lab4

December 14, 2024

[2]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lab4.ipynb")
```

[3]:
```
# Run command for helper functions
%run -i ./helpers/helper_functions.py
```

[4]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.dummy import DummyRegressor
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFECV
from sklearn.metrics import mean_squared_error, r2_score

from lightgbm.sklearn import LGBMRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import ElasticNetCV

import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

import altair as alt
import altair_ally as aly

alt.data_transformers.enable("vegafusion")
```

[4]:
```
DataTransformerRegistry.enable('vegafusion')
```

1 Lab 4: Putting it all together in a mini project

For this 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 GitHub repo (you can create one on github.ubc.ca and set the visibility to "public"). If it takes a prohibitively long time to run any of the steps on your laptop, it is OK if you sample the data to reduce the runtime, just make sure you write a note about this.

1.1 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

TEAM BDFJ GitHub URL

- Best Model: LGBM
- R^2 score on test set: 0.69 _____

1.2 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.

1.3 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

Problem statement: This project focuses on analyzing the **New York City Airbnb listings from 2019** dataset to predict the popularity of a listing. We aim to identify how factors such as location, property attributes, time of year, and host activity influence the number of reviews for a listing. We're operating under the assumption that the number of reviews is representative of the popularity of an AirBNB listing The goal is to help hosts better navigate their business by equipping them with a valuable tool to optimize their listings, enhancing their appeal to potential renters.

Data: The data contains 9 explanatory features and one target feature `reviews_per_month`, which will be used as a metric for a listing's popularity. Some observations for `reviews_per_month` were missing; however, this is because they were linked to `number_of_reviews` having values of 0 and therefore can be imputed.

Listing popularity is likely to be dependent on the location, availability throughout the year, any restrictions (e.g. minimum nights, price), and properties of a listing (e.g. room type). Features that are less likely to contribute to the prediction model's learning are: `id`, `host_id` and `host_name` as these are unique and likely will not provide meaninglful patterns or contribute to the model's good generalization. The `name` feature (the name of the listing) is also unique; however, there could be potentially useful features extracted from it, and will be explored in this project.

Geographical features such as `neighbourhood_group`, `neighbourhood`, `lattitude` and `longtitude` should be examined for any high correlations, as these features may contribute the same amount of information to the model.

Date feature `last_review` could be used to analyze seasonal patterns in the listing's popularity.

Models: We are addressing the regression problem by exploring Ridge Regression, Random Forest, LightGBM (LGBM), and Elastic Net models. Each of these models brings unique strengths that could contribute to solving the problem effectively. Ridge Regression offers simplicity and interpretability, making it suitable for understanding feature relationships. Random Forest provides robust performance by capturing complex interactions and handling non-linear relationships. LGBM is known for its speed and efficiency on large datasets, with built-in regularization to prevent overfitting. Finally, Elastic Net combines the strengths of L1 and L2 regularization, making it effective for handling correlated features and feature selection. By comparing these models, we aim to leverage their strengths to achieve the best balance of accuracy, interpretability, and computational efficiency.

[6]:
```
data = pd.read_csv('./data/raw/AB_NYC_2019.csv')
data.head()
```

[6]:
```
     id                                              name  host_id  \
0  2539                Clean & quiet apt home by the park     2787
1  2595                             Skylit Midtown Castle     2845
2  3647               THE VILLAGE OF HARLEM…NEW YORK !     4632
3  3831                   Cozy Entire Floor of Brownstone     4869
4  5022  Entire Apt: Spacious Studio/Loft by central park     7192

     host_name neighbourhood_group neighbourhood  latitude  longitude  \
0         John            Brooklyn    Kensington  40.64749  -73.97237
1     Jennifer           Manhattan       Midtown  40.75362  -73.98377
2    Elisabeth           Manhattan        Harlem  40.80902  -73.94190
3  LisaRoxanne            Brooklyn  Clinton Hill  40.68514  -73.95976
4        Laura           Manhattan   East Harlem  40.79851  -73.94399

         room_type  price  minimum_nights  number_of_reviews last_review  \
0     Private room    149               1                  9  2018-10-19
1  Entire home/apt    225               1                 45  2019-05-21
2     Private room    150               3                  0         NaN
3  Entire home/apt     89               1                270  2019-07-05
4  Entire home/apt     80              10                  9  2018-11-19

   reviews_per_month  calculated_host_listings_count  availability_365
0               0.21                               6               365
1               0.38                               2               355
2                NaN                               1               365
3               4.64                               1               194
4               0.10                               1                 0
```

1.4 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

[7]:
```
# Checking if all null target values is because the number of reviews is 0
# data[(data['number_of_reviews'] != 0) & (data['reviews_per_month'].isnull())]

# Fix target null values to be 0 when number of reviews is 0.
data['reviews_per_month'] = data['reviews_per_month'].apply(lambda x: 0 if pd.
↪isna(x) else x)

# Split into 75% training and 25% test
train_df, test_df = train_test_split(data, random_state=573)
```

1.5 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 (using describe() method) - Mean `reviews_per_month`: An average of 1.086 reviews per month suggests limited interaction for most listings. - Most features have outliers and widely different ranges (as we will see again in the histograms). As such, using a standard scalar for numeric features would be a good idea.

FEATURE DISTRIBUTION PLOTS - Numeric features and the target variables have highly skewed distributions. Therefore, a 'median' strategy for imputation (if needed) will be more appropriate than the mean, as it is less affected by outliers. - The features also have very different ranges. As such we will use a standard scalr to resolve thhis issue.

UNIQUE COUNTS (using nunique() method) - Unique count is important for categorical features. We see host_id and id are unique and they will be dropped. The neighbourhood column seems to have more than 200 unique values. This will slightly explode our feature count after one hot encoding. We will test our pipelines with both using this feature (with feature selection) and not using this feature. Since we will not be including all of our testing in this report, the final report may not have this feature.

FEATURE-FEATURE CORRELATIONS - We see high feature-feature correlations between host_id and id. We were already dropping both these features due to their uniqueness - We also see a high correlation between the target (`number_of_reviews`) and our feature `reviews_per_month`. This correlation makes intuitive sense and therefore we will not be dropping this feature.

MISSING VALUES (using info() method) - Our `reviews_per_month` or target column had roughly 7000 missing values. But we found out this was due to `number_of_reviews` being zero. As such we were able to 'impute' the target values for all observations. - The `last_review` feature will be used to extract the month of last review. This column has some missing values which should be imputed after feature engineering. - A small few amount of `names` and `host_names` also had missing values

SCORING METRICS - Since our target distribution is skewed and may have potential outliers, we will use MAE as it is easier to interpret and is less affected by outliers. Additionally, it will provide a better measure of absolute accuracy of the predictions. - We will also use R^2 as it provides an interpretable result for assessing goodness of fit.

[8]:
```
train_df.describe()
```

[8]:
```
                 id       host_id      latitude     longitude         price  \
count  3.667100e+04  3.667100e+04  36671.000000  36671.000000  36671.000000
```

```
mean   1.901858e+07  6.753485e+07     40.728871    -73.952174    153.009408
std    1.099541e+07  7.862213e+07      0.054648      0.046113    247.269517
min    3.647000e+03  2.438000e+03     40.499790    -74.244420      0.000000
25%    9.453066e+06  7.848642e+06     40.689930    -73.983090     69.000000
50%    1.968643e+07  3.091859e+07     40.722830    -73.955680    105.000000
75%    2.917369e+07  1.074344e+08     40.763125    -73.936160    175.000000
max    3.648543e+07  2.743213e+08     40.912340    -73.712990  10000.000000

       minimum_nights  number_of_reviews  reviews_per_month  \
count    36671.000000       36671.000000       36671.000000
mean         7.017398          23.086990           1.086294
std         20.734663          44.304601           1.593990
min          1.000000           0.000000           0.000000
25%          1.000000           1.000000           0.040000
50%          2.000000           5.000000           0.370000
75%          5.000000          23.000000           1.570000
max       1250.000000         629.000000          58.500000

       calculated_host_listings_count  availability_365
count                    36671.000000      36671.000000
mean                         7.155109        112.255897
std                         33.242306        131.558583
min                          1.000000          0.000000
25%                          1.000000          0.000000
50%                          1.000000         44.000000
75%                          2.000000        225.000000
max                        327.000000        365.000000
```

[9]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 36671 entries, 4888 to 9822
Data columns (total 16 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              36671 non-null  int64
 1   name                            36658 non-null  object
 2   host_id                         36671 non-null  int64
 3   host_name                       36656 non-null  object
 4   neighbourhood_group             36671 non-null  object
 5   neighbourhood                   36671 non-null  object
 6   latitude                        36671 non-null  float64
 7   longitude                       36671 non-null  float64
 8   room_type                       36671 non-null  object
 9   price                           36671 non-null  int64
 10  minimum_nights                  36671 non-null  int64
 11  number_of_reviews               36671 non-null  int64
 12  last_review                     29187 non-null  object
```

```
 13  reviews_per_month               36671 non-null   float64
 14  calculated_host_listings_count  36671 non-null   int64
 15  availability_365                36671 non-null   int64
dtypes: float64(3), int64(7), object(6)
memory usage: 4.8+ MB
```

[10]:
```
train_df.nunique()
```

[10]:
```
id                                36671
name                              36058
host_id                           29147
host_name                          9556
neighbourhood_group                   5
neighbourhood                       220
latitude                          16986
longitude                         13226
room_type                             3
price                               617
minimum_nights                       97
number_of_reviews                   364
last_review                        1710
reviews_per_month                   890
calculated_host_listings_count       47
availability_365                    366
dtype: int64
```

[11]:
```
col_list = ['price','availability_365','number_of_reviews','minimum_nights']

feature_dist_chart = alt.Chart(data).mark_bar().encode(
    x=alt.X(alt.repeat('column')).bin(alt.Bin(maxbins=25)),
    y='count()'
).properties(
    width=140,
    height=200
).repeat(
    column=col_list
)

target_dist_chart = alt.Chart(data).mark_bar().encode(
    x=alt.X('reviews_per_month').bin(alt.Bin(maxbins=50)),
    y='count()'
).properties(
    width=150,
    height=200
)
```

[12]:
```
target_dist_chart
```

[12]: alt.Chart(…)

[13]:
```
feature_dist_chart
```

[13]: alt.RepeatChart(…)

[14]:
```
aly.corr(train_df)
```

[14]: alt.ConcatChart(…)

1.6 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

We wil introduce two new features: 1. Vader sentiment analysis on the name of the listings. - Motivation: listings with positive or engaging titles may attract more guests, and hence more reviews, while negative sentiments might deter potential renters. 2. Month of the listing from `last_review`. - Motivation: the month information can help identify any seasonal patterns in the listing's popularity, as certain months may see higher booking rates, which can lead to more reviews.

[15]:
```
# nltk.download('vader_lexicon')
# nltk.download('punkt')

sid = SentimentIntensityAnalyzer()

num_to_month_map = {
    '01': "January", '02': "February", '03': "March", '04': "April",
    '05': "May", '06': "June", '07': "July", '08': "August",
    '09': "September", '10': "October", '11': "November", '12': "December"
}
```

ADD SENTIMENT FEATURE ____

[16]:
```
train_df['name_polarity_scores'] = train_df['name'].apply(lambda x: None if pd.
 ↪isna(x) else sid.polarity_scores(x)['compound'])
test_df['name_polarity_scores'] = test_df['name'].apply(lambda x: None if pd.
 ↪isna(x) else sid.polarity_scores(x)['compound'])
```

ADD MONTH OF LAST REVIEW FEATURE ____

```
[17]: train_df['month_of_last_review'] = train_df['last_review'].apply(lambda x: x if␣
      ↪pd.isna(x) else x.split('-')[1])
      train_df['month_of_last_review'] = train_df['month_of_last_review'].
      ↪apply((lambda x: x if pd.isna(x) else num_to_month_map[x]))
```

```
[18]: test_df['month_of_last_review'] = test_df['last_review'].apply(lambda x: x if␣
      ↪pd.isna(x) else x.split('-')[1])
      test_df['month_of_last_review'] = test_df['month_of_last_review'].apply((lambda␣
      ↪x: x if pd.isna(x) else num_to_month_map[x]))
```

```
[19]: # train_df['count_hostID'] = train_df.groupby('host_id')['host_id'].
      ↪transform('count')
      # test_df['count_hostID'] = test_df.groupby('host_id')['host_id'].
      ↪transform('count')
```

1.7 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

```
[20]: 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']
```

```
[21]: categorical_features = ['neighbourhood_group', 'room_type',␣
      ↪'month_of_last_review']
      numeric_features = ['latitude',
                          'longitude',
                          'minimum_nights',
                          'number_of_reviews',
                          'calculated_host_listings_count',
                          'availability_365',
                          'name_polarity_scores'
                          ]
      drop_features = ['id', 'host_id', 'host_name', 'name', 'neighbourhood']

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

      numeric_transformer = make_pipeline(
```

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

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

[22]:
```
preprocessor.fit(X_train)
```

[22]:
```
ColumnTransformer(transformers=[('pipeline-1',
                                 Pipeline(steps=[('simpleimputer',
SimpleImputer(strategy='median')),
                                                 ('standardscaler',
                                                  StandardScaler())]),
                                 ['latitude', 'longitude', 'minimum_nights',
                                  'number_of_reviews',
                                  'calculated_host_listings_count',
                                  'availability_365', 'name_polarity_scores']),
                                ('pipeline-2',
                                 Pipeline(steps=[('simpleimputer',
SimpleImputer(fill_value='missing',
strategy='constant')),
                                                 ('onehotencoder',
OneHotEncoder(handle_unknown='ignore',
sparse_output=False))]),
                                 ['neighbourhood_group', 'room_type',
                                  'month_of_last_review']),
                                ('drop', 'drop',
                                 ['id', 'host_id', 'host_name', 'name',
                                  'neighbourhood'])])
```

1.8 6. Baseline model

rubric={accuracy}

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

Points: 2

[23]:
```
results = {}
```

[24]:
```
dummy = DummyRegressor()
scoring_metric = {
    "R2": "r2",
    "Neg MAE": "neg_mean_absolute_error",
}
```

```python
results["Dummy"] = mean_std_cross_val_scores(
    dummy, X_train, y_train, return_train_score=True, scoring=scoring_metric
)

pd.DataFrame(results)
```

[24]:

```
                          Dummy
fit_time        0.003 (+/- 0.001)
score_time      0.001 (+/- 0.000)
test_R2        -0.000 (+/- 0.000)
train_R2        0.000 (+/- 0.000)
test_Neg MAE   -1.145 (+/- 0.007)
train_Neg MAE  -1.145 (+/- 0.002)
```

The dummy model shows poor performance, as expected, indicating that predicting the average value of the target variable `reviews_per_month` (1.09) for all observations does not capture the complexity of the problem.

1.9 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

We will start by exploring the Ridge model. To optimize its performance, we will use RidgeCV, which automatically tunes the regularization hyperparameter (alpha) using cross-validation. This process allows us to dynamically identify the best alpha within overall cross-validation. See cross validation results below.

The cross-validation results show an improved $R2$ score of 0.52, indicating that the model can explain approximately 52% of the variance in the target variable, reviews_per_month. While this represents a significant improvement over the baseline dummy model, the performance is still moderate and leaves room for enhancement. Further refinements, such as feature engineering or hyperparameter tuning, may help improve the model's predictive power.

Additionally, the Mean Absolute Error (MAE) of 0.62 is better than that of the dummy model, meaning the model's predictions deviate, on average, by about 0.62 reviews per month from the actual values. Considering that the average reviews_per_month is 1.08, this deviation is approximately 60% of the mean value. While the reduction in error is promising, the discrepancy highlights opportunities for further optimization to better align predictions with actual values.

[25]:

```python
alpha_vals =  np.logspace(-8, 8, 10)
```

[26]:
```python
# RidgeCV has built in hyperparameter optimization
ridge = make_pipeline(
    preprocessor,
    RidgeCV(alphas = alpha_vals, cv=10)
)
```

[27]:
```python
results["RidgeCV"] = mean_std_cross_val_scores(
    ridge, X_train, y_train, return_train_score=True, scoring=scoring_metric
)
pd.DataFrame(results)
```

[27]:

	Dummy	RidgeCV
fit_time	0.003 (+/- 0.001)	0.312 (+/- 0.007)
score_time	0.001 (+/- 0.000)	0.004 (+/- 0.000)
test_R2	-0.000 (+/- 0.000)	0.519 (+/- 0.032)
train_R2	0.000 (+/- 0.000)	0.518 (+/- 0.009)
test_Neg MAE	-1.145 (+/- 0.007)	-0.622 (+/- 0.007)
train_Neg MAE	-1.145 (+/- 0.002)	-0.622 (+/- 0.002)

1.10 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

LGBM Regressor The LightGBM (LGBM) model achieved a train R^2 of 0.74 and a CV R^2 of 0.66. This is not a significant deviation, indicating that the model is not that heavily overfitting. Both the train and CV R^2 scores are higher than those of Ridge regression, suggesting that LGBM is better at capturing the data patterns and generalizes more effectively than Ridge.

A similar trend is observed in the MAE (Mean Absolute Error) scores. The CV and train MAE scores for LGBM are -0.484 (+/- 0.011) and -0.441 (+/- 0.004), respectively, which are significantly lower than those of Ridge. This implies that the LGBM model's predictions deviate, on average, by approximately 0.484 reviews per month—an improvement over Ridge.

However, LGBM's fit time is 0.851, which is higher than Ridge's fit time of 0.685, indicating that LGBM is computationally more expensive. Despite the higher computational cost, LGBM outperforms Ridge in terms of capturing data patterns and providing better predictions, making it a more effective model overall for this task.

Random Forest The Random Forest model achieved a train R^2 of 0.949 and a test R^2 of 0.638. The relatively larger deviation between train and test R^2 scores suggests some overfitting, as Random Forest tends to excel in capturing data patterns but may struggle to generalize when compared to RidgeCV, which had train and test R^2 scores of 0.518 and 0.519, respectively. However, Random Forest still performs better on the test R^2 metric, indicating its ability to capture more complex patterns in the data.

In terms of MAE (Mean Absolute Error), Random Forest significantly outperforms RidgeCV. The test MAE score for Random Forest is -0.486 (+/- 0.010), compared to RidgeCV's -0.622 (+/- 0.007), meaning Random Forest's predictions deviate less, on average, by approximately 0.486 reviews per month. The train MAE for Random Forest is even better at -0.181 (+/- 0.002), showing it fits the training data exceptionally well, though at the cost of potential overfitting.

The downside of Random Forest is its computational expense. It has a fit time of 4.470 seconds, which is significantly higher than RidgeCV's 0.685 . Its score time is also longer at 0.072 compared to RidgeCV's 0.009. This makes Random Forest less efficient for scenarios requiring rapid model training or scoring.

Elastic Net The Elastic Net model achieved train and test R^2 scores of 0.518 and 0.521, respectively, which are very similar to those of RidgeCV (train R^2: 0.518, test R^2: 0.519). This similarity indicates Elastic Net and RidgeCV perform comparably in terms of capturing patterns in the data.

In terms of MAE (Mean Absolute Error), Elastic Net and RidgeCV also show minimal differences. Elastic Net's test MAE is -0.622 (+/- 0.012), almost identical to RidgeCV's -0.622 (+/- 0.007). Similarly, the train MAE for Elastic Net (-0.621 (+/- 0.002)) is nearly the same as RidgeCV's (-0.622 (+/- 0.002)). This indicates that both models predict with the same level of average error on the test and training sets.

Elastic Net has a slight computational advantage over RidgeCV. It has a fit time of 0.505 (+/- 0.067), compared to RidgeCV's 0.685 (+/- 0.039). The score time for Elastic Net is also marginally lower at 0.008 (+/- 0.002), compared to RidgeCV's 0.009 (+/- 0.001). This makes Elastic Net slightly more efficient, especially for large-scale applications requiring faster model training.

[28]:
```
# LGBM
pipe_lgbm = make_pipeline(
    preprocessor,
    LGBMRegressor(
        n_jobs=-1,
        verbose=-1,
        random_state=573
    )
)

results['lgbm'] = mean_std_cross_val_scores(
    pipe_lgbm, X_train, y_train, return_train_score=True, scoring=scoring_metric
)
```

[29]:
```
# Random Forest
# Code adapted from Lecture 7: https://pages.github.ubc.ca/mds-2024-25/
↪DSCI_573_feat-model-select_students/lectures/07_ensembles.html#random-forests
pipe_rf = make_pipeline(
    preprocessor,
    RandomForestRegressor(
        n_jobs=-1,
        random_state=573,
    )
```

```
)

results["random_forests"] = mean_std_cross_val_scores(
    pipe_rf, X_train, y_train, return_train_score=True, scoring=scoring_metric
)
```

[30]:
```
# ElasticNet
elastic = make_pipeline(
    preprocessor,
    ElasticNetCV(max_iter=20_000, tol=0.01, cv=10)
)

results['elastic_net'] = mean_std_cross_val_scores(
    elastic, X_train, y_train, return_train_score=True, scoring=scoring_metric,␣
 ↪cv=10
)
```

[31]:
```
pd.DataFrame(results).T
```

[31]:
```
                           fit_time          score_time               test_R2  \
Dummy           0.003 (+/- 0.001)  0.001 (+/- 0.000)  -0.000 (+/- 0.000)
RidgeCV         0.312 (+/- 0.007)  0.004 (+/- 0.000)   0.519 (+/- 0.032)
lgbm            0.654 (+/- 0.053)  0.008 (+/- 0.000)   0.661 (+/- 0.038)
random_forests  0.860 (+/- 0.027)  0.023 (+/- 0.007)   0.638 (+/- 0.035)
elastic_net     0.213 (+/- 0.004)  0.003 (+/- 0.000)   0.521 (+/- 0.037)

                         train_R2        test_Neg MAE       train_Neg MAE
Dummy           0.000 (+/- 0.000)  -1.145 (+/- 0.007)  -1.145 (+/- 0.002)
RidgeCV         0.518 (+/- 0.009)  -0.622 (+/- 0.007)  -0.622 (+/- 0.002)
lgbm            0.736 (+/- 0.006)  -0.484 (+/- 0.011)  -0.441 (+/- 0.004)
random_forests  0.949 (+/- 0.002)  -0.486 (+/- 0.010)  -0.181 (+/- 0.002)
elastic_net     0.518 (+/- 0.005)  -0.622 (+/- 0.012)  -0.621 (+/- 0.002)
```

1.11 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

Here, we test out our neighbourhood categorical feature followed by feature selection using RFE CV.

[32]:
```
rfe_ridge_categorical_features = ['neighbourhood_group', 'room_type',␣
↪'month_of_last_review', 'neighbourhood']

rfe_ridge_drop_features = ['id', 'host_id', 'host_name', 'name']

rfe_ridge_preprocessor = make_column_transformer(
    (numeric_transformer, numeric_features),
    (categorical_transformer, rfe_ridge_categorical_features),
    ("drop", rfe_ridge_drop_features)
)
```

[33]:
```
# Make the rfecv pipeline
pipe_rfe_ridgecv = make_pipeline(
    rfe_ridge_preprocessor,
    RFECV(Ridge(), cv=10, n_jobs=-1),
    RidgeCV()
)

# Get the cv scores
results['rfe_ridgecv'] = mean_std_cross_val_scores(
    pipe_rfe_ridgecv, X_train, y_train, return_train_score=True,␣
↪scoring=scoring_metric
)
```

[34]:
```
pd.DataFrame(results).T
```

[34]:
```
                          fit_time         score_time             test_R2  \
Dummy            0.003 (+/- 0.001)  0.001 (+/- 0.000)  -0.000 (+/- 0.000)
RidgeCV          0.312 (+/- 0.007)  0.004 (+/- 0.000)   0.519 (+/- 0.032)
lgbm             0.654 (+/- 0.053)  0.008 (+/- 0.000)   0.661 (+/- 0.038)
random_forests   0.860 (+/- 0.027)  0.023 (+/- 0.007)   0.638 (+/- 0.035)
elastic_net      0.213 (+/- 0.004)  0.003 (+/- 0.000)   0.521 (+/- 0.037)
rfe_ridgecv      9.642 (+/- 0.811)  0.013 (+/- 0.001)   0.528 (+/- 0.030)

                          train_R2        test_Neg MAE       train_Neg MAE
Dummy            0.000 (+/- 0.000)  -1.145 (+/- 0.007)  -1.145 (+/- 0.002)
RidgeCV          0.518 (+/- 0.009)  -0.622 (+/- 0.007)  -0.622 (+/- 0.002)
lgbm             0.736 (+/- 0.006)  -0.484 (+/- 0.011)  -0.441 (+/- 0.004)
random_forests   0.949 (+/- 0.002)  -0.486 (+/- 0.010)  -0.181 (+/- 0.002)
elastic_net      0.518 (+/- 0.005)  -0.622 (+/- 0.012)  -0.621 (+/- 0.002)
rfe_ridgecv      0.532 (+/- 0.008)  -0.630 (+/- 0.006)  -0.626 (+/- 0.003)
```

We observed that adding the neighborhood feature followed by feature selection slightly improved the performance of RidgeCV, as evidenced by the higher R^2 scores. However, the increased fit times indicate significant computational demands, which, in our opinion, do not justify the marginal improvement.

While we believe that incorporating the neighborhood feature followed by Recursive Feature Elimination (RFE) could marginally enhance the performance of other models as well, we opted not to pursue this approach for this project. The high fit times render it impractical and infeasible given the scope and computational constraints.

1.12 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 performed hyperparameter optimization on Ridge, LGBTM, Random Forest and Elastic Net models using RandomizesSearchCV method of sklearn. Below is the summary of results:

Ridge: Optimized the alpha hyperparameter. Best $\alpha = 13.49$, with R^2 score being 0.52, which is similar to our linear model in Section 7. This is expected since RidgeCV (from section 7) already had built in hyperparameter optimization for α.

LGBM: Optimized learning_rate, max_depth and n_estimators hyperparameters. Best values are 0.05, 50 and 200 respectively, with R^2 score of 0.66, which is similar to the initial model in section 8.

Random Forest: Optimized max_depth and n_estimators hyperparameters. Best values are 16 and 345 respectively, with R^2 score of 0.64 which is also similar to the initial model.

Elastic Net: Optimimized L1 ratio and alpha hyperparameters. Best values are 0.1 and all alphas chosen that were passed. The new is 0.48, which is considerably worse than the base model in section 8. This could be due to a faulty range of hyperparameter values passed into random search.

[35]:
```
from scipy.stats import loguniform
param_dist_ridge = {"ridgecv__alphas": loguniform(1e-5, 1e5)}

random_ridge_pipe = make_pipeline(
    preprocessor,
    RidgeCV()
)

random_ridge_search = RandomizedSearchCV(
    random_ridge_pipe,
    param_distributions=param_dist_ridge,
    n_iter=500,
    n_jobs=-1,
    random_state=573,
    return_train_score=True,
    scoring=scoring_metric,
```

```
    refit='R2'
)
```

[36]:
```
random_ridge_search.fit(X_train, y_train)
```

[36]:
```
RandomizedSearchCV(estimator=Pipeline(steps=[('columntransformer',
ColumnTransformer(transformers=[('pipeline-1',
Pipeline(steps=[('simpleimputer',
                SimpleImputer(strategy='median')),
               ('standardscaler',
                StandardScaler())]),
['latitude',
'longitude',
'minimum_nights',
'number_of_reviews',
'calculated_host_listings_count',
'availability_365',
'name_polarity_score…
'month_of_last_review']),
('drop',
'drop',
['id',
'host_id',
'host_name',
'name',
'neighbourhood'])])),
                                             ('ridgecv', RidgeCV())]),
                   n_iter=500, n_jobs=-1,
                   param_distributions={'ridgecv__alphas':
<scipy.stats._distn_infrastructure.rv_continuous_frozen object at 0x32efeef00>},
                   random_state=573, refit='R2', return_train_score=True,
                   scoring={'Neg MAE': 'neg_mean_absolute_error', 'R2': 'r2'})
```

[37]:
```
opti_results = {}

opti_results['ridge'] = {
    "Optimized Train Score (R2)": random_ridge_search.best_estimator_.
 ↪score(X_train, y_train),
    "Optimized Validation Score (R2)": random_ridge_search.best_score_
}
```

[38]:
```
pipe_lgbm_reduced = make_pipeline(
    preprocessor,
    LGBMRegressor(
        n_jobs=-1,
        random_state=573
    )
```

```
)

param_grid_lgbm =  {
    "lgbmregressor__learning_rate": [0.01, 0.05, 0.1, 0.15, 0.2],
    "lgbmregressor__max_depth": [10,50,100],
    "lgbmregressor__n_estimators": [100,150,200]
}
```

[39]:
```
random_lgbm_search = RandomizedSearchCV(
    pipe_lgbm_reduced,
    param_distributions=param_grid_lgbm,
    n_iter=50,
    n_jobs=-1,
    return_train_score=True,
    scoring=scoring_metric,
    refit='R2',
    verbose=False
)
```

[ ]:
```
random_lgbm_search.fit(X_train, y_train)
```

[41]:
```
opti_results['lgbm'] = {
    "Optimized Train Score (R2)": random_lgbm_search.best_estimator_.
  ↪score(X_train, y_train),
    "Optimized Validation Score (R2)": random_lgbm_search.best_score_
}
```

[42]:
```
# Random Forest Regresser Optimization
pipe_rf_reduced = make_pipeline(
    preprocessor,
    RandomForestRegressor(
        n_jobs=-1,
        random_state=573,
    )
)

param_grid_rf = {
    "randomforestregressor__max_depth": np.linspace(2, 20, 10, dtype=int),
    "randomforestregressor__n_estimators": np.linspace(10, 500, 20, dtype=int)
}

random_rf_search = RandomizedSearchCV(
    pipe_rf_reduced,
    param_distributions = param_grid_rf,
    n_iter=50,
    random_state=573,
    return_train_score=True,
```

```
    scoring=scoring_metric,
    refit='R2'
)
```

[43]:
```
random_rf_search.fit(X_train, y_train)
```

[43]:
```
RandomizedSearchCV(estimator=Pipeline(steps=[('columntransformer',
ColumnTransformer(transformers=[('pipeline-1',
Pipeline(steps=[('simpleimputer',
                 SimpleImputer(strategy='median')),
                ('standardscaler',
                 StandardScaler())]),
['latitude',
'longitude',
'minimum_nights',
'number_of_reviews',
'calculated_host_listings_count',
'availability_365',
'name_polarity_score…
random_state=573))]),
                   n_iter=50,
                   param_distributions={'randomforestregressor__max_depth':
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20]),
                                        'randomforestregressor__n_estimators':
array([ 10,  35,  61,  87, 113, 138, 164, 190, 216, 242, 267, 293, 319,
       345, 371, 396, 422, 448, 474, 500])},
                   random_state=573, refit='R2', return_train_score=True,
                   scoring={'Neg MAE': 'neg_mean_absolute_error', 'R2': 'r2'})
```

[44]:
```
random_rf_search.best_params_
```

[44]:
```
{'randomforestregressor__n_estimators': np.int64(345),
 'randomforestregressor__max_depth': np.int64(16)}
```

[45]:
```
opti_results['rf'] = {
    "Optimized Train Score (R2)": random_rf_search.best_estimator_.
 ↪score(X_train, y_train),
    "Optimized Validation Score (R2)": random_rf_search.best_score_
}
```

[46]:
```
# Elastic Net Optimization
pipe_elasticnet_reduced = make_pipeline(
    preprocessor,
    ElasticNetCV(max_iter=20_000, tol=0.01, cv=5)
)

param_grid_elastic = {
```

```
    "elasticnetcv__l1_ratio": np.linspace(0.1, 1.0, 10),
    "elasticnetcv__alphas": [[0.1, 1.0, 10.0, 100.0]],
}

random_en_search = RandomizedSearchCV(
    pipe_elasticnet_reduced,
    param_distributions=param_grid_elastic,
    n_iter=500,
    verbose=1,
    n_jobs=-1,
    random_state=573,
    return_train_score=True,
    scoring=scoring_metric,
    refit='R2'
)
```

[47]:
```
random_en_search.fit(X_train, y_train)
```

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

```
/Users/brianchang/miniforge3/envs/573/lib/python3.12/site-
packages/sklearn/model_selection/_search.py:320: UserWarning: The total space of
parameters 10 is smaller than n_iter=500. Running 10 iterations. For exhaustive
searches, use GridSearchCV.
  warnings.warn(
```

[47]:
```
RandomizedSearchCV(estimator=Pipeline(steps=[('columntransformer',
ColumnTransformer(transformers=[('pipeline-1',
Pipeline(steps=[('simpleimputer',
                SimpleImputer(strategy='median')),
               ('standardscaler',
                StandardScaler())]),
['latitude',
'longitude',
'minimum_nights',
'number_of_reviews',
'calculated_host_listings_count',
'availability_365',
'name_polarity_score…
'neighbourhood'])])),
                                             ('elasticnetcv',
                                              ElasticNetCV(cv=5, max_iter=20000,
                                                           tol=0.01))]),
                   n_iter=500, n_jobs=-1,
                   param_distributions={'elasticnetcv__alphas': [[0.1, 1.0,
                                                                  10.0,
                                                                  100.0]],
                                        'elasticnetcv__l1_ratio': array([0.1,
```

```
0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ])},
                   random_state=573, refit='R2', return_train_score=True,
                   scoring={'Neg MAE': 'neg_mean_absolute_error', 'R2': 'r2'},
                   verbose=1)
```

[48]:
```
random_en_search.best_params_
```

[48]:
```
{'elasticnetcv__l1_ratio': np.float64(0.1),
 'elasticnetcv__alphas': [0.1, 1.0, 10.0, 100.0]}
```

[49]:
```
opti_results['elastic_net'] = {
    "Optimized Train Score (R2)": random_en_search.best_estimator_.
 ↪score(X_train, y_train),
    "Optimized Validation Score (R2)": random_en_search.best_score_
}
```

[51]:
```
pd.DataFrame(opti_results).to_csv('results/optimized_results.csv')
```

[52]:
```
import pickle
with open('./models/ridge_random.pickle', 'wb') as file:
    pickle.dump(random_ridge_search, file)

with open('./models/lgbm_random.pickle', 'wb') as file:
    pickle.dump(random_lgbm_search, file)

with open('./models/rf_random.pickle', 'wb') as file:
    pickle.dump(random_rf_search, file)

with open('./models/random_en_search.pickle', 'wb') as file:
    pickle.dump(random_en_search, file)
```

1.13 11. Interpretation and feature importances

rubric={accuracy,reasoning}

Your tasks:

1. Use the methods we saw in class (e.g., `permutation_importance` or `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

Permutation importance evaluates the significance of a feature by measuring the impact on model performance when its values are randomly shuffled. If shuffling a feature significantly reduces the model's accuracy, it indicates high importance. Conversely, minimal change suggests low importance.

For our **LGBM** model, the features with the highest permutation importance are: -

`number_of_reviews` - `month_of_last_review_July` - `month_of_last_review_June` - `minimum_nights` - `availability_365` - `calculated_host_listings_count` - `latitude` and `longitude`

These results align intuitively. Location, listing availability, and the duration of a guest's stay likely influence their experience and review. The prominence of June and July also makes sense, as summer months typically see higher Airbnb activity.

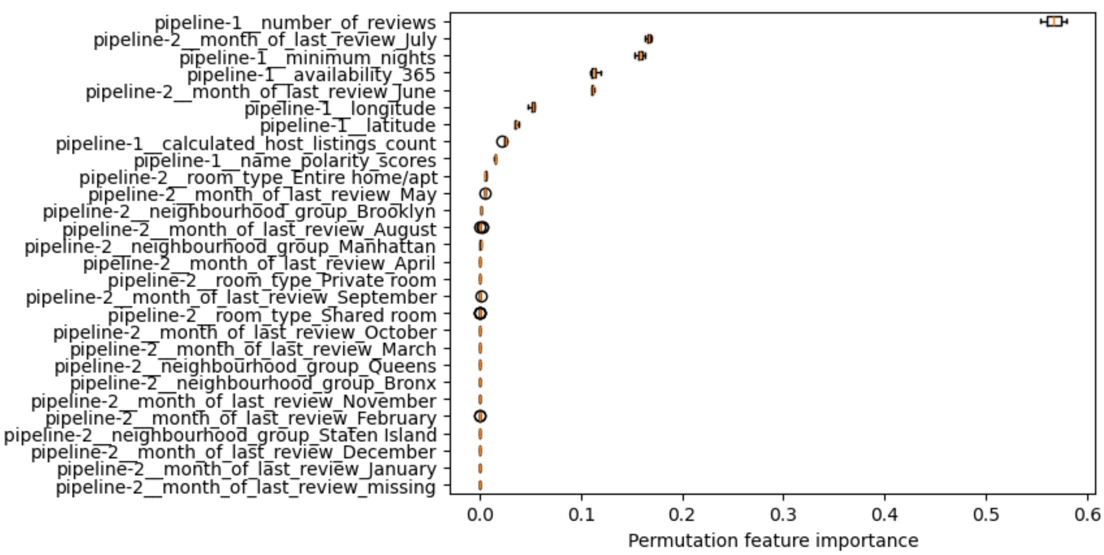
However, it's important to note that permutation importance reflects the model's internal understanding of features and may not directly represent their real-world causal relationships.

[53]:
```
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance
import pandas as pd

# def get_permutation_importance(model):
#     # X_train_perm = X_train.drop(columns=["race", "education.num", "fnlwgt"])
#     result = permutation_importance(model, X_train, y_train, n_repeats=10,␣
 ↪random_state=123)
#     perm_sorted_idx = result.importances_mean.argsort()
#     plt.boxplot(
#         result.importances[perm_sorted_idx].T,
#         vert=False,
#         tick_labels=X_train.columns[perm_sorted_idx],
#     )
#     plt.xlabel('Permutation feature importance')
#     plt.show()

def get_permutation_importance(pipe):
    X_train_transformed = pipe[:-1].transform(X_train)
    feature_names = pipe[:-1].get_feature_names_out()

    result = permutation_importance(
        pipe[-1],  # Final model in the pipeline
        X_train_transformed,
        y_train,
        n_repeats=10,
        random_state=123,
        n_jobs=-1
    )

    perm_sorted_idx = result.importances_mean.argsort()

    # Plot permutation importance
    plt.boxplot(
        result.importances[perm_sorted_idx].T,
        vert=False,
```

```
        labels=[feature_names[i] for i in perm_sorted_idx],  # Use transformed␣
↪feature names
    )
    plt.xlabel('Permutation feature importance')
    plt.show()
```

[54]:
```
get_permutation_importance(random_lgbm_search.best_estimator_)
```

```
/var/folders/1k/qsrq_cwj47l38j9ts17rbcx80000gn/T/ipykernel_62046/2961204474.py:3
3: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been
renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be
dropped in 3.11.
  plt.boxplot(
```



1.14 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

CHECKING PERFORMANCE ON TEST SET

---

[55]:
```
best_pipeline = random_lgbm_search.best_estimator_

y_pred = best_pipeline.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"R-squared (R²): {r2:.4f}")
```

```
Mean Squared Error (MSE): 0.7925
R-squared (R²): 0.6931
```

COMPARING TEST SCORE AND CV SCORE

The mean cross-validation (CV) score for our best LGBM model was 0.662, and the test score is 0.693, which is very close. This consistency indicates that the model's performance on unseen data aligns well with its performance during cross-validation.

We trust our results due to multiple reasons. - LGBM is an ensemble model, meaning it uses many trees instead of just one, which helps reduce overfitting. Additionally, LGBM incorporates regularization techniques within each tree, further enhancing its ability to generalize. This increases our confidence in the model's robustness and generalizability.

- For our best estimator, the standard deviation of test scores across the CV folds is only XXX, which is low. This indicates that the performance is consistent across all CV folds and that the mean CV score was not achieved by chance on just one fold. Additionally, our dataset is large, with over 20,000 observations remaining in the training set after preprocessing. This ensures the model sees diverse data during the CV folds, making it unlikely to achieve a high CV score purely by chance. Therefore, we are confident that optimization bias during cross-validation is minimal.
- We ensured that the test set was kept completely separate from the training set at every step of the pipeline, including during feature engineering. This ensures that the test set provides an unbiased measure of our model's performance on unseen data. Additionally, the test score is slightly higher than the mean CV score, which is plausible and within expectation. Therefore, we trust the reliability of our results.

SHAP

[56]:
```
import shap
shap.initjs()
%matplotlib inline
```

```
<IPython.core.display.HTML object>
```

[57]:
```
X_test_transformed = random_lgbm_search.best_estimator_[:-1].transform(X_test)
feature_names = random_lgbm_search.best_estimator_[:-1].get_feature_names_out()

X_test_enc = pd.DataFrame(
    data=X_test_transformed,
    columns=feature_names,
    index=X_test.index,
)

X_test_enc.columns = X_test_enc.columns.str.split('__').str[1]
```

[58]:
```
y_test_reset = y_test.reset_index(drop=True)
```

[59]:
```
lgbm_explainer = shap.TreeExplainer(
    random_lgbm_search.best_estimator_.named_steps['lgbmregressor'],
)
lgbm_explanation = lgbm_explainer(X_test_enc)
```

[60]:
```
avg_rev_g_2_75_ind = y_test_reset[y_test_reset > 2.75].index.tolist()
avg_rev_l_5_ind = y_test_reset[y_test_reset < 5 ].index.tolist()

ex_g3_index = avg_rev_g_2_75_ind[2]
ex_l5_index = avg_rev_l_5_ind[2]
```

[61]:
```
shap.plots.force(lgbm_explanation[ex_g3_index, :])
```

[61]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x397b3bc20>
```

For this prediction (`reviews_per_month` = 2.14), features like `availability_365` (-0.6253) and `month_of_last_review_July` (1) contribute to increasing the prediction, while features such as `minimum_nights` (-0.1938) reduce the prediction.

[62]:
```
shap.plots.force(lgbm_explanation[ex_l5_index, :])
```

[62]:
```
<shap.plots._force.AdditiveForceVisualizer at 0x3979f3c20>
```

For this prediction (`reviews_per_month` = -0.00), very few features are pushing the model towards a higher prediction. Features such as `number_of_reviews` (-0.5211), `minimum_nights` (-0.04907) and `month_of_last_review_June` (0) are pushing the model towards a lower prediction value.

1.15 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

1.16 Summary of Results

1.16.1 Optimized Model Results

CV Scores

---

[63]:
```
import pandas as pd
pd.read_csv('./results/optimized_results.csv')
```

[63]:

	Unnamed: 0	ridge	lgbm	rf	elastic_net
0	Optimized Train Score (R2)	0.517736	0.728030	0.907381	0.477547
1	Optimized Validation Score (R2)	0.518974	0.662412	0.639151	0.479112

Test Scores

---

[64]:
```
print(f"Mean Squared Error (MSE) on Test Set: {mse:.4f}")
print(f"R-squared (R²) on Test Set: {r2:.4f}")
```

```
Mean Squared Error (MSE) on Test Set: 0.7925
R-squared (R²) on Test Set: 0.6931
```

1.17 Conclusion

Overall, out of the four models explored, LGBM demonstrated the best R^2 scores both before and after hyperparameter optimization, while also showing the least amount of overfitting. Our final LGBM model, with parameters (learning_rate=0.05, max_depth=50, n_estimators=200), achieved an R^2 of 0.693 on the test data. While this is not a perfect score, it is quite adequate for predicting the target. The small gap between the train and test scores further indicates that the model generalizes well.

The SHAP test, showed that native feaatures as number_of_reviews, minimum_nights as well as our engineered feature month_of_last_review have the highest feature importances in predicting the target.

1.17.1 Other Ideas

As seen in our report, the hyperparameter optimization did yield somewhat better results. To potentially see further improvements in the scores we could explore the following ideas:

1. Evaluate other ensemble models such as XGBoost and CatBoost, which may have better handling of categorical features or more robust regularization.

2. Engineer more and better features.
3. From the SHAP test, we observe that latitude was assigned some importance, suggesting that geographical location does influence the listing's popularity. Although the neighbourhood feature was excluded due to the many unique values, exploring other possible groupings based on latitude (e.g., dividing regions into categories or clustering listings by proximity) could help capture additional patterns.
4. Expand hyperparameter optimization ranges

1.18 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.

1.19 15. Your takeaway from the course (Challenging)

rubric={reasoning}

Your tasks:

What is your biggest takeaway from this course?

Points: 0.25

Creating the best model to solve a problem is a challenging process. It involves understanding the problem deeply and choosing an appropriate model based on the nature of the problem and the available data. Often, this requires feature engineering, which depends heavily on domain expertise, trial and error, and significant effort.

Features frequently interact with each other, and introducing polynomial features can help capture these interactions, enabling the model to perform better in an augmented feature space. However, as more features are added, the model becomes increasingly complex, which can lead to overfitting and reduced interpretability. To address this, feature selection methods are typically applied to simplify the model while maintaining its predictive power.

Ultimately, all three processes—model selection, feature engineering, and feature selection—require a thorough understanding of the problem, persistence through trial and error, and, occasionally, a bit of luck. Together, they are crucial for building models that are both effective and generalizable.

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)

1.20 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: 16 hours

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: Having a broader selection of problems (as opposed to 2) would have made it more interesting.