advent_of_code (/github/ab-dum/advent_of_code/tree/main)
/
Apo_DA3_Assignment2.ipynb (/github/ab-dum/advent_of_code/tree/main/Apo_DA3_Assignment2.ipynb)

Data Analysis 3 - Assignment 2

In this project, it's aimed to determine a nightly price for its new apartments to a company operating in Florence / Italy, which can accommodate 2-6 people and rents small and medium-sized apartments. For this purpose, price and feature information of airbnb houses operating in Florence will be obtained from http://insideairbnb.com/get-the-data.html (http://insideairbnb.com/get-the-data.html), Machine Learning price prediction models will be created and then these models will be compared with the models previously obtained for London.

Sample Selection

First of all, data will be downloaded and cleaned for Exploratory Data Analysis (EDA) by applying filters we need.

In [970…

```
#importing the packages

import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
import os
from pathlib import Path
import sys
from patsy import dmatrices
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.impute import SimpleImputer
from sklearn.inspection import permutation_importance
from sklearn.inspection import PartialDependenceDisplay
from sklearn.inspection import partial_dependence
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_squared_error
```

Downloading and Cleaning the Data

In [971…
```
# importing helper functions
from py_helper_functions import *
```

In [972…
```
# importing the data from Github for Florence
data = pd.read_csv('https://raw.githubusercontent.com/ab-dum/DA3/main/Assignm
```

In [973…
```
# checking the data how it looks like
data.head()
```

Out[973]:

	id	listing_url	scrape_id	last_scraped	source	r
0	31840	https://www.airbnb.com/rooms/31840	20231218165100	2023-12-19	city scrape	Ser apart Flore ★4. l
1	222527	https://www.airbnb.com/rooms/222527	20231218165100	2023-12-18	city scrape	R u S Flore · ★ 1
2	32120	https://www.airbnb.com/rooms/32120	20231218165100	2023-12-18	city scrape	R u Flore ★4. bedr
3	224562	https://www.airbnb.com/rooms/224562	20231218165100	2023-12-19	city scrape	Cor Flore ★4. bedr 1 b
4	32180	https://www.airbnb.com/rooms/32180	20231218165100	2023-12-19	city scrape	Cor Flore ★4. bedr · 4 l

5 rows × 75 columns

In [974…
```
# there are 12578 rows and 75 columns
data.shape
```

Out[974]: (12578, 75)

Exploratory Data Analysis (EDA)

First off all, we need to deal with extreme and missing values. Hence ıt makes sense to check the null values for each column

In [975…

```
# checking the columns, types and non-null counts
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12578 entries, 0 to 12577
Data columns (total 75 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   id                                            12578 non-null  int64
 1   listing_url                                   12578 non-null  object
 2   scrape_id                                     12578 non-null  int64
 3   last_scraped                                  12578 non-null  object
 4   source                                        12578 non-null  object
 5   name                                          12578 non-null  object
 6   description                                   0 non-null      float64
 7   neighborhood_overview                         7383 non-null   object
 8   picture_url                                   12578 non-null  object
 9   host_id                                       12578 non-null  int64
 10  host_url                                      12578 non-null  object
 11  host_name                                     12578 non-null  object
 12  host_since                                    12578 non-null  object
 13  host_location                                 10142 non-null  object
 14  host_about                                    7450 non-null   object
 15  host_response_time                            10696 non-null  object
 16  host_response_rate                            10696 non-null  object
 17  host_acceptance_rate                          11333 non-null  object
 18  host_is_superhost                             12525 non-null  object
 19  host_thumbnail_url                            12578 non-null  object
 20  host_picture_url                              12578 non-null  object
 21  host_neighbourhood                            7458 non-null   object
 22  host_listings_count                           12578 non-null  int64
 23  host_total_listings_count                     12578 non-null  int64
 24  host_verifications                            12578 non-null  object
 25  host_has_profile_pic                          12578 non-null  object
 26  host_identity_verified                        12578 non-null  object
 27  neighbourhood                                 7383 non-null   object
 28  neighbourhood_cleansed                        12578 non-null  object
 29  neighbourhood_group_cleansed                  0 non-null      float64
 30  latitude                                      12578 non-null  float64
 31  longitude                                     12578 non-null  float64
 32  property_type                                 12578 non-null  object
 33  room_type                                     12578 non-null  object
 34  accommodates                                  12578 non-null  int64
 35  bathrooms                                     0 non-null      float64
 36  bathrooms_text                                12575 non-null  object
 37  bedrooms                                      11 non-null     float64
 38  beds                                          12468 non-null  float64
 39  amenities                                     12578 non-null  object
 40  price                                         11491 non-null  object
 41  minimum_nights                                12578 non-null  int64
 42  maximum_nights                                12578 non-null  int64
 43  minimum_minimum_nights                        12578 non-null  int64
 44  maximum_minimum_nights                        12578 non-null  int64
 45  minimum_maximum_nights                        12578 non-null  int64
 46  maximum_maximum_nights                        12578 non-null  int64
 47  minimum_nights_avg_ntm                        12578 non-null  float64
 48  maximum_nights_avg_ntm                        12578 non-null  float64
 49  calendar_updated                              0 non-null      float64
 50  has_availability                              11491 non-null  object
 51  availability_30                               12578 non-null  int64
 52  availability_60                               12578 non-null  int64
 53  availability_90                               12578 non-null  int64
 54  availability_365                              12578 non-null  int64
 55  calendar_last_scraped                         12578 non-null  object
```

```
 56  number_of_reviews                             12578 non-null  int64
 57  number_of_reviews_ltm                         12578 non-null  int64
 58  number_of_reviews_l30d                        12578 non-null  int64
 59  first_review                                  10943 non-null  object
 60  last_review                                   10943 non-null  object
 61  review_scores_rating                          10957 non-null  float64
 62  review_scores_accuracy                        10955 non-null  float64
 63  review_scores_cleanliness                     10954 non-null  float64
 64  review_scores_checkin                         10954 non-null  float64
 65  review_scores_communication                   10954 non-null  float64
 66  review_scores_location                        10954 non-null  float64
 67  review_scores_value                           10954 non-null  float64
 68  license                                       2815 non-null   object
 69  instant_bookable                              12578 non-null  object
 70  calculated_host_listings_count                12578 non-null  int64
 71  calculated_host_listings_count_entire_homes   12578 non-null  int64
 72  calculated_host_listings_count_private_rooms  12578 non-null  int64
 73  calculated_host_listings_count_shared_rooms   12578 non-null  int64
 74  reviews_per_month                             10943 non-null  float64
dtypes: float64(18), int64(23), object(34)
memory usage: 7.2+ MB
```

- It is seen that some of the 75 columns have a high rate of missing values. In order to make more effective predictions, these values need to be cleaned or transformed.

In [976…

```
# creating a function to find the percentange of null values for each column
def null_percentage(df):
    # finding the NA values in each column
    null_counts = df.isna().sum()

    # excluding the columns with zero NA values
    non_zero_na_columns = null_counts[null_counts > 0].index
    null_counts = null_counts[non_zero_na_columns]

    # calculating the percentage of NAs in each column
    na_percentage = (null_counts / len(df)) * 100

    # creating a DataFrame to display results
    result_df = pd.DataFrame({
        'Column': null_counts.index,
        'NA Count': null_counts.values,
        'NA Percentage': na_percentage.values
    })

    return result_df

result = null_percentage(data)
print(result)
```

	Column	NA Count	NA Percentage
0	description	12578	100.000000
1	neighborhood_overview	5195	41.302274
2	host_location	2436	19.367149
3	host_about	5128	40.769598
4	host_response_time	1882	14.962633
5	host_response_rate	1882	14.962633
6	host_acceptance_rate	1245	9.898235
7	host_is_superhost	53	0.421371
8	host_neighbourhood	5120	40.705995
9	neighbourhood	5195	41.302274
10	neighbourhood_group_cleansed	12578	100.000000
11	bathrooms	12578	100.000000
12	bathrooms_text	3	0.023851
13	bedrooms	12567	99.912546
14	beds	110	0.874543
15	price	1087	8.642073
16	calendar_updated	12578	100.000000
17	has_availability	1087	8.642073
18	first_review	1635	12.998887
19	last_review	1635	12.998887
20	review_scores_rating	1621	12.887581
21	review_scores_accuracy	1623	12.903482
22	review_scores_cleanliness	1624	12.911433
23	review_scores_checkin	1624	12.911433
24	review_scores_communication	1624	12.911433
25	review_scores_location	1624	12.911433
26	review_scores_value	1624	12.911433
27	license	9763	77.619653
28	reviews_per_month	1635	12.998887

In [977…
```
# calculating the percentage of missing values in each column
missing_percentage = (data.isna().sum() / len(data)) * 100
```

In [978…
```
# identifying the columns with more than 70% missing values
columns_to_drop = missing_percentage[missing_percentage > 70].index
```

In [979…
```
# dropping the columns with more than 70% missing values
data = data.drop(columns=columns_to_drop)
```

In [980…
```
# the number of columns decreased to 69
data.shape
```

Out[980]: (12578, 69)

In [981…
```
# removing the rows with any NA values
data = data.dropna()
```

In [982…
```
data.shape
```

Out[982]: (2683, 69)

In [983…
```
# we have 2.683 rows and 69 columns without any NAs
data.isna().sum().sum()
```

Out[983]: 0

- It is clear that there are other techniques than dropping when evaluating rows with NA values, and care should be taken when making this choice. If NA values represent any pattern, it may not make sense to drop them, but since it is known that there is no such pattern in this data set, the drop method was chosen.

In [984…

```
# we need to filter data on number of accommodates that can accommodate on a
data.accommodates.unique()
```

Out[984]: array([4, 2, 1, 3, 5, 6, 10, 9, 7, 8, 11, 12, 14, 16, 15, 13])

In [985…

```
# we need to focus on apartments that can host 2-6 guests
data = data[(data['accommodates'] >= 2) & (data['accommodates'] <= 6)]
```

In [986…

```
# there are 2428 rows after filtering on number of 'accommodates'
data.shape
```

Out[986]: (2428, 69)

In [987…

```
# showing the statistics for numeric variables in DataFrame
data.describe().T
```

Out[987]:

	count	mean	std	
id	2428.0	1.052271e+17	2.736894e+17	3.91150
scrape_id	2428.0	2.023122e+13	0.000000e+00	2.02312
host_id	2428.0	5.452045e+07	6.925702e+07	3.32100
host_listings_count	2428.0	3.976936e+01	1.034751e+02	1.00000
host_total_listings_count	2428.0	5.518616e+01	1.457494e+02	1.00000
latitude	2428.0	4.377297e+01	8.552385e-03	4.3726
longitude	2428.0	1.125429e+01	1.340070e-02	1.1160
accommodates	2428.0	3.588138e+00	1.344638e+00	2.00000
beds	2428.0	2.097611e+00	1.091199e+00	1.00000
minimum_nights	2428.0	4.180395e+00	3.527156e+01	1.00000
maximum_nights	2428.0	4.455243e+02	5.039758e+02	1.00000
minimum_minimum_nights	2428.0	3.801483e+00	3.490453e+01	1.00000
maximum_minimum_nights	2428.0	5.079489e+00	3.560632e+01	1.00000
minimum_maximum_nights	2428.0	6.979403e+02	5.120941e+02	1.00000
maximum_maximum_nights	2428.0	7.754848e+02	4.865722e+02	3.00000
minimum_nights_avg_ntm	2428.0	4.509061e+00	3.539308e+01	1.00000
maximum_nights_avg_ntm	2428.0	7.437336e+02	4.901132e+02	3.00000
availability_30	2428.0	1.235626e+01	9.787458e+00	0.00000
availability_60	2428.0	2.880560e+01	2.118026e+01	0.00000
availability_90	2428.0	4.563344e+01	3.293004e+01	0.00000
availability_365	2428.0	1.807677e+02	1.280749e+02	0.00000
number_of_reviews	2428.0	1.263542e+02	1.401542e+02	1.00000
number_of_reviews_ltm	2428.0	2.540198e+01	2.467624e+01	0.00000
number_of_reviews_l30d	2428.0	9.876442e-01	1.607772e+00	0.00000
review_scores_rating	2428.0	4.729778e+00	2.999591e-01	1.00000
review_scores_accuracy	2428.0	4.795437e+00	2.675020e-01	1.00000
review_scores_cleanliness	2428.0	4.773031e+00	2.904505e-01	1.00000
review_scores_checkin	2428.0	4.833002e+00	2.356978e-01	1.00000
review_scores_communication	2428.0	4.832916e+00	2.675426e-01	1.00000
review_scores_location	2428.0	4.829572e+00	2.425226e-01	1.00000
review_scores_value	2428.0	4.684580e+00	3.162071e-01	1.00000
calculated_host_listings_count	2428.0	1.869110e+01	3.702579e+01	1.00000
calculated_host_listings_count_entire_homes	2428.0	1.745511e+01	3.629291e+01	0.00000
calculated_host_listings_count_private_rooms	2428.0	1.154448e+00	2.490119e+00	0.00000
calculated_host_listings_count_shared_rooms	2428.0	3.294893e-03	5.731832e-02	0.00000
reviews_per_month	2428.0	1.868303e+00	1.637040e+00	1.0000

In [988…
```
# checking the price column
data.price.unique()
```

```
Out[988]: array(['$60.00', '$86.00', '$83.00', '$61.00', '$95.00', '$159.00',
                 '$82.00', '$65.00', '$90.00', '$66.00', '$120.00', '$106.00',
                 '$130.00', '$126.00', '$279.00', '$85.00', '$129.00', '$113.00',
                 '$114.00', '$124.00', '$161.00', '$81.00', '$257.00', '$137.00',
                 '$179.00', '$88.00', '$93.00', '$231.00', '$51.00', '$140.00',
                 '$176.00', '$184.00', '$154.00', '$214.00', '$63.00', '$213.00',
                 '$119.00', '$76.00', '$100.00', '$186.00', '$150.00', '$162.00',
                 '$146.00', '$180.00', '$69.00', '$181.00', '$50.00', '$110.00',
                 '$286.00', '$72.00', '$67.00', '$80.00', '$92.00', '$59.00',
                 '$75.00', '$78.00', '$128.00', '$200.00', '$109.00', '$99.00',
                 '$56.00', '$91.00', '$450.00', '$131.00', '$241.00', '$44.00',
                 '$141.00', '$350.00', '$225.00', '$122.00', '$152.00', '$73.00',
                 '$132.00', '$171.00', '$160.00', '$117.00', '$84.00', '$173.00',
                 '$372.00', '$54.00', '$55.00', '$116.00', '$134.00', '$139.00',
                 '$400.00', '$96.00', '$238.00', '$149.00', '$39.00', '$97.00',
                 '$38.00', '$421.00', '$167.00', '$47.00', '$148.00', '$151.00',
                 '$40.00', '$133.00', '$156.00', '$70.00', '$271.00', '$105.00',
                 '$143.00', '$28.00', '$310.00', '$174.00', '$42.00', '$32.00',
                 '$222.00', '$37.00', '$74.00', '$550.00', '$260.00', '$198.00',
                 '$205.00', '$45.00', '$111.00', '$138.00', '$182.00', '$57.00',
                 '$98.00', '$121.00', '$104.00', '$112.00', '$248.00', '$142.00',
                 '$87.00', '$357.00', '$250.00', '$101.00', '$118.00', '$107.00',
                 '$48.00', '$1,000.00', '$102.00', '$448.00', '$103.00', '$259.00',
                 '$224.00', '$127.00', '$290.00', '$199.00', '$169.00', '$334.00',
                 '$341.00', '$380.00', '$195.00', '$407.00', '$237.00', '$89.00',
                 '$58.00', '$239.00', '$43.00', '$68.00', '$125.00', '$177.00',
                 '$123.00', '$240.00', '$191.00', '$71.00', '$190.00', '$287.00',
                 '$425.00', '$136.00', '$36.00', '$153.00', '$196.00', '$53.00',
                 '$145.00', '$29.00', '$144.00', '$215.00', '$274.00', '$245.00',
                 '$170.00', '$201.00', '$164.00', '$62.00', '$165.00', '$49.00',
                 '$183.00', '$220.00', '$77.00', '$35.00', '$319.00', '$284.00',
                 '$280.00', '$108.00', '$301.00', '$500.00', '$79.00', '$163.00',
                 '$326.00', '$234.00', '$307.00', '$115.00', '$94.00', '$416.00',
                 '$277.00', '$209.00', '$188.00', '$204.00', '$41.00', '$168.00',
                 '$197.00', '$547.00', '$178.00', '$192.00', '$219.00', '$172.00',
                 '$25.00', '$268.00', '$193.00', '$338.00', '$351.00', '$230.00',
                 '$135.00', '$252.00', '$255.00', '$52.00', '$155.00', '$356.00',
                 '$266.00', '$294.00', '$175.00', '$246.00', '$346.00', '$236.00',
                 '$507.00', '$64.00', '$285.00', '$327.00', '$282.00', '$337.00',
                 '$361.00', '$300.00', '$263.00', '$157.00', '$229.00', '$480.00',
                 '$390.00', '$313.00', '$265.00', '$147.00', '$331.00', '$256.00',
                 '$206.00', '$46.00', '$329.00', '$302.00', '$311.00', '$900.00',
                 '$316.00', '$221.00', '$1,643.00', '$189.00', '$244.00', '$314.00',
                 '$24.00', '$223.00', '$166.00', '$235.00', '$325.00', '$3,000.00',
                 '$850.00', '$17.00', '$304.00', '$471.00', '$276.00', '$333.00',
                 '$376.00', '$299.00', '$185.00', '$373.00', '$296.00', '$369.00',
                 '$30.00', '$305.00', '$343.00', '$194.00', '$2,400.00', '$308.00',
                 '$611.00', '$434.00', '$293.00', '$253.00', '$371.00', '$414.00',
                 '$264.00', '$226.00', '$330.00', '$525.00', '$251.00', '$403.00',
                 '$363.00', '$297.00', '$366.00', '$227.00', '$485.00', '$352.00',
                 '$233.00', '$767.00', '$243.00', '$158.00', '$291.00', '$232.00',
                 '$275.00', '$705.00', '$212.00', '$629.00', '$600.00', '$522.00',
                 '$309.00', '$358.00', '$570.00', '$320.00', '$33.00', '$348.00',
                 '$289.00', '$218.00', '$398.00', '$447.00', '$8,023.00', '$217.00',
                 '$436.00', '$950.00', '$315.00', '$306.00', '$283.00', '$273.00',
                 '$272.00', '$258.00', '$321.00', '$516.00', '$254.00', '$247.00',
                 '$339.00', '$360.00', '$187.00', '$1,095.00', '$438.00', '$541.00',
                 '$385.00', '$270.00', '$269.00', '$384.00', '$249.00', '$406.00',
                 '$432.00', '$345.00', '$1,786.00', '$971.00', '$535.00', '$422.00',
                 '$298.00', '$354.00', '$322.00', '$216.00', '$317.00', '$408.00',
```

```
       '$278.00', '$202.00', '$426.00', '$242.00', '$618.00'],
      dtype=object)
```

In [989…

```
# price column is the our target column to predict but it is not numeric and
# removing dollar sign and commas, then converting to float
data['price'] = data['price'].replace('[\$,]', '', regex=True).astype(float)
data.price.describe(percentiles = [0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99]).ma
```

Out[989]:

```
count    2,428.0
mean       149.9
std        219.8
min         17.0
1%          38.0
10%         60.0
25%         81.0
50%        111.0
75%        163.0
90%        258.0
99%        616.1
max      8,023.0
Name: price, dtype: object
```

- It can be seen that the minimum (17) and maximum (8,023) price range is quite wide. Additionally, the mean is 149 while the median is 111.

In [990…

```
# changing the datatype of the rate columns from object to float and removing
data['host_response_rate'] = data['host_response_rate'].str.rstrip('%').asty
data['host_acceptance_rate'] = data['host_acceptance_rate'].str.rstrip('%').a
```

In [991…

```
# checking the value counts of 'number of reviews' column
data.number_of_reviews.value_counts().sort_index()
```

Out[991]:

```
number_of_reviews
1      64
2      50
3      38
4      32
5      41
       ..
776     1
788     1
818     1
841     1
890     1
Name: count, Length: 469, dtype: int64
```

Categorical Variables

In [992…

```
# 'amenities' has empty list so we will not be able to use this column
data.amenities.unique()
```

Out[992]:

```
array(['[]'], dtype=object)
```

In [993…

```
# checking the value counts of 'room_type' column
data.room_type.value_counts()
```

Out[993]:
```
room_type
Entire home/apt    2110
Private room        285
Hotel room           33
Name: count, dtype: int64
```

In [994…
```
# since we need to focus on 'apartments', the rows where `room_type` column i
data = data[data['room_type'] != 'Hotel room']
```

In [995…
```
# checking the value counts of 'property_type' column
data.property_type.value_counts()
```

Out[995]:
```
property_type
Entire rental unit                    1430
Entire condo                           438
Private room in rental unit            143
Entire loft                            119
Entire home                             54
Private room in bed and breakfast       35
Private room in condo                   34
Entire vacation home                    24
Entire serviced apartment               22
Private room in home                    22
Entire townhouse                         9
Private room                             8
Private room in serviced apartment       7
Entire villa                             6
Room in boutique hotel                   6
Private room in loft                     5
Tiny home                                5
Private room in villa                    5
Private room in guesthouse               4
Private room in townhouse                4
Room in aparthotel                       4
Private room in casa particular          4
Entire cottage                           3
Private room in vacation home            3
Private room in guest suite              1
Name: count, dtype: int64
```

In [996…
```
# removing the rows where `property_type` column contains words 'hotel' or 'l
data = data[~data['property_type'].str.contains('hotel|hostel', case=False)]
```

- Since there are too many property types and most of them have only few observations,it makes sense to drop property types that have less 10 observations

In [997…
```
# counting the occurrences of each property type
property_counts = data['property_type'].value_counts()

# filtering out property types with less than 10 occurrences
property_types_to_keep = property_counts[property_counts >= 10].index

# filtering the DataFrame to keep only rows with property types that have at
data = data[data['property_type'].isin(property_types_to_keep)]
```

In [998…
```
# checking the unique districts in Florence
data.neighbourhood_cleansed.value_counts()
```

Out[998]:
```
neighbourhood_cleansed
Centro Storico       1906
Campo di Marte        172
Rifredi               118
Isolotto Legnaia       63
Gavinana Galluzzo      62
Name: count, dtype: int64
```

- Since 'bathrooms' column is empty we need to focus on 'bathrooms_text' column because of the fact that its a potential explanatory variable

In [999…
```
data.bathrooms_text.unique()
```

Out[999]:
```
array(['1 bath', '0 baths', '1 private bath', '2 baths', '1.5 baths',
       '3 baths', '2 shared baths', '3.5 baths', '1 shared bath',
       '2.5 baths', '4 baths', '1.5 shared baths', '3 shared baths',
       '4.5 baths'], dtype=object)
```

In [100…
```
# converting the string values to numeric values by using a mapping dictiona
mapping = {
    '1 bath' :1,
    '0 baths' :0,
    '1 private bath':1,
    '2 baths':2,
    '1.5 baths':1.5,
    '3 baths':3,
    '2 shared baths':2,
    '3.5 baths':3.5,
    '1 shared bath':1,
    '2.5 baths':2.5,
    '4 baths':4,
    '1.5 shared baths':2.5,
    '3 shared baths':3,
    '4.5 baths':4.5

}



# mapping the new values to a new column 'n_bathrooms'
data['n_bathrooms'] = data['bathrooms_text'].map(mapping)
```

In [100…
```
# checking the datatypes of 'host_is_superhost' column (f and t)
data.host_is_superhost.values
```

Out[1001]:
```
array(['f', 't', 'f', ..., 't', 't', 't'], dtype=object)
```

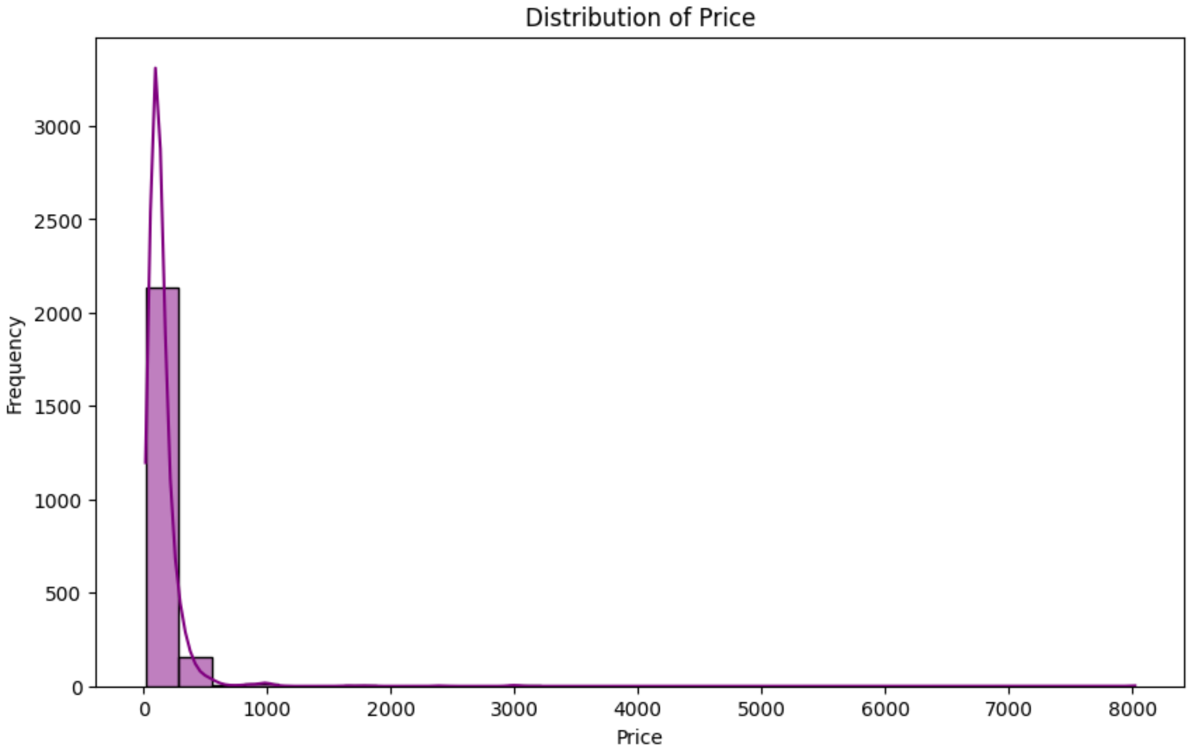
In [100…
```
# converting the 'host_is_superhost' to a dummy variable
data['host_is_superhost'] = data['host_is_superhost'].replace({'t': 1, 'f': (
data.host_is_superhost.unique()
```

Out[1002]:
```
array([0, 1])
```

Checking the Extreme Values

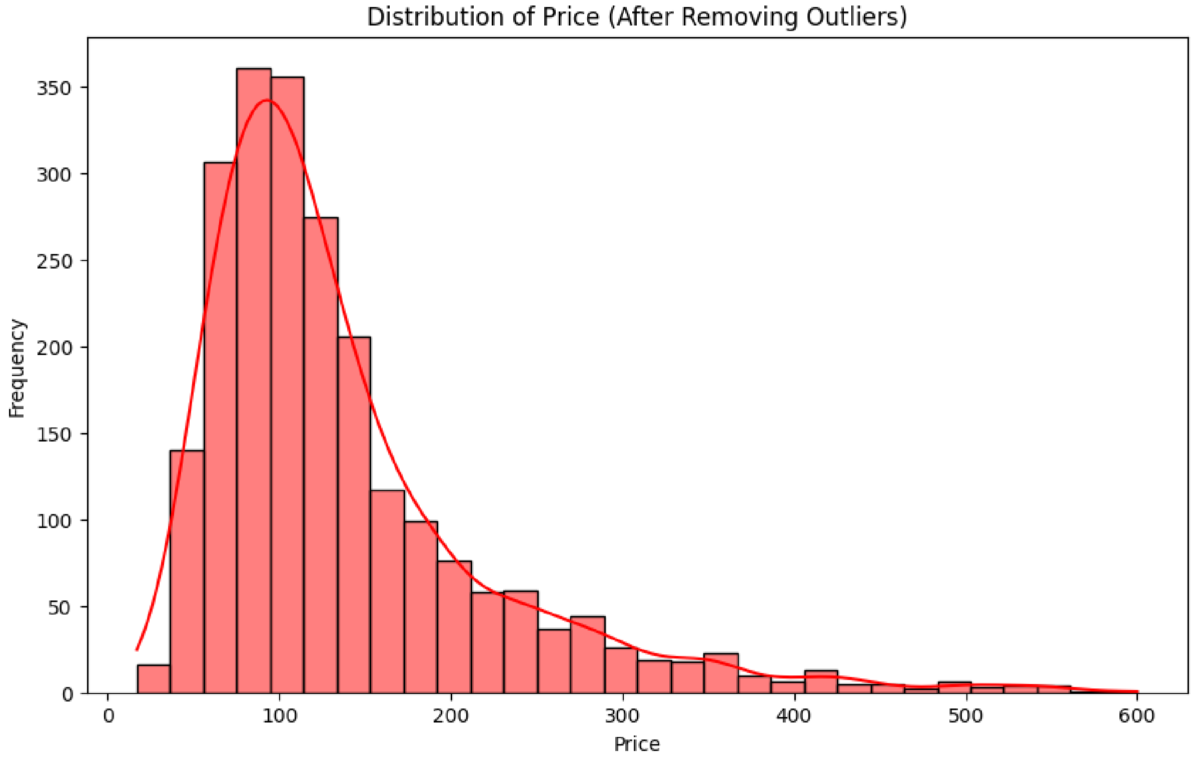
In [100…

```
import matplotlib.pyplot as plt
import seaborn as sns

# showing the distribution of 'price' column
plt.figure(figsize=(10, 6))
sns.histplot(data['price'], kde=True, color='purple', bins=30)
plt.title('Distribution of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



In [100…

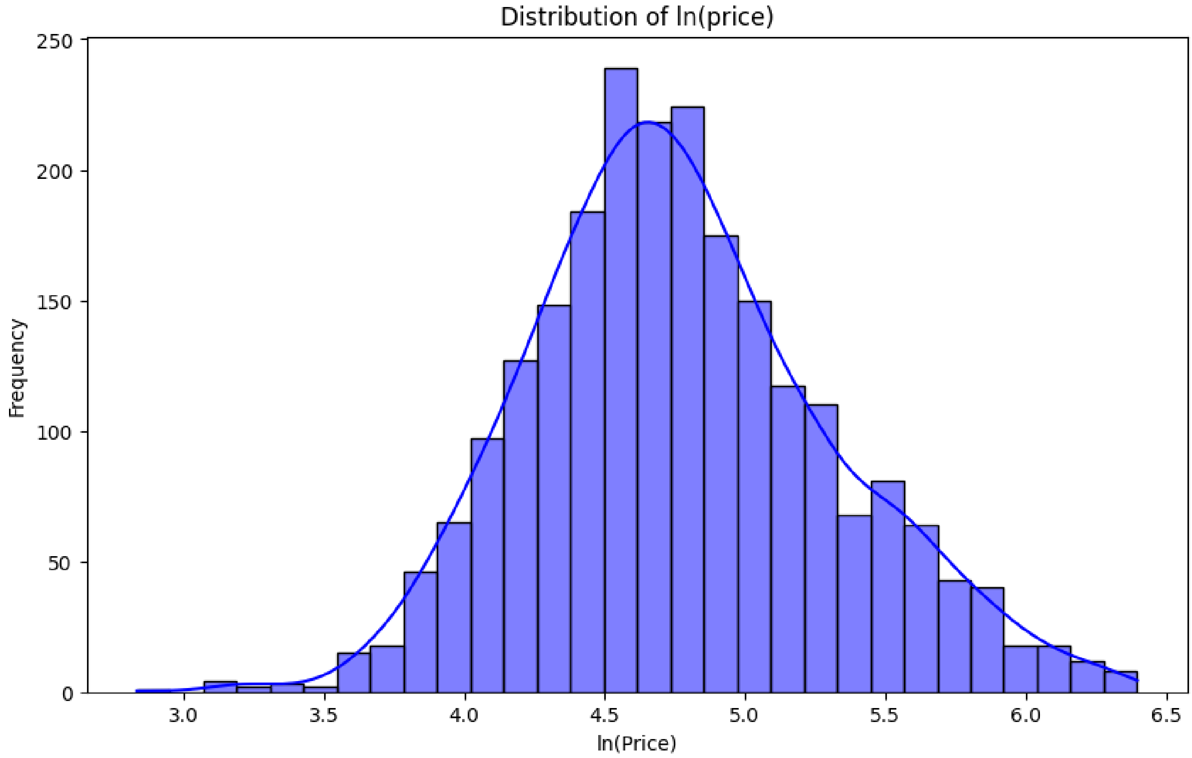
```
# defining threshold for outliers
x = data['price'].quantile(0.99)

# Remove outliers
data = data[data['price'] <= x]


# showing the distribution of 'price' column after removing outliers
plt.figure(figsize=(10, 6))
sns.histplot(data['price'], kde=True, color='red', bins=30)
plt.title('Distribution of Price (After Removing Outliers)')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



- As can be seen from the graph above, even though we dropped the extreme values, the right skewed tail came out. Hence, it might makes sense to use ln(price) to get a plot which is closer to normal distribution

In [100…

```
# Add a new column of ln(price)
data['ln_price'] = np.log(data['price'])

# Visualize distribution of ln(price) column
plt.figure(figsize=(10, 6))
sns.histplot(data['ln_price'], kde=True, color='blue', bins=30)
plt.title('Distribution of ln(price)')
plt.xlabel('ln(Price)')
plt.ylabel('Frequency')
plt.show()
```



Feature Engineering

From this point on, we need to determine which variables and interactions will be used in our models. After this step, we need to divide the data set into two separate parts as train and test (hold_out), train the model with the train data set and test the model with the test data set. The result we obtain with the test data set is the performance result of the model. These steps constitute feature engineering.

In [100…
```
# renaming the columns to have a better understanding and defining a mapping
mapping = {'host_response_time': 'f_host_response_time',
           'host_response_rate': 'n_host_response_rate',
           'host_acceptance_rate': 'n_host_acceptance_rate',
           'host_is_superhost': 'd_host_is_superhost',
           'neighbourhood_cleansed': 'f_neighbourhood_cleansed',
           'property_type': 'f_property_type',
           'room_type': 'f_room_type',
           'accommodates': 'n_accommodates',
           'beds': 'n_beds',
           'maximum_nights': 'n_maximum_nights',
           'availability_90': 'n_availability_90',
           'number_of_reviews': 'n_number_of_reviews',
           'review_scores_rating': 'n_review_scores_rating'}

# renaming the columns using the mapping dictionary
data_final = data.rename(columns=mapping)
```

In [100…
```
# splitting the data into two sets (train and test)
data_train, data_holdout = train_test_split(data_final, train_size=0.8, rand
```

In [100…
```
data_train.shape, data_holdout.shape
```

Out[1008]:
```
((1837, 71), (460, 71))
```

In [100…
```
# renaming the columns in data to be used in our analysis by adding f_, n_ an
data_final.columns
```

Out[1009]:
```
Index(['id', 'listing_url', 'scrape_id', 'last_scraped', 'source', 'name',
       'neighborhood_overview', 'picture_url', 'host_id', 'host_url',
       'host_name', 'host_since', 'host_location', 'host_about',
       'f_host_response_time', 'n_host_response_rate',
       'n_host_acceptance_rate', 'd_host_is_superhost', 'host_thumbnail_ur
l',
       'host_picture_url', 'host_neighbourhood', 'host_listings_count',
       'host_total_listings_count', 'host_verifications',
       'host_has_profile_pic', 'host_identity_verified', 'neighbourhood',
       'f_neighbourhood_cleansed', 'latitude', 'longitude', 'f_property_ty
pe',
       'f_room_type', 'n_accommodates', 'bathrooms_text', 'n_beds',
       'amenities', 'price', 'minimum_nights', 'n_maximum_nights',
       'minimum_minimum_nights', 'maximum_minimum_nights',
       'minimum_maximum_nights', 'maximum_maximum_nights',
       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'has_availabili
ty',
       'availability_30', 'availability_60', 'n_availability_90',
       'availability_365', 'calendar_last_scraped', 'n_number_of_reviews',
       'number_of_reviews_ltm', 'number_of_reviews_l30d', 'first_review',
       'last_review', 'n_review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'instant_bookable',
       'calculated_host_listings_count',
       'calculated_host_listings_count_entire_homes',
       'calculated_host_listings_count_private_rooms',
       'calculated_host_listings_count_shared_rooms', 'reviews_per_month',
       'n_bathrooms', 'ln_price'],
      dtype='object')
```

In [101…

```
# these variables were selected to compare in our ML models
basic_vars = [
    "n_accommodates",
    "n_beds",
    "f_property_type",
    "f_room_type",
    "n_bathrooms",
    "f_neighbourhood_cleansed",
    "n_availability_90", # column gives the number of days a listing is avai
    "n_maximum_nights", # ömaz number of nights to book the place
    "f_host_response_time",
    "n_host_response_rate",
    "n_host_acceptance_rate",
]

# reviews
reviews = [
    "n_number_of_reviews",
    "n_review_scores_rating",
]

# dummy variables
super_host = ["d_host_is_superhost"]

# interactions for the LASSO
X1 = [
    "n_accommodates:f_property_type",
    "f_room_type:f_property_type",
]
# with boroughs
X2 = [
    "f_property_type:f_neighbourhood_cleansed",
    "f_room_type:f_neighbourhood_cleansed",
    "n_accommodates:f_neighbourhood_cleansed",
]
```

In [101…

```
predictors_1 = basic_vars
predictors_2 = basic_vars + reviews + super_host
predictors_E = basic_vars + reviews + X1 + X2
```

Modelling

Model1: Random Forest

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

In [101…

```
# creating y-vector (dependant variable) and X-matrice (explanatory variable
y, X = dmatrices("ln_price ~ " + " + ".join(predictors_2), data_train) # we
y2, X22 = dmatrices("price ~ " + " + ".join(predictors_2), data_train)
```

In [101…

```
# X matrice is created in a way that every category in categorical variable
X
```

Out[1013]:
```
DesignMatrix with shape (1837, 28)
  Columns:
    ['Intercept',
     'f_property_type[T.Entire home]',
     'f_property_type[T.Entire loft]',
     'f_property_type[T.Entire rental unit]',
     'f_property_type[T.Entire serviced apartment]',
     'f_property_type[T.Entire vacation home]',
     'f_property_type[T.Private room in bed and breakfast]',
     'f_property_type[T.Private room in condo]',
     'f_property_type[T.Private room in home]',
     'f_property_type[T.Private room in rental unit]',
     'f_room_type[T.Private room]',
     'f_neighbourhood_cleansed[T.Centro Storico]',
     'f_neighbourhood_cleansed[T.Gavinana Galluzzo]',
     'f_neighbourhood_cleansed[T.Isolotto Legnaia]',
     'f_neighbourhood_cleansed[T.Rifredi]',
     'f_host_response_time[T.within a day]',
     'f_host_response_time[T.within a few hours]',
     'f_host_response_time[T.within an hour]',
     'n_accommodates',
     'n_beds',
     'n_bathrooms',
     'n_availability_90',
     'n_maximum_nights',
     'n_host_response_rate',
     'n_host_acceptance_rate',
     'n_number_of_reviews',
     'n_review_scores_rating',
     'd_host_is_superhost']
  Terms:
    'Intercept' (column 0)
    'f_property_type' (columns 1:10)
    'f_room_type' (column 10)
    'f_neighbourhood_cleansed' (columns 11:15)
    'f_host_response_time' (columns 15:18)
    'n_accommodates' (column 18)
    'n_beds' (column 19)
    'n_bathrooms' (column 20)
    'n_availability_90' (column 21)
    'n_maximum_nights' (column 22)
    'n_host_response_rate' (column 23)
    'n_host_acceptance_rate' (column 24)
    'n_number_of_reviews' (column 25)
    'n_review_scores_rating' (column 26)
    'd_host_is_superhost' (column 27)
  (to view full data, use np.asarray(this_obj))
```

In [101…
```
# a two-dimensional object
y.shape
```

Out[1014]: (1837, 1)

In [101…
```
# using ravel() we flatten it to a one-dimensional data object.
y.ravel().shape
```

Out[1015]: (1837,)

In [101…
```
# creating the Random Forest Regressor of sklearn package
rfr = RandomForestRegressor(random_state = 42)
```

In [101…
```
tune_grid = {"max_features": [6, 8, 10, 12], "min_samples_leaf": [5, 10, 15]
```

In [101…
```
# creating regressors for level prices and one for log prices
# finding the best values for hyperparameters of a model

rf_random = GridSearchCV(
    rfr,
    tune_grid,
    cv=5,
    scoring="neg_root_mean_squared_error",
    verbose=3,
)


rf_random2 = GridSearchCV(
    rfr,
    tune_grid,
    cv=5,
    scoring="neg_root_mean_squared_error",
    verbose=3,
)
```

`GridsearchCV()` is an exhaustive search over specified parameter values for an estimator.

Cross-validated results are saved in the grid search object's `cv_results_` attribute. *RMSE* is displayed as a negative number.

In [101…
```
%%time
rf_model_log = rf_random.fit(X, y.ravel())
rf_model_level = rf_random2.fit(X22, y2.ravel())
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV 1/5] END max_features=6, min_samples_leaf=5;, score=-0.407 total time=
0.1s
[CV 2/5] END max_features=6, min_samples_leaf=5;, score=-0.398 total time=
0.1s
[CV 3/5] END max_features=6, min_samples_leaf=5;, score=-0.396 total time=
0.1s
[CV 4/5] END max_features=6, min_samples_leaf=5;, score=-0.396 total time=
0.1s
[CV 5/5] END max_features=6, min_samples_leaf=5;, score=-0.409 total time=
0.1s
[CV 1/5] END max_features=6, min_samples_leaf=10;, score=-0.416 total time=
0.1s
[CV 2/5] END max_features=6, min_samples_leaf=10;, score=-0.410 total time=
0.1s
[CV 3/5] END max_features=6, min_samples_leaf=10;, score=-0.402 total time=
0.1s
[CV 4/5] END max_features=6, min_samples_leaf=10;, score=-0.403 total time=
0.1s
[CV 5/5] END max_features=6, min_samples_leaf=10;, score=-0.421 total time=
0.1s
[CV 1/5] END max_features=6, min_samples_leaf=15;, score=-0.421 total time=
0.1s
[CV 2/5] END max_features=6, min_samples_leaf=15;, score=-0.416 total time=
0.1s
[CV 3/5] END max_features=6, min_samples_leaf=15;, score=-0.410 total time=
0.1s
[CV 4/5] END max_features=6, min_samples_leaf=15;, score=-0.407 total time=
0.1s
[CV 5/5] END max_features=6, min_samples_leaf=15;, score=-0.428 total time=
0.1s
[CV 1/5] END max_features=8, min_samples_leaf=5;, score=-0.406 total time=
0.1s
[CV 2/5] END max_features=8, min_samples_leaf=5;, score=-0.400 total time=
0.1s
[CV 3/5] END max_features=8, min_samples_leaf=5;, score=-0.394 total time=
0.1s
[CV 4/5] END max_features=8, min_samples_leaf=5;, score=-0.397 total time=
0.1s
[CV 5/5] END max_features=8, min_samples_leaf=5;, score=-0.412 total time=
0.1s
[CV 1/5] END max_features=8, min_samples_leaf=10;, score=-0.408 total time=
0.1s
[CV 2/5] END max_features=8, min_samples_leaf=10;, score=-0.409 total time=
0.1s
[CV 3/5] END max_features=8, min_samples_leaf=10;, score=-0.399 total time=
0.1s
[CV 4/5] END max_features=8, min_samples_leaf=10;, score=-0.400 total time=
0.1s
[CV 5/5] END max_features=8, min_samples_leaf=10;, score=-0.418 total time=
0.1s
[CV 1/5] END max_features=8, min_samples_leaf=15;, score=-0.416 total time=
0.1s
[CV 2/5] END max_features=8, min_samples_leaf=15;, score=-0.415 total time=
0.1s
[CV 3/5] END max_features=8, min_samples_leaf=15;, score=-0.404 total time=
0.1s
[CV 4/5] END max_features=8, min_samples_leaf=15;, score=-0.405 total time=
0.1s
[CV 5/5] END max_features=8, min_samples_leaf=15;, score=-0.423 total time=
0.1s
```

```
[CV 1/5] END max_features=10, min_samples_leaf=5;, score=-0.404 total time=
0.1s
[CV 2/5] END max_features=10, min_samples_leaf=5;, score=-0.402 total time=
0.1s
[CV 3/5] END max_features=10, min_samples_leaf=5;, score=-0.398 total time=
0.1s
[CV 4/5] END max_features=10, min_samples_leaf=5;, score=-0.396 total time=
0.1s
[CV 5/5] END max_features=10, min_samples_leaf=5;, score=-0.409 total time=
0.1s
[CV 1/5] END max_features=10, min_samples_leaf=10;, score=-0.412 total time=
0.1s
[CV 2/5] END max_features=10, min_samples_leaf=10;, score=-0.411 total time=
0.1s
[CV 3/5] END max_features=10, min_samples_leaf=10;, score=-0.400 total time=
0.1s
[CV 4/5] END max_features=10, min_samples_leaf=10;, score=-0.406 total time=
0.1s
[CV 5/5] END max_features=10, min_samples_leaf=10;, score=-0.417 total time=
0.1s
[CV 1/5] END max_features=10, min_samples_leaf=15;, score=-0.416 total time=
0.1s
[CV 2/5] END max_features=10, min_samples_leaf=15;, score=-0.411 total time=
0.1s
[CV 3/5] END max_features=10, min_samples_leaf=15;, score=-0.403 total time=
0.1s
[CV 4/5] END max_features=10, min_samples_leaf=15;, score=-0.406 total time=
0.1s
[CV 5/5] END max_features=10, min_samples_leaf=15;, score=-0.420 total time=
0.1s
[CV 1/5] END max_features=12, min_samples_leaf=5;, score=-0.405 total time=
0.1s
[CV 2/5] END max_features=12, min_samples_leaf=5;, score=-0.399 total time=
0.1s
[CV 3/5] END max_features=12, min_samples_leaf=5;, score=-0.393 total time=
0.1s
[CV 4/5] END max_features=12, min_samples_leaf=5;, score=-0.396 total time=
0.1s
[CV 5/5] END max_features=12, min_samples_leaf=5;, score=-0.407 total time=
0.1s
[CV 1/5] END max_features=12, min_samples_leaf=10;, score=-0.406 total time=
0.1s
[CV 2/5] END max_features=12, min_samples_leaf=10;, score=-0.407 total time=
0.1s
[CV 3/5] END max_features=12, min_samples_leaf=10;, score=-0.399 total time=
0.1s
[CV 4/5] END max_features=12, min_samples_leaf=10;, score=-0.401 total time=
0.1s
[CV 5/5] END max_features=12, min_samples_leaf=10;, score=-0.413 total time=
0.1s
[CV 1/5] END max_features=12, min_samples_leaf=15;, score=-0.411 total time=
0.1s
[CV 2/5] END max_features=12, min_samples_leaf=15;, score=-0.413 total time=
0.1s
[CV 3/5] END max_features=12, min_samples_leaf=15;, score=-0.403 total time=
0.1s
[CV 4/5] END max_features=12, min_samples_leaf=15;, score=-0.406 total time=
0.1s
[CV 5/5] END max_features=12, min_samples_leaf=15;, score=-0.417 total time=
0.1s
Fitting 5 folds for each of 12 candidates, totalling 60 fits
```

```
[CV 1/5] END max_features=6, min_samples_leaf=5;, score=-73.971 total time=
0.1s
[CV 2/5] END max_features=6, min_samples_leaf=5;, score=-64.258 total time=
0.1s
[CV 3/5] END max_features=6, min_samples_leaf=5;, score=-63.333 total time=
0.1s
[CV 4/5] END max_features=6, min_samples_leaf=5;, score=-62.726 total time=
0.1s
[CV 5/5] END max_features=6, min_samples_leaf=5;, score=-70.107 total time=
0.1s
[CV 1/5] END max_features=6, min_samples_leaf=10;, score=-74.588 total time=
0.1s
[CV 2/5] END max_features=6, min_samples_leaf=10;, score=-65.442 total time=
0.1s
[CV 3/5] END max_features=6, min_samples_leaf=10;, score=-64.430 total time=
0.1s
[CV 4/5] END max_features=6, min_samples_leaf=10;, score=-63.998 total time=
0.1s
[CV 5/5] END max_features=6, min_samples_leaf=10;, score=-71.476 total time=
0.1s
[CV 1/5] END max_features=6, min_samples_leaf=15;, score=-75.624 total time=
0.1s
[CV 2/5] END max_features=6, min_samples_leaf=15;, score=-66.175 total time=
0.1s
[CV 3/5] END max_features=6, min_samples_leaf=15;, score=-65.253 total time=
0.1s
[CV 4/5] END max_features=6, min_samples_leaf=15;, score=-64.908 total time=
0.1s
[CV 5/5] END max_features=6, min_samples_leaf=15;, score=-71.973 total time=
0.1s
[CV 1/5] END max_features=8, min_samples_leaf=5;, score=-73.486 total time=
0.1s
[CV 2/5] END max_features=8, min_samples_leaf=5;, score=-65.020 total time=
0.1s
[CV 3/5] END max_features=8, min_samples_leaf=5;, score=-63.371 total time=
0.1s
[CV 4/5] END max_features=8, min_samples_leaf=5;, score=-62.951 total time=
0.1s
[CV 5/5] END max_features=8, min_samples_leaf=5;, score=-69.903 total time=
0.1s
[CV 1/5] END max_features=8, min_samples_leaf=10;, score=-74.141 total time=
0.1s
[CV 2/5] END max_features=8, min_samples_leaf=10;, score=-66.016 total time=
0.1s
[CV 3/5] END max_features=8, min_samples_leaf=10;, score=-64.112 total time=
0.1s
[CV 4/5] END max_features=8, min_samples_leaf=10;, score=-63.481 total time=
0.1s
[CV 5/5] END max_features=8, min_samples_leaf=10;, score=-70.987 total time=
0.1s
[CV 1/5] END max_features=8, min_samples_leaf=15;, score=-74.533 total time=
0.1s
[CV 2/5] END max_features=8, min_samples_leaf=15;, score=-66.113 total time=
0.1s
[CV 3/5] END max_features=8, min_samples_leaf=15;, score=-64.739 total time=
0.1s
[CV 4/5] END max_features=8, min_samples_leaf=15;, score=-64.326 total time=
0.1s
[CV 5/5] END max_features=8, min_samples_leaf=15;, score=-71.497 total time=
0.1s
[CV 1/5] END max_features=10, min_samples_leaf=5;, score=-73.764 total time=
```

```
0.1s
[CV 2/5] END max_features=10, min_samples_leaf=5;, score=-65.084 total time=
0.1s
[CV 3/5] END max_features=10, min_samples_leaf=5;, score=-62.778 total time=
0.1s
[CV 4/5] END max_features=10, min_samples_leaf=5;, score=-62.736 total time=
0.1s
[CV 5/5] END max_features=10, min_samples_leaf=5;, score=-69.723 total time=
0.1s
[CV 1/5] END max_features=10, min_samples_leaf=10;, score=-74.355 total time
=   0.1s
[CV 2/5] END max_features=10, min_samples_leaf=10;, score=-65.541 total time
=   0.1s
[CV 3/5] END max_features=10, min_samples_leaf=10;, score=-63.824 total time
=   0.1s
[CV 4/5] END max_features=10, min_samples_leaf=10;, score=-63.769 total time
=   0.1s
[CV 5/5] END max_features=10, min_samples_leaf=10;, score=-70.745 total time
=   0.1s
[CV 1/5] END max_features=10, min_samples_leaf=15;, score=-75.057 total time
=   0.1s
[CV 2/5] END max_features=10, min_samples_leaf=15;, score=-66.438 total time
=   0.1s
[CV 3/5] END max_features=10, min_samples_leaf=15;, score=-64.377 total time
=   0.1s
[CV 4/5] END max_features=10, min_samples_leaf=15;, score=-64.703 total time
=   0.1s
[CV 5/5] END max_features=10, min_samples_leaf=15;, score=-71.236 total time
=   0.1s
[CV 1/5] END max_features=12, min_samples_leaf=5;, score=-73.923 total time=
0.1s
[CV 2/5] END max_features=12, min_samples_leaf=5;, score=-65.380 total time=
0.1s
[CV 3/5] END max_features=12, min_samples_leaf=5;, score=-63.039 total time=
0.1s
[CV 4/5] END max_features=12, min_samples_leaf=5;, score=-62.540 total time=
0.1s
[CV 5/5] END max_features=12, min_samples_leaf=5;, score=-69.935 total time=
0.1s
[CV 1/5] END max_features=12, min_samples_leaf=10;, score=-74.299 total time
=   0.1s
[CV 2/5] END max_features=12, min_samples_leaf=10;, score=-65.680 total time
=   0.1s
[CV 3/5] END max_features=12, min_samples_leaf=10;, score=-63.666 total time
=   0.1s
[CV 4/5] END max_features=12, min_samples_leaf=10;, score=-63.532 total time
=   0.1s
[CV 5/5] END max_features=12, min_samples_leaf=10;, score=-70.318 total time
=   0.1s
[CV 1/5] END max_features=12, min_samples_leaf=15;, score=-74.470 total time
=   0.1s
[CV 2/5] END max_features=12, min_samples_leaf=15;, score=-66.176 total time
=   0.1s
[CV 3/5] END max_features=12, min_samples_leaf=15;, score=-64.142 total time
=   0.1s
[CV 4/5] END max_features=12, min_samples_leaf=15;, score=-64.321 total time
=   0.1s
[CV 5/5] END max_features=12, min_samples_leaf=15;, score=-71.042 total time
=   0.1s
CPU times: user 12.6 s, sys: 55.2 ms, total: 12.6 s
Wall time: 12.6 s
```

In [102…
```
df_rf_model_cv_results = pd.DataFrame(rf_model_log.cv_results_)[[
    'param_max_features', 'param_min_samples_leaf', 'mean_test_score']]
df_rf_model2_cv_results = pd.DataFrame(rf_model_level.cv_results_)[[
    'param_max_features', 'param_min_samples_leaf', 'mean_test_score']]
```

In [102…
```
# renaming columns of the results of first random forest model (df_rf_model w
df_rf_model_cv_results.columns = ['max features', 'min node size', 'RMSE_ln']

# calculating the mean price
mean_price_ln = np.mean(y)
mean_price_level = np.mean(y2)

# calculating prediction error percentage
rmse_values_ln = -df_rf_model_cv_results['RMSE_ln']
df_rf_model_cv_results['RMSE_ln'] = rmse_values_ln
error_percentage_ln = (rmse_values_ln / mean_price_ln)

rmse_values_level = -df_rf_model2_cv_results['mean_test_score']
error_percentage_level = (rmse_values_level / mean_price_level)

# adding prediction error percentage to the DataFrame
df_rf_model_cv_results['RMSE_percentage_ln'] = error_percentage_ln

# adding RMSE of level price as y to the DataFrame
df_rf_model_cv_results['RMSE_level'] = rmse_values_level

# adding prediction error percentage of level price as y to the DataFrame
df_rf_model_cv_results['RMSE_percentage_level'] = error_percentage_level
```

In [102…
```
df_rf_model_cv_results
```

Out[1022]:

	max features	min node size	RMSE_ln	RMSE_percentage_ln	RMSE_level	RMSE_percentage_level
0	6	5	0.401256	0.084278	66.879155	0.491069
1	6	10	0.410373	0.086193	67.986652	0.499201
2	6	15	0.416263	0.087431	68.786722	0.505075
3	8	5	0.401734	0.084379	66.946072	0.491560
4	8	10	0.406613	0.085404	67.747330	0.497443
5	8	15	0.412805	0.086704	68.241602	0.501073
6	10	5	0.401679	0.084367	66.816975	0.490612
7	10	10	0.409051	0.085916	67.646537	0.496703
8	10	15	0.411237	0.086375	68.362280	0.501959
9	12	5	0.400065	0.084029	66.963358	0.491687
10	12	10	0.405130	0.085092	67.498934	0.495619
11	12	15	0.410045	0.086125	68.030234	0.499521

RMSE for level prices for different models of Random Forest is between 66 and 68. And comparing these values to mean prices, we get approximately 50% error. For log transformed predictions, this raito is around 8%. Although, Random Forest is said to not sensitive to non-normality, the results are a bit confusing.

In [102…
```
print(-rf_model_level.best_score_)
print(rf_model_level.best_params_)
```

```
66.81697525672766
{'max_features': 10, 'min_samples_leaf': 5}
```

- The best Random Forest model has minimum 5 an observations in a terminal node and with max 10 number of features that are considered when splitting a node

In [102…
```
rf_model_level.best_estimator_
```

Out[1024]:
```
▾ RandomForestRegressor

RandomForestRegressor(max_features=10, min_samples_leaf=5, random_
state=42)
```

We have used some different columns than London data in our analysis and found similar results. The best model hast 10 features and 5 leafs similar to London data. Also, we found near proportion between RMSE values and the mean acc. to London results.

We also applied $\log(\text{price})$ as dependant variable and compared the results with level-prices. Random forest finds the best model with same specifications but `RMSE/mean(price)` ratio is very different in case of log-transformed prices as `y-variable`.

Model1 Diagnostics

In [102…
```
rf_model_level.best_estimator_.feature_importances_
```

Out[1025]:
```
array([0.00000000e+00, 8.48171750e-04, 1.78904139e-03, 1.07254255e-02,
       4.68877969e-04, 5.53344136e-06, 8.72031558e-04, 6.69816500e-04,
       1.48719208e-04, 1.83709680e-02, 3.19239568e-02, 4.05974894e-02,
       2.15164813e-04, 1.04700307e-03, 2.66617736e-03, 1.37747011e-03,
       2.39300493e-03, 5.00098074e-03, 9.27754363e-02, 9.78548639e-02,
       2.51491240e-01, 9.63047386e-02, 5.64840976e-02, 2.65505318e-02,
       3.49821858e-02, 1.12446118e-01, 9.60015931e-02, 1.59893627e-02])
```

In [102…
```
X.design_info.column_names
```

Out[1026]:
```
['Intercept',
 'f_property_type[T.Entire home]',
 'f_property_type[T.Entire loft]',
 'f_property_type[T.Entire rental unit]',
 'f_property_type[T.Entire serviced apartment]',
 'f_property_type[T.Entire vacation home]',
 'f_property_type[T.Private room in bed and breakfast]',
 'f_property_type[T.Private room in condo]',
 'f_property_type[T.Private room in home]',
 'f_property_type[T.Private room in rental unit]',
 'f_room_type[T.Private room]',
 'f_neighbourhood_cleansed[T.Centro Storico]',
 'f_neighbourhood_cleansed[T.Gavinana Galluzzo]',
 'f_neighbourhood_cleansed[T.Isolotto Legnaia]',
 'f_neighbourhood_cleansed[T.Rifredi]',
 'f_host_response_time[T.within a day]',
 'f_host_response_time[T.within a few hours]',
 'f_host_response_time[T.within an hour]',
 'n_accommodates',
 'n_beds',
 'n_bathrooms',
 'n_availability_90',
 'n_maximum_nights',
 'n_host_response_rate',
 'n_host_acceptance_rate',
 'n_number_of_reviews',
 'n_review_scores_rating',
 'd_host_is_superhost']
```

In [102…
```
df_var_imp = pd.DataFrame(
    rf_model_level.best_estimator_.feature_importances_,
    X.design_info.column_names)\
    .reset_index()\
    .rename({"index": "variable", 0: "imp"}, axis=1)\
    .sort_values(by=["imp"], ascending=False)\
    .reset_index(drop = True)

df_var_imp['cumulative_imp'] = df_var_imp['imp'].cumsum()
```

In [102…
```
df_var_imp.style.format({
    'imp': lambda x: f'{x:,.1%}',
    'cumulative_imp': lambda x: f'{x:,.1%}'})
```

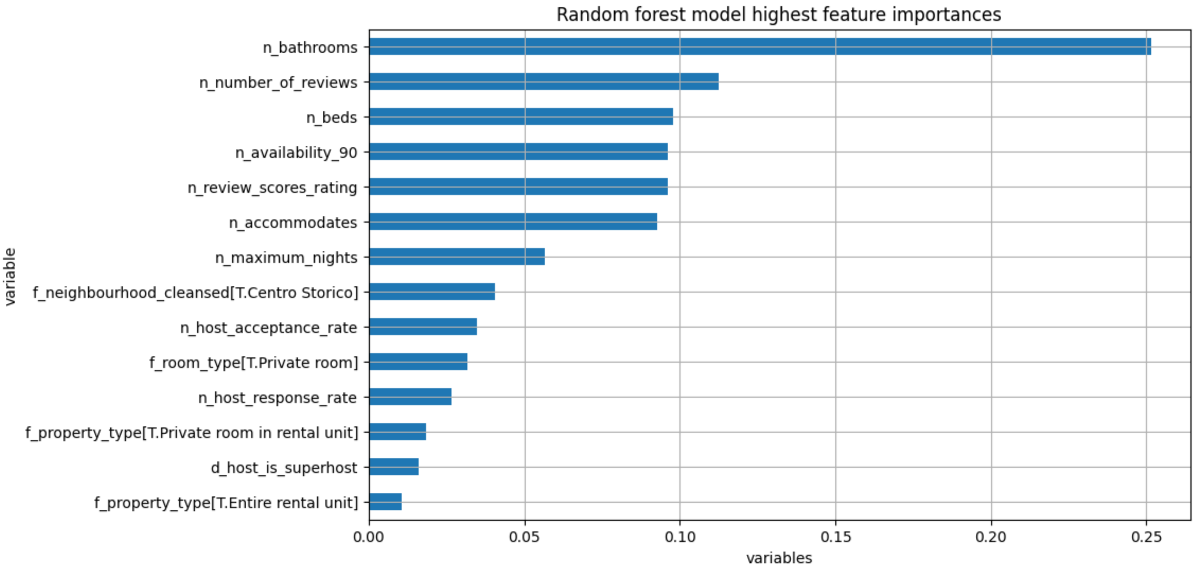
Out[1028]:

	variable	imp	cumulative_imp
0	n_bathrooms	25.1%	25.1%
1	n_number_of_reviews	11.2%	36.4%
2	n_beds	9.8%	46.2%
3	n_availability_90	9.6%	55.8%
4	n_review_scores_rating	9.6%	65.4%
5	n_accommodates	9.3%	74.7%
6	n_maximum_nights	5.6%	80.3%
7	f_neighbourhood_cleansed[T.Centro Storico]	4.1%	84.4%
8	n_host_acceptance_rate	3.5%	87.9%
9	f_room_type[T.Private room]	3.2%	91.1%
10	n_host_response_rate	2.7%	93.7%
11	f_property_type[T.Private room in rental unit]	1.8%	95.6%
12	d_host_is_superhost	1.6%	97.2%
13	f_property_type[T.Entire rental unit]	1.1%	98.2%
14	f_host_response_time[T.within an hour]	0.5%	98.7%
15	f_neighbourhood_cleansed[T.Rifredi]	0.3%	99.0%
16	f_host_response_time[T.within a few hours]	0.2%	99.3%
17	f_property_type[T.Entire loft]	0.2%	99.4%
18	f_host_response_time[T.within a day]	0.1%	99.6%
19	f_neighbourhood_cleansed[T.Isolotto Legnaia]	0.1%	99.7%
20	f_property_type[T.Private room in bed and breakfast]	0.1%	99.8%
21	f_property_type[T.Entire home]	0.1%	99.8%
22	f_property_type[T.Private room in condo]	0.1%	99.9%
23	f_property_type[T.Entire serviced apartment]	0.0%	100.0%
24	f_neighbourhood_cleansed[T.Gavinana Galluzzo]	0.0%	100.0%
25	f_property_type[T.Private room in home]	0.0%	100.0%
26	f_property_type[T.Entire vacation home]	0.0%	100.0%
27	Intercept	0.0%	100.0%

In terms of individual importances, `n_bathrooms` that can stay at the listing have highest importance.

In [102…
```
# we only care for variables with an importance of more than 1%
cutoff = 0.01
```

In [103…
```
df_var_imp[df_var_imp.imp > cutoff]\
    .sort_values(by = 'imp')\
    .plot(kind = 'barh',
          x = 'variable', y = 'imp',
          figsize = (10,6), grid = True,
          title = 'Random forest model highest feature importances',
          xlabel = 'variables', legend = False
         );
```



Now we group variable importance. In other words, instead of looking at the individual categories of a categorical variable, we look at the total effect of the category.

In [103…
```
categorical_columns = [col for col in predictors_2 if col.startswith("f_")]
numerical_columns = [col for col in predictors_2 if col not in categorical_co
```

In [103…
```
categorical_encoder = OneHotEncoder(handle_unknown="ignore")

preprocessing = ColumnTransformer([
    ("cat", categorical_encoder, categorical_columns),
    ("num", "passthrough", numerical_columns)])

rf_pipeline = Pipeline(
    [("preprocess", preprocessing),
     ("regressor", rf_model_level.best_estimator_)] # for level
)

rf_pipeline2 = Pipeline(
    [("preprocess", preprocessing),
     ("regressor", rf_model_log.best_estimator_)] # for log
)
```

In [103…
```
%%time
rf_pipeline.fit(data_train[predictors_2],data_train.price)
```

```
CPU times: user 457 ms, sys: 3.81 ms, total: 461 ms
Wall time: 159 ms
```

Out[1033]:

```
▸ Pipeline
▸ preprocess: ColumnTransformer
▸ cat                ▸ num
▸ OneHotEncoder      ▸ passthrough
▸ RandomForestRegressor
```

In [103…

```
%%time
rf_pipeline2.fit(data_train[predictors_2],data_train.ln_price)
```

```
CPU times: user 166 ms, sys: 1.96 ms, total: 168 ms
Wall time: 167 ms
```

Out[1034]:

```
▸ Pipeline
▸ preprocess: ColumnTransformer
▸ cat                ▸ num
▸ OneHotEncoder      ▸ passthrough
▸ RandomForestRegressor
```

In [103…

```
%%time
result = permutation_importance(
    rf_pipeline,
    data_holdout[predictors_2],
    data_holdout.price,
    n_repeats=10,
    random_state=45,
)
```

```
CPU times: user 738 ms, sys: 3.28 ms, total: 742 ms
Wall time: 741 ms
```

In [103…

```
pd.DataFrame(
        result.importances_mean,
        data_train[predictors_2].columns)
```

Out[1036]:

	0
n_accommodates	0.040229
n_beds	0.035192
f_property_type	0.003505
f_room_type	0.040206
n_bathrooms	0.205473
f_neighbourhood_cleansed	0.026416
n_availability_90	0.011035
n_maximum_nights	0.022245
f_host_response_time	-0.000248
n_host_response_rate	0.021490
n_host_acceptance_rate	0.002834
n_number_of_reviews	0.053304
n_review_scores_rating	0.027173
d_host_is_superhost	0.005743

In [103…

```
grouped = [
    "n_beds",
    "f_property_type",
    "f_room_type",
    "n_accommodates",
    "n_bathrooms",
    "f_neighbourhood_cleansed",
    "n_availability_90",
    "n_maximum_nights",
    "f_host_response_time",
    "n_host_response_rate",
    "n_host_acceptance_rate",
    "n_number_of_reviews",
    "n_review_scores_rating",
    "d_host_is_superhost"
]
```

In [103…

```
df_grouped_var_imp = pd.DataFrame(
        result.importances_mean,
        data_train[predictors_2].columns)\
    .loc[grouped]\
    .sort_values(by = 0, ascending = False)\
    .reset_index()\
    .rename({'index': 'variable', 0: 'imp'}, axis = 1)
df_grouped_var_imp['cumulative_imp'] = df_grouped_var_imp.imp.cumsum()
```

In [103…

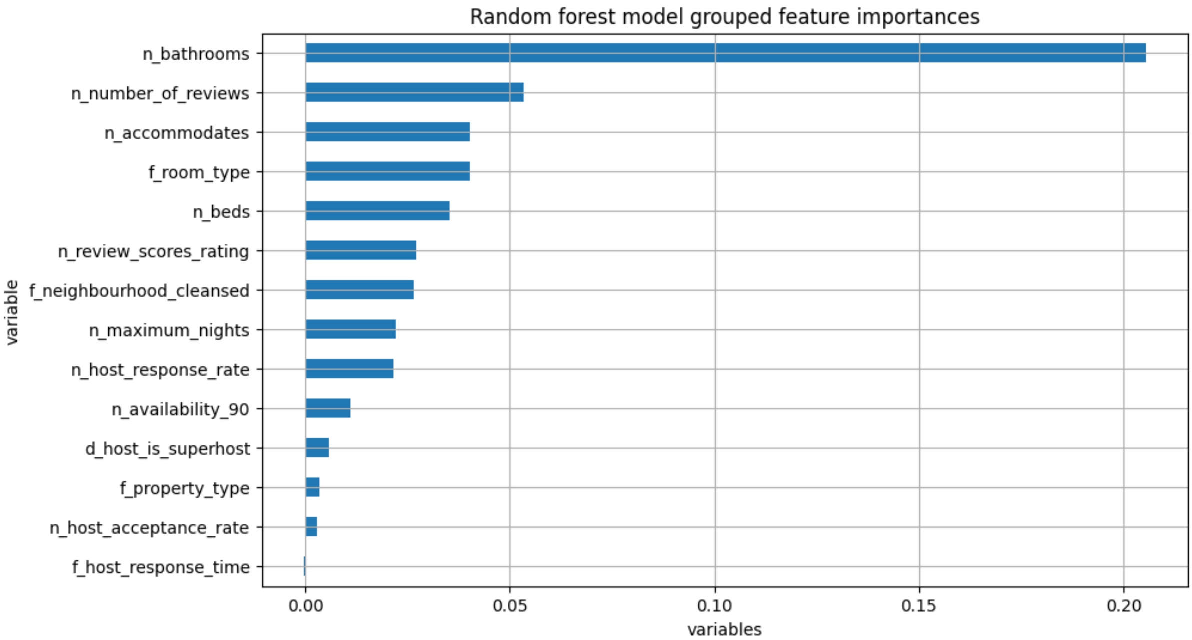
```
df_grouped_var_imp.style.format({
    'imp': lambda x: f'{x:,.1%}',
    'cumulative_imp': lambda x: f'{x:,.1%}'})
```

Out[1039]:

	variable	imp	cumulative_imp
0	n_bathrooms	20.5%	20.5%
1	n_number_of_reviews	5.3%	25.9%
2	n_accommodates	4.0%	29.9%
3	f_room_type	4.0%	33.9%
4	n_beds	3.5%	37.4%
5	n_review_scores_rating	2.7%	40.2%
6	f_neighbourhood_cleansed	2.6%	42.8%
7	n_maximum_nights	2.2%	45.0%
8	n_host_response_rate	2.1%	47.2%
9	n_availability_90	1.1%	48.3%
10	d_host_is_superhost	0.6%	48.9%
11	f_property_type	0.4%	49.2%
12	n_host_acceptance_rate	0.3%	49.5%
13	f_host_response_time	-0.0%	49.5%

In [104…

```
df_grouped_var_imp\
    .sort_values(by = 'imp')\
    .plot(kind = 'barh',
          x = 'variable', y = 'imp',
          figsize = (10,6), grid = True,
          title = 'Random forest model grouped feature importances',
          xlabel = 'variables', legend = False
         );
```

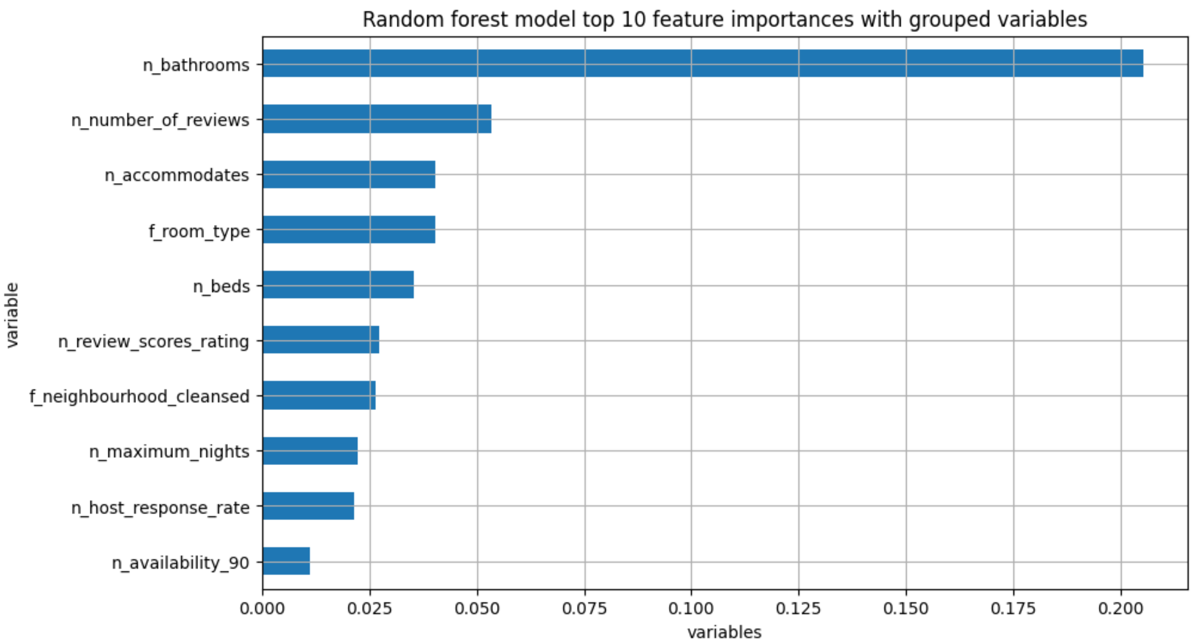


In [104…
```
df_clean_varimp = pd.DataFrame(
        result.importances_mean,
        data_train[predictors_2].columns)\
    .sort_values(by = 0, ascending = False)\
    .reset_index()\
    .rename({'index': 'variable', 0: 'imp'}, axis = 1)

df_clean_varimp['cumulative_imp'] = df_var_imp['imp'].cumsum()
df_clean_varimp[df_clean_varimp.cumulative_imp < 0.91]
```

Out[1041]:

	variable	imp	cumulative_imp
0	n_bathrooms	0.205473	0.251491
1	n_number_of_reviews	0.053304	0.363937
2	n_accommodates	0.040229	0.461792
3	f_room_type	0.040206	0.558097
4	n_beds	0.035192	0.654099
5	n_review_scores_rating	0.027173	0.746874
6	f_neighbourhood_cleansed	0.026416	0.803358
7	n_maximum_nights	0.022245	0.843956
8	n_host_response_rate	0.021490	0.878938

In [104…
```
df_clean_varimp.iloc[0:10]\
    .sort_values(by = 'imp')\
    .plot(kind = 'barh',
          x = 'variable', y = 'imp',
          figsize = (10,6), grid = True,
          title = 'Random forest model top 10 feature importances with groupe
          xlabel = 'variables', legend = False
        );
```



Variable importances analysis show that, the most important factor (20%)for apartments in Florence is the 'n_baths' that the apartment can host. `n_number_reviews` and `n_accommodates` are the 2nd and 3th important factors, respectively.

Partial dependence plots

Partial dependence plots show the dependence between the target function and a set of features of interest, marginalizing over the values of all other features (the complement features). While feature importance shows what variables most affect predictions, partial dependence plots show how a feature affects predictions.

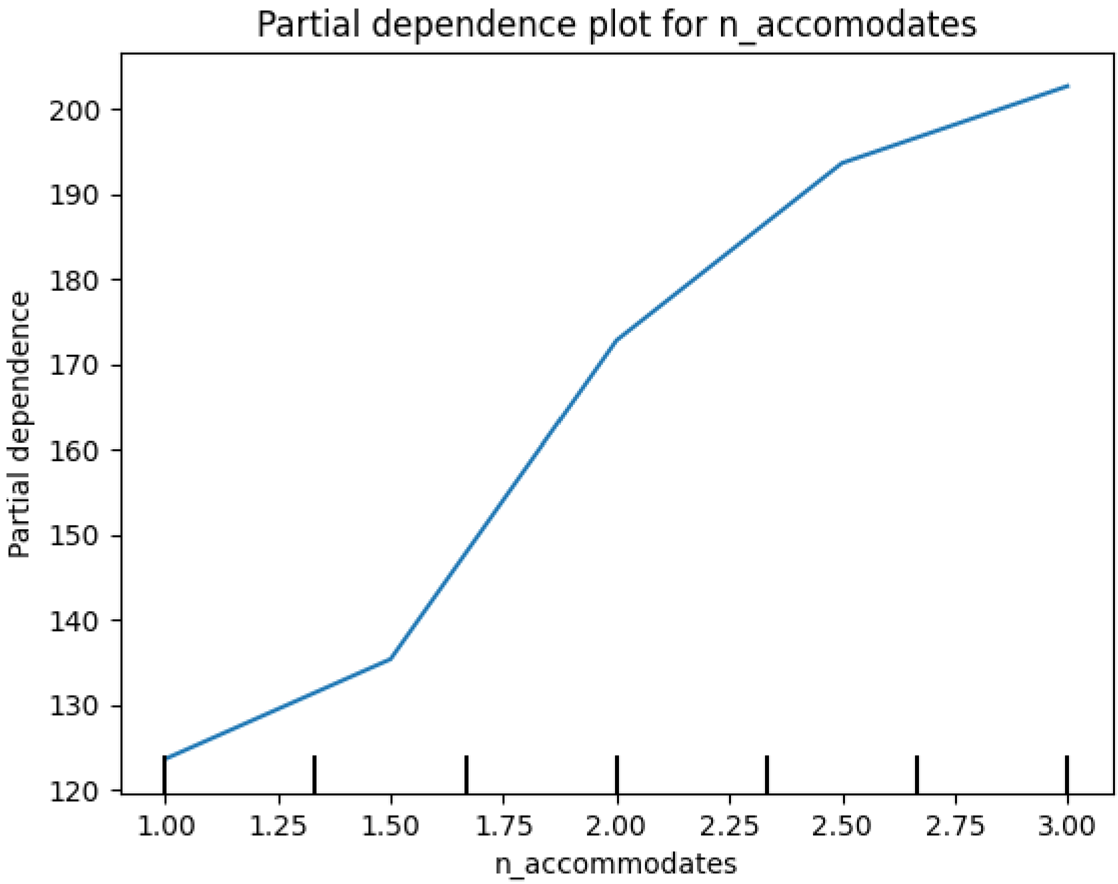
In [104…
```
# checking how number of accommodates (found to most important factor) affec
bathrooms_pdp = partial_dependence(
    rf_pipeline, data_holdout[predictors_2], ["n_bathrooms"], kind="average"
)
```

In [104…
```
pd.DataFrame(
    {'number of bathrooms': bathrooms_pdp['values'][0],
     'average price': bathrooms_pdp['average'][0]}
    )
```

Out[1044]:

	number of bathrooms	average price
0	1.0	123.604621
1	1.5	135.346837
2	2.0	172.758288
3	2.5	193.596473
4	3.0	202.642299

In [110…
```
display = PartialDependenceDisplay(
    pd_results = [bathrooms_pdp],
    features = [(0,)],
    feature_names = data_holdout[predictors_2].columns.tolist(),
    target_idx = 0,
    deciles = {0: np.linspace(1, 3, num=7)}
)
display.plot()
plt.title('Partial dependence plot for n_accomodates')
plt.show();
```


Partial dependence plot for n_accomodates
Partial dependence
200
190
180
170
160
150
140
130
120
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
n_accommodates

```
[CV 3/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 3/5; 1/4] END max_depth=5, n_estimators=200;, score=-61.478 total time=
0.2s
[CV 3/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 3/5; 3/4] END max_depth=10, n_estimators=200;, score=-64.226 total time=
0.3s
[CV 5/5; 4/4] START max_depth=10, n_estimators=30
0..............................
[CV 5/5; 4/4] END max_depth=10, n_estimators=300;, score=-71.261 total time=
0.4s
[CV 5/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 5/5; 1/4] END max_depth=5, n_estimators=200;, score=-0.390 total time=
0.2s
[CV 3/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 3/5; 3/4] END max_depth=10, n_estimators=200;, score=-0.410 total time=
0.4s
[CV 1/5; 2/4] START max_depth=5, n_estimators=30
0...............................
[CV 1/5; 2/4] END max_depth=5, n_estimators=300;, score=-75.139 total time=
0.3s
[CV 5/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 5/5; 3/4] END max_depth=10, n_estimators=200;, score=-70.609 total time=
0.3s
[CV 1/5; 2/4] START max_depth=5, n_estimators=30
0...............................
[CV 1/5; 2/4] END max_depth=5, n_estimators=300;, score=-0.409 total time=
0.3s
[CV 4/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 4/5; 3/4] END max_depth=10, n_estimators=200;, score=-0.411 total time=
0.3s
[CV 5/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 5/5; 1/4] END max_depth=5, n_estimators=200;, score=-66.539 total time=
0.2s
[CV 4/5; 2/4] START max_depth=5, n_estimators=30
0...............................
[CV 4/5; 2/4] END max_depth=5, n_estimators=300;, score=-64.522 total time=
0.3s
[CV 2/5; 4/4] START max_depth=10, n_estimators=30
0..............................
[CV 2/5; 4/4] END max_depth=10, n_estimators=300;, score=-67.841 total time=
0.4s
[CV 2/5; 2/4] START max_depth=5, n_estimators=30
0...............................
[CV 2/5; 2/4] END max_depth=5, n_estimators=300;, score=-0.389 total time=
0.3s
[CV 5/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 5/5; 3/4] END max_depth=10, n_estimators=200;, score=-0.397 total time=
0.4s
[CV 1/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 1/5; 1/4] END max_depth=5, n_estimators=200;, score=-73.977 total time=
0.2s
[CV 5/5; 2/4] START max_depth=5, n_estimators=30
```

```
0..............................
[CV 5/5; 2/4] END max_depth=5, n_estimators=300;, score=-68.891 total time=
0.3s
[CV 3/5; 4/4] START max_depth=10, n_estimators=30
0.............................
[CV 3/5; 4/4] END max_depth=10, n_estimators=300;, score=-65.959 total time=
0.4s
[CV 3/5; 2/4] START max_depth=5, n_estimators=30
0..............................
[CV 3/5; 2/4] END max_depth=5, n_estimators=300;, score=-0.404 total time=
0.3s
[CV 1/5; 4/4] START max_depth=10, n_estimators=30
0.............................
[CV 1/5; 4/4] END max_depth=10, n_estimators=300;, score=-0.419 total time=
0.5s
[CV 2/5; 2/4] START max_depth=5, n_estimators=30
0..............................
[CV 2/5; 2/4] END max_depth=5, n_estimators=300;, score=-65.127 total time=
0.3s
[CV 4/5; 3/4] START max_depth=10, n_estimators=20
0.............................
[CV 4/5; 3/4] END max_depth=10, n_estimators=200;, score=-65.899 total time=
0.3s
[CV 2/5; 1/4] START max_depth=5, n_estimators=20
0..............................
[CV 2/5; 1/4] END max_depth=5, n_estimators=200;, score=-0.390 total time=
0.2s
[CV 4/5; 2/4] START max_depth=5, n_estimators=30
0..............................
[CV 4/5; 2/4] END max_depth=5, n_estimators=300;, score=-0.393 total time=
0.3s
[CV 2/5; 4/4] START max_depth=10, n_estimators=30
0.............................
[CV 2/5; 4/4] END max_depth=10, n_estimators=300;, score=-0.405 total time=
0.5s
[CV 3/5; 2/4] START max_depth=5, n_estimators=30
0..............................
[CV 3/5; 2/4] END max_depth=5, n_estimators=300;, score=-64.076 total time=
0.3s
[CV 1/5; 4/4] START max_depth=10, n_estimators=30
0.............................
[CV 1/5; 4/4] END max_depth=10, n_estimators=300;, score=-76.823 total time=
0.5s
[CV 4/5; 1/4] START max_depth=5, n_estimators=20
0..............................
[CV 4/5; 1/4] END max_depth=5, n_estimators=200;, score=-0.394 total time=
0.2s
[CV 5/5; 2/4] START max_depth=5, n_estimators=30
0..............................
[CV 5/5; 2/4] END max_depth=5, n_estimators=300;, score=-0.386 total time=
0.3s
[CV 3/5; 4/4] START max_depth=10, n_estimators=30
0.............................
[CV 3/5; 4/4] END max_depth=10, n_estimators=300;, score=-0.402 total time=
0.5s
[CV 2/5; 1/4] START max_depth=5, n_estimators=20
0..............................
[CV 2/5; 1/4] END max_depth=5, n_estimators=200;, score=-63.821 total time=
0.2s
[CV 2/5; 3/4] START max_depth=10, n_estimators=20
0.............................
```

```
[CV 2/5; 3/4] END max_depth=10, n_estimators=200;, score=-67.980 total time=
0.3s
[CV 1/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 1/5; 1/4] END max_depth=5, n_estimators=200;, score=-0.408 total time=
0.2s
[CV 1/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 1/5; 3/4] END max_depth=10, n_estimators=200;, score=-0.413 total time=
0.4s
[CV 4/5; 4/4] START max_depth=10, n_estimators=30
0..............................
[CV 4/5; 4/4] END max_depth=10, n_estimators=300;, score=-0.406 total time=
0.5s
[CV 4/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 4/5; 1/4] END max_depth=5, n_estimators=200;, score=-64.669 total time=
0.2s
[CV 1/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 1/5; 3/4] END max_depth=10, n_estimators=200;, score=-75.808 total time=
0.3s
[CV 4/5; 4/4] START max_depth=10, n_estimators=30
0..............................
[CV 4/5; 4/4] END max_depth=10, n_estimators=300;, score=-66.051 total time=
0.4s
[CV 3/5; 1/4] START max_depth=5, n_estimators=20
0...............................
[CV 3/5; 1/4] END max_depth=5, n_estimators=200;, score=-0.395 total time=
0.2s
[CV 2/5; 3/4] START max_depth=10, n_estimators=20
0..............................
[CV 2/5; 3/4] END max_depth=10, n_estimators=200;, score=-0.403 total time=
0.3s
[CV 5/5; 4/4] START max_depth=10, n_estimators=30
0..............................
[CV 5/5; 4/4] END max_depth=10, n_estimators=300;, score=-0.403 total time=
0.5s
```

There is a clear effect in the expected direction of number of acoomodates on prices.

In [104…

```
roomtype_pdp = partial_dependence(
    rf_pipeline, data_holdout[predictors_2], ["f_room_type"], kind="average"
)
```

In [104…

```
roomtype_pdp
```

Out[1047]:

```
{'grid_values': [array(['Entire home/apt', 'Private room'], dtype=objec
t)],
 'values': [array(['Entire home/apt', 'Private room'], dtype=object)],
 'average': array([[137.31847529, 104.48541309]])}
```

In [104…

```
roomtype_pdp['average'][0]
```

Out[1048]:

```
array([137.31847529, 104.48541309])
```

In [104…

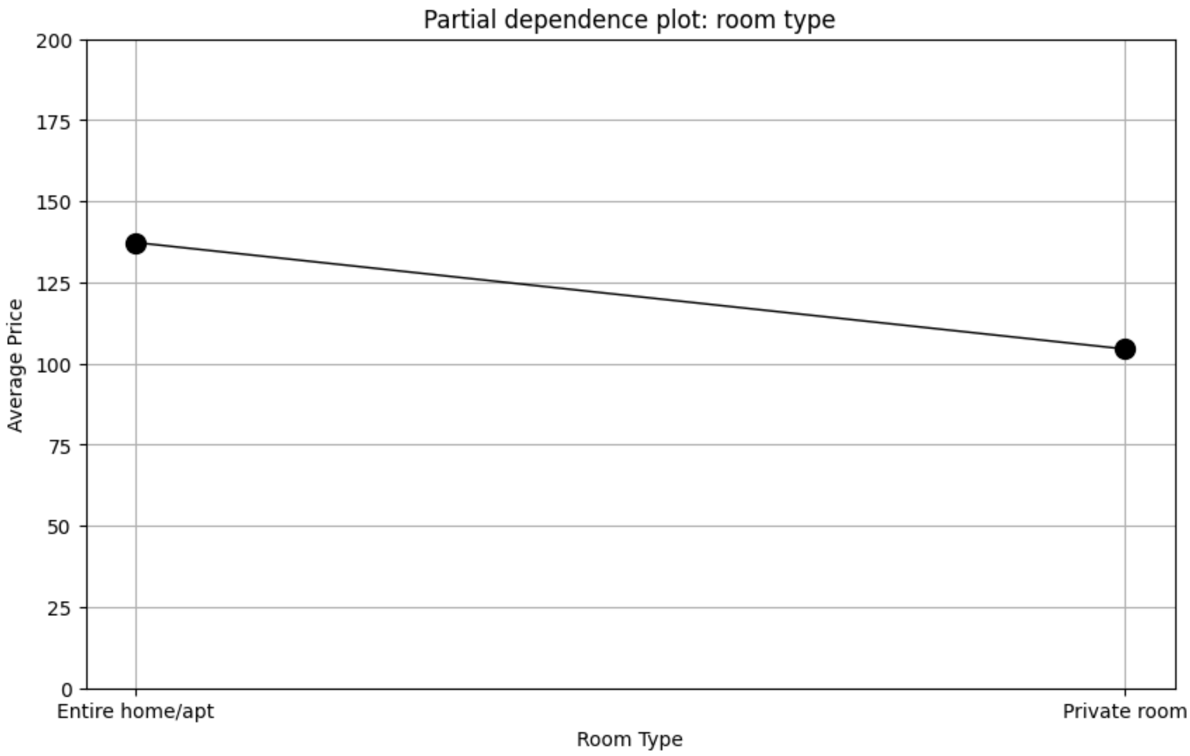
```
roomtype_pdp['values'][0]
```

Out[1049]:

```
array(['Entire home/apt', 'Private room'], dtype=object)
```

In [105…

```
# creating DataFrame from the dictionary
df = pd.DataFrame({'room type': roomtype_pdp['values'][0], 'average price':

# plotting
plt.figure(figsize=(10, 6))
plt.plot(df['room type'], df['average price'], color='k', marker='o', marker
plt.ylim(0, 200)
plt.title('Partial dependence plot: room type')
plt.xlabel('Room Type')
plt.ylabel('Average Price')
plt.grid(True)

# setting x-axis labels manually
plt.xticks(range(len(df['room type'])), df['room type'])

plt.show()
```



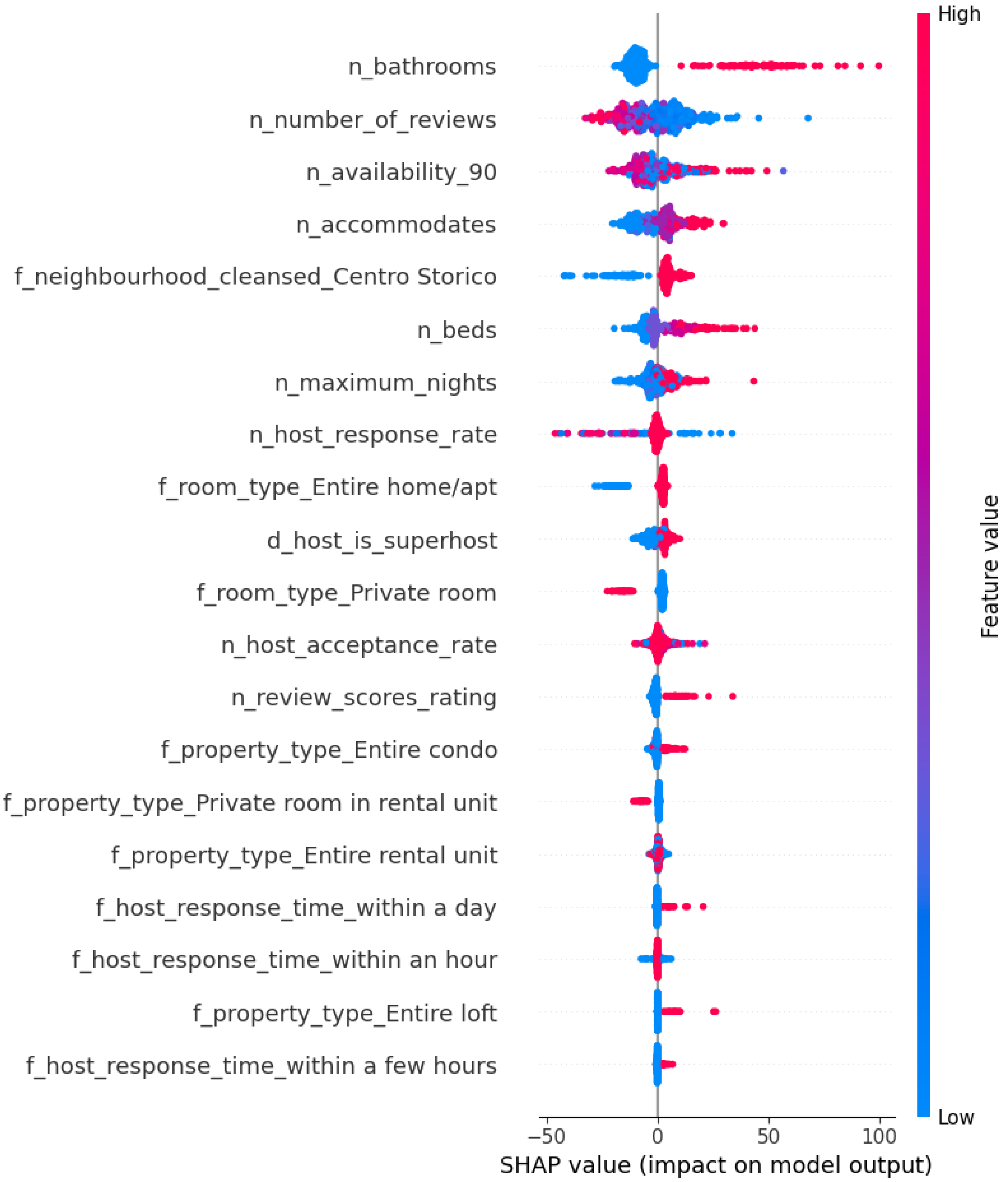
Room type which is categorical variable found to be fourth most important factor that affect the prices. Private rooms seem cheaper than an entire home, which is expected.

SHAP Importance

In [105…

```
import shap

columns_to_filter = [
    "n_accommodates",
    "n_beds",
    "f_property_type",
    "f_room_type",
    "n_bathrooms",
    "f_neighbourhood_cleansed",
    "n_availability_90",
    "n_maximum_nights",
    "f_host_response_time",
    "n_host_response_rate",
    "n_host_acceptance_rate",
    "n_number_of_reviews",
    "n_review_scores_rating",
    "d_host_is_superhost",
    "price",
    "ln_price"]

data_for_shap = data_final[columns_to_filter].copy()

# performing one-hot encoding for the 'category' column
data_encoded = pd.get_dummies(data_for_shap, columns=["f_property_type","f_r
                                                       "f_neighbourhood_clean


data_encoded = data_encoded.astype(int)

# splitting the data into features (X) and target (y)
X_shap = data_encoded.drop(['price', 'ln_price'], axis=1)
y_shap = data_encoded['price']

# splitting the data into training and test sets
X_train_shap, X_test_shap, y_train_shap, y_test_shap = train_test_split(X_sha

# training a Random Forest model
model_shap = RandomForestRegressor(random_state=42)
model_shap.fit(X_train_shap, y_train_shap)

# creating a SHAP explainer
explainer = shap.Explainer(model_shap, X_train_shap)

# computing SHAP values
shap_values = explainer(X_test_shap)

# plotting SHAP values
shap.summary_plot(shap_values, X_test_shap)
```

```
 97%|=================== | 447/460 [00:13<00:00]
```



Subsample performance: RMSE / mean(y)

The Test/Holdout RMSE is computed by applying the test data to the models and using the predicted values for evaluation. We will then contrast both the cross-validated RMSE values and the test RMSE values across the various alternative ML models.

In [105…
```
data_holdout_w_prediction = data_holdout.assign(
    predicted_price=rf_pipeline.predict(data_holdout[predictors_2])
)
```

In [105…
```
data_holdout_w_prediction2 = data_holdout.assign(
    predicted_price=rf_pipeline2.predict(data_holdout[predictors_2])
)
```

In [105…
```
# RMSE on holdout/test data

rf_test_rmse_level_price = np.sqrt((sum((data_holdout_w_prediction['predicte
                                        data_holdout_w_prediction['price']):
print("RMSE on Holdout/Test Data for Random Forest With Level Prices:", rf_t
```

```
RMSE on Holdout/Test Data for Random Forest With Level Prices: 65.4064935335
2404
```

In [105…
```
# RMSE of ln_price on holdout/test data

rf_test_rmse_log_price = np.sqrt((sum((data_holdout_w_prediction2['predicted_
                                      data_holdout_w_prediction2['ln_price']
print("RMSE on Holdout/Test Data for Random Forest With Log Prices:", rf_tes
```

```
RMSE on Holdout/Test Data for Random Forest With Log Prices: 0.3835984346164
461
```

Creating tables of heterogeneity by various grouping factors

In this part we can check how model performance changes for different categories of categorical variables. By creating a new grouping using a numeric variable, model performance can even be tested on these artificially produced groups.

- Apartment size: In our analysis for Florence, apartment sizes are between 2-6 people. If we assume apartments with 2-3 people are small apartments and 4-6 are large apartments, we can compare the model performance between these two grpups.

In [110…
```
data_holdout_w_prediction['is_low_size'] = data_holdout_w_prediction.n_accom
```

In [105…
```
data_holdout_w_prediction.iloc[0:5, -3:]
```

Out[1057]:

	ln_price	predicted_price	is_low_size
1168	4.532599	134.605340	small apt
4093	5.164786	144.229570	large apt
614	3.637586	88.161180	small apt
6728	5.802118	285.513714	large apt
825	4.934474	129.328680	large apt

In [105…
```
data_holdout_w_prediction.groupby('is_low_size').apply(lambda x: mean_square
```

Out[1058]:
```
is_low_size
large apt    70.932979
small apt    56.105188
dtype: float64
```

Putting it in a function with additional columns

In [105…
```
def calculate_rmse(groupby_obj):
    return (
        groupby_obj.apply(
            lambda x: mean_squared_error(x.predicted_price, x.price, squared=
        )
        .to_frame(name="rmse")
        .assign(mean_price=groupby_obj.apply(lambda x: np.mean(x.price)).valu
        .assign(rmse_norm=lambda x: x.rmse / x.mean_price).round(2)
    )
```

In [106…
```
# cheaper or more expensive flats
grouped_object = data_holdout_w_prediction.assign(
    is_low_size=lambda x: np.where(x.n_accommodates <= 3, "small apt", "large
).groupby("is_low_size")
accom_subset = calculate_rmse(grouped_object)
```

In [106…
```
accom_subset
```

Out[1061]:

is_low_size	rmse	mean_price	rmse_norm
large apt	70.93	161.87	0.44
small apt	56.11	107.86	0.52

Although the mean prices is lower for small size apartments, we found a higher RMSE/mean(y) for this category.

The analysis can be extended to compare room types, property types and neighbourhoods. In order not to make the analysis too long, we will only compare model performance according to apartment sizes.

Model2: Ordinary Least Squares Regression (OLS)

For OLS and LASSO, utilizing log-transformed price data is advisable because OLS is susceptible to non-normal data, unlike Random Forest. Subsequently, we can assess RMSE values for all three models when the target variable is ln(price). Each model (Random Forest, OLS, LASSO, and GBM) is executed for both level and log prices.

In [106…
```
from sklearn.linear_model import LinearRegression
```

In [106…
```
y, X = dmatrices("price ~ " + " + ".join(predictors_2), data_train)
y2, X22 = dmatrices("ln_price ~ " + " + ".join(predictors_2), data_train)

ols_model = LinearRegression().fit(X,y)
ols_model2 = LinearRegression().fit(X22,y2)

y_hat = ols_model.predict(X)
y_hat2 = ols_model2.predict(X22)
```

In [106…

```
# 1. Evaluating the Model Performance
ols_rmse = mean_squared_error(y,y_hat,squared=False)
print("RMSE:", ols_rmse)

ols_rmse_log = mean_squared_error(y2,y_hat2,squared=False)
print("RMSE of log model:", ols_rmse_log)

from sklearn.metrics import r2_score
r2 = r2_score(y, y_hat)
print("R-squared:", r2)

r2_ln = r2_score(y2, y_hat2)
print("R-squared of log model:", r2_ln)
```

```
RMSE: 68.44018669673957
RMSE of log model: 0.41804230789625135
R-squared: 0.3443857789583782
R-squared of log model: 0.3969934420761172
```

In [106…

```
# 2. Checking for Assumptions
# Using statsmodels' OLS to perform diagnostic tests
import statsmodels.api as sm

ols_model_stats = sm.OLS(y, X).fit()
print(ols_model_stats.summary())
```

```
                            OLS Regression Results
==============================================================================
==
Dep. Variable:                  price   R-squared:                       0.3
44
Model:                            OLS   Adj. R-squared:                  0.3
35
Method:                 Least Squares   F-statistic:                     36.
57
Date:                Sun, 11 Feb 2024   Prob (F-statistic):          2.29e-1
45
Time:                        22:24:59   Log-Likelihood:                -1037
0.
No. Observations:                1837   AIC:                         2.079e+
04
Df Residuals:                    1810   BIC:                         2.094e+
04
Df Model:                          26
Covariance Type:            nonrobust
==============================================================================
============================================
                                                              coef    std err
t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
--------------------------------------------
Intercept                                                -135.9542     35.844
-3.793      0.000    -206.254     -65.654
f_property_type[T.Entire home]                             -16.0566     11.571
-1.388      0.165     -38.750       6.637
f_property_type[T.Entire loft]                              12.9197      8.217
1.572      0.116      -3.196      29.035
f_property_type[T.Entire rental unit]                        1.6075      4.355
0.369      0.712      -6.933      10.148
f_property_type[T.Entire serviced apartment]                33.5049     17.349
1.931      0.054      -0.522      67.531
f_property_type[T.Entire vacation home]                    -17.1507     16.386
-1.047      0.295     -49.289      14.987
f_property_type[T.Private room in bed and breakfast]        31.6392     11.217
2.821      0.005       9.639      53.639
f_property_type[T.Private room in condo]                   -11.5736     11.358
-1.019      0.308     -33.849      10.702
f_property_type[T.Private room in home]                    -15.6316     13.396
-1.167      0.243     -41.904      10.641
f_property_type[T.Private room in rental unit]             -21.3474      7.254
-2.943      0.003     -35.575      -7.120
f_room_type[T.Private room]                                -16.9133      6.173
-2.740      0.006     -29.020      -4.806
f_neighbourhood_cleansed[T.Centro Storico]                  29.7940      6.397
4.657      0.000      17.248      42.340
f_neighbourhood_cleansed[T.Gavinana Galluzzo]                5.1402     11.623
0.442      0.658     -17.656      27.936
f_neighbourhood_cleansed[T.Isolotto Legnaia]               -16.7253     11.318
-1.478      0.140     -38.924       5.473
f_neighbourhood_cleansed[T.Rifredi]                         -8.0689      9.236
-0.874      0.382     -26.183      10.045
f_host_response_time[T.within a day]                        61.1014     32.870
1.859      0.063      -3.366     125.569
f_host_response_time[T.within a few hours]                  57.6179     34.116
1.689      0.091      -9.293     124.529
f_host_response_time[T.within an hour]                      53.5135     34.662
1.544      0.123     -14.468     121.495
```

```
n_accommodates                                                    2.6131      1.888
1.384      0.167      -1.090       6.316
n_beds                                                           13.0978      2.150
6.091      0.000       8.881      17.315
n_bathrooms                                                      60.0034      3.856
15.561      0.000      52.441      67.566
n_availability_90                                                 0.0072      0.050
0.144      0.885      -0.091       0.106
n_maximum_nights                                                  0.0115      0.003
3.437      0.001       0.005       0.018
n_host_response_rate                                             -0.3233      0.328
-0.985      0.325      -0.967       0.320
n_host_acceptance_rate                                            0.0924      0.127
0.725      0.468      -0.157       0.342
n_number_of_reviews                                              -0.0912      0.013
-7.076      0.000      -0.117      -0.066
n_review_scores_rating                                           23.3075      5.946
3.920      0.000      11.646      34.969
d_host_is_superhost                                               5.7801      3.769
1.534      0.125      -1.612      13.172
============================================================================
==
Omnibus:                      714.003   Durbin-Watson:                   1.9
95
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             3788.7
93
Skew:                           1.760   Prob(JB):                         0.
00
Kurtosis:                       9.092   Cond. No.                     1.41e+
16
============================================================================
==

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is corre
ctly specified.
[2] The smallest eigenvalue is 4.22e-24. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

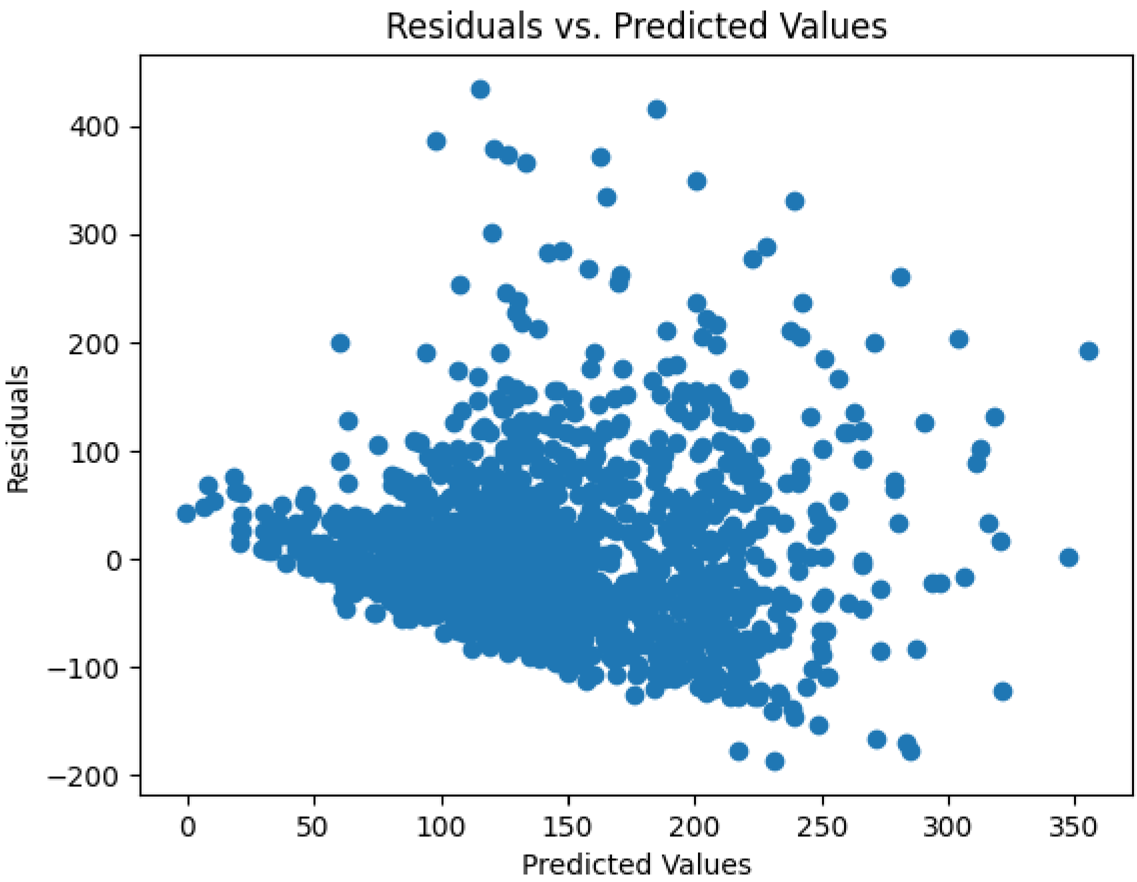
In [106…

```
# 3. Cross-validation
# Performing 5-fold cross-validation
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(ols_model, X, y, cv=5, scoring='neg_mean_squared_
cv_rmse_scores = np.sqrt(-cv_scores)

cv_scores_log = cross_val_score(ols_model, X22, y2, cv=5, scoring='neg_mean_s
cv_rmse_scores_log = np.sqrt(-cv_scores_log)

print("Cross-Validation RMSE Scores:", cv_rmse_scores)
print("Average Cross-Validation RMSE Score (level prices):", np.average(cv_rm

print("Cross-Validation RMSE Scores For Log-Transformed Prices:", cv_rmse_sco
print("Average Cross-Validation RMSE Score (log prices):", np.average(cv_rms
```

```
Cross-Validation RMSE Scores: [75.53853473 67.32622172 65.40804823 66.038313
93 72.7321498 ]
Average Cross-Validation RMSE Score (level prices): 69.4086536844479
Cross-Validation RMSE Scores For Log-Transformed Prices: [0.42195518 0.42736
825 0.41495135 0.41886923 0.44665436]
Average Cross-Validation RMSE Score (log prices): 0.4259596735247565
```

In [106…

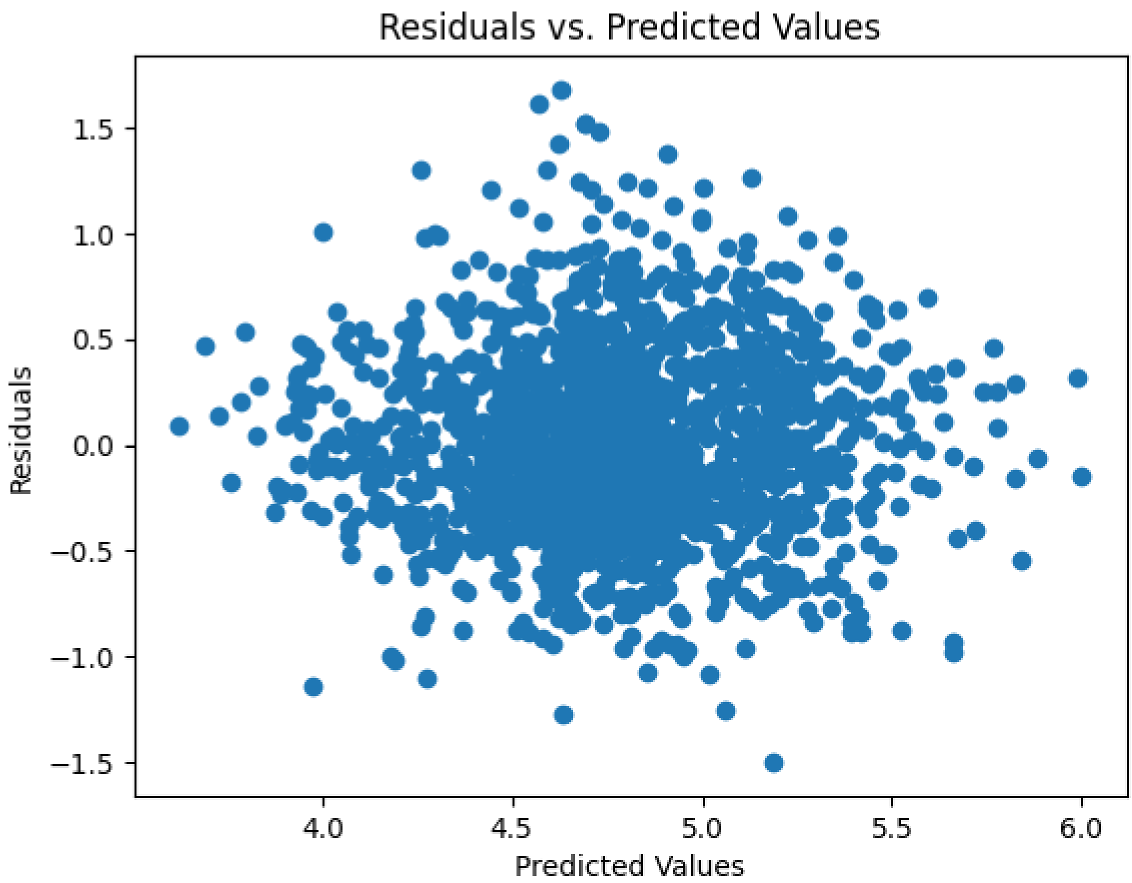
```
# 4. Visualization
# Plotting residuals
residuals = y - y_hat
plt.scatter(y_hat, residuals)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Predicted Values")
plt.show()
```



- Due to the presence of heteroskedastic residuals, it makes sente to utilize log-transformed prices for the analysis.

In [106…

```
# plotting residuals of log transformed prices
residuals2 = y2 - y_hat2
plt.scatter(y_hat2, residuals2)
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs. Predicted Values")
plt.show()
```



- While not entirely eliminating the issue, heteroskedasticity appears to be mitigated to some extent when utilizing log-transformed price data.

In [106…

```
# 5. Calculating RMSE on test/holdout data

y_test, X_test = dmatrices("price ~ " + " + ".join(predictors_2), data_holdo
y_test2, X_test2 = dmatrices("ln_price ~ " + " + ".join(predictors_2), data_

# Predicting the target variable on the test dataset
y_hat_test = ols_model.predict(X_test)
y_hat_test2 = ols_model2.predict(X_test2)


# Computing the RMSE
ols_rmse_test = mean_squared_error(y_test, y_hat_test, squared=False)
ols_rmse_test2 = mean_squared_error(y_test2, y_hat_test2, squared=False)

print("Test RMSE for OLS model:", ols_rmse_test)
print("Test RMSE for log transformed OLS model:", ols_rmse_test2)
```

```
Test RMSE for OLS model: 67.64255207040684
Test RMSE for log transformed OLS model: 0.40662602308793666
```

Model3: LASSO Regression

In [107…

```
from sklearn.linear_model import ElasticNet
```

In [107…

```
lasso_model =  ElasticNet(l1_ratio = 1, fit_intercept = True)
lasso_model_ln =  ElasticNet(l1_ratio = 1, fit_intercept = True)
```

In [107…

```
lasso_model_cv = GridSearchCV(
    lasso_model,
    {"alpha":[i/100 for i in range(5, 26, 5)]},
    cv=5,
    scoring="neg_root_mean_squared_error",
    verbose=3,
)

lasso_model_cv2 = GridSearchCV(
    lasso_model_ln,
    {"alpha":[i/100 for i in range(5, 26, 5)]},
    cv=5,
    scoring="neg_root_mean_squared_error",
    verbose=3,
)
```

In [107…

```
y, X = dmatrices("price ~ " + " + ".join(predictors_E), data_train)
y2, X22 = dmatrices("ln_price ~ " + " + ".join(predictors_E), data_train)
```

In [107…

```
%%time
lasso_model_cv.fit(X, y.ravel())
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 1/5] END ......................alpha=0.05;, score=-75.231 total time=
0.1s
[CV 2/5] END ......................alpha=0.05;, score=-67.875 total time=
0.2s
[CV 3/5] END ......................alpha=0.05;, score=-65.084 total time=
0.3s
[CV 4/5] END ......................alpha=0.05;, score=-66.019 total time=
0.1s
[CV 5/5] END ......................alpha=0.05;, score=-72.523 total time=
0.3s
[CV 1/5] END .......................alpha=0.1;, score=-75.264 total time=
0.1s
[CV 2/5] END .......................alpha=0.1;, score=-67.577 total time=
0.1s
[CV 3/5] END .......................alpha=0.1;, score=-65.212 total time=
0.3s
[CV 4/5] END .......................alpha=0.1;, score=-65.843 total time=
0.2s
[CV 5/5] END .......................alpha=0.1;, score=-72.576 total time=
0.3s
[CV 1/5] END ......................alpha=0.15;, score=-75.356 total time=
0.1s
[CV 2/5] END ......................alpha=0.15;, score=-67.353 total time=
0.3s
[CV 3/5] END ......................alpha=0.15;, score=-65.372 total time=
0.1s
[CV 4/5] END ......................alpha=0.15;, score=-65.738 total time=
0.1s
[CV 5/5] END ......................alpha=0.15;, score=-72.651 total time=
0.3s
[CV 1/5] END .......................alpha=0.2;, score=-75.376 total time=
0.2s
[CV 2/5] END .......................alpha=0.2;, score=-67.180 total time=
0.1s
[CV 3/5] END .......................alpha=0.2;, score=-65.498 total time=
0.2s
[CV 4/5] END .......................alpha=0.2;, score=-65.647 total time=
0.1s
[CV 5/5] END .......................alpha=0.2;, score=-72.661 total time=
0.3s
[CV 1/5] END ......................alpha=0.25;, score=-75.353 total time=
0.1s
[CV 2/5] END ......................alpha=0.25;, score=-67.055 total time=
0.1s
[CV 3/5] END ......................alpha=0.25;, score=-65.577 total time=
0.2s
[CV 4/5] END ......................alpha=0.25;, score=-65.582 total time=
0.1s
[CV 5/5] END ......................alpha=0.25;, score=-72.663 total time=
0.2s
CPU times: user 18.1 s, sys: 15.3 s, total: 33.5 s
Wall time: 4.7 s
```

Out[1074]:

▸ **GridSearchCV**

▸ **estimator: ElasticNet**

▸ ElasticNet

In [107…

```
%%time
lasso_model_cv2.fit(X22, y2.ravel())
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 1/5] END .......................alpha=0.05;, score=-0.460 total time=
0.1s
[CV 2/5] END .......................alpha=0.05;, score=-0.448 total time=
0.1s
[CV 3/5] END .......................alpha=0.05;, score=-0.465 total time=
0.1s
[CV 4/5] END .......................alpha=0.05;, score=-0.445 total time=
0.1s
[CV 5/5] END .......................alpha=0.05;, score=-0.480 total time=
0.1s
[CV 1/5] END ........................alpha=0.1;, score=-0.472 total time=
0.1s
[CV 2/5] END ........................alpha=0.1;, score=-0.464 total time=
0.0s
[CV 3/5] END ........................alpha=0.1;, score=-0.481 total time=
0.1s
[CV 4/5] END ........................alpha=0.1;, score=-0.462 total time=
0.1s
[CV 5/5] END ........................alpha=0.1;, score=-0.490 total time=
0.1s
[CV 1/5] END .......................alpha=0.15;, score=-0.481 total time=
0.1s
[CV 2/5] END .......................alpha=0.15;, score=-0.471 total time=
0.1s
[CV 3/5] END .......................alpha=0.15;, score=-0.488 total time=
0.1s
[CV 4/5] END .......................alpha=0.15;, score=-0.468 total time=
0.1s
[CV 5/5] END .......................alpha=0.15;, score=-0.495 total time=
0.1s
[CV 1/5] END ........................alpha=0.2;, score=-0.489 total time=
0.1s
[CV 2/5] END ........................alpha=0.2;, score=-0.479 total time=
0.1s
[CV 3/5] END ........................alpha=0.2;, score=-0.495 total time=
0.1s
[CV 4/5] END ........................alpha=0.2;, score=-0.474 total time=
0.1s
[CV 5/5] END ........................alpha=0.2;, score=-0.499 total time=
0.1s
[CV 1/5] END .......................alpha=0.25;, score=-0.497 total time=
0.1s
[CV 2/5] END .......................alpha=0.25;, score=-0.486 total time=
0.1s
[CV 3/5] END .......................alpha=0.25;, score=-0.500 total time=
0.1s
[CV 4/5] END .......................alpha=0.25;, score=-0.482 total time=
0.0s
[CV 5/5] END .......................alpha=0.25;, score=-0.505 total time=
0.1s
CPU times: user 9.49 s, sys: 7.63 s, total: 17.1 s
Wall time: 2.39 s
```

Out[1075]:

```
▸     GridSearchCV
▸ estimator: ElasticNet
    ▸ ElasticNet
```

In [107…

```
pd.DataFrame(
    lasso_model_cv.best_estimator_.coef_.tolist(),
    index=X.design_info.column_names,
    columns=["lasso_coefficient"],
).assign(lasso_coefficient=lambda x: x.lasso_coefficient.round(3)).loc[
    lambda x: x.lasso_coefficient != 0
]
```

Out[1076]:

	lasso_coefficient
f_room_type[T.Private room]	-25.945
f_host_response_time[T.within an hour]	-3.505
f_property_type[T.Entire rental unit]:f_neighbourhood_cleansed[T.Centro Storico]	3.917
f_property_type[T.Entire home]:f_neighbourhood_cleansed[T.Gavinana Galluzzo]	7.492
n_accommodates	-2.874
n_accommodates:f_property_type[T.Entire home]	-2.980
n_accommodates:f_property_type[T.Entire loft]	3.472
n_accommodates:f_property_type[T.Entire rental unit]	-0.526
n_accommodates:f_property_type[T.Entire serviced apartment]	7.758
n_accommodates:f_property_type[T.Entire vacation home]	-1.986
n_accommodates:f_property_type[T.Private room in bed and breakfast]	14.817
n_accommodates:f_property_type[T.Private room in home]	-0.424
n_accommodates:f_property_type[T.Private room in rental unit]	-5.519
n_accommodates:f_neighbourhood_cleansed[T.Centro Storico]	8.143
n_accommodates:f_neighbourhood_cleansed[T.Isolotto Legnaia]	-2.977
n_accommodates:f_neighbourhood_cleansed[T.Rifredi]	-1.888
n_beds	12.264
n_bathrooms	57.680
n_availability_90	0.010
n_maximum_nights	0.012
n_host_response_rate	0.112
n_host_acceptance_rate	0.116
n_number_of_reviews	-0.088
n_review_scores_rating	23.991

In [107…

```
lasso_model_cv.best_estimator_
```

Out[1077]:
```
▾ ElasticNet

ElasticNet(alpha=0.25, l1_ratio=1)
```

In [107…
```
lasso_rmse = pd.DataFrame(lasso_model_cv.cv_results_).loc[
    lambda x: x.param_alpha == lasso_model_cv.best_estimator_.alpha
].mean_test_score.values[0] * -1
print("LASSO Cross-Validated RMSE for Level Prices:", lasso_rmse)
```

```
LASSO Cross-Validated RMSE for Level Prices: 69.24594343466117
```

In [107…
```
lasso_rmse_ln = pd.DataFrame(lasso_model_cv2.cv_results_).loc[
    lambda x: x.param_alpha == lasso_model_cv2.best_estimator_.alpha
].mean_test_score.values[0] * -1
print("LASSO Cross-Validated RMSE for Log Prices:", lasso_rmse_ln)
```

```
LASSO Cross-Validated RMSE for Log Prices: 0.45938818146744903
```

In [108…
```
y_test, X_test = dmatrices("price ~ " + " + ".join(predictors_E), data_holdo
y_test2, X_test2 = dmatrices("ln_price ~ " + " + ".join(predictors_E), data_

# predicting the target variable on the test dataset
y_hat_test_lasso_level = lasso_model_cv.predict(X_test)
y_hat_test_lasso_log = lasso_model_cv2.predict(X_test2)

# computing the RMSE
lasso_rmse_test_level = mean_squared_error(y_test, y_hat_test_lasso_level, s
print("Test RMSE for LASSO Model With Level Prices:", lasso_rmse_test_level)

lasso_rmse_test_log = mean_squared_error(y_test2, y_hat_test_lasso_log, squa
print("Test RMSE for LASSO Model With Log Prices:", lasso_rmse_test_log)
```

```
Test RMSE for LASSO Model With Level Prices: 68.30298757754646
Test RMSE for LASSO Model With Log Prices: 0.45263723660073985
```

Model4: Gradient Boosting Machines (GBM)

In [108…
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [108…
```
gbm = GradientBoostingRegressor(learning_rate=0.1, min_samples_split=20, max_
                                )

tune_grid = {"n_estimators": [200, 300], "max_depth": [5, 10]}

gbm_model_cv = GridSearchCV(
    gbm,
    tune_grid,
    cv=5,
    scoring="neg_root_mean_squared_error",
    verbose=10,
    n_jobs=-1
)
```

In [108…
```
categorical_columns = [col for col in predictors_2 if col.startswith("f_")]
numerical_columns = [col for col in predictors_2 if col not in categorical_c

categorical_encoder = OneHotEncoder(handle_unknown="ignore")

preprocessing = ColumnTransformer(
    [
        ("cat", categorical_encoder, categorical_columns),
        ("num", "passthrough", numerical_columns),
    ]
)

gbm_pipe = Pipeline(
    [("preprocess", preprocessing), ("regressor", gbm_model_cv)], verbose=Tr
)
```

In [108…
```
%%time
gbm_pipe.fit(data_train[predictors_2],data_train.price)
```

```
[Pipeline] ........ (step 1 of 2) Processing preprocess, total=   0.0s
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[Pipeline] ......... (step 2 of 2) Processing regressor, total=   2.2s
CPU times: user 244 ms, sys: 91.8 ms, total: 336 ms
Wall time: 2.23 s
```

Out[1084]:

```
▸ Pipeline
  ▸ preprocess: ColumnTransformer
    ▸ cat                  ▸ num
    ▸ OneHotEncoder        ▸ passthrough
  ▸ regressor: GridSearchCV
  ▸ estimator: GradientBoostingRegressor
    ▸ GradientBoostingRegressor
```

In [108…
```
gbm_model_cv.best_estimator_
```

Out[1085]:

```
▾ GradientBoostingRegressor
GradientBoostingRegressor(max_depth=5, max_features=10, min_sample
s_split=20,
                          n_estimators=200)
```

In [108…
```
gbm_rmse = gbm_model_cv.best_score_*-1
```

In [108…
```
gbm_rmse
```

Out[1087]: 66.09698199724521

In [108…
```
# predicting the target variable on the test dataset using the trained pipel
y_hat_test_gbm = gbm_pipe.predict(data_holdout[predictors_2])
```

In [108…
```
# computing the test RMSE
gbm_rmse_test = mean_squared_error(data_holdout.price, y_hat_test_gbm, squar
print("Test RMSE for GBM Model With Level Prices:", gbm_rmse_test)
```

```
Test RMSE for GBM Model With Level Prices: 66.30138608329376
```

In [109…
```
%%time
gbm_pipe.fit(data_train[predictors_2],data_train.ln_price)
```

```
[Pipeline] ........ (step 1 of 2) Processing preprocess, total=   0.0s
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[Pipeline] ......... (step 2 of 2) Processing regressor, total=   1.3s
CPU times: user 228 ms, sys: 13.7 ms, total: 241 ms
Wall time: 1.31 s
```

Out[1090]:
```
▸ Pipeline
▸ preprocess: ColumnTransformer
▸ cat                ▸ num
▸ OneHotEncoder      ▸ passthrough
▸ regressor: GridSearchCV
▸ estimator: GradientBoostingRegressor
▸ GradientBoostingRegressor
```

In [109…
```
gbm_model_cv.best_estimator_
```

Out[1091]:
```
▾ GradientBoostingRegressor
GradientBoostingRegressor(max_depth=5, max_features=10, min_sample
s_split=20,
                          n_estimators=200)
```

In [109…
```
gbm_rmse_ln = gbm_model_cv.best_score_*-1
```

In [109…
```
gbm_rmse_ln
```

Out[1093]: 0.39562161619070085

In [109…
```
# predicting the target variable (log prices) on the test dataset using the
y_hat_test_gbm_log = gbm_pipe.predict(data_holdout[predictors_2])
```

In [109…
```
# computing the test RMSE for log prices
gbm_rmse_test_log = mean_squared_error(data_holdout.ln_price, y_hat_test_gbm
print("Test RMSE for GBM Model With Log Prices:", gbm_rmse_test_log)
```

```
Test RMSE for GBM Model With Log Prices: 0.3783503327041733
```

Comparison of models

In [109…
```
# comparing Cross-validated RMSEs for level prices as target variable
df_cv_level_prices = pd.DataFrame({'Model': ['OLS', 'LASSO', 'Random Forest'
              'CV RMSE (Level Prices)': [np.average(cv_rmse_scores), lasso_r
                                                        -rf_model_level.best_s
```

In [109…
```
# comparing Cross-validated RMSEs for log transformed prices as target varial
df_cv_log_prices = pd.DataFrame({'Model': ['OLS', 'LASSO', 'Random Forest',
              'CV RMSE (Log Prices)': [np.average(cv_rmse_scores_log), lasso_
                                                               -rf_model_lo
```

In [109…
```
# comparing RMSEs on test data for level prices as target variable

df_test_level_prices = pd.DataFrame({'Model': ['OLS', 'LASSO', 'Random Fores
              'Test RMSE (Level Prices)': [ols_rmse_test, lasso_rmse_test_le
```

In [109…
```
# comparing RMSEs on test data for log transformed prices as target variable

df_test_log_prices = pd.DataFrame({'Model': ['OLS', 'LASSO', 'Random Forest'
              'Test RMSE (Log Prices)': [ols_rmse_test2, lasso_rmse_test_log
                                                       rf_test_rmse_log_price,
```

In [110…
```
# merging dataframes
df_combined = pd.merge(df_cv_level_prices, df_test_level_prices, on='Model')
df_combined = pd.merge(df_combined, df_cv_log_prices, on='Model')
df_combined = pd.merge(df_combined, df_test_log_prices, on='Model')
```

In [110…
```
df_combined
```

Out[1101]:

	Model	CV RMSE (Level Prices)	Test RMSE (Level Prices)	CV RMSE (Log Prices)	Test RMSE (Log Prices)
0	OLS	69.408654	67.642552	0.425960	0.406626
1	LASSO	69.245943	68.302988	0.459388	0.452637
2	Random Forest	66.816975	65.406494	0.400065	0.383598
3	GBM	66.096982	66.301386	0.395622	0.378350

- The best performing model in terms of cross-validated RMSEs is GBM and then Random Forest comes. OLS and LASSO have very similar RMSEs.
- The next step is to compare RMSEs on holdout/test data
- Now Random Forest is the best performing model with holdout/test data.

Conclusion

We commenced our analysis using Airbnb data for Florence, which encompasses numerous qualitative variables along with a handful of quantitative ones. Employing exploratory data analysis (EDA) and feature engineering techniques, we leveraged our domain expertise to craft estimation models. Initially, we opted for a Random Forest model and meticulously scrutinized to identify the key factors influencing prices. Subsequently, we gauged the overall performance of the base Random Forest model against three alternative models: OLS, LASSO, and GBM.

In terms of overall performance, GBM emerged as the frontrunner, yielding the lowest RMSE across cross-validated samples and the test dataset. Interestingly, while room type proved to be the most influential factor in London's data, the number of accommodates took precedence in Florence. Similarly, the second crucial factor in London was the number of bathrooms, whereas in Florence, it was the room type. Moreover, the third significant factor in London was the number of accommodates, while in Florence, it was the neighborhood. Notably, certain neighborhoods in both London and Florence exhibited tendencies to either elevate or diminish accommodation prices.