In [120…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import geopandas as gpd
import matplotlib as mpl
import matplotlib.colors as colors
import matplotlib.cm as cm

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegresso
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold, Ra
from sklearn.metrics import auc, accuracy_score, confusion_matrix, mean_squa
```

Problem 1

Build a complete pipeline with a data set of your choice and a tree-based model of your choice in R (using tidymodels) or Python (using scikit-learn). For each step, include a paragraph explaining why you did that step the way you did (what components were included and, possibly, what you decided not to do).

- a brief description of where the data came from

I chose data on the price of Airbnb listings in European cities, originally from this publication, and accessed on Kaggle. The data contains prices for various listings and descriptions of the listings (e.g. type of room, guest satisfaction rating, distance from city center). My goal is to predict the price of a listing.

- some initial investigation of the data (which textual or graphical summaries did you investigate? Did you find anything unusual?)

In [2]:

```
## combine city and weekday/weekend data
berlin_wdays = pd.read_csv("airbnb/berlin_weekdays.csv", index_col=[0])
berlin_wdays['city'] = 'Berlin'
berlin_wdays['weekend'] = False
berlin_wends = pd.read_csv("airbnb/berlin_weekends.csv", index_col=[0])
berlin_wends['city'] = 'Berlin'
berlin_wends['weekend'] = True
budapest_wdays = pd.read_csv("airbnb/budapest_weekdays.csv", index_col=[0])
budapest_wdays['city'] = 'Budapest'
budapest_wdays['weekend'] = False
budapest_wends = pd.read_csv("airbnb/budapest_weekends.csv", index_col=[0])
budapest_wends['city'] = 'Budapest'
budapest_wends['weekend'] = True
lisbon_wdays = pd.read_csv("airbnb/lisbon_weekdays.csv", index_col=[0])
lisbon_wdays['city'] = 'Lisbon'
lisbon_wdays['weekend'] = False
lisbon_wends = pd.read_csv("airbnb/lisbon_weekends.csv", index_col=[0])
lisbon_wends['city'] = 'Lisbon'
lisbon_wends['weekend'] = True
```

```
london_wdays = pd.read_csv("airbnb/london_weekdays.csv", index_col=[0])
london_wdays['city'] = 'London'
london_wdays['weekend'] = False
london_wends = pd.read_csv("airbnb/london_weekends.csv", index_col=[0])
london_wends['city'] = 'London'
london_wends['weekend'] = True
paris_wdays = pd.read_csv("airbnb/paris_weekdays.csv", index_col=[0])
paris_wdays['city'] = 'Paris'
paris_wdays['weekend'] = False
paris_wends = pd.read_csv("airbnb/paris_weekends.csv", index_col=[0])
paris_wends['city'] = 'Paris'
paris_wends['weekend'] = True
rome_wdays = pd.read_csv("airbnb/rome_weekdays.csv", index_col=[0])
rome_wdays['city'] = 'Rome'
rome_wdays['weekend'] = False
rome_wends = pd.read_csv("airbnb/rome_weekends.csv", index_col=[0])
rome_wends['city'] = 'Rome'
rome_wends['weekend'] = True
vienna_wdays = pd.read_csv("airbnb/vienna_weekdays.csv", index_col=[0])
vienna_wdays['city'] = 'Vienna'
vienna_wdays['weekend'] = False
vienna_wends = pd.read_csv("airbnb/vienna_weekends.csv", index_col=[0])
vienna_wends['city'] = 'Vienna'
vienna_wends['weekend'] = True

master_df = pd.concat([berlin_wdays, berlin_wends, budapest_wdays, budapest_
                       lisbon_wdays, lisbon_wends, london_wdays, london_wenc
                      paris_wdays, paris_wends, rome_wdays, rome_wends, vier
master_df.head()
```

Out[2]:

	realSum	room_type	room_shared	room_private	person_capacity	host_is_superhost
0	185.799757	Private room	False	True	2.0	True
1	194.914462	Private room	False	True	5.0	False
2	176.217631	Private room	False	True	2.0	False
3	207.768533	Private room	False	True	3.0	True
4	150.743199	Private room	False	True	2.0	False

5 rows × 21 columns

In [49]:

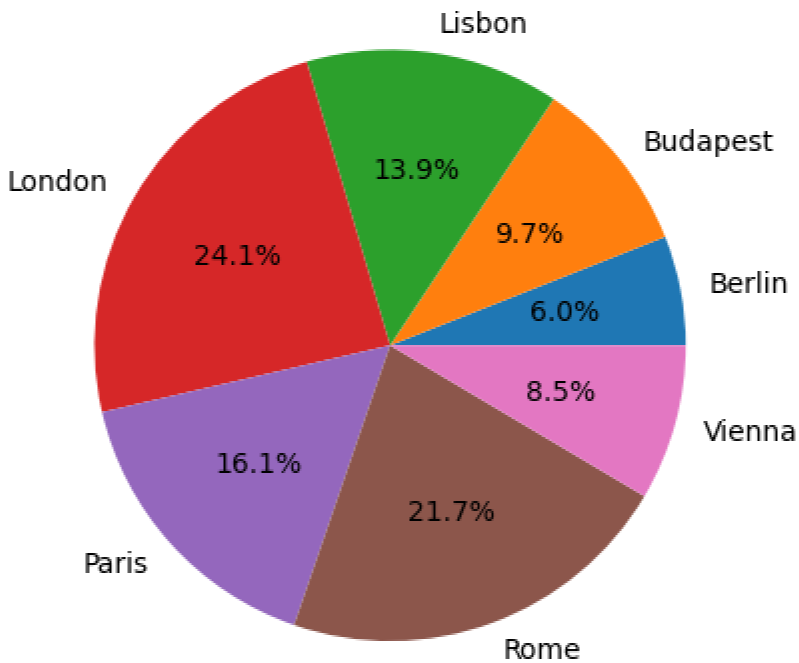
```
for col in ["room_type", "city"]:
    master_df[col] = master_df[col].astype("category")
for col in ["multi", "biz"]:
    master_df[col] = master_df[col].astype(bool)
master_df.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41514 entries, 0 to 1798
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype   
---  ------                      --------------  -----   
 0   realSum                     41514 non-null  float64 
 1   room_type                   41514 non-null  category
 2   room_shared                 41514 non-null  bool    
 3   room_private                41514 non-null  bool    
 4   person_capacity             41514 non-null  float64 
 5   host_is_superhost           41514 non-null  bool    
 6   multi                       41514 non-null  bool    
 7   biz                         41514 non-null  bool    
 8   cleanliness_rating          41514 non-null  float64 
 9   guest_satisfaction_overall  41514 non-null  float64 
 10  bedrooms                    41514 non-null  int64   
 11  dist                        41514 non-null  float64 
 12  metro_dist                  41514 non-null  float64 
 13  attr_index                  41514 non-null  float64 
 14  attr_index_norm             41514 non-null  float64 
 15  rest_index                  41514 non-null  float64 
 16  rest_index_norm             41514 non-null  float64 
 17  lng                         41514 non-null  float64 
 18  lat                         41514 non-null  float64 
 19  city                        41514 non-null  category
 20  weekend                     41514 non-null  bool    
dtypes: bool(6), category(2), float64(12), int64(1)
memory usage: 4.8 MB
```

There appears to be no missing data. Now I want to check if the data are well balanced for features (e.g. city, room type).

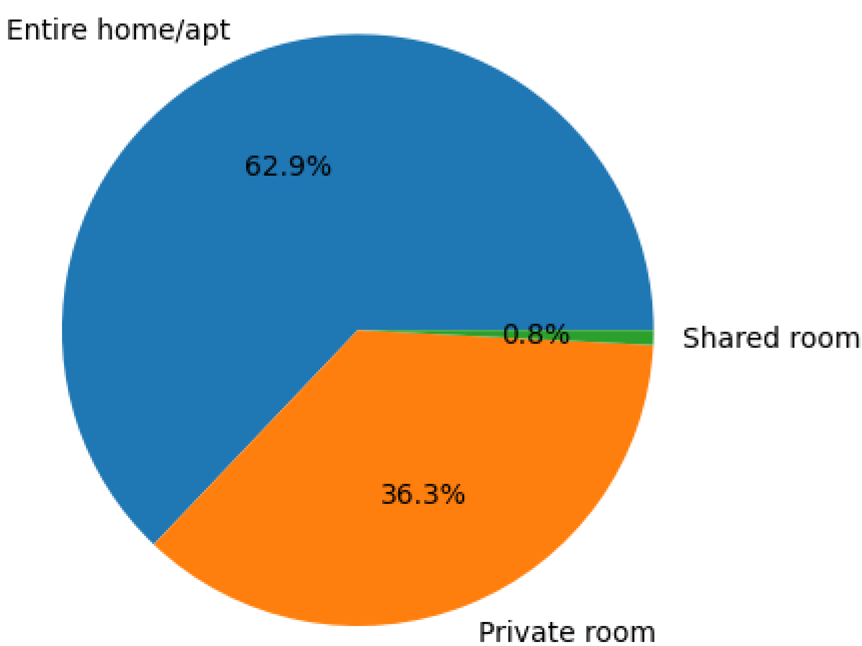
In [4]:
```
## cities are not equally represented in the data

cities, counts = np.unique(master_df["city"], return_counts=True)
plt.pie(counts, labels=cities, autopct='%1.1f%%')
plt.show()
```



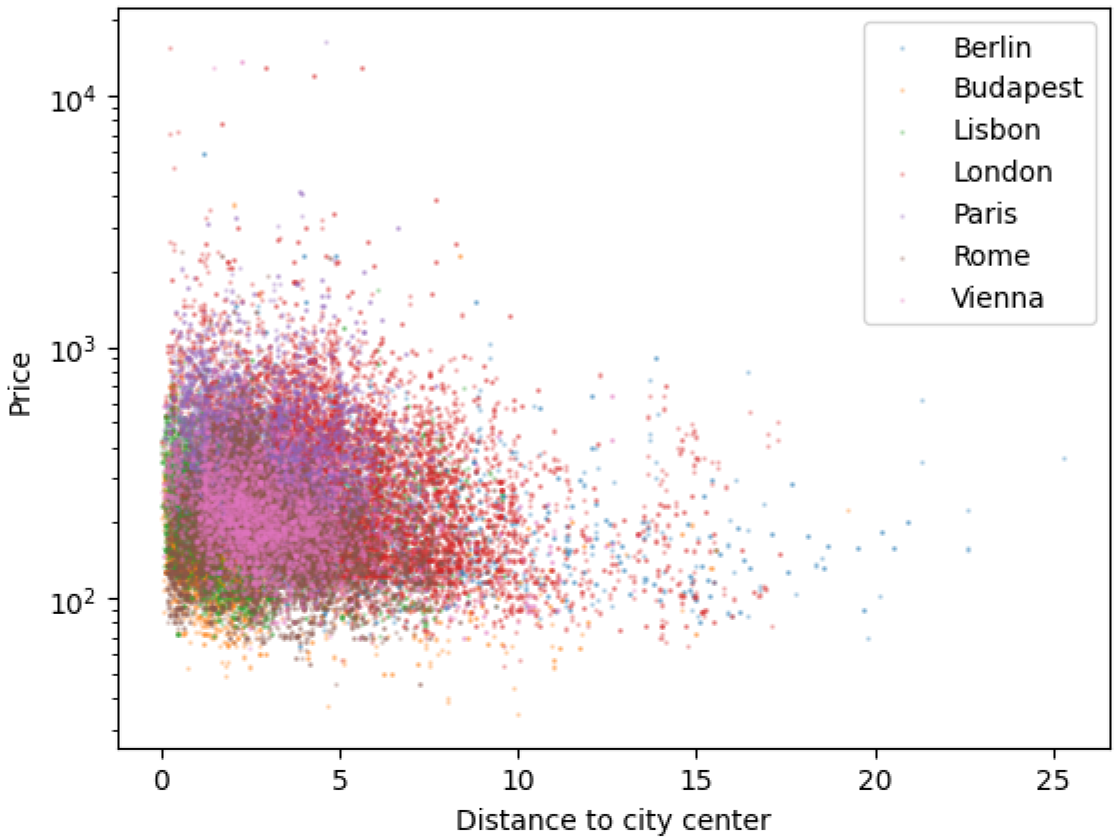
In [5]:
```
## room types are heavily skewed

rooms, r_counts = np.unique(master_df["room_type"], return_counts=True)
plt.pie(r_counts, labels=rooms, autopct='%1.1f%%')
plt.show()
```

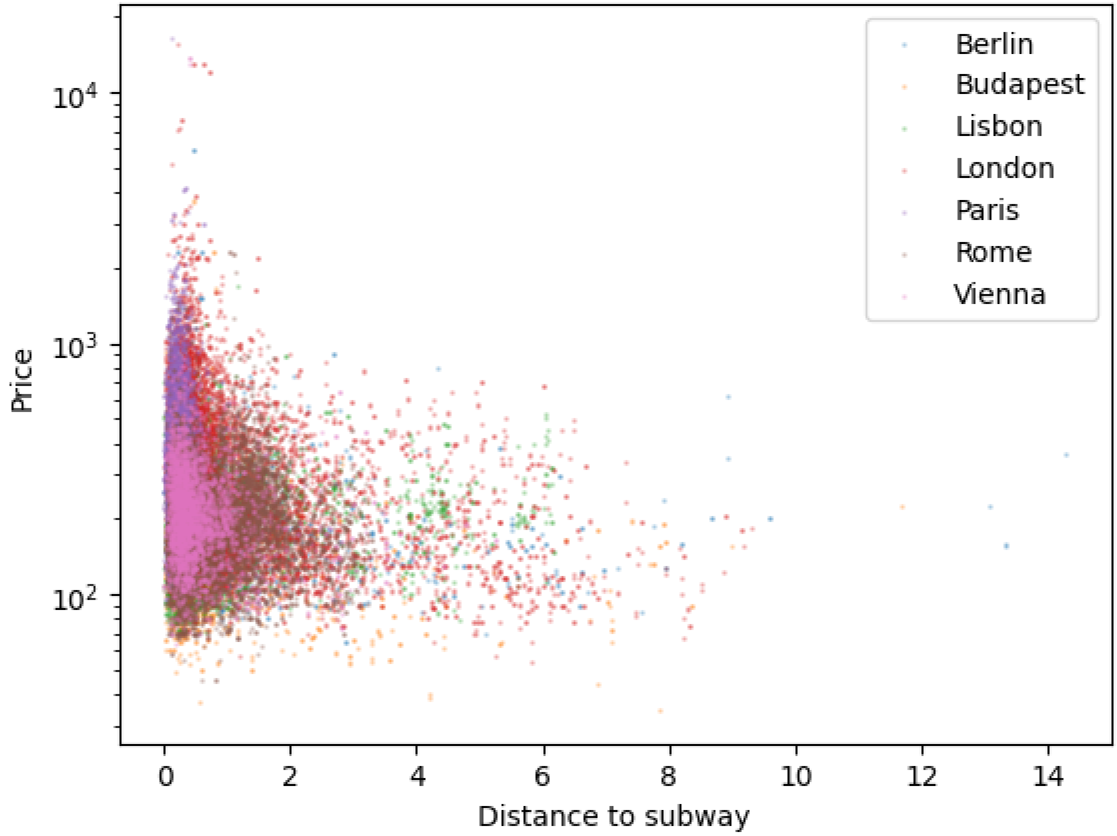


It seems reasonable to think that prices might correlate with the distance to the city center `dist` and distance to the nearest subway station `metro_dist`. I check this below for each of the cities:

In [6]:
```
for c in cities:
    df = master_df[master_df["city"]==c]
    plt.scatter(df["dist"], df["realSum"], label=c, alpha=0.2, s=1)
plt.legend(loc='best')
plt.yscale('log')
plt.xlabel("Distance to city center")
plt.ylabel("Price")
plt.show()
```



In [7]:
```
for c in cities:
    df = master_df[master_df["city"]==c]
    plt.scatter(df["metro_dist"], df["realSum"], label=c, alpha=0.2, s=1)
plt.legend(loc='best')
plt.yscale('log')
plt.xlabel("Distance to subway")
plt.ylabel("Price")
plt.show()
```



I would have liked to convert latitude and longitude data into categorical neighbourhoods/zipcodes, but could not readily find the data to do so. I wanted to see if location played a role in pricing for the city with the largest number of data points: London, to see if these features were worth including in the model.

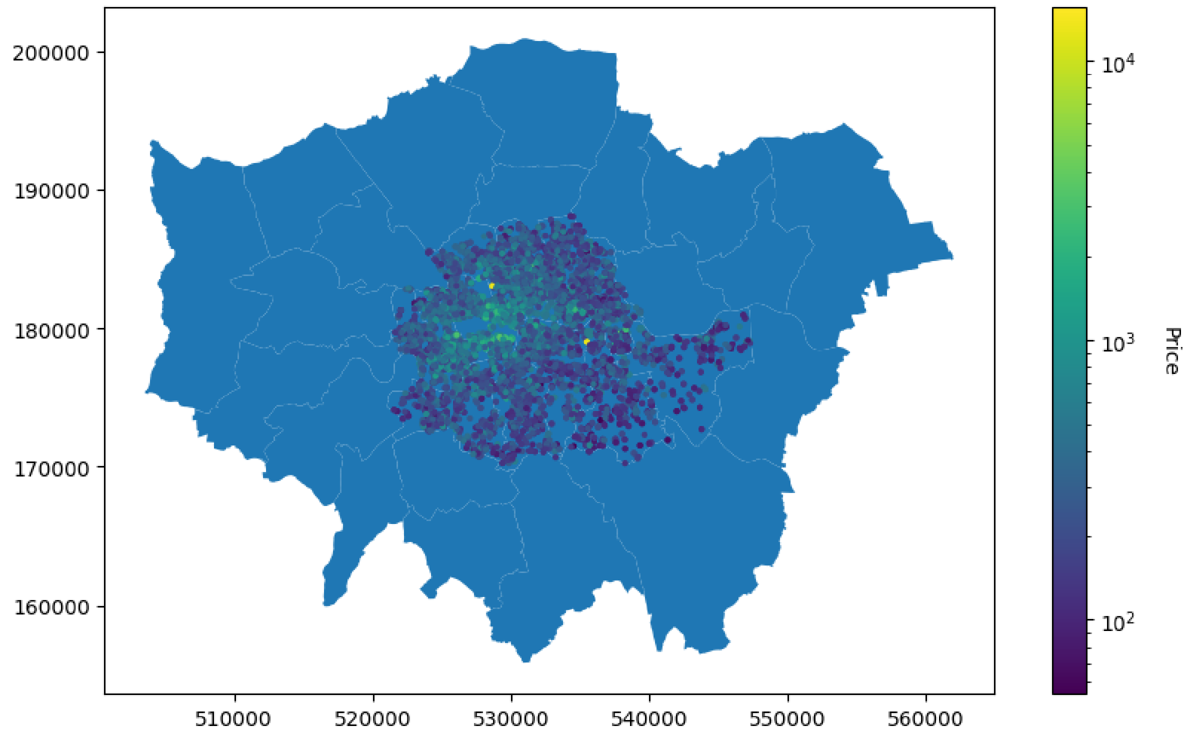
In [42]:
```
## projecting latitude and longitude data onto map of London

london_map = gpd.read_file("London_Boroughs.gpkg")
london_df = master_df[master_df["city"]=="London"]
gdf = gpd.GeoDataFrame(london_df, geometry=gpd.points_from_xy(london_df["lng
gdf = gdf.set_crs("EPSG:4326")
gdf = gdf.to_crs("EPSG:27700")

## assign listings colors according to price
norm = colors.LogNorm(vmin=min(gdf["realSum"]), vmax=max(gdf["realSum"]))
mapper = cm.ScalarMappable(norm=norm, cmap=cm.viridis)
cs = mapper.to_rgba(gdf["realSum"])

gdf.plot(ax=london_map.plot(figsize=(10, 6)), color=cs, markersize=4)
cbar = plt.colorbar(mapper)
cbar.ax.set_ylabel("Price", rotation=270, labelpad=20)
plt.show()
```

```
/var/folders/vh/dkh7yfgd01q_ngwks4pyljzc0000gn/T/ipykernel_3161/3641526311.
py:15: MatplotlibDeprecationWarning: Unable to determine Axes to steal spac
e for Colorbar. Using gca(), but will raise in the future. Either provide t
he *cax* argument to use as the Axes for the Colorbar, provide the *ax* arg
ument to steal space from it, or add *mappable* to an Axes.
  cbar = plt.colorbar(mapper)
```



It appears that latitude and longitude may convey a bit more information than the distance to the center of the city, since it appears the northern listings are priced more than the southern listings. I will keep these features in the following analysis.

- preprocessing step(s) (scaling, feature engineering/variable selection {based on predictors only}, lumping or dropping categories from predictors, one-hot encoding, etc.)

`room_shared` and `room_private` are redundant with the `room_type` column, and are removed. I also remove the unnormalized `attr_index` and `rest_index` and kept the normalized `attr_index_norm` and `rest_index_norm`. In the beginning, by combining all the cities and the weekend and weekday data into the `master_df` table, I introduced the additional features `city` (categorical) and `weekend` (boolean).

In [ ]:
```
## statistical tests for whether multi and biz columns have any impact on pr
```

In [ ]:
```
```

Additionally, I need to convert the categorical features `room_type` and `city` into an encoding that can be fed into my model. As I am interested in applying an ensemble method (random forest or gradient-boosted trees), I want to avoid one-hot-encoding as it would greatly increase the number of features in my data, which might affect performance if I am growing very shallow trees.

Instead, I will use target encoding, because my categorical features are unbalanced (see pie charts above), the ideal is leave-one-out target encoding with regularization described here to minimize data leakage. This introduces an additional hyperparemters per categorical feature, N_{pseudo}, which controls the regularization such that the target encoding is given by

$$\frac{N_{\text{pseudo}}}{N_{\text{category}} + N_{\text{pseudo}}} \times \text{overall average} + \frac{N_{\text{category}}}{N_{\text{category}} + N_{\text{pseudo}}} \times \text{average for category}$$

Leave-one-out target encoding prevents direct data leakage from the target to the feature for each observation, and the regularization (which is stronger for categories for which there are less data) helps to reduce indirect data leakage where the target value may be back-computed from the encodings of the other training data (which may occur in the case where only the average for each category is used).

First, I need to do the test-train split before target encoding to prevent data leakage.

In [52]:
```
X = master_df[["room_type", "person_capacity", "host_is_superhost", \
               "multi", "biz", "cleanliness_rating", "guest_satisfaction_ov
               "bedrooms", "dist", "metro_dist", "attr_index_norm", "rest_i
               "lng", "lat", "city", "weekend"]]
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 41514 entries, 0 to 1798
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   room_type                   41514 non-null  category
 1   person_capacity             41514 non-null  float64
 2   host_is_superhost           41514 non-null  bool
 3   multi                       41514 non-null  bool
 4   biz                         41514 non-null  bool
 5   cleanliness_rating          41514 non-null  float64
 6   guest_satisfaction_overall  41514 non-null  float64
 7   bedrooms                    41514 non-null  int64
 8   dist                        41514 non-null  float64
 9   metro_dist                  41514 non-null  float64
 10  attr_index_norm             41514 non-null  float64
 11  rest_index_norm             41514 non-null  float64
 12  lng                         41514 non-null  float64
 13  lat                         41514 non-null  float64
 14  city                        41514 non-null  category
 15  weekend                     41514 non-null  bool
dtypes: bool(4), category(2), float64(9), int64(1)
memory usage: 3.7 MB
```

In [104…
```
y = master_df["realSum"]
```

In [107…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

I ran out of time for implementing my own target encoder described above

I used this answer to efficiently compute leave-one-out means https://stackoverflow.com/questions/30274561/pandas-aggregating-average-while-excluding-current-row def target_encoder(col, y, N_pseudo): Xy = pd.concat([col, y], axis=1) grouped_Xy = Xy.groupby(col.name) N_category = grouped_Xy[y.name].transform("count") category_mean = grouped_Xy[y.name].transform("mean") smoothing = N_pseudo/(N_pseudo+N_category) Xy["encoding"] = smoothing * y.mean() + (1-smoothing)*((category_mean*N_category - y)/(N_category-1)) return Xy

I ended up using an out of the box target encoder which did not implmenent leave-one-out, but added gaussian noise.

In [ ]:
```
from category_encoders import MEstimateEncoder
```

- model choice (What model classes did you pick? Why?)

I chose a random forest because few of the features are expected to be irrelevant to the price, and a random forest is expected to give fairly accurate predictions in this case. It also allows me to analyse the variable importance, which I want in order to gain some intuition about the strongest determinants of price.

In [121…
```
pipeline = Pipeline([
    ("room_encoder", MEstimateEncoder(cols=["room_type"], m=5.0)),
    ("city_encoder", MEstimateEncoder(cols=["city"], m=5.0)),
    ('regressor', RandomForestRegressor(n_estimators=100))
])


## R^2 value with default parameters:
pipeline.fit(X_train, y_train)
r2 = pipeline.score(X_test, y_test)
print(r2)
```

```
0.391147915056
```

- model tuning (What hyperparameters did you tune? How? What loss function did you use and why? What was the range of achieved/minimized loss functions?)

I tuned the smoothing parameter for the target encodings, the maximum depth of the tree, the minimum number of samples required to split an internal node, the minimum number of samples required to be at a leaf node, the number of features to consider when looking for the best split, the maximum number of leaf nodes, and the number of trees.

I used the squared error as this was a regression problem, and alternatives like the absolute error turned out to be computationally too expensive.

In [153…
```
pipeline = Pipeline([
    ("room_encoder", MEstimateEncoder(cols=["room_type"])),
    ("city_encoder", MEstimateEncoder(cols=["city"])),
    ('regressor', RandomForestRegressor(n_estimators=100))
])

params = {
    "room_encoder__m":[1.0, 5.0], # smoothing factor for target encoding
    "city_encoder__m":[1.0, 5.0],
    "regressor__max_depth": [2, 3, None], # default None
    "regressor__max_leaf_nodes": [10, 20, None], # default None
    "regressor__min_samples_split": [2, 10], # default 2
    "regressor__max_features": ["sqrt", None], # default None=n_features
    "regressor__min_samples_leaf":[1, 2, 5], # default is 1
    "regressor__n_estimators": [100, 200], # default 100
}
```

In [152…
```
pipeline.get_params().keys()
```

Out[152]:
```
dict_keys(['memory', 'steps', 'verbose', 'room_encoder', 'city_encoder',
'regressor', 'room_encoder__cols', 'room_encoder__drop_invariant', 'room_e
ncoder__handle_missing', 'room_encoder__handle_unknown', 'room_encoder__
m', 'room_encoder__random_state', 'room_encoder__randomized', 'room_encode
r__return_df', 'room_encoder__sigma', 'room_encoder__verbose', 'city_encod
er__cols', 'city_encoder__drop_invariant', 'city_encoder__handle_missing',
'city_encoder__handle_unknown', 'city_encoder__m', 'city_encoder__random_s
tate', 'city_encoder__randomized', 'city_encoder__return_df', 'city_encode
r__sigma', 'city_encoder__verbose', 'regressor__bootstrap', 'regressor__cc
p_alpha', 'regressor__criterion', 'regressor__max_depth', 'regressor__max_
features', 'regressor__max_leaf_nodes', 'regressor__max_samples', 'regress
or__min_impurity_decrease', 'regressor__min_samples_leaf', 'regressor__min
_samples_split', 'regressor__min_weight_fraction_leaf', 'regressor__n_esti
mators', 'regressor__n_jobs', 'regressor__oob_score', 'regressor__random_s
tate', 'regressor__verbose', 'regressor__warm_start'])
```

- determining and fitting the best model

I used a randomized hyperparameters search using cross validation, which randomly selected parameters from the list above and executed 20 times to find a good set of parameters.

In [154…
```
clf = RandomizedSearchCV(pipeline, params, n_iter=20)
clf.fit(X_train,y_train)
```

Out[154]:

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
    ▸ MEstimateEncoder
    ▸ MEstimateEncoder
  ▸ RandomForestRegressor
```

In [164…

```python
results = pd.DataFrame(clf.cv_results_)
results = results.sort_values("rank_test_score")
results.head()
```

Out[164]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_room_encoder__
10	3.314962	0.015983	0.121402	0.001881	1
18	20.988758	0.131085	0.173398	0.002384	5
19	2.071283	0.028087	0.036335	0.000682	5
4	7.407877	0.106901	0.034256	0.000660	5
12	6.251591	0.076048	0.027774	0.000379	5

5 rows × 21 columns

In [156…

```python
results["params"][10]
```

Out[156]:

```
{'room_encoder__m': 1.0,
 'regressor__n_estimators': 100,
 'regressor__min_samples_split': 2,
 'regressor__min_samples_leaf': 1,
 'regressor__max_leaf_nodes': None,
 'regressor__max_features': 'sqrt',
 'regressor__max_depth': None,
 'city_encoder__m': 1.0}
```

The best model out of the 20 randomly tried was applied to the test set

In [157…

```python
pipeline = Pipeline([
    ("room_encoder", MEstimateEncoder(cols=["room_type"], m=1.0)),
    ("city_encoder", MEstimateEncoder(cols=["city"], m=1.0)),
    ('regressor', RandomForestRegressor(n_estimators=100, min_samples_split=
])
```

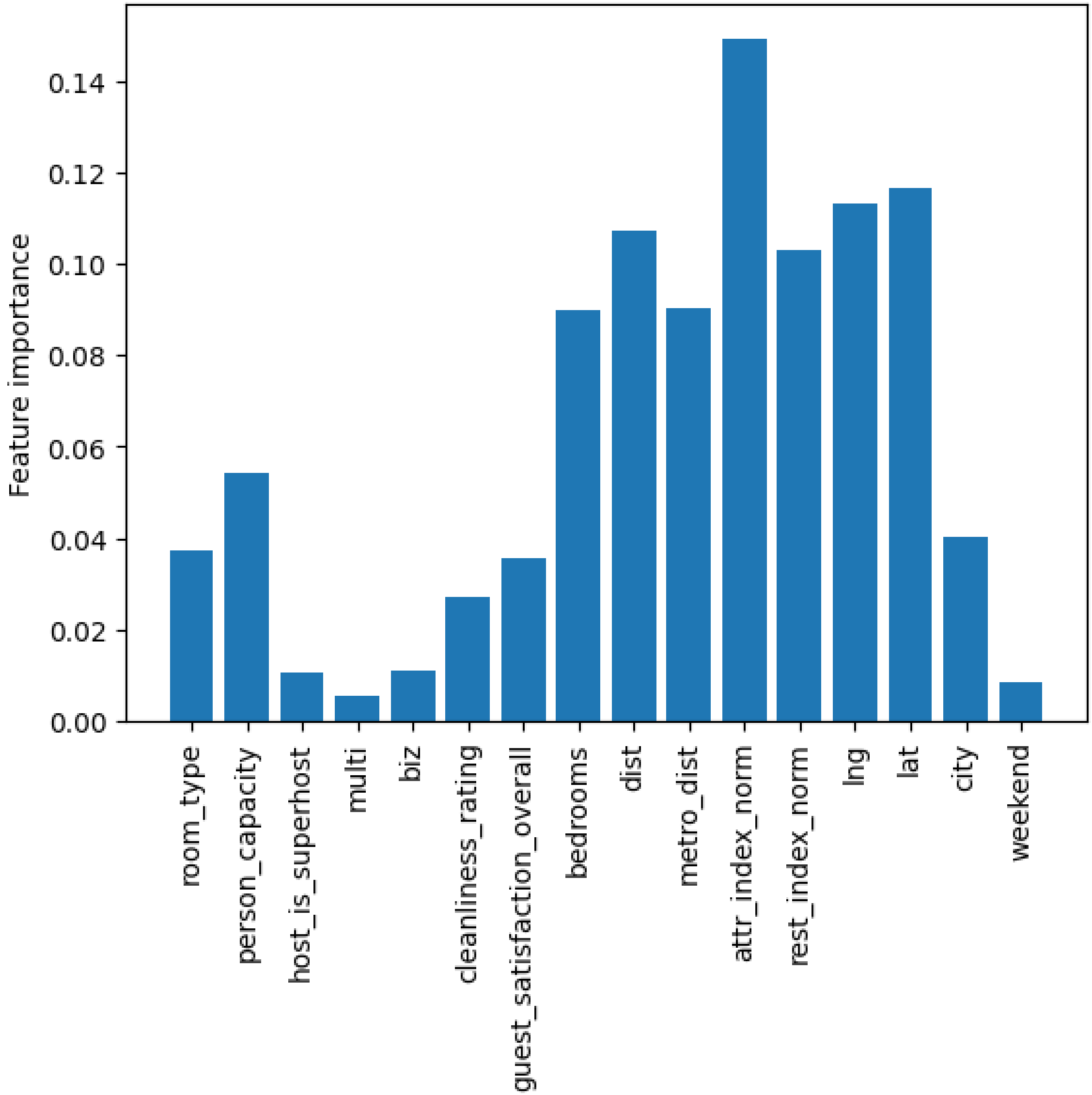
```
## R^2 value with optimized model:
pipeline.fit(X_train, y_train)
r2 = pipeline.score(X_test, y_test)
print(r2)
```

0.4912129239028461

- evaluate and explain the results of the model (partial dependence plots, variable importance, etc.)

In [158…
```
rf_regressor = pipeline['regressor']
feature_importances = rf_regressor.feature_importances_
```

In [160…
```
plt.bar(X_train.columns.values, feature_importances)
plt.xticks(rotation='vertical')
plt.ylabel("Feature importance")
plt.show()
```



The R^2 value of 0.49 of even the best model is quite low, and there are some aspects of the feature importance plot which are surprising. I would have thought the type of room (e.g. entire apartment vs shared room), and the city would have been more important features.

```
In [ ]:
```

```
In [ ]:
```