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

Lab 4: Putting it all together in a mini project

This lab is an optional group lab. You can choose to work alone of 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 of your GitHub repos.

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.ubc.ca/MDS-2020-21/DSCI_531_labX_yourcwl.

Points: 2

https://github.com/UBC-MDS/DSCI_573_lab4_Randy

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

We are picking the problem statement 2.

Objective:

> The Objective of this Data Science project is to develop a model based on the available dataset to predict the popularity of an AirBnb listing in New York City. The target variable `reviews_per_month` is used as a proxy for popularity of the listing.

This significance of this project lies in the possibility that with a reliable Machine Learning model which can estimate the popularity of an AirBnb listing, we can determine the parameters that influence popularity of the Ad. This will in turn help the AirBnb hosts to write effective listings. It will help the company, AirBnb to increase number of hosts that can meet these parameters. Thus AirBNB can revitalize its business model to increase profits by focusing attention selective and promising listings. It will also enable the users/renters to have a more positive experience in using AirBnb.

License:

The materials used in this Data Science project are licensed under Creative Commons Attribution (CC0 1.0 Universal (CC0 1.0) Public Domain Dedication)

Data:

The Data for this project can be downloaded at:

https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data

FINAL RESULT

- Best TEST Score Achieved: 67.8%
- Scoring Metric: R-Squared
- Best Performing Model: LGBM

KEY TAKEAWAY: Airbnb could look into promoting hosts who are available for more days during the year and also those that allow larger number of subsequent nights for booking. Find out this inference from our analysis below!

Collaborators:

- Andy Wang

- Ranjitprakash Sundaramurthi

In [ ]:

In [2]:

```python
# Importin the necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

from sklearn.compose import (
    ColumnTransformer,
    TransformedTargetRegressor,
    make_column_transformer,
)
from sklearn.dummy import DummyRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.metrics import make_scorer, mean_squared_error, r2_score
from sklearn.model_selection import (
    GridSearchCV,
    cross_val_score,
    cross_validate,
    train_test_split,
)
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.tree import DecisionTreeRegressor
from sklearn.feature_selection import SelectFromModel

from sklearn.model_selection import RandomizedSearchCV
import random
from scipy.stats import randint

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

%matplotlib inline
```

In [3]:

```python
house_df = pd.read_csv('data/data.csv')
house_df.head()
```

Out[3]:

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

Understanding the features in dataset

`name` : The title of the listing that the host has listed on AirBnB.

`host_id` and `host_name` : Id and name of the host for Airbnb listing.

`Neighbourhood_group` : The location in NYC where the listing is present. There are 5 unique values.

`neighbourhood` : The specific neighborhood withing the group where the property is location.

`latitude` and `longitude` : The geographical coordinates of the location.

`room_type` : The type of room the property. This can be entire home, private room or shared room.

`price` : The listed price per night on the Ad.

`minimum_nights` : The minimum number of nights the property can be booked for.

`number_of_reviews` : The number of user reviews previously posted on the property by users.

`last_review` : The date of the last review made on the property.

`reviews_per_month` : This is the proxy for the target that we need to predict. This represents the number of reviews on average per month for the property.

`calculated_host_listings_count` : The total number of listings per host.

`availability_365` : The numebr of days in the year that property is available to occupy.

In [4]:

```
print(house_df.info())
# last_review and review_per_month has null value
```

```
<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
None
```

In [5]:
```
# category
print("Neighbourhood Groups:", house_df['neighbourhood_group'].unique().tolist())
print("Room Types:", house_df['room_type'].unique().tolist())
house_df.dtypes
```

```
Neighbourhood Groups: ['Brooklyn', 'Manhattan', 'Queens', 'Staten Island', 'Bronx']
Room Types: ['Private room', 'Entire home/apt', 'Shared room']
```

Out[5]:
```
id                                  int64
name                               object
host_id                             int64
host_name                          object
neighbourhood_group                object
neighbourhood                      object
latitude                          float64
longitude                         float64
room_type                          object
price                               int64
minimum_nights                      int64
number_of_reviews                   int64
last_review                        object
reviews_per_month                 float64
calculated_host_listings_count      int64
availability_365                    int64
dtype: object
```

In [6]:
```
# Details regarding the host
print(f'Number of unique host id and host names: {len(house_df["host_id"].unique().tolist())} an

# host_id and maybe host_name, can be dropped

# OPTIONAL: house_df.drop(['host_name'], axis=1, inplace =True) # drop user name for pravicy
```

```
Number of unique host id and host names: 37457 and 11453 respectively
```

In [7]:
```
house_df.isnull().sum()
```

Out[7]:
```
id                                0
name                             16
host_id                           0
host_name                        21
neighbourhood_group               0
neighbourhood                     0
latitude                          0
longitude                         0
room_type                         0
price                             0
minimum_nights                    0
number_of_reviews                 0
last_review                   10052
reviews_per_month             10052
calculated_host_listings_count    0
availability_365                  0
dtype: int64
```

In [8]:
```
# Appears that both last_review and reviews_per_month have the same rows with missing values. Le
house_df[house_df["last_review"].isna() & house_df["reviews_per_month"].isna()].shape
```

Out[8]:
```
(10052, 16)
```

Since both `last_review` and `reviews_per_month` are both Null together. Thus the missing values are missing not at random (MNAR). Thus the `reviews_per_month` is not computed likely because of missing values in `last_review`. Since it is the target variable imputation is not recommended as it will introduce bias into the model.

Thus we are choosing to drop these rows from the dataframe.

In [9]:
```
# Looking at the rows with name feature which has missing values
house_df[house_df['name'].isna()].head()
```

Out[9]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_t
2854	1615764	NaN	6676776	Peter	Manhattan	Battery Park City	40.71239	-74.01620	En home/
3703	2232600	NaN	11395220	Anna	Manhattan	East Village	40.73215	-73.98821	En home/
5775	4209595	NaN	20700823	Jesse	Manhattan	Greenwich Village	40.73473	-73.99244	En home/
5975	4370230	NaN	22686810	Michaël	Manhattan	Nolita	40.72046	-73.99550	En home/
6269	4581788	NaN	21600904	Lucie	Brooklyn	Williamsburg	40.71370	-73.94378	Priv rc

In [10]:
```
# Looking at the rows with host_name feature which has missing values
house_df[house_df['host_name'].isna()].head()
```

Out[10]:

	id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	roo
360	100184	Bienvenue	526653	NaN	Queens	Queens Village	40.72413	-73.76133	
2700	1449546	Cozy Studio in Flatbush	7779204	NaN	Brooklyn	Flatbush	40.64965	-73.96154	hc
5745	4183989	SPRING in the City!! Zen-Style Tranquil Bedroom	919218	NaN	Manhattan	Harlem	40.80606	-73.95061	
6075	4446862	Charming Room in Prospect Heights!	23077718	NaN	Brooklyn	Crown Heights	40.67512	-73.96146	
6582	4763327	Luxurious, best location, spa inc'l	24576978	NaN	Brooklyn	Greenpoint	40.72035	-73.95355	hc

Several of the `name` feature with missing values are also missing values in the `last_review` and `reviews_per_month`. We decided there is no necessity to consider imputation on a `name` feature as it is a text feature which is harder to impute. Furthermore we are dropping `last_review` and `reviews_per_month` missing value rows which overlap with missing values on `name`.

In [11]:

```
# drop rows with the missing values
house_df = house_df[house_df['name'].notna()]
house_df = house_df[house_df['host_name'].notna()]
house_df = house_df[house_df['last_review'].notna()]
house_df = house_df[house_df['reviews_per_month'].notna()]
```

In [12]:

```
# checking the structure of the dataframe again
house_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 38821 entries, 0 to 48852
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              38821 non-null  int64  
 1   name                            38821 non-null  object 
 2   host_id                         38821 non-null  int64  
 3   host_name                       38821 non-null  object 
 4   neighbourhood_group             38821 non-null  object 
 5   neighbourhood                   38821 non-null  object 
 6   latitude                        38821 non-null  float64
 7   longitude                       38821 non-null  float64
 8   room_type                       38821 non-null  object 
 9   price                           38821 non-null  int64  
 10  minimum_nights                  38821 non-null  int64  
 11  number_of_reviews               38821 non-null  int64  
 12  last_review                     38821 non-null  object 
 13  reviews_per_month               38821 non-null  float64
 14  calculated_host_listings_count  38821 non-null  int64  
 15  availability_365                38821 non-null  int64  
dtypes: float64(3), int64(7), object(6)
memory usage: 5.0+ MB
```

In [13]:

```
# change dates of review to pandas inbuilt date
house_df['last_review'] = pd.to_datetime(
    house_df['last_review'],
    infer_datetime_format=True
)
house_df.dtypes
```

Out[13]:

```
id                                         int64
name                                      object
host_id                                    int64
host_name                                 object
neighbourhood_group                       object
neighbourhood                             object
latitude                                 float64
longitude                                float64
room_type                                 object
price                                      int64
minimum_nights                             int64
number_of_reviews                          int64
last_review                       datetime64[ns]
reviews_per_month                        float64
calculated_host_listings_count             int64
availability_365                           int64
dtype: object
```

In [14]:

```
# Create a new feature column "days_since_review"

house_df= house_df.assign(
    days_since_review = house_df['last_review'].apply(lambda x: (np.datetime64('today')-x).days)
)
```

In [15]:

```
# Now we can drop the last_review column as we have captured its effect through days_since_review
# Dropping host_id since it has unique identifier for most rows
# OPTIONAL: Consider dropping host_name?

house_df = house_df.drop(columns = ["last_review", "host_id", "id"])
```

In [16]:

```
house_df.head()
```

Out[16]:

	name	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minim
0	Clean & quiet apt home by the park	John	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	
1	Skylit Midtown Castle	Jennifer	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	
3	Cozy Entire Floor of Brownstone	LisaRoxanne	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	
4	Entire Apt: Spacious Studio/Loft by central park	Laura	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	
5	Large Cozy 1 BR Apartment In Midtown East	Chris	Manhattan	Murray Hill	40.74767	-73.97500	Entire home/apt	200	

In [17]:
```
# Suggestion for feature transformation to reduce the number of categories
```

In [18]:
```
# to reduce the number of unique categories for host_name we could bunch together under "other" c
```

In [19]:
```
name_df = pd.DataFrame(house_df["host_name"].value_counts())

# Determining the threshold for which roughly half of the data set is "other"
threshold = 14

name_df[name_df["host_name"]<threshold].sum()
```

Out[19]:
```
host_name    19304
dtype: int64
```

In [20]:
```
name_df[name_df["host_name"]>=threshold].sum()
```

Out[20]:
```
host_name    19517
dtype: int64
```

In [21]:
```
name_other = name_df[name_df["host_name"]<threshold].index.tolist()
```

In [22]:
```
house_df = house_df.assign(host_name_mod = house_df["host_name"].apply(lambda x: "other" if x in
```

2. Data splitting

rubric={reasoning}

Your tasks:

1. Split the data into train and test portions.

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

Points: 1

In [23]:
```
from sklearn.model_selection import train_test_split

size = 0.2

train_df, test_df = train_test_split(house_df, test_size = size, random_state = 573)
train_df.head()
```

Out[23]:

	name	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	mi
21151	GORGEOUS Very Large Room next to Central Park!	Mike	Manhattan	Upper East Side	40.76281	-73.96718	Private room	117	
34227	Modern Newly Renovated Home Away From Home	Yva	Brooklyn	Mill Basin	40.61768	-73.91669	Entire home/apt	85	
19218	Clean and Sunny room near Midtown Manhattan	Daljit	Queens	Astoria	40.75786	-73.92808	Private room	49	
2008	2 Bedroom Bushwick Apartment	Robert	Brooklyn	Bushwick	40.69982	-73.91957	Entire home/apt	139	
31911	Home away from home in Harlem	Jessica	Manhattan	Harlem	40.80793	-73.95127	Entire home/apt	225	

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

Exploring the Target variable

In [24]:
```
import altair as alt
```

```
# alt.renderers.enable('mimetype')
alt.data_transformers.enable('default', max_rows=None)
```

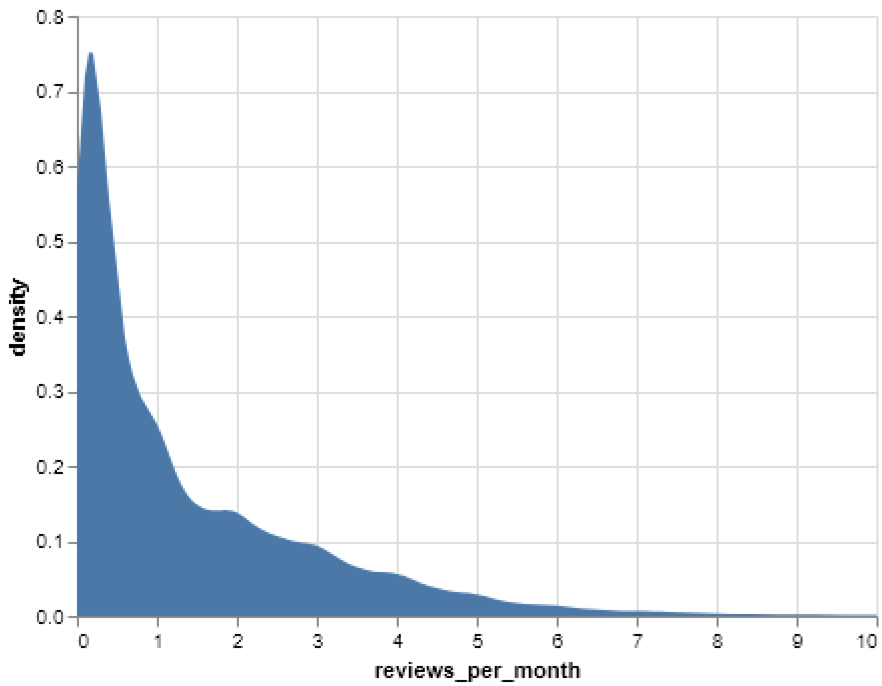
Out[24]:

```
DataTransformerRegistry.enable('default')
```

In [25]:

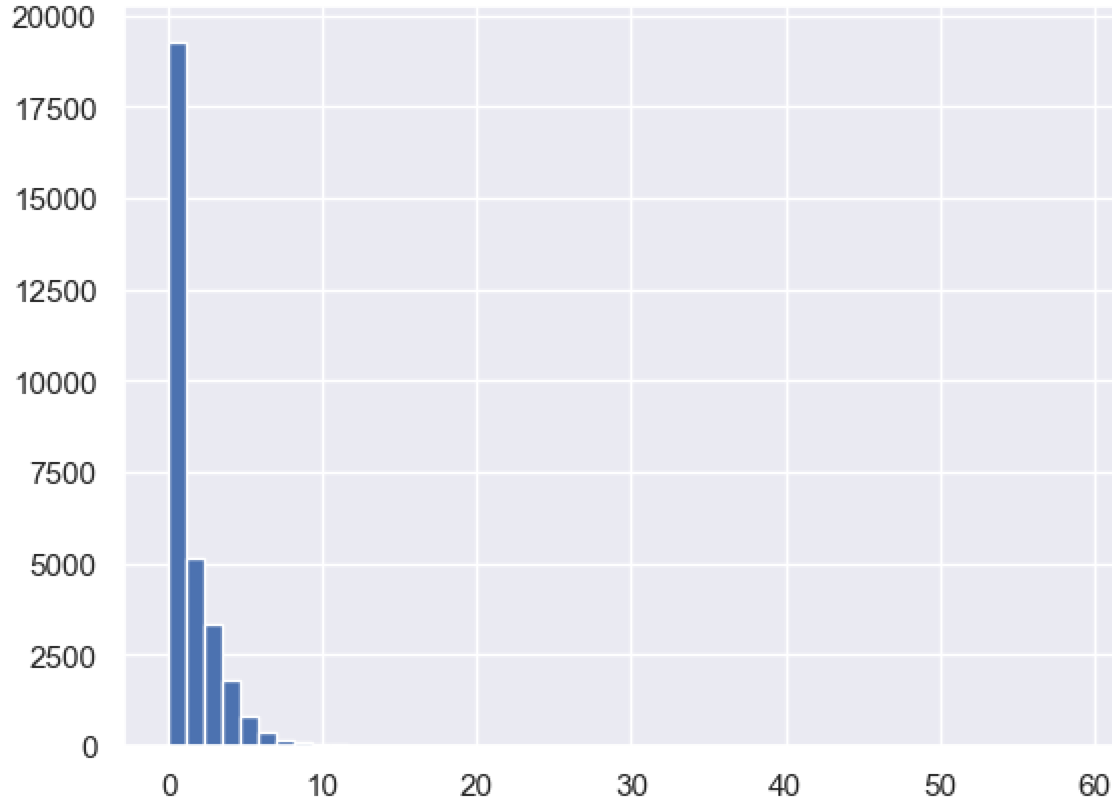
```
# target distribution
(alt.Chart(train_df)
 .transform_density(
    'reviews_per_month',
    as_=['reviews_per_month', 'density'],
    extent=[0, 10])  # Give the name "density" the KDE columns we just created
 .mark_area().encode(
    x='reviews_per_month',
    y='density:Q')
)
```

Out[25]:



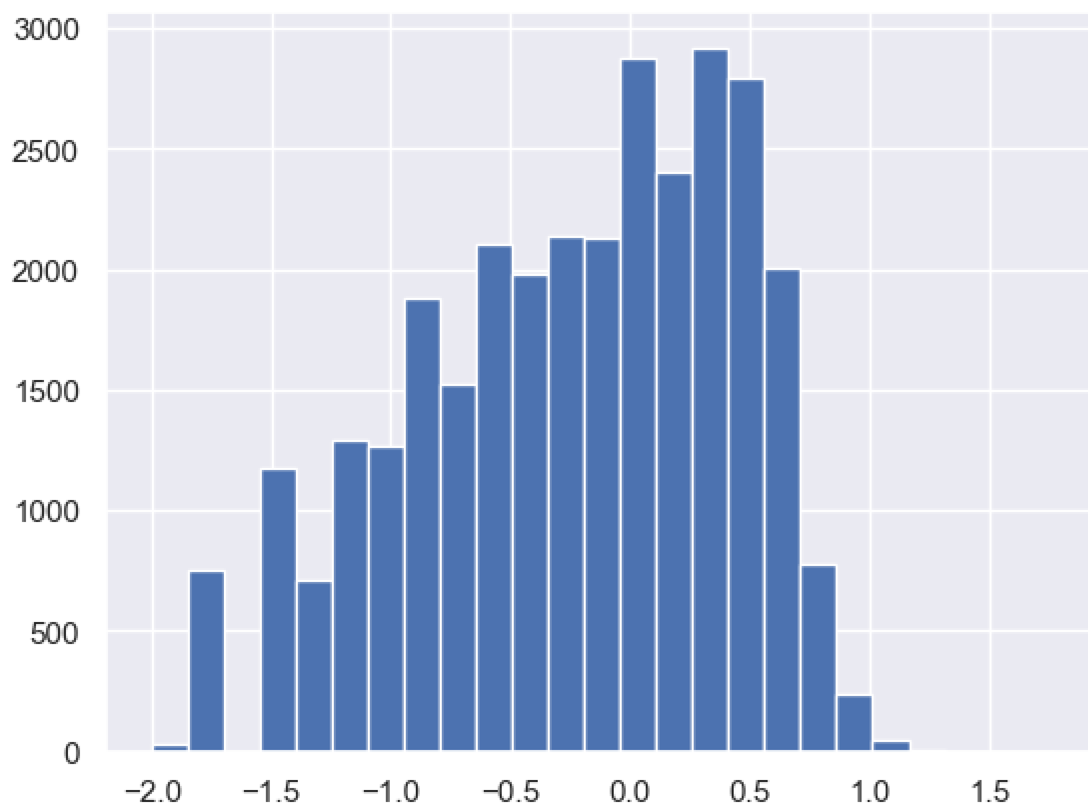
In [26]:

```
# target variable histogram distribution
plt.hist(train_df['reviews_per_month'], bins=50);
```



Since the target variable is highly right skewed, we can consider a logarithmic transformation on it to see if it looks more bell shaped. This may improve the performance of linear models as Linear Regression will usually work better to predict something that looks Normally distributed.

In [27]:
```
# target after log transformation - histogram distribution
plt.hist(np.log10(train_df['reviews_per_month']), bins=25);
```



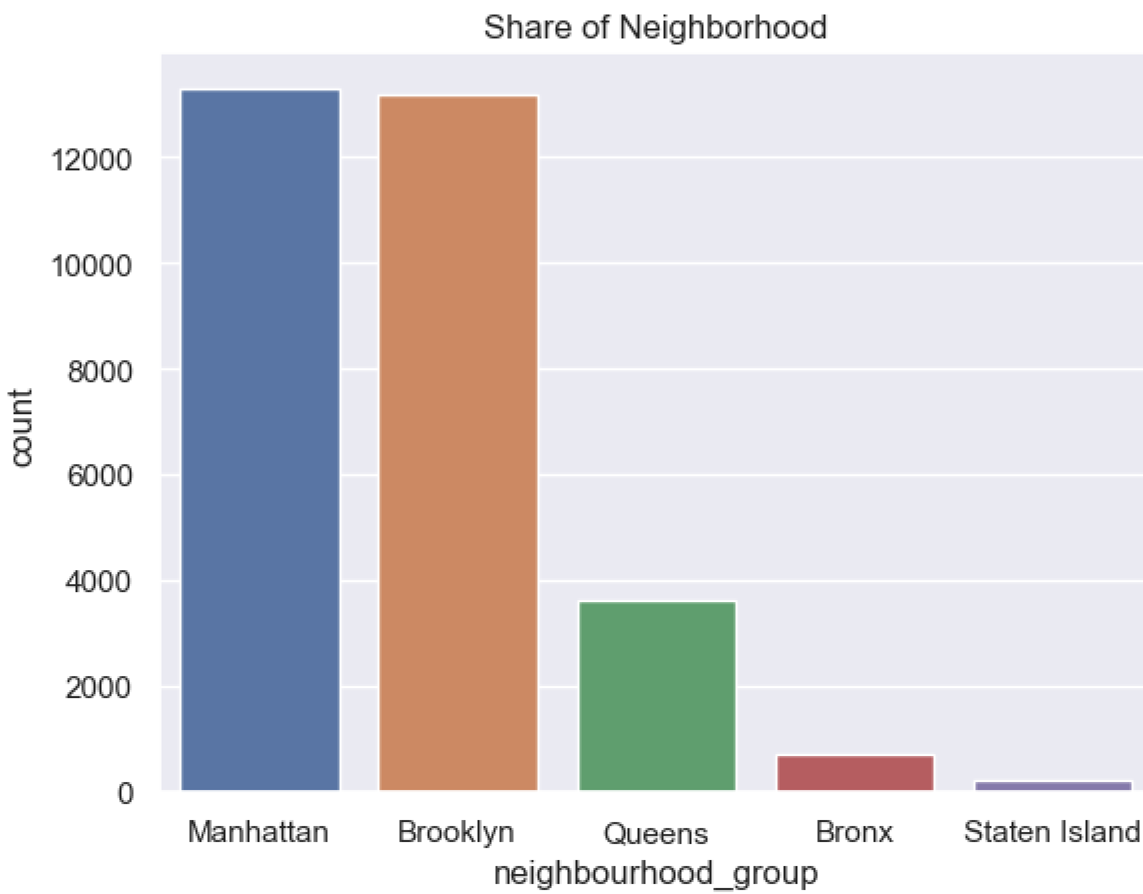
The transformed target variable doesn't look much bell shaped as expected and is left skewed. Thus transformation of the target variable. We are choosing not to pursue this transformation on target variable.

Instead, non-linear models will be tested for improving performance.

Exploring the Categorical features

In [28]:

```
# visualize the number of Airbnbs in each neighbourhood_group
my_order = train_df['neighbourhood_group'].value_counts().index
ax = sns.countplot(x=train_df['neighbourhood_group'],order=my_order)
ax.set_title('Share of Neighborhood')
plt.show()
```



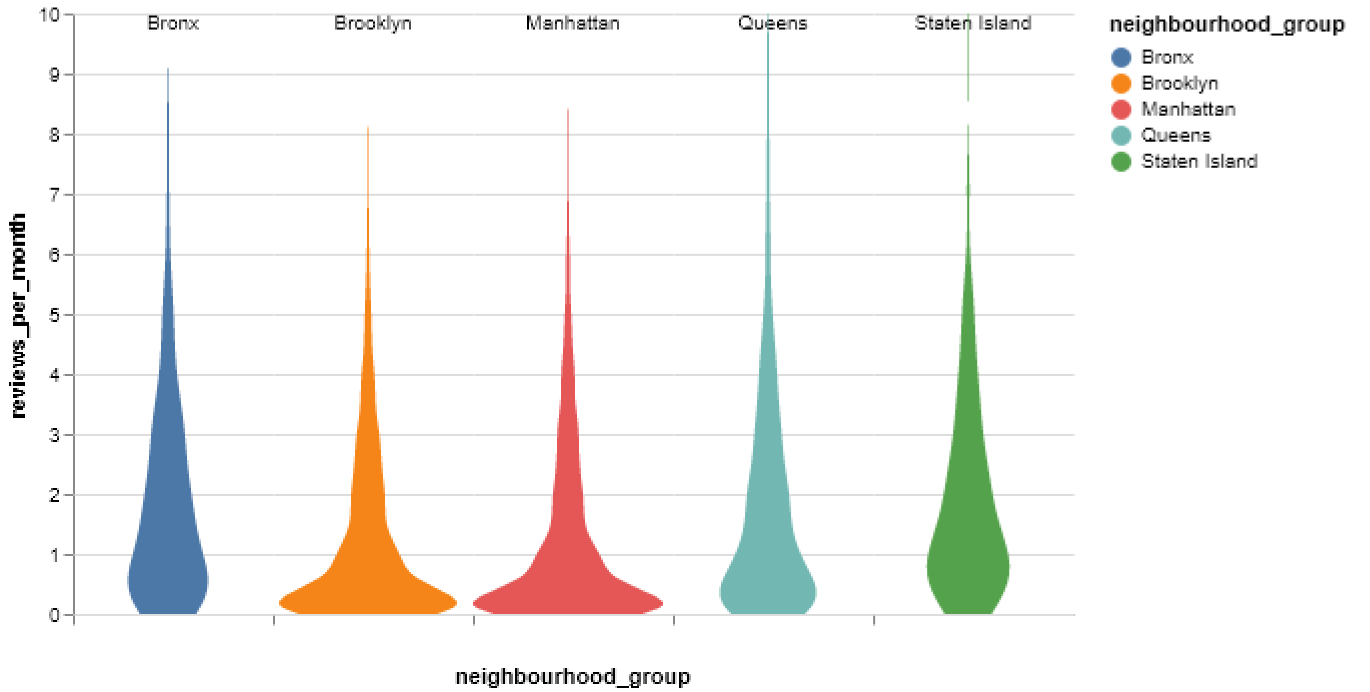
There are lot of examples for Manhattan and Brooklyn categories as compared to the other categories

In [29]:

```
# target distribution for each group
alt.Chart(train_df).transform_density(
    'reviews_per_month',
    as_=['reviews_per_month', 'density'],
    extent=[0, 10],
    groupby=['neighbourhood_group']
).mark_area(orient='horizontal').encode(
    y='reviews_per_month:Q',
    color='neighbourhood_group:N',
    x=alt.X(
        'density:Q',
        stack='center',
        impute=None,
        title=None,
        axis=alt.Axis(labels=False, values=[0],grid=False, ticks=True),
    ),
    column=alt.Column(
        'neighbourhood_group:N',
        header=alt.Header(
            titleOrient='bottom',
```

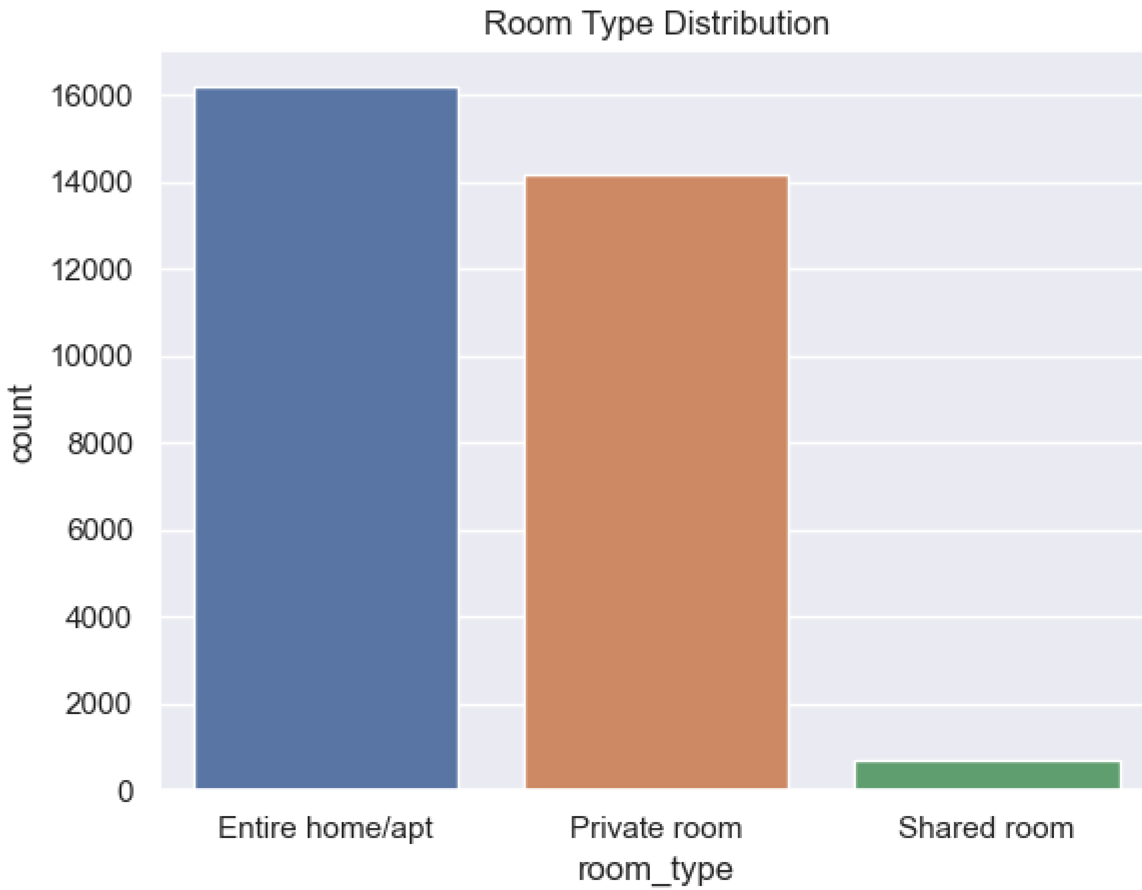
```
            labelOrient='bottom',
            labelPadding=0,
        ),
    )
).properties(
    width=100
).configure_facet(
    spacing=0
).configure_view(
    stroke=None
)
```

Out[29]:



In [30]:

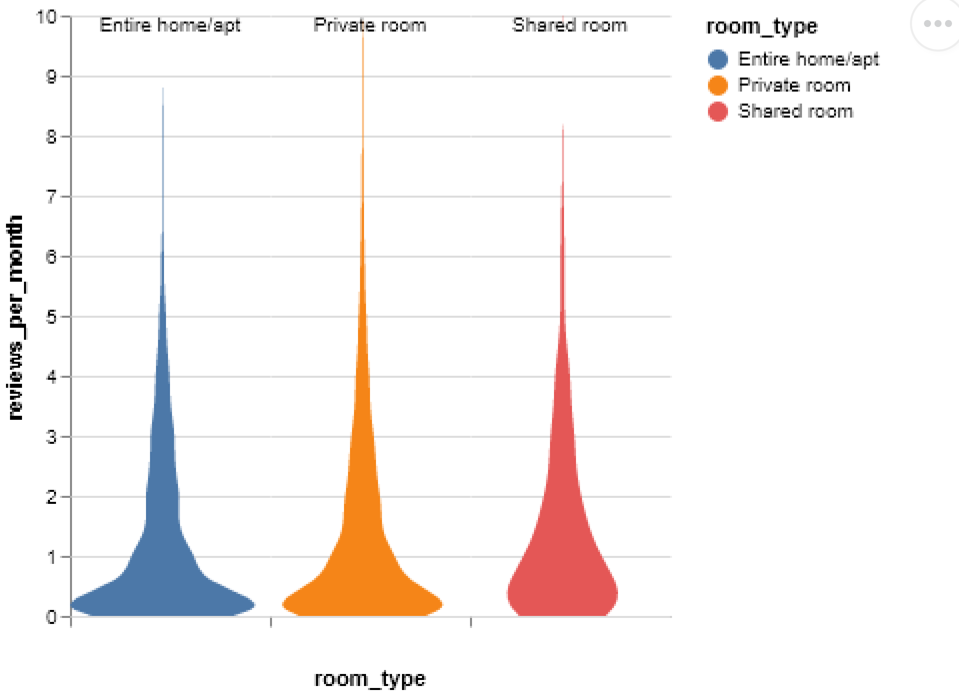
```
# visualize the number of Airbnbs in room-type
order_room = train_df['room_type'].value_counts().index
ax = sns.countplot(x=train_df['room_type'],order=order_room)
ax.set_title('Room Type Distribution')
plt.show()
```



In [31]:

```
# target distribution for room type
alt.Chart(train_df).transform_density(
    'reviews_per_month',
    as_=['reviews_per_month', 'density'],
    extent=[0, 10],
    groupby=['room_type']
).mark_area(orient='horizontal').encode(
    y='reviews_per_month:Q',
    color='room_type:N',
    x=alt.X(
        'density:Q',
        stack='center',
        impute=None,
        title=None,
        axis=alt.Axis(labels=False, values=[0],grid=False, ticks=True),
    ),
    column=alt.Column(
        'room_type:N',
        header=alt.Header(
            titleOrient='bottom',
            labelOrient='bottom',
            labelPadding=0,
        ),
    )
).properties(
    width=100
).configure_facet(
    spacing=0
).configure_view(
    stroke=None
)
```

Out[31]:



Shared room seems to be getting more or atleast 1 review per month. Other room types have a fat base on the violin plot indicating that their reviews per month are close to 0. Perhaps with shared members there is greater possibility that atleast one person will leave a review.

Exploring the Numerical features

In [32]:
```
import numpy as np

# numeric value correlation
numeric_df = train_df.select_dtypes(include=np.number)

numeric_df.corr().style.background_gradient()
```

Out[32]:

	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_m
latitude	1.000000	0.086377	0.033529	0.024805	-0.006874	-0.00
longitude	0.086377	1.000000	-0.157438	-0.053628	0.055004	0.14
price	0.033529	-0.157438	1.000000	0.027509	-0.037135	-0.03
minimum_nights	0.024805	-0.053628	0.027509	1.000000	-0.068136	-0.11
number_of_reviews	-0.006874	0.055004	-0.037135	-0.068136	1.000000	0.54
reviews_per_month	-0.008246	0.142564	-0.030907	-0.116488	0.548508	1.00
calculated_host_listings_count	0.003910	-0.093329	0.054437	0.066996	-0.059084	-0.00
availability_365	-0.017285	0.103867	0.078909	0.098610	0.193951	0.18
days_since_review	0.017736	-0.108290	0.022686	0.053004	-0.283844	-0.44

Based in the target columns `reviews_per_month`, there is a noticeable correlation with the `number_of_reviews`, `longitude`, `minimum_nights`, `availability_365` and `days_since_review`.

Exploring the text feature

In [33]:
```
## name count
```

```
names = train_df['name'].tolist()
word_in_name = []
for name in names:
    name = str(name).split()
    for word in name:
        word_in_name.append(word.lower())
#word_in_name
```

In [34]:
```
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ranji\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[34]: True

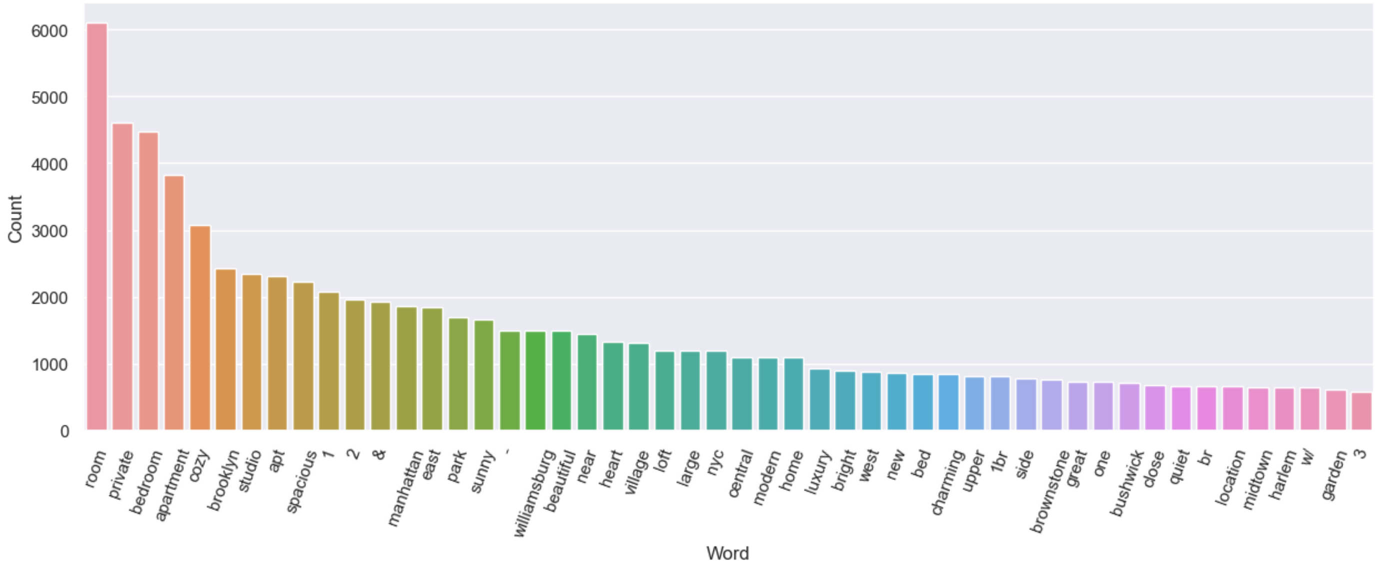
In [35]:
```
word_in_eng = []
for word in word_in_name:
    if word not in stopwords.words('english'):
        word_in_eng.append(word)

#word_in_eng
```

In [36]:
```
# count and get most 50 words
from collections import Counter
count_result = Counter(word_in_eng)
most = count_result.most_common()[:50]
df = pd.DataFrame(most, columns =['Word', 'Count'])
df.head()
```

Out[36]:

	Word	Count
0	room	6095
1	private	4604
2	bedroom	4472
3	apartment	3826
4	cozy	3074

In [37]:
```
fig=plt.figure(figsize=(15,5))
sns.barplot(x='Word', y='Count', data=df)
plt.xticks(rotation=70)
plt.show()
```



EDA Summary

Inference:

1. The EDA shows that the target variable is to be predicted by regression. However, its distribution is highly skewed. Thus the performance of linear regression models may not be good. Non-linear models may also need to be utilized.
2. The features `Room Type` and `Neighbourhood group` are categorical and they are imblanced in their number of counts for each category. This can introduce challenges in the model performance.
3. The `last_review` feature was transformed to a more usable numerical column as `day_since_review`. This should help in improving training and performance.
4. The word analysis on the `name` feature shows that there is a distribution of count of unique words used in the corpus. Sentiment analysis could be tried as a feature engineering measure to see if it helps improve the model.

In [38]:
```
# Splitting the datasets into X and y
```

In [39]:
```
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"]
```

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

Sentiment Analysis

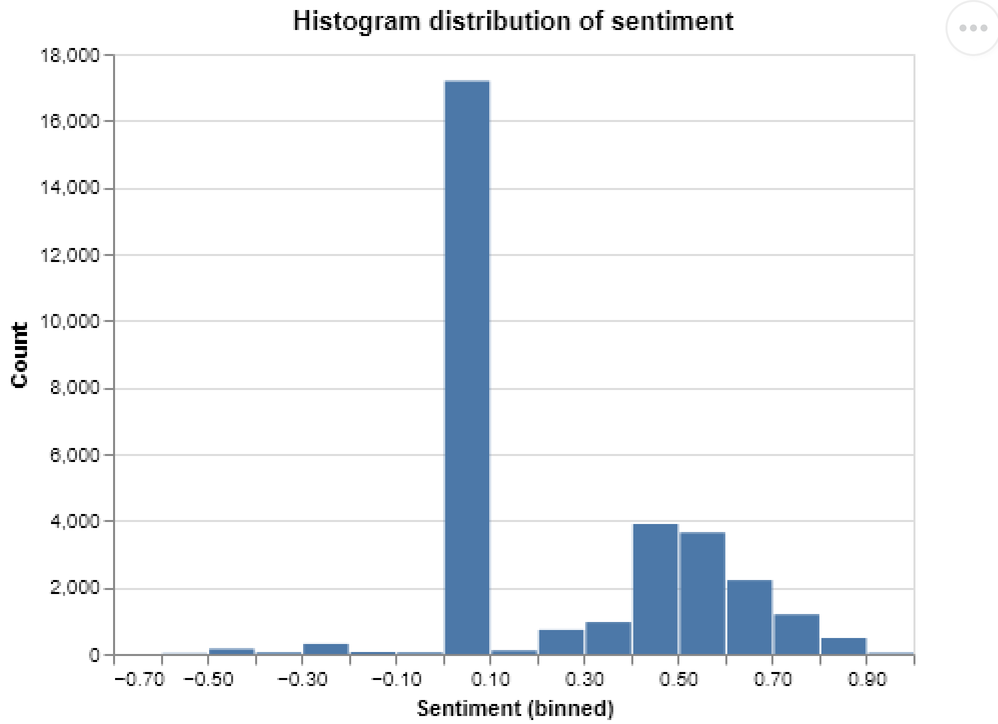
In [41]:
```
# from 573 Lab 2 code sample

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

nltk.download("vader_lexicon")
nltk.download("punkt")

sid = SentimentIntensityAnalyzer()


def get_sentiment(text):
    """
    Returns the compound score representing the sentiment: -1 (most extreme negative) and +1 (mos
    The compound score is a normalized score calculated by summing the valence scores of each wor

    Parameters:
    ------
    text: (str)
    the input text

    Returns:
    -------
    sentiment of the text: (str)
    """
    scores = sid.polarity_scores(text)
    return scores["compound"]
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\ranji\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ranji\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [42]:
```
# from 573 Lab 2 code sample

X_train = X_train.assign(vader_sentiment=train_df["name"].apply(get_sentiment))
X_test = X_test.assign(vader_sentiment=test_df["name"].apply(get_sentiment))
```

In [43]:
```
# Looking at the distribution of the engineered feature

#plt.hist(train_df['vader_sentiment'], bins = 20);

alt.Chart(X_train, title = "Histogram distribution of sentiment").mark_bar().encode(
    x = alt.X('vader_sentiment', bin=alt.Bin(maxbins=30), title = 'Sentiment (binned)'),
    y = alt.Y('count()', title = 'Count')
)
```

Out[43]:



Correlation matrix with the Vader_sentiment feature

In [44]:
```
X_train.corr()
```

Out[44]:

	latitude	longitude	price	minimum_nights	number_of_reviews	calculated_hos
latitude	1.000000	0.086377	0.033529	0.024805	-0.006874	
longitude	0.086377	1.000000	-0.157438	-0.053628	0.055004	
price	0.033529	-0.157438	1.000000	0.027509	-0.037135	
minimum_nights	0.024805	-0.053628	0.027509	1.000000	-0.068136	
number_of_reviews	-0.006874	0.055004	-0.037135	-0.068136	1.000000	
calculated_host_listings_count	0.003910	-0.093329	0.054437	0.066996	-0.059084	
availability_365	-0.017285	0.103867	0.078909	0.098610	0.193951	
days_since_review	0.017736	-0.108290	0.022686	0.053004	-0.283844	
vader_sentiment	-0.035553	-0.039110	-0.019142	0.016665	-0.011381	

Feature Engineering Summary

1. Sentiment analysis was performed on the `name` title to generate the feature `vader_sentiment` with the compound score. The assumption here is that there could be a pattern between a highly positive sentiment listing and the popularity of the property.

2. The feature `days_since_review` is engineered from the `last_review` feature and captures the number of days since the last review for the listing. This feature can be useful since the larger the number of days since last review the lesser is the apparent popularity of a listing. The correlation matrix also supports this line of thinking.

5. Preprocessing and transformations

rubric={accuracy,reasoning}

Your tasks:

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

Points: 4

In [45]:
```
numeric_columns = X_train.select_dtypes(include=np.number).columns.tolist()
print(numeric_columns)
```

```
['latitude', 'longitude', 'price', 'minimum_nights', 'number_of_reviews', 'calculated_host_listi
ngs_count', 'availability_365', 'days_since_review', 'vader_sentiment']
```

In [46]:
```
cols = list(X_train.columns)
cols
```

Out[46]:
```
['name',
 'host_name',
 'neighbourhood_group',
 'neighbourhood',
 'latitude',
 'longitude',
 'room_type',
 'price',
 'minimum_nights',
 'number_of_reviews',
 'calculated_host_listings_count',
 'availability_365',
 'days_since_review',
 'host_name_mod',
 'vader_sentiment']
```

In [47]:
```
# remaining columns
set(cols) - set(numeric_columns)
```

Out[47]:
```
{'host_name',
 'host_name_mod',
 'name',
 'neighbourhood',
 'neighbourhood_group',
 'room_type'}
```

In [48]:
```
numeric_features = numeric_columns
categorical_features = ['neighbourhood', 'neighbourhood_group', 'room_type', 'host_name_mod']
text_feature = "name"
drop_features = ["host_name"]
```

In [49]:
```
#checking that the count of columns matches with the dataframe

assert(len(numeric_features) + len(categorical_features) + len(drop_features) + 1) == len(X_trai
```

In [50]:
```
# Transformations, adding imputations if needed later in the the process

numeric_transformer = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
```

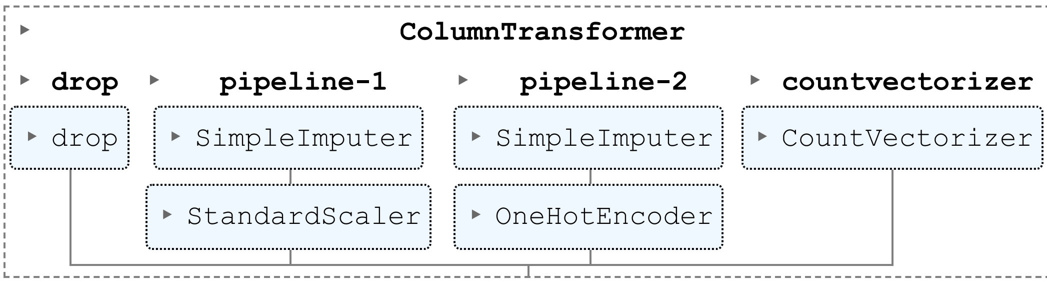
```
categorical_transformer = make_pipeline(
    SimpleImputer(strategy="constant", fill_value="missing"),
    OneHotEncoder(handle_unknown="ignore", sparse = False)
)


preprocessor = make_column_transformer(
    ("drop", drop_features),
    (numeric_transformer, numeric_features),
    (categorical_transformer, categorical_features),
    (CountVectorizer(stop_words="english"), text_feature),
)
```

Final Transformer

In [51]:

```
preprocessor
```

Out[51]:



6. Baseline model

rubric={accuracy}

Your tasks:

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

Points: 2

In [52]:

```
# function adopted from 571
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
    """
```

```python
    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 [53]:

```python
# function to get performance for different model
def model_score(model, X_train, X_test, y_train, y_test):
    """
    Returns the fitted model and the list of train and test scores.

    Parameters:
    ------
    model: (obj)
    the ML molel

    X_train: (dataframe)
    the train dataset

    X_test: (dataframe)
    the test dataset

    y_train: (series)
    the target variable of train data

    y_test: (series)
    the target variable of test data

    Returns:
    -------
    model: (obj)
    the model object fitted on the train data

    results: (list)
    the R2 and rmse scores on the train and test data
    """
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    r2_score_train = r2_score(y_train, y_train_pred)
    r2_score_test = r2_score(y_test, y_test_pred)

    rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
    rmse_test = np.sqrt(mean_squared_error(y_test, y_test_pred))

    results = [r2_score_train, r2_score_test, rmse_train, rmse_test]

    return model, results
```

In [54]:

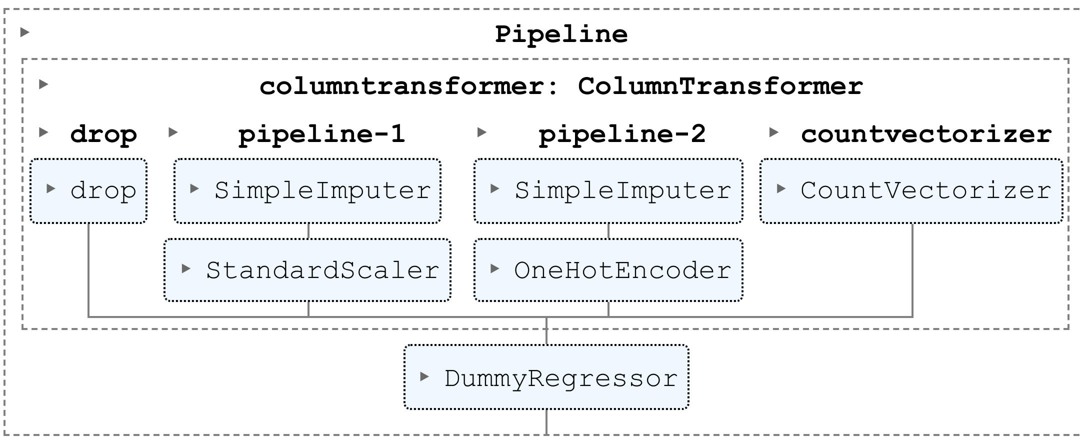
```python
def showResults(results, name):
    """
    Prints the results computed by model_score() function.

    Parameters:
    ------
```

```
    results: (list)
    list of R2 and rmse scores

    Returns:
    -------
    None:
    No value returned
    """
    r2_score_train, r2_score_test, rmse_train, rmse_test = results
    lst = [name + '_r2_score_train_', name+ '_r2_score_test', name+'_rmse_train', name+'_rmse_te
    df_result = pd.DataFrame(list(zip(lst, results)),
                  columns =['Score_Name', 'Score'])

    print('Results:')
    print('R2 score on training set: {:.3f}'.format(r2_score_train))
    print('R2 score on testing set: {:.3f}'.format(r2_score_test))
    print('RMSE on training set: {:.3f}'.format(rmse_train))
    print('RMSE on testing set: {:.3f}'.format(rmse_test))
    return df_result
```

In [55]:

```
from sklearn.dummy import DummyClassifier, DummyRegressor
results_dict = {}

dummy = DummyRegressor()

dummy_pipe = make_pipeline(preprocessor, dummy)
dummy_pipe
```

Out[55]:



In [56]:

```
results_dict["dummy"] = mean_std_cross_val_scores(
    dummy_pipe, X_train, y_train, cv=10, return_train_score=True
)
```

In [57]:

```
pd.DataFrame(results_dict)
```

Out[57]:

	dummy
fit_time	0.431 (+/- 0.028)
score_time	0.050 (+/- 0.002)
test_score	-0.000 (+/- 0.001)
train_score	0.000 (+/- 0.000)

Results with Dummy model

In [58]:

```
model_res, model_stat = model_score(dummy_pipe, X_train, X_test, y_train, y_test)
```

```
showResults(model_stat, 'dummy')
```

```
Results:
R2 score on training set: 0.000
R2 score on testing set: -0.000
RMSE on training set: 1.680
RMSE on testing set: 1.680
```

Out[58]:

	Score_Name	Score
0	dummy_r2_score_train_	0.000000
1	dummy_r2_score_test	-0.000018
2	dummy_rmse_train	1.680407
3	dummy_rmse_test	1.679907

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

Type your answer here, replacing this text.

In [59]:

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso

# first run all default regression by using above function to see performance then decide regula
linear_model = {
    "linear regression": LinearRegression(),
    "ridge":Ridge(alpha=1),
    "lasso": Lasso(alpha=0.1)
}
```

In [60]:

```
for model_name, model in linear_model.items():
    pipe = make_pipeline(preprocessor, model)
    results_dict[model_name] = mean_std_cross_val_scores(
        pipe, X_train, y_train, cv=10, return_train_score=True
    )
```

In [61]:

```
linear_results_df = pd.DataFrame(results_dict).T
linear_results_df
```

Out[61]:

	fit_time	score_time	test_score	train_score
dummy	0.431 (+/- 0.028)	0.050 (+/- 0.002)	-0.000 (+/- 0.001)	0.000 (+/- 0.000)
linear regression	1.924 (+/- 0.269)	0.057 (+/- 0.006)	0.357 (+/- 0.024)	0.586 (+/- 0.006)
ridge	0.591 (+/- 0.013)	0.053 (+/- 0.003)	0.418 (+/- 0.026)	0.565 (+/- 0.005)
lasso	1.326 (+/- 0.019)	0.053 (+/- 0.003)	0.393 (+/- 0.028)	0.390 (+/- 0.004)

Inference:

Overall the linear models are not performing well as the performance on the validation set is below 0.5 R-squared score. The Ridge linear regression performs slightly better compared to others and its test scores are slightly better. We will try to optimize the hyperparameters for Ridge to see how much improvement can be achieved.

In [62]:
```
from sklearn.linear_model import RidgeCV
alphas = 10.0 ** np.arange(-6, 6, 1)
ridgecv_pipe = make_pipeline(preprocessor, RidgeCV(alphas=alphas, cv=10))
ridgecv_pipe.fit(X_train, y_train);

best_alpha = ridgecv_pipe.named_steps["ridgecv"].alpha_

best_alpha
```

Out[62]: 100.0

In [63]:
```
ridge_tuned = make_pipeline(preprocessor, Ridge(alpha=best_alpha))

results_dict["ridge_tuned"] = mean_std_cross_val_scores(
        ridge_tuned, X_train, y_train, cv=10, return_train_score=True
    )
```

In [64]:
```
pd.DataFrame(results_dict).T
```

Out[64]:

	fit_time	score_time	test_score	train_score
dummy	0.431 (+/- 0.028)	0.050 (+/- 0.002)	-0.000 (+/- 0.001)	0.000 (+/- 0.000)
linear regression	1.924 (+/- 0.269)	0.057 (+/- 0.006)	0.357 (+/- 0.024)	0.586 (+/- 0.006)
ridge	0.591 (+/- 0.013)	0.053 (+/- 0.003)	0.418 (+/- 0.026)	0.565 (+/- 0.005)
lasso	1.326 (+/- 0.019)	0.053 (+/- 0.003)	0.393 (+/- 0.028)	0.390 (+/- 0.004)
ridge_tuned	0.454 (+/- 0.010)	0.057 (+/- 0.003)	0.444 (+/- 0.031)	0.463 (+/- 0.004)

Results Linear Model

In [110…
```
model_res, model_stat = model_score(ridge_tuned, X_train, X_test, y_train, y_test)
showResults(model_stat, 'Ridge')
```

```
Results:
R2 score on training set: 0.463
R2 score on testing set: 0.451
RMSE on training set: 1.232
RMSE on testing set: 1.244
```

Out[110]:

	Score_Name	Score
0	Ridge_r2_score_train_	0.462909
1	Ridge_r2_score_test	0.451228
2	Ridge_rmse_train	1.231511
3	Ridge_rmse_test	1.244451

Summary

- Slight improvement is obtained in the performance of the Ridge linear regression model through hyperparameter tuning.
- However it appears that the relationship between the features that are not well captured by linear models. Hence let's try the non-linear models next.

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

Non-Linear and Ensemble Models

In [66]:
```
# try different models with default parameter for decision tree and random forest
other_model = {
    "decision tree": DecisionTreeRegressor(),
    "random forest":RandomForestRegressor(),
    "GradientBoosting": GradientBoostingRegressor(),
    "lgbm": LGBMRegressor(),
    "xgboost": XGBRegressor()
}
```

In [67]:
```
# this will take some time

for model_name, model in other_model.items():
    pipe = make_pipeline(preprocessor, model)
    results_dict[model_name] = mean_std_cross_val_scores(
        pipe, X_train, y_train, cv=5, n_jobs =-1, return_train_score=True
    )
```

In [68]:
```
all_results_df = pd.DataFrame(results_dict).T
all_results_df
```

Out[68]:

	fit_time	score_time	test_score	train_score
dummy	0.431 (+/- 0.028)	0.050 (+/- 0.002)	-0.000 (+/- 0.001)	0.000 (+/- 0.000)
linear regression	1.924 (+/- 0.269)	0.057 (+/- 0.006)	0.357 (+/- 0.024)	0.586 (+/- 0.006)
ridge	0.591 (+/- 0.013)	0.053 (+/- 0.003)	0.418 (+/- 0.026)	0.565 (+/- 0.005)
lasso	1.326 (+/- 0.019)	0.053 (+/- 0.003)	0.393 (+/- 0.028)	0.390 (+/- 0.004)
ridge_tuned	0.454 (+/- 0.010)	0.057 (+/- 0.003)	0.444 (+/- 0.031)	0.463 (+/- 0.004)
decision tree	8.558 (+/- 0.096)	0.110 (+/- 0.022)	0.389 (+/- 0.043)	1.000 (+/- 0.000)
random forest	525.944 (+/- 1.115)	0.302 (+/- 0.022)	0.657 (+/- 0.014)	0.951 (+/- 0.003)
GradientBoosting	9.318 (+/- 0.105)	0.125 (+/- 0.006)	0.630 (+/- 0.025)	0.682 (+/- 0.005)
lgbm	1.132 (+/- 0.017)	0.144 (+/- 0.004)	0.659 (+/- 0.034)	0.746 (+/- 0.008)
xgboost	3.650 (+/- 0.063)	0.130 (+/- 0.024)	0.645 (+/- 0.024)	0.786 (+/- 0.004)

Summary

The non-linear models are performing better than the linear models in general.

- Decision Tree is severely overfitting with a perfect train score and a poor validation score.
- The Random Forest model is performing decently, however it is still suffering from overfitting.
- GradientBoosting Machine model adds base learners sequentially to progressively reduce error in the model. It performed well and the train and validation scores have the least difference. However the overall validation score is decent around 63%
- Light Gradient Boosted Machine, is a faster implementation of GradientBoosting. This is evident from the dramatically smaller fit time. The performance is also improved slightly wit the validation score around 65%.
- XGBoost is extreme GradientBoosting which is computationally faster and gives a better performance. In this case we see that it's fit time lies between that of GradientBoosting and LGBM, the performance is improved over GradientBoosting and the validation score is around 64%.

9. Feature selection (Challenging)

rubric={reasoning}

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 unless you think there are other benefits with using less features.

Points: 0.5

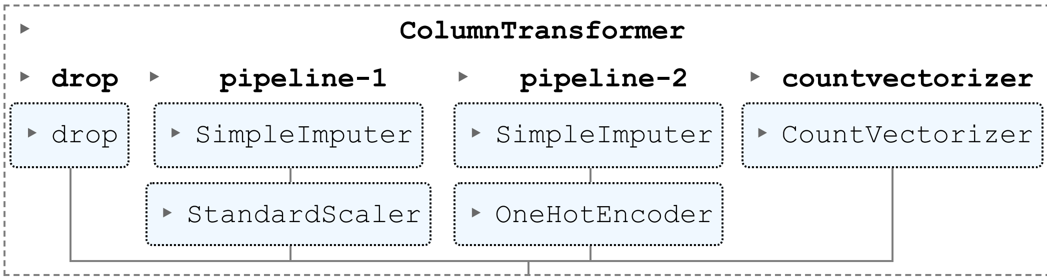
Intuition

Even though linear model performance is not good for this case study, linear models have the advnatage of being very interpretable. This helps in performing and interpreting selection. In this case we are using the linear regression to perform Feature Selection though the LassoCV model. The big benefit of LassoCV is that it enables feature selection by driving the coefficients of less important features to zero.

Determine the original number of features in model

In [69]:
```
# The number of features in the model

preprocessor.fit(X_train, y_train)
```

Out[69]:



In [70]:
```
numeric_feature_names = numeric_features
ohe_feature_names = list(preprocessor.fit(X_train, y_train).named_transformers_['pipeline-2'].ge
categorical_feature_names = list(preprocessor.fit(X_train, y_train).named_transformers_['countve
```

In [71]:
```
feature_name_list = numeric_feature_names + ohe_feature_names + categorical_feature_names
```

In [72]:
```
# Total number of features in the model

n_features_original = len(feature_name_list)
n_features_original
```

Out[72]: 6757

In [73]:
```
# Code in this section is adapted from Lab3 exercises

lassoCV_pipe = make_pipeline(
preprocessor,
LassoCV(max_iter= 200, tol = 0.01)
)
```

In [74]:
```
cv_df = pd.DataFrame(
cross_validate(
lassoCV_pipe,
X_train,
y_train,
return_train_score = True,
cv= 10,
n_jobs = -1,
scoring = 'r2'
)
)
```

In [75]:
```
cross_val_results = {}
cross_val_results['lassoCV'] = cv_df.agg(['mean', 'std']).round(3).T
cross_val_results['lassoCV']
```

Out[75]:

	mean	std
fit_time	241.024	3.125
score_time	0.059	0.009
test_score	0.438	0.029
train_score	0.446	0.003

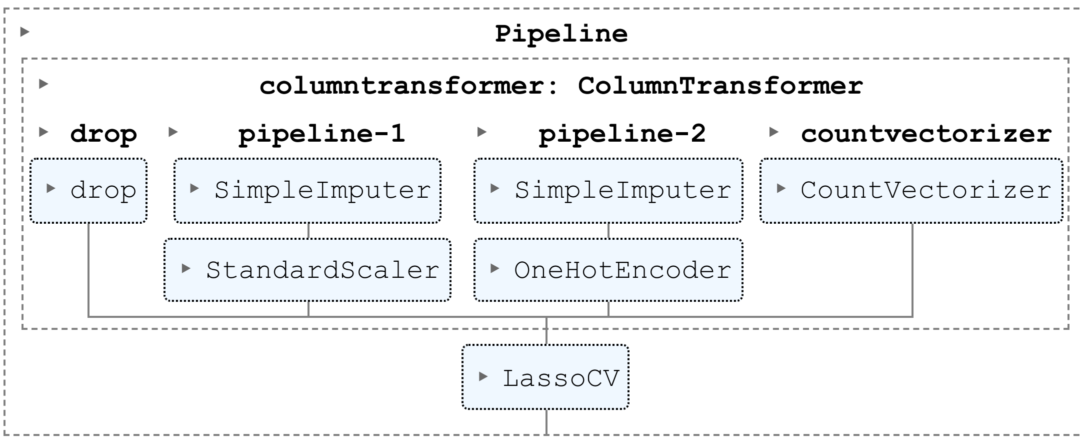
In [76]:
```
# number of coefficients
lassoCV_model = lassoCV_pipe.fit(X_train, y_train)
```

In [77]:
```
n_coefs_nonzero_lasso = np.count_nonzero(lassoCV_model.named_steps['lassocv'].coef_)
n_coefs_nonzero_lasso
```

Out[77]: 160

In [78]:
```
lassoCV_pipe
```

Out[78]:



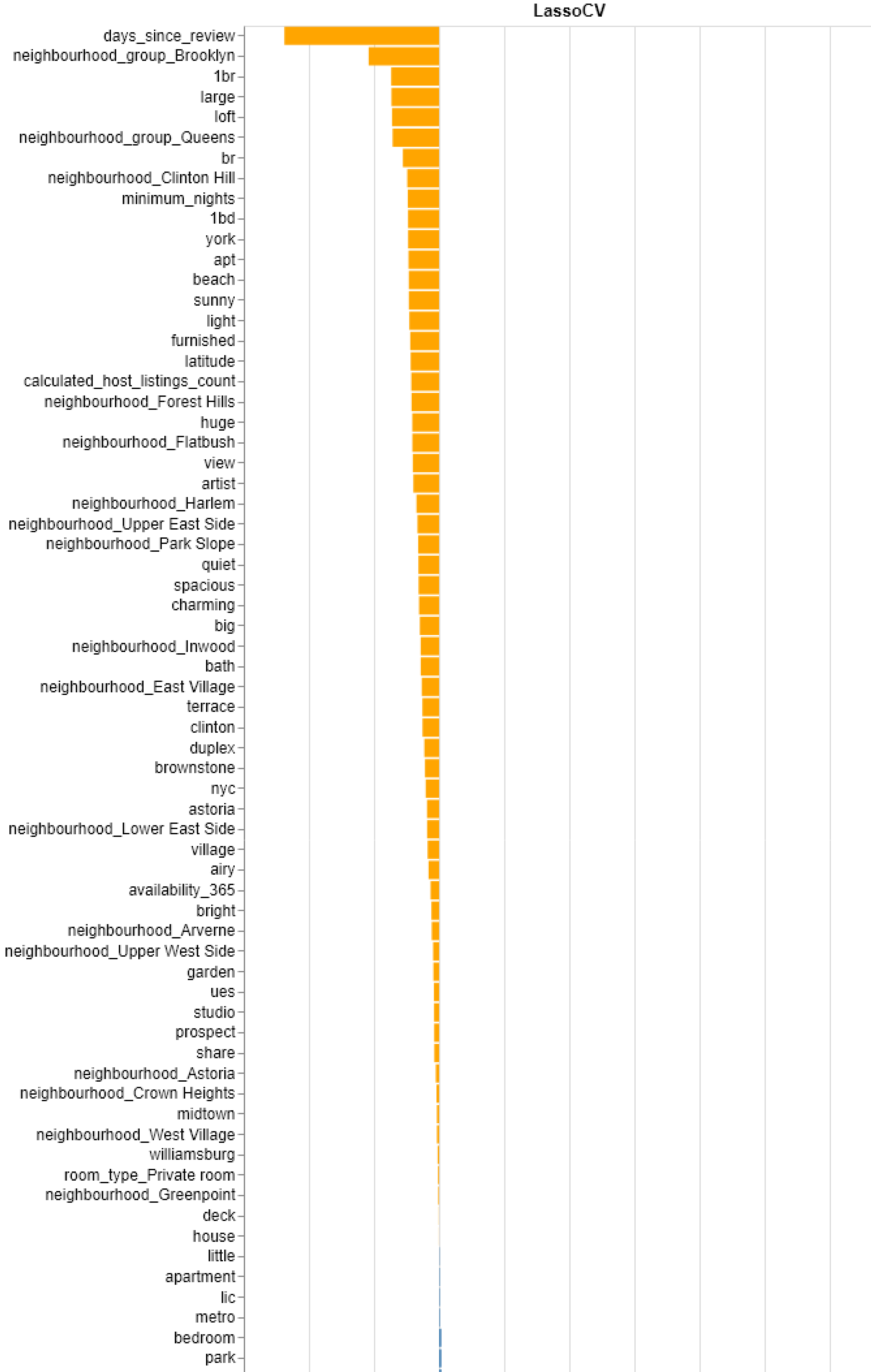
In [79]:
```
# Get the feature names
coefs_lasso = pd.DataFrame(
{'variable' : feature_name_list,
'coef': list(lassoCV_model.named_steps['lassocv'].coef_)
}
)
```

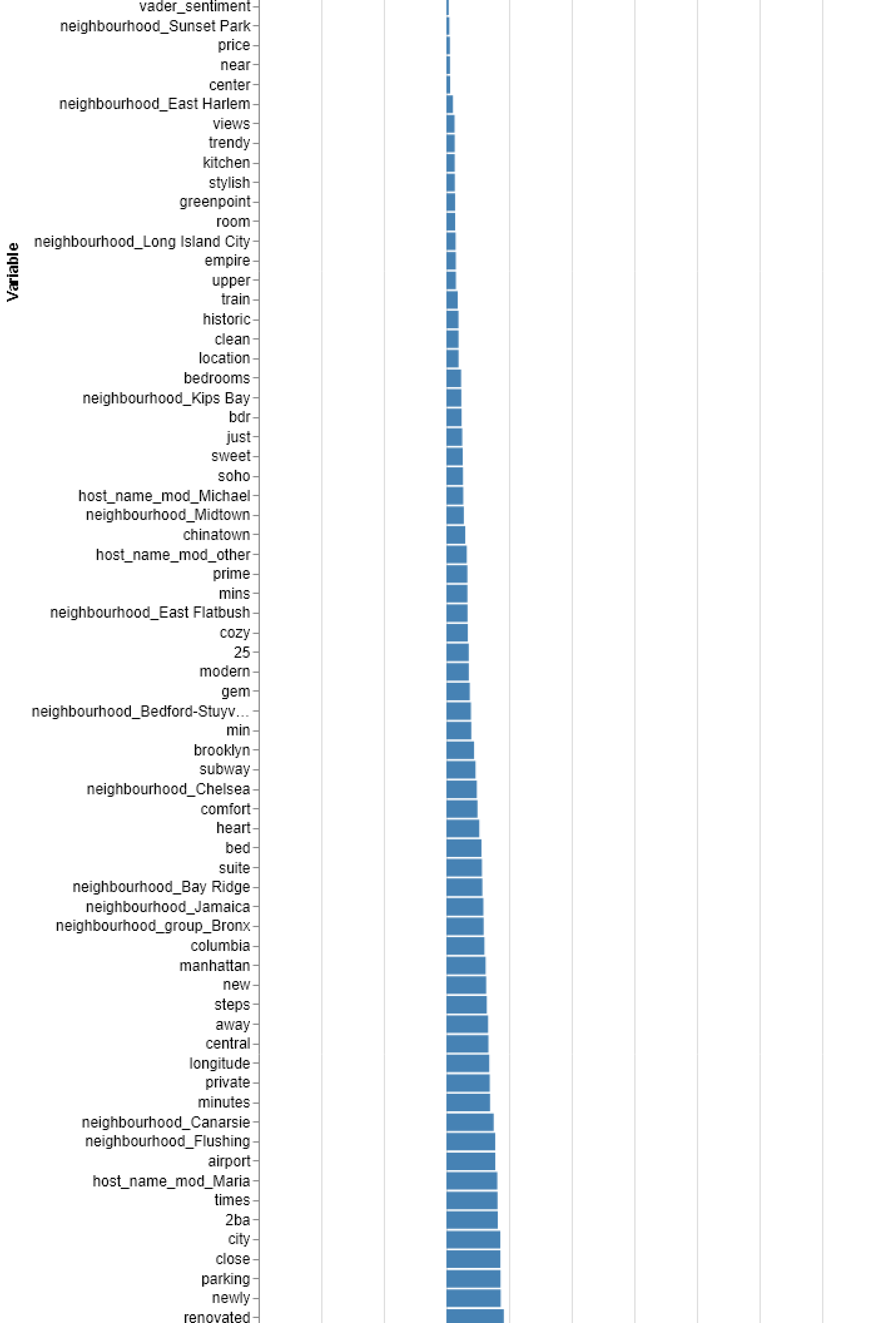
In [80]:
```
# filtering the dataframe where coefficients are not 0
coefs_lasso_filter = coefs_lasso[coefs_lasso['coef'] != 0]
```

In [81]:
```
# Visualize the coeffcients
coefs_barplot_lasso = alt.Chart(
coefs_lasso_filter,
title = 'LassoCV'
).mark_bar().encode(
x = alt.X('coef', title = 'Coefficients'),
y = alt.Y('variable', title = 'Variable', sort = 'x'),
color = alt.condition(
alt.datum.coef > 0,
alt.value("steelblue"), # Positive color
alt.value("orange") # Negative color
)
).properties(
width = 500,
height = 2500
).configure_axis(
labelFontSize = 12,
```

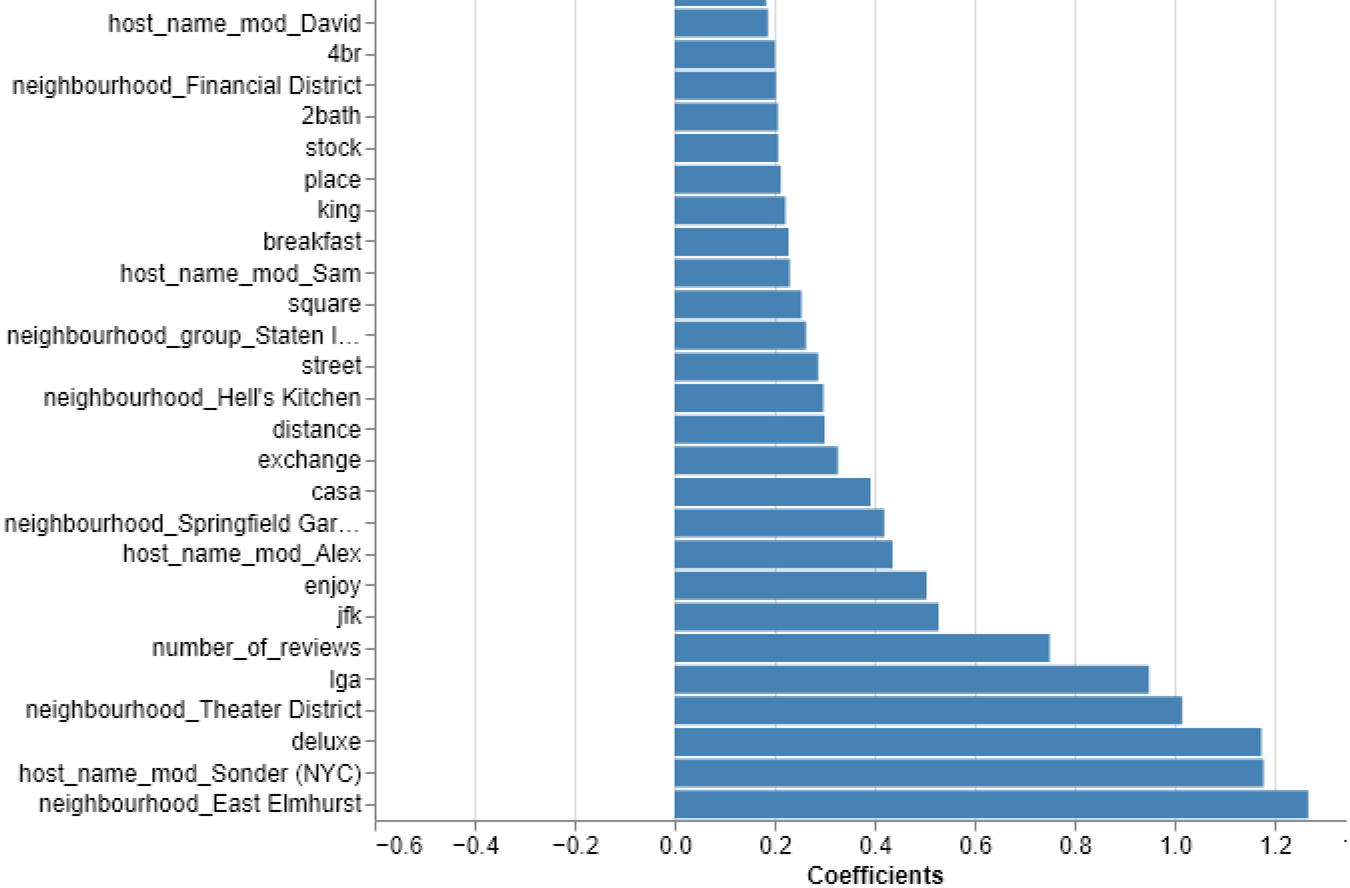
```
titleFontSize = 12
)
coefs_barplot_lasso
```

Out[81]:


LassoCV
days_since_review
neighbourhood_group_Brooklyn
1br
large
loft
neighbourhood_group_Queens
br
neighbourhood_Clinton Hill
minimum_nights
1bd
york
apt
beach
sunny
light
furnished
latitude
calculated_host_listings_count
neighbourhood_Forest Hills
huge
neighbourhood_Flatbush
view
artist
neighbourhood_Harlem
neighbourhood_Upper East Side
neighbourhood_Park Slope
quiet
spacious
charming
big
neighbourhood_Inwood
bath
neighbourhood_East Village
terrace
clinton
duplex
brownstone
nyc
astoria
neighbourhood_Lower East Side
village
airy
availability_365
bright
neighbourhood_Arverne
neighbourhood_Upper West Side
garden
ues
studio
prospect
share
neighbourhood_Astoria
neighbourhood_Crown Heights
midtown
neighbourhood_West Village
williamsburg
room_type_Private room
neighbourhood_Greenpoint
deck
house
little
apartment
lic
metro
bedroom
park



Variable
vader_sentiment
neighbourhood_Sunset Park
price
near
center
neighbourhood_East Harlem
views
trendy
kitchen
stylish
greenpoint
room
neighbourhood_Long Island City
empire
upper
train
historic
clean
location
bedrooms
neighbourhood_Kips Bay
bdr
just
sweet
soho
host_name_mod_Michael
neighbourhood_Midtown
chinatown
host_name_mod_other
prime
mins
neighbourhood_East Flatbush
cozy
25
modern
gem
neighbourhood_Bedford-Stuyv…
min
brooklyn
subway
neighbourhood_Chelsea
comfort
heart
bed
suite
neighbourhood_Bay Ridge
neighbourhood_Jamaica
neighbourhood_group_Bronx
columbia
manhattan
new
steps
away
central
longitude
private
minutes
neighbourhood_Canarsie
neighbourhood_Flushing
airport
host_name_mod_Maria
times
2ba
city
close
parking
newly
renovated



Inference

1. The LassoCV is eliminates the features drastically by assigning a zero coefficient to most of the less useful features. The initial features were was 6700 (approx). After LassoCV there are only 160 features with non-zero coefficients.

2. The features with coefficients and showing positive relationship with the target variable are interpretable. They provide some interesting insights. The popularity of the listing may be high in certain neighbourhoods such as Theater District, jfk, East Elmhurst etc. It also shows certain hosts with name Sonder and Sam tend to get high reviews.

3. Since LassoCV is a linear model, there is a high risk that the features with non-linear relationships with the target variable might have been assigned a zero coefficient. If used for model selection, these features will get eliminated.

However we are prepared to experiment with this and next step we test the performance of LassoCV for model selection and LGBM for performance on the selected features.

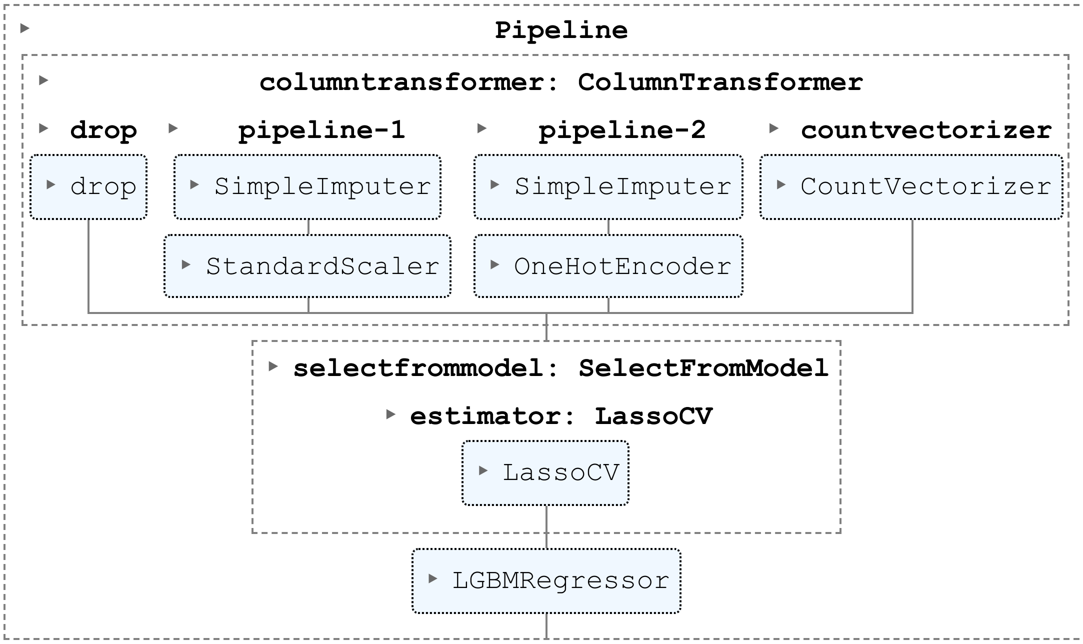
Feature Selection LassoCV and LGBM Model

In [82]:

```
# LassoCV and bestLGBM

lasso_lgbm = make_pipeline(
preprocessor,
    SelectFromModel(
        LassoCV(
            max_iter= 200,
            n_jobs = -1,
            tol = 0.01
```

```
        )
    ),
LGBMRegressor()
)

lasso_lgbm
```

Out[82]:



In [83]:

```
cv_df = pd.DataFrame(
cross_validate(
    lasso_lgbm,
    X_train,
    y_train,
    return_train_score = True,
    cv= 5,
    n_jobs = -1,
    scoring = 'r2'
)
)
```

In [84]:

```
cross_val_results['lassoCV_lgbm'] = cv_df.agg(['mean', 'std']).round(3).T
pd.concat(cross_val_results, axis = 1)
```

Out[84]:

	lassoCV		lassoCV_lgbm	
	mean	std	mean	std
fit_time	241.024	3.125	90.501	1.524
score_time	0.059	0.009	0.128	0.011
test_score	0.438	0.029	0.659	0.033
train_score	0.446	0.003	0.746	0.007

Summary

1. The performance of LGBM post feature selection is similar to its performance without any feature selection. Thus we conclude that this version of feature selection is not playing a significant role in influencing performance of the model.

2. Since the performance did not decrease either, we conclude LassoCV did not inadvertently remove significant non-linearly related features. There could also be a possibility that there are no influential non-linear features yet present in the dataset.

3. Perhaps non-linear features are needed to further improve the overall performance. This could be done in feature engineering with more domain knowledge that what is currently available.

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.

- GridSearchCV
- RandomizedSearchCV
- scikit-optimize

Points: 6

Intuition

- GradientBoostingRegressor performs decently and gives the least overfitting. However if takes a lot of time for computation.
- LGBM gives a very comparable performance and is much faster.
- We chose to perform hyperparameter optimization on both these models to investigate if even after tuning, are their performances similar? If yes, then we can choose LGBM because of its benefit on computation time. However in real world this may not always be the priority.

GradientBoosting Hyperparameter tuning

In [85]:
```
GradientBoosting_pipe = make_pipeline(preprocessor, GradientBoostingRegressor())

param_dist = {
    "gradientboostingregressor__n_estimators": [750, 1000, 1200],
    "gradientboostingregressor__max_depth": [1,3,5,7,9],
    "gradientboostingregressor__learning_rate": [0.01,0.1,1,10]
}

random_search_gb = RandomizedSearchCV(
    GradientBoosting_pipe,
    param_distributions=param_dist,
    n_iter=10,
    verbose=1,
    n_jobs=-1,
```

```
    cv=5,
    random_state=123
)

random_search_gb.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
C:\Users\ranji\miniconda3\envs\573\lib\site-packages\sklearn\model_selection\_search.py:953: Use
rWarning: One or more of the test scores are non-finite: [0.66247462 0.42295363 0.58779    0.565
17075 0.41169286        nan
 0.63833353 0.38321356        nan 0.66639493]
  warnings.warn(
```

Out[85]:

```
RandomizedSearchCV
estimator: Pipeline
columntransformer: ColumnTransformer
drop | pipeline-1 | pipeline-2 | countvectorizer
drop | SimpleImputer | SimpleImputer | CountVectorizer
     | StandardScaler | OneHotEncoder |
GradientBoostingRegressor
```

In [86]:
```
best_parameters_gradient_boosting = random_search_gb.best_params_
best_parameters_gradient_boosting
```

Out[86]:
```
{'gradientboostingregressor__n_estimators': 1000,
 'gradientboostingregressor__max_depth': 9,
 'gradientboostingregressor__learning_rate': 0.01}
```

In [87]:
```
results_dict["best_GradientBoosting"] = mean_std_cross_val_scores(
        random_search_gb.best_estimator_, X_train, y_train, cv=5, n_jobs =-1, return_train_score=
    )
```

Performance Summary

In [88]:
```
pd.DataFrame(results_dict).T
```

Out[88]:

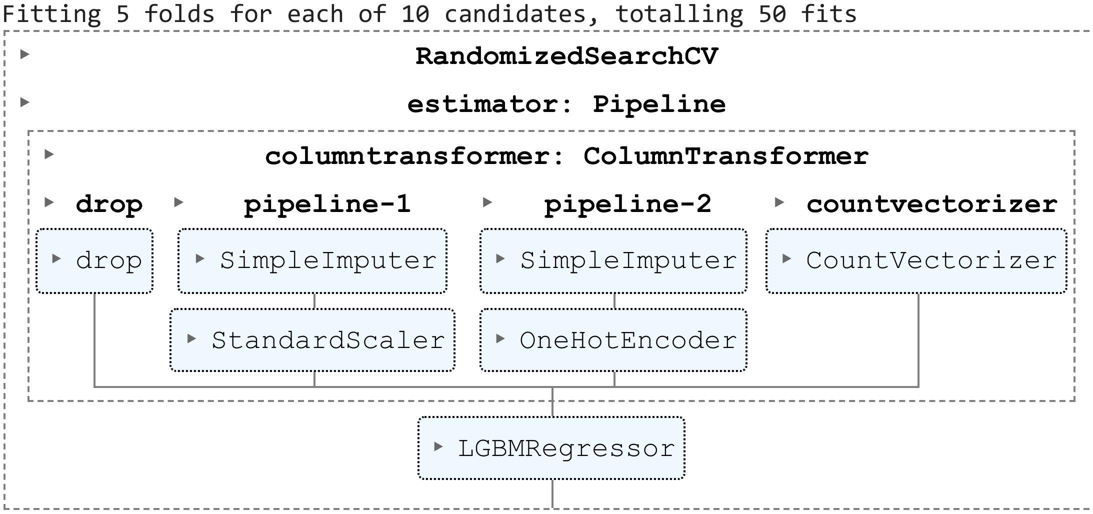
	fit_time	score_time	test_score	train_score
dummy	0.431 (+/- 0.028)	0.050 (+/- 0.002)	-0.000 (+/- 0.001)	0.000 (+/- 0.000)
linear regression	1.924 (+/- 0.269)	0.057 (+/- 0.006)	0.357 (+/- 0.024)	0.586 (+/- 0.006)
ridge	0.591 (+/- 0.013)	0.053 (+/- 0.003)	0.418 (+/- 0.026)	0.565 (+/- 0.005)
lasso	1.326 (+/- 0.019)	0.053 (+/- 0.003)	0.393 (+/- 0.028)	0.390 (+/- 0.004)
ridge_tuned	0.454 (+/- 0.010)	0.057 (+/- 0.003)	0.444 (+/- 0.031)	0.463 (+/- 0.004)
decision tree	8.558 (+/- 0.096)	0.110 (+/- 0.022)	0.389 (+/- 0.043)	1.000 (+/- 0.000)
random forest	525.944 (+/- 1.115)	0.302 (+/- 0.022)	0.657 (+/- 0.014)	0.951 (+/- 0.003)
GradientBoosting	9.318 (+/- 0.105)	0.125 (+/- 0.006)	0.630 (+/- 0.025)	0.682 (+/- 0.005)
lgbm	1.132 (+/- 0.017)	0.144 (+/- 0.004)	0.659 (+/- 0.034)	0.746 (+/- 0.008)
xgboost	3.650 (+/- 0.063)	0.130 (+/- 0.024)	0.645 (+/- 0.024)	0.786 (+/- 0.004)
best_GradientBoosting	385.789 (+/- 1.485)	0.921 (+/- 0.098)	0.665 (+/- 0.017)	0.838 (+/- 0.003)

LGBM Hyperparameter tuning

In [89]:
```
lgbm_pipe = make_pipeline(preprocessor, LGBMRegressor())

param_dist = {
    "lgbmregressor__num_leaves": [8, 25, 50, 75, 100, 150, 200, 350, 500],
    "lgbmregressor__max_depth": [1,3,5,7,9],
    "lgbmregressor__learning_rate": [0.01,0.1,1,10],
    "lgbmregressor__n_estimators": [750, 1000, 1200]
}

random_search_lgbm = RandomizedSearchCV(
    lgbm_pipe,
    param_distributions=param_dist,
    n_iter=10,
    verbose=1,
    n_jobs=-1,
    cv=5,
    random_state=123
)

random_search_lgbm.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[89]:



In [90]:
```python
best_parameters_lgbm = random_search_lgbm.best_params_
best_parameters_lgbm
```

Out[90]:
```
{'lgbmregressor__num_leaves': 200,
 'lgbmregressor__n_estimators': 1000,
 'lgbmregressor__max_depth': 9,
 'lgbmregressor__learning_rate': 0.1}
```

In [91]:
```python
results_dict["best_lgbm"] = mean_std_cross_val_scores(
        random_search_lgbm.best_estimator_, X_train, y_train, cv=5, n_jobs =-1, return_train_scoı
    )
```

Performance Summary

In [92]:
```python
pd.DataFrame(results_dict).T
```

Out[92]:

	fit_time	score_time	test_score	train_score
dummy	0.431 (+/- 0.028)	0.050 (+/- 0.002)	-0.000 (+/- 0.001)	0.000 (+/- 0.000)
linear regression	1.924 (+/- 0.269)	0.057 (+/- 0.006)	0.357 (+/- 0.024)	0.586 (+/- 0.006)
ridge	0.591 (+/- 0.013)	0.053 (+/- 0.003)	0.418 (+/- 0.026)	0.565 (+/- 0.005)
lasso	1.326 (+/- 0.019)	0.053 (+/- 0.003)	0.393 (+/- 0.028)	0.390 (+/- 0.004)
ridge_tuned	0.454 (+/- 0.010)	0.057 (+/- 0.003)	0.444 (+/- 0.031)	0.463 (+/- 0.004)
decision tree	8.558 (+/- 0.096)	0.110 (+/- 0.022)	0.389 (+/- 0.043)	1.000 (+/- 0.000)
random forest	525.944 (+/- 1.115)	0.302 (+/- 0.022)	0.657 (+/- 0.014)	0.951 (+/- 0.003)
GradientBoosting	9.318 (+/- 0.105)	0.125 (+/- 0.006)	0.630 (+/- 0.025)	0.682 (+/- 0.005)
lgbm	1.132 (+/- 0.017)	0.144 (+/- 0.004)	0.659 (+/- 0.034)	0.746 (+/- 0.008)
xgboost	3.650 (+/- 0.063)	0.130 (+/- 0.024)	0.645 (+/- 0.024)	0.786 (+/- 0.004)
best_GradientBoosting	385.789 (+/- 1.485)	0.921 (+/- 0.098)	0.665 (+/- 0.017)	0.838 (+/- 0.003)
best_lgbm	4.522 (+/- 0.198)	0.597 (+/- 0.062)	0.662 (+/- 0.025)	0.880 (+/- 0.021)

Summary

- Increasing the number of estimators is helpful in improving the training and validation scores.
- However after the benefits in performance gain diminish after a certain number of estimators.
- Based on our trials we have chosen the appropriate range of parameters for the `n_estimators` for both the models.
- Based on the above results the best model chosen is **best_lgbm** due to similar performance with less overfitting.

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

Type your answer here, replacing this text.

In [93]:
```
import eli5

ohe_feature_names = (random_search_lgbm.best_estimator_["columntransformer"]
.named_transformers_["pipeline-2"]
.named_steps["onehotencoder"]
.get_feature_names_out(categorical_features).tolist()
                    )

text_feature_names = (random_search_lgbm.best_estimator_["columntransformer"]
    .named_transformers_["countvectorizer"]
    .get_feature_names_out().tolist()
)

feature_names = numeric_features + ohe_feature_names + text_feature_names
```

In [94]:
```
eli5.explain_weights(
    random_search_lgbm.best_estimator_.named_steps["lgbmregressor"], feature_names=feature_names
)
```

Out[94]:

Weight	Feature
0.5029	days_since_review
0.1810	number_of_reviews
0.0850	minimum_nights
0.0388	availability_365
0.0340	longitude
0.0226	latitude
0.0220	price
0.0186	calculated_host_listings_count
0.0085	vader_sentiment
0.0059	neighbourhood_Theater District
0.0051	neighbourhood_East Elmhurst
0.0050	deluxe
0.0043	enjoy
0.0029	lga
0.0025	host_name_mod_other
0.0022	room_type_Entire home/apt
0.0021	room
0.0019	jfk
0.0018	neighbourhood_group_Brooklyn
0.0015	room_type_Private room
	... 6737 more ...

Inference

The most influential features in the best model are the `days_since_review`, `number_of_reviews` and `minimum_nights`. However, the coefficients are not as interpretable as the ones obtained in linear models.

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

In [95]:
```
# Code below is adapted from Lecture 8 in 573
```

In [96]:
```
X_train_enc = pd.DataFrame(
    data=preprocessor.fit_transform(X_train).toarray(),
    columns=feature_names,
    index=X_train.index,
)
X_train_enc.head()
```

Out[96]:

	latitude	longitude	price	minimum_nights	number_of_reviews	calculated_host_listings_count	availal
21151	0.630471	-0.344654	-0.126578	-0.157628	0.781433	-0.157286	
34227	-2.008246	0.738502	-0.292923	-0.212078	0.138213	-0.157286	
19218	0.540472	0.494153	-0.480060	-0.212078	1.632143	-0.081294	
2008	-0.514797	0.676717	-0.012217	-0.157628	1.590645	-0.157286	
31911	1.450832	-0.003339	0.434833	-0.212078	-0.546505	-0.157286	

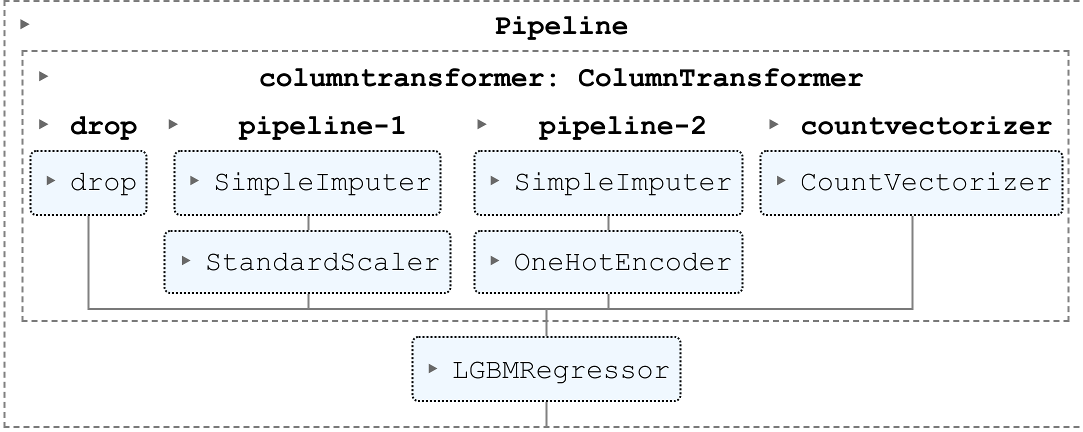
5 rows × 6757 columns

In [97]:
```
X_test_enc = pd.DataFrame(
    data=preprocessor.transform(X_test).toarray(),
    columns=feature_names,
    index=X_test.index,
)
X_test_enc.shape
```

Out[97]:
```
(7765, 6757)
```

In [98]:
```
lgbm_pipe.fit(X_train, y_train)
```

Out[98]:



Type your answer here, replacing this text.

In [100…

```
score_type, score = model_score(random_search_lgbm.best_estimator_, X_train, X_test, y_train, y_
```

In [101…

```
non_linear_result = showResults(score, 'best_lgbm')
non_linear_result
```

```
Results:
R2 score on training set: 0.878
R2 score on testing set: 0.678
RMSE on training set: 0.588
RMSE on testing set: 0.953
```

Out[101]:

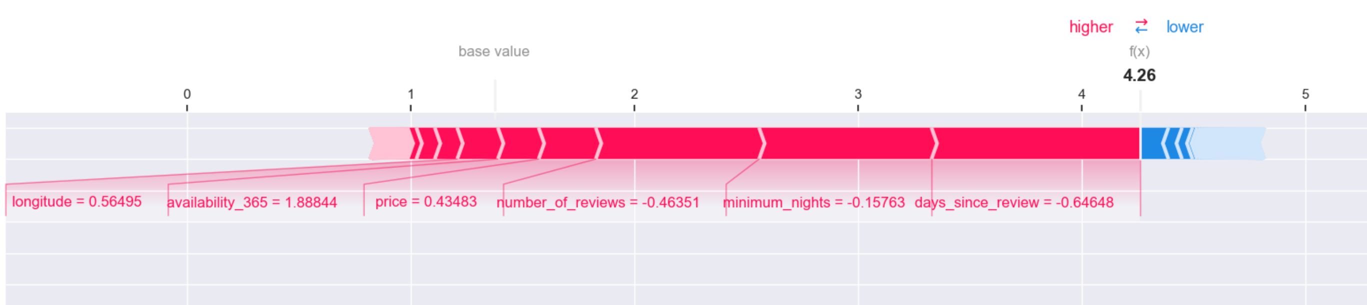
	Score_Name	Score
0	best_lgbm_r2_score_train_	0.877671
1	best_lgbm_r2_score_test	0.678464
2	best_lgbm_rmse_train	0.587733
3	best_lgbm_rmse_test	0.952569

In [102…

```
import shap

lgbm_explainer = shap.TreeExplainer(random_search_lgbm.best_estimator_.named_steps["lgbmregresso
train_lgbm_shap_values = lgbm_explainer.shap_values(X_train_enc)
test_lgbm_shap_values = lgbm_explainer.shap_values(X_test_enc)
```

In [103…

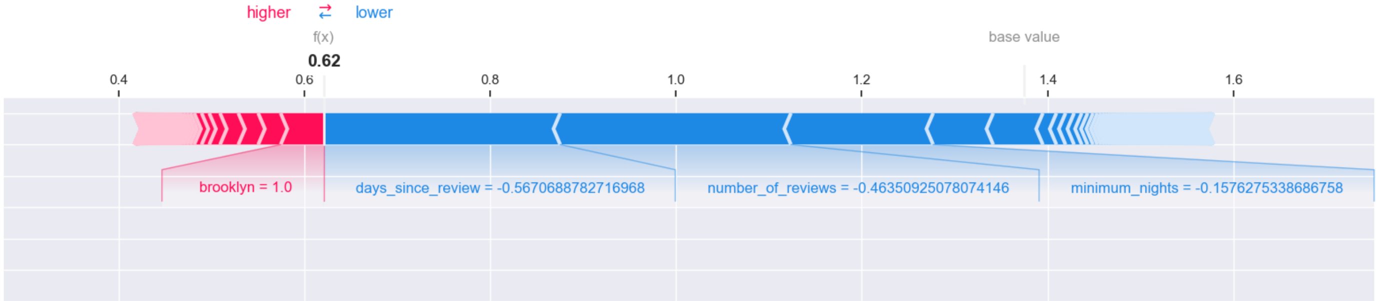
```
shap.force_plot(lgbm_explainer.expected_value, test_lgbm_shap_values[0], X_test_enc.iloc[3,:].rou
```



From above plot, we can see the first example in our test-data in which the features: longitude, availability_365, price, number of reviews, minimum nights, days since review are making the predicted value for reviews_per_month higher than the base value.

In [104…

```
shap.force_plot(lgbm_explainer.expected_value, test_lgbm_shap_values[-1], X_test_enc.iloc[-1,:],
```



Inference

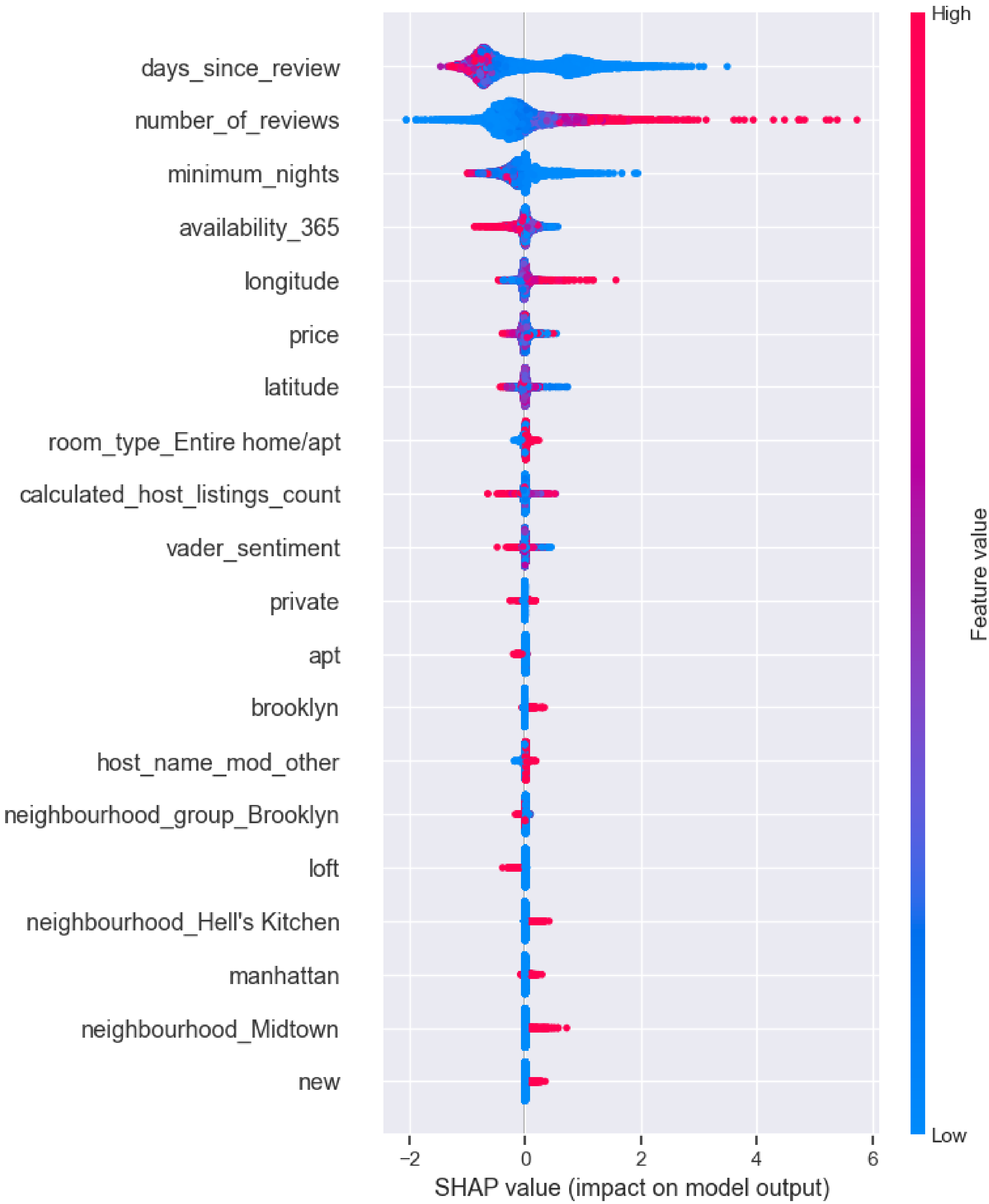
From above plot, we can see the last example in our test-data in which the neighbourhood as brooklyn is making the predicted value for reviews_per_month higher and days since review, number of reviews, and minimum nights are making our prediction smaller than the base value.

In conclusion, we can see number of reviews and days since reviews are two biggest factors to determine the number of reviews per month which is also matching with above result in feature importance section's result.

In [105…

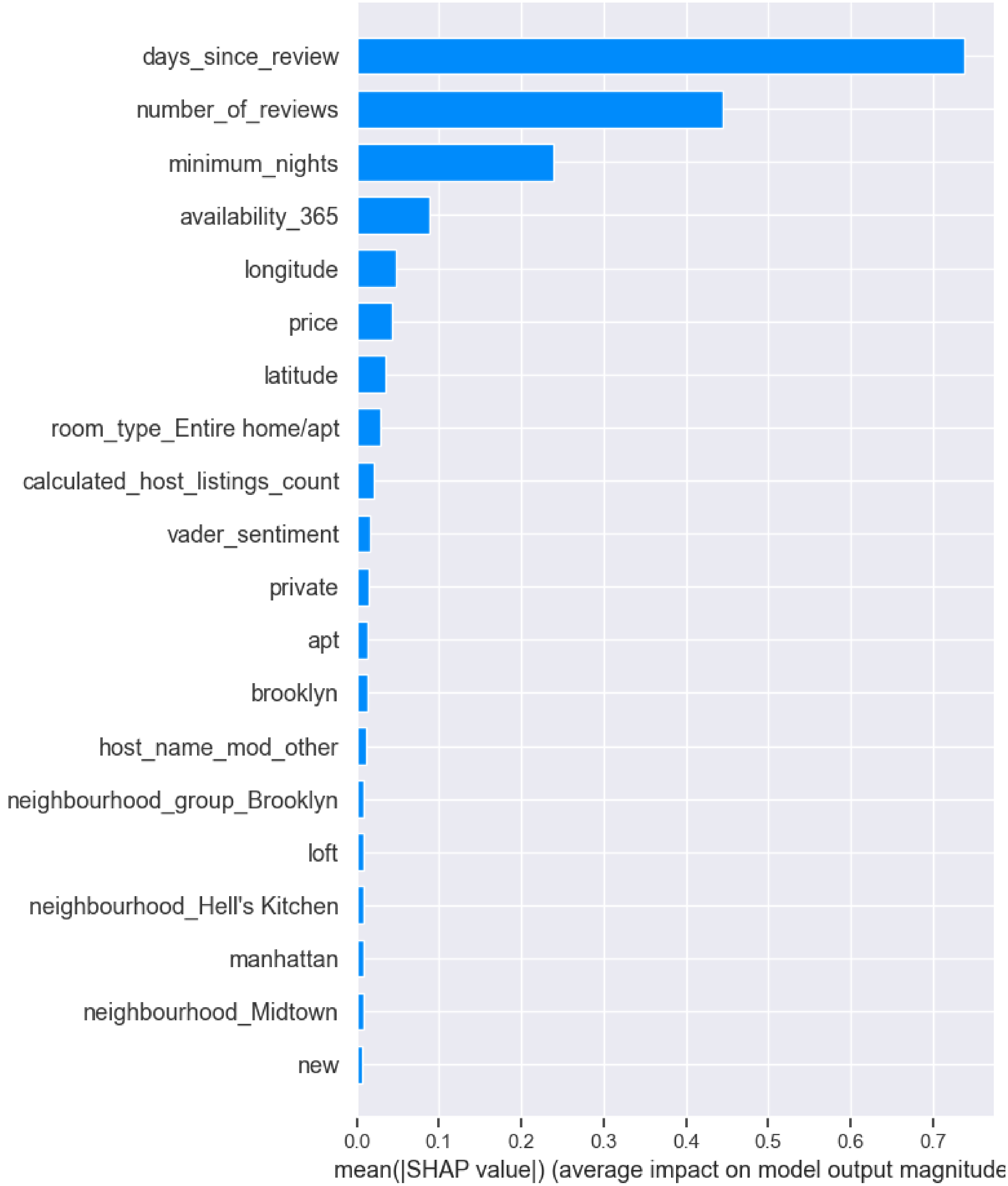
```
shap.summary_plot(test_lgbm_shap_values, X_test_enc, feature_names=feature_names)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignored
```



In [106…

```
shap.summary_plot(test_lgbm_shap_values, X_test_enc, feature_names=feature_names, plot_type="bar'
```



Summary

The SHAP plots show that `days_since_review`, `number_of_reviews`, `minimum_nights`, `availability_365` are some of the more influential features and this is consistent with our previous inferences on feature importance.

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

Table of BEST Performing MODEL - LBGM

In [107…
```
non_linear_result
```

Out[107]:

	Score_Name	Score
0	best_lgbm_r2_score_train_	0.877671
1	best_lgbm_r2_score_test	0.678464
2	best_lgbm_rmse_train	0.587733
3	best_lgbm_rmse_test	0.952569

Predictions vs Actual

In [108…
```
predictions = random_search_lgbm.best_estimator_.predict(X_test)
pred_vs_actual_df = pd.DataFrame({
    "prediction": predictions,
    "actual": y_test
})
pred_vs_actual_df
```

Out[108]:

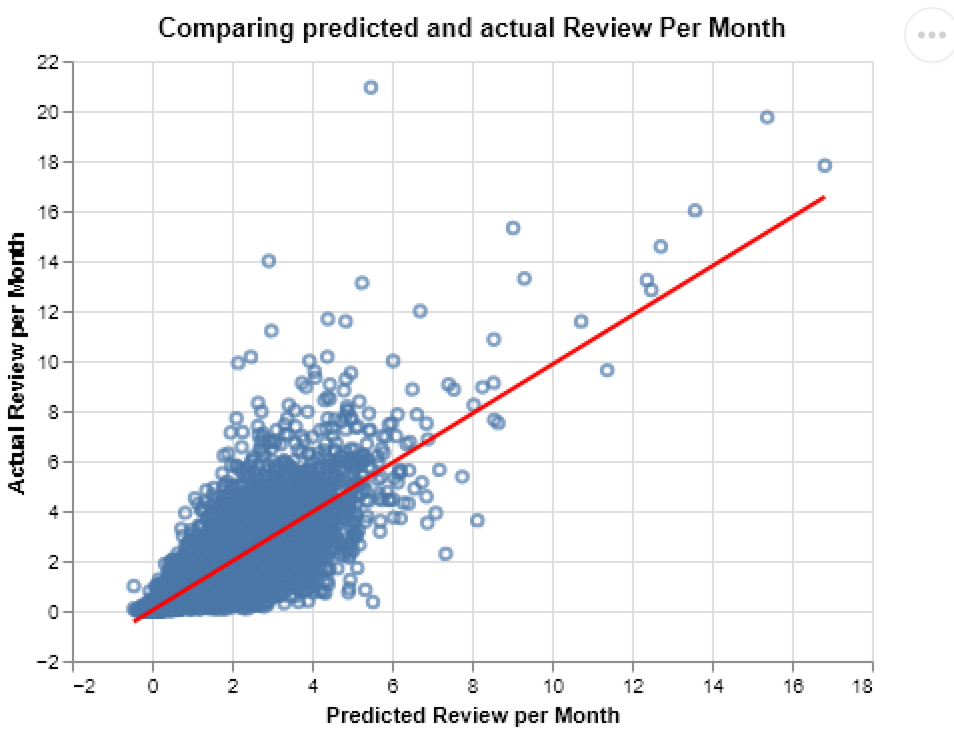
	prediction	actual
26836	4.261101	3.20
37099	4.368276	3.30
13183	3.130505	3.31
45356	1.772439	4.38
31048	2.115774	3.00
...	...	...
9739	0.069837	0.03
6107	3.869614	3.27
45250	5.140759	4.62
5069	0.195333	0.20
35562	0.620617	0.71

7765 rows × 2 columns

In [109…

```
chart = alt.Chart(pred_vs_actual_df, title = "Comparing predicted and actual Review Per Month").
    x = alt.X("prediction", title="Predicted Review per Month"),
    y = alt.Y("actual", title="Actual Review per Month")
)

chart + chart.transform_regression('prediction', 'actual').mark_line(color="red")
```

Out[109]:



Concluding Remarks

This project concludes with the selection of the best performing ML model selected on the basis of the R2 score. The best model selected is the ensemble model LGBM which achieved an good Test score of 67%. The RMSE on the Test data is 0.95 while the RMSE of the Train data is 0.58. The plot above shows that deviations of the prediction against the actual values in the test data. The deviations are indicative of the performance of the model.

Overall this project deeply investigated the original features, engineered a couple of new features, tested linear models and feature selection. Ultimately the non-linear Ensemble models performed the best among all iterations.

Following are some **limitaions** that can be overcome in Future:

- The linear models did not perform well but the non-linear models performed better. This can hint towards the fact that polynomial features might improve performance even further. This remains to be explored.
- Suitable transformation of the target variable to make predictions improve is another avenue to explore and was not tested in this project.
- The scoring metric of Mean Absolute Percentage Error could be used to make the errors in the final model more interpretable.
- Feature Engineering with domain expertise is needed to optimally further improve model performance.

Recommendations:

- Apart from the `day_since_last_review` and `number_of_reviews` features are significant, however they are not actionable from a business perspective.
- The `minimum_nights` and `availability_365` are features that could tangibly improve the popularity of a listing based on our inference. Thus Airbnb could look into promoting hosts who are available for more days during the year and also those that allow larger number of nights for booking. This could be a Key Takeaway from our project.

14. Creating a data analysis pipeline (Challenging)

rubric={reasoning}

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. 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

Feature importance -> running time and the quality of our model.

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: