# Template for In-Class Kaggle Competition Writeup

CompSci 671

Due: Nov 26th 2024

Kaggle Competition Link

Kaggle ID: `YQL_Skorpion`

## 1 Exploratory Analysis

*How did you make sense of the provided dataset and get an idea of what might work? Did you use histograms, scatter plots or some sort of clustering algorithm? Did you do any feature engineering? Describe your thought process in detail for how you approached the problem and if you did any feature engineering in order to get the most out of the data.*

The first thing that I did was to check for badly formatted data, since this is a rather messy real-world dataset and needs significant data cleansing. In this step, I check for two things: (i) whether it makes sense to drop a column, for example `has_availability` for (almost) all entries are of the same value, and (ii) how to deal with missing entries. For the latter one, I used mostly median/mode for missing numerical values, and "None" for missing review strings.

Next, for all columns that could be converted into numerical data, I did so; these mostly include Boolean data and DateTime object. For example, `host_response_time`'s original options are a list of strings, ranging from "within an hour" all the way up to "a few days or more." Instead of these, I assigned an integer value to each option. Likewise for a few other columns, which can be seen in the notebook.

Having cleansed the data format, I now check the distribution of each variable. I noticed that certain columns are extremely skewed. For example, upon checking `maximum_nights`, I see that there are barely any listings with above 1000, but outliers can go up to 2000. These shouldn't have any direct relation to the price bin, so I pruned and replaced outliers with median/mode for columns where I believe such action is justified. A counterexample would be `accomodates`, where usually a higher value most likely leads to a higher price.

Next up, I looked into string-based features. The first one is `amenities`, which consists of a list of strings representing the amenities that a listing has. This is a *very* open-ended column, as there are a vast amount of amenities that a host may choose to include. To this end, we look over all amenities in the training set, and analyze those that appear relatively frequently — I choose the threshold to be appear 150 times or more, in the entire training set of roughly $15,000$ entries, so a total of at least 1%. Among these, I calculate the top $k$ amenities that have a highest impact on the price (either positive or negative). For each of them, I include a binary column indicating whether this amenity is present in a listing or not.

Perhaps the most interesting column is `reviews`, where each entry has a list of string-based reviews. To make sense of this unstructured data, I applied sentiment analysis to gauge the overall tone of reviews for each listing. Specifically, I used a pre-trained multilingual BERT model, `nlptown/bert-base-multilingual-uncased-sentiment`, to compute a sentiment score for each review, ranging from very negative to very positive. After obtaining the sentiment scores for all reviews of a listing, I computed their average to produce a single numerical feature, `avg_sentiment`, representing the general sentiment for that listing. Listings without reviews were assigned a default sentiment score of 0.

After I was done with string-based columns, they were no longer needed and hence dropped, along many others that I considered useless for predictions, such as `name` and `first_review`.

One last addition is that I believed there might be more geospatial data intricacies in addition to "neighborhood groups." Thus, given pairs of longitude-latitude coordiantes, I ran a PCA on them and partitioned them into a few clusters, and created a binary column for each of them. This later turned out to be quite cool!

We are now ready to train the models.

# 2 Models

*You are required to use at least two different algorithms for generating predictions (although you can choose the best one as your Kaggle submission). These do not need to be algorithms we used in class. It would not be acceptable to use the same algorithm but with two different parameters or kernels. In this section, you will explain your reasoning behind the choice of algorithms. Specific motivations for choosing a certain algorithm may include computational efficiency, simple parameterization, ease of use, ease of training, or the avail-*

*ability of high-quality libraries online, among many other possible factors. If external libraries were used, describe them and identify the source or authors of the code (make sure to cite all references and figures that you use if someone else designed them). Try to be adventurous!*

I tried several models, including mostly random forest (RF), $k$-NN, XGBoost, as well as LightGBM.

- Intuitively $k$-NN may work because the price bins are somewhat geographically correlated: clearly, there are neighborhoods (e.g. Manhattan) that tend to have much higher rents than others. Hopefully, $k$-NN is able to capture some of these geospatial structures.
- Random Forest: Robust to noises is certainly a plus, as we are working with real-world data. It is parameter-free, so we do not need to worry about data distribution, or even the messiness of our dataset.
- XGB and LGB. High performance potential, as illustrated by some of our previous HWs as well.
- As a comic relief, I even tried neural networks, which of course clearly overfits the training set given the structure of my dataset.

## 3−5 Models / Hyperparameter Selection / Data Splits

*Here, for each of the algorithms used, briefly describe (5-6 sentences) the training algorithm used to optimize the parameter settings for that model. For example, if you used a support vector regression approach, you would probably need to reference the quadratic solver that works under-the-hood to fit the model. You may need to read the documentation for the code libraries you use to determine how the model is fit. This is part of the applied machine learning process! Also, provide estimates of runtime (either wall time or CPU time) required to train your model.*

*You also need to explain how the model hyperparameters were tuned to achieve some degree of optimality. Examples of what we consider hyperparameters are the number of trees used in a random forest model, the regularization parameter for LASSO or the type of activation / number of neurons in a neural network model. These must be chosen according to some search or heuristic. It would not be acceptable to pick a single setting of your hyperparameters and not tune them further. You also need to make at least one plot showing the functional relation between predictive accuracy on some subset of the training data and a varying hyperparameter.*

*Finally, we need to know how you split up the training data provided for cross validation. Again, briefly describe your scheme for making sure that you did not overfit to the training data.*

Firstly, to prevent overfitting, two approaches were simultaneously adopted. The first one is a $k$-fold cross validation (usually with $k = 5$) performed on `train.csv`, imported via `RandomizedSearchCV` from `sklearn`. Because I am interested also in the model's performance on unseen data, I further splitted the data into $80\% - 20\%$, where I conduct a $k$-fold on the 80%, and use the 20% for testing purposes only, so I may estimate the model's performance on the true test data. This is done via `sklearn`'s `train_test_split`.

Secondly, I also added regularization terms in model fitting, such as `gamma` in XGB.

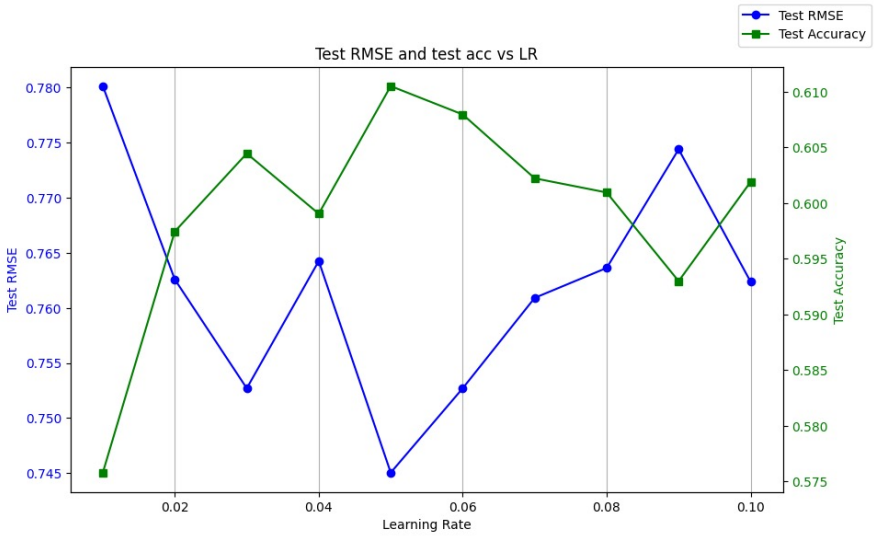
For every model, in order to find the potentially best parameter, I use a parameter grid and perform a grid search, also using `RandomizedSearchCV`. During my initial searches, I usually consider three potential values for each of the important parameters. An example for XGB is provided below,

```
param_grid = {
'n_estimators': [200, 300],
'learning_rate': [0.01, 0.02, 0.05],
'max_depth': [5, 7, 10],
'min_child_weight': [3, 4, 5],
'subsample': [0.6, 0.8, 1.0],
'gamma': [0, 0.1, 0.2]
}
```

and another example for RF here.

```
param_grid = {
'n_estimators': [200, 300],
'max_depth': [5, 7, 10, None],
'min_samples_split': [2, 5, 10],
'min_samples_leaf': [1, 2, 4],
'max_features': ['sqrt', 'log2', None],
'bootstrap': [True, False]
}
```

Some of these choices were also manually verified. For example, below is a graph where using optimal parameters but allowing a varying learning rate for XGB, we see that the two

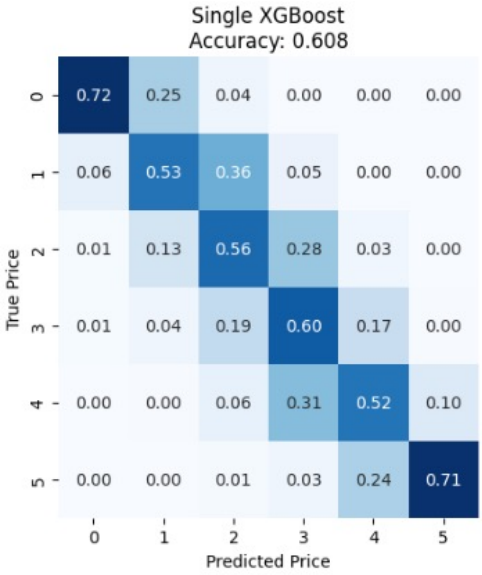



optimal choices (low RMSE, high accuracy) are 0.01 and 0.05, both of which remain in our grid.

Once I had a general idea of the rough range that the optimal parameters may fall under, I would then conduct a more refined grid search (or reduce the number of fits to increase runtime).

The regression models are loaded via `xgboost.XGBRegressor`, `lightbgm.LGBMRegressor`, `sklearn.ensemble.RandomForestRegressor`, and `sklearn.neighbors.KNeighborsRegressor`. The first two has built-in GPU acceleration, which I made use of, while the latter two don't. Overall, they all run relatively fast, with each grid search finishing in magnitudes of single minutes. Helper functions such as `sklearn.metrics.accuracy_score` were also used.

Overall, the best results were obtained by XGB, which is what I used for submission, with accuracy of roughly 60% and RMSE of around 0.75. In comparison, LGB is slightly worse. RF achieves about 50% accuracy and 0.85 RMSE, and $k$-NN achieves sub-50% accuracy and RMSE of around 0.93. Below is the confusion matrix of the best XGB model. We see that most mistakes arise in adjacent price-class misclassification, which makes sense. If we blur the boundary by counting anything within $\pm 1$ as correct, then this model in fact achieves mostly perfect accuracy. Overall I am quite satisfied with it. To combat this issue, one workaround I attempted was to train mulitple similar models with different seeds, then consider a majority vote on the predicted price bin. This helps boost the performance by a slight margin.




# 6 Reflection on Progress

*Making missteps is a natural part of the process. If there were any steps or bugs that really slowed your progress, put them here! What was the hardest part of this competition?*

There were a few. The first one had to be dealing with this messy dataset. It occurred quite a few times that my columns are not correctly formatted/filled and I had to go back and check what was going on. The biggest challenge, certainly, was to identify potential approaches that give rise to stronger models at task predictions — Which features matter, and why? What new features would boost the model's performance?

# 7 Predictive Performance

*Upload the submissions from your best model to Kaggle and put your Kaggle username in this section so we can verify that you uploaded something. Also, compare the effectiveness of the models that you used via the Root Mean Squared Error (RMSE) score that we are using to evaluate you on the Kaggle site. Half (10) of the points from this section will be awarded based on your performance relative to your peers. We will use the following formula to grade performance in the Kaggle competition:*

$$Points = \min\left(10, \left\lfloor \frac{Percentile\ Rank}{10} \right\rfloor + 1\right) \tag{1}$$

*For example, being in the 90th percentile will give you* $\frac{90}{10} + 1 = 10$ *points. Being in the 34th*

*percentile will give you* $\left\lfloor \frac{34}{10} \right\rfloor + 1 = 4$ *points.*

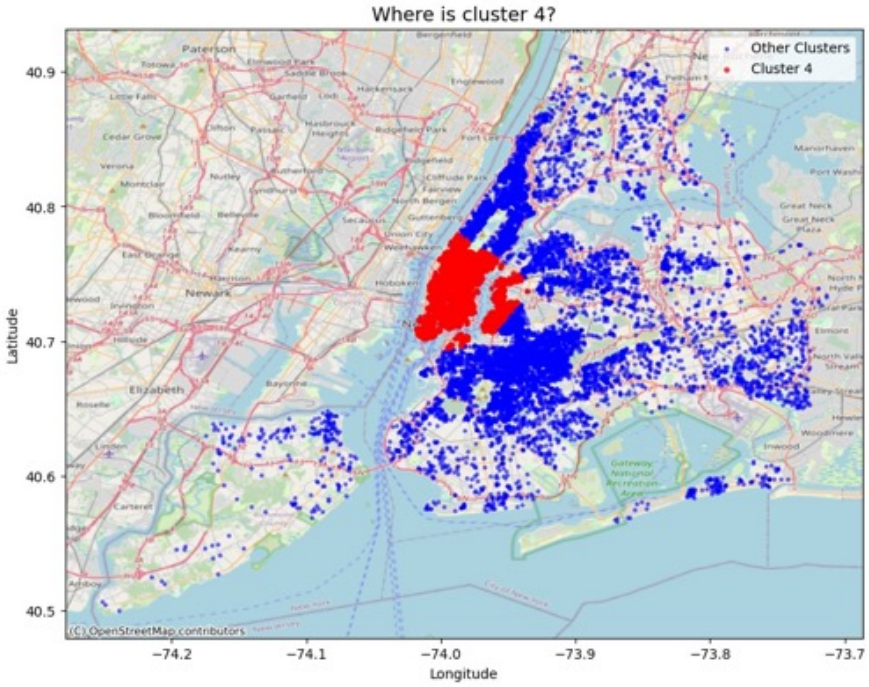
Kaggle Username: `YQL_Skorpion`, RMSE: around 0.749.

***Bonus Criteria:*** *Up to 5 points are awarded for using interpretable modeling approaches or feature engineering techniques that enhance interpretability. Possible ways to earn these points include:*

- ***Fully Interpretable Model (e.g., Decision Tree, Generalized Additive Models):*** *Use of inherently interpretable models, like a decision tree or linear/logistic regression, that allow for clear reasoning about predictions.* Unfortunately I did not use these approaches (initially tried, but pretty bad, so switched to something else).
- ***Interpretable Feature Engineering Pipeline:*** *Creation of features that are easily interpretable, or clear explanations provided for any engineered features that contribute to interpretability.* Yes, see above, for example the binary columns on "important" amenities, as well as sentiment scores and transformed binary/integer-valued columns for originally non-integer columns (like `host_response_time`).
- ***Feature Importance Analysis:*** *Provides and discusses a feature importance analysis to identify and explain key predictive factors.* Here are the top 10 important features according to the model. Many of these make sense and would intuitively be decisive factors. The only one non-intuitive is my custom-defined `cluster_4`, which, after visualization, turns out to be precisely lower Manhattan! (Not surprising, is it??) See figures on next page.

Table 1: Top 10 Most Important Features

| Feature | Importance |
|---|---|
| room_type_Private_room | 0.539124 |
| minimum_nights | 0.064432 |
| cluster_4 | 0.059632 |
| accommodates | 0.017773 |
| amenity_Dishwasher | 0.016725 |
| bedrooms | 0.014718 |
| amenity_Crib | 0.013662 |
| host_listings_count | 0.013369 |
| room_type_Shared_room | 0.012943 |
| calculated_host_listings_count_private_rooms | 0.012009 |



# 8 Code

*Copy and paste your code into your write-up document. Also, attach all the code needed for your competition. The code should be commented so that the grader can understand what is going on. Points will be taken off if the code or the comments do not explain what is taking place. Your code should be executable with the installed libraries and only minor modifications.*

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

from scipy.stats import norm
from sklearn import preprocessing
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split, KFold, ParameterSampler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,
    confusion_matrix
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from xgboost import XGBRegressor, plot_importance
from tqdm import tqdm
from transformers import pipeline
import torch




################ some exploratory data analysis, as well as data cleaning

data = pd.read_csv("data/train.csv")
pd.set_option('display.max_columns', None)
data.head()

def reload():
data = pd.read_csv("data/train.csv")
return data.copy()


# check each column's missing entry percentage
missing = data.isnull().sum()
missing = missing[missing > 0]
missing.sort_values(inplace=True)
missing_percentage = missing / len(data) * 100
missing_percentage


mode_imputer = SimpleImputer(strategy='most_frequent')
data['bathrooms'] = mode_imputer.fit_transform(data['bathrooms'].values.reshape(-1, 1))
data['bedrooms'] = mode_imputer.fit_transform(data['bedrooms'].values.reshape(-1, 1))
data['beds'] = mode_imputer.fit_transform(data['beds'].values.reshape(-1, 1))
```

```
47
48
49  # drop 'has_availability' column because all values are the same (checked via
        value_counts(), same for many columns below)
50  data['has_availability'].value_counts()
51  data.drop('has_availability', axis=1, inplace=True)
52
53
54  data['host_is_superhost'].fillna(False, inplace=True)
55  # check if we messed up column type
56  data['host_is_superhost'].value_counts() # nope
57
58
59  # check categories of bathrooms_text
60  # for bathrooms_text: if it contains 'private', set has_private_bath True.
61  # else (no bathrooms_text label or does not contain 'private'), set has_private_bath
        False
62  data['has_private_bath'] = data['bathrooms_text'].apply(lambda x: True if 'private' in
        str(x) else False)
63  data['has_private_bath'].value_counts()
64  data.drop('bathrooms_text', axis=1, inplace=True)
65
66  # description: fill missing values with 'None'
67  data['description'].fillna('None', inplace=True)
68
69
70
71  data['host_response_time'].value_counts() # 'within an hour' to 'a few days or more',
        so we integer-encode these
72  median_imputer = SimpleImputer(strategy='median')
73  data['host_acceptance_rate'] =
        median_imputer.fit_transform(data['host_acceptance_rate'].values.reshape(-1, 1))
74  data['host_response_rate'] =
        median_imputer.fit_transform(data['host_response_rate'].values.reshape(-1, 1))
75  data['host_response_time'] = data['host_response_time'].map({
76     'within an hour': 1,
77     'within a few hours': 2,
78     'within a day': 3,
79     'a few days or more': 4
80  })
81  data['host_response_time'] =
        mode_imputer.fit_transform(data['host_response_time'].values.reshape(-1, 1))
82
83
84
85  # for missing reviews: fill with medians
86  for col in ['review_scores_rating', 'review_scores_accuracy',
        'review_scores_cleanliness',
87  'review_scores_checkin', 'review_scores_communication', 'review_scores_value',
88  'review_scores_location', 'reviews_per_month']:
89  data[col].fillna(data[col].median(), inplace=True)
90
91
92
93
94  print(data[['first_review', 'last_review', 'reviews']].head())
95
96
```

```
data['host_since'] = pd.to_datetime(data['host_since'], errors='coerce')

data['first_review'] = pd.to_datetime(data['first_review'], errors='coerce')
data['last_review'] = pd.to_datetime(data['last_review'], errors='coerce')



# probably no longer needed
med_first = data['first_review'].dropna().median()
med_last = data['last_review'].dropna().median()

data['first_review'].fillna(med_first, inplace=True)
data['last_review'].fillna(med_last, inplace=True)

data['reviews'].fillna("No review provided", inplace=True)



data.describe()
sns.heatmap(data.isnull(), cbar=False) # at this point, all entries should have been
    filled already, so this plot is empty


# violin plot for every numerical column to detect obvious outliers
for col in data.select_dtypes(include=np.number).columns:
sns.violinplot(data[col])
plt.title(col)
plt.show()

# drop some
for col in data.select_dtypes(include=np.number).columns:
if col != 'price' and len(data[col].unique() > 2): # binary columns will be fine
Q1 = data[col].quantile(0.25)
Q3 = data[col].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

med = data[col].median()
data[col] = np.where(
(data[col] < lower_bound) | (data[col] > upper_bound),
med,
data[col]
)



# visualization of geospatial distribution of listings, by neighborhoods
plt.figure(figsize=(10, 8))
for neighborhood in data['neighbourhood_group_cleansed'].unique():
subset = data[data['neighbourhood_group_cleansed'] == neighborhood]
plt.scatter(subset['longitude'], subset['latitude'], label=neighborhood, alpha=0.6)

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Scatter Plot of Listings by Neighborhood')
```

```
plt.legend(loc='upper right', bbox_to_anchor=(1.25, 1))
plt.show()
```

```
data['host_verifications'].value_counts()
# if ['email', 'phone', 'work_email']: set host_verification_level to 3
# if ['email', 'phone'] or ['phone', 'work_email'], set to 2
# else set to 1
data['host_verification_level'] = 1
data.loc[data['host_verifications'].apply(lambda x: ('email' in x or 'work_email' in x) and 'phone' in x), 'host_verification_level'] = 2
data.loc[data['host_verifications'].apply(lambda x: 'email' in x and 'phone' in x and 'work_email' in x), 'host_verification_level'] = 3

data.drop('host_verifications', axis=1, inplace=True)
data['host_verification_level'].value_counts()
```

```
###################### more involved feature analysis ######################


# sentiment analysis on 'reviews'

from sentence_transformers import SentenceTransformer
from tqdm import tqdm
device = "cuda" if torch.cuda.is_available() else "cpu"
batch_size = 32

reviews = data['reviews'].fillna('none').tolist()
reviews_split = [r.split("\n------------------------------\n") for r in reviews]

sentiment_model = pipeline("sentiment-analysis", model="nlptown/bert-base-multilingual-uncased-sentiment", device=0)
sentiments = []
for review_set in tqdm(reviews_split):
review_sentiments = []
truncated_reviews = [review[:512] for review in review_set]

for i in range(0, len(truncated_reviews), batch_size):
batch = truncated_reviews[i:i + batch_size]
results = sentiment_model(batch, truncation=True, max_length=256)
review_sentiments.extend([r['score'] for r in results])

sentiments.append(review_sentiments)

data['avg_sentiment'] = [np.mean(s) if s else 0.0 for s in sentiments]

embedding_model = SentenceTransformer('all-MiniLM-L6-v2', device=device)
embeddings = []

for review_set in reviews_split:
combined_text = " ".join(review_set)
embedding = embedding_model.encode(combined_text, convert_to_tensor=False)
embeddings.append(embedding)
```

```
data['embeddings'] = embeddings # this later turns out to be unused so can be deleted, but included for sake of completeness
```

```
# 'amenities' column analysis

amenities_list = data['amenities'].apply(lambda x: eval(x) if isinstance(x, str) else []).sum()
amenities_counter = Counter(amenities_list)

# only common amenities
common_amenities = {amenity for amenity, count in amenities_counter.items() if count >= 150}

# binarize each of them
binary_columns = {
   f'has_{amenity.lower().replace(" ", "_")}': data['amenities'].apply(
   lambda x: 1 if amenity in eval(x) else 0
   )
   for amenity in common_amenities
}


binary_df = pd.DataFrame(binary_columns)
data = pd.concat([data, binary_df], axis=1)

# analyze the effect of each such amenity being present vs not being present on prices
amenity_price_analysis = []

for amenity in common_amenities:
amenity_column = f'has_{amenity.lower().replace(" ", "_")}'
has_amenity = data[data[amenity_column] == 1]['price']
no_amenity = data[data[amenity_column] == 0]['price']

amenity_price_analysis.append({
   'amenity': amenity,
   'count': len(has_amenity),
   'avg_price_with': has_amenity.mean(),
   'avg_price_without': no_amenity.mean(),
   'price_difference': has_amenity.mean() - no_amenity.mean()
})


amenity_price_df = pd.DataFrame(amenity_price_analysis)
amenity_price_df = amenity_price_df.sort_values(by='price_difference', ascending=False)

# find, plot, and keep only the amenities that have significant impacts on prices

significant_amenities = amenity_price_df[
(amenity_price_df['price_difference'] >= 1.0) | (amenity_price_df['price_difference'] <= -0.5)
]

plt.figure(figsize=(10, 6))
```

```
plt.barh(significant_amenities['amenity'], significant_amenities['price_difference'],
    color='skyblue')
plt.xlabel('Price Difference')
plt.ylabel('Amenity')
plt.title('Significant amenities')
plt.grid(axis='x')
plt.show()

data = data.drop(columns=[col for col in data.columns if col.startswith('has_')])

for amenity in significant_amenities['amenity']:
column_name = f'has_{amenity.lower().replace(" ", "_")}'
data[column_name] = data['amenities'].apply(
lambda x: 1 if amenity in eval(x) else 0
)
print(f"Added column {column_name}")

data = data.drop(columns=['amenities'])

print(data.columns)




# additional geospatial clustering

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()
data[['latitude', 'longitude']] = scaler.fit_transform(data[['latitude', 'longitude']])

kmeans = KMeans(n_clusters=8, random_state=42)
data['location_cluster'] = kmeans.fit_predict(data[['latitude', 'longitude']])

data = pd.get_dummies(data, columns=['location_cluster'], prefix='cluster')
data_test = pd.get_dummies(data_test, columns=['location_cluster'], prefix='cluster')

bool_cols = data.select_dtypes(include=[bool]).columns
data[bool_cols] = data[bool_cols].astype(int)




# final sanity check
pd.set_option('display.max_columns', None)
data.columns = data.columns.str.replace(' ', '_')
data.head()
```

```
319
320 ################################## final training ##############################
321
322
323
324 from sklearn.model_selection import train_test_split, ParameterSampler, KFold
325 from sklearn.metrics import mean_squared_error, confusion_matrix
326 from xgboost import XGBRegressor
327 from scipy.stats import mode
328
329 y = data['price']
330 X = data.drop('price', axis=1)
331
332 X_train, X_test, y_train, y_test = train_test_split(
333 X, y, test_size=0.2, random_state=42, stratify=y
334 )
335
336 # we use grid search + k-fold CV for training
337 # the same applies to other models as well; below is a sample for RGB
338 param_grid = {
339    'n_estimators': [100, 200, 300, 500],
340    'learning_rate': [0.01, 0.05, 0.1, 0.2],
341    'max_depth': [5, 7, 10],
342    'min_child_weight': [3, 4, 5],
343    'subsample': [0.6, 0.8, 1.0],
344    'gamma': [0, 0.1, 0.2]
345 }
346
347
348 # to reduce variance, we consider training (2k+1) models, each with a grid search and a
        different seed, then use majority vote
349 # this will hopefully improve the performance on adjacent price-bin misclassification
350
351 n_models = 9
352 kf = KFold(n_splits=5, shuffle=True, random_state=42)
353 models = []
354 cv_predictions = {
355    'train': [],
356    'test': []
357 }
358 param_list = list(ParameterSampler(param_grid, n_iter=n_models, random_state=42))
359
360 for params in param_list:
361 fold_train_preds = []
362 fold_test_preds = []
363 for train_idx, val_idx in kf.split(X_train):
364 X_fold_train, X_fold_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
365 y_fold_train, y_fold_val = y_train.iloc[train_idx], y_train.iloc[val_idx]
366 model = XGBRegressor(**params, objective='reg:squarederror', random_state=42)
367 model.fit(X_fold_train, y_fold_train)
368 fold_train_preds.append(model.predict(X_fold_val))
369 fold_test_preds.append(model.predict(X_test))
370 models.append(model)
371 cv_predictions['train'].append(np.concatenate(fold_train_preds))
372 cv_predictions['test'].append(np.mean(fold_test_preds, axis=0))
373
374 train_majority_vote = mode(np.round(np.array(cv_predictions['train'])), axis=0).mode[0]
375 test_majority_vote = mode(np.round(np.array(cv_predictions['test'])), axis=0).mode[0]
```

```
376
377 train_rmse = mean_squared_error(y_train, train_majority_vote, squared=False)
378 test_rmse = mean_squared_error(y_test, test_majority_vote, squared=False)
379
380 print(f"Train RMSE: {train_rmse}")
381 print(f"Test RMSE: {test_rmse}")
382
383
384
385 cm = confusion_matrix(y_test, test_majority_vote)
386 cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
387
388 plt.figure(figsize=(8, 6))
389 sns.heatmap(cm_normalized, annot=True, cmap='Blues', fmt='.2f', xticklabels=range(6),
        yticklabels=range(6))
390 plt.title(f'Majority vote confusion matrix')
391 plt.xlabel('prediction')
392 plt.ylabel('ground truth')
393 plt.show()
394
395 print("\nTop 10 Important Features:")
396 final_model = models[-1]
397 feature_importances = final_model.feature_importances_
398 important_features = sorted(zip(X.columns, feature_importances), key=lambda x: x[1],
        reverse=True)[:10]
399 for feature, importance in important_features:
400 print(f"{feature}: {importance:.4f}")
401
402 plot_importance(final_model, max_num_features=10)
403 plt.show()
```

# 9 Logistics

The report must be submitted to Gradescope by November 26th. Good luck!