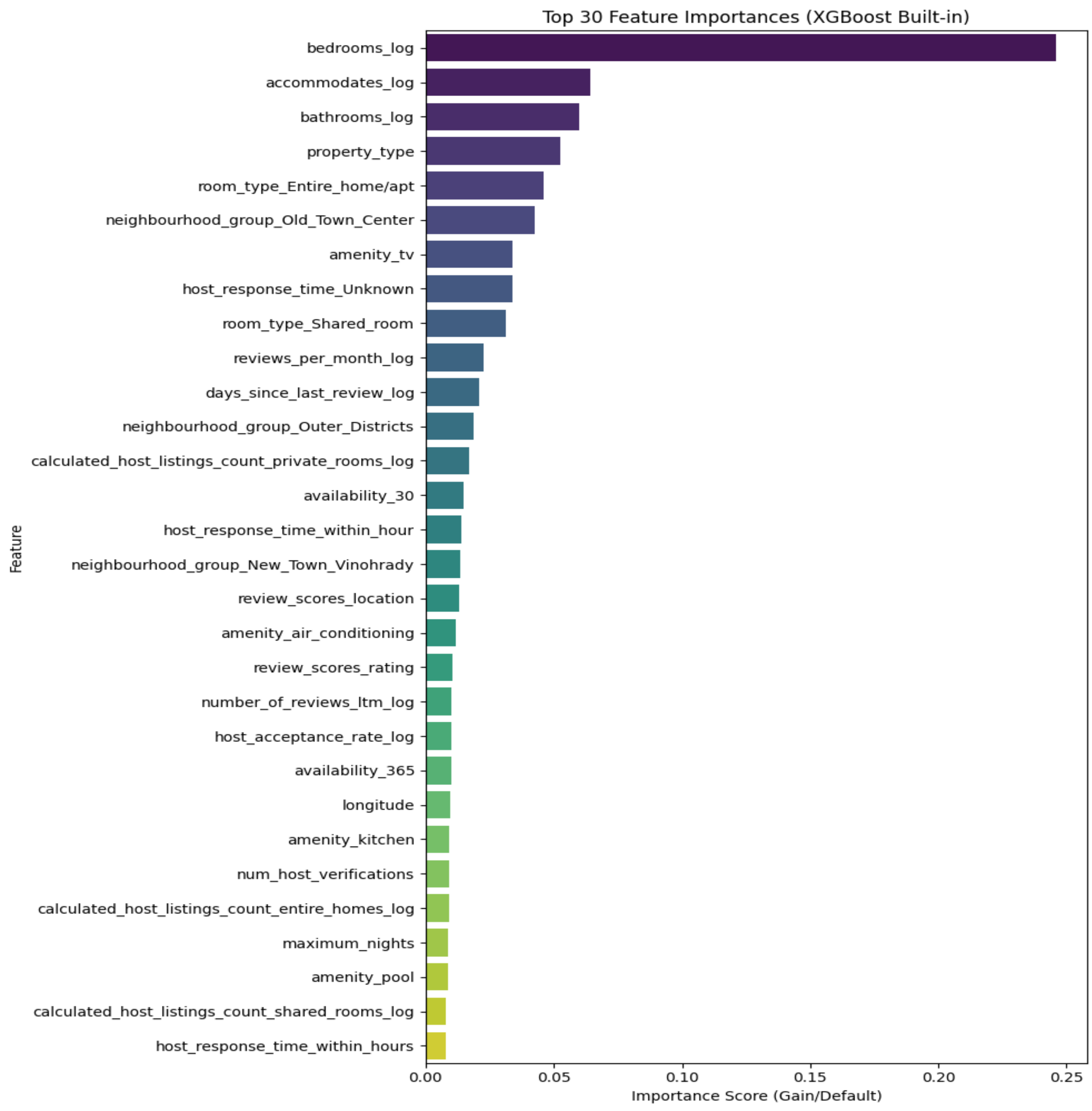


Figure 19: feature importance figure

Permutation Importance (Test Set)

Permutation importance was computed by observing the decrease in R^2 score when each feature was shuffled. This approach is model-agnostic and evaluates the **true predictive contribution** of each variable on unseen data.

Unsurprisingly, bedrooms_log and accommodates_log again emerged as the most important features. However, unlike the built-in method, this technique also highlighted:

- days_since_last_review_log – freshness of reviews
- reviews_per_month_log – review velocity
- availability_30 – near-term availability
- review_scores_rating – general satisfaction

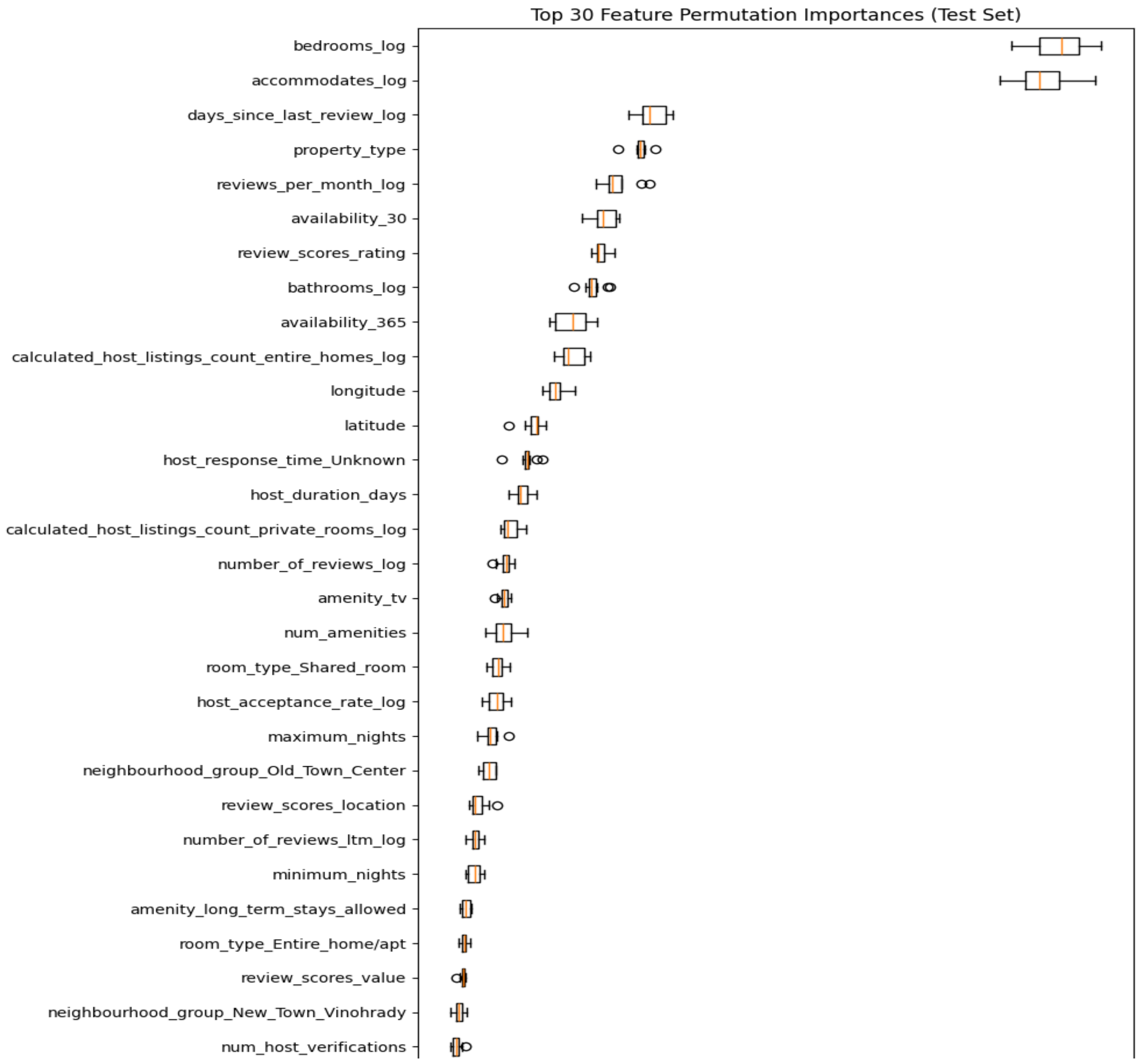


Figure 20: feature permutation importances

This method also elevated the role of **location coordinates** (longitude, latitude) and **host activity history**.

Comparison and Interpretation

The overlap in results across the two methods provides strong evidence that **listing size**, **type**, and **review/availability signals** are the most reliable drivers of price. The permutation importance method arguably offers a more realistic estimate of how well each feature generalizes to unseen data, while the built-in importance reflects how frequently the model used the feature during training.

In particular:

- bedrooms_log and accommodates_log consistently ranked highest in both methods.
- Review-based features (velocity, recency, and rating) were emphasized more in permutation-based results.
- Built-in importance tended to rank categorical OHE variables (like room type or neighborhood) higher, due to frequent splits.

We will prioritize these shared top-ranked features in subsequent SHAP analyses for local interpretability.

4.7 SHAP (SHapley Additive exPlanations) Analysis

To gain deeper insight into how individual features influence the model's predictions, SHAP (SHapley Additive exPlanations) values were computed for the final XGBoost model. SHAP is a model-agnostic, game-theoretically founded approach that assigns each feature an importance value for a particular prediction, making it highly suitable for interpreting complex machine learning models like XGBoost.

4.7.1 SHAP Summary Plots

Two global SHAP summary plots were generated to visualize the contribution of features across all predictions.

- **Beeswarm Plot**: Displays the distribution of SHAP values for the top 20 features, showing both magnitude and direction of influence for each feature (Figure 21).

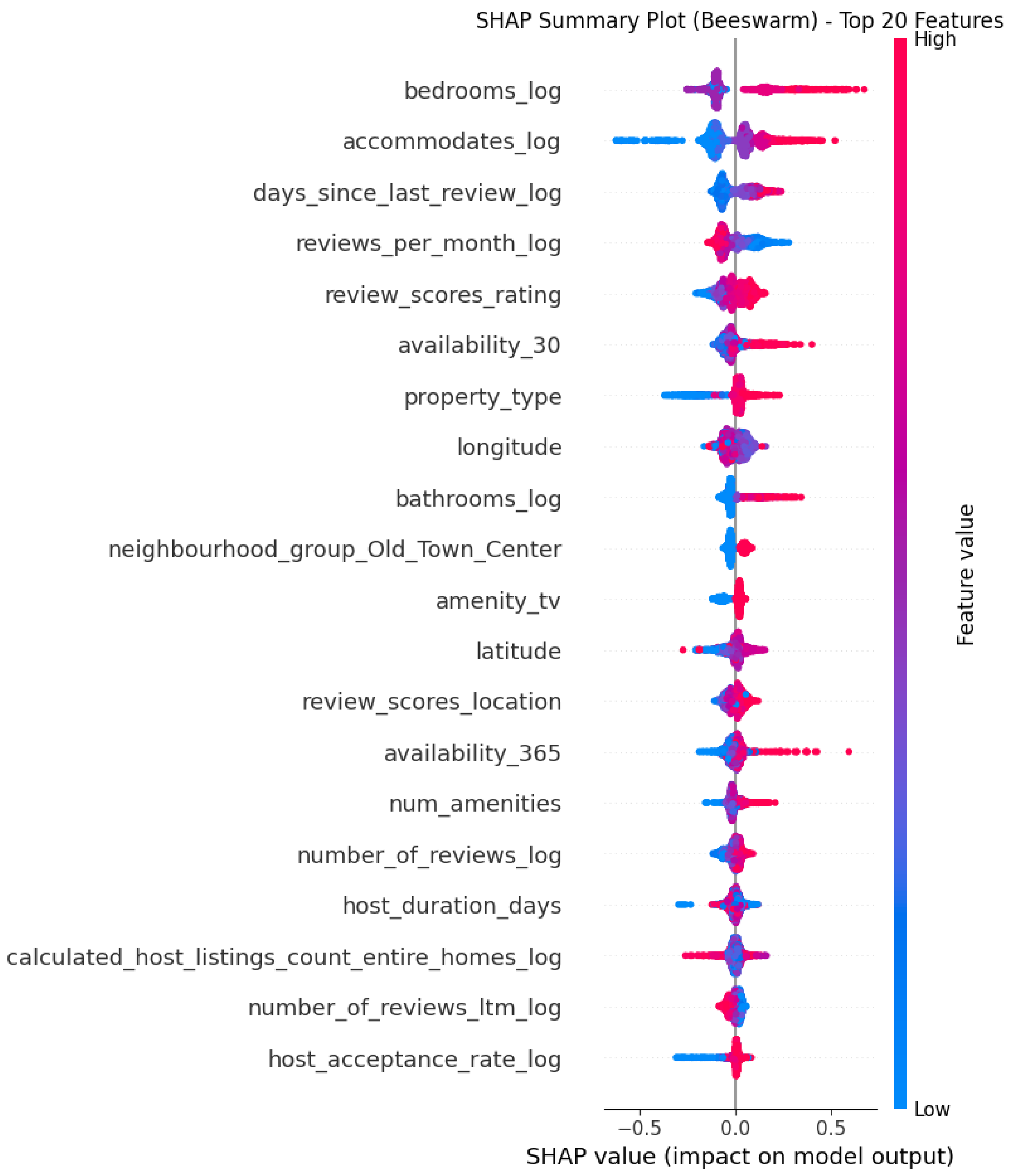


Figure 21: shap summary plot

- **Bar Plot**: Shows the mean absolute SHAP value for each feature, summarizing its overall contribution to the model output (Figure 22).

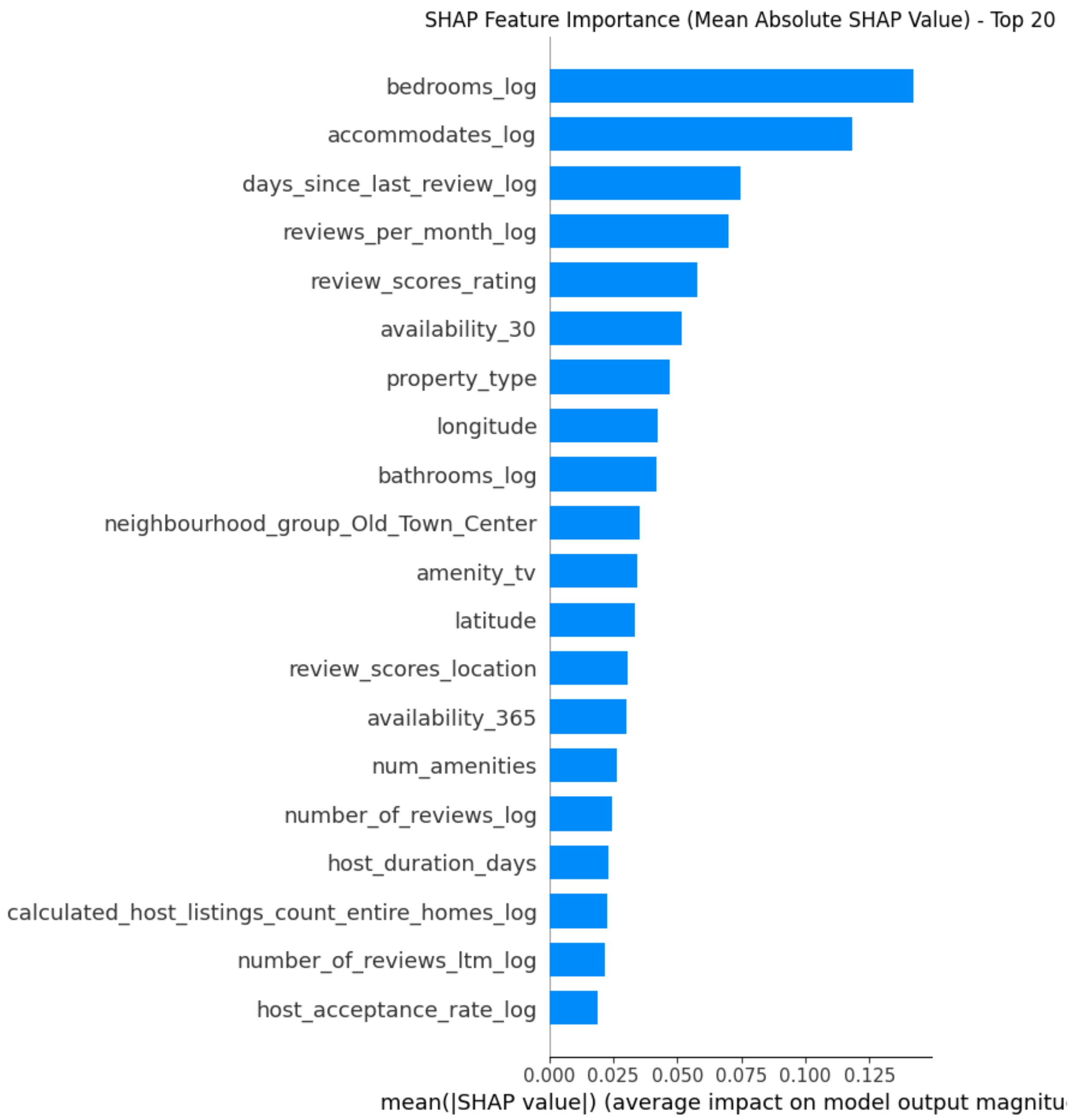


Figure 22: shap feature importance plot

Both plots consistently identify the most influential features: bedrooms_log, accommodates_log, days_since_last_review_log, and reviews_per_month_log dominate the top ranks, confirming previous findings from permutation importance.

4.7.2 SHAP Dependence Plots

To explore non-linear effects and feature interactions, SHAP dependence plots were generated for the top features. These plots show how the SHAP value of a feature changes with its value and highlight potential interaction effects by color coding another variable.

- **bedrooms_log** (Figure 23): Positive relationship with price; effect is amplified for listings with higher property_type encoding.
- **accommodates_log** (Figure 24): Shows a nearly linear positive relationship; weak interaction with calculated_host_listings_count_private_rooms_log.
- **bathrooms_log** (Figure 25): Effect grows with more bathrooms; enhanced further by more bedrooms (interaction with bedrooms_log).

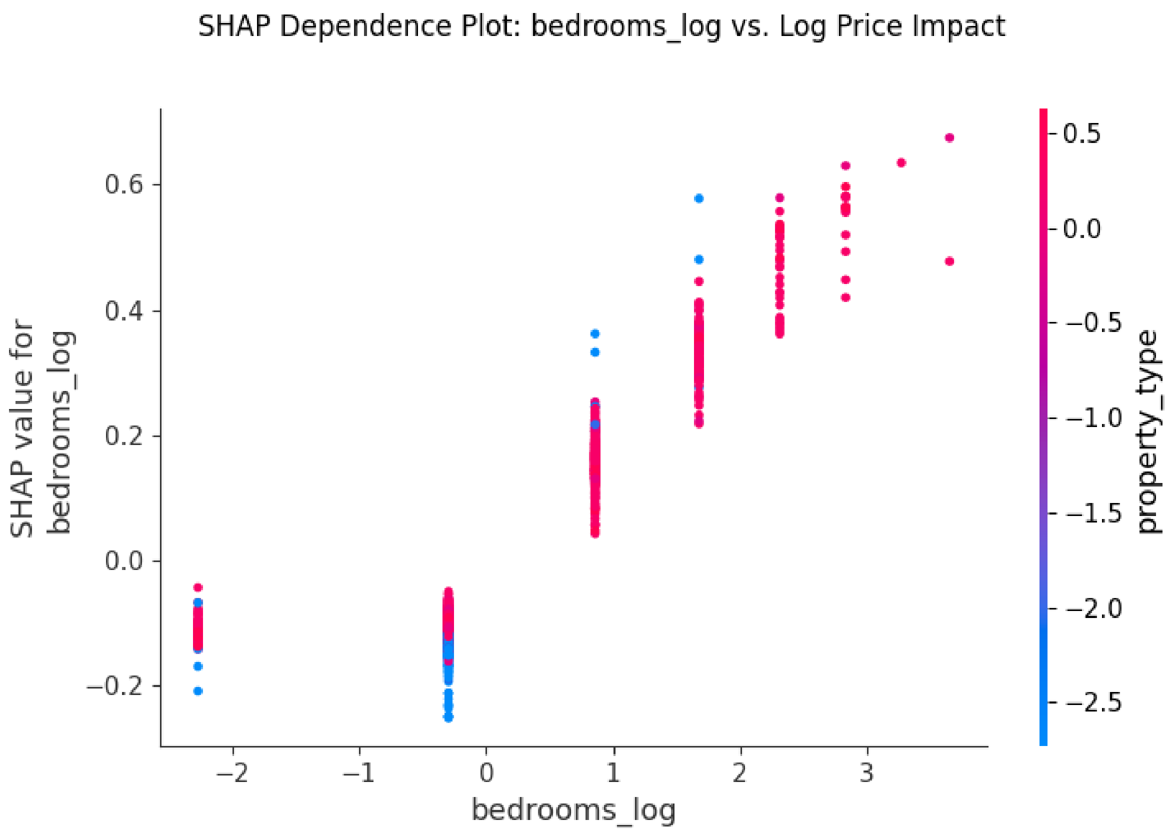


Figure 23: shap plot - bedrooms log

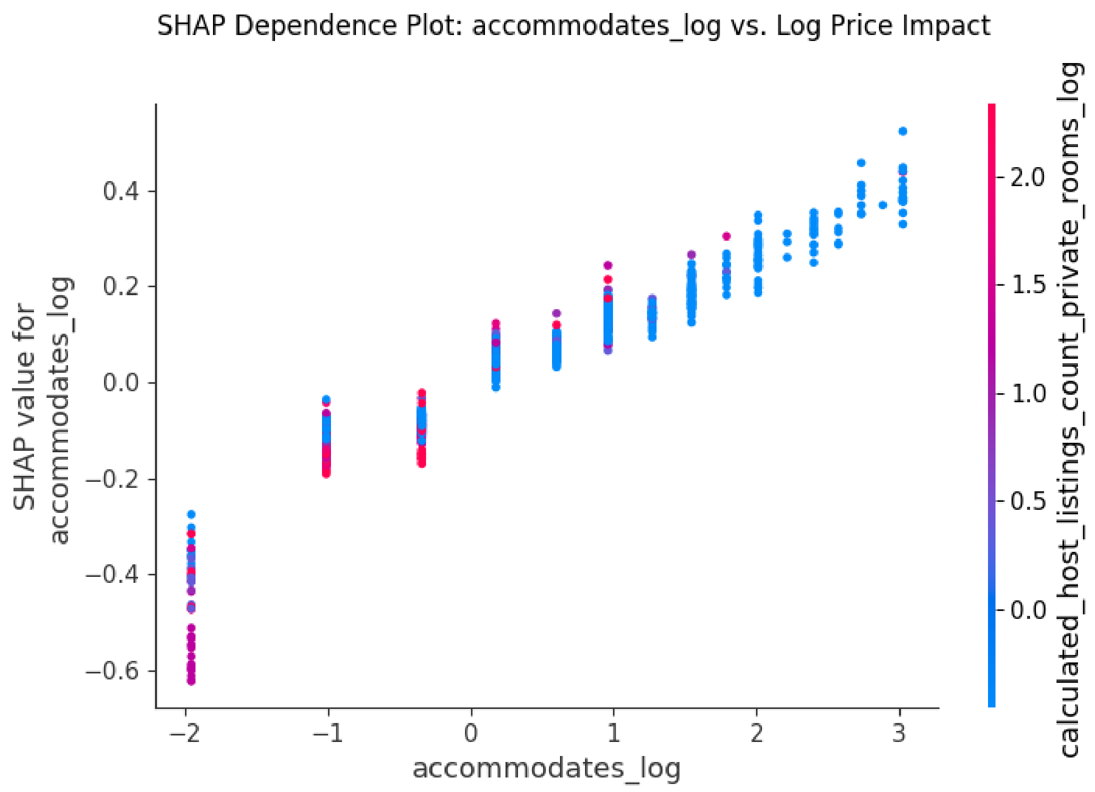


Figure 24: shap plot - accommodates log

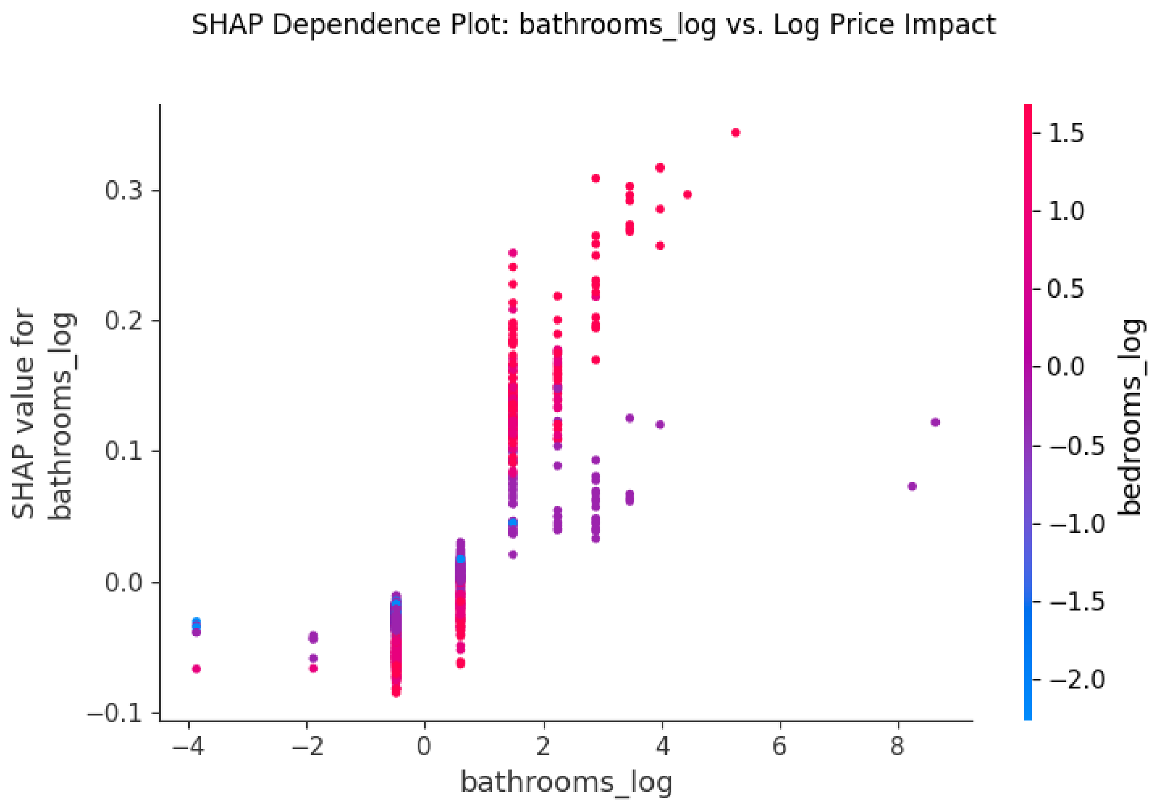


Figure 25: shap plot - bathrooms log

4.7.3 Key Findings from SHAP Analysis

- **Room Count as Primary Driver**: bedrooms_log and accommodates_log show the strongest and most consistent positive impact on the predicted log price.
- **Review Signals Matter**: days_since_last_review_log and reviews_per_month_log both influence price predictions significantly, with more frequent and recent reviews contributing positively.
- **Availability and Location**: Lower short-term availability (availability_30) and being located in neighbourhood_group_Old_Town_Center also push predictions upward.
- **Model Behavior Confirmed**: SHAP results are well-aligned with permutation and built-in importance scores, enhancing trust in the model's decision process.

4.8 Summarize Interpretation Findings

This section consolidates the insights gained from the previous analyses of feature importance using XGBoost's built-in metrics, permutation importance, and SHAP (SHapley Additive exPlanations) values, including both summary and dependence plots.

4.8.1 Summary of Key Findings

Most Important Features
There was strong agreement across all importance estimation methods (Permutation Importance, SHAP Mean Absolute Value, XGBoost Built-in Gain) regarding the most influential predictors of log_price. These include:

- **Listing Size**:
  - bedrooms_log and accommodates_log were consistently ranked as the top two features. Larger listings (more bedrooms or guest capacity) strongly drive price upwards.
  - bathrooms_log also plays a significant role, although with slightly smaller relative impact.
- **Review Characteristics**:
  - **Recency**: Listings with more recent reviews (days_since_last_review_log) are predicted to be more expensive.
  - **Velocity**: Higher review frequency (reviews_per_month_log) also positively affects the price.
  - **Quality**: Higher overall ratings (review_scores_rating) increase predicted log prices.
- **Availability**:

    - Short-term availability (availability_30) is negatively correlated with price—lower availability (i.e., higher occupancy or exclusivity) increases price.
    - Long-term availability (availability_365) also contributes, though with a smaller effect.
- **Location**:
    - Latitude and longitude were both found to be significant, indicating the model captures localized geographic pricing trends beyond general neighborhood groupings.
    - Specific neighborhood categories, especially neighbourhood_group_Old_Town_Center, positively influence predicted price.
- **Property Type**:
    - The target-encoded property_type ranks highly, showing that the average inherent pricing associated with different property types (e.g., entire homes vs. shared rooms) is a critical driver.

4.8.2 Feature Impact Direction & Shape (Based on SHAP)

- **Positive Effects**:
    - Increasing values of bedrooms_log, accommodates_log, bathrooms_log, reviews_per_month_log, review_scores_rating, num_amenities, and host_duration_days all increase the predicted log price.
    - Binary/categorical flags like neighbourhood_group_Old_Town_Center and room_type_Entire_home/apt have positive impacts.
    - Lower values of availability_30 (less availability) are associated with higher predicted prices.
- **Negative Effects**:
    - Higher values of days_since_last_review_log (older or no reviews) tend to decrease predicted price.
    - room_type_Shared_room and unknown host response time (host_response_time_Unknown) negatively influence the prediction, suggesting that private and responsive hosts are perceived as more valuable.
- **Non-Linearities**:
    - Several SHAP dependence plots revealed non-linear relationships, particularly diminishing returns at higher values of bathrooms_log, and complex patterns involving longitude and latitude.

4.8.3 Key Interaction Effects

- **Bedrooms × Property Type**:
  The SHAP dependence plot for bedrooms_log colored by property_type showed that the price impact of having more bedrooms is higher for more expensive property types (e.g., entire homes).

- **Bathrooms × Bedrooms**:
  The value of additional bathrooms is greater in listings that already have more bedrooms, indicating interaction between listing size features.

- **Accommodates × Listing Type Complexity**:
  Some weak interaction was observed with the number of private rooms when plotting accommodates_log, although it was not as strong or consistent as others.

These interaction effects highlight the model's ability to capture subtle, yet meaningful compound relationships between variables.

4.8.4 Other Observations & Surprises

- **Encoding Effectiveness**:
  Target Encoding proved highly effective for high-cardinality categorical features like property_type. One-hot encoding retained importance for select variables like room_type_Entire_home/apt and neighbourhood_group_Old_Town_Center. In contrast, earlier attempts using frequency encoding showed poor performance.

- **Feature Redundancy**:
  Initially, both original and log-transformed features were kept. Eventually, dropping the original versions clarified feature rankings and improved model stability.

- **Low-Importance Features**:
  Several features showed limited value in this model, including num_host_verifications, specific amenity flags (excluding amenity_tv), and host-related binary variables like host_is_superhost or host_identity_verified. Their effects may either be minor or already captured by other correlated features.

4.8.5 Overall Interpretation

The final XGBoost model's predictions of Airbnb listing prices in Prague are primarily driven by:

- **Listing Size**: Bigger listings are more expensive.
- **Location**: Both at the granular level (coordinates) and general level (neighbourhood group).
- **Review Metrics**: Recency, frequency, and rating of reviews significantly contribute to price formation.
- **Availability**: Listings that are frequently booked tend to have higher prices.
- **Property Type**: Differentiates pricing across rental styles (entire home vs. private/shared room).

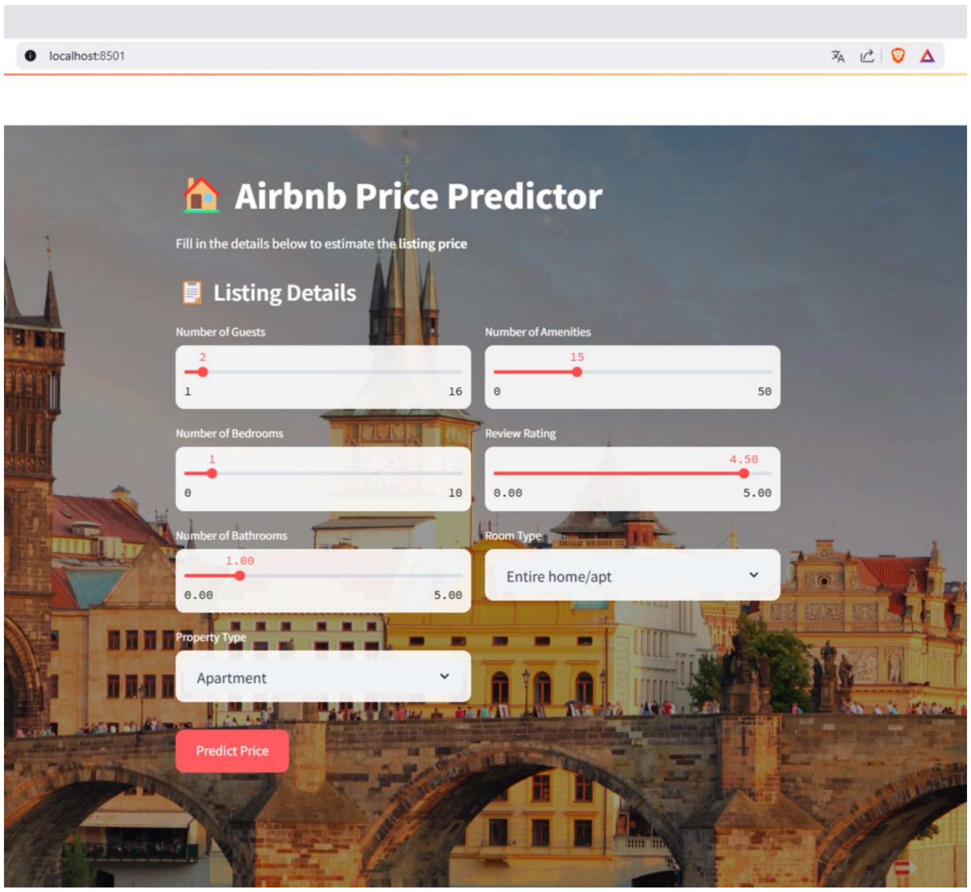
These relationships are intuitive and aligned with domain expectations, yet the model also captures more complex interaction patterns—especially between size and property type or between location and other features. The unexplained variance ($R^2 \approx 0.62$) indicates room for further improvement, especially by incorporating dynamic features (e.g., calendar-based demand, upcoming events) or data from the listing descriptions and photos.

5 Bonus Features & Extensions

5.1 Streamlit Application – Airbnb Price Predictor

As a bonus beyond the core assignment requirements, we developed an interactive **Streamlit application** that allows users to simulate listings and obtain predicted nightly prices. App provides a simple and intuitive user interface where the user can adjust key input features:

- **Number of Guests**
- **Number of Bedrooms**
- **Number of Bathrooms**
- **Number of Amenities**
- **Review Rating**
- **Room Type**



localhost:8501

Airbnb Price Predictor

Fill in the details below to estimate the listing price

Listing Details

Number of Guests: 2 (1–16)

Number of Amenities: 15 (0–50)

Number of Bedrooms: 1 (0–10)

Review Rating: 4.50 (0.00–5.00)

Number of Bathrooms: 1.00 (0.00–5.00)

Room Type: Entire home/apt

Property Type: Apartment

Figure 26: streamlit application preview

Upon entering values and clicking the **Predict Price** button, the app loads the pre-trained XGBoost model (see Sections 3.6 and 3.7), applies preprocessing using the saved scaler and feature list, and predicts the log-transformed price. This value is then inverse transformed to obtain the final price prediction.

The full implementation details of this application are included in the attached Streamlit source code. The app contains all necessary preprocessing steps, model loading logic, and transformation utilities to run independently.

5.2 Analyse the relation between the sentiment and price

To explore whether more expensive listings correlate with greater guest satisfaction, an additional analysis was conducted comparing the **log-transformed price** (log_price) and the **review score rating** (review_scores_rating).

Data Characteristics:

- The review_scores_rating variable is heavily **right skewed**, with most values clustered between 4.5 and 5.0, as shown in the histogram below.

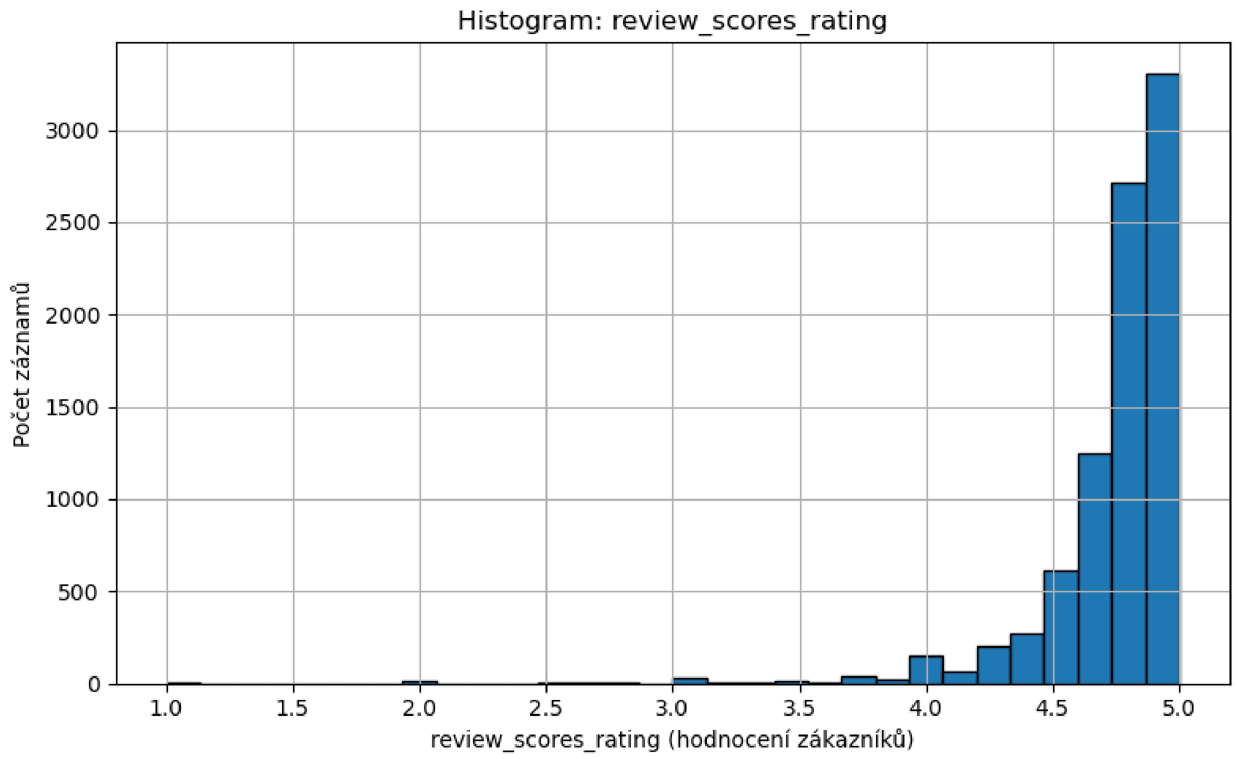


Figure 27: review scores histogram

Correlation Results:

Method	Correlation	p-value	Interpretation
Pearson	0.147	2.75e-43	Weak positive linear relationship
Spearman	0.206	2.24e-84	Weak positive monotonic relationship

Both correlations are statistically significant, but the coefficients indicate a **weak** relationship.

Visual Evidence:

- **Scatter plot** (figure 20) shows no strong trend between price and rating.
- **Boxplot** (figure 21) by price quartile reveals only a **slight increase** in average satisfaction with higher prices.
- **Regression plot** figure 22) shows a very shallow positive slope.

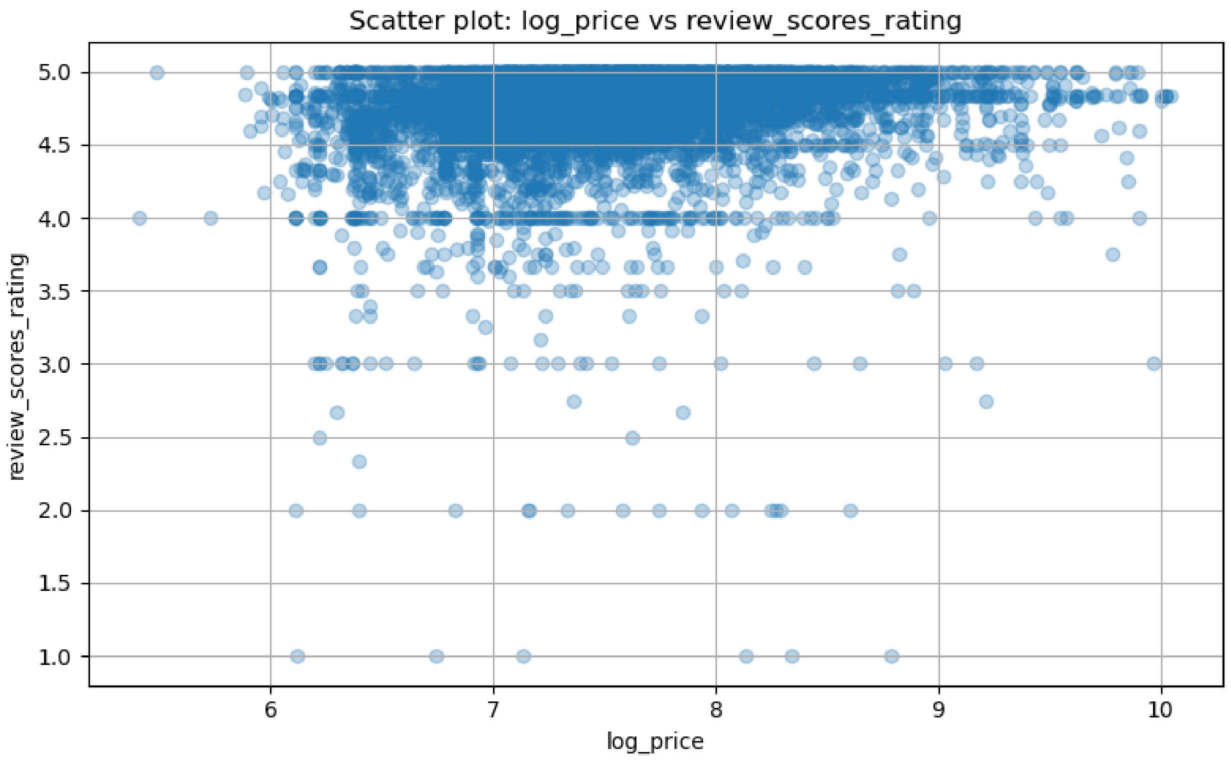


Figure 28: scatter plot log_price vs review_score

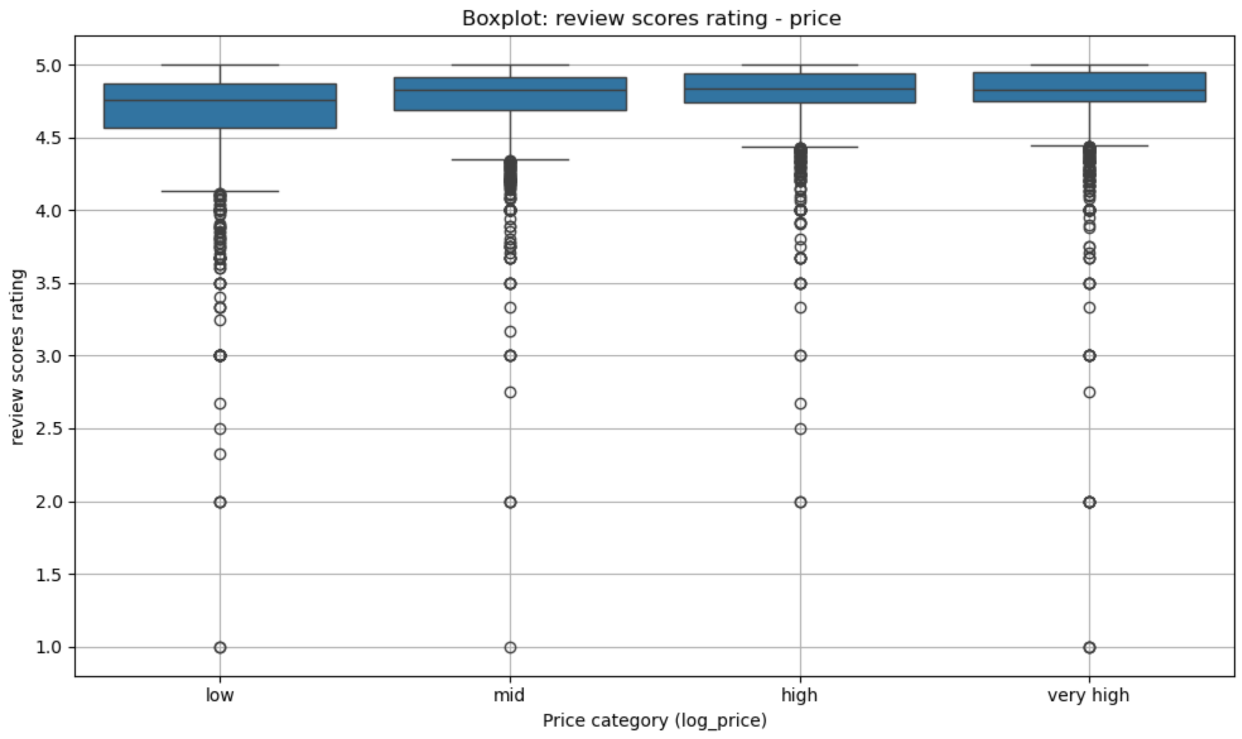


Figure 29: boxplot review scores rating – price

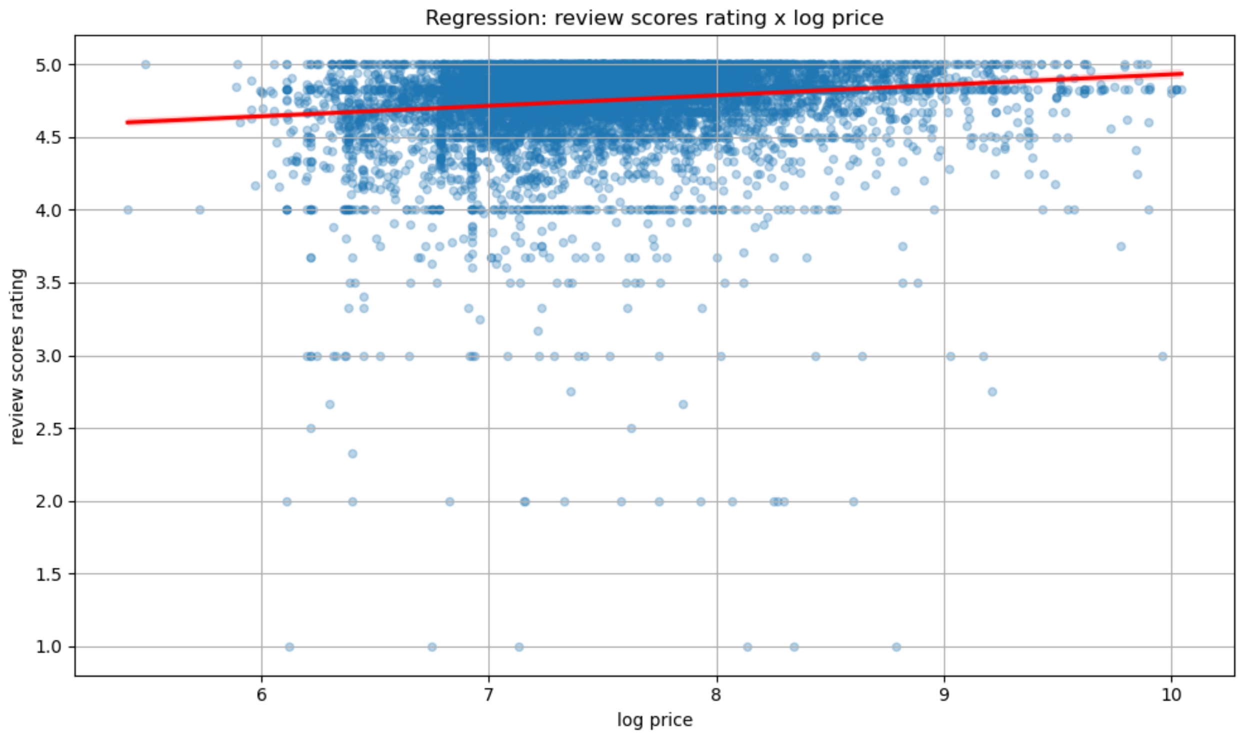


Figure 30: regression review scores x log_price

Final Interpretation:

There is some evidence that guests who paid more were **slightly more satisfied**, but the effect is marginal. Satisfaction is likely influenced by a broader set of factors beyond price (e.g., location, host behavior, amenities). Therefore, **price is not a reliable predictor of satisfaction** on its own.

5.3 Seasonality Analysis and Booking Behavior

This bonus task investigates **seasonal patterns in Airbnb availability** as a proxy for booking intensity and explores the relationship between **price and booking probability**. The goal is to identify peak demand periods (high season) and understand how price influences consumer booking behavior.

5.3.1 Average Availability per Month

We computed the monthly average availability using calendar data, where each day's availability was converted to a binary format (1 = available, 0 = booked). The resulting values represent the proportion of days still available for booking in each month.

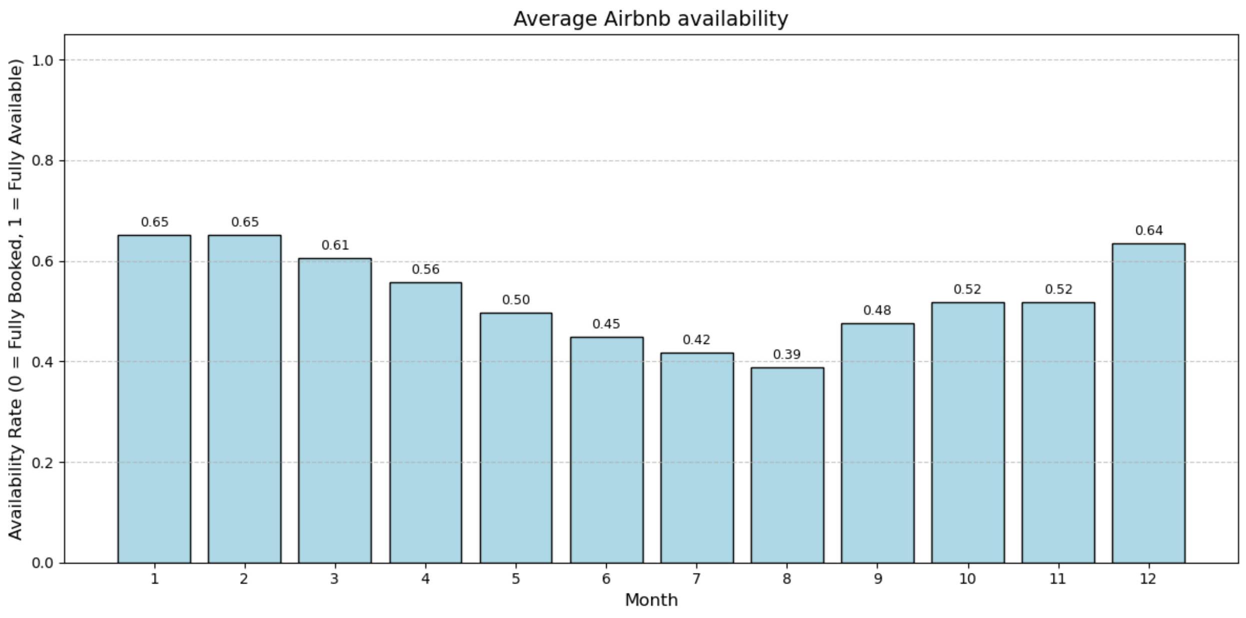


Figure 31: average availability plot

Key Observations:

- **Lowest availability** (i.e., highest occupancy and demand) was recorded in **July (0.42)** and **August (0.39)**, indicating the **peak tourist season**.
- **Highest availability** occurred in **January (0.65)** and **February (0.65)**, consistent with **off-season travel patterns**.
- A **U-shaped pattern** is clearly visible, with mid-year months experiencing significantly higher booking activity than winter months.

These trends reflect typical European travel behavior, where summer holidays dominate rental activity.

5.3.2 Booking Probability vs. Price

To assess the impact of pricing on consumer booking behavior, we plotted the probability of being booked (binary: 1 = booked, 0 = not) against nightly price. A regression line was fitted to identify overall trends.

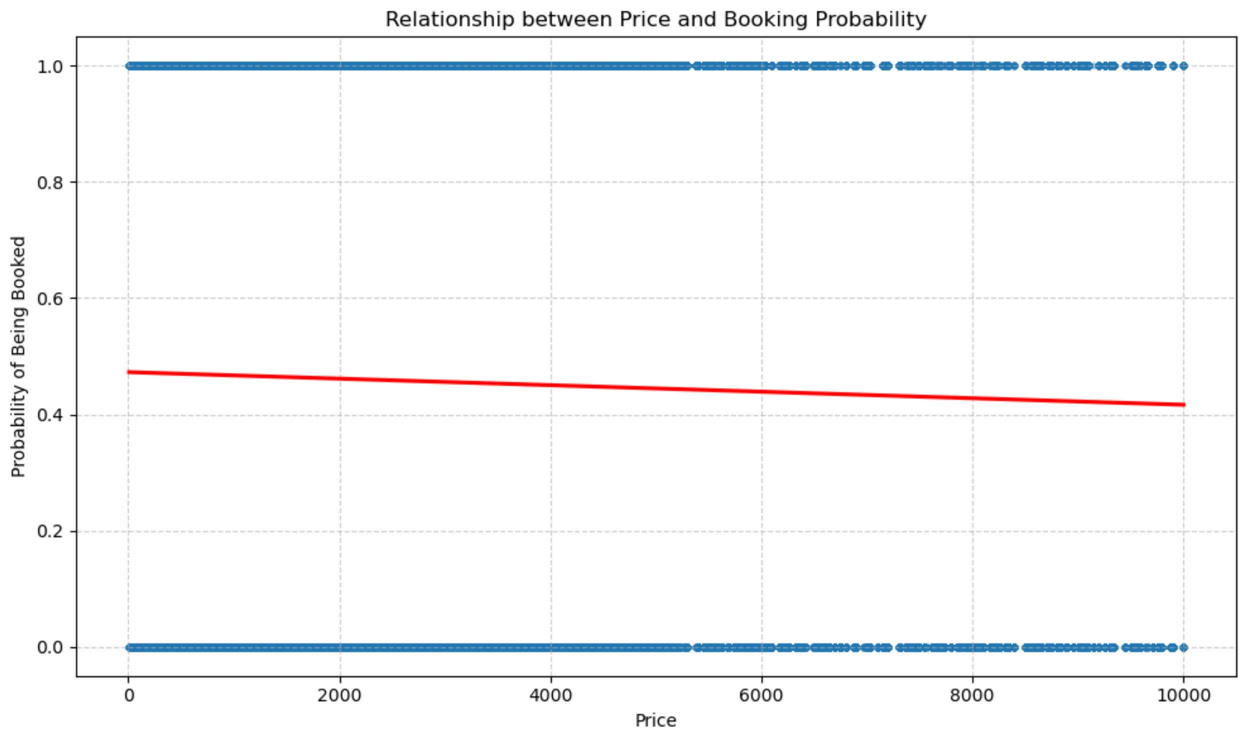


Figure 32: plot relationship price vs booking probability

Key Observations:

- The scatter plot shows a wide range of prices with booking decisions largely binary (booked or not).
- The **negative slope of the red regression line** suggests a **slight inverse relationship**: higher-priced listings are **less likely to be booked**, although the effect is **weak**.
- This confirms expectations that price-sensitive behavior exists but is moderated by other factors such as location, quality, and availability.

5.3.3 Conclusion

This seasonality analysis confirms that **Airbnb demand in Prague peaks during summer**, particularly in **July and August**, and drops in **winter months**. Additionally, there is a **slight negative correlation between price and booking probability**, indicating that **affordability plays a role in consumer decision-making**, though it is not the sole factor.

These insights can inform **pricing strategies and seasonal promotions**, helping hosts optimize occupancy and revenue.