In [1]:
```python
# Initialize Otter
import otter
grader = otter.Notebook("lab4.ipynb")
```

In [2]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import Ridge, LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from src.utils import custom_train_test_split, compute_feature_engineering, get_imp
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score, root_mean_squared_error
import shap
import os


DATA_PATH = "data/AB_NYC_2019.csv"
TARGET = "reviews_per_month"
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
```

Lab 4: Putting it all together in a mini project

For this lab, **you can choose to work alone or in a group of up to three students**. You are in charge of how you want to work and who you want to work with. Maybe you really want to go through all the steps of the ML process yourself or maybe you want to practice your collaboration skills, it is up to you! Just remember to indicate who your group members are (if any) when you submit on Gradescope. If you choose to work in a group, you only need to use one GitHub repo (you can create one on github.ubc.ca and set the visibility to "public"). If it takes a prohibitively long time to run any of the steps on your laptop, it is OK if you sample the data to reduce the runtime, just make sure you write a note about this.

Submission instructions

rubric={mechanics}

You receive marks for submitting your lab correctly, please follow these instructions:

- Follow the general lab instructions.
- Click here to view a description of the rubrics used to grade the questions

- Make at least three commits.
- Push your `.ipynb` file to your GitHub repository for this lab and upload it to Gradescope.
  - Before submitting, make sure you restart the kernel and rerun all cells.
- Make sure to only make one gradescope submission per group, and to assign all group members on gradescope at submission time.
- Also upload a `.pdf` export of the notebook to facilitate grading of manual questions (preferably WebPDF, you can select two files when uploading to gradescope)
- Don't change any variable names that are given to you, don't move cells around, and don't include any code to install packages in the notebook.
- The data you download for this lab **SHOULD NOT BE PUSHED TO YOUR REPOSITORY** (there is also a `.gitignore` in the repo to prevent this).
- Include a clickable link to your GitHub repo for the lab just below this cell
  - It should look something like this https://github.ubc.ca/MDS-2020-21/DSCI_531_labX_yourcwl.

Points: 2

FIND HERE OUR REPOSITORY 👻

Collaborators:

- Elí González Zequeida (`@Eligozo75`): Section 1
- Héctor Palafox Prieto (`@hpalafoxp`): Section 1
- Luis Alonso Álvarez Portugal (`@luisalonso8`): Section 2

Introduction

In this lab you will be working on an open-ended mini-project, where you will put all the different things you have learned so far in 571 and 573 together to solve an interesting problem.

A few notes and tips when you work on this mini-project:

Tips

1. Since this mini-project is open-ended there might be some situations where you'll have to use your own judgment and make your own decisions (as you would be doing when you work as a data scientist). Make sure you explain your decisions whenever necessary.
2. **Do not include everything you ever tried in your submission** -- it's fine just to have your final code. That said, your code should be reproducible and well-documented. For example, if you chose your hyperparameters based on some hyperparameter optimization experiment, you should leave in the code for that experiment so that

someone else could re-run it and obtain the same hyperparameters, rather than mysteriously just setting the hyperparameters to some (carefully chosen) values in your code.
3. If you realize that you are repeating a lot of code try to organize it in functions. Clear presentation of your code, experiments, and results is the key to be successful in this lab. You may use code from lecture notes or previous lab solutions with appropriate attributions.

Assessment

We don't have some secret target score that you need to achieve to get a good grade. **You'll be assessed on demonstration of mastery of course topics, clear presentation, and the quality of your analysis and results.** For example, if you just have a bunch of code and no text or figures, that's not good. If you instead try several reasonable approaches and you have clearly motivated your choices, but still get lower model performance than your friend, don't sweat it.

A final note

Finally, the style of this "project" is different from other assignments. It'll be up to you to decide when you're "done" -- in fact, this is one of the hardest parts of real projects. But please don't spend WAY too much time on this... perhaps "several hours" but not "many hours" is a good guideline for a high quality submission. Of course if you're having fun you're welcome to spend as much time as you want! But, if so, try not to do it out of perfectionism or getting the best possible grade. Do it because you're learning and enjoying it. Students from the past cohorts have found such kind of labs useful and fun and we hope you enjoy it as well.

1. Pick your problem and explain the prediction problem

rubric={reasoning}

In this mini project, you will pick one of the following problems:

1. A classification problem of predicting whether a credit card client will default or not. For this problem, you will use Default of Credit Card Clients Dataset. In this data set, there are 30,000 examples and 24 features, and the goal is to estimate whether a person will default (fail to pay) their credit card bills; this column is labeled "default.payment.next.month" in the data. The rest of the columns can be used as features. You may take some ideas and compare your results with the associated research paper, which is available through the UBC library.

OR

2. A regression problem of predicting `reviews_per_month`, as a proxy for the popularity of the listing with New York City Airbnb listings from 2019 dataset. Airbnb could use this sort of model to predict how popular future listings might be before they are posted, perhaps to help guide hosts create more appealing listings. In reality they might instead use something like vacancy rate or average rating as their target, but we do not have that available here.

Your tasks:

1. Spend some time understanding the problem and what each feature means. Write a few sentences on your initial thoughts on the problem and the dataset.
2. Download the dataset and read it as a pandas dataframe.
3. Carry out any preliminary preprocessing, if needed (e.g., changing feature names, handling of NaN values etc.)

Points: 3

Initially, problem 2 is an interesting topic, but perhaps not the best features or target to predict. Reviews per month can serve as an acceptable proxy for predicting the popularity of future listings, though in reality, Airbnb might prefer metrics like vacancy rate or average rating as targets if they were available. Either way, reviews_per_month can serve as a reasonable target for this regression problem. Some features seem promising based on preliminary insights:

1. Neighborhood is an important variable, as people often consider location when choosing where to stay and may be more likely to write reviews for listings in desirable areas with good subway or public transportation access.
2. Price could provide the most predictive power to the model (as it is in many times, the most important variable when a person choose a place at Airbnb and write a review if it was a competitive price and a good place).
3. Availability could also be useful, as there could be be "good places in photos" or with good reviews but if there is enough availability, people can not write a review or choose it as good place to stay.
4. However, number of reviews is probably highly correlated with the target variable, which could potentially lead to data leakage.
5. Room type may also affect the visitor experience and influence review behavior, but maybe in a smaller magnitude than other features.
6. Calculated host listing counts: here we can see a difference between people that have professional or experience with Airbnb compared to new ones that are just beginning to list a property.
7. Geographic features may be also a factor as people intend to be nearer tourist attractions as Central Park, Wall Street, Manhattan or Times Square than in other places.

8. Booking constraints can deter some guests or attracts longer travelers (who maybe have higher income).

Additionally, there are challenges with the review data itself: many people stay in a place and enjoy it but don't write a review, while dissatisfied guests are far more likely to leave feedback, which can create imbalance in the data and also a clear selection bias. Finally, missing values are likely present and will need to be addressed and handled correctly. All of this is preliminary, even before looking into the data, splitting into the train/test and performing an EDA.

In [ ]:

```

```

2. Data splitting

rubric={reasoning}

Your tasks:

1. Split the data into train and test portions.

> Make the decision on the `test_size` based on the capacity of your laptop.

Points: 1

In [3]:

```
path = 'data/split_data/train_set.csv'

# Check if the path is a file
if os.path.isfile(path):
    print(f"The file exists.")
else:
    print("Didn't find the datasets. Creating them from scratch:")
    %run src/jobs/01_split_data.py
```

```
The file exists.
```

In [4]:

```
df = pd.read_csv(DATA_PATH)
df = df.dropna(subset=[TARGET])
print(df.shape)
df.head()
```

```
(38843, 16)
```

Out[4]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude
0	2539	Clean & quiet apt home by the park	2787	John	Brooklyn	Kensington	40.6474
1	2595	Skylit Midtown Castle	2845	Jennifer	Manhattan	Midtown	40.75362
3	3831	Cozy Entire Floor of Brownstone	4869	LisaRoxanne	Brooklyn	Clinton Hill	40.68514
4	5022	Entire Apt: Spacious Studio/Loft by central park	7192	Laura	Manhattan	East Harlem	40.79851
5	5099	Large Cozy 1 BR Apartment In Midtown East	7322	Chris	Manhattan	Murray Hill	40.74767

In [5]:
```
df_train, df_test = custom_train_test_split(df, "host_id")


print("PCT of rows in train:", 100.0 * len(df_train) / len(df))
print("PCT of rows in test:", 100.0 * len(df_test) / len(df))
```

```
PCT of rows in train: 69.99974255335582
PCT of rows in test: 30.000257446644184
```

In [6]:
```
not_use_cols = [
    'id',
    'host_id',
    'host_name',
    'last_review',
]
use_cols = [c for c in df if c not in not_use_cols]
df_train = df_train[use_cols].copy()
df_test = df_test[use_cols].copy()
```

In [7]:
```
df_train.to_csv("data/split_data/train_set.csv",index=False)
df_test.to_csv("data/split_data/test_set.csv",index=False)
```

3. EDA

rubric={viz,reasoning}

Perform exploratory data analysis on the train set.

Your tasks:

1. Include at least two summary statistics and two visualizations that you find useful, and accompany each one with a sentence explaining it.
2. Summarize your initial observations about the data.
3. Pick appropriate metric/metrics for assessment.

Points: 6

First, it is important to note that we split the data into train and test sets (70-30) by host_id to prevent data leakage. This ensures that all listings from the same host are in either the train or test set, but not both. This way, the model learns to generalize to new hosts rather than just new listings from already known hosts.

EDA

a) After splitting, both categorical and numeric data types are clean with no missing values in the training set (27,190 entries). We removed 20% of the original data that had missing reviews_per_month values (corresponding to listings with zero reviews). The target variable distribution in the training set shows: median is 0.72 reviews per month, the minimum is 0.01 "reviews per month", the mean is 1.37 reviews and the maximum is 58.5 reviews per month. So, we are going to model "reviews per month" with reviews that have, at least, one review so it effectively predict review frequency for listings that attract visitors, rather than if a listing receive a review, at all.

b) Geographic Patterns in reviews: The mean of "reviews_per_month" is much higher in outer boroughs than in NYC's tourist centers. Counterintuitively, outer boroughs (Queens: 1.96, Bronx: 1.76) have higher average reviews per month than tourist centers (Manhattan: 1.26, Brooklyn: 1.28). Possible explanations include: (a) budget travelers stay longer and are more likely to leave reviews, (b) sample size bias in outer boroughs where only the best listings survive, or (c) tourists in Manhattan prefer hotels over Airbnb, as they may be higher-income travelers.

c) Price Outliers: We observe very large outliers across many numeric features, with right-skewed distributions. Price ranges up to 10000 USD.

d) Additional Outliers Across Features: Several other features contain extreme outliers:

- One listing requires a minimum stay of 999 days
- Maximum "reviews_per_month" is 58.5, equivalent to nearly two reviews per day, which is highly unlikely
- Maximum total reviews is 629, which seems unreasonably high
- One host has up to 232 listings

- Some listings show 365 days of availability, which is unrealistic for popular properties

e) Neighborhood-Level Patterns: At the neighborhood level, there are no clear trends. Additionally, some neighborhoods with high mean reviews have very few listings, suggesting these patterns may not be reliable or generalizable.

f) Room Type Analysis: Private rooms and shared rooms (1.44 and 1.43 reviews/month, respectively) have slightly higher average reviews_per_month than entire homes/apartments (1.30). However, the dataset contains more listings in Brooklyn and Manhattan (44% and 41%, respectively) than in the other boroughs. We hypothesize that private rooms attract budget travelers who are more likely to leave reviews compared to higher-income travelers who book entire homes.

g) Geographic Distributions: Latitude and longitude show approximately normal distributions within reasonable ranges for NYC.

h) Spatial Relationships: Visualizations reveal mostly no clear spatial trends between geographic features and reviews_per_month. The only strong relationship observed is between "number_of_reviews" and "reviews_per_month", which is expected but indicates potential data leakage if "number_of_reviews" is used as a feature.

In [8]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27190 entries, 0 to 27189
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   name                            27184 non-null  object 
 1   neighbourhood_group             27190 non-null  object 
 2   neighbourhood                   27190 non-null  object 
 3   latitude                        27190 non-null  float64
 4   longitude                       27190 non-null  float64
 5   room_type                       27190 non-null  object 
 6   price                           27190 non-null  int64  
 7   minimum_nights                  27190 non-null  int64  
 8   number_of_reviews               27190 non-null  int64  
 9   reviews_per_month               27190 non-null  float64
 10  calculated_host_listings_count  27190 non-null  int64  
 11  availability_365                27190 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 2.5+ MB
```

In [9]:
```
df_train.isna().mean()
```

Out[9]:
```
name                              0.000221
neighbourhood_group               0.000000
neighbourhood                     0.000000
latitude                          0.000000
longitude                         0.000000
room_type                         0.000000
price                             0.000000
minimum_nights                    0.000000
number_of_reviews                 0.000000
reviews_per_month                 0.000000
calculated_host_listings_count    0.000000
availability_365                  0.000000
dtype: float64
```

In [10]:
```
df_train.describe()
```

Out[10]:

	latitude	longitude	price	minimum_nights	number_of_reviews	rev
count	27190.000000	27190.000000	27190.000000	27190.000000	27190.000000	
mean	40.727681	-73.950990	142.489702	5.790953	29.232953	
std	0.055250	0.046678	206.368954	16.108038	48.545982	
min	40.508680	-74.244420	0.000000	1.000000	1.000000	
25%	40.688240	-73.982270	68.000000	1.000000	3.000000	
50%	40.721180	-73.954550	100.000000	2.000000	9.000000	
75%	40.762910	-73.935342	169.000000	4.000000	33.000000	
max	40.908040	-73.712990	10000.000000	999.000000	629.000000	

In [11]:
```
df_train.groupby("neighbourhood_group")[TARGET].mean().sort_values(ascending=False)
```

Out[11]:
```
neighbourhood_group
Queens           1.958524
Staten Island    1.854872
Bronx            1.761455
Brooklyn         1.283295
Manhattan        1.260512
Name: reviews_per_month, dtype: float64
```

In [12]:
```
df_train["neighbourhood"].value_counts(True)
```

```
Out[12]: neighbourhood
Williamsburg           0.083854
Bedford-Stuyvesant     0.083339
Harlem                 0.056933
Bushwick               0.048547
Upper West Side        0.038911
                         ...
West Farms             0.000037
Westerleigh            0.000037
Shore Acres            0.000037
Lighthouse Hill        0.000037
Silver Lake            0.000037
Name: proportion, Length: 216, dtype: float64
```

```
In [13]: df_train.groupby("neighbourhood", as_index=False).agg(
    {TARGET: ["mean", "count"]}
).sort_values(by=("reviews_per_month", "mean"), ascending=False)
```

Out[13]:

	neighbourhood	reviews_per_month	
		mean	count
139	New Dorp Beach	5.500000	2
173	Silver Lake	5.490000	1
58	East Elmhurst	5.150650	123
208	Whitestone	5.060000	2
179	Springfield Gardens	4.743731	67
...	...	...	...
96	Holliswood	0.310000	1
145	Oakwood	0.285000	2
172	Shore Acres	0.280000	1
204	West Farms	0.160000	1
119	Marble Hill	0.138571	7

216 rows × 3 columns

```
In [14]: df_train["room_type"].value_counts(True)
```

```
Out[14]: room_type
Entire home/apt    0.519934
Private room       0.457595
Shared room        0.022471
Name: proportion, dtype: float64
```

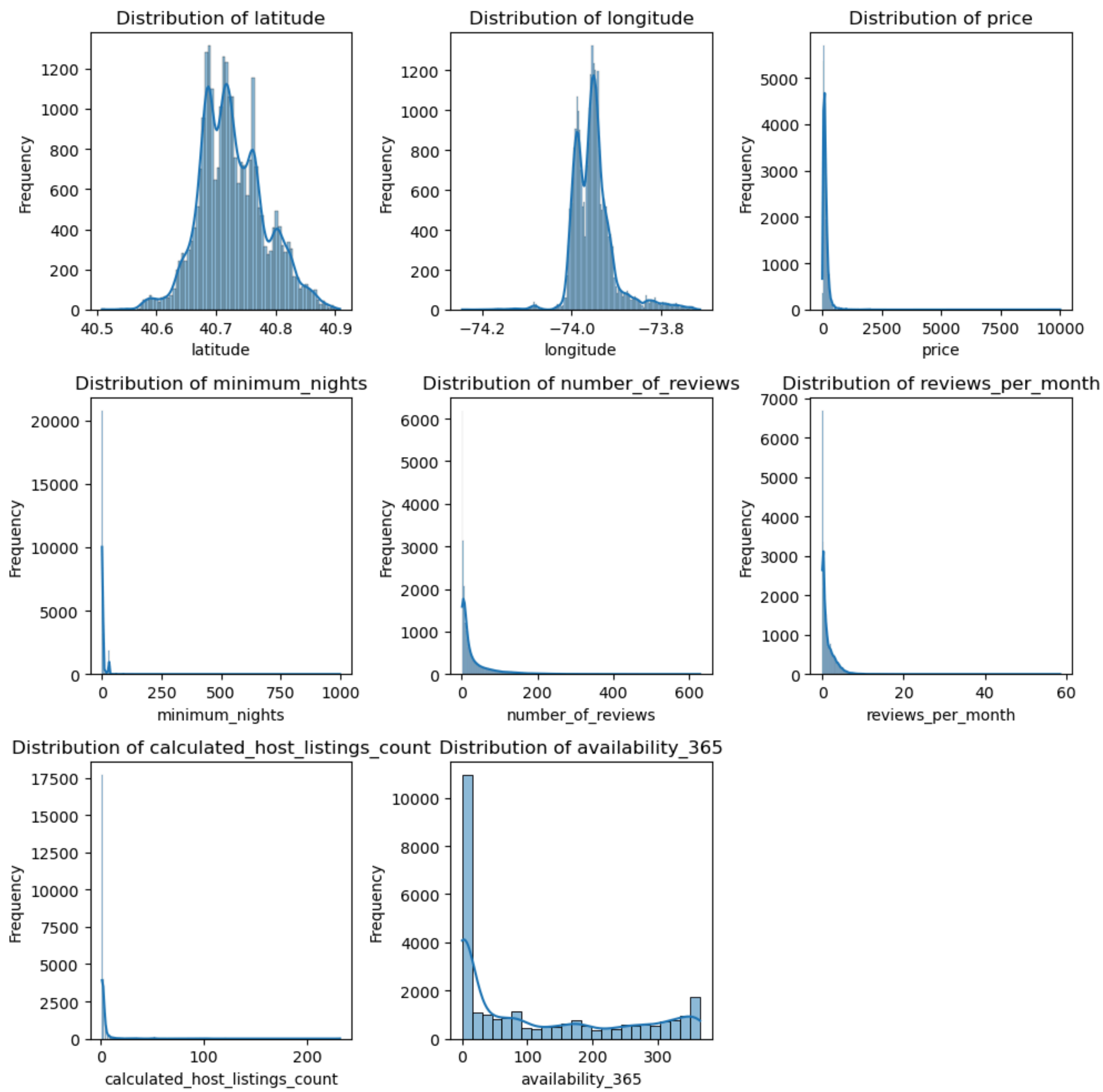
```
In [15]: df_train.groupby("room_type")[TARGET].mean().sort_values(ascending=False)
```

```
Out[15]: room_type
Private room       1.445947
Shared room        1.433863
Entire home/apt    1.297404
Name: reviews_per_month, dtype: float64
```

```
In [16]: df_train["neighbourhood_group"].value_counts(True).sort_values(ascending=True)
```

```
Out[16]: neighbourhood_group
Staten Island    0.008606
Bronx            0.021478
Queens           0.117837
Manhattan        0.421736
Brooklyn         0.430342
Name: proportion, dtype: float64
```

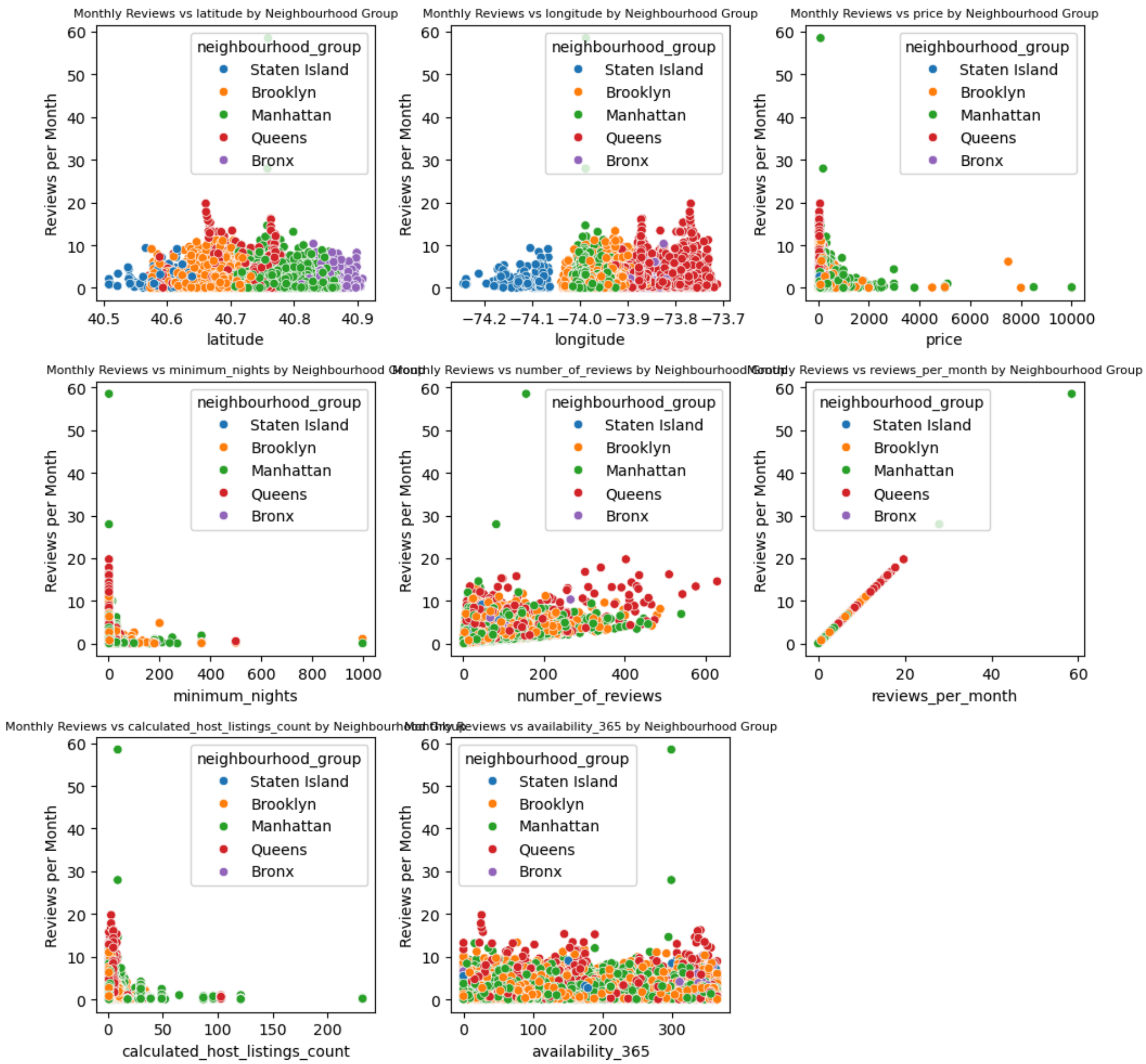
```
In [17]: cont_cols = [c for c in df_train.describe()]

num_cols_per_row = 3
num_rows = int(np.ceil(len(cont_cols) / num_cols_per_row))

# Create the figure and subplots
fig, axes = plt.subplots(num_rows, num_cols_per_row, figsize=(10, 10))
axes = axes.flatten()  # Flatten the 2D array of axes for easier iteration

# Plot histograms for each column
for i, col in enumerate(cont_cols):
    sns.histplot(df_train[col], kde=True, ax=axes[i])
    axes[i].set_title(f"Distribution of {col}")
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Frequency")

# Remove any unused subplots if the number of columns is not a multiple of 3
for j in range(len(cont_cols), len(axes)):
    fig.delaxes(axes[j])

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



In [18]:

```
num_cols_per_row = 3
num_rows = int(np.ceil(len(cont_cols) / num_cols_per_row))

# Create the figure and subplots
fig, axes = plt.subplots(num_rows, num_cols_per_row, figsize=(10, 10))
axes = axes.flatten()  # Flatten the 2D array of axes for easier iteration

# Plot histograms for each column
for i, col in enumerate(cont_cols):
    sns.scatterplot(
        x=df_train[col],
        y=df_train[TARGET],
        hue=df_train["neighbourhood_group"],
        ax=axes[i],
    )
    axes[i].set_title(f"Monthly Reviews vs {col} by Neighbourhood Group", fontsize=
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Reviews per Month")
```

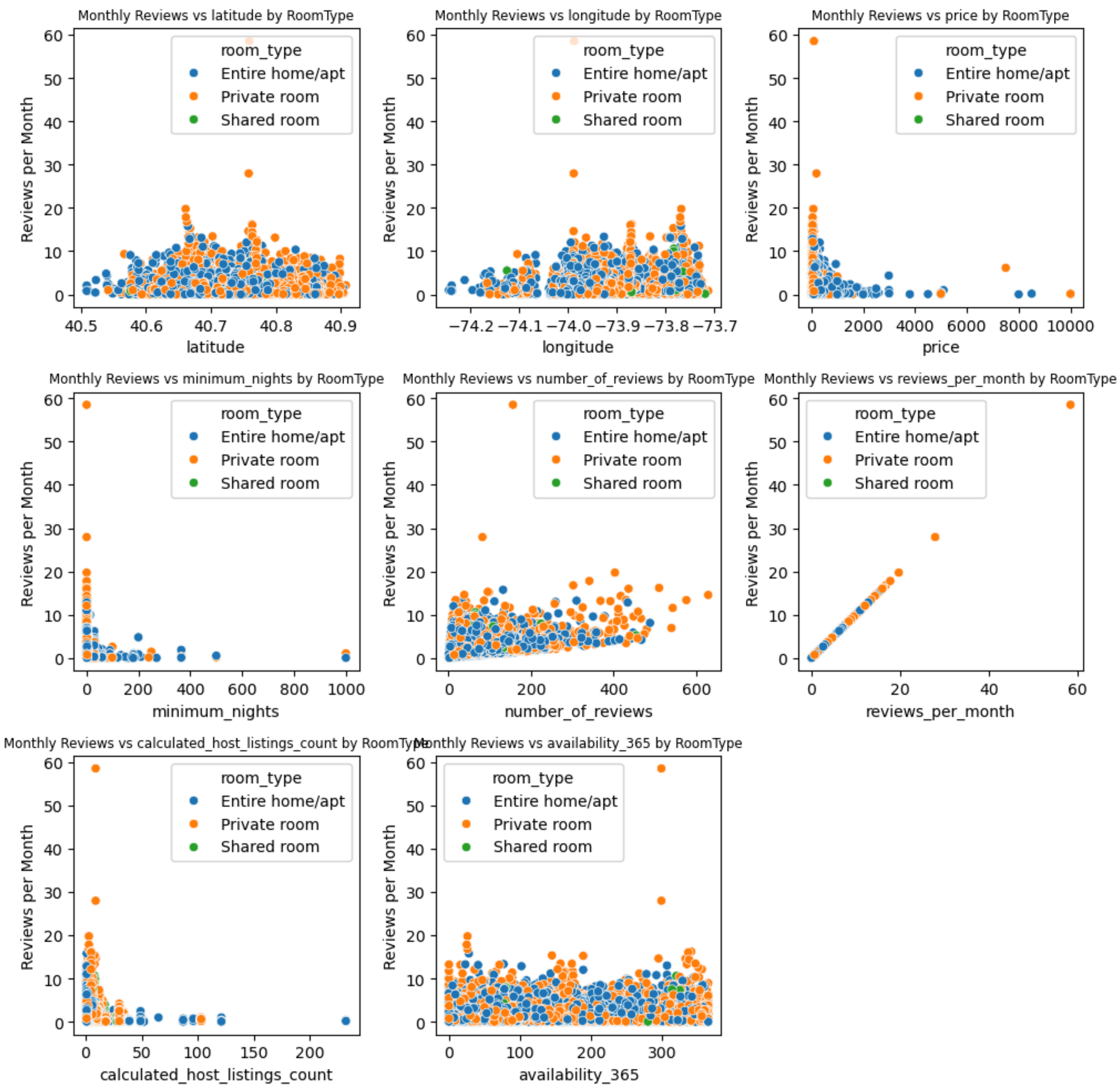
```python
# Remove any unused subplots if the number of columns is not a multiple of 3
for j in range(len(cont_cols), len(axes)):
    fig.delaxes(axes[j])

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



In [19]:

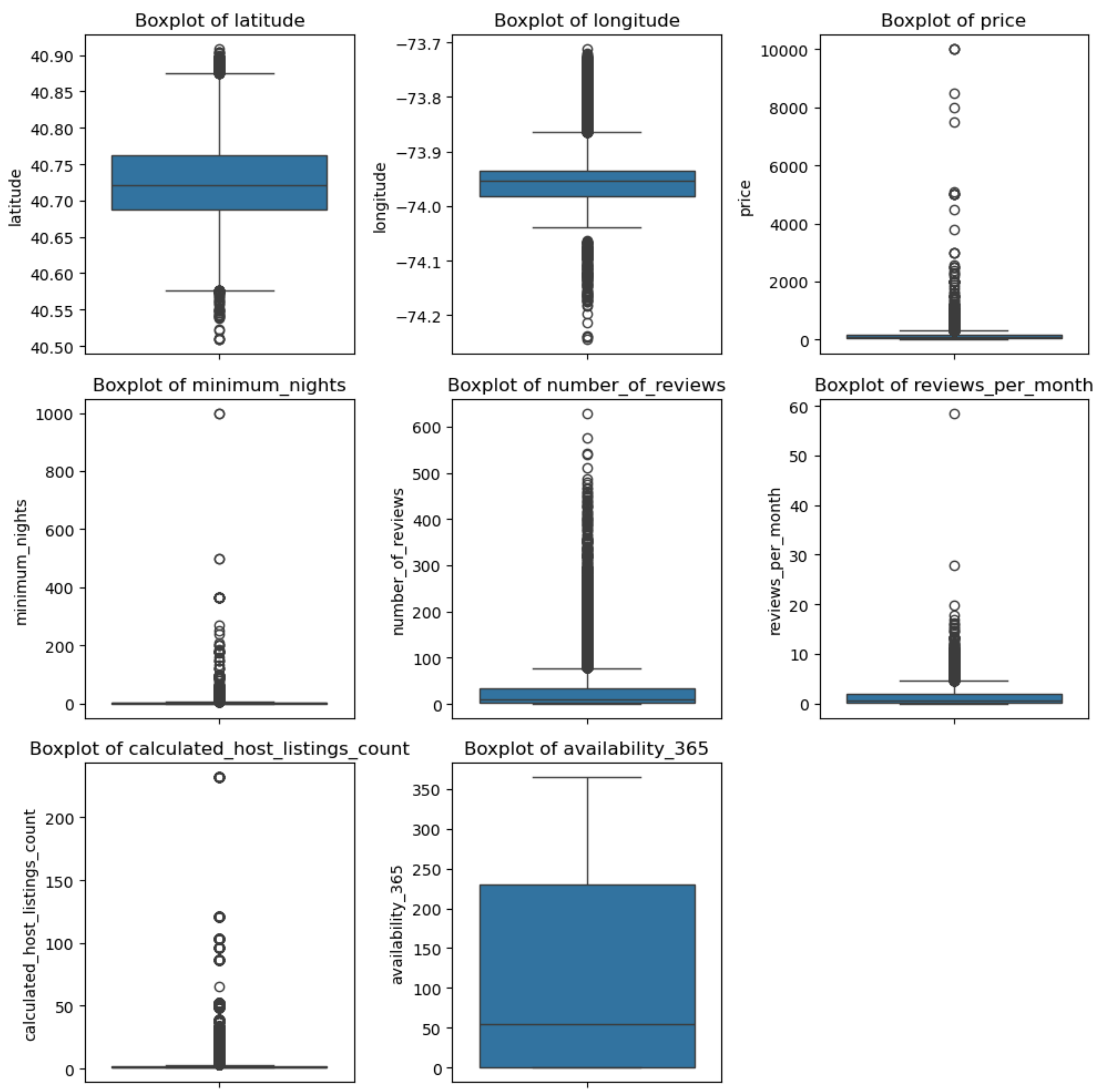
```python
num_cols_per_row = 3
num_rows = int(np.ceil(len(cont_cols) / num_cols_per_row))

# Create the figure and subplots
fig, axes = plt.subplots(num_rows, num_cols_per_row, figsize=(10, 10))
axes = axes.flatten()  # Flatten the 2D array of axes for easier iteration

# Plot histograms for each column
for i, col in enumerate(cont_cols):
    sns.scatterplot(
        x=df_train[col],
        y=df_train[TARGET],
        hue=df_train["room_type"],
        ax=axes[i],
    )
```

```python
    axes[i].set_title(f"Monthly Reviews vs {col} by RoomType",fontsize=8.5)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Reviews per Month")

# Remove any unused subplots if the number of columns is not a multiple of 3
for j in range(len(cont_cols), len(axes)):
    fig.delaxes(axes[j])

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



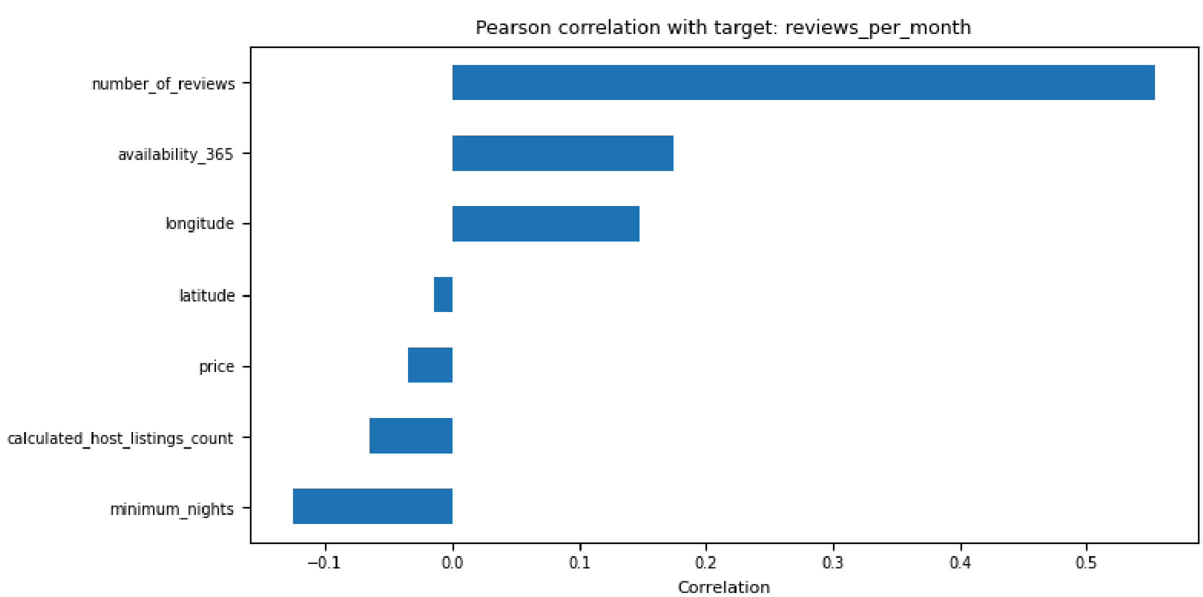
In [20]:
```python
num_cols_per_row = 3
num_rows = int(np.ceil(len(cont_cols) / num_cols_per_row))

# Create the figure and subplots
fig, axes = plt.subplots(num_rows, num_cols_per_row, figsize=(10, 10))
axes = axes.flatten()  # Flatten the 2D array of axes for easier iteration

# Plot histograms for each column
for i, col in enumerate(cont_cols):
```

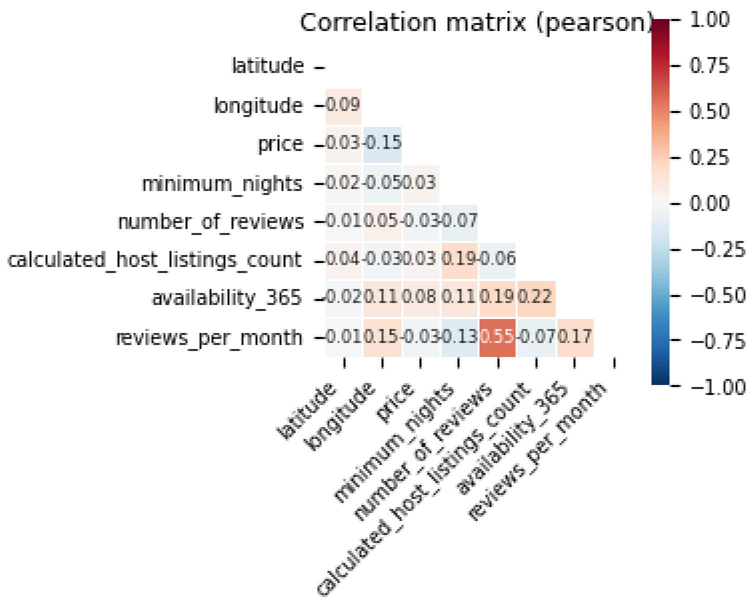
```python
    sns.boxplot(df_train[col], ax=axes[i])
    axes[i].set_title(f"Boxplot of {col}")

# Remove any unused subplots if the number of columns is not a multiple of 3
for j in range(len(cont_cols), len(axes)):
    fig.delaxes(axes[j])

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



In [21]:
```python
plt.rcParams.update({
    "font.size": 8,
    "axes.titlesize": 9,
    "axes.labelsize": 8,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
})

num_cols = df_train.select_dtypes(include="number").columns.tolist()
cont_cols = [c for c in num_cols if c != TARGET]
```

```
method = "pearson"

corr_df = df_train[cont_cols + [TARGET]].corr(method=method)

feat_corr = corr_df[TARGET].drop(TARGET).sort_values()

plt.figure(figsize=(8, max(4, 0.28 * len(feat_corr))))
feat_corr.plot(kind="barh")
plt.title(f"{method.title()} correlation with target: {TARGET}", fontsize=9)
plt.xlabel("Correlation", fontsize=8)
plt.yticks(fontsize=7)
plt.xticks(fontsize=7)
plt.tight_layout()
plt.show()

mask = np.triu(np.ones_like(corr_df, dtype=bool))

plt.figure(figsize=(1 + 0.35 * len(corr_df.columns), 1 + 0.35 * len(corr_df.columns
sns.heatmap(
    corr_df,
    mask=mask,
    center=0,
    vmin=-1, vmax=1,
    cmap="RdBu_r",
    square=True,
    linewidths=0.5,
    annot=True,
    fmt=".2f",
    annot_kws={"size": 6},
    cbar_kws={"shrink": 0.7}
)
plt.title(f"Correlation matrix ({method})", fontsize=9)
plt.xticks(fontsize=7, rotation=45, ha="right")
plt.yticks(fontsize=7)
plt.tight_layout()
plt.show()
```





4. Feature engineering (Challenging)

rubric={reasoning}

Your tasks:

1. Carry out feature engineering. In other words, extract new features relevant for the problem and work with your new feature set in the following exercises. You may have to go back and forth between feature engineering and preprocessing.

Points: 0.5

In [22]:
```
df_train = pd.read_csv("data/split_data/train_set.csv")
df_test = pd.read_csv("data/split_data/test_set.csv")
```

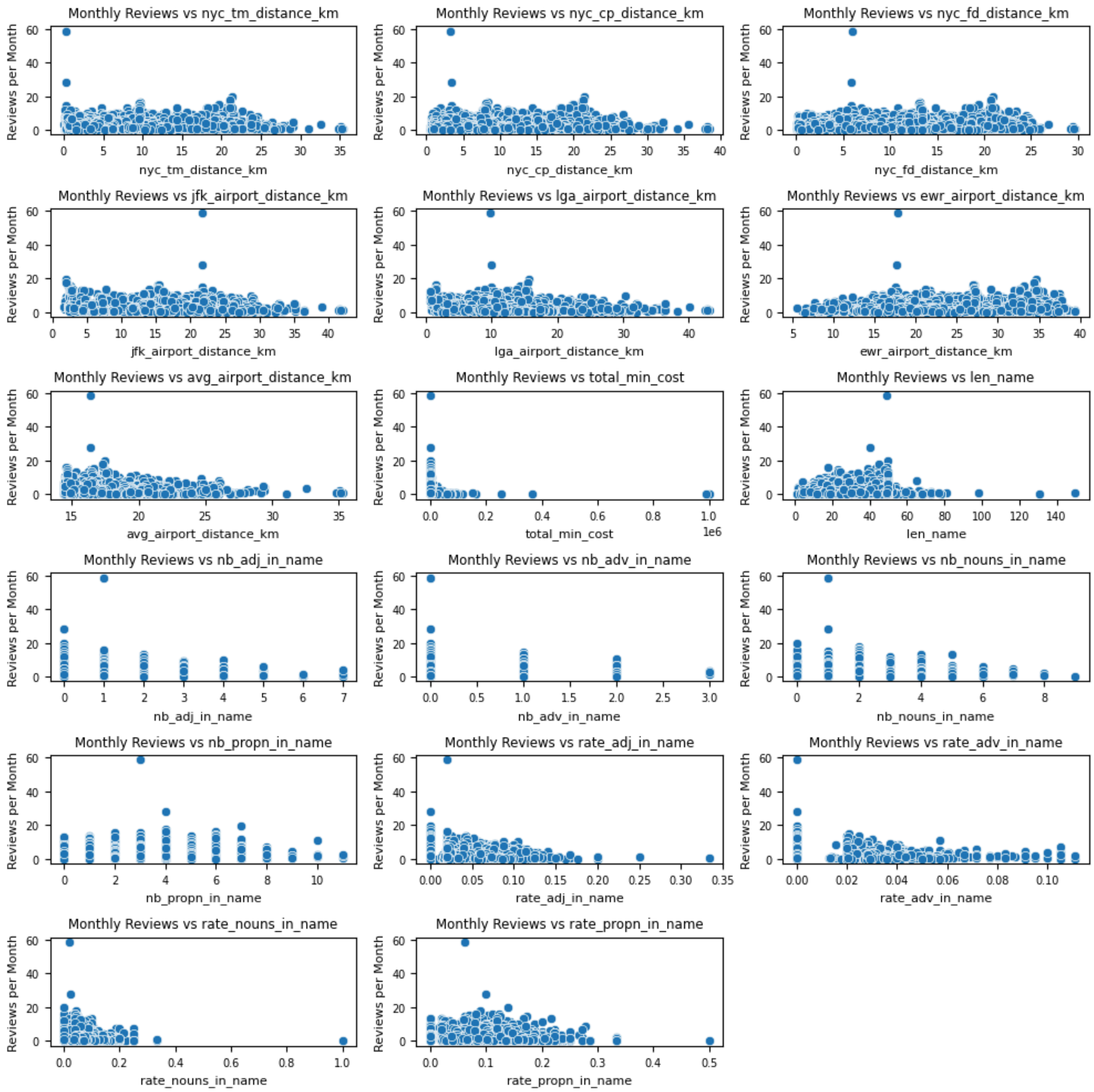
In [23]:
```
df_train = compute_feature_engineering(df_train)
print("\n\nFinished TRAIN feature engineering...\n")
df_test = compute_feature_engineering(df_test)
print("\n\nFinished TEST feature engineering...")
```

```
COMPUTING FEATURE: nyc_tm_distance_km
COMPUTING FEATURE: nyc_cp_distance_km
COMPUTING FEATURE: nyc_fd_distance_km
COMPUTING FEATURE: jfk_airport_distance_km
COMPUTING FEATURE: lga_airport_distance_km
COMPUTING FEATURE: ewr_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km


Finished TRAIN feature engineering...

COMPUTING FEATURE: nyc_tm_distance_km
COMPUTING FEATURE: nyc_cp_distance_km
COMPUTING FEATURE: nyc_fd_distance_km
COMPUTING FEATURE: jfk_airport_distance_km
COMPUTING FEATURE: lga_airport_distance_km
COMPUTING FEATURE: ewr_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km
COMPUTING FEATURE: avg_airport_distance_km


Finished TEST feature engineering...
```

In [24]:

```
new_features = [
    "nyc_tm_distance_km",
    "nyc_cp_distance_km",
    "nyc_fd_distance_km",
    "jfk_airport_distance_km",
    "lga_airport_distance_km",
    "ewr_airport_distance_km",
    "avg_airport_distance_km",
    "total_min_cost",
    "len_name",
    "nb_adj_in_name",
    "nb_adv_in_name",
    "nb_nouns_in_name",
    "nb_propn_in_name",
```

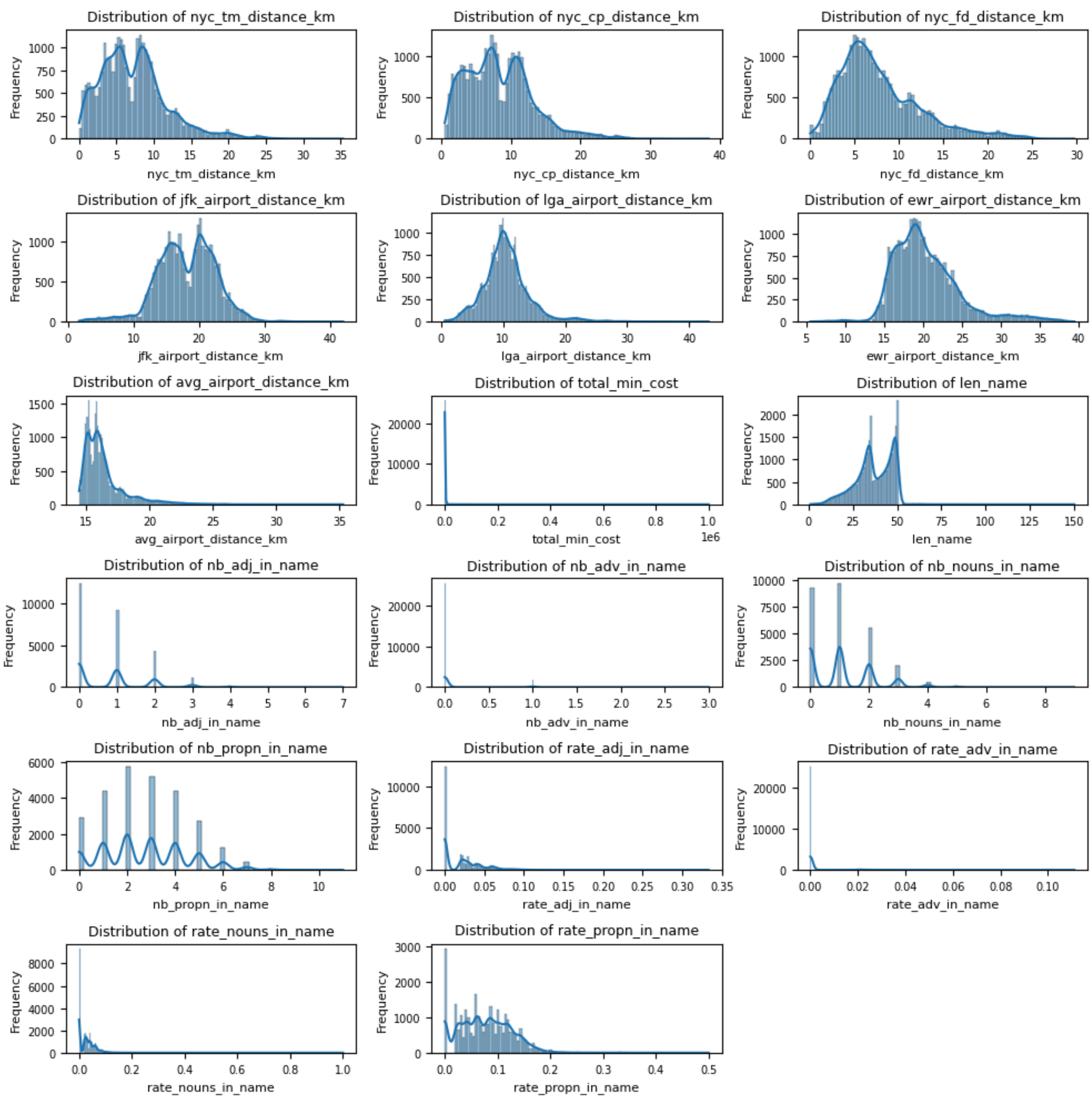
```python
    "rate_adj_in_name",
    "rate_adv_in_name",
    "rate_nouns_in_name",
    "rate_propn_in_name",
]

num_cols_per_row = 3
num_rows = int(np.ceil(len(new_features) / num_cols_per_row))

# Create the figure and subplots
fig, axes = plt.subplots(num_rows, num_cols_per_row, figsize=(10, 10))
axes = axes.flatten()  # Flatten the 2D array of axes for easier iteration

# Plot histograms for each column
for i, col in enumerate(new_features):
    sns.scatterplot(
        x=df_train[col],
        y=df_train[TARGET],
        ax=axes[i],
    )
    axes[i].set_title(f"Monthly Reviews vs {col}",fontsize=8.5)
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Reviews per Month")

# Remove any unused subplots if the number of columns is not a multiple of 3
for j in range(len(new_features), len(axes)):
    fig.delaxes(axes[j])

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



In [25]:

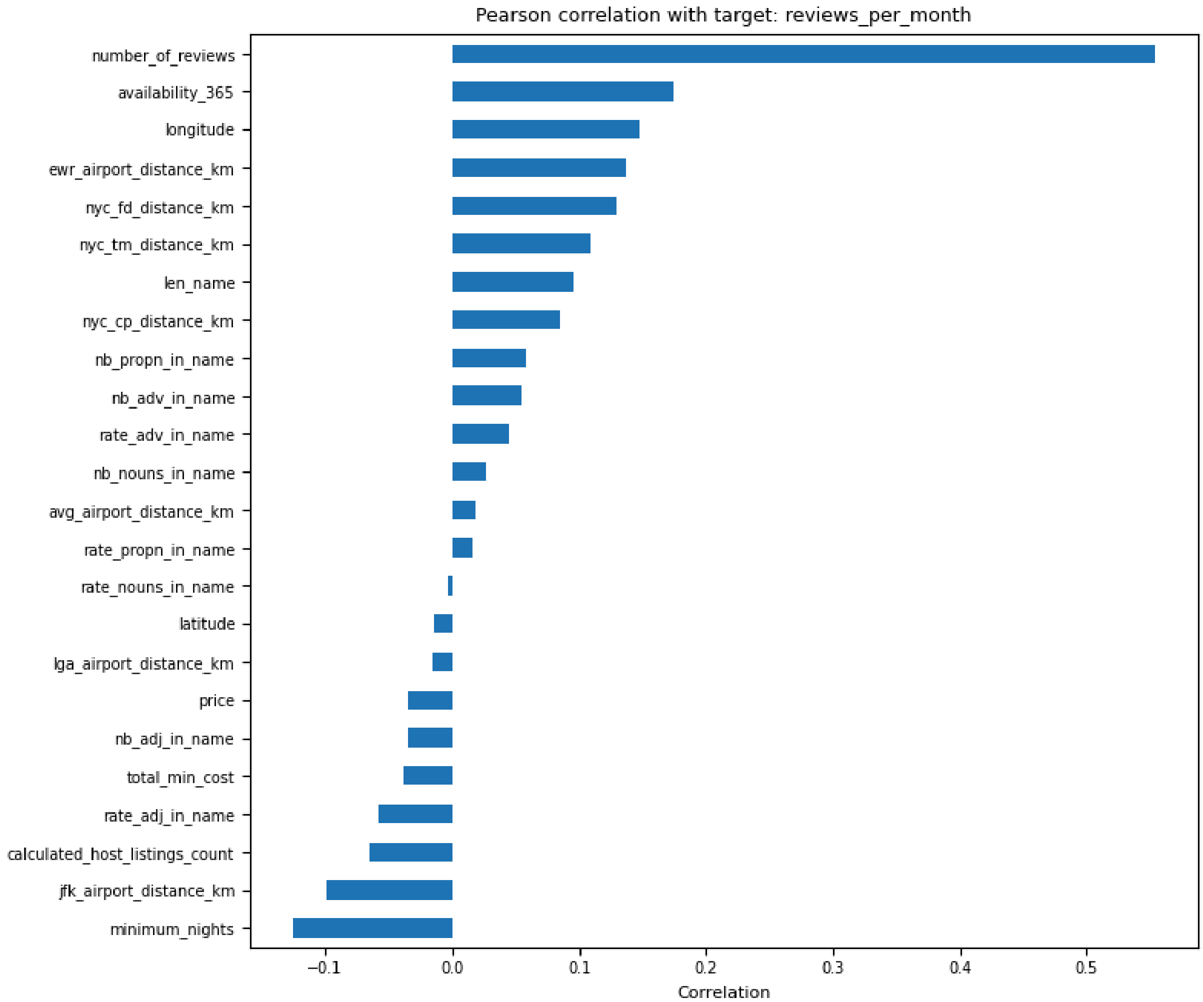
```
num_cols_per_row = 3
num_rows = int(np.ceil(len(new_features) / num_cols_per_row))

# Create the figure and subplots
fig, axes = plt.subplots(num_rows, num_cols_per_row, figsize=(10, 10))
axes = axes.flatten()  # Flatten the 2D array of axes for easier iteration

# Plot histograms for each column
for i, col in enumerate(new_features):
    sns.histplot(df_train[col], kde=True, ax=axes[i])
    axes[i].set_title(f"Distribution of {col}")
    axes[i].set_xlabel(col)
    axes[i].set_ylabel("Frequency")

# Remove any unused subplots if the number of columns is not a multiple of 3
for j in range(len(new_features), len(axes)):
    fig.delaxes(axes[j])
```

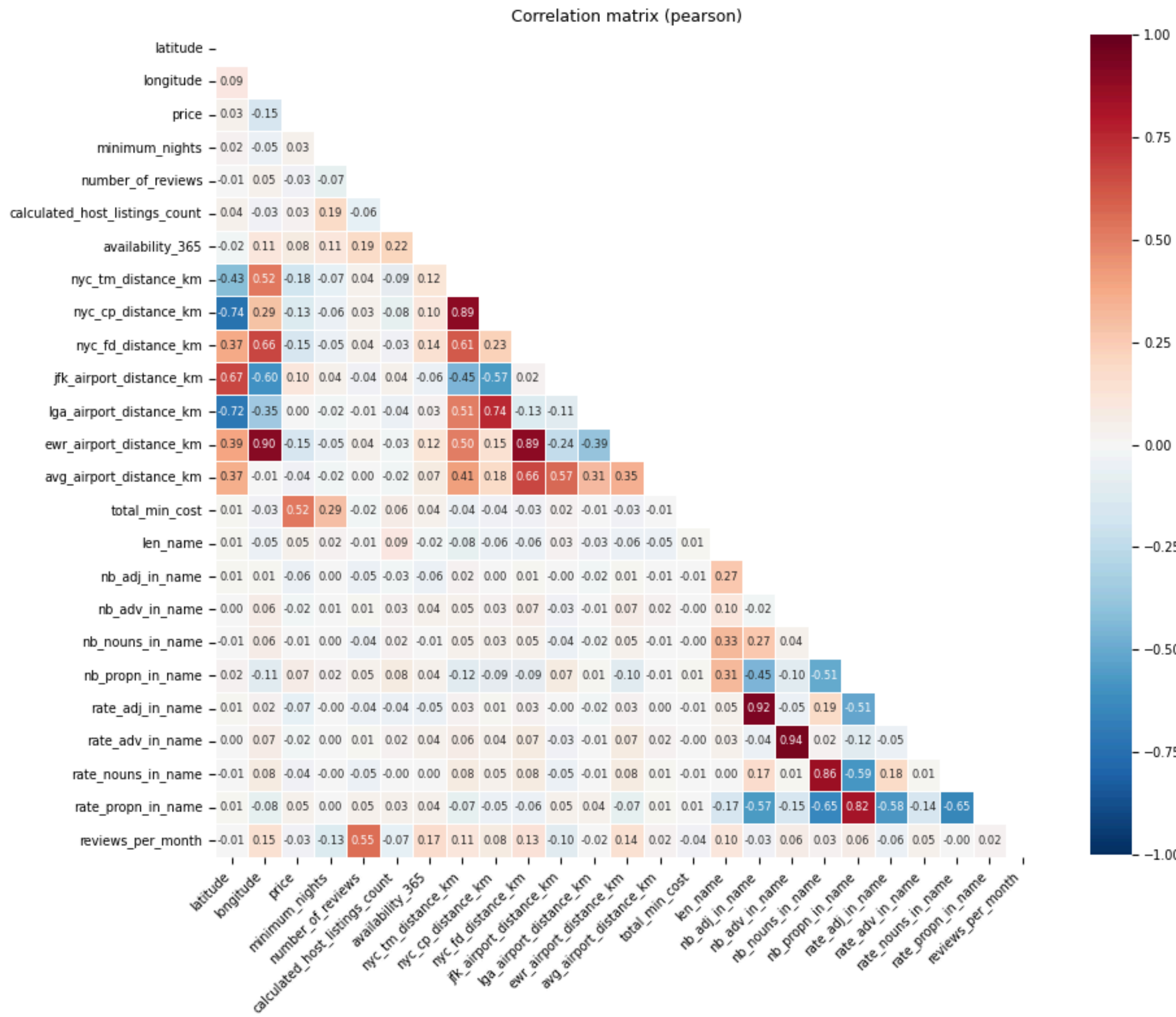
```python
plt.tight_layout()
plt.show()
```



In [26]:
```python
plt.rcParams.update({
    "font.size": 8,
    "axes.titlesize": 9,
    "axes.labelsize": 8,
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
})

num_cols = df_train.select_dtypes(include="number").columns.tolist()
cont_cols = [c for c in num_cols if c != TARGET]

method = "pearson"

corr_df = df_train[cont_cols + [TARGET]].corr(method=method)

feat_corr = corr_df[TARGET].drop(TARGET).sort_values()
```

```
plt.figure(figsize=(8, max(4, 0.28 * len(feat_corr))))
feat_corr.plot(kind="barh")
plt.title(f"{method.title()} correlation with target: {TARGET}", fontsize=9)
plt.xlabel("Correlation", fontsize=8)
plt.yticks(fontsize=7)
plt.xticks(fontsize=7)
plt.tight_layout()
plt.show()

mask = np.triu(np.ones_like(corr_df, dtype=bool))

plt.figure(figsize=(1 + 0.35 * len(corr_df.columns), 1 + 0.35 * len(corr_df.columns
sns.heatmap(
    corr_df,
    mask=mask,
    center=0,
    vmin=-1, vmax=1,
    cmap="RdBu_r",
    square=True,
    linewidths=0.5,
    annot=True,
    fmt=".2f",
    annot_kws={"size": 6},
    cbar_kws={"shrink": 0.7}
)
plt.title(f"Correlation matrix ({method})", fontsize=9)
plt.xticks(fontsize=7, rotation=45, ha="right")
plt.yticks(fontsize=7)
plt.tight_layout()
plt.show()
```


Pearson correlation with target: reviews_per_month
number_of_reviews
availability_365
longitude
ewr_airport_distance_km
nyc_fd_distance_km
nyc_tm_distance_km
len_name
nyc_cp_distance_km
nb_propn_in_name
nb_adv_in_name
rate_adv_in_name
nb_nouns_in_name
avg_airport_distance_km
rate_propn_in_name
rate_nouns_in_name
latitude
lga_airport_distance_km
price
nb_adj_in_name
total_min_cost
rate_adj_in_name
calculated_host_listings_count
jfk_airport_distance_km
minimum_nights
−0.1
0.0
0.1
0.2
0.3
0.4
0.5
Correlation



We performed feature engineering to improve the predictive power of the model. The new features include:

- Distance Features: We calculated distances (in kilometers) from each listing to key NYC landmarks and transportation hubs, including Times Square, Central Park, the Financial District, and three major airports (JFK, LaGuardia, Newark). We also computed the average distance to all airports. Proximity to tourist attractions and transportation hubs is likely to affect booking frequency, which impacts the number of reviews received.
- Total Minimum Cost: This feature captures the actual financial commitment required to book a listing by multiplying price by minimum_nights. This distinguishes between, for example, a low-cost listing with a longer day minimum stay versus an expensive listing with a one day minimum, providing a more realistic measure of booking accessibility.
- Text Features (NLP from Listing Names): We extracted linguistic features from listing names using spaCy, including the length of the name, counts and proportions of different parts of speech (adjectives, adverbs, nouns, proper nouns). Listing names with more descriptive language ("cozy", "spacious") or specific location mentions (proper nouns like "Brooklyn", "Midtown") may attract more bookings and by consequence more reviews.

5. Preprocessing and transformations

rubric={accuracy,reasoning}

Your tasks:

1. Identify different feature types and the transformations you would apply on each feature type.
2. Define a column transformer, if necessary.

Points: 4

In [27]:
```
cols_w_outliers = [
    "price",
    "minimum_nights",
    "reviews_per_month",
    "calculated_host_listings_count",
]
df_train[cols_w_outliers].describe(percentiles=[0.5,0.75,0.85,0.90,0.95,0.97,0.99,0
```

Out[27]:

	price	minimum_nights	reviews_per_month	calculated_host_listings_count
count	27190.000000	27190.000000	27190.000000	27190.000000
mean	142.489702	5.790953	1.368443	3.478301
std	206.368954	16.108038	1.706042	12.036135
min	0.000000	1.000000	0.010000	1.000000
50%	100.000000	2.000000	0.700000	1.000000
75%	169.000000	4.000000	2.000000	2.000000
85%	200.000000	7.000000	3.000000	3.000000
90%	250.000000	14.000000	3.630000	5.000000
95%	330.000000	30.000000	4.675500	9.000000
97%	400.000000	30.000000	5.420000	17.000000
99%	699.000000	31.000000	7.301100	52.000000
99.5%	900.000000	55.165000	8.560550	96.000000
max	10000.000000	999.000000	58.500000	232.000000

In [28]:
```
price_outlier_cond = df_train["price"] < np.percentile(df_train["price"], 99)
minimum_nights_outlier_cond = df_train["minimum_nights"] <= np.percentile(
    df_train["minimum_nights"], 99
)
```

```
reviews_per_month_outlier_cond = df_train["reviews_per_month"] <= np.percentile(
    df_train["reviews_per_month"], 99.5
)
listing_count_outlier_cond = df_train["calculated_host_listings_count"] < np.percen
    df_train["calculated_host_listings_count"], 99
)
```

In [29]:
```
df_train = df_train.loc[
    price_outlier_cond &
    minimum_nights_outlier_cond &
    reviews_per_month_outlier_cond &
    listing_count_outlier_cond
].copy()
```

In [30]:
```
df_train[cols_w_outliers].describe(percentiles=[0.5,0.75,0.85,0.90,0.95,0.97,0.99,0
```

Out[30]:

	price	minimum_nights	reviews_per_month	calculated_host_listings_count
count	26311.000000	26311.000000	26311.000000	26311.000000
mean	130.162442	4.781992	1.342145	2.424575
std	91.830858	7.152393	1.535088	4.568263
min	0.000000	1.000000	0.010000	1.000000
50%	100.000000	2.000000	0.720000	1.000000
75%	165.000000	4.000000	2.010000	2.000000
85%	200.000000	6.000000	2.990000	3.000000
90%	250.000000	9.000000	3.600000	4.000000
95%	300.000000	30.000000	4.570000	7.000000
97%	360.000000	30.000000	5.200000	11.000000
99%	499.000000	30.000000	6.589000	30.000000
99.5%	550.000000	30.000000	7.330000	34.000000
max	695.000000	31.000000	8.560000	50.000000

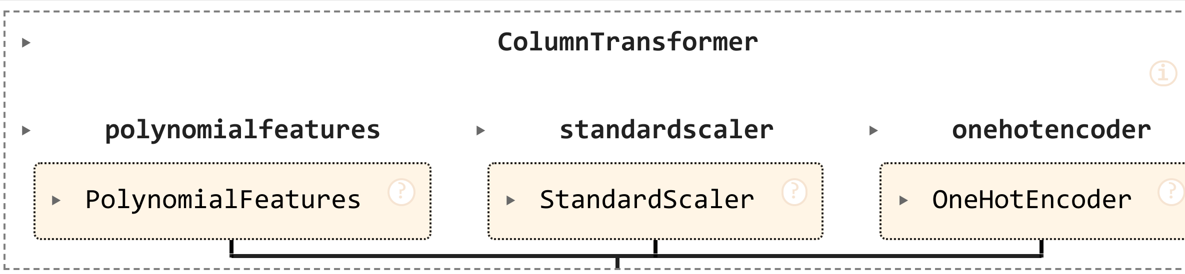
In [31]:
```
features = [
    'neighbourhood_group',
    'latitude',
    'longitude',
    'room_type',
    'price',
    'minimum_nights',
    'calculated_host_listings_count',
    'availability_365',
    'nyc_tm_distance_km',
    'nyc_cp_distance_km',
    'nyc_fd_distance_km',
    'jfk_airport_distance_km',
```

```
    'lga_airport_distance_km',
    'ewr_airport_distance_km',
    'avg_airport_distance_km',
    'total_min_cost',
    'len_name',
    'nb_adj_in_name',
    'nb_adv_in_name',
    'nb_nouns_in_name',
    'nb_propn_in_name',
    'rate_adj_in_name',
    'rate_adv_in_name',
    'rate_nouns_in_name',
    'rate_propn_in_name'
]
categorical_features = [
    'neighbourhood_group',
    'room_type',
]
cont_features = [
    f for f in features if f not in categorical_features
]
```

In [32]:
```
df_train.to_csv("data/preprocessed/df_train.csv", index=False)
df_test.to_csv("data/preprocessed/df_test.csv", index=False)
```

In [33]:
```
general_preprocessor = make_column_transformer(
    (PolynomialFeatures(degree=2),cont_features),
    (StandardScaler(), cont_features),
    (OneHotEncoder(drop='first', handle_unknown="ignore", sparse_output=False), cat
)
general_preprocessor.set_output(transform="pandas")
general_preprocessor
```

Out[33]:



In [34]:
```
X_train = df_train[features].copy()
y_train = df_train[TARGET].copy()
X_test = df_test[features].copy()
y_test = df_test[TARGET].copy()

X_train_s = general_preprocessor.fit_transform(X_train)
X_test_s = general_preprocessor.transform(X_test)
```

Preprocessing Steps

a) Outlier Removal: We removed extreme outliers from the training set by capping features at (99 or 99.5th percentile): price, minimum_nights, reviews_per_month, and calculated_host_listing_count.

b) Feature Scaling: We applied "StandardScaler" to all continuous features to ensure features are on comparable scales.

c) Categorical Encoding: Applied "OneHotEncoder" with drop='first' to try to avoid multicollinearity and handle_unknown='ignore' for robustness to new categories in test data.

d) Final Feature Set: 25 features total (23 continuous, 2 categorical → ~26 after one-hot encoding).

6. Baseline model

rubric={accuracy}

Your tasks:

1. Train a baseline model for your task and report its performance.

Points: 2

In [35]:
```
from sklearn.dummy import DummyRegressor

dummy_regr = DummyRegressor(strategy="mean")
dummy_regr.fit(X_train_s, y_train)

y_train_pred = dummy_regr.predict(X_train_s)
y_test_pred = dummy_regr.predict(X_test_s)

print("BASELINE MODEL RESULTS:")
print("TRAIN MSE", round(mean_squared_error(y_true=y_train, y_pred=y_train_pred), 3
print("TEST MSE", round(mean_squared_error(y_true=y_test, y_pred=y_test_pred), 3))

print("TRAIN R2", round(r2_score(y_true=y_train, y_pred=y_train_pred), 3))
print("TEST R2", round(r2_score(y_true=y_test, y_pred=y_test_pred), 3))

print(
    "TRAIN RMSE", round(root_mean_squared_error(y_true=y_train, y_pred=y_train_pred
)
print("TEST RMSE", round(root_mean_squared_error(y_true=y_test, y_pred=y_test_pred)
```

```
BASELINE MODEL RESULTS:
TRAIN MSE 2.356
TEST MSE 2.623
TRAIN R2 0.0
TEST R2 -0.001
TRAIN RMSE 1.535
TEST RMSE 1.62
```

We chose a baseline using DummyRegressor with a mean prediction strategy, which predicts the training set mean for all observations.

This baseline represents the simplest possible model and provides a benchmark for evaluating more sophisticated approaches. Any model with R squared > 0 demonstrates improvement over this naive baseline. The RMSE of approximately 1.6 represents the standard deviation of the target variable, indicating that on average, predictions are off by about 1.6 reviews per month when simply predicting the mean.

7. Linear models

rubric={accuracy,reasoning}

Your tasks:

1. Try a linear model as a first real attempt.
2. Carry out hyperparameter tuning to explore different values for the regularization hyperparameter.
3. Report cross-validation scores along with standard deviation.
4. Summarize your results.

Points: 8

Our Ridge regression model is performing not well with R squared = 0.253, meaning only 25% of variance is explained. Your interpretation is on track - there's likely significant omitted variable bias (OVB).

The main problems that we see are:

a) R squared: Test determination coefficient of 0.253 is indeed weak. The model is missing some important predictors. b) Also, there is a high alpha (266,322): This extreme regularization is shrinking coefficients heavily, suggesting multicollinearity and that our features aren't predictive. c) We have very tiny coefficients and as are sclaed this means features have minimal impact.

Interpretation of Top Polynomial Features:

- Minimum Nights**2 (0.0054): A one-unit increase in minimum_nights to the square increases the target by 0.0054. The squared term suggests a non linear relationship that is extremely long minimum stay requirements might have diminishing or accelerating effects on your target variable.
- Nyc_fd_distance_km * Jfk_airport_distance_km (0.0048): This interaction suggests that being far from both Fire Department landmarks and JFK airport simultaneously has a small positive effect. This could capture outer borough or remote listings.

In [41]:
```
from sklearn.linear_model import Ridge
```

```python
raw_ridge = make_pipeline(
    general_preprocessor,
    Ridge()
)
```

In [58]:
```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform, uniform, randint

param_grid = {
    "ridge__alpha": loguniform(1e-10, 1e10),
}

random_search_ridge = RandomizedSearchCV(
    raw_ridge,
    param_distributions=param_grid,
    n_iter=100,
    n_jobs= -1,
    random_state=RANDOM_STATE,
    return_train_score=True,
)

random_search_ridge.fit(X_train, y_train)

print(
    "Random Search best model score: \t %0.3f" % random_search_ridge.best_score_
)
print(
    "Random Search best alpha: \t\t\t %0.3f" %
    random_search_ridge.best_params_["ridge__alpha"]
)


pd.DataFrame(random_search_ridge.cv_results_)[
    [
        "mean_test_score",
        "param_ridge__alpha",
        "mean_fit_time",
        "rank_test_score",
    ]
].set_index("rank_test_score").sort_index()
```

```
Random Search best model score: 	 0.253
Random Search best alpha: 			 266321.933
```

Out[58]:

	mean_test_score	param_ridge__alpha	mean_fit_time
rank_test_score			
1	0.252686	2.663219e+05	1.300377
2	0.252686	2.626296e+05	1.206110
3	0.252686	2.785450e+05	1.187796
4	0.252680	3.181680e+05	1.190700
5	0.252632	1.643240e+05	1.406712
...	...	...	...
96	0.241700	1.033442e+09	1.023568
97	0.240047	2.054190e+09	1.244403
98	0.239541	2.464294e+09	1.213482
99	0.239498	2.501480e+09	1.114498
100	0.236836	5.466870e+09	1.231342

100 rows × 3 columns

In [53]:

```
from sklearn.model_selection import cross_validate

def mean_std_cross_val_scores(model, X_train, y_train, scoring, **kwargs):
    scores = cross_validate(
        model,
        X_train,
        y_train,
        return_train_score=True,
        scoring=scoring,
        **kwargs
    )
    df = pd.DataFrame(scores)
    return df.mean(), df.std()


score_types_reg = {
    "neg_root_mean_squared_error": "neg_root_mean_squared_error",
    "r2": "r2",
}

results_mean, results_std = {}, {}

mean, std = mean_std_cross_val_scores(
    random_search_ridge.best_estimator_, X_train, y_train, scoring=score_types_reg
)
results_mean["Ridge"] = mean
results_std["Ridge"] = std

mean_df = pd.DataFrame(results_mean).T
```

```python
std_df = pd.DataFrame(results_std).T
results_df = mean_df.round(3).astype(str) + " ± " + std_df.round(3).astype(str)
results_df.T
```

Out[53]:

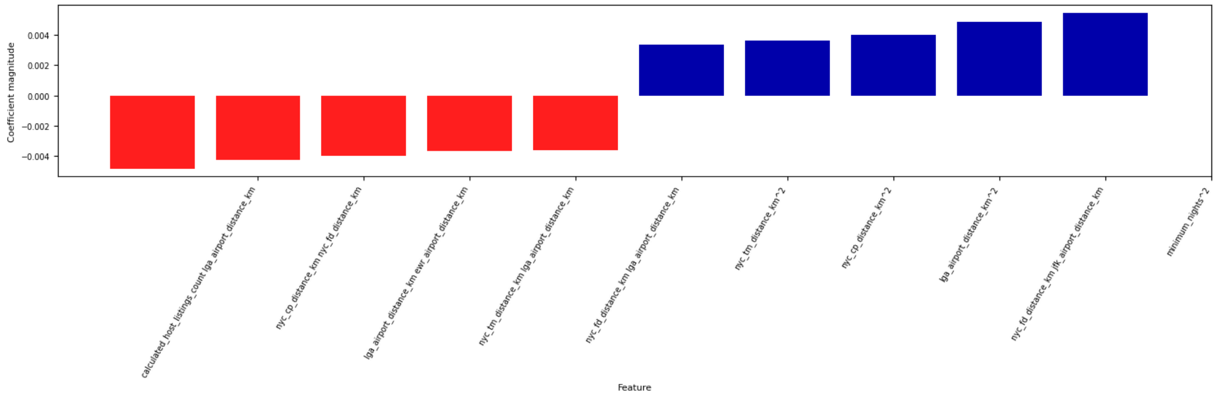
	Ridge
fit_time	0.183 ± 0.028
score_time	0.037 ± 0.007
test_neg_root_mean_squared_error	-1.327 ± 0.01
train_neg_root_mean_squared_error	-1.314 ± 0.003
test_r2	0.253 ± 0.011
train_r2	0.268 ± 0.002

In [72]:

```python
import mglearn

best_pipe = random_search_ridge.best_estimator_

ridge = best_pipe.named_steps["ridge"]
coef = np.ravel(ridge.coef_)

feature_names = best_pipe[:-1].get_feature_names_out()

coef_df = pd.DataFrame({"feature": feature_names, "coefficient": coef})
coef_df["magnitude"] = coef_df["coefficient"].abs()

split = coef_df["feature"].str.split("__", n=1, expand=True)


if split.shape[1] == 2:
    coef_df["source"] = split[0]
    coef_df["feature_clean"] = split[1]
else:
    coef_df["source"] = None
    coef_df["feature_clean"] = split[0]

top30 = (coef_df.sort_values("magnitude", ascending=False)
               .head(30)
               .reset_index(drop=True))

display(top30[["feature_clean", "source", "coefficient", "magnitude"]])


mglearn.tools.visualize_coefficients(
    coef,
    coef_df["feature_clean"].to_numpy(),
    n_top_features=5
)
plt.tight_layout()
plt.show()
```

	feature_clean	source	coefficient	magnitude
0	minimum_nights^2	polynomialfeatures	0.005445	0.005445
1	nyc_fd_distance_km jfk_airport_distance_km	polynomialfeatures	0.004828	0.004828
2	calculated_host_listings_count lga_airport_dis...	polynomialfeatures	-0.004824	0.004824
3	nyc_cp_distance_km nyc_fd_distance_km	polynomialfeatures	-0.004268	0.004268
4	lga_airport_distance_km ewr_airport_distance_km	polynomialfeatures	-0.003994	0.003994
5	lga_airport_distance_km^2	polynomialfeatures	0.003979	0.003979
6	nyc_tm_distance_km lga_airport_distance_km	polynomialfeatures	-0.003644	0.003644
7	nyc_cp_distance_km^2	polynomialfeatures	0.003632	0.003632
8	nyc_fd_distance_km lga_airport_distance_km	polynomialfeatures	-0.003599	0.003599
9	nyc_tm_distance_km^2	polynomialfeatures	0.003341	0.003341
10	jfk_airport_distance_km^2	polynomialfeatures	0.003245	0.003245
11	nyc_fd_distance_km nb_adv_in_name	polynomialfeatures	-0.003179	0.003179
12	calculated_host_listings_count nb_adj_in_name	polynomialfeatures	0.002957	0.002957
13	len_name nb_propn_in_name	polynomialfeatures	0.002812	0.002812
14	nyc_fd_distance_km^2	polynomialfeatures	0.002794	0.002794
15	calculated_host_listings_count nyc_fd_distance_km	polynomialfeatures	-0.002732	0.002732
16	calculated_host_listings_count jfk_airport_dis...	polynomialfeatures	0.002645	0.002645
17	nyc_cp_distance_km ewr_airport_distance_km	polynomialfeatures	-0.002590	0.002590
18	price rate_propn_in_name	polynomialfeatures	0.002547	0.002547
19	minimum_nights nyc_tm_distance_km	polynomialfeatures	0.002510	0.002510
20	len_name nb_adj_in_name	polynomialfeatures	0.002282	0.002282
21	room_type_Private room	onehotencoder	-0.002277	0.002277
22	nyc_tm_distance_km jfk_airport_distance_km	polynomialfeatures	-0.002241	0.002241
23	jfk_airport_distance_km nb_adj_in_name	polynomialfeatures	0.002224	0.002224
24	minimum_nights calculated_host_listings_count	polynomialfeatures	-0.002194	0.002194
25	minimum_nights nyc_fd_distance_km	polynomialfeatures	-0.002100	0.002100
26	nyc_tm_distance_km nb_adj_in_name	polynomialfeatures	-0.001986	0.001986
27	longitude jfk_airport_distance_km	polynomialfeatures	0.001956	0.001956
28	availability_365 rate_propn_in_name	polynomialfeatures	0.001950	0.001950

	feature_clean	source	coefficient	magnitude
29	nyc_cp_distance_km jfk_airport_distance_km	polynomialfeatures	-0.001918	0.001918



8. Different models

rubric={accuracy,reasoning}

Your tasks:

1. Try out three other models aside from the linear model.
2. Summarize your results in terms of overfitting/underfitting and fit and score times. Can you beat the performance of the linear model?

Points: 10

Now, we decide to compare three models in order to improve the predictive performance and we have the following:

1. The best model performance (by the R squared on test set) is Gradient Boosting (GBR) with 0.369, then Random Forest (0.338), Ridge (0.253) and Decision Tree is the worst (0.30).

So, we can say that GBR improves the previous Ridge because it can explains 37% of the variance of the model and also has the lowest prediction error (RMSE of 1.219). Also, it has overfitting but not so much (0.563 in train vs 0.369 on test). Random Forest is also close but it has a great overfit (0.847 in train but in test 0.338).

In summary, with that we can be certain that Ridge is not so great because it underfits and needs a lot of regularization. Decision Tree is the worst model of all (performing worse than just predicting the mean).

In [57]:
```
from sklearn.ensemble import HistGradientBoostingRegressor

models = {
```

```python
    "DecisionTree": DecisionTreeRegressor(
        random_state=RANDOM_STATE,
        max_depth=200,
        max_features=100
    ),
    "RandomForest": RandomForestRegressor(
        random_state=RANDOM_STATE,
        max_depth=20,
        n_estimators=50,
        n_jobs=-1,
        max_features=100,
    ),
    "GBR":  HistGradientBoostingRegressor(
        max_depth=8,
        learning_rate=0.05,
        max_iter=300,
        random_state=RANDOM_STATE
    )
}

for name, model in models.items():
    pipe = make_pipeline(
        general_preprocessor,
        model
    )
    mean, std = mean_std_cross_val_scores(
        pipe, X_train, y_train, scoring=score_types_reg
    )
    results_mean[name] = mean
    results_std[name] = std
    print(name)

mean_df = pd.DataFrame(results_mean).T
std_df = pd.DataFrame(results_std).T
results_df = mean_df.round(3).astype(str) + " ± " + std_df.round(3).astype(str)
results_df.T
```

```
DecisionTree
RandomForest
GBR
```

Out[57]:

	Ridge	RandomForest	DecisionTree	GBR
fit_time	0.183 ± 0.028	12.006 ± 0.815	3.04 ± 0.09	8.828 ± 0.867
score_time	0.037 ± 0.007	0.077 ± 0.01	0.038 ± 0.011	0.109 ± 0.021
test_neg_root_mean_squared_error	-1.327 ± 0.01	-1.248 ± 0.018	-1.75 ± 0.012	-1.219 ± 0.018
train_neg_root_mean_squared_error	-1.314 ± 0.003	-0.601 ± 0.011	-0.0 ± 0.0	-1.015 ± 0.025
test_r2	0.253 ± 0.011	0.338 ± 0.008	-0.3 ± 0.024	0.369 ± 0.007
train_r2	0.268 ± 0.002	0.847 ± 0.005	1.0 ± 0.0	0.563 ± 0.021

9. Feature selection (Challenging)

rubric={reasoning}

Your tasks:

Make some attempts to select relevant features. You may try `RFECV`, forward/backward selection or L1 regularization for this. Do the results improve with feature selection? Summarize your results. If you see improvements in the results, keep feature selection in your pipeline. If not, you may abandon it in the next exercises unless you think there are other benefits with using fewer features.

Points: 0.5

Type your answer here, replacing this text.

In [14]:

```python
X_train_soft = X_train.copy()

soft_features = [
    "neighbourhood_group",
    "latitude",
    "longitude",
    "room_type",
    "price",
    "minimum_nights",
    "calculated_host_listings_count",
    "availability_365",
    "nyc_tm_distance_km",
    "nyc_cp_distance_km",
    "nyc_fd_distance_km",
    "jfk_airport_distance_km",
```

```
    "lga_airport_distance_km",
    "ewr_airport_distance_km",
    "avg_airport_distance_km",
    "total_min_cost",
    "len_name",
    "nb_adj_in_name",
    "nb_adv_in_name",
    "nb_nouns_in_name",
    "nb_propn_in_name",
    "rate_adj_in_name",
    "rate_adv_in_name",
    "rate_nouns_in_name",
    "rate_propn_in_name",
]

soft_cont_features = [f for f in soft_features if f not in categorical_features]
soft_preprocessor = make_column_transformer(
    (PolynomialFeatures(degree=2), cont_features),
    (StandardScaler(), cont_features),
    (
        OneHotEncoder(drop="first", handle_unknown="ignore", sparse_output=False),
        categorical_features,
    ),
)
soft_preprocessor.set_output(transform="pandas")
X_train_soft_s = soft_preprocessor.fit_transform(X_train_soft)
X_train_soft_s["___RANDOM_1____"] = np.random.rand(len(X_train_soft))
X_train_soft_s["___RANDOM_2____"] = np.random.rand(len(X_train_soft))
soft_rf = RandomForestRegressor(n_jobs=-1, random_state=RANDOM_STATE, max_depth=6)
soft_rf.fit(X_train_soft_s, y_train)
```

Out[14]:

```
▾ RandomForestRegressor
▶ Parameters
```

In [15]:

```
explainer = shap.TreeExplainer(soft_rf)
shap_values = explainer.shap_values(X_train_soft_s)
```

In [32]:

```
random_feats = ["___RANDOM_1____", "___RANDOM_2____"]
returned_features = soft_preprocessor.get_feature_names_out()
clean_features = [
    v.split("__", 1)[1] for v in returned_features if v not in random_feats
] + random_feats
ranked_features, selected_features = get_important_features_important_than(
    shap_values, clean_features, random_feats
)
print(f"{len(selected_features)} valuable features selected")
selected_features
```

```
90 valuable features selected
```

```
Out[32]: ['latitude availability_365',
 'minimum_nights avg_airport_distance_km',
 'price availability_365',
 'longitude minimum_nights',
 'availability_365 ewr_airport_distance_km',
 'longitude availability_365',
 'latitude minimum_nights',
 'calculated_host_listings_count len_name',
 'availability_365 jfk_airport_distance_km',
 'latitude len_name',
 'calculated_host_listings_count ewr_airport_distance_km',
 'availability_365 nyc_fd_distance_km',
 'longitude len_name',
 'minimum_nights jfk_airport_distance_km',
 'room_type_Private room',
 'calculated_host_listings_count nyc_fd_distance_km',
 'minimum_nights calculated_host_listings_count',
 'avg_airport_distance_km len_name',
 'availability_365 len_name',
 'minimum_nights total_min_cost',
 'availability_365 avg_airport_distance_km',
 'availability_365 nb_propn_in_name',
 'nyc_tm_distance_km lga_airport_distance_km',
 'minimum_nights ewr_airport_distance_km',
 'minimum_nights availability_365',
 'latitude calculated_host_listings_count',
 'availability_365 rate_adj_in_name',
 'minimum_nights lga_airport_distance_km',
 'nyc_tm_distance_km total_min_cost',
 'minimum_nights nyc_tm_distance_km',
 'calculated_host_listings_count availability_365',
 'longitude calculated_host_listings_count',
 'jfk_airport_distance_km len_name',
 'ewr_airport_distance_km len_name',
 'nyc_tm_distance_km nyc_cp_distance_km',
 'calculated_host_listings_count total_min_cost',
 'price calculated_host_listings_count',
 'availability_365 rate_propn_in_name',
 'nyc_tm_distance_km ewr_airport_distance_km',
 'jfk_airport_distance_km ewr_airport_distance_km',
 'price nyc_tm_distance_km',
 'minimum_nights len_name',
 'ewr_airport_distance_km total_min_cost',
 'total_min_cost len_name',
 'nyc_cp_distance_km total_min_cost',
 'len_name nb_propn_in_name',
 'availability_365 rate_nouns_in_name',
 'nyc_fd_distance_km lga_airport_distance_km',
 'avg_airport_distance_km total_min_cost',
 'minimum_nights rate_adj_in_name',
 'lga_airport_distance_km total_min_cost',
 'nyc_tm_distance_km nyc_fd_distance_km',
 'calculated_host_listings_count avg_airport_distance_km',
 'longitude total_min_cost',
 'nyc_fd_distance_km jfk_airport_distance_km',
 'minimum_nights nyc_fd_distance_km',
```

```
 'availability_365 nyc_tm_distance_km',
 'rate_adv_in_name rate_propn_in_name',
 'jfk_airport_distance_km lga_airport_distance_km',
 'jfk_airport_distance_km avg_airport_distance_km',
 'price ewr_airport_distance_km',
 'calculated_host_listings_count jfk_airport_distance_km',
 'jfk_airport_distance_km total_min_cost',
 'nyc_tm_distance_km len_name',
 'price avg_airport_distance_km',
 'ewr_airport_distance_km nb_propn_in_name',
 'calculated_host_listings_count nyc_tm_distance_km',
 'nyc_fd_distance_km total_min_cost',
 'nyc_tm_distance_km nb_propn_in_name',
 'nyc_tm_distance_km jfk_airport_distance_km',
 'calculated_host_listings_count rate_propn_in_name',
 'lga_airport_distance_km avg_airport_distance_km',
 'nyc_fd_distance_km rate_propn_in_name',
 'nyc_tm_distance_km rate_adj_in_name',
 'nyc_cp_distance_km jfk_airport_distance_km',
 'latitude longitude',
 'calculated_host_listings_count lga_airport_distance_km',
 'ewr_airport_distance_km rate_propn_in_name',
 'price total_min_cost',
 'calculated_host_listings_count nyc_cp_distance_km',
 'nb_nouns_in_name nb_propn_in_name',
 'availability_365 total_min_cost',
 'nyc_fd_distance_km nb_propn_in_name',
 'latitude avg_airport_distance_km',
 'longitude nyc_tm_distance_km',
 'nyc_tm_distance_km rate_propn_in_name',
 'lga_airport_distance_km len_name',
 'nyc_cp_distance_km rate_propn_in_name',
 'longitude jfk_airport_distance_km',
 'nyc_cp_distance_km nb_adj_in_name']
```

In [26]:

```
for idx, v in enumerate(ranked_features):
    print(f"{idx + 1} - {v[0]} -> shap: {round(v[1],4)}")
```

```
1 - latitude availability_365 -> shap: 0.2634
2 - minimum_nights avg_airport_distance_km -> shap: 0.2056
3 - price availability_365 -> shap: 0.1541
4 - longitude minimum_nights -> shap: 0.1267
5 - availability_365 ewr_airport_distance_km -> shap: 0.0821
6 - longitude availability_365 -> shap: 0.0648
7 - latitude minimum_nights -> shap: 0.0424
8 - calculated_host_listings_count len_name -> shap: 0.0307
9 - availability_365 jfk_airport_distance_km -> shap: 0.0274
10 - latitude len_name -> shap: 0.025
11 - calculated_host_listings_count ewr_airport_distance_km -> shap: 0.0249
12 - availability_365 nyc_fd_distance_km -> shap: 0.0244
13 - longitude len_name -> shap: 0.0206
14 - minimum_nights jfk_airport_distance_km -> shap: 0.0168
15 - room_type_Private room -> shap: 0.0151
16 - calculated_host_listings_count nyc_fd_distance_km -> shap: 0.0111
17 - minimum_nights calculated_host_listings_count -> shap: 0.0096
18 - avg_airport_distance_km len_name -> shap: 0.0086
19 - availability_365 len_name -> shap: 0.007
20 - minimum_nights total_min_cost -> shap: 0.0065
21 - availability_365 avg_airport_distance_km -> shap: 0.0063
22 - availability_365 nb_propn_in_name -> shap: 0.0057
23 - nyc_tm_distance_km lga_airport_distance_km -> shap: 0.005
24 - minimum_nights ewr_airport_distance_km -> shap: 0.0049
25 - minimum_nights availability_365 -> shap: 0.0045
26 - latitude calculated_host_listings_count -> shap: 0.0029
27 - availability_365 rate_adj_in_name -> shap: 0.0029
28 - minimum_nights lga_airport_distance_km -> shap: 0.0027
29 - nyc_tm_distance_km total_min_cost -> shap: 0.0026
30 - minimum_nights nyc_tm_distance_km -> shap: 0.0025
31 - calculated_host_listings_count availability_365 -> shap: 0.0024
32 - longitude calculated_host_listings_count -> shap: 0.0024
33 - jfk_airport_distance_km len_name -> shap: 0.0023
34 - ewr_airport_distance_km len_name -> shap: 0.0022
35 - nyc_tm_distance_km nyc_cp_distance_km -> shap: 0.0022
36 - calculated_host_listings_count total_min_cost -> shap: 0.0021
37 - price calculated_host_listings_count -> shap: 0.002
38 - availability_365 rate_propn_in_name -> shap: 0.0018
39 - nyc_tm_distance_km ewr_airport_distance_km -> shap: 0.0018
40 - jfk_airport_distance_km ewr_airport_distance_km -> shap: 0.0017
41 - price nyc_tm_distance_km -> shap: 0.0015
42 - minimum_nights len_name -> shap: 0.0015
43 - ewr_airport_distance_km total_min_cost -> shap: 0.0014
44 - total_min_cost len_name -> shap: 0.0014
45 - nyc_cp_distance_km total_min_cost -> shap: 0.0014
46 - len_name nb_propn_in_name -> shap: 0.0013
47 - availability_365 rate_nouns_in_name -> shap: 0.0013
48 - nyc_fd_distance_km lga_airport_distance_km -> shap: 0.0012
49 - avg_airport_distance_km total_min_cost -> shap: 0.0012
50 - minimum_nights rate_adj_in_name -> shap: 0.0011
51 - lga_airport_distance_km total_min_cost -> shap: 0.0011
52 - nyc_tm_distance_km nyc_fd_distance_km -> shap: 0.0011
53 - calculated_host_listings_count avg_airport_distance_km -> shap: 0.001
54 - longitude total_min_cost -> shap: 0.0009
55 - nyc_fd_distance_km jfk_airport_distance_km -> shap: 0.0009
56 - minimum_nights nyc_fd_distance_km -> shap: 0.0009
```

```
57 - availability_365 nyc_tm_distance_km -> shap: 0.0008
58 - rate_adv_in_name rate_propn_in_name -> shap: 0.0008
59 - jfk_airport_distance_km lga_airport_distance_km -> shap: 0.0007
60 - jfk_airport_distance_km avg_airport_distance_km -> shap: 0.0007
61 - price ewr_airport_distance_km -> shap: 0.0007
62 - calculated_host_listings_count jfk_airport_distance_km -> shap: 0.0007
63 - jfk_airport_distance_km total_min_cost -> shap: 0.0007
64 - nyc_tm_distance_km len_name -> shap: 0.0007
65 - price avg_airport_distance_km -> shap: 0.0006
66 - ewr_airport_distance_km nb_propn_in_name -> shap: 0.0006
67 - calculated_host_listings_count nyc_tm_distance_km -> shap: 0.0006
68 - nyc_fd_distance_km total_min_cost -> shap: 0.0006
69 - nyc_tm_distance_km nb_propn_in_name -> shap: 0.0006
70 - nyc_tm_distance_km jfk_airport_distance_km -> shap: 0.0006
71 - calculated_host_listings_count rate_propn_in_name -> shap: 0.0006
72 - lga_airport_distance_km avg_airport_distance_km -> shap: 0.0006
73 - nyc_fd_distance_km rate_propn_in_name -> shap: 0.0006
74 - nyc_tm_distance_km rate_adj_in_name -> shap: 0.0006
75 - nyc_cp_distance_km jfk_airport_distance_km -> shap: 0.0005
76 - availability_365 -> shap: 0.0005
77 - latitude longitude -> shap: 0.0005
78 - calculated_host_listings_count lga_airport_distance_km -> shap: 0.0005
79 - ewr_airport_distance_km rate_propn_in_name -> shap: 0.0005
80 - price total_min_cost -> shap: 0.0005
81 - calculated_host_listings_count nyc_cp_distance_km -> shap: 0.0004
82 - nb_nouns_in_name nb_propn_in_name -> shap: 0.0004
83 - availability_365 total_min_cost -> shap: 0.0004
84 - nyc_fd_distance_km nb_propn_in_name -> shap: 0.0004
85 - latitude avg_airport_distance_km -> shap: 0.0004
86 - longitude nyc_tm_distance_km -> shap: 0.0004
87 - nyc_tm_distance_km rate_propn_in_name -> shap: 0.0004
88 - lga_airport_distance_km len_name -> shap: 0.0004
89 - nyc_cp_distance_km rate_propn_in_name -> shap: 0.0004
90 - longitude jfk_airport_distance_km -> shap: 0.0004
91 - nyc_cp_distance_km nb_adj_in_name -> shap: 0.0004
92 - ___RANDOM_2____ -> shap: 0.0004
93 - ___RANDOM_1____ -> shap: 0.0004
94 - latitude rate_adj_in_name -> shap: 0.0004
95 - calculated_host_listings_count nb_propn_in_name -> shap: 0.0004
96 - availability_365 lga_airport_distance_km -> shap: 0.0003
97 - nyc_cp_distance_km nb_propn_in_name -> shap: 0.0003
98 - nyc_fd_distance_km len_name -> shap: 0.0003
99 - longitude lga_airport_distance_km -> shap: 0.0003
100 - total_min_cost rate_adj_in_name -> shap: 0.0003
101 - latitude total_min_cost -> shap: 0.0003
102 - longitude rate_adj_in_name -> shap: 0.0003
103 - availability_365 -> shap: 0.0003
104 - calculated_host_listings_count rate_adj_in_name -> shap: 0.0003
105 - rate_nouns_in_name rate_propn_in_name -> shap: 0.0003
106 - price jfk_airport_distance_km -> shap: 0.0003
107 - jfk_airport_distance_km nb_propn_in_name -> shap: 0.0003
108 - longitude nb_propn_in_name -> shap: 0.0003
109 - len_name nb_adj_in_name -> shap: 0.0003
110 - total_min_cost nb_nouns_in_name -> shap: 0.0003
111 - latitude price -> shap: 0.0003
112 - len_name nb_nouns_in_name -> shap: 0.0003
```

```
113 - price lga_airport_distance_km -> shap: 0.0003
114 - longitude nb_adv_in_name -> shap: 0.0003
115 - price rate_propn_in_name -> shap: 0.0002
116 - availability_365^2 -> shap: 0.0002
117 - nyc_tm_distance_km avg_airport_distance_km -> shap: 0.0002
118 - nyc_cp_distance_km rate_adj_in_name -> shap: 0.0002
119 - longitude ewr_airport_distance_km -> shap: 0.0002
120 - nyc_cp_distance_km ewr_airport_distance_km -> shap: 0.0002
121 - nb_propn_in_name rate_adv_in_name -> shap: 0.0002
122 - price len_name -> shap: 0.0002
123 - price nyc_fd_distance_km -> shap: 0.0002
124 - longitude nyc_fd_distance_km -> shap: 0.0002
125 - total_min_cost rate_nouns_in_name -> shap: 0.0002
126 - nyc_fd_distance_km avg_airport_distance_km -> shap: 0.0002
127 - price nb_propn_in_name -> shap: 0.0002
128 - nb_propn_in_name rate_nouns_in_name -> shap: 0.0002
129 - nyc_cp_distance_km lga_airport_distance_km -> shap: 0.0002
130 - calculated_host_listings_count rate_adv_in_name -> shap: 0.0002
131 - latitude nb_nouns_in_name -> shap: 0.0002
132 - minimum_nights nyc_cp_distance_km -> shap: 0.0002
133 - latitude nb_propn_in_name -> shap: 0.0002
134 - minimum_nights rate_propn_in_name -> shap: 0.0002
135 - total_min_cost rate_adv_in_name -> shap: 0.0002
136 - avg_airport_distance_km nb_adj_in_name -> shap: 0.0002
137 - calculated_host_listings_count nb_adj_in_name -> shap: 0.0002
138 - total_min_cost^2 -> shap: 0.0002
139 - longitude avg_airport_distance_km -> shap: 0.0002
140 - lga_airport_distance_km nb_adj_in_name -> shap: 0.0002
141 - nb_adj_in_name rate_nouns_in_name -> shap: 0.0002
142 - ewr_airport_distance_km nb_nouns_in_name -> shap: 0.0002
143 - ewr_airport_distance_km avg_airport_distance_km -> shap: 0.0002
144 - nyc_tm_distance_km nb_nouns_in_name -> shap: 0.0002
145 - len_name nb_adv_in_name -> shap: 0.0002
146 - ewr_airport_distance_km nb_adv_in_name -> shap: 0.0001
147 - nyc_tm_distance_km -> shap: 0.0001
148 - nb_adv_in_name rate_nouns_in_name -> shap: 0.0001
149 - total_min_cost rate_propn_in_name -> shap: 0.0001
150 - minimum_nights rate_nouns_in_name -> shap: 0.0001
151 - lga_airport_distance_km rate_propn_in_name -> shap: 0.0001
152 - longitude nb_nouns_in_name -> shap: 0.0001
153 - price rate_nouns_in_name -> shap: 0.0001
154 - price nyc_cp_distance_km -> shap: 0.0001
155 - jfk_airport_distance_km rate_propn_in_name -> shap: 0.0001
156 - longitude -> shap: 0.0001
157 - avg_airport_distance_km rate_propn_in_name -> shap: 0.0001
158 - lga_airport_distance_km nb_propn_in_name -> shap: 0.0001
159 - calculated_host_listings_count rate_nouns_in_name -> shap: 0.0001
160 - nyc_cp_distance_km len_name -> shap: 0.0001
161 - latitude ewr_airport_distance_km -> shap: 0.0001
162 - nb_propn_in_name rate_propn_in_name -> shap: 0.0001
163 - total_min_cost nb_adj_in_name -> shap: 0.0001
164 - nyc_cp_distance_km nyc_fd_distance_km -> shap: 0.0001
165 - availability_365 nyc_cp_distance_km -> shap: 0.0001
166 - jfk_airport_distance_km -> shap: 0.0001
167 - latitude -> shap: 0.0001
168 - nyc_cp_distance_km nb_nouns_in_name -> shap: 0.0001
```

```
169 - rate_adj_in_name rate_nouns_in_name -> shap: 0.0001
170 - longitude -> shap: 0.0001
171 - jfk_airport_distance_km nb_nouns_in_name -> shap: 0.0001
172 - price -> shap: 0.0001
173 - nyc_fd_distance_km rate_nouns_in_name -> shap: 0.0001
174 - longitude^2 -> shap: 0.0001
175 - nb_nouns_in_name rate_propn_in_name -> shap: 0.0001
176 - latitude nyc_fd_distance_km -> shap: 0.0001
177 - lga_airport_distance_km nb_nouns_in_name -> shap: 0.0001
178 - jfk_airport_distance_km rate_adv_in_name -> shap: 0.0001
179 - lga_airport_distance_km -> shap: 0.0001
180 - nyc_cp_distance_km avg_airport_distance_km -> shap: 0.0001
181 - longitude rate_propn_in_name -> shap: 0.0001
182 - latitude nb_adj_in_name -> shap: 0.0001
183 - latitude jfk_airport_distance_km -> shap: 0.0001
184 - latitude lga_airport_distance_km -> shap: 0.0001
185 - total_min_cost -> shap: 0.0001
186 - latitude nyc_cp_distance_km -> shap: 0.0001
187 - lga_airport_distance_km rate_nouns_in_name -> shap: 0.0001
188 - jfk_airport_distance_km nb_adj_in_name -> shap: 0.0001
189 - total_min_cost nb_adv_in_name -> shap: 0.0001
190 - total_min_cost nb_propn_in_name -> shap: 0.0001
191 - lga_airport_distance_km ewr_airport_distance_km -> shap: 0.0001
192 - avg_airport_distance_km rate_nouns_in_name -> shap: 0.0001
193 - ewr_airport_distance_km rate_adv_in_name -> shap: 0.0001
194 - price nb_nouns_in_name -> shap: 0.0001
195 - lga_airport_distance_km nb_adv_in_name -> shap: 0.0001
196 - price nb_adv_in_name -> shap: 0.0001
197 - nyc_tm_distance_km nb_adj_in_name -> shap: 0.0001
198 - longitude price -> shap: 0.0001
199 - nb_nouns_in_name rate_adj_in_name -> shap: 0.0001
200 - nyc_cp_distance_km rate_nouns_in_name -> shap: 0.0001
201 - rate_adj_in_name rate_propn_in_name -> shap: 0.0001
202 - ewr_airport_distance_km^2 -> shap: 0.0001
203 - nyc_fd_distance_km rate_adv_in_name -> shap: 0.0001
204 - nyc_tm_distance_km^2 -> shap: 0.0001
205 - jfk_airport_distance_km rate_adj_in_name -> shap: 0.0001
206 - latitude nyc_tm_distance_km -> shap: 0.0001
207 - nb_adj_in_name nb_nouns_in_name -> shap: 0.0001
208 - minimum_nights nb_propn_in_name -> shap: 0.0001
209 - minimum_nights nb_nouns_in_name -> shap: 0.0001
210 - avg_airport_distance_km rate_adj_in_name -> shap: 0.0001
211 - nb_propn_in_name rate_adj_in_name -> shap: 0.0001
212 - longitude rate_nouns_in_name -> shap: 0.0001
213 - jfk_airport_distance_km -> shap: 0.0001
214 - nb_adj_in_name nb_propn_in_name -> shap: 0.0001
215 - nb_adv_in_name rate_propn_in_name -> shap: 0.0001
216 - price rate_adj_in_name -> shap: 0.0001
217 - price rate_adv_in_name -> shap: 0.0001
218 - nyc_fd_distance_km nb_adv_in_name -> shap: 0.0001
219 - price nb_adj_in_name -> shap: 0.0001
220 - nb_adv_in_name nb_propn_in_name -> shap: 0.0
221 - nyc_fd_distance_km nb_adj_in_name -> shap: 0.0
222 - nyc_fd_distance_km nb_nouns_in_name -> shap: 0.0
223 - calculated_host_listings_count nb_nouns_in_name -> shap: 0.0
224 - latitude^2 -> shap: 0.0
```

```
225 - longitude nb_adj_in_name -> shap: 0.0
226 - avg_airport_distance_km^2 -> shap: 0.0
227 - latitude rate_propn_in_name -> shap: 0.0
228 - ewr_airport_distance_km -> shap: 0.0
229 - nyc_fd_distance_km rate_adj_in_name -> shap: 0.0
230 - rate_adv_in_name rate_nouns_in_name -> shap: 0.0
231 - lga_airport_distance_km rate_adv_in_name -> shap: 0.0
232 - nyc_tm_distance_km -> shap: 0.0
233 - nyc_fd_distance_km ewr_airport_distance_km -> shap: 0.0
234 - nb_adj_in_name nb_adv_in_name -> shap: 0.0
235 - longitude rate_adv_in_name -> shap: 0.0
236 - lga_airport_distance_km^2 -> shap: 0.0
237 - nb_adv_in_name rate_adj_in_name -> shap: 0.0
238 - minimum_nights rate_adv_in_name -> shap: 0.0
239 - jfk_airport_distance_km rate_nouns_in_name -> shap: 0.0
240 - jfk_airport_distance_km nb_adv_in_name -> shap: 0.0
241 - nyc_fd_distance_km -> shap: 0.0
242 - nyc_cp_distance_km -> shap: 0.0
243 - ewr_airport_distance_km rate_nouns_in_name -> shap: 0.0
244 - ewr_airport_distance_km -> shap: 0.0
245 - len_name -> shap: 0.0
246 - price^2 -> shap: 0.0
247 - lga_airport_distance_km -> shap: 0.0
248 - len_name^2 -> shap: 0.0
249 - nb_adj_in_name rate_adj_in_name -> shap: 0.0
250 - rate_adj_in_name rate_adv_in_name -> shap: 0.0
251 - nb_nouns_in_name rate_adv_in_name -> shap: 0.0
252 - rate_propn_in_name -> shap: 0.0
253 - nyc_cp_distance_km rate_adv_in_name -> shap: 0.0
254 - avg_airport_distance_km nb_nouns_in_name -> shap: 0.0
255 - lga_airport_distance_km rate_adj_in_name -> shap: 0.0
256 - avg_airport_distance_km nb_propn_in_name -> shap: 0.0
257 - nyc_tm_distance_km rate_adv_in_name -> shap: 0.0
258 - latitude rate_nouns_in_name -> shap: 0.0
259 - latitude -> shap: 0.0
260 - nb_nouns_in_name rate_nouns_in_name -> shap: 0.0
261 - longitude nyc_cp_distance_km -> shap: 0.0
262 - rate_adj_in_name^2 -> shap: 0.0
263 - availability_365 nb_adj_in_name -> shap: 0.0
264 - price minimum_nights -> shap: 0.0
265 - nb_nouns_in_name -> shap: 0.0
266 - ewr_airport_distance_km nb_adj_in_name -> shap: 0.0
267 - avg_airport_distance_km -> shap: 0.0
268 - nyc_fd_distance_km^2 -> shap: 0.0
269 - nyc_cp_distance_km -> shap: 0.0
270 - rate_adj_in_name -> shap: 0.0
271 - rate_nouns_in_name -> shap: 0.0
272 - nyc_tm_distance_km rate_nouns_in_name -> shap: 0.0
273 - rate_propn_in_name -> shap: 0.0
274 - jfk_airport_distance_km^2 -> shap: 0.0
275 - rate_nouns_in_name -> shap: 0.0
276 - avg_airport_distance_km -> shap: 0.0
277 - rate_propn_in_name^2 -> shap: 0.0
278 - minimum_nights -> shap: 0.0
279 - rate_adv_in_name^2 -> shap: 0.0
280 - nb_adv_in_name nb_nouns_in_name -> shap: 0.0
```

```
281 - len_name rate_nouns_in_name -> shap: 0.0
282 - latitude rate_adv_in_name -> shap: 0.0
283 - nyc_cp_distance_km nb_adv_in_name -> shap: 0.0
284 - calculated_host_listings_count^2 -> shap: 0.0
285 - nb_propn_in_name -> shap: 0.0
286 - nb_adj_in_name rate_propn_in_name -> shap: 0.0
287 - rate_adj_in_name -> shap: 0.0
288 - avg_airport_distance_km rate_adv_in_name -> shap: 0.0
289 - nb_propn_in_name -> shap: 0.0
290 - neighbourhood_group_Manhattan -> shap: 0.0
291 - nyc_fd_distance_km -> shap: 0.0
292 - rate_nouns_in_name^2 -> shap: 0.0
293 - nyc_cp_distance_km^2 -> shap: 0.0
294 - nb_adj_in_name -> shap: 0.0
295 - total_min_cost -> shap: 0.0
296 - len_name rate_propn_in_name -> shap: 0.0
297 - nb_adj_in_name^2 -> shap: 0.0
298 - ewr_airport_distance_km rate_adj_in_name -> shap: 0.0
299 - avg_airport_distance_km nb_adv_in_name -> shap: 0.0
300 - len_name rate_adj_in_name -> shap: 0.0
301 - minimum_nights nb_adv_in_name -> shap: 0.0
302 - calculated_host_listings_count -> shap: 0.0
303 - len_name rate_adv_in_name -> shap: 0.0
304 - rate_adv_in_name -> shap: 0.0
305 - nb_adv_in_name -> shap: 0.0
306 - nyc_tm_distance_km nb_adv_in_name -> shap: 0.0
307 - calculated_host_listings_count -> shap: 0.0
308 - len_name -> shap: 0.0
309 - price -> shap: 0.0
310 - latitude nb_adv_in_name -> shap: 0.0
311 - minimum_nights nb_adj_in_name -> shap: 0.0
312 - 1 -> shap: 0.0
313 - minimum_nights -> shap: 0.0
314 - nb_adv_in_name -> shap: 0.0
315 - nb_nouns_in_name -> shap: 0.0
316 - minimum_nights^2 -> shap: 0.0
317 - calculated_host_listings_count nb_adv_in_name -> shap: 0.0
318 - availability_365 nb_adv_in_name -> shap: 0.0
319 - availability_365 nb_nouns_in_name -> shap: 0.0
320 - availability_365 rate_adv_in_name -> shap: 0.0
321 - nb_adj_in_name rate_adv_in_name -> shap: 0.0
322 - nb_adv_in_name^2 -> shap: 0.0
323 - nb_adv_in_name rate_adv_in_name -> shap: 0.0
324 - nb_nouns_in_name^2 -> shap: 0.0
325 - nb_propn_in_name^2 -> shap: 0.0
326 - nb_adj_in_name -> shap: 0.0
327 - rate_adv_in_name -> shap: 0.0
328 - neighbourhood_group_Brooklyn -> shap: 0.0
329 - neighbourhood_group_Queens -> shap: 0.0
330 - neighbourhood_group_Staten Island -> shap: 0.0
331 - room_type_Shared room -> shap: 0.0
```

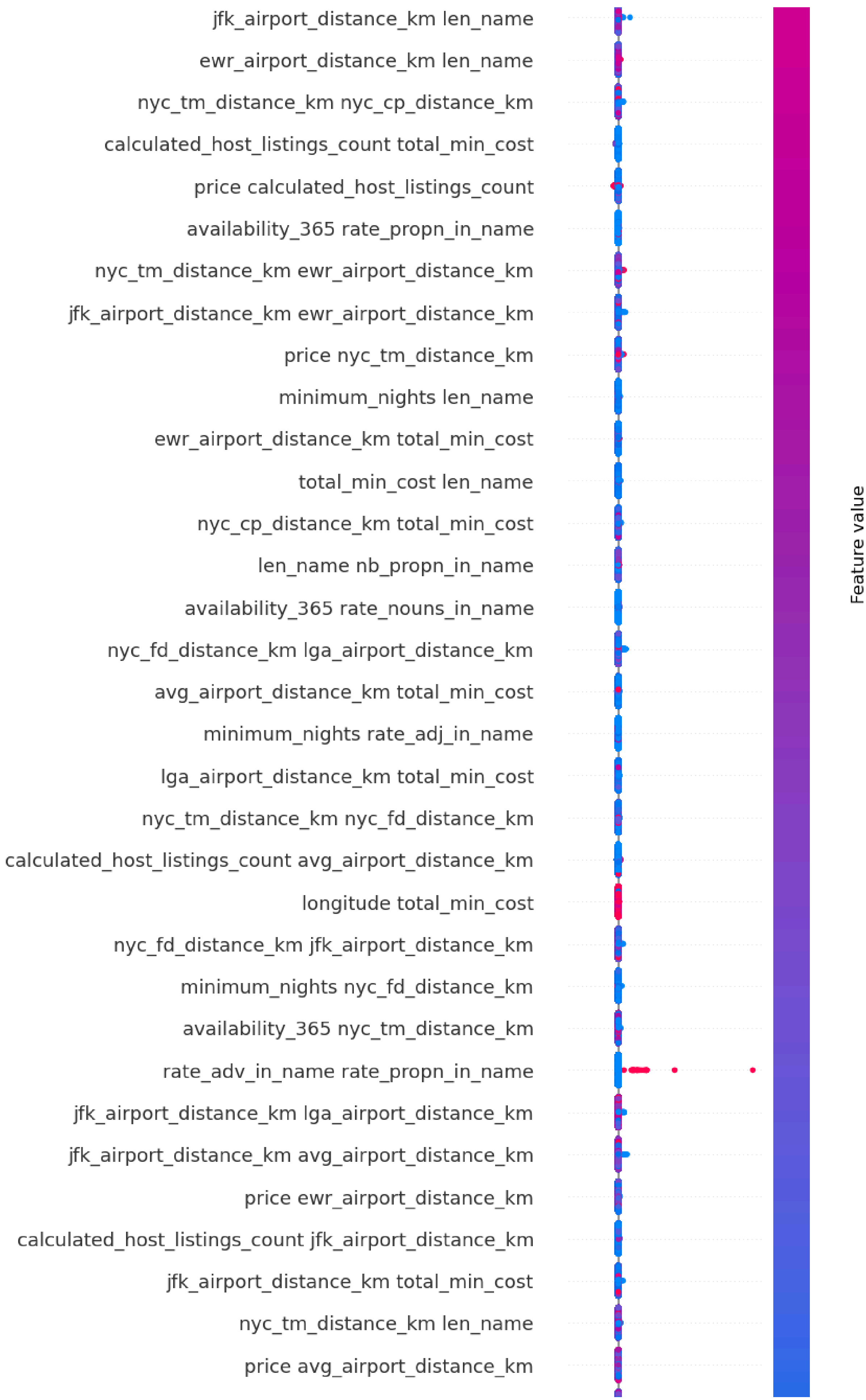
In [ ]:

```
plt.rcParams.update({
    "font.size": 8,
    "axes.titlesize": 9,
    "axes.labelsize": 8,
```

```
    "xtick.labelsize": 7,
    "ytick.labelsize": 7,
})

shap.summary_plot(
    shap_values,
    X_train_soft_s,
    feature_names=clean_features,
    max_display=len(selected_features),
)
```

```
/var/folders/gc/x3q0m4h564d61kh41xdz_gy00000gn/T/ipykernel_4013/4231117112.py:1: Fut
ureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future
version this function will no longer use the global RNG. Pass `rng` explicitly to op
t-in to the new behaviour and silence this warning.
  shap.summary_plot(shap_values, X_train_soft_s, feature_names=clean_features, max_d
isplay=len(selected_features))
```


latitude availability_365
minimum_nights avg_airport_distance_km
price availability_365
longitude minimum_nights
availability_365 ewr_airport_distance_km
longitude availability_365
latitude minimum_nights
calculated_host_listings_count len_name
availability_365 jfk_airport_distance_km
latitude len_name
calculated_host_listings_count ewr_airport_distance_km
availability_365 nyc_fd_distance_km
longitude len_name
minimum_nights jfk_airport_distance_km
room_type_Private room
calculated_host_listings_count nyc_fd_distance_km
minimum_nights calculated_host_listings_count
avg_airport_distance_km len_name
availability_365 len_name
minimum_nights total_min_cost
availability_365 avg_airport_distance_km
availability_365 nb_propn_in_name
nyc_tm_distance_km lga_airport_distance_km
minimum_nights ewr_airport_distance_km
minimum_nights availability_365
latitude calculated_host_listings_count
availability_365 rate_adj_in_name
minimum_nights lga_airport_distance_km
nyc_tm_distance_km total_min_cost
minimum_nights nyc_tm_distance_km
calculated_host_listings_count availability_365
longitude calculated_host_listings_count
High



jfk_airport_distance_km len_name
ewr_airport_distance_km len_name
nyc_tm_distance_km nyc_cp_distance_km
calculated_host_listings_count total_min_cost
price calculated_host_listings_count
availability_365 rate_propn_in_name
nyc_tm_distance_km ewr_airport_distance_km
jfk_airport_distance_km ewr_airport_distance_km
price nyc_tm_distance_km
minimum_nights len_name
ewr_airport_distance_km total_min_cost
total_min_cost len_name
nyc_cp_distance_km total_min_cost
len_name nb_propn_in_name
availability_365 rate_nouns_in_name
nyc_fd_distance_km lga_airport_distance_km
avg_airport_distance_km total_min_cost
minimum_nights rate_adj_in_name
lga_airport_distance_km total_min_cost
nyc_tm_distance_km nyc_fd_distance_km
calculated_host_listings_count avg_airport_distance_km
longitude total_min_cost
nyc_fd_distance_km jfk_airport_distance_km
minimum_nights nyc_fd_distance_km
availability_365 nyc_tm_distance_km
rate_adv_in_name rate_propn_in_name
jfk_airport_distance_km lga_airport_distance_km
jfk_airport_distance_km avg_airport_distance_km
price ewr_airport_distance_km
calculated_host_listings_count jfk_airport_distance_km
jfk_airport_distance_km total_min_cost
nyc_tm_distance_km len_name
price avg_airport_distance_km
Feature value

We used SHAP for feature selection as this method is used to measure each feature contribution to predictions (based on a game theory approach model). SHAP accounts for interactions among features (polynomial) and helps to separate valuable features in the model from random noise.

Here we added two random features as a baseline and we expect to have some valuable interactions among features in the model. If there are values lower than noise, is nothing that can be predicted. If we see the SHAP value graph we can overlook that geographic features as longitude and latitude appear in the most frequent interactions and availability and booking constraints matter (minimum_nights and availability_365).

The most important interaction among features are:

1. Latitude * availability_365 (0.2634): Most important interaction, so the location combined with how often the listing is available strongly impacts the target.
2. Minimum_nights * avg_airport_distance_km (0.2056): Booking requirements and airport accessibility together matter significantly.
3. Price * availability_365 (0.1541): Pricing strategy and availability interact to influence outcomes.
4. Longitude * minimum_nights (0.1267): East-west location combined with stay requirements.

10. Hyperparameter optimization

rubric={accuracy,reasoning}

Your tasks:

Make some attempts to optimize hyperparameters for the models you've tried and summarize your results. In at least one case you should be optimizing multiple hyperparameters for a single model. You may use `sklearn`'s methods for hyperparameter optimization or fancier Bayesian optimization methods. Briefly summarize your results.

- GridSearchCV
- RandomizedSearchCV
- scikit-optimize

Points: 6

Type your answer here, replacing this text.

In [68]:
```
param_grid = {
    "histgradientboostingregressor__max_depth": randint(40, 60),
    "histgradientboostingregressor__max_iter": randint(700, 1000),
    "histgradientboostingregressor__learning_rate": loguniform(0.005, 0.015)
}

random_search_hgbr = RandomizedSearchCV(
    make_pipeline(
        general_preprocessor,
        HistGradientBoostingRegressor(
            random_state=RANDOM_STATE
```

```
        )
    ),
    param_distributions=param_grid,
    n_iter=5,
    n_jobs= -1,
    return_train_score=True,
    verbose=3
)

random_search_hgbr.fit(X_train, y_train)

print(
    "Random Search best model score: \t %0.3f" % random_search_hgbr.best_score_
)



pd.DataFrame(random_search_hgbr.cv_results_)[
    [
        "mean_test_score",
        "mean_train_score",
        "param_histgradientboostingregressor__max_depth",
        "param_histgradientboostingregressor__max_iter",
        "param_histgradientboostingregressor__learning_rate",
        "mean_fit_time",
        "rank_test_score",
    ]
].set_index("rank_test_score").sort_index()
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
Random Search best model score:          0.375
```

Out[68]:

rank_test_score	mean_test_score	mean_train_score	param_histgradientboostingregressor__ma
1	0.374527	0.526722	
2	0.374371	0.535432	
3	0.373817	0.538871	
4	0.373741	0.511925	
5	0.373281	0.541082	

Our hyperparameter optimization is improving model performance. The best model (GBR) improves from R squared of 0.369 to 0.375 after tuning , a modest but meaningful gain.

The train score of 0.527 indicates a reasonable overfitting gap of 0.15. The optimal configuration uses moderately deep trees (max_depth = 24) with a low learning rate (0.006), balancing model complexity with generalization. So, the model seem stable with a good hyperparameter selection.

In [ ]:

```
ss.
```

```
In [ ]: ...
```

```
In [ ]: ...
```

```
In [ ]: ...
```

```
In [ ]: ...
```

11. Interpretation and feature importances

rubric={accuracy,reasoning}

Your tasks:

1. Use the methods we saw in class (e.g., `permutation_importance` or `shap`) (or any other methods of your choice) to examine the most important features of one of the non-linear models.
2. Summarize your observations.

Points: 8

Using SHAP values as explained before we can summarize the features that have the most significant impact:

- Price ** availability_365: Most important feature as the interaction between listing price and how often it's available has the strongest impact on predictions. Both high and low values influence the model.
- Longitude ** minimum_nights: It seems that geographic location combined with booking requirements is highly influential.
- Availability_365 ** airport distances. It seems that how often a listing is available, combined with its distance from airports/landmarks, matters significantly.
- Latitude ** minimum_nights: North-south location interacting with stay requirements.

We can also see that availability is all over the place in terms of appearing in many features, so this feature seems crucial. It is also noted that geography is not enough to have a good predictive power, we need a set of combinations or interactions with other valuable features as price or availability in order to have better power or prediction.

And also and very important, we can see that the polynomial transformations have better coefficients than single features, so we can almost be certain (as Ridge model also it does not perform well) that there is not a linear relationship here.

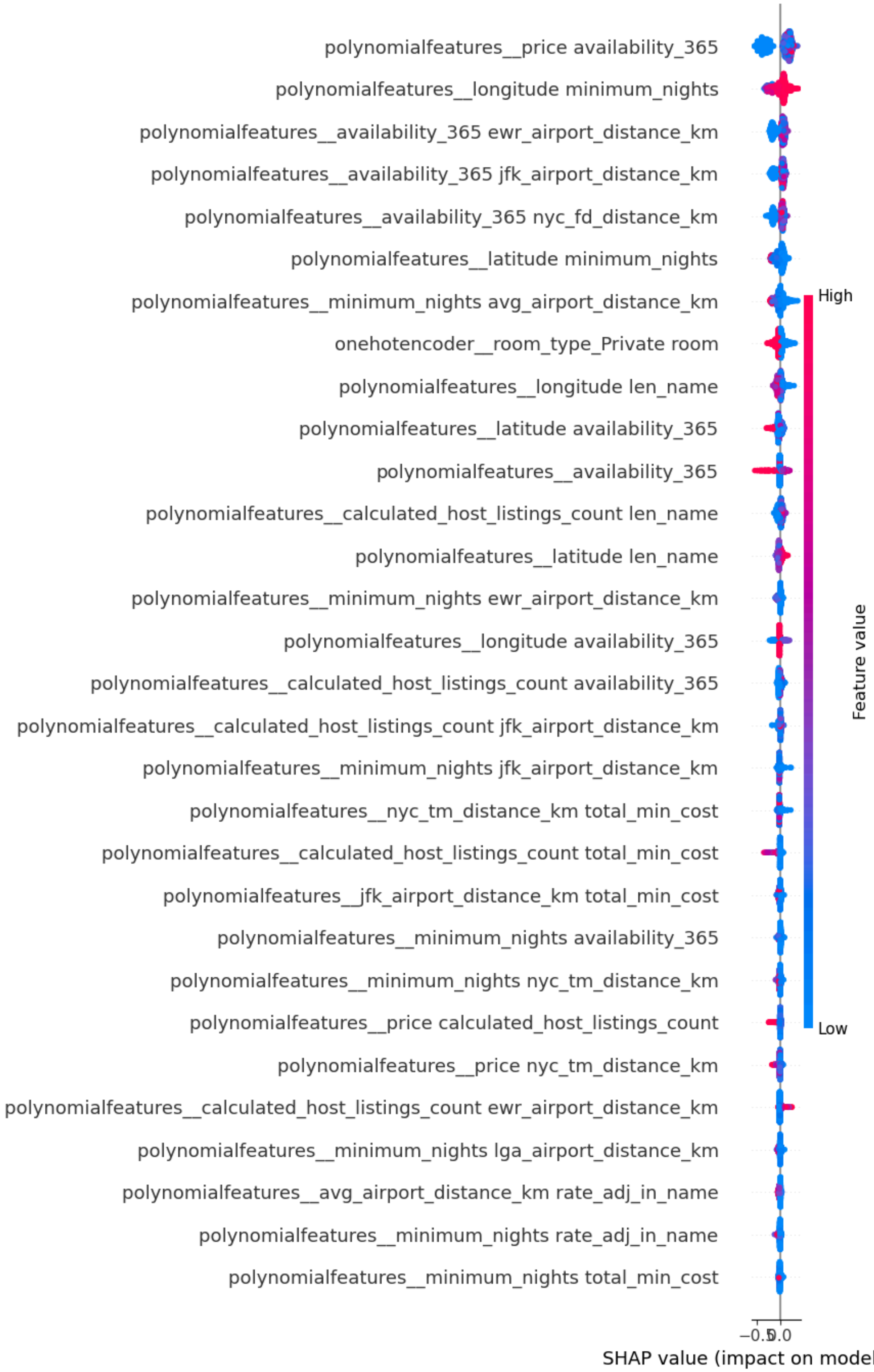
```
In [ ]: # For random forest!
```

```
# explainer = shap.TreeExplainer(tr)
# shap_values = explainer.shap_values(X_train_soft_s)
# shap.summary_plot(
#     shap_values,
#     X_train_soft_s,
#     feature_names=clean_features,
#     max_display=len(selected_features),
# )
```

In [75]:
```
from scipy import sparse


best_pipe = random_search_hgbr.best_estimator_
preprocess = best_pipe[:-1]
model = best_pipe.named_steps["histgradientboostingregressor"]


n = min(2000, len(X_train))
X_shap = X_train.sample(n=n, random_state=RANDOM_STATE)


X_shap_t = preprocess.transform(X_shap)
feature_names = preprocess.get_feature_names_out()


if sparse.issparse(X_shap_t):
    X_shap_t = X_shap_t.toarray()

X_shap_df = pd.DataFrame(X_shap_t, columns=feature_names)


explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_shap_df)


plt.rcParams.update({
    "font.size": 3,
    "axes.titlesize": 3,
    "axes.labelsize": 3,
    "xtick.labelsize": 3,
    "ytick.labelsize": 3,
})

shap.summary_plot(
    shap_values,
    X_shap_df,
    max_display=30
)
plt.tight_layout()
plt.show()


shap.summary_plot(
    shap_values,
    X_shap_df,
    plot_type="bar",
```

```
    max_display=30
)
plt.tight_layout()
plt.show()
```

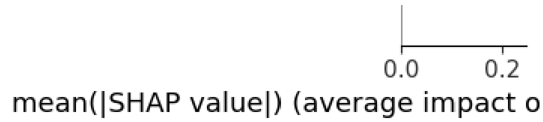
```
C:\Users\hpala\AppData\Local\Temp\ipykernel_25104\2574226130.py:35: FutureWarning: T
he NumPy global RNG was seeded by calling `np.random.seed`. In a future version this
function will no longer use the global RNG. Pass `rng` explicitly to opt-in to the n
ew behaviour and silence this warning.
  shap.summary_plot(
```


polynomialfeatures__price availability_365
polynomialfeatures__longitude minimum_nights
polynomialfeatures__availability_365 ewr_airport_distance_km
polynomialfeatures__availability_365 jfk_airport_distance_km
polynomialfeatures__availability_365 nyc_fd_distance_km
polynomialfeatures__latitude minimum_nights
polynomialfeatures__minimum_nights avg_airport_distance_km
onehotencoder__room_type_Private room
polynomialfeatures__longitude len_name
polynomialfeatures__latitude availability_365
polynomialfeatures__availability_365
polynomialfeatures__calculated_host_listings_count len_name
polynomialfeatures__latitude len_name
polynomialfeatures__minimum_nights ewr_airport_distance_km
polynomialfeatures__longitude availability_365
polynomialfeatures__calculated_host_listings_count availability_365
polynomialfeatures__calculated_host_listings_count jfk_airport_distance_km
polynomialfeatures__minimum_nights jfk_airport_distance_km
polynomialfeatures__nyc_tm_distance_km total_min_cost
polynomialfeatures__calculated_host_listings_count total_min_cost
polynomialfeatures__jfk_airport_distance_km total_min_cost
polynomialfeatures__minimum_nights availability_365
polynomialfeatures__minimum_nights nyc_tm_distance_km
polynomialfeatures__price calculated_host_listings_count
polynomialfeatures__price nyc_tm_distance_km
polynomialfeatures__calculated_host_listings_count ewr_airport_distance_km
polynomialfeatures__minimum_nights lga_airport_distance_km
polynomialfeatures__avg_airport_distance_km rate_adj_in_name
polynomialfeatures__minimum_nights rate_adj_in_name
polynomialfeatures__minimum_nights total_min_cost
High
Low
Feature value
−0.5
0.0
SHAP value (impact on model

```
<Figure size 640x480 with 0 Axes>
```

```
C:\Users\hpala\AppData\Local\Temp\ipykernel_25104\2574226130.py:44: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`. In a future version this function will no longer use the global RNG. Pass `rng` explicitly to opt-in to the new behaviour and silence this warning.
  shap.summary_plot(
```


polynomialfeatures__price availability_365
polynomialfeatures__longitude minimum_nights
polynomialfeatures__availability_365 ewr_airport_distance_km
polynomialfeatures__availability_365 jfk_airport_distance_km
polynomialfeatures__availability_365 nyc_fd_distance_km
polynomialfeatures__latitude minimum_nights
polynomialfeatures__minimum_nights avg_airport_distance_km
onehotencoder__room_type_Private room
polynomialfeatures__longitude len_name
polynomialfeatures__latitude availability_365
polynomialfeatures__availability_365
polynomialfeatures__calculated_host_listings_count len_name
polynomialfeatures__latitude len_name
polynomialfeatures__minimum_nights ewr_airport_distance_km
polynomialfeatures__longitude availability_365
polynomialfeatures__calculated_host_listings_count availability_365
polynomialfeatures__calculated_host_listings_count jfk_airport_distance_km
polynomialfeatures__minimum_nights jfk_airport_distance_km
polynomialfeatures__nyc_tm_distance_km total_min_cost
polynomialfeatures__calculated_host_listings_count total_min_cost
polynomialfeatures__jfk_airport_distance_km total_min_cost
polynomialfeatures__minimum_nights availability_365
polynomialfeatures__minimum_nights nyc_tm_distance_km
polynomialfeatures__price calculated_host_listings_count
polynomialfeatures__price nyc_tm_distance_km
polynomialfeatures__calculated_host_listings_count ewr_airport_distance_km
polynomialfeatures__minimum_nights lga_airport_distance_km
polynomialfeatures__avg_airport_distance_km rate_adj_in_name
polynomialfeatures__minimum_nights rate_adj_in_name
polynomialfeatures__minimum_nights total_min_cost




```
<Figure size 640x480 with 0 Axes>
```

In [ ]:

```

```

In [ ]:

```
...
```

12. Results on the test set

rubric={accuracy,reasoning}

Your tasks:

1. Try your best performing model on the test data and report test scores.
2. Do the test scores agree with the validation scores from before? To what extent do you trust your results? Do you think you've had issues with optimization bias?
3. Take one or two test predictions and explain them with SHAP force plots.

Points: 6

We have a test score of 0.3648

In a comparison with validation:

- Validation: 0.3745 (from hyperparameter tuning)
- Test: 0.3648
- Difference: 0.0097

The test score closely matches the validation score, with only a 2.6% relative decrease. This strong agreement indicates:

1. **Minimal optimization bias**: Our hyperparameter search didn't overfit to the validation set
2. **Good generalization**: The model performs consistently on unseen data
3. **Trustworthy results**: The 37% variance explained represents genuine predictive capability

The modest absolute R squared (0.37) reflects the inherent difficulty of predicting subjective review scores from listing features alone. Unmeasured factors (host responsiveness, cleanliness, guest expectations) contribute to the remaining 63% of variance. Given these limitations, capturing 37% of variance is reasonable performance for this prediction task.

The prediction models tells us that expensive and remote listings receive poor reviews. Despite its high total_min_cost (160,000) and price × total_min_cost (32,000), the model predicts a review score of only 0.30. The combination of high costs, inconvenient airport access (high distance interactions), and remote location creates guest dissatisfaction. This demonstrates the model learned that expensive listings in less accessible locations tend to disappoint guests. And we have confirmation that price alone doesn't guarantee good reviews; location and accessibility matter significantly.

SHAP VALUES: This listing has much lower predicted value (0.3 vs baseline 1.336) primarily because its price-availability interaction is zero, and several airport distance interactions are also zero or negative. This suggests it might be a listing with low availability or unusual pricing.

In [74]:
```
best_pipe.score(X_test, y_test)
```

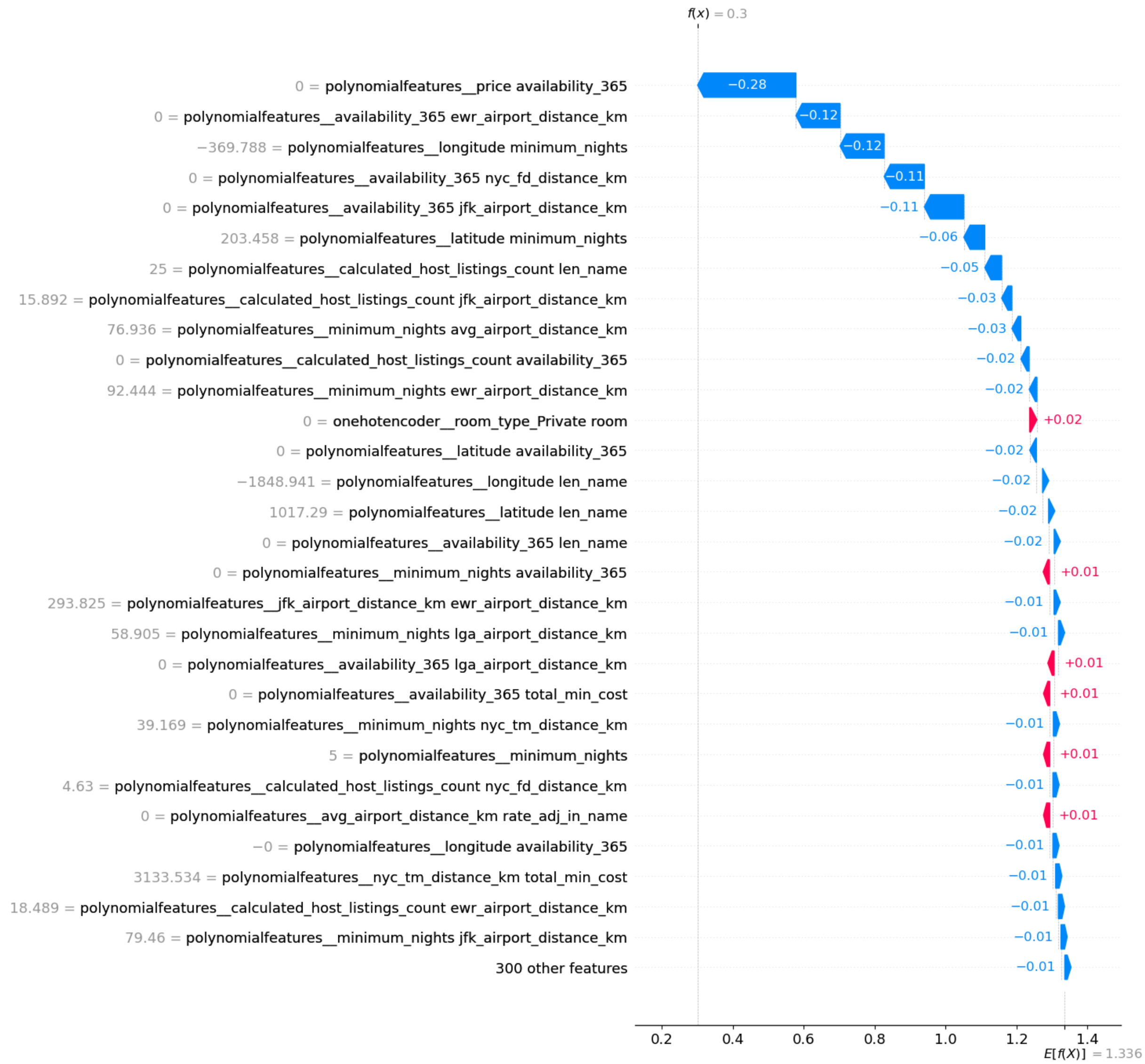
Out[74]:
```
0.3648321574290111
```

In [76]:
```
ex_index = 573

best_pipe = random_search_hgbr.best_estimator_
preprocess = best_pipe[:-1]
model = best_pipe.named_steps["histgradientboostingregressor"]


x_raw = X_test.iloc[[ex_index]]
x_enc = preprocess.transform(x_raw)

feature_names = preprocess.get_feature_names_out()

if sparse.issparse(x_enc):
    x_enc = x_enc.toarray()

x_enc_df = pd.DataFrame(x_enc, columns=feature_names)


display(x_enc_df.iloc[0].sort_values(ascending=False).head(30))


pred = best_pipe.predict(x_raw)[0]
print("Prediction:", pred)



explainer = shap.TreeExplainer(model)


shap_values = explainer.shap_values(x_enc_df)


shap_df = (pd.DataFrame(
    {"feature": feature_names, "shap_value": shap_values[0], "value": x_enc_df.iloc
).assign(abs_shap=lambda d: d.shap_value.abs())
```

```python
 .sort_values("abs_shap", ascending=False))

display(shap_df[["feature", "value", "shap_value"]].head(30))


shap.initjs()


base_value = explainer.expected_value
exp = shap.Explanation(
    values=shap_values[0],
    base_values=base_value,
    data=x_enc_df.iloc[0].values,
    feature_names=feature_names,
)

shap.plots.waterfall(exp, max_display=30)
```

```
polynomialfeatures__total_min_cost^2                          160000.000000
polynomialfeatures__price total_min_cost                       32000.000000
polynomialfeatures__latitude total_min_cost                    16276.636000
polynomialfeatures__total_min_cost len_name                    10000.000000
polynomialfeatures__ewr_airport_distance_km total_min_cost      7395.550686
polynomialfeatures__price^2                                     6400.000000
polynomialfeatures__jfk_airport_distance_km total_min_cost      6356.802021
polynomialfeatures__avg_airport_distance_km total_min_cost      6154.918861
polynomialfeatures__longitude^2                                 5469.733994
polynomialfeatures__lga_airport_distance_km total_min_cost      4712.403876
polynomialfeatures__nyc_cp_distance_km total_min_cost           4056.797245
polynomialfeatures__latitude price                              3255.327200
polynomialfeatures__nyc_tm_distance_km total_min_cost           3133.534280
polynomialfeatures__price len_name                              2000.000000
polynomialfeatures__minimum_nights total_min_cost               2000.000000
polynomialfeatures__nyc_fd_distance_km total_min_cost           1852.158863
polynomialfeatures__latitude^2                                  1655.805497
polynomialfeatures__price ewr_airport_distance_km               1479.110137
polynomialfeatures__price jfk_airport_distance_km               1271.360404
polynomialfeatures__price avg_airport_distance_km               1230.983772
polynomialfeatures__total_min_cost nb_propn_in_name             1200.000000
polynomialfeatures__latitude len_name                           1017.289750
polynomialfeatures__price lga_airport_distance_km                942.480775
polynomialfeatures__price nyc_cp_distance_km                     811.359449
polynomialfeatures__latitude ewr_airport_distance_km             752.341791
polynomialfeatures__latitude jfk_airport_distance_km             646.670954
polynomialfeatures__price nyc_tm_distance_km                     626.706856
polynomialfeatures__latitude avg_airport_distance_km             626.133587
polynomialfeatures__len_name^2                                   625.000000
polynomialfeatures__latitude lga_airport_distance_km             479.388016
Name: 573, dtype: float64
Prediction: 0.3003480226733975
```

	feature	value	shap_value
72	polynomialfeatures__price availability_365	0.000000	-0.277241
135	polynomialfeatures__availability_365 ewr_airpo...	0.000000	-0.124474
49	polynomialfeatures__longitude minimum_nights	-369.788250	-0.124358
132	polynomialfeatures__availability_365 nyc_fd_di...	0.000000	-0.113238
133	polynomialfeatures__availability_365 jfk_airpo...	0.000000	-0.111357
27	polynomialfeatures__latitude minimum_nights	203.457950	-0.059163
120	polynomialfeatures__calculated_host_listings_c...	25.000000	-0.047551
115	polynomialfeatures__calculated_host_listings_c...	15.892005	-0.028901
99	polynomialfeatures__minimum_nights avg_airport...	76.936486	-0.025167
111	polynomialfeatures__calculated_host_listings_c...	0.000000	-0.023566
98	polynomialfeatures__minimum_nights ewr_airport...	92.444384	-0.021954
327	onehotencoder__room_type_Private room	0.000000	0.020540
29	polynomialfeatures__latitude availability_365	0.000000	-0.019084
60	polynomialfeatures__longitude len_name	-1848.941250	-0.017290
38	polynomialfeatures__latitude len_name	1017.289750	-0.017076
138	polynomialfeatures__availability_365 len_name	0.000000	-0.015625
92	polynomialfeatures__minimum_nights availabilit...	0.000000	0.013395
197	polynomialfeatures__jfk_airport_distance_km ew...	293.825322	-0.013034
97	polynomialfeatures__minimum_nights lga_airport...	58.905048	-0.012519
134	polynomialfeatures__availability_365 lga_airpo...	0.000000	0.012338
137	polynomialfeatures__availability_365 total_min...	0.000000	0.012272
93	polynomialfeatures__minimum_nights nyc_tm_dist...	39.169179	-0.010146
4	polynomialfeatures__minimum_nights	5.000000	0.010106
114	polynomialfeatures__calculated_host_listings_c...	4.630397	-0.008888
241	polynomialfeatures__avg_airport_distance_km ra...	0.000000	0.008823
51	polynomialfeatures__longitude availability_365	-0.000000	-0.008736
154	polynomialfeatures__nyc_tm_distance_km total_m...	3133.534280	-0.007799
117	polynomialfeatures__calculated_host_listings_c...	18.488877	-0.007549
96	polynomialfeatures__minimum_nights jfk_airport...	79.460025	-0.007480
11	polynomialfeatures__lga_airport_distance_km	11.781010	0.007046



In [ ]:
```
...
```

In [ ]:
```
...
```

In [ ]:
```
...
```

In [ ]:
```
...
```

13. Summary of results

rubric={reasoning}

Imagine that you want to present the summary of these results to your boss and co-workers.

Your tasks:

1. Create a table summarizing important results.
2. Write concluding remarks.
3. Discuss other ideas that you did not try but could potentially improve the performance/interpretability .
4. Report your final test score along with the metric you used at the top of this notebook.

Points: 8

Model Performance

Our gradient boosting model achieved a test R squared of 0.365, successfully explaining approximately 37% of the variance in Airbnb listing review frequency. While this may appear modest in absolute terms, it represents meaningful predictive capability given the inherent challenges of the problem:

What we have found?

- Strong Generalization: The close alignment between validation (0.375) and test (0.365) scores indicates minimal optimization bias and robust model performance on unseen data.
- Feature Interactions : Polynomial features significantly outperformed single features, demonstrating that review frequency depends on complex interactions rather than linear relationships.

Model Comparison Insights

1. Ridge regression severely underfit (R squared = 0.253), requiring extreme regularization due to multicollinearity
2. Decision trees catastrophically overfit (perfect train, negative test R squared)
3. Random Forest showed promise but significant overfitting (0.847 train vs 0.338 test)
4. Gradient Boosting achieved the best balance with modest overfitting (0.527 train vs 0.365 test)

For Airbnb hosts, this model provides some insights:

- Availability is crucial: Listings with higher availability combined with competitive pricing receive more reviews
- Location matters differently than expected: Proximity to airports and landmarks interacts with other features rather than having direct effects
- Booking constraints impact reviews: Minimum night requirements influence review frequency, especially when combined with geographic factors
- Room type has limited direct impact: Being a private room vs entire home shows minimal effect (SHAP: 0.021)

Model Limitations

The 63% unexplained variance reflects legitimate constraints:

- Missing subjective factors: Host responsiveness, cleanliness, guest experience, accuracy of listing descriptions
- Selection bias in reviews: Dissatisfied guests are more likely to leave reviews than satisfied ones (this we transformed in the beginning of the data in order to prevent data leakage)
- Temporal dynamics: Seasonal patterns, booking trends, and time-varying host behavior not captured
- Guest characteristics: Demographics, travel purpose, and expectations unavailable in the dataset

In [ ]:
```
...
```

14. Creating a data analysis pipeline (Challenging)

rubric={reasoning}

Your tasks:

- Convert this notebook into scripts to create a reproducible data analysis pipeline with appropriate documentation. Submit your project folder in addition to this notebook on GitHub and briefly comment on your organization in the text box below.

Points: 0.5

Type your answer here, replacing this text.

15. Your takeaway from the course (Challenging)

rubric={reasoning}

Your tasks:

What is your biggest takeaway from this course?

Points: 0.5

Type your answer here, replacing this text.

Restart, run all and export a PDF before submitting

Before submitting, don't forget to run all cells in your notebook to make sure there are no errors and so that the TAs can see your plots on Gradescope. You can do this by clicking the ⏩ button or going to `Kernel -> Restart Kernel and Run All Cells...` in the menu. This is not only important for MDS, but a good habit you should get into before ever committing a notebook to GitHub, so that your collaborators can run it from top to bottom without issues.

After running all the cells, export a PDF of the notebook (preferably the WebPDF export) and upload this PDF together with the ipynb file to Gradescope (you can select two files when uploading to Gradescope)

Help us improve the labs

The MDS program is continually looking to improve our courses, including lab questions and content. The following optional questions will not affect your grade in any way nor will they be used for anything other than program improvement:

1. Approximately how many hours did you spend working or thinking about this assignment (including lab time)?

Ans:

2. Do you have any feedback on the lab you be willing to share? For example, any part or question that you particularly liked or disliked?

Ans:

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

In [ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(run_tests=True)
```