# DSCI 573 - Feature and Model Selection

# Lab 4: A mini project - Putting it all together

## Table of contents



## Submission instructions

---

rubric={mechanics:4}

You will receive marks for correctly submitting this assignment. To submit this assignment, follow the instructions below:

- **Which problem did you pick, classification or regression?** Regression
- **Report your test score here along with the metric used:** $R^2$ test score of the CatBoost model is 0.63.
- **Please add a link to your GitHub repository here: https://github.ubc.ca/mds-2021-22/DSCI_573_lab4_ming0701.git and https://github.ubc.ca/mds-2021-22/DSCI_573_lab4_leungws

- **You don't have to but you may work on this assignment in a group (group size <= 4) and submit your assignment as a group.**

- Below are some instructions on working as a group.
  - The maximum group size is 4.
  - You can choose your own group members. Since I don't know your groups in advance, I am not opening this lab as a group lab. So you all will have a separate GitHub repository for your labs and you'll have to decide how you want to collaborate.
  - Use group work as an opportunity to collaborate and learn new things from each other.
  - Be respectful to each other and make sure you understand all the concepts in the assignment well.

    - It's your responsibility to make sure that the assignment is submitted by one of the group members before the deadline. Here are some instructions on adding group members in Gradescope.
- Be sure to follow the general lab instructions.
- Make at least three commits in your lab's GitHub repository.
- Push the final .ipynb file with your solutions to your GitHub repository for this lab.
- Upload the .ipynb file to Gradescope.
- If the .ipynb file is too big or doesn't render on Gradescope for some reason, also upload a pdf or html in addition to the .ipynb.
- Make sure that your plots/output are rendered properly in Gradescope.

> Here you will find the description of each rubric used in MDS.
>
> As usual, do not push the data to the repository.

# Imports

In [1]:

```
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn import datasets
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.dummy import DummyRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.metrics import make_scorer

from sklearn.model_selection import (
    cross_val_score,
    cross_validate,
    RandomizedSearchCV,
    train_test_split,
)

from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.tree import DecisionTreeRegressor, export_graphviz

from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from sklearn.feature_selection import RFE, RFECV

import shap
```

# Introduction

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

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

### Tips

1. This mini-project is open-ended, and while working on it, there might be some situations where you'll have to use your own judgment and make your own decisions (as you would be doing when you work as a data scientist). Make sure you explain your decisions whenever necessary.
2. **Do not include everything you ever tried in your submission** -- it's fine just to have your final code. That said, your code should be reproducible and well-documented. For example, if you chose your hyperparameters based on some hyperparameter optimization experiment, you should leave in the code for that experiment so that someone else could re-run it and obtain the same hyperparameters, rather than mysteriously just setting the hyperparameters to some (carefully chosen) values in your code.
3. If you realize that you are repeating a lot of code try to organize it in functions. Clear presentation of your code, experiments, and results is the key to be successful in this lab. You may use code from lecture notes or previous lab solutions with appropriate attributions.

### Assessment

We plan to grade fairly and leniently. We don't have some secret target score that you need to achieve to get a good grade. **You'll be assessed on demonstration of mastery of course topics, clear presentation, and the quality of your analysis and results.** For example, if you just have a bunch of code and no text or figures, that's not good. If you do a bunch of sane things and get a lower accuracy than your friend, don't sweat it.

### A final note

Finally, this style of this "project" question is different from other assignments. It'll be up to you to decide when you're "done" -- in fact, this is one of the hardest parts of real projects. But please don't spend WAY too much time on this... perhaps "a few hours" (2-8 hours???) is a good guideline for a typical submission. Of course if you're having fun you're welcome to spend as much time as you want! But, if so, try not to do it out of perfectionism or getting the best possible grade. Do it because you're learning and enjoying it. Students from the past cohorts have found such kind of labs useful and fun and I hope you enjoy it as well.

# 1. Pick your problem and explain what exactly you are trying to predict

---

rubric={reasoning:4}

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

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

OR

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

**Your tasks:**

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

Initial thoughts on the dataset:

- name is text feature which is likely to provide useful infomation for prediction
- host_name is not likely useful as there should be a lot of duplicated names (e.g. John) but they are not the same host
- reviews per month is likely positively correlated to number_of_reviews

In [2]:
```
airbnb_df = pd.read_csv("AB_NYC_2019.csv")
airbnb_df.head()
```

Out[2]:

| | id | name | host_id | host_name | neighbourhood_group | neighbourhood | latitude | longitude | r |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 2539 | Clean & quiet apt home by the park | 2787 | John | Brooklyn | Kensington | 40.64749 | -73.97237 | |
| **1** | 2595 | Skylit Midtown Castle | 2845 | Jennifer | Manhattan | Midtown | 40.75362 | -73.98377 | |
| **2** | 3647 | THE VILLAGE OF HARLEM....NEW YORK ! | 4632 | Elisabeth | Manhattan | Harlem | 40.80902 | -73.94190 | |
| **3** | 3831 | Cozy Entire Floor of Brownstone | 4869 | LisaRoxanne | Brooklyn | Clinton Hill | 40.68514 | -73.95976 | |
| **4** | 5022 | Entire Apt: Spacious Studio/Loft by central park | 7192 | Laura | Manhattan | East Harlem | 40.79851 | -73.94399 | |

In [3]:
```
airbnb_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48895 entries, 0 to 48894
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              48895 non-null  int64
 1   name                            48879 non-null  object
 2   host_id                         48895 non-null  int64
 3   host_name                       48874 non-null  object
 4   neighbourhood_group             48895 non-null  object
 5   neighbourhood                   48895 non-null  object
```

```
 6   latitude                        48895 non-null  float64
 7   longitude                       48895 non-null  float64
 8   room_type                       48895 non-null  object
 9   price                           48895 non-null  int64
 10  minimum_nights                  48895 non-null  int64
 11  number_of_reviews               48895 non-null  int64
 12  last_review                     38843 non-null  object
 13  reviews_per_month               38843 non-null  float64
 14  calculated_host_listings_count  48895 non-null  int64
 15  availability_365                48895 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 6.0+ MB
```

There are missing values in our target column "reviews_per_month". Let's drop those rows. There are also a few rows of missing values in column "name". In order not to complicate the preprocessing, let's also drop these few rows.

In [4]:
```
airbnb_df = airbnb_df.dropna(subset=['reviews_per_month'])
airbnb_df = airbnb_df.dropna(subset=['name'])
```

Column "last_review" is date, so we convert it to number of days by subtracting the date by the minimum date of the column.

In [5]:
```
from datetime import datetime
airbnb_df['last_review'] = pd.to_datetime(airbnb_df['last_review'])
airbnb_df['last_review'] = (airbnb_df.last_review - airbnb_df.last_review.min()).dt.days
```

Column "price" is likely to crash with other features from countvectorizer, so we change the name to made it clear.

In [6]:
```
airbnb_df = airbnb_df.rename(columns={"price": "price_per_night"})
```

## 2. Data splitting

---

rubric={reasoning:2}

**Your tasks:**

1. Split the data into train and test portions.

> Make decision on the `test_size` based on the capacity of your laptop. Don't forget to use a random state.

In [7]:
```
train_df, test_df = train_test_split(airbnb_df, test_size=0.9, random_state=123)
```

## 3. EDA

---

rubric={viz:4,reasoning:6}

**Your tasks:**

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

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3883 entries, 36503 to 18946
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              3883 non-null   int64  
 1   name                            3883 non-null   object 
 2   host_id                         3883 non-null   int64  
 3   host_name                       3883 non-null   object 
 4   neighbourhood_group             3883 non-null   object 
 5   neighbourhood                   3883 non-null   object 
 6   latitude                        3883 non-null   float64
 7   longitude                       3883 non-null   float64
 8   room_type                       3883 non-null   object 
 9   price_per_night                 3883 non-null   int64  
 10  minimum_nights                  3883 non-null   int64  
 11  number_of_reviews               3883 non-null   int64  
 12  last_review                     3883 non-null   int64  
 13  reviews_per_month               3883 non-null   float64
 14  calculated_host_listings_count  3883 non-null   int64  
 15  availability_365                3883 non-null   int64  
dtypes: float64(3), int64(8), object(5)
memory usage: 515.7+ KB
```

## 3.1 - Datatype and counts

- There are numeric and non-numeric features.
- There are no missing values after dropping null ID and names.

In [9]:
```
train_df.describe()
```

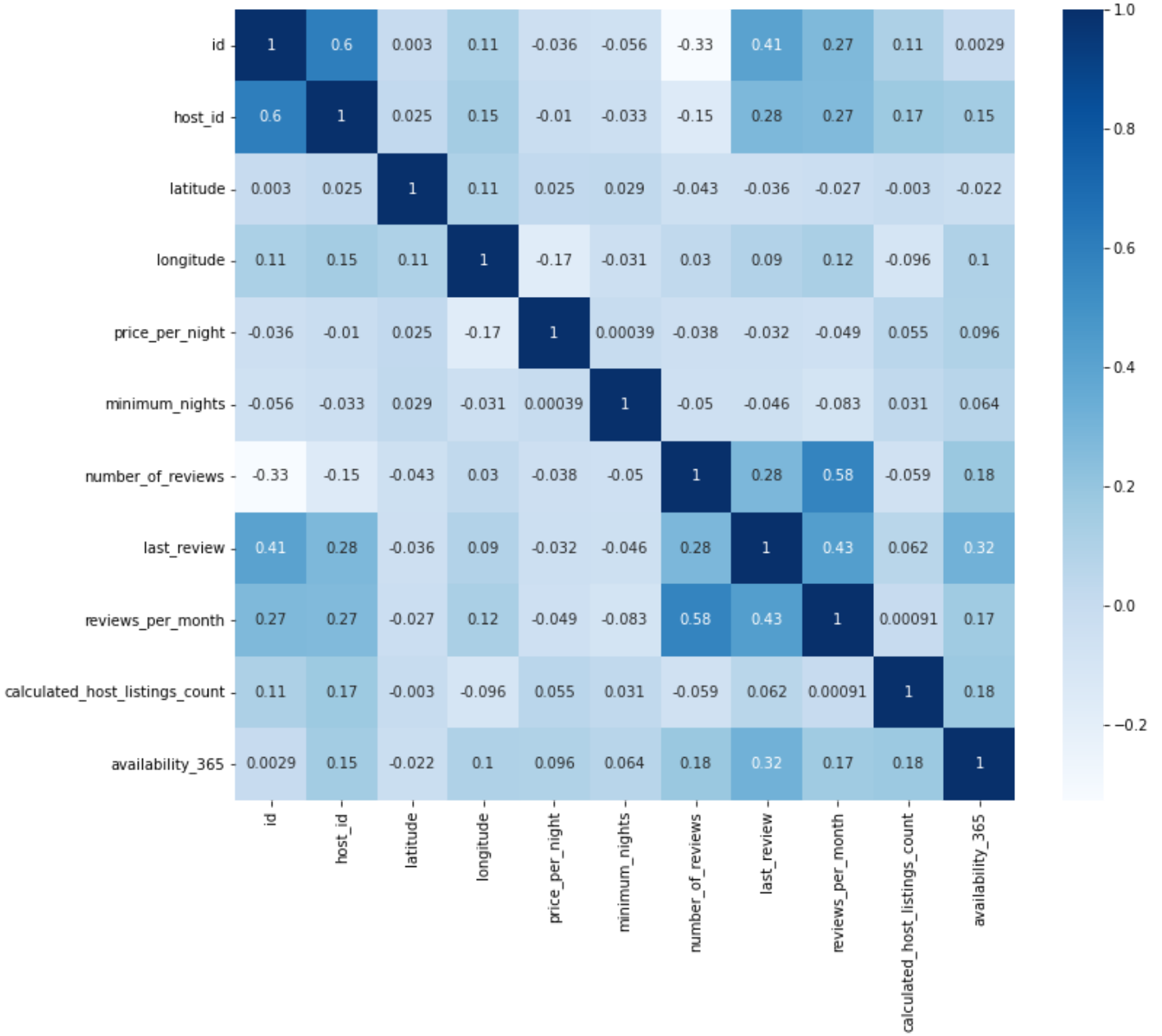
Out[9]:

| | id | host_id | latitude | longitude | price_per_night | minimum_nights | number_ |
|---|---|---|---|---|---|---|---|
| **count** | 3.883000e+03 | 3.883000e+03 | 3883.000000 | 3883.000000 | 3883.000000 | 3883.000000 | 38 |
| **mean** | 1.782766e+07 | 6.406118e+07 | 40.728180 | -73.951291 | 144.245429 | 6.476436 | |
| **std** | 1.071418e+07 | 7.661151e+07 | 0.054516 | 0.046711 | 183.299469 | 28.096501 | |
| **min** | 6.090000e+03 | 2.881000e+03 | 40.541790 | -74.210170 | 0.000000 | 1.000000 | |
| **25%** | 8.421444e+06 | 6.893108e+06 | 40.689875 | -73.982785 | 70.000000 | 1.000000 | |
| **50%** | 1.836104e+07 | 2.837092e+07 | 40.721500 | -73.954780 | 105.000000 | 2.000000 | |
| **75%** | 2.728145e+07 | 9.699893e+07 | 40.762405 | -73.935130 | 175.000000 | 4.000000 | |
| **max** | 3.643834e+07 | 2.712752e+08 | 40.897020 | -73.727780 | 6000.000000 | 999.000000 | 6 |

## 3.2 - Summary statistics of numeric features

- Unique values are observed in some features such as id and host_id.
- Different scales are observed for features.

In [10]:
```
cor = train_df.corr()
import seaborn as sns
plt.figure(figsize=(12, 10))
sns.heatmap(cor, annot=True, cmap=plt.cm.Blues)
plt.show()
```



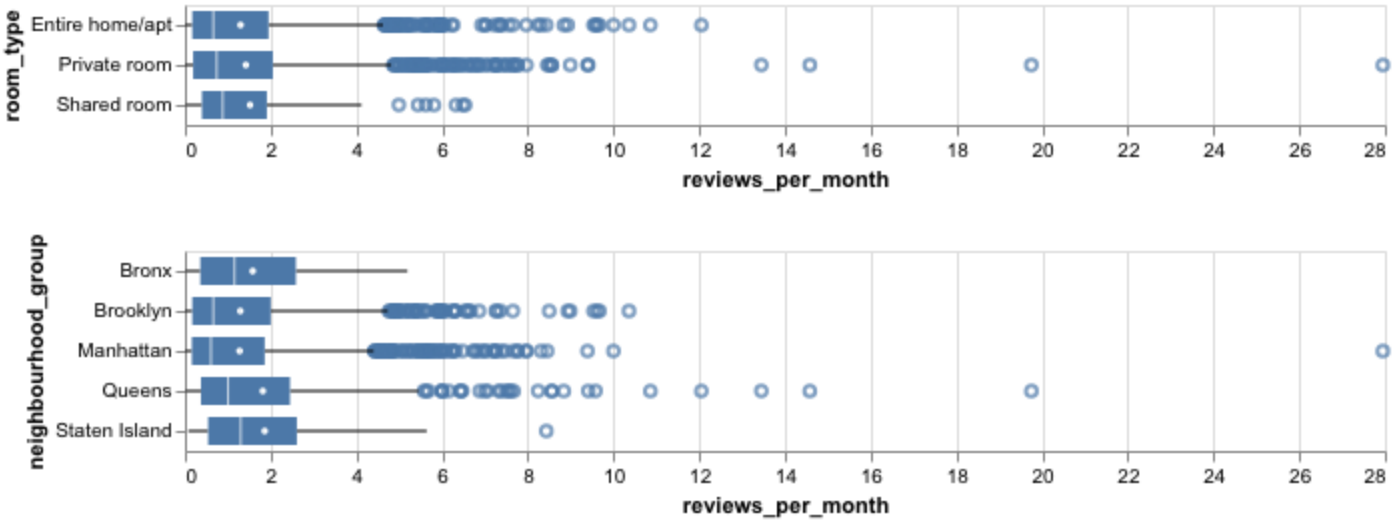

## 3.3 - Correlation heatmap of numeric features

- Correlation of all numeric features in pairs are shown in a heatmap with correlation values.

In [11]:
```
import altair as alt

box_room = alt.Chart(train_df).mark_boxplot().encode(
    y=alt.Y('room_type', sort='-x'),
    x='reviews_per_month'
)
```

```
point_room = alt.Chart(train_df).mark_point(color='white', size=2).encode(
    y=alt.Y('room_type'),
    x=alt.X('mean(reviews_per_month)')
)

box_neighbourhood = alt.Chart(train_df).mark_boxplot().encode(
    y=alt.Y('neighbourhood_group'),
    x=alt.X('reviews_per_month',
            sort=alt.EncodingSortField(
                    field='reviews_per_month',
                    op='median',
                    order='descending'))
)
box_neighbourhood
point_neighbourhood = alt.Chart(train_df).mark_point(
    color='white', size=2).encode(
    y=alt.Y('neighbourhood_group:N'),
    x=alt.X('mean(reviews_per_month)')
)

room = (box_room + point_room).properties(width=600)
neighbour = (box_neighbourhood + point_neighbourhood).properties(width=600)
room & neighbour
```

Out[11]:




## 3.4 - Boxplots for categorical features

- Boxplots were used to explore relationships of response variable and categorical features such as room type and neighbourhood groups with median and mean in white dot as well as outliers.

## 3.5 - EDA Observations and Summary

- The target is reviews per month that is a continuous data and a regression model would be a good choice for prediction.
- Some features are unique for each observations that is not useful in the prediction model (eg. id, host_id, host name)
- There are several categorical features that requires transformation with One-hot encoding and there are outliers in values (eg. room_type, neighbourhood_group, neighbourhood).
- Numeric features have different scales that requires standardised scaling (eg. latitude, price_per_night, minimum_nights and etc).
- The heatmap of correlation of numeric features show that there is a stronger association between reviews_per_month and last_review or number_of_review.

- There is a text feature of property name that could have impact on prediction that requires CountVectorizer transformation.

## (Optional) 4. Feature engineering

---

rubric={reasoning:1}

**Your tasks:**

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

In [12]:
```
#train_df.info()
```

## 5. Preprocessing and transformations

---

rubric={accuracy:6,reasoning:4}

**Your tasks:**

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

In [13]:
```
drop_features = ["id", "host_id", "host_name"]
categorical_features = ["neighbourhood_group", "neighbourhood", "room_type"]
text_feature = "name"
target = "reviews_per_month"
numeric_features = list(
    set(train_df.columns)
    - set(drop_features)
    - set([text_feature])
    - set(categorical_features)
    - set([target])
)
```

In [14]:
```
preprocessor = make_column_transformer(
    (StandardScaler(), numeric_features),
    (OneHotEncoder(handle_unknown="ignore", sparse=False, dtype="int"),
     categorical_features),
    (CountVectorizer(stop_words="english"), text_feature),
    ("drop", drop_features),
)
```

In [15]:
```
X_train, y_train = train_df.drop(columns=[target]), train_df[target]
X_test, y_test = test_df.drop(columns=[target]), test_df[target]
```

# 6. Baseline model

---

rubric={accuracy:2}

**Your tasks:**

1. Try `scikit-learn` 's baseline model and report results.

In [16]:
```
from sklearn.metrics import make_scorer


def mape(true, pred):
    return 100.0 * np.mean(np.abs((pred - true) / true))


# make a scorer function that we can pass into cross-validation
mape_scorer = make_scorer(mape, greater_is_better=False)

scoring_metrics = {
    "neg RMSE": "neg_root_mean_squared_error",
    "r2": "r2",
    "mape": mape_scorer,
}
```

In [17]:
```
def mean_std_cross_val_scores(model, X_train, y_train, **kwargs):
    """
    Returns mean and std of cross validation

    Parameters
    ----------
    model :
        scikit-learn model
    X_train : numpy array or pandas DataFrame
        X in the training data
    y_train :
        y in the training data

    Returns
    ----------
        pandas Series with mean scores from cross_validation
    """

    scores = cross_validate(model, X_train, y_train, **kwargs)

    mean_scores = pd.DataFrame(scores).mean()
    std_scores = pd.DataFrame(scores).std()
    out_col = []

    for i in range(len(mean_scores)):
        out_col.append((f"%0.3f (+/- %0.3f)" % (mean_scores[i], std_scores[i])))

    return pd.Series(data=out_col, index=mean_scores.index)
```

In [18]:
```
results = {}
```

In [19]:
```
dummy = DummyRegressor()
results["Dummy"] = mean_std_cross_val_scores(
    dummy, X_train, y_train, return_train_score=True, scoring = scoring_metrics
)
pd.DataFrame(results)
```

Out[19]:

| | Dummy |
|---|---|
| **fit_time** | 0.001 (+/- 0.000) |
| **score_time** | 0.001 (+/- 0.000) |
| **test_neg RMSE** | -1.700 (+/- 0.169) |
| **train_neg RMSE** | -1.706 (+/- 0.043) |
| **test_r2** | -0.001 (+/- 0.001) |
| **train_r2** | 0.000 (+/- 0.000) |
| **test_mape** | -676.365 (+/- 45.614) |
| **train_mape** | -676.403 (+/- 13.640) |

# 7. Linear models

---

rubric={accuracy:6,reasoning:4}

**Your tasks:**

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

In [20]:
```
pipe_ridge = make_pipeline(preprocessor, Ridge(random_state=123))
results["Ridge"] = mean_std_cross_val_scores(
    pipe_ridge, X_train, y_train, return_train_score=True, scoring = scoring_metrics
)
pd.DataFrame(results)
```

Out[20]:

| | Dummy | Ridge |
|---|---|---|
| **fit_time** | 0.001 (+/- 0.000) | 0.051 (+/- 0.006) |
| **score_time** | 0.001 (+/- 0.000) | 0.010 (+/- 0.001) |
| **test_neg RMSE** | -1.700 (+/- 0.169) | -1.404 (+/- 0.156) |
| **train_neg RMSE** | -1.706 (+/- 0.043) | -0.954 (+/- 0.046) |
| **test_r2** | -0.001 (+/- 0.001) | 0.315 (+/- 0.070) |
| **train_r2** | 0.000 (+/- 0.000) | 0.687 (+/- 0.020) |
| **test_mape** | -676.365 (+/- 45.614) | -336.259 (+/- 30.196) |
| **train_mape** | -676.403 (+/- 13.640) | -233.005 (+/- 7.054) |

In [21]:
```
param_grid = {
    "ridge__alpha": 10.0 ** np.arange(-3, 5),
    "columntransformer__countvectorizer__max_features": [500, 1000, 5000, 10000, 20000]
}
```

In [22]:
```
random_search = RandomizedSearchCV(
    pipe_ridge,
    param_grid,
    n_iter=40,
    n_jobs=-1,
    random_state = 123,
    scoring = "r2"
)

random_search.fit(X_train, y_train);
```

In [23]:
```
pd.DataFrame(random_search.cv_results_).set_index("rank_test_score").sort_index().T
```

Out[23]:

| rank_test_score | 1 | 1 |
|---|---|---|
| mean_fit_time | 0.060681 | 0.064381 |
| std_fit_time | 0.008825 | 0.003903 |
| mean_score_time | 0.012445 | 0.017472 |
| std_score_time | 0.003968 | 0.005075 |
| param_ridge__alpha | 100.0 | 100.0 |
| param_columntransformer__countvectorizer__max_features | 5000 | 10000 |
| params | {'ridge__alpha': 100.0, 'columntransformer__co... | {'ridge__alpha': 100.0, 'columntransformer__co... |
| split0_test_score | 0.353411 | 0.353411 |
| split1_test_score | 0.418031 | 0.418031 |
| split2_test_score | 0.4488 | 0.4488 |
| split3_test_score | 0.476891 | 0.476891 |
| split4_test_score | 0.468178 | 0.468178 |
| mean_test_score | 0.433062 | 0.433062 |
| std_test_score | 0.044659 | 0.044659 |

14 rows × 40 columns

In [24]:
```
print("Best hyperparameter values: ", random_search.best_params_)
print("Best score: %0.3f" % (random_search.best_score_))
```

```
Best hyperparameter values:  {'ridge__alpha': 100.0, 'columntransformer__countvectorizer__
max_features': 5000}
Best score: 0.433
```

## 7. Results

- The first attempt results of the Ridge model is better than the baseline for all metrics with $R^2$ validation score 0.315 compared to -0.001 of Dummy Regressor.

- Other metrics are also better such as MAPE is 336, which is half of that of Dummy Regressor. The RMSE is also lower.
- Although the fit time of Ridge is longer than Dummy Regressor, the score time is much faster and similar to Dummy Regressor.
- The first attempt Ridge model has overfitting with validation score much lower than the train score.
- To further improve the $R^2$ validation score, we performed hyperparameter tuning and got better result of 0.433 with hyperparameter values of alpha at 100 and max features at 5000.

# 8. Different models

---

rubric={accuracy:10,reasoning:6}

**Your tasks:**

1. Try at least 3 other models aside from a linear model.
2. Summarize your results in terms of overfitting/underfitting and fit and score times. Can you beat a linear model?

In [25]:
```
pipe_rf = make_pipeline(preprocessor, RandomForestRegressor(random_state=123))
pipe_xgb = make_pipeline(preprocessor, XGBRegressor(random_state=123, verbosity=0))
pipe_lgbm = make_pipeline(preprocessor, LGBMRegressor(random_state=123))
pipe_catboost = make_pipeline(preprocessor, CatBoostRegressor(verbose=0,
                                                              random_state=123))

models = {
    "Random Forest": pipe_rf,
    "XGBoost": pipe_xgb,
    "LightGBM": pipe_lgbm,
    "CatBoost": pipe_catboost
}
```

In [26]:
```
for (name, model) in models.items():
    results[name] = mean_std_cross_val_scores(
        model, X_train, y_train, return_train_score=True, scoring=scoring_metrics
    )
pd.DataFrame(results)
```

Out[26]:

| | Dummy | Ridge | Random Forest | XGBoost | LightGBM | CatBoost |
|---|---|---|---|---|---|---|
| **fit_time** | 0.001 (+/- 0.000) | 0.051 (+/- 0.006) | 8.482 (+/- 0.047) | 0.481 (+/- 0.033) | 0.399 (+/- 0.022) | 3.071 (+/- 0.081) |
| **score_time** | 0.001 (+/- 0.000) | 0.010 (+/- 0.001) | 0.025 (+/- 0.001) | 0.015 (+/- 0.002) | 0.014 (+/- 0.001) | 0.013 (+/- 0.000) |
| **test_neg RMSE** | -1.700 (+/- 0.169) | -1.404 (+/- 0.156) | -1.087 (+/- 0.165) | -1.106 (+/- 0.154) | -1.083 (+/- 0.137) | -1.063 (+/- 0.171) |
| **train_neg RMSE** | -1.706 (+/- 0.043) | -0.954 (+/- 0.046) | -0.408 (+/- 0.018) | -0.511 (+/- 0.014) | -0.594 (+/- 0.019) | -0.647 (+/- 0.010) |
| **test_r2** | -0.001 (+/- 0.001) | 0.315 (+/- 0.070) | 0.593 (+/- 0.042) | 0.578 (+/- 0.036) | 0.595 (+/- 0.034) | 0.611 (+/- 0.051) |

| | Dummy | Ridge | Random Forest | XGBoost | LightGBM | CatBoost |
|---|---|---|---|---|---|---|
| **train_r2** | 0.000 (+/- 0.000) | 0.687 (+/- 0.020) | 0.943 (+/- 0.003) | 0.910 (+/- 0.010) | 0.879 (+/- 0.003) | 0.856 (+/- 0.006) |
| **test_mape** | -676.365 (+/- 45.614) | -336.259 (+/- 30.196) | -57.752 (+/- 4.537) | -87.139 (+/- 5.519) | -79.677 (+/- 4.872) | -82.616 (+/- 9.938) |
| **train_mape** | -676.403 (+/- 13.640) | -233.005 (+/- 7.054) | -21.536 (+/- 0.625) | -55.935 (+/- 2.326) | -56.251 (+/- 0.960) | -63.314 (+/- 1.921) |

## 8. Results comparison of different models

- The four other models can beat the linear model with the below reasons:
- All the four other models have higher validation $R^2$ scores than Ridge or Dummy Regressor.
- CatBoost has the best $R^2$ validation score 0.611 and lowest RMSE. Also it overfits less than all other models with a narrower gap between train and validation scores.
- All four models have better RMSE and MAPE than Ridge or Dummy Regressor.
- All four models have similar RMSE. XGBoost, LightGBM and Catboost have similar MAPE while Random Forest is slightly better in MAPE.
- Although fit time of CatBoost is long but it is still faster than Random Forest. XGBoost, LightGBC and Catboost have similar score time as Ridge, and they are also faster than Random Forest.

# (Optional) 9. Feature selection

---

rubric={reasoning:1}

**Your tasks:**

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

In [27]:
```
#run super long time, give up!
#pipe_rfe_ridgecv = make_pipeline(preprocessor, RFECV(Ridge(), cv=10), RidgeCV())
#results['rfe+ridgecv'] = mean_std_cross_val_scores(pipe_rfe_ridgecv, X_train, y_train, re
#pd.DataFrame(results)
```

# 10. Hyperparameter optimization

---

rubric={accuracy:6,reasoning:4}

**Your tasks:**

Make some attempts to optimize hyperparameters for the models you've tried and summarize your results. In at least one case you should be optimizing multiple hyperparameters for a single model. You may use

`sklearn` 's methods for hyperparameter optimization or fancier Bayesian optimization methods.

- GridSearchCV
- RandomizedSearchCV
- scikit-optimize

In [28]:
```
param_grid_rf = {
    "randomforestregressor__max_depth": [2, 5, 10, 15, 20, 25, 50],
    "randomforestregressor__min_samples_split": [2, 5, 10],
    "randomforestregressor__bootstrap": [True, False],
    "randomforestregressor__n_estimators": [1, 5, 10, 25, 50, 100, 200]
}
```

In [29]:
```
random_search_rf = RandomizedSearchCV(
    pipe_rf,
    param_grid_rf,
    n_iter=20,
    n_jobs=-1,
    random_state = 123,
    scoring = "r2"
)

random_search_rf.fit(X_train, y_train);
```

In [30]:
```
pd.DataFrame(random_search_rf.cv_results_).set_index("rank_test_score").sort_index().T
```

Out[30]:

| rank_test_score | 1 | |
|---|---|---|
| mean_fit_time | 16.792904 | |
| std_fit_time | 0.181 | |
| mean_score_time | 0.046063 | |
| std_score_time | 0.003155 | |
| param_randomforestregressor__n_estimators | 100 | |
| param_randomforestregressor__min_samples_split | 2 | |
| param_randomforestregressor__max_depth | 50 | |
| param_randomforestregressor__bootstrap | True | |
| params | {'randomforestregressor__n_estimators': 100, '... | {'randomforestregresso |
| split0_test_score | 0.525709 | |
| split1_test_score | 0.629862 | |
| split2_test_score | 0.627466 | |
| split3_test_score | 0.584265 | |
| split4_test_score | 0.597122 | |
| mean_test_score | 0.592885 | |
| std_test_score | 0.037863 | |

In [31]:
```
print("Best hyperparameter values: ", random_search_rf.best_params_)
print("Best score: %0.3f" % (random_search_rf.best_score_))
```

```
Best hyperparameter values:  {'randomforestregressor__n_estimators': 100, 'randomforestreg
ressor__min_samples_split': 2, 'randomforestregressor__max_depth': 50, 'randomforestregres
sor__bootstrap': True}
Best score: 0.593
```

In [32]:

```
param_grid_cb = {
    "catboostregressor__max_depth": [2, 5, 10, 15],
    "catboostregressor__learning_rate": [0.01, 0.1, 0.2, 0.5, 1],
    "catboostregressor__n_estimators": [1, 5, 10, 25, 50, 100, 200]
}
```

In [33]:

```
random_search_cb = RandomizedSearchCV(
    pipe_catboost,
    param_grid_cb,
    n_iter=20,
    n_jobs=-1,
    random_state = 123,
    scoring = "r2"
)

random_search_cb.fit(X_train, y_train);
```

```
A worker stopped while some jobs were given to the executor. This can be caused by a too s
hort worker timeout or by a memory leak.
```

In [34]:

```
pd.DataFrame(random_search_cb.cv_results_).set_index("rank_test_score").sort_index().T
```

Out[34]:

| rank_test_score | 1 | 2 |
|---|---|---|
| mean_fit_time | 4.743081 | 2.229183 |
| std_fit_time | 0.155495 | 0.124751 |
| mean_score_time | 0.029467 | 0.16112 |
| std_score_time | 0.013574 | 0.065441 |
| param_catboostregressor__n_estimators | 50 | 100 |
| param_catboostregressor__max_depth | 10 | 2 |
| param_catboostregressor__learning_rate | 0.2 | 0.1 |
| params | {'catboostregressor__n_estimators': 50, 'catbo... | {'catboostregressor__n_estimators': 100, 'catb... |
| split0_test_score | 0.525593 | 0.505174 |
| split1_test_score | 0.637797 | 0.599285 |
| split2_test_score | 0.626295 | 0.625835 |
| split3_test_score | 0.624617 | 0.595134 |
| split4_test_score | 0.615584 | 0.598589 |
| mean_test_score | 0.605977 | 0.584803 |
| std_test_score | 0.040809 | 0.041306 |

In [35]:

```
print("Best hyperparameter values: ", random_search_cb.best_params_)
print("Best score: %0.3f" % (random_search_cb.best_score_))
```

```
Best hyperparameter values:  {'catboostregressor__n_estimators': 50, 'catboostregressor__m
ax_depth': 10, 'catboostregressor__learning_rate': 0.2}
```

```
Best score: 0.606
```

In [36]:
```
param_grid_lgbm = {
    "lgbmregressor__reg_alpha": [0.01, 0.1, 1, 10, 100],
    "lgbmregressor__reg_lambda": [0.01, 0.1, 1, 10, 100],
    "lgbmregressor__num_leaves": [2, 5, 10, 25, 50]
}
```

In [37]:
```
random_search_lgbm = RandomizedSearchCV(
    pipe_lgbm,
    param_grid_lgbm,
    n_iter=20,
    n_jobs=-1,
    random_state = 123,
    scoring = "r2"
)

random_search_lgbm.fit(X_train, y_train);
```

In [38]:
```
pd.DataFrame(random_search_lgbm.cv_results_).set_index("rank_test_score").sort_index().T
```

Out[38]:

| rank_test_score | 1 | 2 | |
|---|---|---|---|
| mean_fit_time | 0.139361 | 0.293515 | |
| std_fit_time | 0.013819 | 0.016206 | |
| mean_score_time | 0.020635 | 0.034303 | |
| std_score_time | 0.001093 | 0.001818 | |
| param_lgbmregressor__reg_lambda | 1 | 100 | |
| param_lgbmregressor__reg_alpha | 10 | 1 | |
| param_lgbmregressor__num_leaves | 5 | 50 | |
| params | {'lgbmregressor__reg_lambda': 1, 'lgbmregresso... | {'lgbmregressor__reg_lambda': 100, 'lgbmregres... | {'lgbmregressor 1, ' |
| split0_test_score | 0.553592 | 0.537554 | |
| split1_test_score | 0.624335 | 0.652737 | |
| split2_test_score | 0.608861 | 0.630668 | |
| split3_test_score | 0.623007 | 0.603157 | |
| split4_test_score | 0.60609 | 0.583164 | |
| mean_test_score | 0.603177 | 0.601456 | |
| std_test_score | 0.025847 | 0.039759 | |

In [39]:
```
print("Best hyperparameter values: ", random_search_lgbm.best_params_)
print("Best score: %0.3f" % (random_search_lgbm.best_score_))
```

```
Best hyperparameter values:  {'lgbmregressor__reg_lambda': 1, 'lgbmregressor__reg_alpha':
10, 'lgbmregressor__num_leaves': 5}
Best score: 0.603
```

## 10. Hyperparameter optimization results

- We performed hyperparameter optimization for top 3 performing model, randomforest, catboost and lgbm.
- Randomsearch was used with different hyperparameters.
- For randomforest, the $R^2$ test score remained the same at 0.593 with best parameters n_estimators: 100, min_samples_split': 2, max_depth: 50 and bootstrap: True.
- For catboost, the $R^2$ test score did not improve and the default hyperparameters give the best score.
- For lgbm, the $R^2$ test score improved from 0.595 to 0.603 with best parameters lambda:1, alpha:10 and num_leaves:5.

## 11. Interpretation and feature importances

---

rubric={accuracy:6,reasoning:4}

**Your tasks:**

1. Use the methods we saw in class (e.g., `eli5`, `shap`), or any other methods of your choice, to examine the most important features of one of the non-linear models.
2. Summarize your observations.

In [40]:
```
preprocessor.fit(X_train)
preprocessor.named_transformers_
```

Out[40]:
```
{'standardscaler': StandardScaler(),
 'onehotencoder': OneHotEncoder(dtype='int', handle_unknown='ignore', sparse=False),
 'countvectorizer': CountVectorizer(stop_words='english'),
 'drop': 'drop'}
```

In [41]:
```
ohe_columns = list(
    preprocessor.named_transformers_["onehotencoder"]
    .get_feature_names_out(categorical_features)
)
cv_columns = list(
    preprocessor.named_transformers_["countvectorizer"]
    .get_feature_names_out()
)

new_columns = (numeric_features + ohe_columns + cv_columns)
```

In [42]:
```
pipe_catboost.fit(X_train, y_train)
data={'importances': pipe_catboost.named_steps['catboostregressor'].feature_importances_
      }
rf_df=pd.DataFrame(data, index=new_columns).sort_values("importances",
                                                        ascending = False)
rf_df.head(10)
```

Out[42]:

| | importances |
|---|---|
| **last_review** | 37.918860 |
| **number_of_reviews** | 26.225895 |
| **minimum_nights** | 10.865854 |

| | importances |
|---|---|
| **availability_365** | 4.038547 |
| **longitude** | 2.661107 |
| **price_per_night** | 2.064837 |
| **neighbourhood_Theater District** | 1.820210 |
| **latitude** | 0.949476 |
| **calculated_host_listings_count** | 0.784448 |
| **jfk** | 0.784099 |

In [43]:
```
X_train_enc = pd.DataFrame(
    data=preprocessor.transform(X_train).toarray(),
    columns=new_columns,
    index=X_train.index,
)
X_train_enc.head()
```

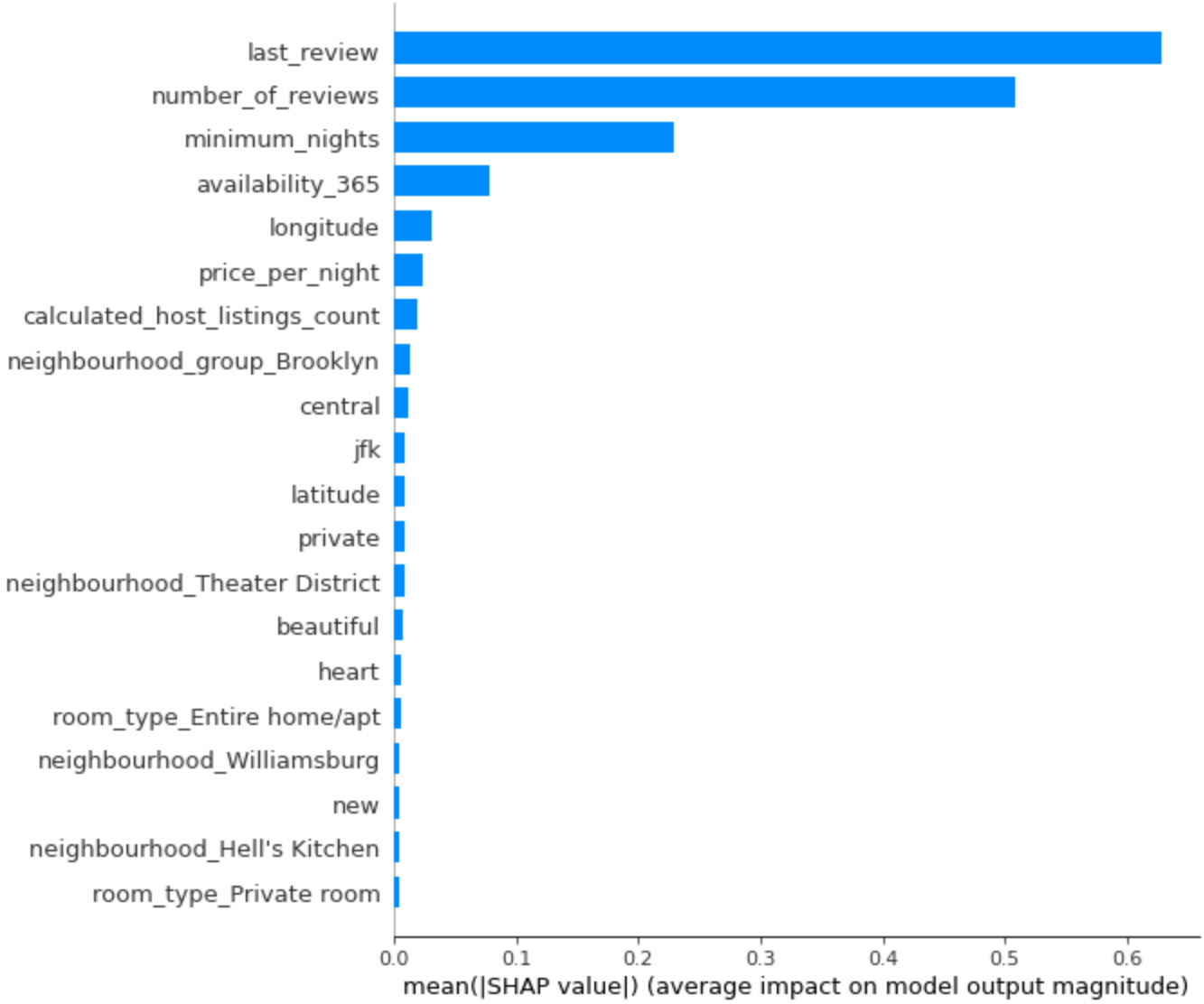
Out[43]:

| | number_of_reviews | minimum_nights | latitude | last_review | longitude | price_per_night | availability_ |
|---|---|---|---|---|---|---|---|
| **36503** | -0.458102 | 0.766157 | 1.478834 | 0.589008 | 0.067681 | 0.577024 | 0.512 |
| **27178** | 0.061270 | -0.123748 | 1.599180 | 0.564809 | 0.089092 | 0.167805 | -0.893 |
| **8678** | 0.580642 | -0.194940 | 1.757501 | -0.923446 | 0.060616 | -0.377821 | -0.90 |
| **8494** | -0.577957 | -0.159344 | -0.306915 | -2.912627 | -0.037019 | -0.361452 | -0.90 |
| **34164** | -0.298295 | -0.159344 | 0.197402 | 0.196980 | -1.066033 | 0.577024 | -0.90 |

5 rows × 2108 columns

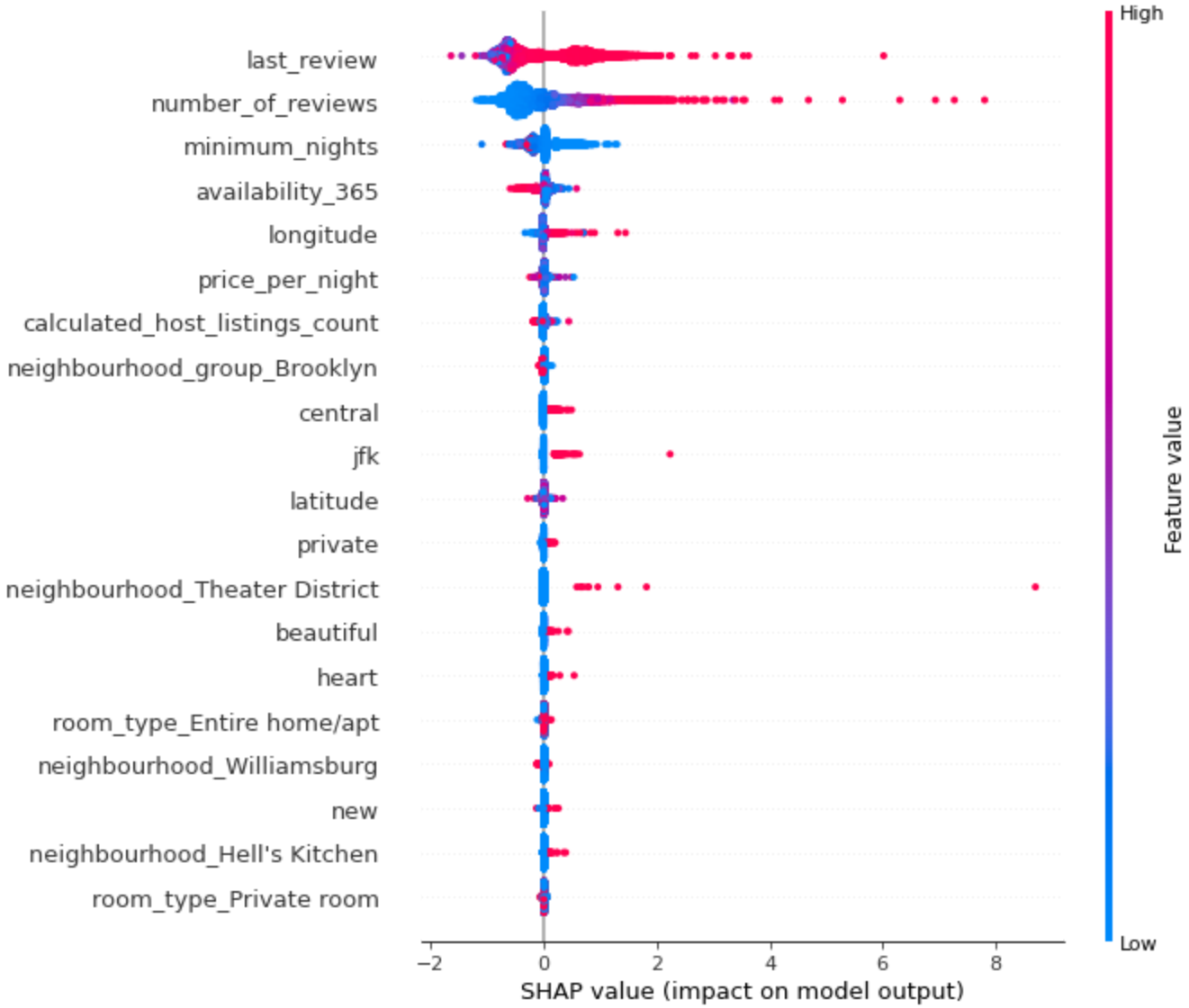
In [44]:
```
cb_explainer = shap.TreeExplainer(pipe_catboost.named_steps["catboostregressor"])
train_cb_shap_values = cb_explainer.shap_values(X_train_enc)
```

In [45]:
```
shap.summary_plot(train_cb_shap_values, X_train_enc, plot_type="bar")
```




In [46]:

```
shap.summary_plot(train_cb_shap_values, X_train_enc)
```




## 11. Feature importance results

- The top 3 most important features on the prediction are `last_review` and `number_of_reviews` and `minimum_nights` .
- They have the biggest SHAP value that would have the largest magnitude of impact on prediction of the popularity of listing with `reviews_per_month` .
- The higher the value of `last_review` and `number_of_reviews` would likely to drive the result to a positive SHAP value, meaning that there would be a higher number of `reviews_per_month` .
- Most of the values of `minimum_nights` feature are low. The shorter the duration of stay would have a larger positive SHAP value.

# 12. Results on the test set

---

rubric={accuracy:6,reasoning:4}

**Your tasks:**

1. Try your best performing model on the test data and report test scores.

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

In [47]:
```
pipe_catboost.score(X_test, y_test)
```

Out[47]:
```
0.6327417166428613
```

In [48]:
```
X_test_enc = pd.DataFrame(
    data=preprocessor.transform(X_test).toarray(),
    columns=new_columns,
    index=X_test.index,
)
X_test_enc.head()
```

Out[48]:

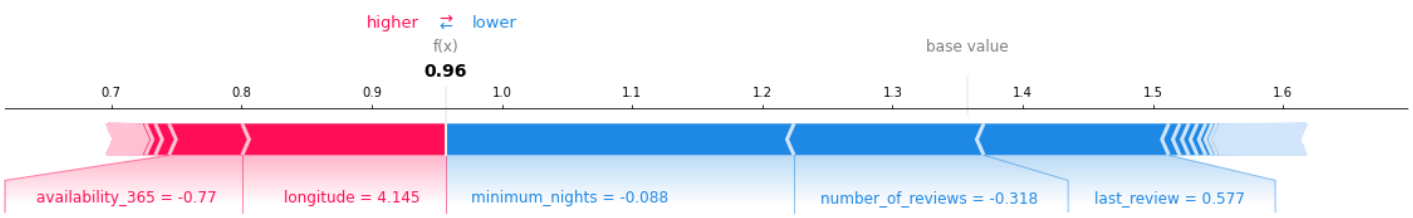
| | number_of_reviews | minimum_nights | latitude | last_review | longitude | price_per_night | availability |
|---|---|---|---|---|---|---|---|
| **38551** | -0.498053 | -0.088152 | 0.755288 | 0.647087 | -0.045155 | 0.713430 | -0.30 |
| **28522** | -0.557981 | -0.159344 | -1.336280 | -0.659674 | -0.110244 | -0.241414 | -0.16 |
| **35908** | 0.820352 | -0.194940 | -0.223260 | 0.668866 | 0.249889 | -0.214133 | -0.47 |
| **36508** | -0.538005 | -0.123748 | 1.790890 | 0.424453 | -0.056288 | -0.132289 | 1.37 |
| **16183** | -0.538005 | -0.088152 | -0.004949 | 0.601108 | -0.810813 | 0.195086 | -0.90 |

5 rows × 2108 columns

In [49]:
```
test_cb_shap_values = cb_explainer.shap_values(X_test_enc[:100])
```

In [50]:
```
X_train_enc = X_train_enc.round(3)
X_test_enc = X_test_enc.round(3)
```

In [51]:
```
shap.force_plot(
    cb_explainer.expected_value,
    test_cb_shap_values[20],
    X_test_enc.iloc[20, :],
    matplotlib=True,
)
```




In [52]:
```
y_test.iloc[20]
```

Out[52]:
```
0.53
```

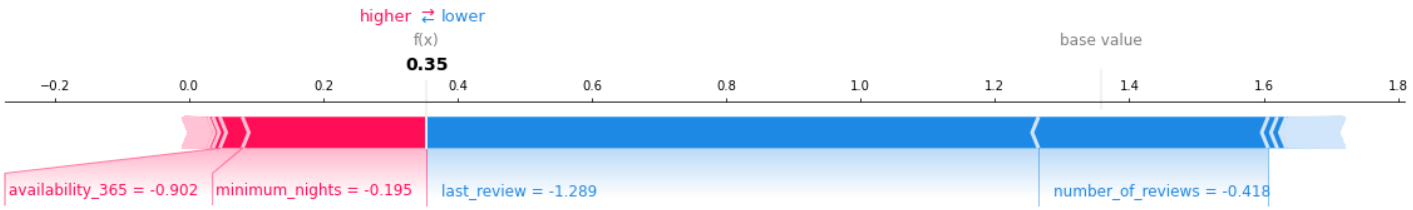
In [53]:
```
pd.DataFrame(X_test_enc.iloc[20,:]).T
```

Out[53]:

| | number_of_reviews | minimum_nights | latitude | last_review | longitude | price_per_night | availability_36 |
|---|---|---|---|---|---|---|---|
| **20862** | -0.318 | -0.088 | -2.444 | 0.577 | 4.145 | -0.241 | -0.7 |

1 rows × 2108 columns

In [54]:
```
shap.force_plot(
    cb_explainer.expected_value,
    test_cb_shap_values[15],
    X_test_enc.iloc[15, :],
    matplotlib=True,
)
```




In [55]:
```
y_test.iloc[15]
```

Out[55]:
```
0.31
```

## 12. Result of test set

12.2:

- The CatBoost test score is 0.63, which is similar to the validation $R^2$ score of 0.611 with default hyperparameters and 0.60 with tuned hyperparameters.
- There is no overfitting and so optimisation bias is not an issue.
- Hence, I would trust this test result as it is in line with the results of train set.

12.3:

- Prediction 1: The expected value of 20th example prediction is at 0.96 while the true popularity value is 0.53, which makes sense with the $R^2$ score at only 0.63. The features such as availability and longitude push the prediction towards a higher score, while the minimum nights, number of reviews and last reviews push the prediction towards a lower score.

- Prediction 2: The expected value of 15th example prediction is at 0.35 while the true popularity value is 0.31, which is quite close to the prediction. The features such as availability and minimum nights push the prediction towards a higher score, while the number of reviews and last reviews push the prediction towards a lower score.

# 13. Summary of results

---

rubric={reasoning:12}

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

**Your tasks:**

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

In [56]:
```
est_best = random_search_cb.best_params_['catboostregressor__n_estimators']
dep_best = random_search_cb.best_params_['catboostregressor__max_depth']
learn_best = random_search_cb.best_params_['catboostregressor__learning_rate']
```

In [57]:
```
pipe_catboost_tuned = make_pipeline(preprocessor,
                                    CatBoostRegressor(n_estimators=est_best,
                                                      max_depth=dep_best,
                                                      learning_rate=learn_best,
                                                      verbose=0,
                                                      random_state=123))
```

In [58]:
```
results["CatBoost_tuned"] = mean_std_cross_val_scores(
    pipe_catboost_tuned,
    X_train,
    y_train,
    return_train_score=True,
```

```
    scoring=scoring_metrics
    )
pd.DataFrame(results)
```

Out[58]:

| | Dummy | Ridge | Random Forest | XGBoost | LightGBM | CatBoost | CatBoost_tuned |
|---|---|---|---|---|---|---|---|
| **fit_time** | 0.001 (+/- 0.000) | 0.051 (+/- 0.006) | 8.482 (+/- 0.047) | 0.481 (+/- 0.033) | 0.399 (+/- 0.022) | 3.071 (+/- 0.081) | 0.908 (+/- 0.085) |
| **score_time** | 0.001 (+/- 0.000) | 0.010 (+/- 0.001) | 0.025 (+/- 0.001) | 0.015 (+/- 0.002) | 0.014 (+/- 0.001) | 0.013 (+/- 0.000) | 0.014 (+/- 0.000) |
| **test_neg RMSE** | -1.700 (+/- 0.169) | -1.404 (+/- 0.156) | -1.087 (+/- 0.165) | -1.106 (+/- 0.154) | -1.083 (+/- 0.137) | -1.063 (+/- 0.171) | -1.069 (+/- 0.162) |
| **train_neg RMSE** | -1.706 (+/- 0.043) | -0.954 (+/- 0.046) | -0.408 (+/- 0.018) | -0.511 (+/- 0.014) | -0.594 (+/- 0.019) | -0.647 (+/- 0.010) | -0.825 (+/- 0.011) |
| **test_r2** | -0.001 (+/- 0.001) | 0.315 (+/- 0.070) | 0.593 (+/- 0.042) | 0.578 (+/- 0.036) | 0.595 (+/- 0.034) | 0.611 (+/- 0.051) | 0.606 (+/- 0.046) |
| **train_r2** | 0.000 (+/- 0.000) | 0.687 (+/- 0.020) | 0.943 (+/- 0.003) | 0.910 (+/- 0.010) | 0.879 (+/- 0.003) | 0.856 (+/- 0.006) | 0.766 (+/- 0.007) |
| **test_mape** | -676.365 (+/- 45.614) | -336.259 (+/- 30.196) | -57.752 (+/- 4.537) | -87.139 (+/- 5.519) | -79.677 (+/- 4.872) | -82.616 (+/- 9.938) | -80.977 (+/- 5.549) |
| **train_mape** | -676.403 (+/- 13.640) | -233.005 (+/- 7.054) | -21.536 (+/- 0.625) | -55.935 (+/- 2.326) | -56.251 (+/- 0.960) | -63.314 (+/- 1.921) | -74.785 (+/- 2.863) |

## 13. Summary of results

13.1: Results of training different models are summarised in the above table.

13.2:

(i) Comparison of various models in train set:

- Across all models with default parameters, CatBoost has the highest $R^2$ score 0.611 and lowest RMSE -1.063. CatBoost also overfits less than all other models.
- Dummy Regressor and Ridge are not appropriate models with very low validation score and high errors (MAPE and RMSE). The MAPE errors are double of other models.
- The fit time for Random Forest and CatBoost with default parameters are much longer than other models. CatBoost with tuned hyperparameters is the third slowest and XGBoost and LightGBM have similar fit time that are faster.
- However, the score time are similar for the five models around 0.015 seconds. Even though Random Forest and CatBoost takes longer to train the model, the scoring time is short and the duration of scoring is as good as the other models. Although Dummy Regressor and Ridge are the fastest in fit time and score time, the scores are not promising.

(ii) Summary after hyperparameter tuning with various models: The new validation scores are similar to those with default parameters for every model. The same trend of results still applies, meaning that CatBoost is still the best performing model on test set.

(iii) Recommended model: Although overfitting between train and validation scores is less for the tuned CatBoost model, the $R^2$ validation score is lower than that with default parameters. Therefore, CatBoost

with default parameters is recommended as the prediction model.

(iv) Test set results: The test set $R^2$ score is in line with the validation set without overfitting. The sampled prediction results also echo the top 5 most important features as shown in the interpretation with SHAP plots.

(v) - The top 3 most important features on the prediction are `last_review` and `number_of_reviews` and `minimum_nights` based on SHAP value.

(vi) Conclusion: CatBoost with default parameters is recommended as the regression prediction model for the popularity of listing with the below reasons:

- It is the best performing model with the highest $R^2$ score in validation across all models;
- It has the lowest error (RMSE) across all models;
- No overfitting is observed between the test and validation results meaning the prediction would likely be accurate with no optimisation bias.

13.3: Other ideas to improve the performance would be doing some feature engineering by creating new features with the text feature of the listing names to further break down and extract additional information on whether there is any sentiment related factors to the name that could impact on popularity.

13.4: $R^2$ test score of the CatBoost model is 0.63.

# (Optional) 14. Creating a data analysis pipeline

rubric={reasoning:2}

**Your tasks:**

- In 522 you learned how build a reproducible data analysis pipeline. Convert this notebook into scripts and create a reproducible data analysis pipeline with appropriate documentation.

# (Optional) 15. Your takeaway from the course

---

rubric={reasoning:1}

**Your tasks:**

What is your biggest takeaway from this course?

- The biggest takeaway is learning the various models (linear and non-linear) and feature selection that could be leveraged to tackle different problems of predictions. Also, learning the interpretation of features allows me to understand the model.

**PLEASE READ BEFORE YOU SUBMIT:**

When you are ready to submit your assignment do the following:

1. Run all cells in your notebook to make sure there are no errors by doing `Kernel -> Restart Kernel and Clear All Outputs` and then `Run -> Run All Cells`.
2. Notebooks with cell execution numbers out of order or not starting from "1" will have marks deducted. Notebooks without the output displayed may not be graded at all (because we need to see the output in order to grade your work).
3. Push all your work to your GitHub lab repository.
4. Upload the assignment using Gradescope's drag and drop tool. Check out this Gradescope Student Guide if you need help with Gradescope submission.
5. Make sure that the plots and output are rendered properly in your submitted file. If the .ipynb file is too big and doesn't render on Gradescope, also upload a pdf or html in addition to the .ipynb so that the TAs can view your submission on Gradescope.

Well done!! Have a great weekend!

In [59]:
```
from IPython.display import Image

Image("eva-well-done.png")
```

Out[59]: