

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

FINAL TEST SCORE: $R^2 = 0.39$

Lab 4: Putting it all together in a mini project

For this lab, **you can choose to work alone or in a group of up to four students**. You are in charge of how you want to work and who you want to work with. Maybe you really want to go through all the steps of the ML process yourself or maybe you want to practice your collaboration skills, it is up to you! Just remember to indicate who your group members are (if any) when you submit on Gradescope. If you choose to work in a group, you only need to use one GitHub repo (you can create one on [github.ubc.ca](https://github.com) and set the visibility to "public").

Submission instructions

rubric={mechanics}

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

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

Points: 2

<https://github.com/nassim9/573-lab4-Airbnb-Popularity-NEWYORK.git>

Introduction

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

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

Tips

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

Assessment

We don't have some secret target score that you need to achieve to get a good grade.

You'll be assessed on demonstration of mastery of course topics, clear presentation, and the quality of your analysis and results. For example, if you just have a bunch of code and no text or figures, that's not good. If you instead do a bunch of sane things and you have clearly motivated your choices, but still get lower model performance than your friend, don't sweat it.

A final note

Finally, the style of this "project" 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 "several hours" but not "many hours" is a good guideline for a high quality submission. Of course if you're having fun you're welcome to spend as much time as you want! But, if so, try not to do it out of perfectionism or getting the best possible grade. Do it because you're learning and enjoying it. Students from the past cohorts have found such kind of labs useful and fun and we hope you enjoy it as well.

1. Pick your problem and explain the prediction problem

rubric={reasoning}

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

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

OR

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

Your tasks:

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

Points: 3

New York City Airbnb Listings Problem

1. Initially, this problem seems like it could be challenging. There are potential features with collinearity that need to be considered, along with related categorical columns like `neighbourhood_group` and `neighbourhood`. Additionally, there are several null values in the target column to explore further.

```
In [ ]: #imports
import sklearn
import numpy as np
import pandas as pd
import shap
import matplotlib.pyplot as plt
```

```

import seaborn as sns
import altair_ally as aly
import altair as alt
from xgboost import XGBRegressor
from sklearn.feature_selection import RFECV
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_validate

from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import (
    OneHotEncoder,
    StandardScaler,
)
from sklearn.compose import ColumnTransformer
from sklearn.dummy import DummyRegressor
from sklearn.pipeline import make_pipeline

from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import Lasso
from sklearn.ensemble import StackingRegressor
from sklearn.svm import SVR

```

```

In [ ]: data= pd.read_csv('data/AB_NYC_2019.csv')

data.head()

```

```

Out[ ]:

```

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

2. Data splitting

rubric={reasoning}

Your tasks:

1. Split the data into train and test portions.

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

Points: 1

Given the large size of the dataset, only 60% of the dataset is being used for training the model while the testing is being done on the remaining 40%.

```
In [ ]: train_df, test_df = train_test_split(data, test_size = 0.4, random_state=123)
        train_df.head()
```

```
Out [ ]:
```

	id	name	host_id	host_name	neighbourhood_group	neighbour
--	----	------	---------	-----------	---------------------	-----------

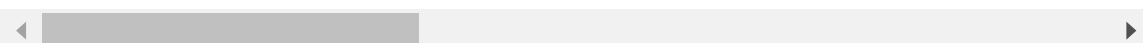
17877	14010200	Scandinavean design in Crown Heights, BK	1683437	Boram	Brooklyn	Crown H
--------------	----------	--	---------	-------	----------	---------

14638	11563821	Private bedroom located in the heart of Chelsea	10307134	Anna	Manhattan	C
--------------	----------	---	----------	------	-----------	---

7479	5579629	Lovely sunlit room in Brooklyn	329917	Clémentine	Brooklyn	Green
-------------	---------	--------------------------------	--------	------------	----------	-------

47058	35575853	Great view, 1 BR right next to Central Park!	35965489	Meygan	Manhattan	East H
--------------	----------	--	----------	--------	-----------	--------

9769	7509362	Great BIG Upper West Side Apartment	29156329	Andrew	Manhattan	Upper
-------------	---------	-------------------------------------	----------	--------	-----------	-------

**3. EDA**

rubric={viz,reasoning}

Perform exploratory data analysis on the train set.

Your tasks:

1. Include at least two summary statistics and two visualizations that you find useful, and accompany each one with a sentence explaining it.

2. Summarize your initial observations about the data.
3. Pick appropriate metric/metrics for assessment.

Points: 6

Summary Statistics:

The `describe` function provides key statistics including mean, min, max, and standard deviation, which provide an overview of the central tendencies of the data, as well the spread. Also, the `nunique` function helps identify the features which are unique identifiers such as id, host id, and name, which do not help the model in learning any patterns.

Visualizations:

The distribution of the numerical features is helpful to understand the spread of the features and if there is any skewness in any of them. Also, the correlation of the numerical features help identify if there are any highly correlated features which are to be dealt with accordingly.

Metric:

For scoring models we will primarily use R^2 since we will be comparing several regression models. Additional metrics, described in the later questions, will be used for feature importance and further analysis.

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 29337 entries, 17877 to 15725
Data columns (total 16 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     29337 non-null  int64
1   name                                  29328 non-null  object
2   host_id                               29337 non-null  int64
3   host_name                             29325 non-null  object
4   neighbourhood_group                   29337 non-null  object
5   neighbourhood                         29337 non-null  object
6   latitude                             29337 non-null  float64
7   longitude                             29337 non-null  float64
8   room_type                             29337 non-null  object
9   price                                 29337 non-null  int64
10  minimum_nights                        29337 non-null  int64
11  number_of_reviews                     29337 non-null  int64
12  last_review                           23386 non-null  object
13  reviews_per_month                     23386 non-null  float64
14  calculated_host_listings_count        29337 non-null  int64
15  availability_365                       29337 non-null  int64
dtypes: float64(3), int64(7), object(6)
memory usage: 3.8+ MB
```

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

Out[]:

	count	mean	std	min	
id	29337.0	1.891988e+07	1.102155e+07	2539.00000	9.3507
host_id	29337.0	6.714579e+07	7.835404e+07	2438.00000	7.7401
latitude	29337.0	4.072901e+01	5.459419e-02	40.50641	4.0690
longitude	29337.0	-7.395222e+01	4.609134e-02	-74.24442	-7.3983
price	29337.0	1.509391e+02	2.282242e+02	0.00000	6.9000
minimum_nights	29337.0	7.141971e+00	2.227211e+01	1.00000	1.0000
number_of_reviews	29337.0	2.335450e+01	4.469248e+01	0.00000	1.0000
reviews_per_month	23386.0	1.369867e+00	1.706732e+00	0.01000	1.9000
calculated_host_listings_count	29337.0	7.003340e+00	3.251162e+01	1.00000	1.0000
availability_365	29337.0	1.128036e+02	1.315445e+02	0.00000	0.0000

In []:

```
def missing_zero_values_table(df):

    zero_val = (df == 0.00).astype(int).sum(axis=0)
    mis_val = df.isnull().sum()
    mis_val_percent = 100 * df.isnull().sum() / len(df)
    mz_table = pd.concat([zero_val, mis_val, mis_val_percent], axis=1)
    mz_table = mz_table.rename(
        columns = {0 : 'Zero Values', 1 : 'Missing Values', 2 : '% of Total Values'})
    mz_table['Total Zero Missing Values'] = mz_table['Zero Values'] + mz_table['
    mz_table['% Total Zero Missing Values'] = 100 * mz_table['Total Zero Missing
    mz_table['Data Type'] = df.dtypes
    mz_table = mz_table[
        mz_table.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
    print ("Your selected dataframe has " + str(df.shape[1]) + " columns and " +
          "There are " + str(mz_table.shape[0]) +
          " columns that have missing values.")
    return mz_table
missing_zero_values_table(train_df)
```

Your selected dataframe has 16 columns and 29337 Rows.
There are 4 columns that have missing values.

Out[]:

	Zero Values	Missing Values	% of Total Values	Total Zero Missing Values	% Total Zero Missing Values	Data Type
last_review	0	5951	20.3	5951	20.3	object
reviews_per_month	0	5951	20.3	5951	20.3	float64
host_name	0	12	0.0	12	0.0	object
name	0	9	0.0	9	0.0	object

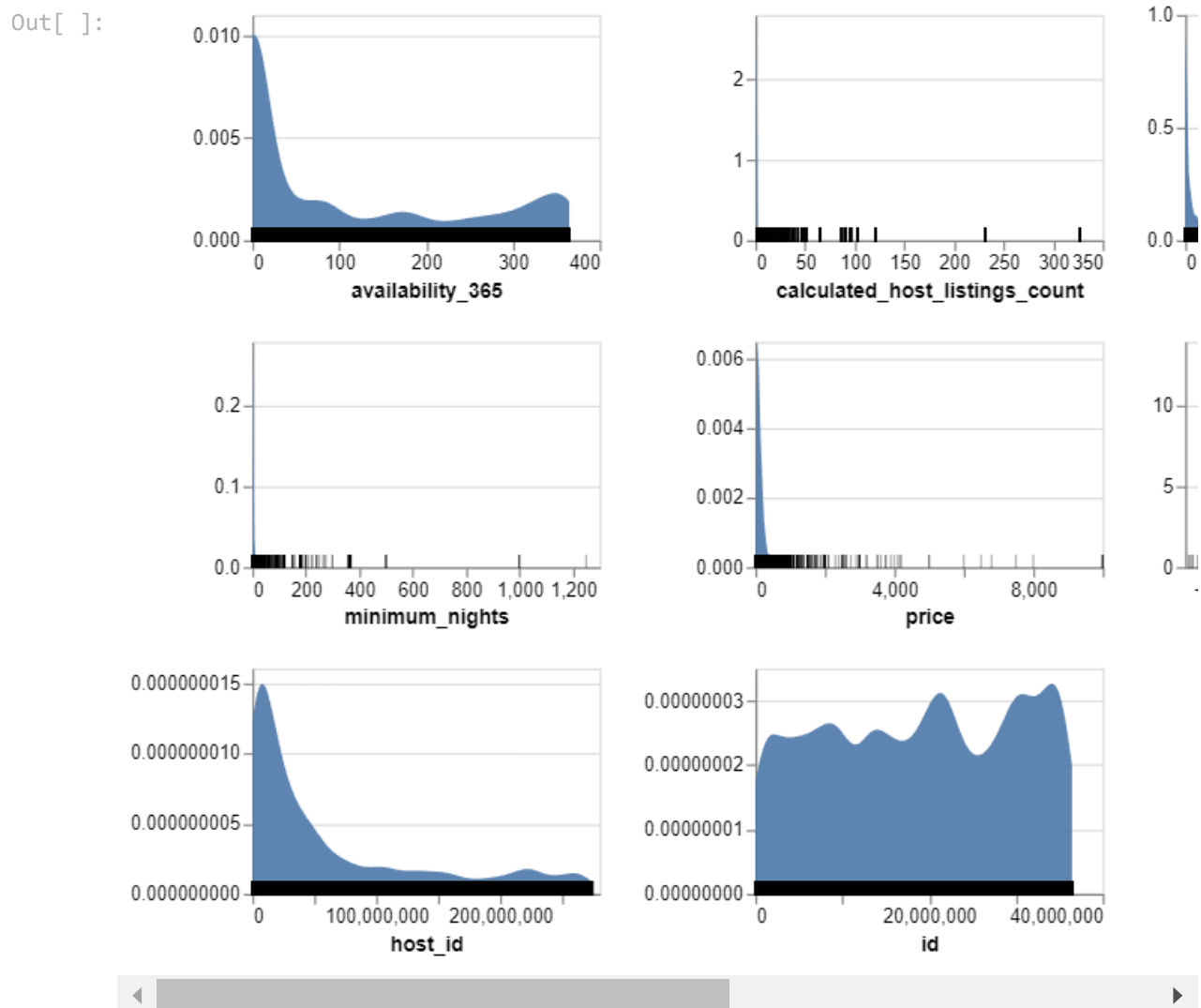
In []:

```
train_df.nunique()
```

```
Out[ ]: id                29337
        name              28894
        host_id           23964
        host_name         8375
        neighbourhood_group 5
        neighbourhood      217
        latitude           15269
        longitude          12151
        room_type           3
        price              591
        minimum_nights      94
        number_of_reviews   360
        last_review         1632
        reviews_per_month   867
        calculated_host_listings_count 47
        availability_365     366
        dtype: int64
```

```
In [ ]: # Distribution of Numerical features
        aly.alt.data_transformers.enable('vegafusion')

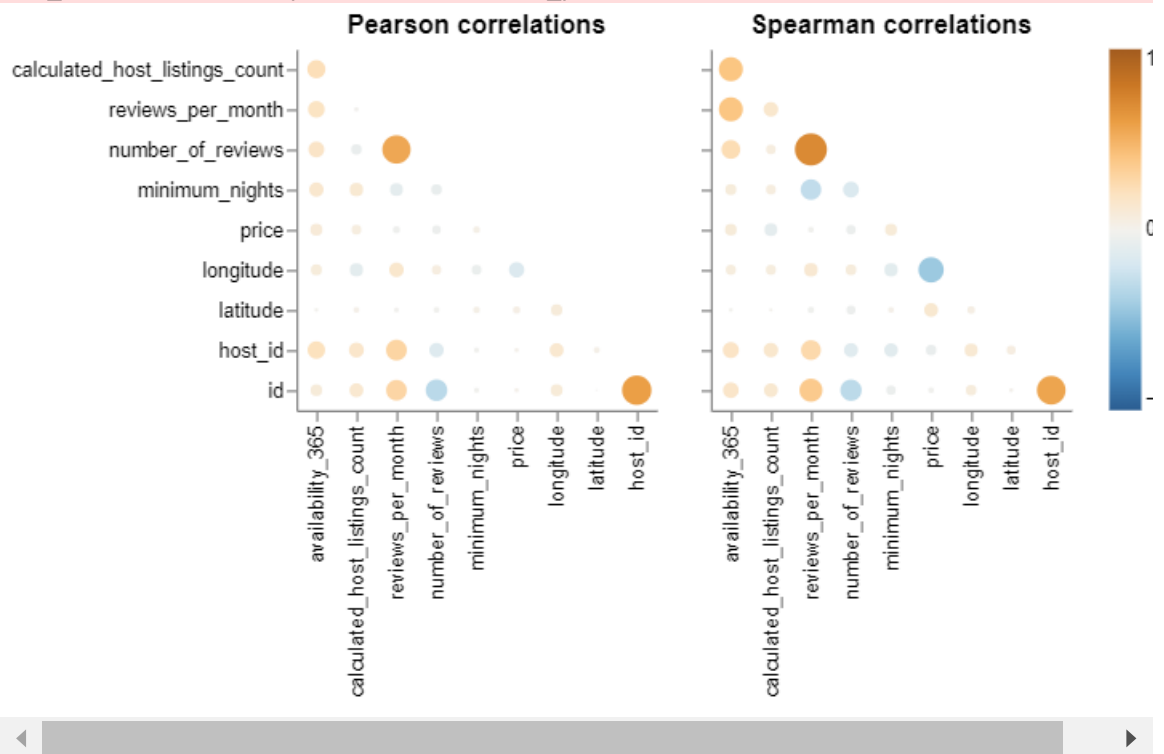
        aly.dist(train_df)
```



```
In [ ]: aly.corr(train_df)
```


'selection_multi' is deprecated. Use 'selection_point'
 The value of 'empty' should be True or False.
 'add_selection' is deprecated. Use 'add_params' instead.
 'add_selection' is deprecated. Use 'add_params' instead.

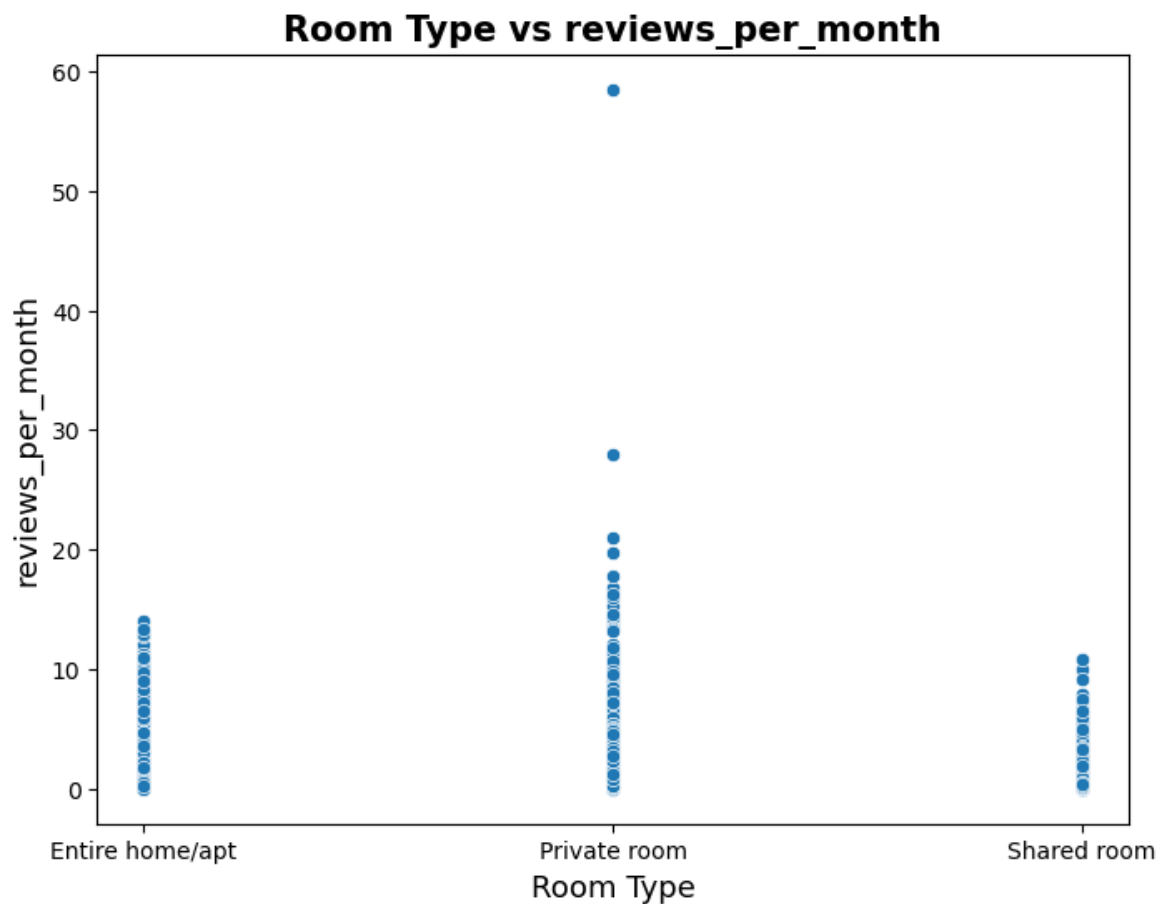
Out[]:



```
In [ ]: plt.figure(figsize=(8,6))
sns.scatterplot(x='room_type', y='reviews_per_month', data=train_df)

plt.xlabel("Room Type", size=13)
plt.ylabel("reviews_per_month", size=13)
plt.title("Room Type vs reviews_per_month",size=15, weight='bold')
```

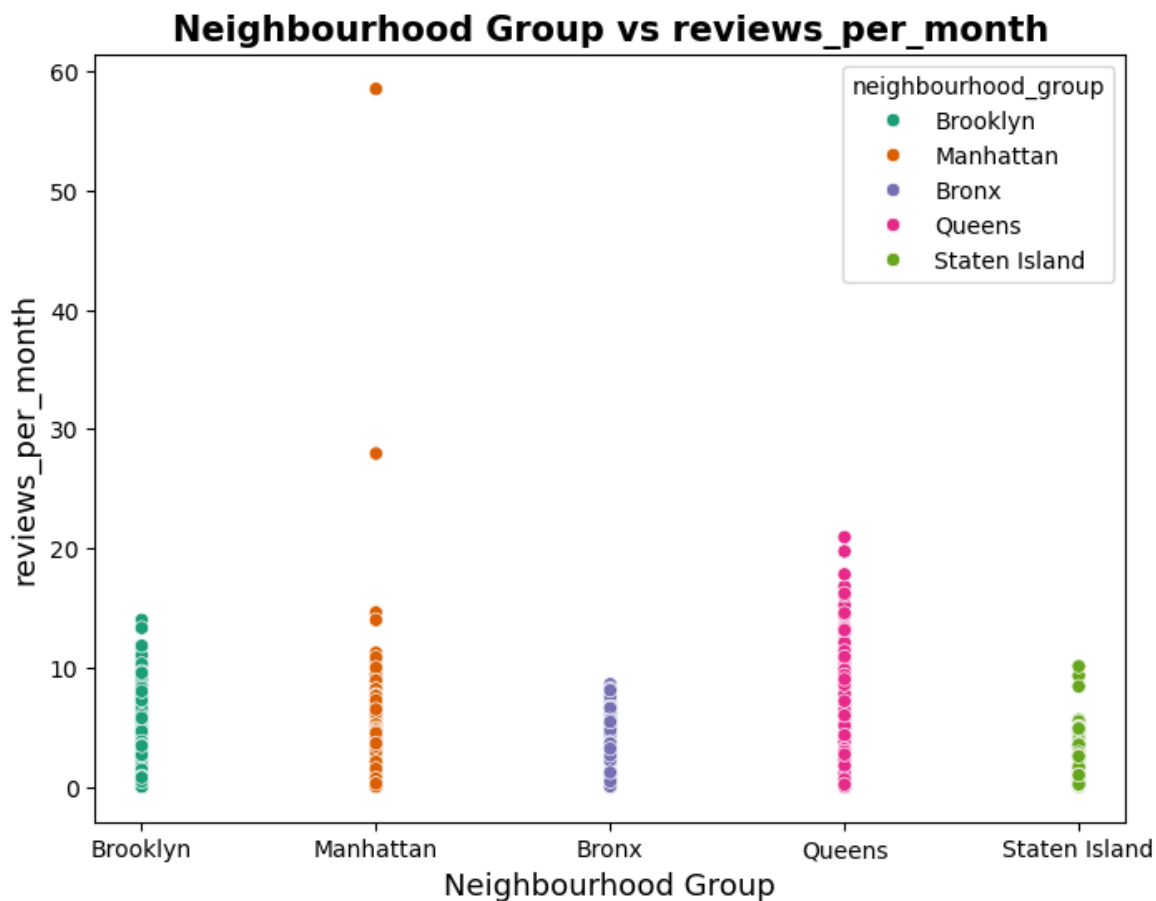
Out[]: Text(0.5, 1.0, 'Room Type vs reviews_per_month')



```
In [ ]: plt.figure(figsize=(8,6))
sns.scatterplot(x='neighbourhood_group', y='reviews_per_month', data=train_df, h

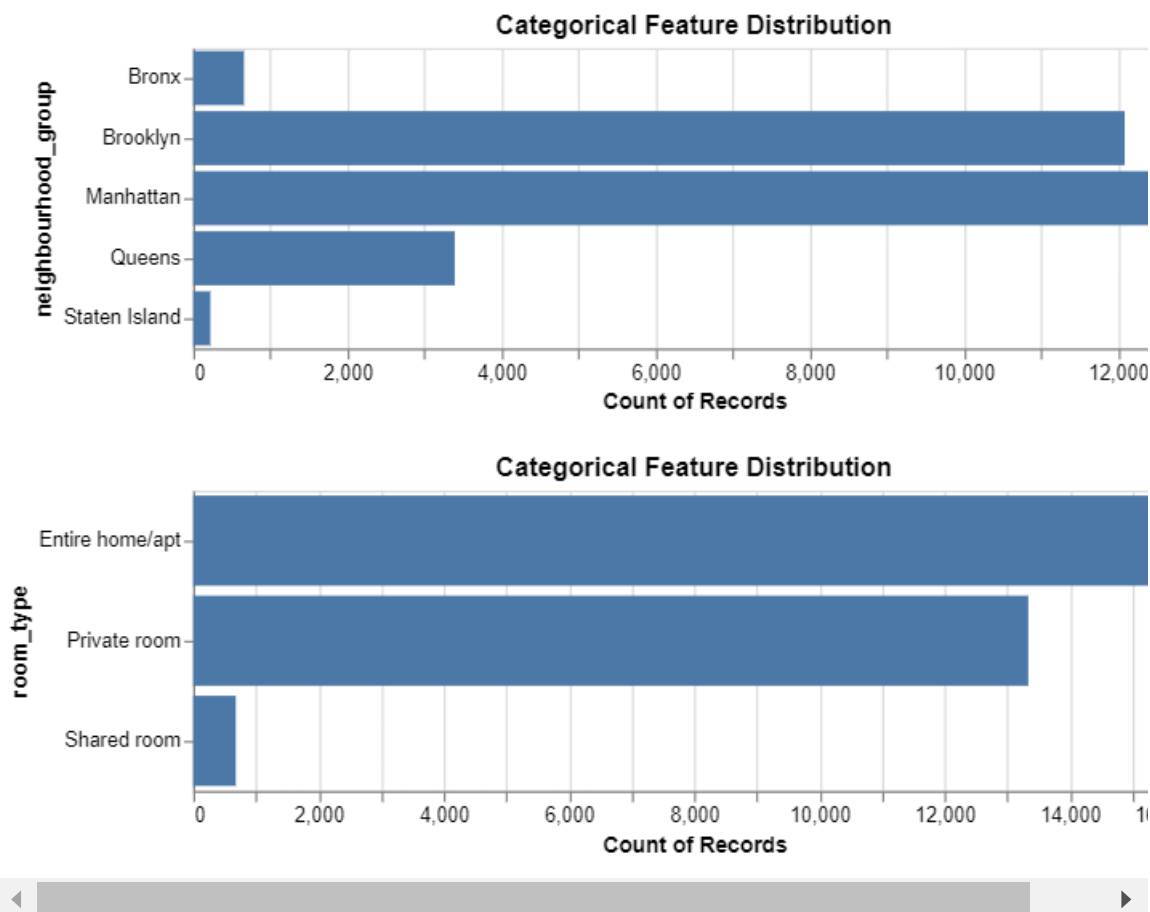
plt.xlabel("Neighbourhood Group", size=13)
plt.ylabel("reviews_per_month", size=13)
plt.title("Neighbourhood Group vs reviews_per_month",size=15, weight='bold')
```

```
Out[ ]: Text(0.5, 1.0, 'Neighbourhood Group vs reviews_per_month')
```



```
In [ ]: alt.data_transformers.enable('vegafusion')
        categ_cols = ['neighbourhood_group', 'room_type']
        categ_cols_dist = alt.Chart(train_df, title = "Categorical Feature Distribution"
                                     y = alt.Y(alt.repeat(), type = "nominal"),
                                     x = "count()",
        ).properties(
            width = 500,
            height = 150
        ).repeat(
            categ_cols,
            columns = 1
        )
        categ_cols_dist
```

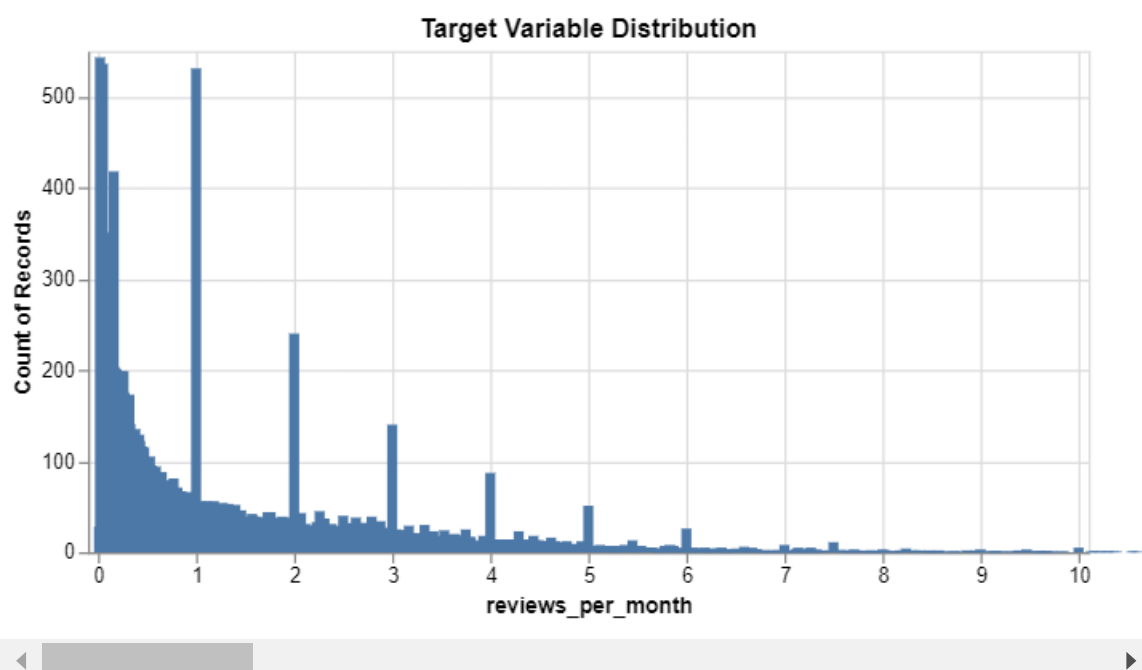
Out[]:



In []:

```
# Distribution of the Target
last_review_plot = alt.Chart(train_df, title = "Target Variable Distribution").mark_bar(
    x = alt.X('reviews_per_month').scale(domain = [0,10]),
    y = alt.Y("count()")
).properties(
    width = 500,
    height = 250
)
last_review_plot
```

Out[]:



It is clearly visible from the above that the target variable seems to be right skewed.

4. Feature engineering (Challenging)

rubric={reasoning}

Your tasks:

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

Points: 0.5

After experimenting with various models using our initial features (including the `numeric_features` and `categorical_features` identified below) we decided to use polynomial feature engineering. Our suspicion was that the relationship between our numeric features and the target variable might not be adequately explained by a linear model. We performed polynomial feature engineering up to degree 2, aiming to capture more complex relationships in the data, while staying within our computational limitations.

5. Preprocessing and transformations

rubric={accuracy,reasoning}

Your tasks:

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

Points: 4

Dropped Features & Reasoning:

- `id`, `name`, `host_id`, `host_name` : all unique identifiers
- `last_review` : dropped since several null values, alternatives like, binning weren't useful.
- `neighbourhood` : We tried out models with both `neighbourhood_group` and `neighbourhood`, and compared it with dropping either one of them. Dropping `neighbourhood` had the best results. Additionally, there are 4 features all related to geographical location, so it makes sense to at least drop 1.

Categorical Features & Reasoning:

- `neighbourhood_group` and `room_type` : one-hot-encoder, with `handle_unknown=True` since both contain nominal values.

Numeric Features & Reasoning:

- `latitude`, `longitude`, `price`, `minimum_nights`, `calculated_host_listing_count`, `availability_365` : polynomial feature engineering, degree 2, to capture more complex relationships (compared to linear), followed by standard scaler, so all features are on the same scale.

```
In [ ]: train_df['reviews_per_month'] = train_df['reviews_per_month'].fillna(0)
test_df['reviews_per_month'] = test_df['reviews_per_month'].fillna(0)

X_train = train_df.drop(columns=['reviews_per_month'])
y_train = train_df['reviews_per_month']
X_test = test_df.drop(columns=['reviews_per_month'])
y_test = test_df['reviews_per_month']

categorical_features = ['room_type',
                        'neighbourhood_group']
numeric_features = ['latitude',
                    'longitude',
                    'price',
                    'minimum_nights',
                    'calculated_host_listings_count',
                    'availability_365']
drop_features = ['last_review',
                 'id',
                 'name',
                 'host_id',
                 'host_name',
                 'neighbourhood']
```

```
In [ ]: numeric_transformer = make_pipeline(
    PolynomialFeatures(degree=2),
    StandardScaler()
)

categorical_transformer = make_pipeline(
    OneHotEncoder(handle_unknown='ignore')
)

preprocessor = make_column_transformer(
    (numeric_transformer, numeric_features),
    (categorical_transformer, categorical_features),
    ('drop', drop_features)
)
```

6. Baseline model

rubric={accuracy}

Your tasks:

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

Points: 2

We have considered DummyRegressor as our baseline model.

```
In [ ]: model_dummy = DummyRegressor()

pipe_dummy = make_pipeline(preprocessor, model_dummy)

cv_df = pd.DataFrame(cross_validate(pipe_dummy,
                                   X_train,
                                   y_train,
                                   return_train_score=True,
                                   scoring='r2'
                                ))
cv_df
```

```
Out[ ]:   fit_time  score_time  test_score  train_score
0  0.030595   0.006721 -2.209298e-04         0.0
1  0.029098   0.006001 -1.822243e-04         0.0
2  0.028172   0.006577 -1.327336e-04         0.0
3  0.027919   0.007088 -1.944606e-07         0.0
4  0.020099   0.006742 -3.851852e-04         0.0
```

7. Linear models

rubric={accuracy,reasoning}

Your tasks:

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

Points: 8

Summary:

We tried out Ridge as our first linear model which had poor performance. The train and test(validation) R^2 scores were 0.164 and 0.161, respectively. This indicates that this model is not performing much better than the dummy model that simply predicts the mean. The standard deviation for the test score, 0.018, is slightly larger than the train

score standard deviation of 0.002. After performing hyperparameter tuning on alpha, the best value of alpha is 0.001. This still results in a best score of approximately 0.162. We further explore other models to seek better results.

```
In [ ]: # 1. Linear Model first attempt
pipe_ridge = make_pipeline(preprocessor, Ridge())

cv_df = pd.DataFrame(cross_validate(pipe_ridge, X_train, y_train, scoring='r2',
cv_df

cross_val_results = {}
cross_val_results['ridge'] = cv_df.agg(['mean', 'std']).round(3).T
cross_val_results['ridge']
```

```
Out[ ]:
```

	mean	std
fit_time	0.037	0.007
score_time	0.002	0.003
test_score	0.161	0.018
train_score	0.164	0.002

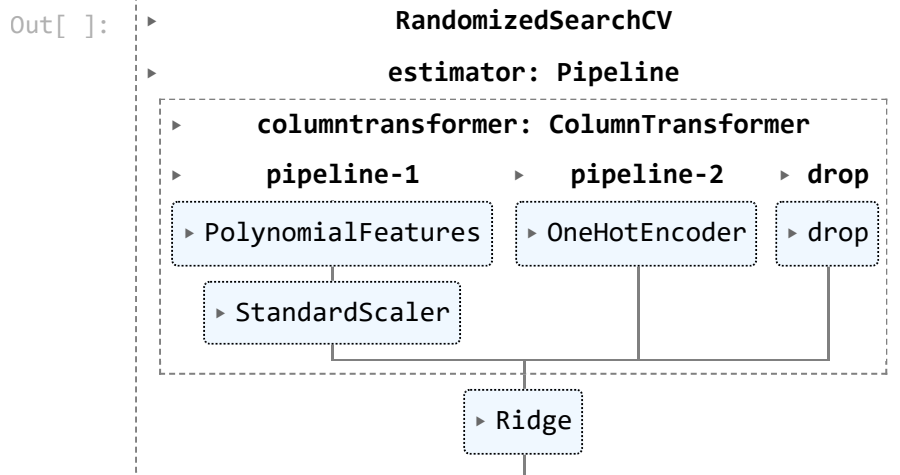
```
In [ ]: # 2. Hyperparameter tuning

param_dist = {"ridge__alpha": 10.0**np.arange(-5, 5) }

random_search = RandomizedSearchCV(
    pipe_ridge,
    param_dist,
    n_iter=100,
    return_train_score=True,
    random_state=123,
    n_jobs=-1,
    scoring='r2',
    cv=5
)

random_search.fit(X_train, y_train)
```

The total space of parameters 10 is smaller than n_iter=100. Running 10 iterations. For exhaustive searches, use GridSearchCV.




```
In [ ]: best_params = random_search.best_params_  
best_r2 = random_search.best_score_  
  
print("Best alpha:", best_params)  
print("best score:", best_r2)
```

```
Best alpha: {'ridge__alpha': 0.001}  
best score: 0.16155765643948067
```

8. Different models

rubric={accuracy,reasoning}

Your tasks:

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

Points: 10

Along with the Linear Ridge model, we tried out: Random Forest Regressor (RFR), XGBoost Regressor, LightGBM (LGBM) Regressor, and Stacking with SVR and Lasso as base estimators and RFR as the final estimator.

(Note: the summary table of all results is at the end of this question)

Timing:

- Fit time: Ridge is the fastest to fit, with a mean of 0.037 seconds, while the Stacked model took the longest, with a mean of 126.130 seconds. This is consistent with the scoring times.
- Score time: Ridge is the quickest to score, with a mean of 0.005 seconds. While the Stacked model took the longest, with a mean of 7.789 seconds.
- (Note - The values might differ based on the system on which the model is run.)

Test Scores:

- Test Score: LGBM has the highest mean test score of 0.374. Ridge has the lowest mean test score of 0.161.

Train Scores & Overfitting:

- Train score: RFR has the highest mean training score (0.506), suggesting its tending towards overfitting, especially compared to its test score of 0.344.
- Overfitting: Along with RFR and XGBoost also shows signs of overfitting, with training scores considerably higher than test scores.
- Underfitting: Ridge and the Stacking model both appear to be underfitting. Their mean train and test scores are both very low, with a small gap.

Comparison to Linear Model (Ridge):

Overall, Ridge is very fast compared to the other models, when considering both fit and score times. That said, it was the lowest scoring model on both train and test scores of all models. Notably, LGBM has a slightly longer fit time (0.229 seconds), but outperforms Ridge with a test score of 0.374, compared to 0.161. LGBM is the top model in terms of performance.

RANDOM FOREST REGRESSOR

```
In [ ]: from sklearn.ensemble import RandomForestRegressor

pipe_rf = make_pipeline(
    preprocessor,
    RandomForestRegressor(max_depth=10, random_state=123)
)

cv_rf = pd.DataFrame(cross_validate(pipe_rf, X_train, y_train, cv = 5, return_train_score=True))
```

```
Out [ ]:
```

	fit_time	score_time	test_score	train_score
0	34.760748	0.033602	0.331346	0.504834
1	34.400973	0.052438	0.306457	0.501309
2	34.577586	0.050025	0.381432	0.508694
3	34.609401	0.034490	0.358352	0.506464
4	35.179457	0.046562	0.341476	0.506866

```
In [ ]: cross_val_results['randomforest'] = cv_rf.agg(['mean', 'std']).round(3).T
cross_val_results['randomforest']
```

```
Out [ ]:
```

	mean	std
fit_time	34.706	0.294
score_time	0.043	0.009
test_score	0.344	0.028
train_score	0.506	0.003

```
In [ ]: pipe_rf.fit(X_train, y_train)
model = pipe_rf.named_steps['randomforestregressor']
```

```
In [ ]: feature_importances_rf = model.feature_importances_
feature_importances_rf
feature_names_1 = pipe_rf.named_steps['columntransformer'].named_transformers_['
feature_names_2 = pipe_rf.named_steps['columntransformer'].named_transformers_['

feature_names = feature_names_1 + feature_names_2
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': fe
```

```
feature_importances_rf_sorted = feature_importance_df.sort_values(by='Importance')
feature_importances_rf_sorted
```

Out[]:

	Feature	Importance
15	longitude minimum_nights	0.198454
21	price availability_365	0.186377
10	latitude minimum_nights	0.111016
17	longitude availability_365	0.073399
12	latitude availability_365	0.073268
8	latitude longitude	0.042744
16	longitude calculated_host_listings_count	0.039327
20	price calculated_host_listings_count	0.030094
11	latitude calculated_host_listings_count	0.028987
26	calculated_host_listings_count availability_365	0.028479

XGBOOST REGRESSOR

In []:

```
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
pipe_xgb = make_pipeline(
    preprocessor,
    XGBRegressor(
        random_state=123, verbosity=0))
cv_df_xg = pd.DataFrame(cross_validate(pipe_xgb, X_train, y_train, cv = 5, return_df=True))
```

Out[]:

	fit_time	score_time	test_score	train_score
0	0.923379	0.006506	0.332493	0.692865
1	1.063242	0.015623	0.302324	0.687489
2	0.908994	0.000000	0.370466	0.699875
3	0.891119	0.000000	0.347683	0.704054
4	0.898113	0.009004	0.365206	0.692671

In []:

```
cross_val_results['XGBoost'] = cv_df_xg.agg(['mean', 'std']).round(3).T
cross_val_results['XGBoost']
```

Out[]:

	mean	std
fit_time	0.937	0.072
score_time	0.006	0.007
test_score	0.344	0.028
train_score	0.695	0.007

```
In [ ]: pipe_xgb.fit(X_train, y_train)
model = pipe_xgb.named_steps['xgbregressor']
```

```
In [ ]: feature_importances_xgb = model.feature_importances_
feature_importances_xgb
feature_names_1 = pipe_xgb.named_steps['columntransformer'].named_transformers_
feature_names_2 = pipe_xgb.named_steps['columntransformer'].named_transformers_

feature_names = feature_names_1 + feature_names_2
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': fe
feature_importances_xgb_sorted = feature_importance_df.sort_values(by='Importanc
feature_importances_xgb_sorted
```

```
Out[ ]:
```

	Feature	Importance
17	longitude availability_365	0.180704
21	price availability_365	0.114236
15	longitude minimum_nights	0.089334
28	room_type_Entire home/apt	0.065287
10	latitude minimum_nights	0.057335
12	latitude availability_365	0.043035
19	price minimum_nights	0.037120
26	calculated_host_listings_count availability_365	0.035704
24	minimum_nights availability_365	0.032444
33	neighbourhood_group_Manhattan	0.031137

LIGHTGBM REGRESSOR

```
In [ ]: import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
from lightgbm.sklearn import LGBMRegressor
pipe_lgbm = make_pipeline(
    preprocessor, LGBMRegressor(random_state=123)
)
cv_df_lgbm = pd.DataFrame(cross_validate(pipe_lgbm, X_train, y_train, cv = 5, re
cv_df_lgbm
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.002186 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5973
[LightGBM] [Info] Number of data points in the train set: 23469, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 1.087050
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.001754 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5967
[LightGBM] [Info] Number of data points in the train set: 23469, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 1.087324
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.001365 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5970
[LightGBM] [Info] Number of data points in the train set: 23470, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 1.095608
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.001812 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5964
[LightGBM] [Info] Number of data points in the train set: 23470, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 1.091851
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.000998 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 5976
[LightGBM] [Info] Number of data points in the train set: 23470, number of used f
eatures: 35
[LightGBM] [Info] Start training from score 1.098115
```

Out[]:

	fit_time	score_time	test_score	train_score
--	----------	------------	------------	-------------

0	0.278799	0.015544	0.392953	0.523223
---	----------	----------	----------	----------

1	0.161391	0.016121	0.316021	0.542603
---	----------	----------	----------	----------

2	0.141537	0.016158	0.405498	0.531330
---	----------	----------	----------	----------

3	0.142247	0.016209	0.384006	0.534224
---	----------	----------	----------	----------

4	0.126260	0.015626	0.372930	0.534002
---	----------	----------	----------	----------

```
In [ ]: cross_val_results['LGBMRegressor'] = cv_df_lgbm.agg(['mean', 'std']).round(3).T
cross_val_results['LGBMRegressor']
```

Out[]:

	mean	std
--	------	-----

fit_time	0.170	0.062
----------	-------	-------

score_time	0.016	0.000
------------	-------	-------

test_score	0.374	0.035
------------	-------	-------

train_score	0.533	0.007
-------------	-------	-------

```
In [ ]: pipe_lgbm.fit(X_train, y_train)
        model_lgbm = pipe_lgbm.named_steps['lgbmregressor']
```

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.002067 seconds.

You can set `force_col_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 5988

[LightGBM] [Info] Number of data points in the train set: 29337, number of used features: 35

[LightGBM] [Info] Start training from score 1.091990

```
In [ ]: feature_importances_lgbm = model_lgbm.feature_importances_
        feature_names_1 = pipe_lgbm.named_steps['columntransformer'].named_transformers_
        feature_names_2 = pipe_lgbm.named_steps['columntransformer'].named_transformers_

        feature_names = feature_names_1 + feature_names_2
        feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': fe
        feature_importances_lgbm_sorted = feature_importance_df.sort_values(by='Importan
        feature_importances_lgbm_sorted
```

Out[]:

	Feature	Importance
8	latitude longitude	220
26	calculated_host_listings_count availability_365	200
15	longitude minimum_nights	184
2	longitude	181
19	price minimum_nights	181
20	price calculated_host_listings_count	180
6	availability_365	177
16	longitude calculated_host_listings_count	165
1	latitude	156
10	latitude minimum_nights	154

STACKING (SVR, LASSO & RFR)

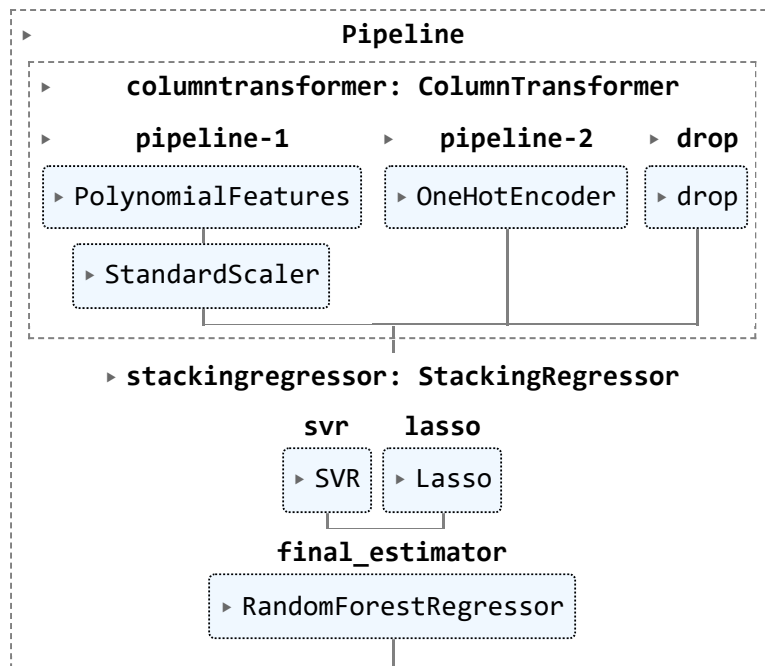
```
In [ ]: estimators = [ ('svr', SVR(kernel='rbf')), ('lasso', Lasso())]

        model_stack = StackingRegressor(estimators=estimators,
                                         final_estimator=(RandomForestRegressor(max_depth=5)))

        pipe_stack = make_pipeline(preprocessor, model_stack)

        pipe_stack.fit(X_train, y_train)
```

Out[]:



```
In [ ]: cv_df_stack = pd.DataFrame(cross_validate(pipe_stack,
                                                X_train,
                                                y_train,
                                                return_train_score=True,
                                                scoring='r2'))
```

```
In [ ]: cross_val_results['svr_lasso_rfr_stack'] = cv_df_stack.agg(['mean', 'std']).round(2)
cross_val_results['svr_lasso_rfr_stack']
```

Out[]:

	mean	std
fit_time	129.229	23.389
score_time	8.405	0.205
test_score	0.242	0.021
train_score	0.250	0.005

```
In [ ]: pd.concat(cross_val_results, axis = 1)
```

Out[]:

	ridge		randomforest		XGBoost		LGBMRegressor		svr_lasso_rfr_stac	
	mean	std	mean	std	mean	std	mean	std	mean	std
fit_time	0.037	0.007	34.706	0.294	0.937	0.072	0.170	0.062	129.229	23.38
score_time	0.002	0.003	0.043	0.009	0.006	0.007	0.016	0.000	8.405	0.20
test_score	0.161	0.018	0.344	0.028	0.344	0.028	0.374	0.035	0.242	0.02
train_score	0.164	0.002	0.506	0.003	0.695	0.007	0.533	0.007	0.250	0.00

9. Feature selection (Challenging)

```
rubric={reasoning}
```

Your tasks:

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

Points: 0.5

(Note: The results summary table is included at the end of this question)

```
In [ ]: import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.simplefilter(action="ignore", category=DeprecationWarning)
warnings.simplefilter(action="ignore", category=UserWarning)

from sklearn.feature_selection import RFECV
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
rfecv = RFECV(Ridge())

pipe_rf_rfecv = make_pipeline(
    preprocessor, rfecv, LGBMRegressor(random_state=123)
)
cv_df_rfecv = pd.DataFrame(cross_validate(pipe_rf_rfecv, X_train, y_train, cv = 1
```


[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001288 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3483
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 17
[LightGBM] [Info] Start training from score 1.090539
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001307 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 4045
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 21
[LightGBM] [Info] Start training from score 1.089050
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001616 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3997
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 22
[LightGBM] [Info] Start training from score 1.092965
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000384 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 4043
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 22
[LightGBM] [Info] Start training from score 1.086867
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001357 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3742
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 20
[LightGBM] [Info] Start training from score 1.094014
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000088 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 499
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 2
[LightGBM] [Info] Start training from score 1.093150
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000074 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 501
[LightGBM] [Info] Number of data points in the train set: 26403, number of used features: 2
[LightGBM] [Info] Start training from score 1.091178
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000068 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 499
[LightGBM] [Info] Number of data points in the train set: 26404, number of used features: 2
[LightGBM] [Info] Start training from score 1.092744

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001747 seconds.

You can set `force_col_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 5213

[LightGBM] [Info] Number of data points in the train set: 26404, number of used features: 31

[LightGBM] [Info] Start training from score 1.090174

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001758 seconds.

You can set `force_col_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 4703

[LightGBM] [Info] Number of data points in the train set: 26404, number of used features: 28

[LightGBM] [Info] Start training from score 1.099217

```
In [ ]: warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.simplefilter(action="ignore", category=DeprecationWarning)
warnings.simplefilter(action="ignore", category=UserWarning)

pipe_rf_rfecv.fit(X_train, y_train)
```

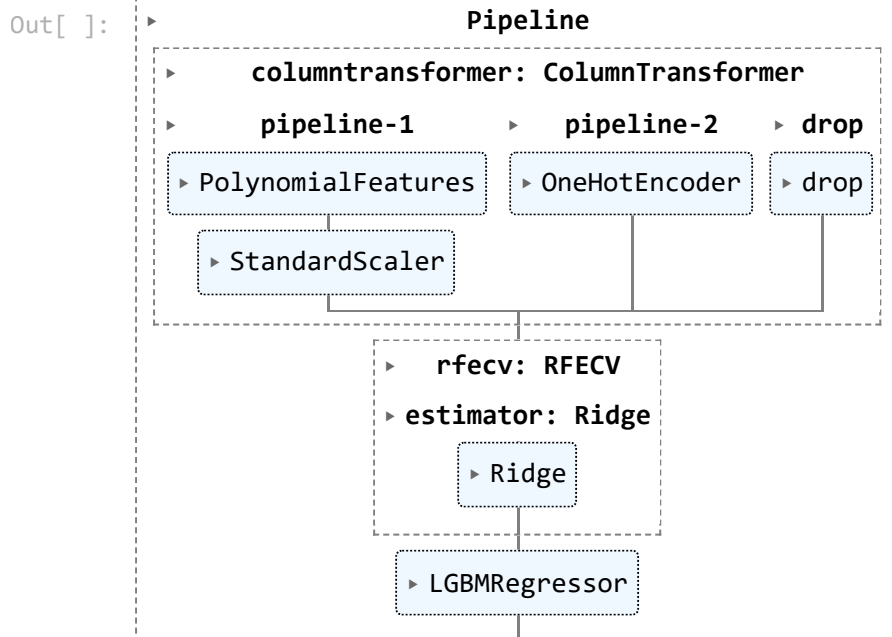
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.003221 seconds.

You can set `force_col_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 5476

[LightGBM] [Info] Number of data points in the train set: 29337, number of used features: 32

[LightGBM] [Info] Start training from score 1.091990



```
In [ ]: rfecv_fs = pipe_rf_rfecv.named_steps["rfecv"]
rfecv_fs.n_features_
```

Out []: 32

```
In [ ]: cross_val_results['RFECV'] = cv_df_rfecv.agg(['mean', 'std']).round(3).T
cross_val_results['RFECV']
```

Out[]:

	mean	std
fit_time	1.264	0.078
score_time	0.013	0.007
test_score	0.302	0.095
train_score	0.410	0.156

Crossvalidation Summary

In []: `pd.concat(cross_val_results, axis = 1)`

Out[]:

	ridge		randomforest		XGBoost		LGBMRegressor		svr_lasso_rfr_stac	
	mean	std	mean	std	mean	std	mean	std	mean	st
fit_time	0.037	0.007	34.706	0.294	0.937	0.072	0.170	0.062	129.229	23.38
score_time	0.002	0.003	0.043	0.009	0.006	0.007	0.016	0.000	8.405	0.20
test_score	0.161	0.018	0.344	0.028	0.344	0.028	0.374	0.035	0.242	0.02
train_score	0.164	0.002	0.506	0.003	0.695	0.007	0.533	0.007	0.250	0.00

As observed from the crossvalidation summary table, the LGBM regressor seems to perform well on the validation data. RFECV did not surpass LGBM regressor, so we will move forward with LGBM regressor.

10. Hyperparameter optimization

rubric={accuracy,reasoning}

Your tasks:

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

- `GridSearchCV`
- `RandomizedSearchCV`
- `scikit-optimize`

Points: 6

We are performing Hyperparameter Optimization on our best model `LGBM Regressor` to see if we can further improve the performance of the model.

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

        param_dist = {
            'lgbmregressor__learning_rate': uniform(loc=0.01, scale=0.3), # Continuous
            'lgbmregressor__num_leaves': randint(24, 80), # Discrete uniform distribution
            'lgbmregressor__max_depth': randint(3, 15), # Discrete uniform distribution
        }

        random_search = RandomizedSearchCV(estimator=pipe_lgbm, param_distributions=param_dist,
                                           n_iter=100, scoring='r2', cv=5, n_jobs=-1, random_state=42)

        random_search.fit(X_train, y_train)
```

[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.002827 seconds.

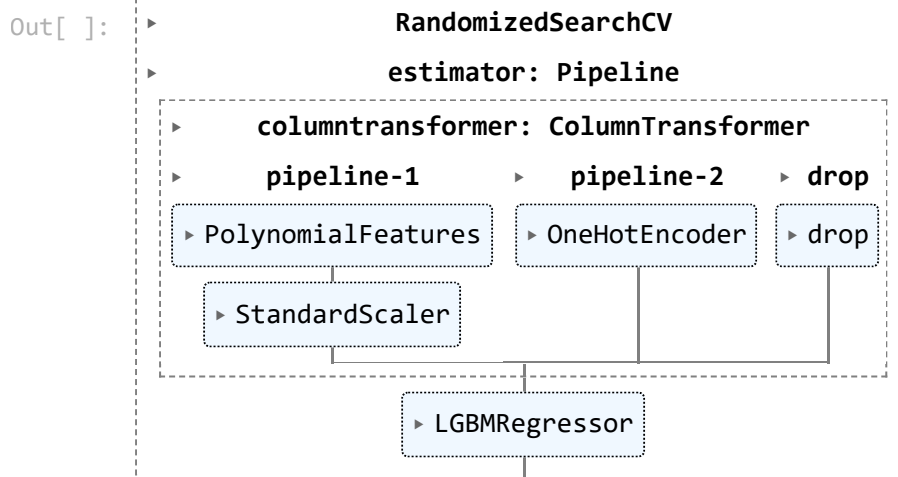
You can set `force_col_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 5988

[LightGBM] [Info] Number of data points in the train set: 29337, number of used features: 35

[LightGBM] [Info] Start training from score 1.091990

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf



```
In [ ]: print("Best Parameters:", random_search.best_params_)
```

Best Parameters: {'lgbmregressor__learning_rate': 0.04926848519922422, 'lgbmregressor__max_depth': 11, 'lgbmregressor__num_leaves': 79}

```
In [ ]: print("Best Score:", random_search.best_score_)
```

Best Score: 0.37876367272498035

11. Interpretation and feature importances

rubric={accuracy,reasoning}

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.

Points: 8

Answer - The following are the top 10 important features for the best model(lgbm). Most of the top features are the variables generated using the polynomialfeatures transformation which represent the interaction between the variables with a few individual variables such as longitude, availability_365 and latitude. We have used shap to identify how important each feature is to the model prediction. We observe high overlap in the features for the training data shown in the table below and the shap plot for the test data. The average impact is the highest for the feature longitude availability_365 while it is the lowest for longitude price and all average values are positive indicating contributions that push the predictions higher.

```
In [ ]: # feature importance for lgbm
feature_importances_lgbm_sorted
```

```
Out[ ]:
```

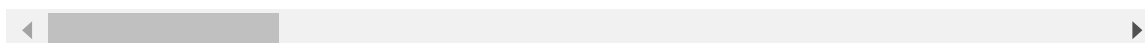
	Feature	Importance
8	latitude longitude	220
26	calculated_host_listings_count availability_365	200
15	longitude minimum_nights	184
2	longitude	181
19	price minimum_nights	181
20	price calculated_host_listings_count	180
6	availability_365	177
16	longitude calculated_host_listings_count	165
1	latitude	156
10	latitude minimum_nights	154

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

Out[]:

	1	latitude	longitude	price	minimum_nights	calculated_host_listings_co
29122	0.0	-0.638220	0.864528	-0.508014	-0.185974	-0.184
2964	0.0	0.336809	-0.569605	-0.288928	-0.275774	-0.184
14184	0.0	-1.603907	-0.308163	-0.420379	-0.096174	-0.184
1174	0.0	1.212375	-0.220944	-0.227583	-0.275774	-0.184
42853	0.0	-0.903454	0.932655	-0.332745	-0.230874	-0.184

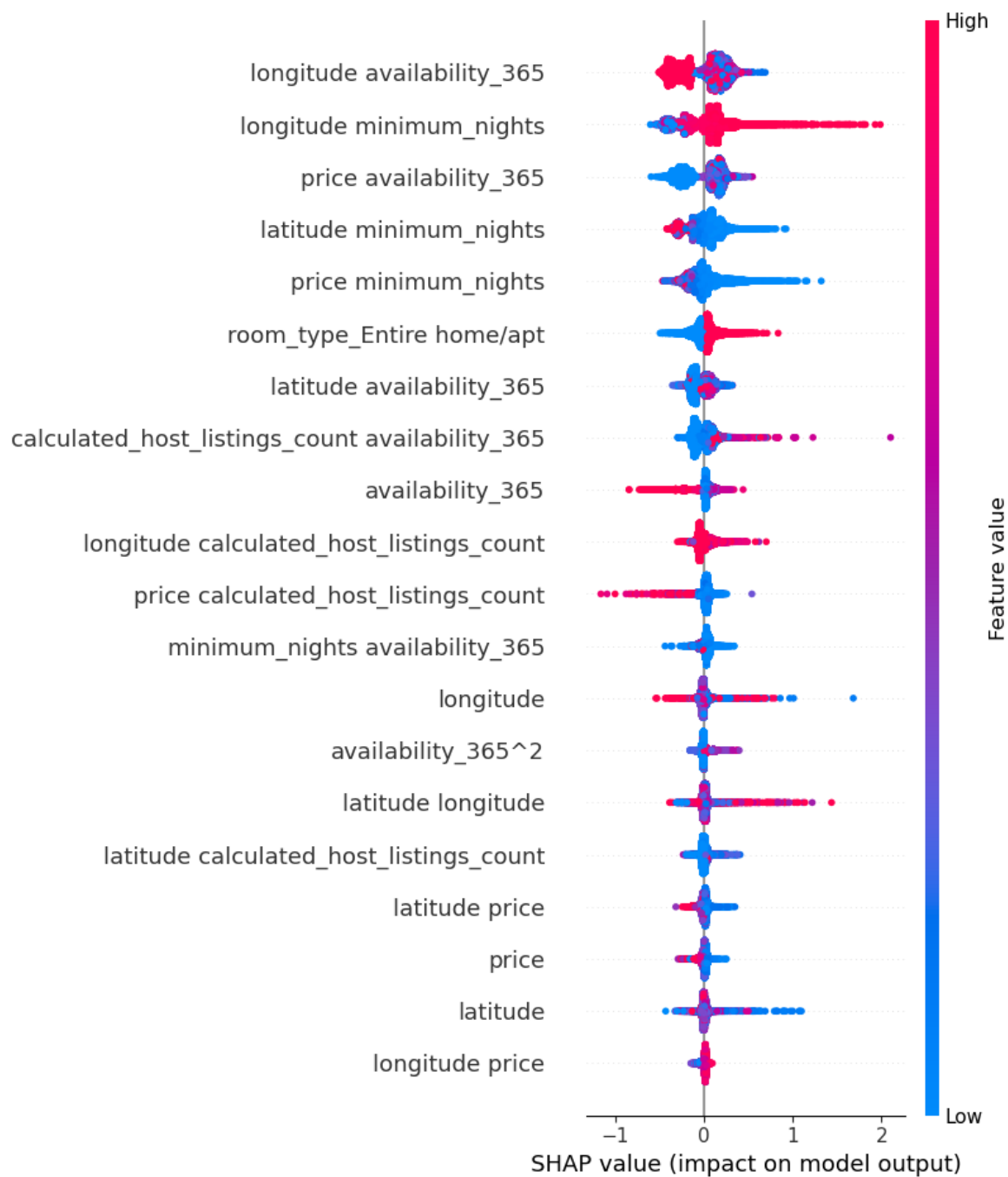
5 rows × 36 columns



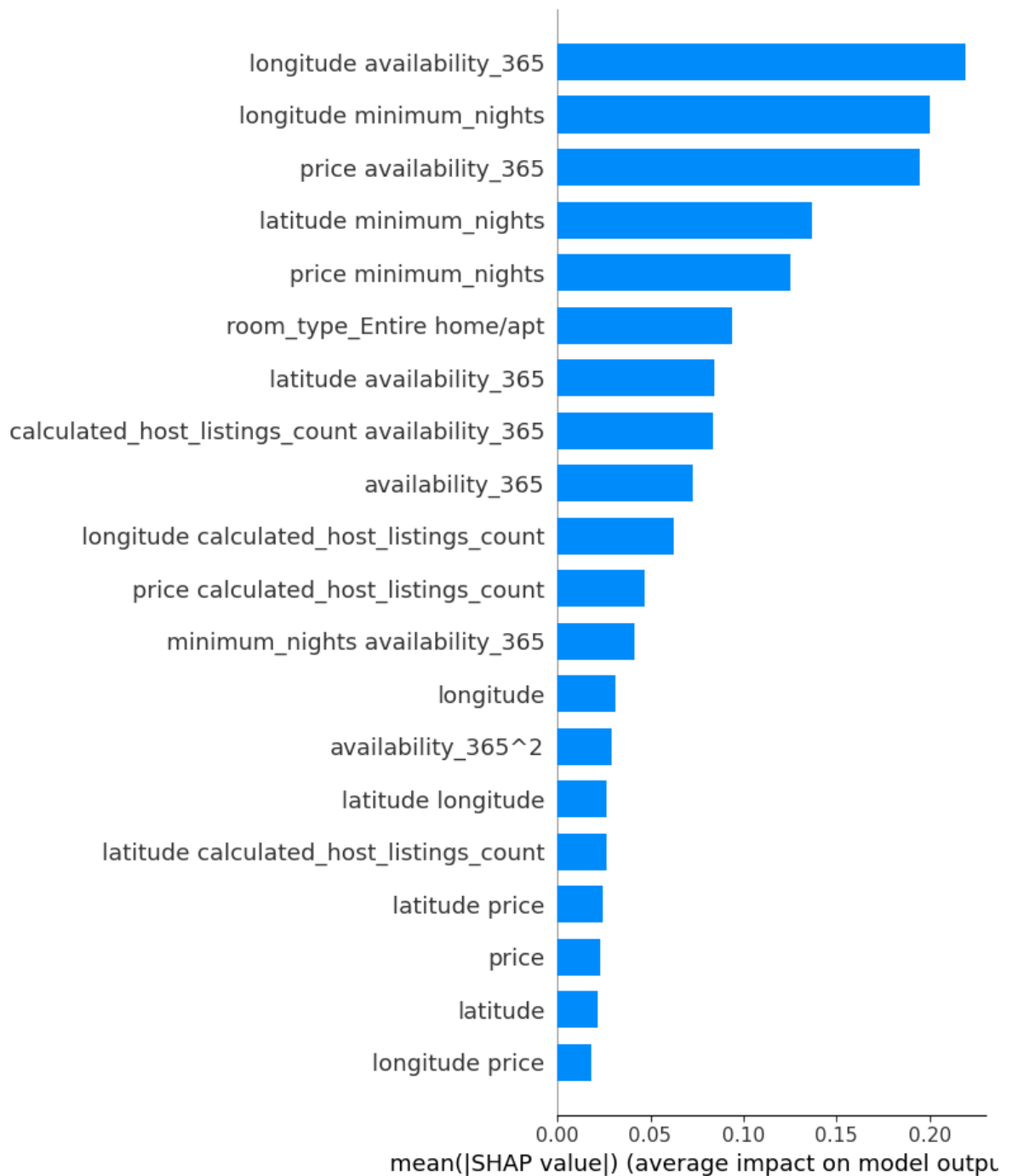
```
In [ ]: lgbm_explainer = shap.TreeExplainer(pipe_lgbm.named_steps["lgbmregressor"])
lgbm_explanation = lgbm_explainer(X_test_enc)
lgbm_explanation.shape
```

Out[]: (19558, 36)

```
In [ ]: lgbm_model = pipe_lgbm.named_steps['lgbmregressor']
explainer = shap.TreeExplainer(lgbm_model)
shap_values = explainer.shap_values(X_test_enc)
shap.summary_plot(shap_values, X_test_enc)
```



```
In [ ]: shap.summary_plot(shap_values, X_test_enc, plot_type="bar")
```



12. Results on the test set

rubric={accuracy,reasoning}

Your tasks:

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

Points: 6

The achieved test score of 0.39, derived from our best model using the LGBMRegressor, reflects a moderate level of predictive performance. This score is consistent with the validation score, which had a mean value of 0.374. However, the relatively low R^2 score indicates limitations in the model's accuracy, suggesting caution in relying on its predictions. One possible reason for this suboptimal performance could be the scarcity of pertinent features associated with the target variable. The insufficiency of relevant features might be a key factor contributing to the model's inability to achieve a higher test score.

These shap values quantify the impact of each feature on the model's output for a specific prediction (in this case, for `ex_one_ind` and `ex_g1_ind`). SHAP values show the direction and magnitude of each feature's contribution to the prediction. Negative SHAP values imply the feature's contribution decreased the output (prediction), while positive values suggest it increased the output. The value 0.441 represents the predicted output (target variable or regression prediction) of our LightGBM model for the particular instance indexed by `ex_one_ind` in the test data `X_test_enc`. For the specific instance `ex_one_ind`, longitude availability_365 has the most considerable positive impact on the model's prediction, followed by price availability_365. Conversely, longitude minimum_nights, latitude minimum_nights, and other negatively impactful features contributed to reducing the model's prediction for this instance.

The predicted value generated by the LightGBM regressor model for the second instance indexed as `ex_g1_ind` is approximately 2.944. This prediction suggests an expected outcome for that particular instance. For this instance, the most influential features affecting the prediction are longitude, price availability_365, price minimum_nights, latitude minimum_nights, and longitude calculated_host_listings_count, having the largest positive SHAP values. On the other hand, minimum_nights availability_365, room_type_Entire home/apt, and neighbourhood_group_Manhattan have notably negative SHAP values, suggesting they contribute negatively to the predicted value. These SHAP values signify the relative importance and directionality (positive or negative impact) of each feature towards the prediction made by the model for this specific data instance.

```
In [ ]: round(pipe_lgbm.score(X_test, y_test),2)
```

```
Out[ ]: 0.39
```

```
In [ ]: y_test_reset = y_test.reset_index(drop=True)  
y_test_reset
```

```
Out[ ]: 0      0.00
        1      1.93
        2      0.17
        3      3.51
        4      2.62
        ...
        19553  0.59
        19554  0.15
        19555  0.00
        19556  0.90
        19557  0.39
        Name: reviews_per_month, Length: 19558, dtype: float64
```

```
In [ ]: one_ind = y_test_reset[y_test_reset <= 1].index.tolist()
        g1_ind = y_test_reset[y_test_reset > 1].index.tolist()

        ex_one_ind = one_ind[20]
        ex_g1_ind = g1_ind[20]
```

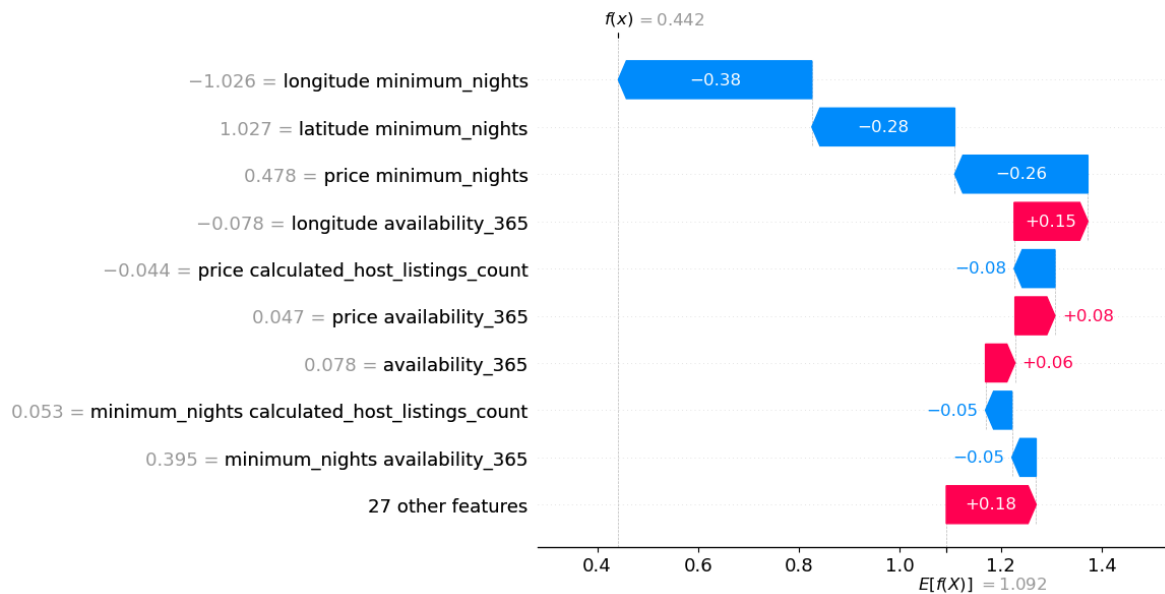
```
In [ ]: X_test_enc.iloc[ex_one_ind]
```

```
Out[ ]: 1      0.000000
        latitude      0.661940
        longitude     -0.312503
        price          0.149246
        minimum_nights 1.026325
        calculated_host_listings_count -0.030862
        availability_365 0.077514
        latitude^2     0.661456
        latitude longitude -0.757439
        latitude price  0.149910
        latitude minimum_nights 1.027213
        latitude calculated_host_listings_count -0.030724
        latitude availability_365 0.078355
        longitude^2    0.312345
        longitude price -0.149244
        longitude minimum_nights -1.026288
        longitude calculated_host_listings_count 0.030879
        longitude availability_365 -0.077750
        price^2        -0.028895
        price minimum_nights 0.477734
        price calculated_host_listings_count -0.043540
        price availability_365 0.047322
        minimum_nights^2 0.024065
        minimum_nights calculated_host_listings_count 0.052882
        minimum_nights availability_365 0.395121
        calculated_host_listings_count^2 -0.114723
        calculated_host_listings_count availability_365 -0.103109
        availability_365^2 -0.336419
        room_type_Entire home/apt 1.000000
        room_type_Private room 0.000000
        room_type_Shared room 0.000000
        neighbourhood_group_Bronx 0.000000
        neighbourhood_group_Brooklyn 0.000000
        neighbourhood_group_Manhattan 1.000000
        neighbourhood_group_Queens 0.000000
        neighbourhood_group_Staten Island 0.000000
        Name: 34615, dtype: float64
```

```
In [ ]: pipe_lgbm.named_steps["lgbmregressor"].predict(X_test_enc)[ex_one_ind]
```

```
Out[ ]: 0.44186971039103645
```

```
In [ ]: shap.plots.waterfall(lgbm_explanation[ex_one_ind])
```



```
In [ ]: explainer = shap.TreeExplainer(lgbm_model)
shap_values = explainer.shap_values(X_test_enc)
shap_df_ex_one_ind = pd.DataFrame(shap_values[ex_one_ind], index=feature_names,
shap_df_ex_one_ind
```

Out[]:

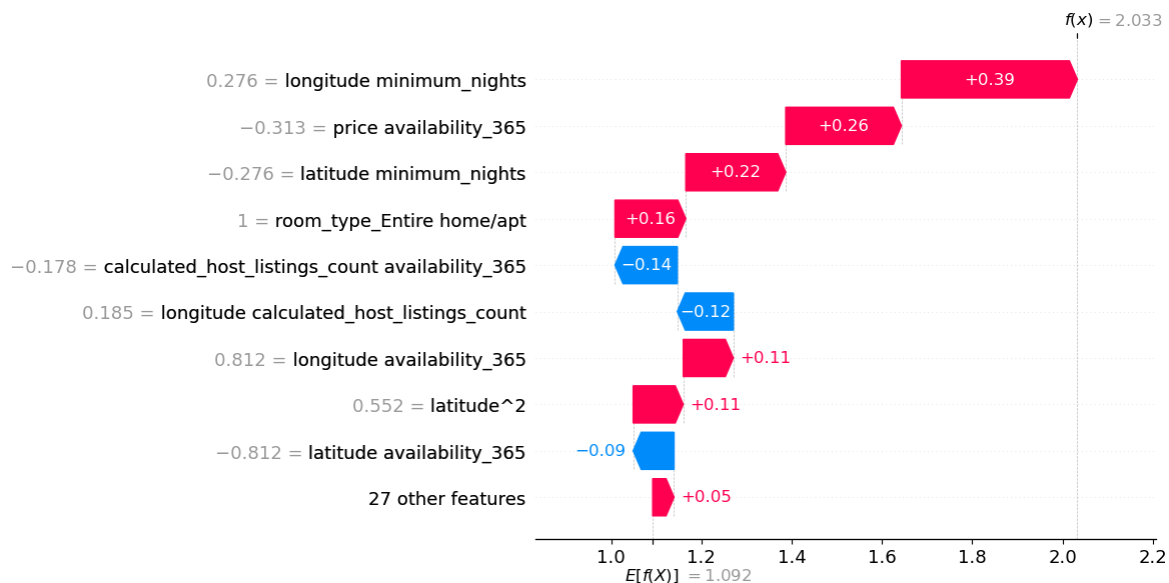
	SHAP values
longitude minimum_nights	-0.383678
latitude minimum_nights	-0.283262
price minimum_nights	-0.263833
price calculated_host_listings_count	-0.080868
minimum_nights calculated_host_listings_count	-0.052397
minimum_nights availability_365	-0.048046
longitude	-0.021107
latitude price	-0.009827
minimum_nights	-0.008693
room_type_Private room	-0.000922
1	0.000000
calculated_host_listings_count^2	0.000000
minimum_nights^2	0.000000
neighbourhood_group_Staten Island	0.000000
price^2	0.000321
neighbourhood_group_Queens	0.000364
price	0.001356
neighbourhood_group_Bronx	0.001571
room_type_Shared room	0.002242
longitude price	0.003036
neighbourhood_group_Brooklyn	0.003716
longitude^2	0.004528
availability_365^2	0.006230
calculated_host_listings_count	0.006524
latitude availability_365	0.008513
latitude^2	0.012340
neighbourhood_group_Manhattan	0.015564
latitude	0.016679
latitude calculated_host_listings_count	0.018178
latitude longitude	0.018719
calculated_host_listings_count availability_365	0.022317
longitude calculated_host_listings_count	0.035174
room_type_Entire home/apt	0.041279

SHAP values	
availability_365	0.058544
price availability_365	0.079186
longitude availability_365	0.146134

```
In [ ]: pipe_lgbm.named_steps["lgbmregressor"].predict(X_test_enc)[ex_g1_ind]
```

```
Out[ ]: 2.0328970436638616
```

```
In [ ]: shap.plots.waterfall(lgbm_explanation[ex_g1_ind])
```



```
In [ ]: shap_df_ex_g1_ind = pd.DataFrame(shap_values[ex_g1_ind], index=feature_names, columns=feature_names)
shap_df_ex_g1_ind
```

Out[]:

SHAP values

calculated_host_listings_count availability_365	-0.138006
longitude calculated_host_listings_count	-0.124163
latitude availability_365	-0.090538
longitude	-0.064412
latitude longitude	-0.055156
availability_365^2	-0.031378
latitude price	-0.028444
neighbourhood_group_Manhattan	-0.010464
minimum_nights calculated_host_listings_count	-0.005729
neighbourhood_group_Queens	-0.005244
room_type_Private room	-0.001694
neighbourhood_group_Brooklyn	-0.000771
price^2	-0.000011
neighbourhood_group_Bronx	-0.000006
calculated_host_listings_count^2	0.000000
minimum_nights^2	0.000000
1	0.000000
neighbourhood_group_Staten Island	0.000000
minimum_nights	0.002432
calculated_host_listings_count	0.002723
latitude	0.004774
longitude^2	0.007061
room_type_Shared room	0.008369
price	0.010104
latitude calculated_host_listings_count	0.011472
longitude price	0.013759
price calculated_host_listings_count	0.031107
availability_365	0.050214
minimum_nights availability_365	0.053752
price minimum_nights	0.054871
latitude^2	0.110922
longitude availability_365	0.110956
room_type_Entire home/apt	0.156880

SHAP values	
latitude minimum_nights	0.221371
price availability_365	0.256154
longitude minimum_nights	0.390001

13. Summary of results

rubric={reasoning}

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

Your tasks:

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

Points: 8

```
In [ ]: pd.concat(cross_val_results, axis = 1)
```

```
Out[ ]:
```

	ridge		randomforest		XGBoost		LGBMRegressor		svr_lasso_rfr_stac	
	mean	std	mean	std	mean	std	mean	std	mean	st
fit_time	0.037	0.007	34.706	0.294	0.937	0.072	0.170	0.062	129.229	23.38
score_time	0.002	0.003	0.043	0.009	0.006	0.007	0.016	0.000	8.405	0.20
test_score	0.161	0.018	0.344	0.028	0.344	0.028	0.374	0.035	0.242	0.02
train_score	0.164	0.002	0.506	0.003	0.695	0.007	0.533	0.007	0.250	0.00

Overall Conclusion:

In this analysis we tried out a variety of models and methods with the goal of finding a suitable model to predict the number of reviews per month. While LGBM Regressor exhibits a suitable balance between fitting time, scoring time, and predictive performance (achieving the best overall results) the final test score of 0.39 is moderate. For practical applications and future deployment, a higher-scoring model would be desirable.

Potential Improvements:

Despite the comprehensive strategies implemented in the analysis, there are avenues for further exploration. Additional feature engineering, particularly focusing on `minimum_nights`, could be beneficial. The relationship between the number of reviews per month and the number of different bookings per month may provide valuable insights. This is due to the fact that a place mainly booking several short-term stays has more opportunities to get reviews compared to an equally-booked place with mainly long-term stays. This may in fact raise concern in the legitimacy of using number of reviews per month as a popularity proxy.

The research question itself may introduce bias. In addition to needing account for the average length of stays, using the number of reviews per month as a proxy for a place's popularity may overlook the sentiment of the reviews. Incorporating data on the positivity or negativity of reviews is important. Additionally, 4 of the features were related to geographical location. It is plausible that these features have more impact on the pricing of the Airbnb, rather than the number of reviews. Location is out of the hosts control. Factors that often show up in reviews are those within the hosts control. Factors like the number of amenities, ease of access, cleaning fees, and host responsiveness, could enhance the predictive power of the model.

Addressing these considerations could lead to a more accurate prediction of the number of reviews and perhaps, the popularity of Airbnb listings.

14. Creating a data analysis pipeline (Challenging)

rubric={reasoning}

Your tasks:

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

Points: 2

Type your answer here, replacing this text.

15. Your takeaway from the course (Challenging)

rubric={reasoning}



Your tasks:

What is your biggest takeaway from this course?

Points: 0.25

Type your answer here, replacing this text.

Restart, run all and export a PDF before submitting

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

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

Help us improve the labs

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

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

Ans:

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

Ans: