MACHINE LEARNING FINAL ASSIGNMENT

Name: Prathamesh Sai Email: *saisankp@tcd.ie* Student ID: 19314123

I. Feasibility of predicting Airbnb ratings

Predicting review scores for the overall rating, accuracy, cleanliness, checkin, communication, location and value is a regression based problem as we are predicting a number between 0 and 5. For regression based problems, it is essential to choose highly correlated features to ensure the best predictions.

A. *Feature engineering*

1) Feature creation

Using the given `listings.csv` and `reviews.csv` files, I had to do some pre-processing on the data to create features out of every column. Some columns were already fine to use but some were in a format that a machine learning model would not understand, so I copied the contents of the CSV files into a new file called `updatedListings.csv` and as I read in each row I:

- Removed $ and % signs from columns.
- Converted 't' and 'f' into '1' and '0'.
- One-hot encoded columns with multiple strings in them and appended the resulting feature vector to the end of the CSV.
- Grouped more difficult columns such as amenities into entertainment, self-care, storage, wifi, leisure, kitchen, safety, parking, long-term stay, single-level home, open 24 hours, and self-checkin to one-hot encode these 12 categories.
- Converted columns such as bathrooms_text into 2 potential features such that "1.5 shared baths" = 1.5 number_of_bathrooms, 1 shared_bathroom.
- Filled in 0 for empty cells and ignored rows with no ratings (nothing to predict).
- Converted dates to UNIX time stamps.
- Ignored neighbourhood and one-hot encoded neighbourhood-cleansed instead.
- Made features host_from_ireland from host_location and multiple_hosts from host_name.
- Concatenated all the review comments for each listing ID into a list, removing all non-English words (using the nltk library), converting all emoji's into their textual equivalents (using the emoji library), extracting 20 features (bigrams only) from this list using a TfidfVectorizer and appended the resulting TF-IDF weighted document-term matrix to the end of the CSV.

2) Feature selection

Firstly, I removed columns that would not affect the ratings using human intuition such as id, listing_url, scrape_id etc. I also removed columns which had already been one-hot encoded and appended to the end of `updatedListings.csv` such as neighbourhood_cleansed. This left me with 110 columns to use as features. I realised that the data could be differentiated into binary (0 or 1) or continuous (numerical) data, and decided to choose 10 features from each. I did this so that I would end up with 20 features and avoid over-fitting with too many features. I initially chose the features by simply looking at if the ratings raised with the feature but this was insufficient and the features weren't great to begin with. Since regression models need features with a correlation, it was best to use scikit-learn's SelectKBest function using the f_regression function (which uses the F-Statistic and p-values to measure correlation). The reasoning for this is that it can provide a more accurate correlation statistic (measuring how the feature and ratings change together) to see how much each feature correlates with the ratings, and we can choose the top 10 using the SelectKBest function. I used the overall review ratings as a starting point to select features so that I would get a better idea of the features available.

2.1) Binary Features

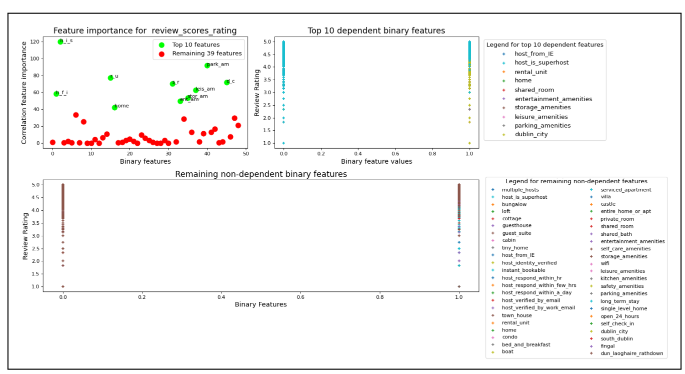

Fig. 1. Selecting initial features & visualization of binary data

In the image above, the top row shows the top 10 binary dependent features for overall review rating. The plot at the top left shows the correlation feature importance (where the top 10 features are in green), and the feature host_is_superhost (abbreviated as h_i_s and mentioned in the legend at the top right) is the highest correlated feature. We can see a smaller range on the top right plot when binary feature=1 (which indicates a correlation). The remaining non-dependent binary features are shown on the bottom row of figure 1 which show no correlation.

2.2) ***Continuous Features***

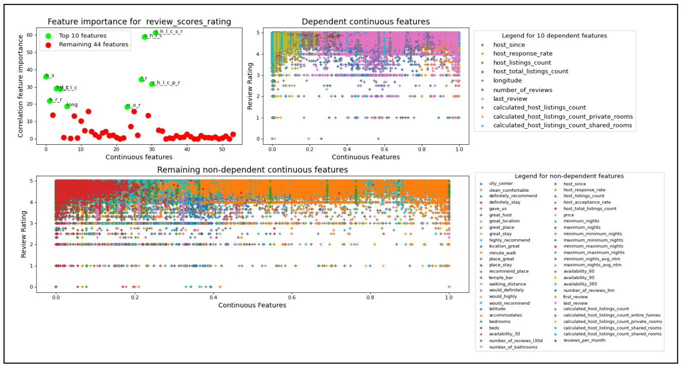

Fig. 2. Selecting initial features & visualizing continuous data

In the image above, the top row shows the top 10 continuous dependent features for overall review rating. The plot at the top left shows the correlation feature importance (where the top 10 features are in green), and the feature calculated_host_listings_count_shared_rooms (abbreviated as c_h_l_c_s_r and mentioned in the legend at the top right) is the highest correlated feature. The remaining non-dependent continuous features are shown on the bottom row of figure 2. In the top right plot, we can see that the dependent features follow a particular curve (going up and down) meanwhile the non-dependent features follow no shape or any particular curvature.

3) ***Features associated with high rating***

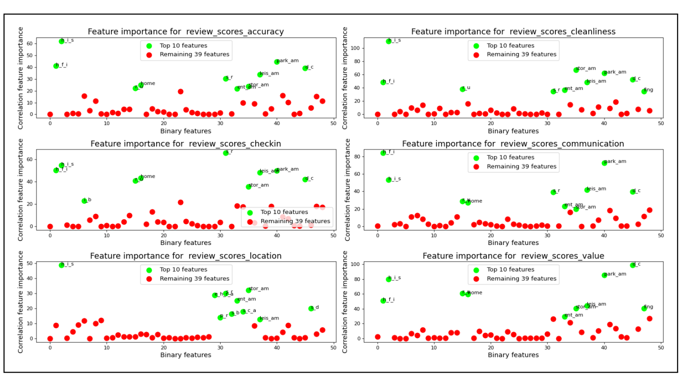

Fig. 3. Top 10 binary features for remaining types of ratings

We already saw from figure 1 that host_is_superhost was the best binary feature for overall review rating, and above we can see it is also the case for accuracy, cleanliness, and location. However, for checkin ratings it's shared_room (s_r), for communication it's host_from_ireland (h_f_i) and for value it's dublin_city (d_c). Therefore, we can say that superhosts definitely tend to have higher ratings, but things such as having a shared room, if the host is from Ireland, or if the listing is in Dublin city (which is derived from neighbourhood_cleansed) increases the ratings for different types of ratings as shown above.

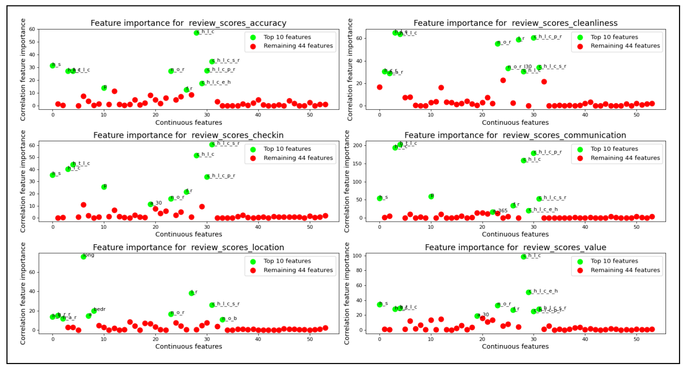

Fig. 4. Top 10 continuous features for the remaining ratings

We saw in figure 2 that the best continuous feature for overall review rating was calculated_host_listings_count_shared_rooms, and above it's also the case for checkin. However for accuracy and value it's calculated_host_listings_count (c_h_l_c), for cleanliness it's host_listings_count (h_l_c), for communication it's host_total_listings_count (h_t_l_c), and for location it's longitude (long). Therefore, we can say that listings with a higher calculated_host_listings_count_shared_rooms also tend to have higher ratings (light blue horizontal lines getting thinner as the feature increases in the top right plot of figure 2 shows this), and the number_of_reviews (n_o_r) doesn't correlate as much, but there is a small correlation (generally n_o_r is at the middle of figure 3 and 4). It's important to note that since the rating values are nearly all in the range of 4-5, using the TF-IDF weighted document-term matrix from the review comments as features ended up with poor correlation (as shown in figure 4 with the lines of red points at the end of the plots). This is because the range at which the predictions have to be within is extremely small, therefore using common words as features was not as correlated as the remaining features. Also you can notice that the 20 features chosen for predicting overall review rating also contain the top 20 features for predicting the 6 other types of ratings, so I decided to go ahead with the 20 chosen features. Before I trained models with these features, I used normalized the data using min-max scaling so everything would be in the range of 0-1 except the predicted values.

*B. **Machine Learning Methodology***

For this regression based problem, I decided to use two distinct machine learning models, kNN (k-nearest neighbours) as well as Lasso regression.

*1) **kNN (k-nearest neighbours)***

My reasoning for choosing kNN is that it is an instance based model, meaning it makes predictions based directly on the training data (represented by $(x^{(i)}, y^{(i)})$ for $i = 1, 2, ..., 6079$ from `updatedListings.csv`) which suits the problem statement at hand. We are trying to see if it is feasible with the *given data* to predict Airbnb ratings, and making predictions *based directly on the given data* is a good indicator of that. The kNN algorithm uses the distance between a training data point $x^{(i)}$ and input feature vector x (which is called the distance metric $d(x^{(i)}, x)$) and chooses the smallest distances ($d(x^{(i)}, x)$) to get the training data points closest to x (i.e. k-nearest neighbours). This is good for predicting ratings as it is instinctual that many features (such as host_is_superhost) will have similar values for similar ratings. You can also use gaussian weights where you give less of a weight to training points that are further away from x ($w^{(i)} = e^{-\gamma d(x^{(i)},x)^2}$) which also suits the fact that training points that are not similarly valued (such as one having high and low number_of_reviews) will have dissimilar ratings. The two hyperparameters to be chosen are k (number of neighbours to use) and γ (for weighting), which can be done with 5-fold cross-validation. The final kNN prediction for regression problems like ours is calculated by the average of the $y^{(i)}$ for the k closest training points:

$$\hat{y} = \frac{\sum_{i \in N_k} w^{(i)} y^{(i)}}{\sum_{i \in N_k} w^{(i)}}$$

Firstly, to choose the hyperparameter k using 5-fold cross-validation, I plotted the mean squared error (commonly abbreviated as MSE and represents the average squared difference between the predicted rating and the actual rating) vs a range of values of k from 1 to 250. We want to find the smallest value of k that yields the lowest MSE (to prevent overfitting since a big value for k will make the model fit the data too precisely and struggle with new data).

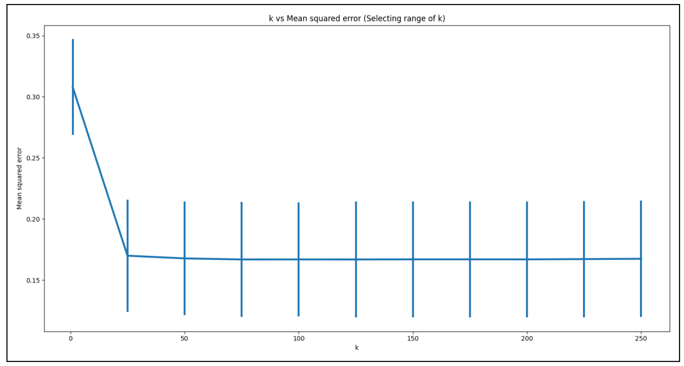

Fig. 5. Plot of MSE vs a wide range of values of k

It was evident that the MSE decreases initially but plateaus between the range of 1 and 100, so I performed 5-fold validation on this range. After plotting values in this range, the smallest value of k that yields the lowest MSE was *k=50*.

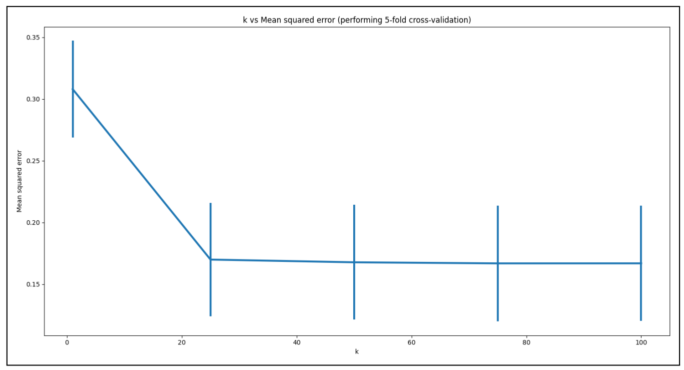

Fig. 6. Plot of MSE vs a smaller range of values of k

Secondly, to choose the hyperparameter γ using 5-fold cross-validation, I plotted the MSE vs a range of values for gamma from 10 to 150.

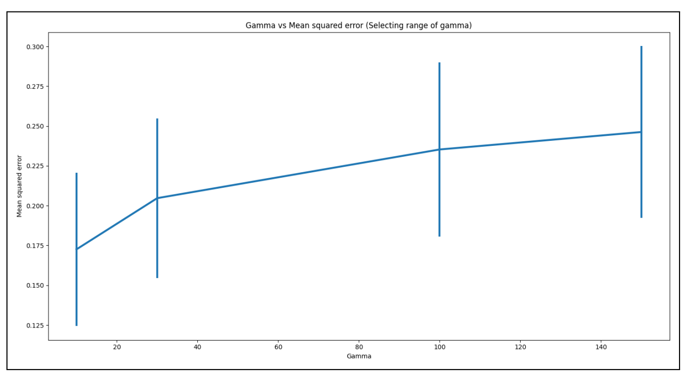

Fig. 7. Plot of MSE vs a wide range of values of γ

It was evident that the lowest value of γ (10) gave the lowest MSE, so I decided to try in the range of 1 to 10 to see if I could get a lower MSE (with caution since lowering γ too much can cause underfitting as it smooths out the function too much instead of fitting the data more precisely). After plotting values in this range, the smallest value of γ that yields the lowest MSE was *γ=1*.

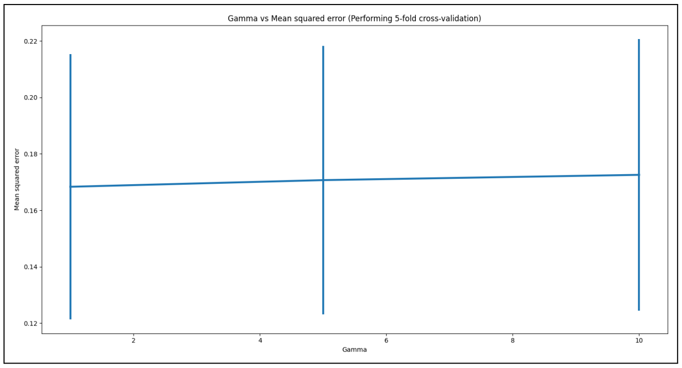

Fig. 8. Plot of MSE vs a smaller range of values of γ

*2) **Lasso regression***

My reasoning for choosing Lasso (Least Absolute Shrinkage and Selection Operator) regression is that *it performs its own version of feature selection*. Although I had performed feature selection already, I thought this model would be a good addition alongside the kNN model which makes predictions directly based on the training data from my feature selection. Lasso regression does this by the use of regularization. Lasso regression is a form of linear regression that uses this L1 regularization penalty:

$$R(\theta) = \sum_{j=1}^{n} |\theta_j|$$

This results in it encouraging sparsity of solution (few non-zero elements in θ). Therefore, unimportant features have a weight of 0 which will improve overall feature selection. This model is especially good for high levels of multicollinearity (correlation between two variables) which is evident in the problem we are solving. It is clear that many of our features will highly correlate with each other (such as host_listings_count and host_total_listings_count) so a change in one feature might cause the other to change, resulting in the model suffering. This is fixed with the use of Lasso regression. We can also try to include polynomial features which includes polynomial combinations of the features with degree less than or equal to an integer q to see if it has any impact. Therefore, the hyperparameters to be chosen are C and q. As C increases, the L1 penalty is small so there will be very little 0 values in θ. As C decreases, the L1 penalty is high so there will be many 0 values in θ.

Firstly, to choose the hyperparameter C using 5-fold cross-validation, I plotted the mean squared error vs a range of values of C from 1 to 800 to find the smallest value of C that yields the lowest MSE (to avoid overfitting).

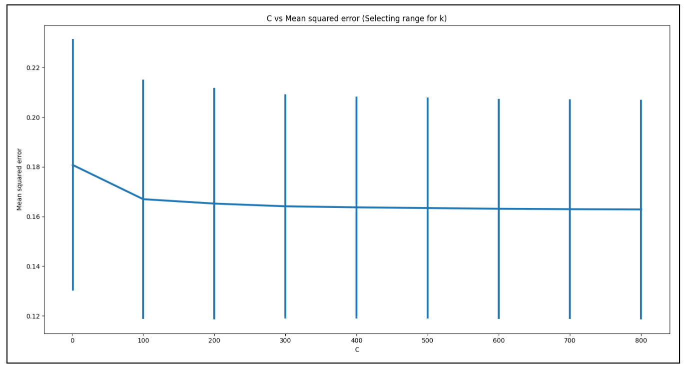

Fig. 9. Plot of MSE vs a wide range of values of C

It was evident that the MSE decreases initially but plateaus between the range of 100 and 300, so I performed 5-fold cross-validation on this range. After plotting values in this range, the smallest value of C that yields the lowest MSE was *C=300*.

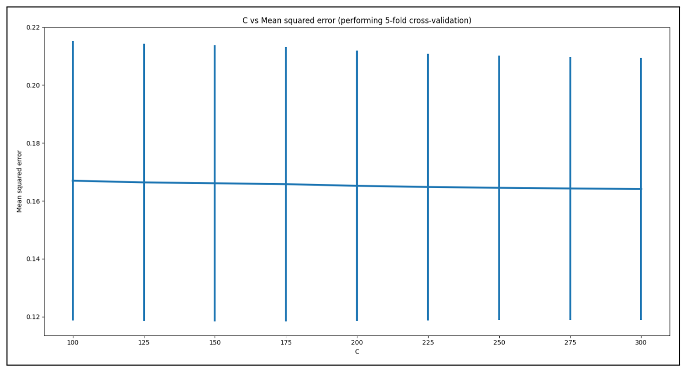

Fig. 10. Plot of MSE vs a smaller range of values of C

Secondly, to choose the hyperparameter q to try polynomial features, I used 5-fold cross-validation to plot the MSE vs a range of values for q from 1 to 4. However, there was no increase/decrease in MSE, so I decided to leave it as *q=1* which is essentially the same as not using the PolynomialFeatures function at all in terms of MSE.

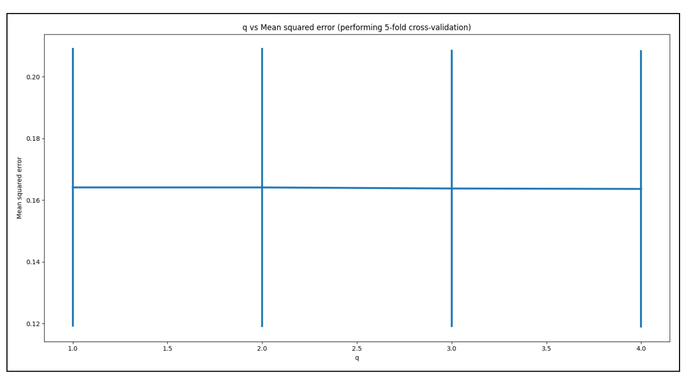

Fig. 11. Plot of MSE vs q (degree of polynomial features)

The hyperparameters for the kNN and lasso regression models were chosen with the overall rating, but the same values for k, γ, C and q are optimal even with different types of ratings.

C. Evaluation

To evaluate my models, I split the data into training and testing using a 80/20 split and trained my 2 models with a dummy regressor that always predicts the mean of the training set to compare them to a baseline. In addition to the MSE being used to evaluate their performance, it is important to note that most ratings are already within the range of 4-5. Therefore, it is necessary to judge how well the model fits the data to get a better judgement of its predictions. This can be done with the R^2 value, which tells us how well the model fits the data using a measure from 0 to 100 representing the proportion of variance in the dependent variable that can be explained by the independent variable i.e. the goodness of fit.

1) Mean Squared Error

Below are the MSE values for each of the models predicting each of the rating types (rounded to 3 decimal places). The results show us that the lasso regression model consistently has the lowest MSE. The kNN model is slightly worse but slightly better than the dummy regressor, but only by a small margin. This shows that the lasso regression model is the best suited to predict all types of rating. The rating with the lowest MSE is communication however it's important to keep in mind that the dummy regressor's MSE was generally only 0.005 to 0.01 above the lasso regression model for every rating. This shows that although I completed thorough feature selection (with lasso regression's own feature selection with the L1 penalty too) alongside optimal hyperparameters C and q, it's not feasible to predict these ratings using the data provided. The kNN model that makes predictions directly based on the training data only being negligibly better also proves this.

Rating type	kNN	Lasso	Dummy
Overall	0.162	0.157	0.176
Accuracy	0.121	0.119	0.124
Cleanliness	0.299	0.290	0.318
Checkin	0.172	0.168	0.178
Communication	0.095	0.091	0.100
Location	0.116	0.114	0.119
Value	0.198	0.192	0.206

2) R-squared

Below are the R^2 values for each of the models predicting each of the rating types (rounded to 5 decimal places). The results once again show us that the lasso regression model is the best, since it consistently has the highest R^2 value while predicting every type of rating (best fit to the data). The kNN model is worse than the lasso regression model but still better than the dummy regressor which always has $R^2 \approx 0$. Unfortunately, overall the R^2 values are fairly low, and this once again shows that it is infeasible to predict the ratings as the data doesn't work well with the models, even if thorough feature selection and hyperparameter tuning is completed.

Rating type	kNN	Lasso	Dummy
Overall	0.08167	0.10811	-0.00126
Accuracy	0.01564	0.03093	-0.00545
Cleanliness	0.05716	0.08543	-0.00039
Checkin	0.03303	0.05385	-0.00244
Communication	0.04776	0.08423	-0.00072
Location	0.02652	0.04029	-0.00005
Value	0.03695	0.0691	0.00013

D. Conclusion

To predict various types of Airbnb ratings, I trained a kNN and lasso regression model with features that are as correlated as possible to the particular rating that is being predicted using the provided data. After finetuning the hyperparameters for each model, it was found that it is infeasible to predict for a listing the individual ratings for accuracy, cleanliness, checkin, communication, location and value and also the overall review rating. This is because most of the features are not highly correlated to the Airbnb ratings to begin with (as shown by the line of red points in figure 3 and 4). Even after careful feature selection and avoiding underfitting/overfitting, the models don't have a low MSE or high R^2 value compared to the dummy regressor that simply always predicts the mean of the training set. This means that our models were most likely not worth it, as they only barely out-compete the dummy regressor. Therefore, it is not feasible to apply any real world use to these models. My recommendation would be to instead find more highly correlated features from new data (with the same level of correlation of features such as host_is_superhost or calculated_host_listings_count_shared_rooms or higher). Since the range to make predictions (mainly between 4 and 5) is extremely small, they have to be extremely accurate. Finding more highly correlated features will drastically improve the prediction accuracy as the models will have better features to make predictions from.

II. Questions

Part (i)

There can be times when logistic regression would give inaccurate predictions:

1) The logistic regression model calculates $\theta^T x$ to predict +1 ($\theta^T x > 0$) or -1 ($\theta^T x < 0$), where a line called the decision boundary (when $\theta^T x = 0$) is formed which informs the model where to start switching predictions from one type to another. When data is not linearly separable (when a straight line can't be drawn between the two types of the data to separate them), which is usually the case with real world data, the logistic regression model will struggle to make accurate predictions as the decision boundary is not 100% accurate (wrong types of data on either side of it).
2) As mentioned, the logistic regression model calculates a decision boundary to make predictions. If the training data is not enough or is relatively small (with most likely poor quality data since the size is small), it will cause the logistic regression model to give inaccurate predictions as there is not enough data to define where exactly the switching boundary is and therefore it has to make do with the limited data and make poor predictions.

Part (ii)

There are some advantages and disadvantages between kNN classifier and a MLP neural net classifier:

kNN classifier advantages:

- Very easy to use, as it only has 1 parameter (k, which is the number of neighbours to use) if you use the default uniform weighting.
- There is no training period since all the predictions are based directly on the training data.
- Works well with small data only (meaning less work needed to collect this data) as explained below.

kNN classifier disadvantages:

- If the dataset is large, it will struggle because it needs to calculate the distance between the point we're interested in and all of the training data, then sort it to find the closest one. This is computationally expensive and takes time.
- It struggles to extrapolate outside of the range of values in the training data since doesn't extrapolate well outside the range of the training data. i.e. it is only good for interpolation (predicting within the range of the training data).
- To make new predictions it still needs the training data since it's an instance based model (makes predictions directly based on training data).

MLP neural net classifier advantages:

- Once trained, when making new predictions it doesn't need the training data anymore.
- Can be optimised and trained using stochastic gradient descent which is an optimization algorithm that minimizes the cost function by iteratively changing the weights of the network with mini-batches of size q to approximate the actual gradient.
- Not restricted to one output, so it can be used for multi-label classification for things like images since it can have many outputs.

MLP neural net classifier disadvantages:

- Tricky/slow to use, as it has lots of parameters (more hyperparameter tuning) and weights which are hard to interpret (acting as a black box model where you just give an input and get some output).
- Generally require more data than regular machine learning models such as kNN to be accurate.
- Hard to train since the cost function is non convex in weights/parameters so it may not converge or converge to a bad local minimum.

Part (iii)

We care about how well a model generalises (i.e. how it performs with unseen data). The idea behind resampling a dataset multiple times during k-fold cross-validation is that we evaluate the model on a new portion of the data each time, so that we can evaluate the generalisation performance on a larger range of tests. We would split our data into k equal sized parts, and 1 part would go to testing and k-1 parts (the rest) would go to training, and each time we resample we choose a different partition for testing and the rest for training to get the spread of values for the prediction error (hence seeing how our model performs multiple times with unseen data). If we didn't resample the data multiple times, we wouldn't get as accurate of a idea of the generalisation performance as it would be over a limited amount of tests (unlike resampling where we can use new unseen testing data k times to get a more accurate generalisation performance). k=5 is recommended because it correlates nicely to a training/testing split of 80/20. This is because for every resampling, while 1 part is used for testing (i.e. 20%), the remaining 4 parts (i.e. k-1) are used for training (i.e. 80%). k=10 is recommended because it correlates nicely to a training/testing split of 90/10. This is because for every resampling, while 1 part is used for testing (i.e. 10%), the remaining 9 parts (i.e. k-1) are used for training (i.e. 90%).

Part (iv)

Lagged output values can be used to construct features for time series data with the assumption that things that happened in the past are likely to be similar in the future. Therefore, if you have a dataset containing information about the past, you can shift the data so that you predict a value at t by using data from t-1. I can illustrate this with an example - if you have data measuring the number of cars a salesman sells every day as such:

Date (t)	Cars sold
1/12/2022	2
2/12/2022	3
3/12/2022	1
4/12/2022	3
5/12/2022	5
6/12/2022	3
7/12/2022	2
8/12/2022	4
9/12/2022	1
10/12/2022	2

You could shift the data such that you create new input features that allow your model to predict the cars sold at t by using the data from t-1 as such:

cars(t-1)	cars(t)
NaN	2
2	3
3	1
1	3
3	5
5	3
3	2
2	4
4	1
1	2

Now we can simply ignore the first row (with incomplete data), and we have made the lagged feature cars(t-1) which has the cars sold from the past (t-1) to predict the cars sold for it's relative future (t i.e. the original table). This is known commonly as the window shift method, and in this case we used a window width of 1 but you could increase the window width and add more features if you wish.

1 Appendix

1.1 main.py

```
from sklearn.dummy import DummyRegressor
from sklearn.metrics import mean_squared_error, r2_score
from kNN import *
from lasso import *
from feature_creation import *
from feature_selection import *
import numpy as np

matplotlib.use("TkAgg")

if __name__ == '__main__':
    # Step 1: Feature engineering

    # Step 1 (A): Feature creation:
    # Concatenate all review comment text for each listing ID and store it
    in a list to extract features.
    listOfReviews =
    getListOfReviewsForEachListingID("../data/reviews.csv",
    "../data/listings.csv")
    # Get the features and TF-IDF weighted document-term matrix from the
    list of reviews.
    featuresNamesFromReviewComments, TFIDF_Matrix =
    featureExtraction(listOfReviews)
    # Copy the current data into another file, and while doing so we
    simultaneously do pre-processing and add features
    preprocessing("../data/listings.csv", "../data/updated-listings.csv",
    featuresNamesFromReviewComments, TFIDF_Matrix)

    # Step 1 (B): Feature selection
    # Remove useless columns that could not act as features
    deleteUnnecessaryFeatures("../data/updated-listings.csv")
    # Plot dependent vs non-dependent binary features
    showBinaryFeatures("../data/updated-listings.csv")
    # Plot dependent vs non-dependent continuous features
    showContinuousFeatures("../data/updated-listings.csv")

    # Step 2. Train machine learning models and evaluate them

    dataframe = pd.read_csv("../data/updated-listings.csv")
    scaler = MinMaxScaler()

    # Chosen binary features
    host_from_ireland = dataframe.iloc[:, 2]
    host_is_superhost = dataframe.iloc[:, 5]
    rental_unit = dataframe.iloc[:, 55]
    home = dataframe.iloc[:, 56]
    shared_room = dataframe.iloc[:, 71]
    entertainment_amenities = dataframe.iloc[:, 74]
    storage_amenities = dataframe.iloc[:, 76]
    leisure_amenities = dataframe.iloc[:, 78]
```

```
parking_amenities = dataframe.iloc[:, 81]
dublin_city = dataframe.iloc[:, 86]

# Chosen continuous features
host_since = scaler.fit_transform(dataframe.iloc[:,
1].values.reshape(-1, 1))
host_response_rate = scaler.fit_transform(dataframe.iloc[:,
3].values.reshape(-1, 1))
host_listings_count = scaler.fit_transform(dataframe.iloc[:,
6].values.reshape(-1, 1))
host_total_listings_count = scaler.fit_transform(dataframe.iloc[:,
7].values.reshape(-1, 1))
longitude = scaler.fit_transform(dataframe.iloc[:,
11].values.reshape(-1, 1))
number_of_reviews = scaler.fit_transform(dataframe.iloc[:,
29].values.reshape(-1, 1))
last_review = scaler.fit_transform(dataframe.iloc[:,
33].values.reshape(-1, 1))
calculated_host_listings_count =
scaler.fit_transform(dataframe.iloc[:, 42].values.reshape(-1, 1))
calculated_host_listings_count_private_rooms =
scaler.fit_transform(dataframe.iloc[:, 44].values.reshape(-1, 1))
calculated_host_listings_count_shared_rooms =
scaler.fit_transform(dataframe.iloc[:, 45].values.reshape(-1, 1))

# Predicting these values
review_scores_rating = dataframe.iloc[:, 34]
review_scores_accuracy = dataframe.iloc[:, 35]
review_scores_cleanliness = dataframe.iloc[:, 36]
review_scores_checkin = dataframe.iloc[:, 37]
review_scores_communication = dataframe.iloc[:, 38]
review_scores_location = dataframe.iloc[:, 39]
review_scores_value = dataframe.iloc[:, 40]

X = np.column_stack((host_from_ireland, host_is_superhost,
rental_unit, home, shared_room, entertainment_amenities,
                     storage_amenities, leisure_amenities,
                     parking_amenities, dublin_city, host_since,
                     host_response_rate, host_listings_count,
                     host_total_listings_count, longitude,
                     number_of_reviews,
                     last_review, calculated_host_listings_count,
                     calculated_host_listings_count_private_rooms,
                     calculated_host_listings_count_shared_rooms))

np.set_printoptions(suppress=True)

# Tune hyperparameters (values for k, gamma, C and q end up being the
same for predicting every type of rating [y])
y = review_scores_rating

# 5-Fold cross validation tells us we should use k=50 and gamma=1
select_k_range(X, y)
choose_k_using_CV(X, y)
select_kNN_gamma_range_for_CV(X, y)
```

```
choose_kNN_gamma_using_CV(X, y)

# 5-Fold cross validation tells us we should use C=300 and q=1
select_c_range(X, y)
choose_c_using_CV(X, y)
choose_q_using_CV(X, y)

# Step 2 (A). Predicting review_scores_rating
print("1. Predicting review_scores_rating:")
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = kNN(x_train, y_train)
y_pred = model.predict(x_test)
print("kNN MSE predicting review_scores_rating: " +
str(mean_squared_error(y_test, y_pred)))
print("kNN R-Squared predicting review_scores_rating: " +
str(r2_score(np.array(y_test), np.array(y_pred))))

model = lassoRegression(x_train, y_train)
y_pred = model.predict(x_test)
print("Lasso Regression MSE predicting review_scores_rating: " +
str(mean_squared_error(y_test, y_pred)))
print("Lasso Regression R-Squared predicting review_scores_rating: " +
str(
    r2_score(np.array(y_test), np.array(y_pred))))

dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
y_pred = dummyModel.predict(x_test)
print("Dummy Regressor MSE predicting review_scores_rating: " +
str(mean_squared_error(y_test, y_pred)))
print("Dummy Regressor R-Squared predicting review_scores_rating: " +
str(r2_score(np.array(y_test), y_pred)))

# Step 2 (B). Predicting review_scores_accuracy
print("2. Predicting review_scores_accuracy:")
y = review_scores_accuracy
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = kNN(x_train, y_train)
y_pred = model.predict(x_test)
print("kNN MSE predicting review_scores_rating: " +
str(mean_squared_error(y_test, y_pred)))
print("kNN R-Squared predicting review_scores_rating: " +
str(r2_score(np.array(y_test), np.array(y_pred))))

model = lassoRegression(x_train, y_train)
y_pred = model.predict(x_test)
print("Lasso Regression MSE predicting review_scores_rating: " +
str(mean_squared_error(y_test, y_pred)))
print("Lasso Regression R-Squared predicting review_scores_rating: " +
str(
    r2_score(np.array(y_test), np.array(y_pred))))
```

```
dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
y_pred = dummyModel.predict(x_test)
print("Dummy Regressor MSE predicting review_scores_rating: " +
str(mean_squared_error(y_test, y_pred)))
print("Dummy Regressor R-Squared predicting review_scores_rating: " +
str(r2_score(np.array(y_test), y_pred)))

# Step 2 (C). Predicting review_scores_cleanliness
print("3. Predicting review_scores_cleanliness:")
y = review_scores_cleanliness
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = kNN(x_train, y_train)
y_pred = model.predict(x_test)
print("kNN MSE predicting review_scores_cleanliness: " +
str(mean_squared_error(y_test, y_pred)))
print("kNN R-Squared predicting review_scores_cleanliness: " +
str(r2_score(np.array(y_test), np.array(y_pred))))

model = lassoRegression(x_train, y_train)
y_pred = model.predict(x_test)
print("Lasso Regression MSE predicting review_scores_cleanliness: " +
str(mean_squared_error(y_test, y_pred)))
print("Lasso Regression R-Squared predicting
review_scores_cleanliness: " + str(
    r2_score(np.array(y_test), np.array(y_pred))))

dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
y_pred = dummyModel.predict(x_test)
print("Dummy Regressor MSE predicting review_scores_cleanliness: " +
str(mean_squared_error(y_test, y_pred)))
print("Dummy Regressor R-Squared predicting review_scores_cleanliness:
" + str(r2_score(np.array(y_test), y_pred)))

# Step 2 (D). Predicting review_scores_checkin
print("4. Predicting review_scores_checkin:")
y = review_scores_checkin
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = kNN(x_train, y_train)
y_pred = model.predict(x_test)
print("kNN MSE predicting review_scores_checkin: " +
str(mean_squared_error(y_test, y_pred)))
print("kNN R-Squared predicting review_scores_checkin: " +
str(r2_score(np.array(y_test), np.array(y_pred))))

model = lassoRegression(x_train, y_train)
y_pred = model.predict(x_test)
print("Lasso Regression MSE predicting review_scores_checkin: " +
str(mean_squared_error(y_test, y_pred)))
print("Lasso Regression R-Squared predicting review_scores_checkin: "
+ str(
    r2_score(np.array(y_test), np.array(y_pred))))
```

```
dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
y_pred = dummyModel.predict(x_test)
print("Dummy Regressor MSE predicting review_scores_checkin: " +
str(mean_squared_error(y_test, y_pred)))
print("Dummy Regressor R-Squared predicting review_scores_checkin: " +
str(r2_score(np.array(y_test), y_pred)))

# Step 2 (E). Predicting review_scores_communication
print("5. Predicting review_scores_communication:")
y = review_scores_communication
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = kNN(x_train, y_train)
y_pred = model.predict(x_test)
print("kNN MSE predicting review_scores_communication: " +
str(mean_squared_error(y_test, y_pred)))
print("kNN R-Squared predicting review_scores_communication: " +
str(r2_score(np.array(y_test), np.array(y_pred))))

model = lassoRegression(x_train, y_train)
y_pred = model.predict(x_test)
print("Lasso Regression MSE predicting review_scores_communication: "
+ str(mean_squared_error(y_test, y_pred)))
print("Lasso Regression R-Squared predicting
review_scores_communication: " + str(
    r2_score(np.array(y_test), np.array(y_pred))))

dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
y_pred = dummyModel.predict(x_test)
print("Dummy Regressor MSE predicting review_scores_communication: " +
str(mean_squared_error(y_test, y_pred)))
print(
    "Dummy Regressor R-Squared predicting review_scores_communication:
    " + str(r2_score(np.array(y_test), y_pred)))

# Step 2 (F). Predicting review_scores_location
print("6. Predicting review_scores_location:")
y = review_scores_location
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

model = kNN(x_train, y_train)
y_pred = model.predict(x_test)
print("kNN MSE predicting review_scores_location: " +
str(mean_squared_error(y_test, y_pred)))
print("kNN R-Squared predicting review_scores_location: " +
str(r2_score(np.array(y_test), np.array(y_pred))))

model = lassoRegression(x_train, y_train)
y_pred = model.predict(x_test)
print("Lasso Regression MSE predicting review_scores_location: " +
str(mean_squared_error(y_test, y_pred)))
```

```
    print("Lasso Regression R-Squared predicting review_scores_location: "
    + str(
        r2_score(np.array(y_test), np.array(y_pred))))

    dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
    y_pred = dummyModel.predict(x_test)
    print("Dummy Regressor MSE predicting review_scores_location: " +
    str(mean_squared_error(y_test, y_pred)))
    print("Dummy Regressor R-Squared predicting review_scores_location: "
    + str(r2_score(np.array(y_test), y_pred)))

    # Step 2 (G). Predicting review_scores_location
    print("7. Predicting review_scores_value:")
    y = review_scores_value
    x_train, x_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2)

    model = kNN(x_train, y_train)
    y_pred = model.predict(x_test)
    print("kNN MSE predicting review_scores_value: " +
    str(mean_squared_error(y_test, y_pred)))
    print("kNN R-Squared predicting review_scores_value: " +
    str(r2_score(np.array(y_test), np.array(y_pred))))

    model = lassoRegression(x_train, y_train)
    y_pred = model.predict(x_test)
    print("Lasso Regression MSE predicting review_scores_value: " +
    str(mean_squared_error(y_test, y_pred)))
    print("Lasso Regression R-Squared predicting review_scores_value: " +
    str(
        r2_score(np.array(y_test), np.array(y_pred))))

    dummyModel = DummyRegressor(strategy="mean").fit(x_train, y_train)
    y_pred = dummyModel.predict(x_test)
    print("Dummy Regressor MSE predicting review_scores_value: " +
    str(mean_squared_error(y_test, y_pred)))
    print("Dummy Regressor R-Squared predicting review_scores_value: " +
    str(r2_score(np.array(y_test), y_pred)))
```

1.2 feature_creation.py

```
from sklearn.feature_extraction.text import TfidfVectorizer
import emoji
from datetime import datetime
import csv
import re
import nltk
import warnings

warnings.filterwarnings("ignore")
nltk.download('words')
nltk.download('stopwords')
words = set(nltk.corpus.words.words())
```

```
# Using the review comment text for the Airbnb listings,  We need to
choose 25 features (i.e. words) that
# occur in the reviews for each listing and store them in a list to add
them to the updated-listings file later.
def getListOfReviewsForEachListingID(reviewsCSV, listingsCSV):
    dictionary_with_id_mapped_to_reviews = dict()
    with open(reviewsCSV) as inputFile:
        reader = csv.reader(inputFile.readlines())
        header = True
        for line in reader:
            if header:
                header = False
            else:
                reviewWithEnglishOnly = " ".join(
                    x for x in nltk.wordpunct_tokenize(str(line[5])) if
                    x.lower() in words or not x.isalpha())
                reviewsWithCleanedText = emoji.demojize(
                    " ".join((reviewWithEnglishOnly.replace("< />",
                    "").replace("<br/>", "")).split()))
                dictionary_with_id_mapped_to_reviews[line[0]] = str(
                    dictionary_with_id_mapped_to_reviews.get(line[0]) or
                    "") + " " + reviewsWithCleanedText
    listOfReviewsInOrder = []
    with open(listingsCSV) as inputFile:
        reader = csv.reader(inputFile.readlines())
        header = True
        for line in reader:
            if header:
                header = False
            else:
                listOfReviewsInOrder.append(dictionary_with_id_mapped_to_
                reviews.get(line[0]) or "")
    return listOfReviewsInOrder


# Use the list of reviews for each listing ID to get features and a TF-IDF
weighted document-term matrix.
def featureExtraction(listOfReviewsInOrder):
    vectorizer = TfidfVectorizer(norm="l2",
    stop_words=nltk.corpus.stopwords.words("english"), ngram_range=(2, 2),
                                 max_features=20)
    X = vectorizer.fit_transform(listOfReviewsInOrder)
    return vectorizer.get_feature_names_out(), X.toarray()


# Copy listing data to a new file while simultaneously doing
pre-processing and adding previous features
def preprocessing(listingsCSV, updatedListingsCSV,
featuresNamesFromReviewComments, TFIDF_Matrix):
    with open(listingsCSV) as inputFile:
        reader = csv.reader(inputFile.readlines())
    with open(updatedListingsCSV, 'w') as outputFile:
        writer = csv.writer(outputFile)
```

```
    header = True
    index = 0
    for line in reader:
        if header:
            line[11] = "multiple_hosts"
            line[13] = "host_from_ireland"
            # Extra features we will make from the existing columns
            with one-hot encodings.
            line.append("host_respond_within_an_hour")
            line.append("host_respond_within_a_few_hours")
            line.append("host_respond_within_a_day")
            line.append("host_verified_by_phone")
            line.append("host_verified_by_email")
            line.append("host_verified_by_work_email")
            line.append("bungalow")
            line.append("town_house")
            line.append("rental_unit")
            line.append("home")
            line.append("loft")
            line.append("condo")
            line.append("cottage")
            line.append("guesthouse")
            line.append("bed_and_breakfast")
            line.append("boat")
            line.append("serviced_apartment")
            line.append("guest_suite")
            line.append("cabin")
            line.append("villa")
            line.append("castle")
            line.append("tiny_home")
            line.append("entire_home_or_apt")
            line.append("private_room")
            line.append("shared_room")
            line.append("number_of_bathrooms")
            line.append("shared_bath")
            line.append("entertainment_amenities")
            line.append("self_care_amenities")
            line.append("storage_amenities")
            line.append("wifi")
            line.append("leisure_amenities")
            line.append("kitchen_amenities")
            line.append("safety_amenities")
            line.append("parking_amenities")
            line.append("long_term_stay")
            line.append("single_level_home")
            line.append("open_24_hours")
            line.append("self_check_in")
            line.append("dublin_city")
            line.append("south_dublin")
            line.append("fingal")
            line.append("dun_laoghaire_rathdown")
            for feature in featuresNamesFromReviewComments:
                line.append(feature)
            header = False
        else:
```

```
                # If any of the review ratings are empty then don't keep
                this line (as we cannot predict anything)
                if line[61] == "" or line[62] == "" or line[63] == "" or
                line[64] == "" or line[65] == "" or line[
                    66] == "" or line[67] == "":
                    continue
                # We have 63 new rows from the new features, so add 0 in
                all their columns for now.
                for x in range(63):
                    line.append(0)
                # Do preprocessing to make data cleaner
                convertStringDateToUNIX(line)
                removeDollarAndPercentageSigns(line)
                convertColumnsToNumbers(line)
                groupAmenitiesAndOneHotEncoding(line)
                addFeatureValuesFromReviews(line, TFIDF_Matrix[index])
                index = index + 1
            writer.writerow(line)
        writer.writerows(reader)


def convertStringDateToUNIX(line):
    # Convert last_scraped column into UNIX timestamps.
    if any(chr.isdigit() for chr in line[3]):
        line[3] = datetime.fromisoformat(line[3]).timestamp()

    # Convert host_since column into UNIX timestamps.
    if any(chr.isdigit() for chr in line[12]):
        line[12] = datetime.fromisoformat(line[12]).timestamp()

    # Convert calendar_last_scraped column into UNIX timestamps.
    if any(chr.isdigit() for chr in line[55]):
        line[55] = datetime.fromisoformat(line[55]).timestamp()

    # Convert first_review column into UNIX timestamps.
    if any(chr.isdigit() for chr in line[59]):
        line[59] = datetime.fromisoformat(line[59]).timestamp()

    # Convert last_review column into UNIX timestamps.
    if any(chr.isdigit() for chr in line[60]):
        line[60] = datetime.fromisoformat(line[60]).timestamp()


def removeDollarAndPercentageSigns(line):
    # Remove dollar sign and ".00" in price column.
    if "$" in line[40]:
        line[40] = float(line[40].replace("$", "").replace(",", ""))

    # Remove percentage sign or 'N/A' in host_response_rate column.
    if "%" in line[16]:
        line[16] = float(line[16].replace("%", ""))
    elif "N/A" in line[16]:
        line[16] = 0

    # Remove percentage sign or 'N/A' in host_acceptance_rate column.
```

```
    if "%" in line[17]:
        line[17] = float(line[17].replace("%", ""))
    elif "N/A" in line[17]:
        line[17] = 0


def convertColumnsToNumbers(line):
    # Change source column to 1 or 0 (from 'city scrape' or 'previous
    scrape' previously).
    if line[4] == "city scrape":
        line[4] = 1
    elif line[4] == "previous scrape":
        line[4] = 0

    # Change host_name column to 1 or 0 (depending on if there are 1 or 2
    hosts)
    if "And" in line[11]:
        line[11] = 1
    elif "&" in line[11]:
        line[11] = 1
    else:
        line[11] = 0

    # Change host_location column to 1 or 0 (depending on if they are from
    Ireland or not)
    if "Ireland" in line[13]:
        line[13] = 1
    else:
        line[13] = 0

    # Use one-hot encodings with host_response_time column to make 3
    features (i.e. columns):
    # 1. host_respond_within_an_hour
    # 2. host_respond_within_a_few_hours
    # 3. host_respond_within_a_day
    # Using a binary matrix (where N/A or False is 0, and 1 is True).
    if line[15] == "within an hour":
        line[75] = 1
    elif line[15] == "within a few hours":
        line[76] = 1
    elif line[15] == "within a day":
        line[77] = 1

    # Change host_is_superhost column to 1 or 0 (from 't' [true] or 'f'
    [false] previously).
    if line[18] == "t":
        line[18] = 1
    elif line[18] == "f":
        line[18] = 0

    # Use one-hot encodings with host_verifications column to make 3
    features (i.e. columns):
    # 1. host_verified_by_phone
    # 2. host_verified_by_email
    # 3. host_verified_by_work_email
```

```python
# Using a binary matrix (False is 0, and 1 is True).
if "'phone'" in line[24]:
    line[78] = 1
if "'email'" in line[24]:
    line[79] = 1
if "'work_email'" in line[24]:
    line[80] = 1

# Change host_has_profile_pic column to 1 or 0 (from 't' [true] or 'f'
[false] previously).
if line[25] == "t":
    line[25] = 1
elif line[25] == "f":
    line[25] = 0

# Change host_identity_verified column to 1 or 0 (from 't' [true] or
'f' [false] previously).
if line[26] == "t":
    line[26] = 1
elif line[26] == "f":
    line[26] = 0

# Use one-hot encodings with neighbourhood_cleansed column to make 4
features (i.e. columns):
# 1. dublin_city
# 2. south_dublin
# 3. fingal
# 4. dun_laoghaire_rathdown
if "Dublin City" in line[28]:
    line[114] = 1
elif "South Dublin" in line[28]:
    line[115] = 1
elif "Fingal" in line[28]:
    line[116] = 1
elif "Dn Laoghaire-Rathdown" in line[28]:
    line[117] = 1

# Use one-hot encodings with property_type column to make 16 features
(i.e. columns):
# 1. bungalow
# 2. town_house
# 3. rental_unit
# 4. home
# 5. loft
# 6. condo
# 7. cottage
# 8. guesthouse
# 9. bed_and_breakfast
# 10. boat
# 11. serviced_apartment
# 12. guest_suite
# 13. cabin
# 14. villa
# 15. castle
# 16. tiny_home
```

```
# Using a binary matrix (False is 0, and 1 is True).
if "bungalow" in line[32]:
    line[81] = 1
elif "townhouse" in line[32]:
    line[82] = 1
elif "rental unit" in line[32]:
    line[83] = 1
elif "Tiny home" in line[32]:
    line[96] = 1
elif "home" in line[32]:
    line[84] = 1
elif "loft" in line[32]:
    line[85] = 1
elif "condo" in line[32]:
    line[86] = 1
elif "cottage" in line[32]:
    line[87] = 1
elif "guesthouse" in line[32]:
    line[88] = 1
elif "bed and breakfast" in line[32]:
    line[89] = 1
elif "boat" in line[32]:
    line[90] = 1
elif "serviced apartment" in line[32]:
    line[91] = 1
elif "guest suite" in line[32]:
    line[92] = 1
elif "cabin" in line[32]:
    line[93] = 1
elif "villa" in line[32]:
    line[94] = 1
elif "Castle" in line[32]:
    line[95] = 1

# Use one-hot encodings with room_type column to make 3 features (i.e.
columns):
# 1. entire_home_or_apt
# 2. private_room
# 3. shared_room
# Using a binary matrix (False is 0, and 1 is True).
if line[33] == "Entire home/apt":
    line[97] = 1
elif line[33] == "Private room":
    line[98] = 1
elif line[33] == "Shared room":
    line[99] = 1

# Split bathrooms_text column to make 2 features (i.e. columns):
# 1. number_of_bathrooms
# 2. shared_bath
# where number_of_bathrooms is simply the float in the bathrooms_text
column, and shared_bath is a one-hot encoding.

# Ensure the string representing baths is in a format so that we can
use regex.
```

```
    if "Shared half-bath" in line[36]:
        line[36] = "0.5 shared bath"
    elif "Private half-bath" in line[36]:
        line[36] = "0.5 private bath"
    elif "Half-bath" in line[36]:
        line[36] = "0.5 bath"
    elif line[36] == "":
        line[36] = "0"

    # Remove characters and get only the number, then z`store it into the
    number_of_bathrooms column.
    line[100] = float(re.sub(r'[a-z]', '', line[36].lower()))
    if "shared" in line[36]:
        line[101] = 1

    # Change empty values in bedrooms column and beds column to 0.
    if line[37] == "":
        line[37] = 0
    if line[38] == "":
        line[38] = 0

    # Change has_availability column to 1 or 0 (from 't' [true] or 'f'
    [false] previously).
    if line[50] == "t":
        line[50] = 1
    elif line[50] == "f":
        line[50] = 0

    # Change instant_bookable column to 1 or 0 (from 't' [true] or 'f'
    [false] previously).
    if line[69] == "t":
        line[69] = 1
    elif line[69] == "f":
        line[69] = 0


def groupAmenitiesAndOneHotEncoding(line):
    amenitiesString = str(
        line[39][1:-1].replace('"', '').replace(", ",
        ",").replace("\\u2013", "").replace("\\u2019", "").replace(
            "\\u00", ""))
    amenitiesList = amenitiesString.split(",")

    # Group previously 1000+ amenities into 12 groups and use one-hot
    encoding to make 12 new features (i.e. columns)
    for amenity in amenitiesList:
        # 1. entertainment_amenities
        if "TV".casefold() in amenity.casefold():
            line[102] = 1
        elif "game".casefold() in amenity.casefold():
            line[102] = 1
        elif "video".casefold() in amenity.casefold():
            line[102] = 1
        elif "PS".casefold() in amenity.casefold():  # Playstation
        (PS3/PS4 etc)
```

```
        line[102] = 1
    elif "sound".casefold() in amenity.casefold():
        line[102] = 1
    elif "HBO".casefold() in amenity.casefold():
        line[102] = 1
    elif "chromecast".casefold() in amenity.casefold():
        line[102] = 1
    elif "netflix".casefold() in amenity.casefold():
        line[102] = 1
    elif "ping".casefold() in amenity.casefold():
        line[102] = 1
    elif "toys".casefold() in amenity.casefold():
        line[102] = 1
    elif "player".casefold() in amenity.casefold():
        line[102] = 1
    elif "roku".casefold() in amenity.casefold():
        line[102] = 1
    elif "ethernet".casefold() in amenity.casefold():
        line[102] = 1
    elif "cable".casefold() in amenity.casefold():
        line[102] = 1
    elif "piano".casefold() in amenity.casefold():
        line[102] = 1

    # 2. self_care_amenities
    elif "conditioner".casefold() in amenity.casefold():
        line[103] = 1
    elif "shampoo".casefold() in amenity.casefold():
        line[103] = 1
    elif "soap".casefold() in amenity.casefold():
        line[103] = 1
    elif "hair".casefold() in amenity.casefold():
        line[103] = 1
    elif "shower".casefold() in amenity.casefold():
        line[103] = 1
    elif "essentials".casefold() in amenity.casefold():
        line[103] = 1
    elif "bidet".casefold() in amenity.casefold():
        line[103] = 1
    elif "iron".casefold() in amenity.casefold():
        line[103] = 1
    elif "washer".casefold() in amenity.casefold():
        line[103] = 1
    elif "dryer".casefold() in amenity.casefold():
        line[103] = 1
    elif "bath".casefold() in amenity.casefold():
        line[103] = 1
    elif "Laundromat".casefold() in amenity.casefold():
        line[103] = 1
    elif "shades".casefold() in amenity.casefold():
        line[103] = 1
    elif "blanket".casefold() in amenity.casefold():
        line[103] = 1
    elif "fan".casefold() in amenity.casefold():
        line[103] = 1
```

```python
        elif "hot".casefold() in amenity.casefold():
            line[103] = 1
        elif "linen".casefold() in amenity.casefold():
            line[103] = 1
        elif "comfort".casefold() in amenity.casefold():
            line[103] = 1

        # 3. storage_amenities
        elif "storage".casefold() in amenity.casefold():
            line[104] = 1
        elif "wardrobe".casefold() in amenity.casefold():
            line[104] = 1
        elif "dresser".casefold() in amenity.casefold():
            line[104] = 1
        elif "hanger".casefold() in amenity.casefold():
            line[104] = 1
        elif "table".casefold() in amenity.casefold():
            line[104] = 1
        elif "closet".casefold() in amenity.casefold():
            line[104] = 1
        elif "luggage".casefold() in amenity.casefold():
            line[104] = 1

        # 4. wifi
        elif "wifi".casefold() in amenity.casefold():
            line[105] = 1

        # 5. leisure_amenities
        elif "gym".casefold() in amenity.casefold():
            line[106] = 1
        elif "pool".casefold() in amenity.casefold():
            line[106] = 1
        elif "lake".casefold() in amenity.casefold():
            line[106] = 1
        elif "sauna".casefold() in amenity.casefold():
            line[106] = 1
        elif "tub".casefold() in amenity.casefold():  # Bathtub or hot tub
            line[106] = 1
        elif "kayak".casefold() in amenity.casefold():
            line[106] = 1
        elif "balcony".casefold() in amenity.casefold():
            line[106] = 1
        elif "AC".casefold() in amenity.casefold():
            line[106] = 1
        elif "air".casefold() in amenity.casefold():
            line[106] = 1
        elif "heat".casefold() in amenity.casefold():
            line[106] = 1
        elif "fire pit".casefold() in amenity.casefold():
            line[106] = 1
        elif "waterfront".casefold() in amenity.casefold():
            line[106] = 1
        elif "bikes".casefold() in amenity.casefold():
            line[106] = 1
        elif "boat".casefold() in amenity.casefold():
```

```
        line[106] = 1
    elif "ski".casefold() in amenity.casefold():
        line[106] = 1
    elif "babysitter".casefold() in amenity.casefold():
        line[106] = 1
    elif "staff".casefold() in amenity.casefold():
        line[106] = 1
    elif "elevator".casefold() in amenity.casefold():
        line[106] = 1
    elif "outdoor".casefold() in amenity.casefold():
        line[106] = 1
    elif "pets".casefold() in amenity.casefold():
        line[106] = 1
    elif "glass top".casefold() in amenity.casefold():
        line[106] = 1

    # 6. kitchen_amenities
    elif "fri".casefold() in amenity.casefold():  # Fridge or
    refrigerator
        line[107] = 1
    elif "oven".casefold() in amenity.casefold():
        line[107] = 1
    elif "stove".casefold() in amenity.casefold():
        line[107] = 1
    elif "toaster".casefold() in amenity.casefold():
        line[107] = 1
    elif "dish".casefold() in amenity.casefold():
        line[107] = 1
    elif "nespresso".casefold() in amenity.casefold():
        line[107] = 1
    elif "maker".casefold() in amenity.casefold():  #
    Coffee/Bread/Rice maker
        line[107] = 1
    elif "kettle".casefold() in amenity.casefold():
        line[107] = 1
    elif "glasses".casefold() in amenity.casefold():
        line[107] = 1
    elif "baking".casefold() in amenity.casefold():
        line[107] = 1
    elif "cooking".casefold() in amenity.casefold():
        line[107] = 1
    elif "coffee".casefold() in amenity.casefold():
        line[107] = 1
    elif "grill".casefold() in amenity.casefold():
        line[107] = 1
    elif "freezer".casefold() in amenity.casefold():
        line[107] = 1
    elif "barbecue".casefold() in amenity.casefold():
        line[107] = 1
    elif "microwave".casefold() in amenity.casefold():
        line[107] = 1
    elif "dining".casefold() in amenity.casefold():
        line[107] = 1
    elif "breakfast".casefold() in amenity.casefold():
        line[107] = 1
```

```
    elif "dinner".casefold() in amenity.casefold():
        line[107] = 1
    elif "kitchen".casefold() in amenity.casefold():
        line[107] = 1

    # 7. safety_amenities
    elif "alarm".casefold() in amenity.casefold():
        line[108] = 1
    elif "lock".casefold() in amenity.casefold():
        line[108] = 1
    elif "mosquito".casefold() in amenity.casefold():
        line[108] = 1
    elif "monitor".casefold() in amenity.casefold():
        line[108] = 1
    elif "guard".casefold() in amenity.casefold():
        line[108] = 1
    elif "gate".casefold() in amenity.casefold():
        line[108] = 1
    elif "first aid".casefold() in amenity.casefold():
        line[108] = 1
    elif "cover".casefold() in amenity.casefold():
        line[108] = 1
    elif "security".casefold() in amenity.casefold():
        line[108] = 1
    elif "extinguisher".casefold() in amenity.casefold():
        line[108] = 1
    elif "safe".casefold() in amenity.casefold():
        line[108] = 1
    elif "clean".casefold() in amenity.casefold():
        line[108] = 1
    elif "crib".casefold() in amenity.casefold():
        line[108] = 1
    elif "keypad".casefold() in amenity.casefold():
        line[108] = 1
    elif "Private entrance".casefold() in amenity.casefold():
        line[108] = 1

    # 8. parking_amenities
    elif "park".casefold() in amenity.casefold():
        line[109] = 1
    elif "carport".casefold() in amenity.casefold():
        line[109] = 1
    elif "garage".casefold() in amenity.casefold():
        line[109] = 1
    elif "EV charger".casefold() in amenity.casefold():
        line[109] = 1

    # 9. long_term_stay
    elif "long".casefold() in amenity.casefold():
        line[110] = 1

    # 10. single_level_home
    elif "single level" in amenity.casefold():
        line[111] = 1
```

```
        # 11. open_24_hours
        elif "open 24 hours" in amenity.casefold():
            line[112] = 1

        # 12. self_check_in
        elif "self check-in" in amenity.casefold():
            line[113] = 1


# Add the new features using common words in review comments
def addFeatureValuesFromReviews(line, values):
    for i in range(20):
        line[118] = values[0]
        line[119] = values[1]
        line[120] = values[2]
        line[121] = values[3]
        line[122] = values[4]
        line[123] = values[5]
        line[124] = values[6]
        line[125] = values[7]
        line[126] = values[8]
        line[127] = values[9]
        line[128] = values[10]
        line[129] = values[11]
        line[130] = values[12]
        line[131] = values[13]
        line[132] = values[14]
        line[133] = values[15]
        line[134] = values[16]
        line[135] = values[17]
        line[136] = values[18]
        line[137] = values[19]
```

1.3 feature_selection.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

matplotlib.use("TkAgg")


# Delete features from CSV that are not related to rating (using intuition)
def deleteUnnecessaryFeatures(updatedListingsCSV):
    data = pd.read_csv(updatedListingsCSV)
    # id column has no dependence on the ratings
    data.drop('id', inplace=True, axis=1)
    # listing_url column has no dependence on the ratings
    data.drop('listing_url', inplace=True, axis=1)
```

```
# scrape_id column has no dependence on the ratings
data.drop('scrape_id', inplace=True, axis=1)
# last_scraped column has no dependence on the ratings
data.drop('last_scraped', inplace=True, axis=1)
# source column has no dependence on the ratings
data.drop('source', inplace=True, axis=1)
# name column contains details we already have from other features
such as: bedrooms,
# neighbourhood_cleansed, property_type etc
data.drop('name', inplace=True, axis=1)
# description column contains details we already have from other
features such as: bedrooms,
# neighbourhood_cleansed, property_type and features from review
comments.
data.drop('description', inplace=True, axis=1)
# neighborhood_overview column contains details we already have from
other features such as
# neighbourhood_cleansed and the features from review comments.
data.drop('neighborhood_overview', inplace=True, axis=1)
# picture_url column has no dependence on the ratings
data.drop('picture_url', inplace=True, axis=1)
# host_id column has no dependence on the ratings
data.drop('host_id', inplace=True, axis=1)
# host_url column has no dependence on the ratings
data.drop('host_url', inplace=True, axis=1)
# host_about column contains details we already have from other
features such as: host_response_time,
# host_response_rate, host_acceptance rate and features from review
comments.
data.drop('host_about', inplace=True, axis=1)
# host_response_time column has been one-hot encoded already (appended
at end of CSV), so we delete this
data.drop('host_response_time', inplace=True, axis=1)
# host_thumbnail_url column has no dependence on the ratings
data.drop('host_thumbnail_url', inplace=True, axis=1)
# host_picture_url column has no dependence on the ratings
data.drop('host_picture_url', inplace=True, axis=1)
# host_neighbourhood column has little dependence on the rating (if it
did, it would already be covered
# by the host's actions from features host_response_time,
host_response_rate, host_acceptance rate etc.)
data.drop('host_neighbourhood', inplace=True, axis=1)
# host_verifications column has been one-hot encoded already (appended
at end of CSV), so we delete this
data.drop('host_verifications', inplace=True, axis=1)
# neighbourhood column is more simply described by the feature
neighbourhood_cleansed, so delete this
data.drop('neighbourhood', inplace=True, axis=1)
# neighbourhood_cleansed column has been one-hot encoded already
(appended at end of CSV), so we delete this
data.drop('neighbourhood_cleansed', inplace=True, axis=1)
# neighbourhood_group_cleansed column is empty, so delete this
data.drop('neighbourhood_group_cleansed', inplace=True, axis=1)
# property_type column has been one-hot encoded already (appended at
end of CSV), so we delete this
```

```
    data.drop('property_type', inplace=True, axis=1)
    # room_type column has been one-hot encoded already (appended at end
    of CSV), so we delete this
    data.drop('room_type', inplace=True, axis=1)
    # bathrooms column is empty, so delete this
    data.drop('bathrooms', inplace=True, axis=1)
    # bathrooms_text column has been one-hot encoded already (appended at
    end of CSV), so we delete this
    data.drop('bathrooms_text', inplace=True, axis=1)
    # amenities column has been one-hot encoded already (appended at end
    of CSV), so we delete this
    data.drop('amenities', inplace=True, axis=1)
    # calendar_updated column is empty, so delete this
    data.drop('calendar_updated', inplace=True, axis=1)
    # calendar_last_scraped column is not dependent on the ratings
    data.drop('calendar_last_scraped', inplace=True, axis=1)
    # license column is empty, so delete this
    data.drop('license', inplace=True, axis=1)
    data.to_csv(updatedListingsCSV, index=False, sep=',')


# Use correlation statistics to choose top 10 features for ratings
def select_features_from_correlation(X, y, dictionaryWithFeatures,
location, typeOfFeatures, typeOfRatings):
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.33, random_state=1)
    fs = SelectKBest(score_func=f_regression, k=10)
    fs.fit(X_train, y_train)
    topTenFeatures = fs.get_feature_names_out()
    plt.ylabel("Correlation feature importance", fontsize=13)
    plt.xlabel(typeOfFeatures + " features", fontsize=13)
    plt.title("Feature importance for  " + typeOfRatings, fontsize=16)
    index = 0
    for score in fs.scores_:
        if "x" + str(index) in topTenFeatures:
            plt.scatter(index, score, marker="o", s=100, c='lime')
            plt.annotate(dictionaryWithFeatures[index], (index, score),
            size=10)
        else:
            plt.scatter(index, score, marker="o", s=100, c='r')
        index = index + 1
    if typeOfFeatures == "Binary":
        plt.legend(loc=location, labels=["Top 10 features", "Remaining 39
        features"], fontsize=12)
    else:
        plt.legend(loc=location, labels=["Top 10 features", "Remaining 44
        features"], fontsize=12)
    ax = plt.gca()
    leg = ax.get_legend()
    leg.legendHandles[0].set_color('lime')
    leg.legendHandles[1].set_color('red')


# Show plots with chosen top 10 binary features and remaining 39
non-dependent features
```

```python
def showBinaryFeatures(updatedListingsCSV):
    dataframe = pd.read_csv(updatedListingsCSV)

    # Potential binary features:
    multiple_hosts = dataframe.iloc[:, 0]
    host_from_ireland = dataframe.iloc[:, 2]
    host_is_superhost = dataframe.iloc[:, 5]
    host_has_profile_pic = dataframe.iloc[:, 8]
    host_identity_verified = dataframe.iloc[:, 9]
    has_availability = dataframe.iloc[:, 24]
    instant_bookable = dataframe.iloc[:, 41]
    host_respond_within_an_hour = dataframe.iloc[:, 47]
    host_respond_within_a_few_hours = dataframe.iloc[:, 48]
    host_respond_within_a_day = dataframe.iloc[:, 49]
    host_verified_by_phone = dataframe.iloc[:, 50]
    host_verified_by_email = dataframe.iloc[:, 51]
    host_verified_by_work_email = dataframe.iloc[:, 52]
    bungalow = dataframe.iloc[:, 53]
    town_house = dataframe.iloc[:, 54]
    rental_unit = dataframe.iloc[:, 55]
    home = dataframe.iloc[:, 56]
    loft = dataframe.iloc[:, 57]
    condo = dataframe.iloc[:, 58]
    cottage = dataframe.iloc[:, 59]
    guesthouse = dataframe.iloc[:, 60]
    bed_and_breakfast = dataframe.iloc[:, 61]
    boat = dataframe.iloc[:, 62]
    serviced_apartment = dataframe.iloc[:, 63]
    guest_suite = dataframe.iloc[:, 64]
    cabin = dataframe.iloc[:, 65]
    villa = dataframe.iloc[:, 66]
    castle = dataframe.iloc[:, 67]
    tiny_home = dataframe.iloc[:, 68]
    entire_home_or_apt = dataframe.iloc[:, 69]
    private_room = dataframe.iloc[:, 70]
    shared_room = dataframe.iloc[:, 71]
    shared_bath = dataframe.iloc[:, 73]
    entertainment_amenities = dataframe.iloc[:, 74]
    self_care_amenities = dataframe.iloc[:, 75]
    storage_amenities = dataframe.iloc[:, 76]
    wifi = dataframe.iloc[:, 77]
    leisure_amenities = dataframe.iloc[:, 78]
    kitchen_amenities = dataframe.iloc[:, 79]
    safety_amenities = dataframe.iloc[:, 80]
    parking_amenities = dataframe.iloc[:, 81]
    long_term_stay = dataframe.iloc[:, 82]
    single_level_home = dataframe.iloc[:, 83]
    open_24_hours = dataframe.iloc[:, 84]
    self_check_in = dataframe.iloc[:, 85]
    dublin_city = dataframe.iloc[:, 86]
    south_dublin = dataframe.iloc[:, 87]
    fingal = dataframe.iloc[:, 88]
    dun_laoghaire_rathdown = dataframe.iloc[:, 89]
```

```
# Use abbreviated words for annotating top 10 features for feature
importance plot
binaryFeatures = dict()
binaryFeatures[0] = "m_h"
binaryFeatures[1] = "h_f_i"
binaryFeatures[2] = "h_i_s"
binaryFeatures[3] = "h_h_p_p"
binaryFeatures[4] = "h_i_v"
binaryFeatures[5] = "h_a"
binaryFeatures[6] = "i_b"
binaryFeatures[7] = "h_r_w_a_h"
binaryFeatures[8] = "h_r_w_a_f_h"
binaryFeatures[9] = "h_r_w_a_d"
binaryFeatures[10] = "h_v_b_p"
binaryFeatures[11] = "h_v_b_e"
binaryFeatures[12] = "h_v_b_w_e"
binaryFeatures[13] = "bun"
binaryFeatures[14] = "t_h"
binaryFeatures[15] = "r_u"
binaryFeatures[16] = "home"
binaryFeatures[17] = "loft"
binaryFeatures[18] = "con"
binaryFeatures[19] = "cot"
binaryFeatures[20] = "guesthouse"
binaryFeatures[21] = "b_a_b"
binaryFeatures[22] = "boat"
binaryFeatures[23] = "s_apart"
binaryFeatures[24] = "g_s"
binaryFeatures[25] = "cab"
binaryFeatures[26] = "villa"
binaryFeatures[27] = "cast"
binaryFeatures[28] = "t_h"
binaryFeatures[29] = "e_h_r_a"
binaryFeatures[30] = "p_r"
binaryFeatures[31] = "s_r"
binaryFeatures[32] = "s_b"
binaryFeatures[33] = "ent_am"
binaryFeatures[34] = "s_c_a"
binaryFeatures[35] = "stor_am"
binaryFeatures[36] = "wifi"
binaryFeatures[37] = "leis_am"
binaryFeatures[38] = "kitc_am"
binaryFeatures[39] = "saf_am"
binaryFeatures[40] = "park_am"
binaryFeatures[41] = "l_t_s"
binaryFeatures[42] = "s_l_h"
binaryFeatures[43] = "o_24"
binaryFeatures[44] = "s_c_i"
binaryFeatures[45] = "d_c"
binaryFeatures[46] = "s_d"
binaryFeatures[47] = "fing"
binaryFeatures[48] = "d_l_r"

# Predicting these values
review_scores_rating = dataframe.iloc[:, 34]
```

```
review_scores_accuracy = dataframe.iloc[:, 35]
review_scores_cleanliness = dataframe.iloc[:, 36]
review_scores_checkin = dataframe.iloc[:, 37]
review_scores_communication = dataframe.iloc[:, 38]
review_scores_location = dataframe.iloc[:, 39]
review_scores_value = dataframe.iloc[:, 40]

X = np.column_stack((multiple_hosts, host_from_ireland,
host_is_superhost, host_has_profile_pic,
                    host_identity_verified, has_availability,
                    instant_bookable, host_respond_within_an_hour,
                    host_respond_within_a_few_hours,
                    host_respond_within_a_day, host_verified_by_phone,
                    host_verified_by_email,
                    host_verified_by_work_email, bungalow,
                    town_house, rental_unit,
                    home, loft, condo, cottage, guesthouse,
                    bed_and_breakfast, boat, serviced_apartment,
                    guest_suite, cabin, villa, castle, tiny_home,
                    entire_home_or_apt, private_room, shared_room,
                    shared_bath, entertainment_amenities,
                    self_care_amenities, storage_amenities, wifi,
                    leisure_amenities, kitchen_amenities,
                    safety_amenities, parking_amenities,
                    long_term_stay,
                    single_level_home, open_24_hours, self_check_in,
                    dublin_city, south_dublin, fingal,
                    dun_laoghaire_rathdown))

y = review_scores_rating
plt.rcParams["figure.constrained_layout.use"] = True
plt.figure(figsize=(50, 30), dpi=80, tight_layout=True)
plt.subplot(2, 2, 1)

# Find correlated features for review_scores_rating
select_features_from_correlation(X, y, binaryFeatures, "upper right",
"Binary", "review_scores_rating")
# From correlation feature selection, we see that these binary
features are dependent:
# host_from_ireland, host_is_superhost, rental_unit, home,
shared_room, entertainment_amenities,
# storage_amenities, leisure_amenities, parking_amenities, dublin_city

# 1. Plotting binary features that do have a dependence:
plt.subplot(2, 2, 2)

host_from_ireland_vs_review_scores_rating =
plt.scatter(host_from_ireland, review_scores_rating, marker="+",
  label="host_from_IE")
rental_unit_vs_review_scores_rating = plt.scatter(rental_unit,
review_scores_rating, marker="+", label="rental_unit")
home_vs_review_scores_rating = plt.scatter(home, review_scores_rating,
marker="+", label="home")
shared_room_vs_review_scores_rating = plt.scatter(shared_room,
review_scores_rating, marker="+", label="shared_room")
```

```
entertainment_amenities_vs_review_scores_rating =
plt.scatter(entertainment_amenities, review_scores_rating,
        marker="+", label="entertainment_amenities")
storage_amenities_vs_review_scores_rating =
plt.scatter(storage_amenities, review_scores_rating, marker="+",
  label="storage_amenities")
leisure_amenities_vs_review_scores_rating =
plt.scatter(leisure_amenities, review_scores_rating, marker="+",
  label="leisure_amenities")
parking_amenities_vs_review_scores_rating =
plt.scatter(parking_amenities, review_scores_rating, marker="+",
  label="parking_amenities")
dublin_city_vs_review_scores_rating = plt.scatter(dublin_city,
review_scores_rating, marker="+", label="dublin_city")
host_is_superhost_vs_review_scores_rating =
plt.scatter(host_is_superhost, review_scores_rating, marker="+",
  label="host_is_superhost")

plt.ylabel("Review Rating", fontsize=13)
plt.xlabel("Binary feature values", fontsize=13)
plt.title("Top 10 dependent binary features", fontsize=16)
plt.legend(handles=[host_from_ireland_vs_review_scores_rating,
                    host_is_superhost_vs_review_scores_rating,
                    rental_unit_vs_review_scores_rating,
                    home_vs_review_scores_rating,
                    shared_room_vs_review_scores_rating,
                    entertainment_amenities_vs_review_scores_rating,
                    storage_amenities_vs_review_scores_rating,
                    leisure_amenities_vs_review_scores_rating,
                    parking_amenities_vs_review_scores_rating,
                    dublin_city_vs_review_scores_rating],
                    title='Legend for top 10 dependent features',
           bbox_to_anchor=(1.01, 1), loc='upper left', fontsize=12,
           title_fontsize=12)

# 2. Plotting remaining features with no/weak dependence:
plt.subplot(2, 1, 2)

multiple_hosts_vs_review_scores_rating = plt.scatter(multiple_hosts,
review_scores_rating, marker="+",
label="multiple_hosts")

bungalow_vs_review_scores_rating = plt.scatter(bungalow,
review_scores_rating, marker="+", label="bungalow")

loft_vs_review_scores_rating = plt.scatter(loft, review_scores_rating,
marker="+", label="loft")

cottage_vs_review_scores_rating = plt.scatter(cottage,
review_scores_rating, marker="+", label="cottage")

guesthouse_vs_review_scores_rating = plt.scatter(guesthouse,
review_scores_rating, marker="+", label="guesthouse")
```

```
guest_suite_vs_review_scores_rating = plt.scatter(guest_suite,
review_scores_rating, marker="+", label="guest_suite")

cabin_vs_review_scores_rating = plt.scatter(cabin,
review_scores_rating, marker="+", label="cabin")

tiny_home_vs_review_scores_rating = plt.scatter(tiny_home,
review_scores_rating, marker="+", label="tiny_home")

host_identity_verified_vs_review_scores_rating =
plt.scatter(host_identity_verified, review_scores_rating,
       marker="+", label="host_identity_verified")

instant_bookable_vs_review_scores_rating =
plt.scatter(instant_bookable, review_scores_rating, marker="+",
 label="instant_bookable")

host_respond_within_an_hour_vs_review_scores_rating =
plt.scatter(host_respond_within_an_hour, review_scores_rating,
            marker="+", label="host_respond_within_hr")

host_respond_within_a_few_hours_vs_review_scores_rating =
plt.scatter(host_respond_within_a_few_hours,
                review_scores_rating, marker="+",
                label="host_respond_within_few_hrs")

host_respond_within_a_day_vs_review_scores_rating =
plt.scatter(host_respond_within_a_day, review_scores_rating,
          marker="+", label="host_respond_within_a_day")

host_verified_by_email_vs_review_scores_rating =
plt.scatter(host_verified_by_email, review_scores_rating,
       marker="+", label="host_verified_by_email")

host_verified_by_work_email_vs_review_scores_rating =
plt.scatter(host_verified_by_work_email, review_scores_rating,
            marker="+", label="host_verified_by_work_email")

town_house_vs_review_scores_rating = plt.scatter(town_house,
review_scores_rating, marker="+", label="town_house")

condo_vs_review_scores_rating = plt.scatter(condo,
review_scores_rating, marker="+", label="condo")

bed_and_breakfast_vs_review_scores_rating =
plt.scatter(bed_and_breakfast, review_scores_rating, marker="+",
  label="bed_and_breakfast")

boat_vs_review_scores_rating = plt.scatter(boat, review_scores_rating,
marker="+", label="boat")

serviced_apartment_vs_review_scores_rating =
plt.scatter(serviced_apartment, review_scores_rating, marker="+",
   label="serviced_apartment")
```

```
villa_vs_review_scores_rating = plt.scatter(villa,
review_scores_rating, marker="+", label="villa")

castle_vs_review_scores_rating = plt.scatter(castle,
review_scores_rating, marker="+", label="castle")

entire_home_or_apt_vs_review_scores_rating =
plt.scatter(entire_home_or_apt, review_scores_rating, marker="+",
   label="entire_home_or_apt")

private_room_vs_review_scores_rating = plt.scatter(private_room,
review_scores_rating, marker="+", label="private_room")

shared_bath_vs_review_scores_rating = plt.scatter(shared_bath,
review_scores_rating, marker="+", label="shared_bath")

self_care_amenities_vs_review_scores_rating =
plt.scatter(self_care_amenities, review_scores_rating, marker="+",
    label="self_care_amenities")

wifi_vs_review_scores_rating = plt.scatter(wifi, review_scores_rating,
marker="+", label="wifi")

kitchen_amenities_vs_review_scores_rating =
plt.scatter(kitchen_amenities, review_scores_rating, marker="+",
  label="kitchen_amenities")

safety_amenities_vs_review_scores_rating =
plt.scatter(safety_amenities, review_scores_rating, marker="+",
 label="safety_amenities")

long_term_stay_vs_review_scores_rating = plt.scatter(long_term_stay,
review_scores_rating, marker="+",
label="long_term_stay")

single_level_home_vs_review_scores_rating =
plt.scatter(single_level_home, review_scores_rating, marker="+",
  label="single_level_home")

open_24_hours_vs_review_scores_rating = plt.scatter(open_24_hours,
review_scores_rating, marker="+",
label="open_24_hours")

self_check_in_vs_review_scores_rating = plt.scatter(self_check_in,
review_scores_rating, marker="+", label="self_check_in")

south_dublin_vs_review_scores_rating = plt.scatter(south_dublin,
review_scores_rating, marker="+", label="south_dublin")

fingal_vs_review_scores_rating = plt.scatter(fingal,
review_scores_rating, marker="+", label="fingal")

dun_laoghaire_rathdown_vs_review_scores_rating =
plt.scatter(dun_laoghaire_rathdown, review_scores_rating,
       marker="+", label="dun_laoghaire_rathdown")
```

```
plt.ylabel("Review Rating", fontsize=13)
plt.xlabel("Binary Features", fontsize=13)
plt.title("Remaining non-dependent binary features", fontsize=16)
plt.legend(handles=[multiple_hosts_vs_review_scores_rating,
                    host_is_superhost_vs_review_scores_rating,
                    bungalow_vs_review_scores_rating,
                    loft_vs_review_scores_rating,
                    cottage_vs_review_scores_rating,
                    guesthouse_vs_review_scores_rating,
                    guest_suite_vs_review_scores_rating,
                    cabin_vs_review_scores_rating,
                    tiny_home_vs_review_scores_rating,
                    host_from_ireland_vs_review_scores_rating,
                    host_identity_verified_vs_review_scores_rating,
                    instant_bookable_vs_review_scores_rating,
                    host_respond_within_an_hour_vs_review_scores_rating,
                    host_respond_within_a_few_hours_vs_review_scores_
                    rating,
                    host_respond_within_a_day_vs_review_scores_rating,
                    host_verified_by_email_vs_review_scores_rating,
                    host_verified_by_work_email_vs_review_scores_rating,
                    town_house_vs_review_scores_rating,
                    rental_unit_vs_review_scores_rating,
                    home_vs_review_scores_rating,
                    condo_vs_review_scores_rating,
                    bed_and_breakfast_vs_review_scores_rating,
                    boat_vs_review_scores_rating,
                    serviced_apartment_vs_review_scores_rating,
                    villa_vs_review_scores_rating,
                    castle_vs_review_scores_rating,
                    entire_home_or_apt_vs_review_scores_rating,
                    private_room_vs_review_scores_rating,
                    shared_room_vs_review_scores_rating,
                    shared_bath_vs_review_scores_rating,
                    entertainment_amenities_vs_review_scores_rating,
                    self_care_amenities_vs_review_scores_rating,
                    storage_amenities_vs_review_scores_rating,
                    wifi_vs_review_scores_rating,
                    leisure_amenities_vs_review_scores_rating,
                    kitchen_amenities_vs_review_scores_rating,
                    safety_amenities_vs_review_scores_rating,
                    parking_amenities_vs_review_scores_rating,
                    long_term_stay_vs_review_scores_rating,
                    single_level_home_vs_review_scores_rating,
                    open_24_hours_vs_review_scores_rating,
                    self_check_in_vs_review_scores_rating,
                    dublin_city_vs_review_scores_rating,
                    south_dublin_vs_review_scores_rating,
                    fingal_vs_review_scores_rating,
                    dun_laoghaire_rathdown_vs_review_scores_rating],
           title='Legend for remaining non-dependent features',
           bbox_to_anchor=(1.01, 1.05), loc='upper left', fontsize=10,
           title_fontsize=12, ncol=2)
plt.show()
```

```
    plt.rcParams["figure.constrained_layout.use"] = True
    plt.figure(figsize=(50, 30), dpi=80, tight_layout=True)
    plt.subplot(3, 2, 1)
    # Find correlated features for review_scores_accuracy
    y = review_scores_accuracy
    select_features_from_correlation(X, y, binaryFeatures, "upper center",
    "Binary", "review_scores_accuracy")
    plt.subplot(3, 2, 2)
    # Find correlated features for review_scores_cleanliness
    y = review_scores_cleanliness
    select_features_from_correlation(X, y, binaryFeatures, "upper center",
    "Binary", "review_scores_cleanliness")
    plt.subplot(3, 2, 3)
    # Find correlated features for review_scores_checkin
    y = review_scores_checkin
    select_features_from_correlation(X, y, binaryFeatures, "lower right",
    "Binary", "review_scores_checkin")
    plt.subplot(3, 2, 4)
    # Find correlated features for review_scores_communication
    y = review_scores_communication
    select_features_from_correlation(X, y, binaryFeatures, "upper center",
    "Binary", "review_scores_communication")
    plt.subplot(3, 2, 5)
    # Find correlated features for review_scores_location
    y = review_scores_location
    select_features_from_correlation(X, y, binaryFeatures, "upper center",
    "Binary", "review_scores_location")
    plt.subplot(3, 2, 6)
    # Find correlated features for review_scores_value
    y = review_scores_value
    select_features_from_correlation(X, y, binaryFeatures, "upper center",
    "Binary", "review_scores_value")
    plt.show()


# Show plots with chosen top 10 continuous features and remaining 44
non-dependent features
def showContinuousFeatures(updatedListingsCSV):
    # Use Min-Max scaling to normalise the data to be between 0 and 1.
    dataframe = pd.read_csv(updatedListingsCSV)
    scaler = MinMaxScaler()

    # Potential continuous features:
    host_since = scaler.fit_transform(dataframe.iloc[:,
    1].values.reshape(-1, 1))
    host_response_rate = scaler.fit_transform(dataframe.iloc[:,
    3].values.reshape(-1, 1))
    host_acceptance_rate = scaler.fit_transform(dataframe.iloc[:,
    4].values.reshape(-1, 1))
    host_listings_count = scaler.fit_transform(dataframe.iloc[:,
    6].values.reshape(-1, 1))
    host_total_listings_count = scaler.fit_transform(dataframe.iloc[:,
    7].values.reshape(-1, 1))
    latitude = scaler.fit_transform(dataframe.iloc[:,
    10].values.reshape(-1, 1))
```

```
longitude = scaler.fit_transform(dataframe.iloc[:,
11].values.reshape(-1, 1))
accommodates = scaler.fit_transform(dataframe.iloc[:,
12].values.reshape(-1, 1))
bedrooms = scaler.fit_transform(dataframe.iloc[:,
13].values.reshape(-1, 1))
beds = scaler.fit_transform(dataframe.iloc[:, 14].values.reshape(-1,
1))
price = scaler.fit_transform(dataframe.iloc[:, 15].values.reshape(-1,
1))
minimum_nights = scaler.fit_transform(dataframe.iloc[:,
16].values.reshape(-1, 1))
maximum_nights = scaler.fit_transform(dataframe.iloc[:,
17].values.reshape(-1, 1))
minimum_minimum_nights = scaler.fit_transform(dataframe.iloc[:,
18].values.reshape(-1, 1))
maximum_minimum_nights = scaler.fit_transform(dataframe.iloc[:,
19].values.reshape(-1, 1))
minimum_maximum_nights = scaler.fit_transform(dataframe.iloc[:,
20].values.reshape(-1, 1))
maximum_maximum_nights = scaler.fit_transform(dataframe.iloc[:,
21].values.reshape(-1, 1))
minimum_nights_avg_ntm = scaler.fit_transform(dataframe.iloc[:,
22].values.reshape(-1, 1))
maximum_nights_avg_ntm = scaler.fit_transform(dataframe.iloc[:,
23].values.reshape(-1, 1))
availability_30 = scaler.fit_transform(dataframe.iloc[:,
25].values.reshape(-1, 1))
availability_60 = scaler.fit_transform(dataframe.iloc[:,
26].values.reshape(-1, 1))
availability_90 = scaler.fit_transform(dataframe.iloc[:,
27].values.reshape(-1, 1))
availability_365 = scaler.fit_transform(dataframe.iloc[:,
28].values.reshape(-1, 1))
number_of_reviews = scaler.fit_transform(dataframe.iloc[:,
29].values.reshape(-1, 1))
number_of_reviews_ltm = scaler.fit_transform(dataframe.iloc[:,
30].values.reshape(-1, 1))
number_of_reviews_l30d = scaler.fit_transform(dataframe.iloc[:,
31].values.reshape(-1, 1))
first_review = scaler.fit_transform(dataframe.iloc[:,
32].values.reshape(-1, 1))
last_review = scaler.fit_transform(dataframe.iloc[:,
33].values.reshape(-1, 1))
calculated_host_listings_count =
scaler.fit_transform(dataframe.iloc[:, 42].values.reshape(-1, 1))
calculated_host_listings_count_entire_homes =
scaler.fit_transform(dataframe.iloc[:, 43].values.reshape(-1, 1))
calculated_host_listings_count_private_rooms =
scaler.fit_transform(dataframe.iloc[:, 44].values.reshape(-1, 1))
calculated_host_listings_count_shared_rooms =
scaler.fit_transform(dataframe.iloc[:, 45].values.reshape(-1, 1))
reviews_per_month = scaler.fit_transform(dataframe.iloc[:,
46].values.reshape(-1, 1))
```

```
number_of_bathrooms = scaler.fit_transform(dataframe.iloc[:,
72].values.reshape(-1, 1))
city_center = dataframe.iloc[:, 90]
clean_comfortable = dataframe.iloc[:, 91]
definitely_recommend = dataframe.iloc[:, 92]
definitely_stay = dataframe.iloc[:, 93]
gave_us = dataframe.iloc[:, 94]
great_host = dataframe.iloc[:, 95]
great_location = dataframe.iloc[:, 96]
great_place = dataframe.iloc[:, 97]
great_stay = dataframe.iloc[:, 98]
highly_recommend = dataframe.iloc[:, 99]
location_great = dataframe.iloc[:, 100]
minute_walk = dataframe.iloc[:, 101]
place_great = dataframe.iloc[:, 102]
place_stay = dataframe.iloc[:, 103]
recommend_place = dataframe.iloc[:, 104]
temple_bar = dataframe.iloc[:, 105]
walking_distance = dataframe.iloc[:, 106]
would_definitely = dataframe.iloc[:, 107]
would_highly = dataframe.iloc[:, 108]
would_recommend = dataframe.iloc[:, 109]

# Use abbreviated words for annotating top 10 features for feature
importance plot
continuousFeatures = dict()
continuousFeatures[0] = "h_s"
continuousFeatures[1] = "h_r_r"
continuousFeatures[2] = "h_a_r"
continuousFeatures[3] = "h_l_c"
continuousFeatures[4] = "h_t_l_c"
continuousFeatures[5] = "lat"
continuousFeatures[6] = "long"
continuousFeatures[7] = "a"
continuousFeatures[8] = "bedr"
continuousFeatures[9] = "bed"
continuousFeatures[10] = "p"
continuousFeatures[11] = "min_n"
continuousFeatures[12] = "max_n"
continuousFeatures[13] = "min_min_n"
continuousFeatures[14] = "max_min_n"
continuousFeatures[15] = "min_max_n"
continuousFeatures[16] = "max_max_n"
continuousFeatures[17] = "max_n_a_n"
continuousFeatures[18] = "min_n_a_n"
continuousFeatures[19] = "a_30"
continuousFeatures[20] = "a_60"
continuousFeatures[21] = "a_90"
continuousFeatures[22] = "a_365"
continuousFeatures[23] = "n_o_r"
continuousFeatures[24] = "n_o_r_lt"
continuousFeatures[25] = "n_o_r_l30"
continuousFeatures[26] = "f_r"
continuousFeatures[27] = "l_r"
continuousFeatures[28] = "c_h_l_c"
```

```
continuousFeatures[29] = "c_h_l_c_e_h"
continuousFeatures[30] = "c_h_l_c_p_r"
continuousFeatures[31] = "c_h_l_c_s_r"
continuousFeatures[32] = "r_p_m"
continuousFeatures[33] = "n_o_b"
continuousFeatures[34] = "city_c"
continuousFeatures[35] = "clean_c"
continuousFeatures[36] = "d_rec"
continuousFeatures[37] = "d_stay"
continuousFeatures[38] = "g_u"
continuousFeatures[39] = "g_h"
continuousFeatures[40] = "g_l"
continuousFeatures[41] = "g_p"
continuousFeatures[42] = "g_s"
continuousFeatures[43] = "h_r"
continuousFeatures[44] = "l_g"
continuousFeatures[45] = "m_w"
continuousFeatures[46] = "p_g"
continuousFeatures[47] = "p_s"
continuousFeatures[48] = "r_p"
continuousFeatures[49] = "t_b"
continuousFeatures[50] = "walk_d"
continuousFeatures[51] = "w_def"
continuousFeatures[52] = "w_high"
continuousFeatures[53] = "w_rec"

# Predicting these values
review_scores_rating = dataframe.iloc[:, 34]
review_scores_accuracy = dataframe.iloc[:, 35]
review_scores_cleanliness = dataframe.iloc[:, 36]
review_scores_checkin = dataframe.iloc[:, 37]
review_scores_communication = dataframe.iloc[:, 38]
review_scores_location = dataframe.iloc[:, 39]
review_scores_value = dataframe.iloc[:, 40]

X = np.column_stack((host_since, host_response_rate,
host_acceptance_rate, host_listings_count,
                    host_total_listings_count, latitude, longitude,
                    accommodates, bedrooms, beds, price,
                    minimum_nights, maximum_nights,
                    minimum_minimum_nights, maximum_minimum_nights,
                    minimum_maximum_nights, maximum_maximum_nights,
                    minimum_nights_avg_ntm,
                    maximum_nights_avg_ntm, availability_30,
                    availability_60, availability_90,
                    availability_365,
                    number_of_reviews, number_of_reviews_ltm,
                    number_of_reviews_l30d, first_review, last_review,
                    calculated_host_listings_count,
                    calculated_host_listings_count_entire_homes,
                    calculated_host_listings_count_private_rooms,
                    calculated_host_listings_count_shared_rooms,
                    reviews_per_month, number_of_bathrooms,
                    city_center, clean_comfortable,
                    definitely_recommend,
```

```
                    definitely_stay , gave_us , great_host ,
                    great_location , great_place , great_stay ,
                    highly_recommend ,
                    location_great , minute_walk , place_great ,
                    place_stay , recommend_place , temple_bar ,
                    walking_distance , would_definitely , would_highly ,
                    would_recommend ))
y = review_scores_rating

plt.rcParams["figure.constrained_layout.use"] = True
plt.figure(figsize=(50, 30), dpi=80, tight_layout=True)
plt.subplot(2, 2, 1)
# Find correlated features for review_scores_rating
select_features_from_correlation(X, y, continuousFeatures, "upper
left", "Continuous", "review_scores_rating")
# From correlation feature selection, we see that these binary
features are dependent:
# host_since, host_response_rate, host_listings_count,
host_total_listings_count, longitude, number_of_reviews,
# last_review, calculated_host_listings_count,
calculated_host_listings_count_private_rooms,
# calculated_host_listings_count_shared_rooms

# 1. Plot continuous features that have a dependence:
plt.subplot(2, 2, 2)
host_since_vs_review_scores_rating = plt.scatter(host_since,
review_scores_value, marker="+", label="host_since")

host_response_rate_vs_review_scores_rating =
plt.scatter(host_response_rate, review_scores_value, marker="+",
   label="host_response_rate")

host_listings_count_vs_review_scores_rating =
plt.scatter(host_listings_count, review_scores_value, marker="+",
    label="host_listings_count")

host_total_listings_count_vs_review_scores_rating =
plt.scatter(host_total_listings_count, review_scores_value,
          marker="+", label="host_total_listings_count")

longitude_vs_review_scores_rating = plt.scatter(longitude,
review_scores_value, marker="+", label="longitude")

number_of_reviews_vs_review_scores_rating =
plt.scatter(number_of_reviews, review_scores_value, marker="+",
  label="number_of_reviews")

last_review_vs_review_scores_rating = plt.scatter(last_review,
review_scores_value, marker="+", label="last_review")

calculated_host_listings_count_vs_review_scores_rating =
plt.scatter(calculated_host_listings_count,
               review_scores_value, marker="+",
               label="calculated_host_listings_count")
```

```
calculated_host_listings_count_private_rooms_vs_review_scores_rating =
plt.scatter(
    calculated_host_listings_count_private_rooms, review_scores_value,
    marker="+",
    label="calculated_host_listings_count_private_rooms")

calculated_host_listings_count_shared_rooms_vs_review_scores_rating =
plt.scatter(
    calculated_host_listings_count_shared_rooms, review_scores_value,
    marker="+",
    label="calculated_host_listings_count_shared_rooms")

plt.ylabel("Review Rating", fontsize=13)
plt.xlabel("Continuous Features", fontsize=13)
plt.title("Dependent continuous features", fontsize=16)
plt.legend(handles=[
    host_since_vs_review_scores_rating,
    host_response_rate_vs_review_scores_rating,
    host_listings_count_vs_review_scores_rating,
    host_total_listings_count_vs_review_scores_rating,
    longitude_vs_review_scores_rating,
    number_of_reviews_vs_review_scores_rating,
    last_review_vs_review_scores_rating,
    calculated_host_listings_count_vs_review_scores_rating,
    calculated_host_listings_count_private_rooms_vs_review_scores_rating,
    calculated_host_listings_count_shared_rooms_vs_review_scores_rating],
    title='Legend for 10 dependent features', bbox_to_anchor=(1.01,
    1), loc='upper left', fontsize=12,
    title_fontsize=12, ncol=1)

# 2 Plot remaining features with no/weak dependence:
plt.subplot(2, 1, 2)

city_center_vs_review_scores_rating = plt.scatter(city_center,
review_scores_value, marker="+", label="city_center")

clean_comfortable_vs_review_scores_rating =
plt.scatter(clean_comfortable, review_scores_value, marker="+",
  label="clean_comfortable")

definitely_recommend_vs_review_scores_rating =
plt.scatter(definitely_recommend, review_scores_value, marker="+",
     label="definitely_recommend")

definitely_stay_vs_review_scores_rating = plt.scatter(definitely_stay,
review_scores_value, marker="+",
label="definitely_stay")

gave_us_vs_review_scores_rating = plt.scatter(gave_us,
review_scores_value, marker="+", label="gave_us")

great_host_vs_review_scores_rating = plt.scatter(great_host,
review_scores_value, marker="+", label="great_host")
```

```
    great_location_vs_review_scores_rating = plt.scatter(great_location,
    review_scores_value, marker="+",
   label="great_location")

    great_place_vs_review_scores_rating = plt.scatter(great_place,
    review_scores_value, marker="+", label="great_place")

    great_stay_vs_review_scores_rating = plt.scatter(great_stay,
    review_scores_value, marker="+", label="great_stay")

    highly_recommend_vs_review_scores_rating =
    plt.scatter(highly_recommend, review_scores_value, marker="+",
     label="highly_recommend")

    location_great_vs_review_scores_rating = plt.scatter(location_great,
    review_scores_value, marker="+",
   label="location_great")

    minute_walk_vs_review_scores_rating = plt.scatter(minute_walk,
    review_scores_value, marker="+", label="minute_walk")

    place_great_vs_review_scores_rating = plt.scatter(place_great,
    review_scores_value, marker="+", label="place_great")

    place_stay_vs_review_scores_rating = plt.scatter(place_stay,
    review_scores_value, marker="+", label="place_stay")

    recommend_place_vs_review_scores_rating = plt.scatter(recommend_place,
    review_scores_value, marker="+",
    label="recommend_place")

    temple_bar_vs_review_scores_rating = plt.scatter(temple_bar,
    review_scores_value, marker="+", label="temple_bar")

    walking_distance_vs_review_scores_rating =
    plt.scatter(walking_distance, review_scores_value, marker="+",
     label="walking_distance")

    would_definitely_vs_review_scores_rating =
    plt.scatter(would_definitely, review_scores_value, marker="+",
     label="would_definitely")

    would_highly_vs_review_scores_rating = plt.scatter(would_highly,
    review_scores_value, marker="+",
label="would_highly")

    would_recommend_vs_review_scores_rating = plt.scatter(would_recommend,
    review_scores_value, marker="+",
    label="would_recommend")

    latitude_vs_review_scores_rating = plt.scatter(latitude,
    review_scores_value, marker="+", label="latitude")

    accommodates_vs_review_scores_rating = plt.scatter(accommodates,
    review_scores_value, marker="+",
```

```
label="accommodates")

    bedrooms_vs_review_scores_rating = plt.scatter(bedrooms,
    review_scores_value, marker="+", label="bedrooms")

    beds_vs_review_scores_rating = plt.scatter(beds, review_scores_value,
    marker="+", label="beds")

    availability_30_vs_review_scores_rating = plt.scatter(availability_30,
    review_scores_value, marker="+",
    label="availability_30")

    number_of_reviews_l30d_vs_review_scores_rating =
    plt.scatter(number_of_reviews_l30d, review_scores_value,
           marker="+", label="number_of_reviews_l30d")

    number_of_bathrooms_vs_review_scores_rating =
    plt.scatter(number_of_bathrooms, review_scores_value, marker="+",
        label="number_of_bathrooms")

    host_acceptance_rate_vs_review_scores_rating =
    plt.scatter(host_acceptance_rate, review_scores_value, marker="+",
         label="host_acceptance_rate")

    price_vs_review_scores_rating = plt.scatter(price,
    review_scores_value, marker="+", label="price")

    minimum_nights_vs_review_scores_rating = plt.scatter(minimum_nights,
    review_scores_value, marker="+",
  label="minimum_nights")

    maximum_nights_vs_review_scores_rating = plt.scatter(maximum_nights,
    review_scores_value, marker="+",
  label="maximum_nights")

    minimum_minimum_nights_vs_review_scores_rating =
    plt.scatter(minimum_minimum_nights, review_scores_value,
           marker="+", label="minimum_minimum_nights")

    maximum_minimum_nights_vs_review_scores_rating =
    plt.scatter(maximum_minimum_nights, review_scores_value,
           marker="+", label="maximum_minimum_nights")

    minimum_maximum_nights_vs_review_scores_rating =
    plt.scatter(minimum_maximum_nights, review_scores_value,
           marker="+", label="minimum_maximum_nights")

    maximum_maximum_nights_vs_review_scores_rating =
    plt.scatter(maximum_maximum_nights, review_scores_value,
           marker="+", label="maximum_maximum_nights")

    minimum_nights_avg_ntm_vs_review_scores_rating =
    plt.scatter(minimum_nights_avg_ntm, review_scores_value,
           marker="+", label="minimum_nights_avg_ntm")
```

```
    maximum_nights_avg_ntm_vs_review_scores_rating =
    plt.scatter(maximum_nights_avg_ntm, review_scores_value,
           marker="+", label="maximum_nights_avg_ntm")

    availability_60_vs_review_scores_rating = plt.scatter(availability_60,
    review_scores_value, marker="+",
    label="availability_60")

    availability_90_vs_review_scores_rating = plt.scatter(availability_90,
    review_scores_value, marker="+",
    label="availability_90")

    availability_365_vs_review_scores_rating =
    plt.scatter(availability_365, review_scores_value, marker="+",
     label="availability_365")

    number_of_reviews_ltm_vs_review_scores_rating =
    plt.scatter(number_of_reviews_ltm, review_scores_value, marker="+",
          label="number_of_reviews_ltm")

    first_review_vs_review_scores_rating = plt.scatter(first_review,
    review_scores_value, marker="+",
label="first_review")

    calculated_host_listings_count_entire_homes_vs_review_scores_rating =
    plt.scatter(
        calculated_host_listings_count_entire_homes, review_scores_value,
        marker="+",
        label="calculated_host_listings_count_entire_homes")

    reviews_per_month_vs_review_scores_rating =
    plt.scatter(reviews_per_month, review_scores_value, marker="+",
      label="reviews_per_month")

    plt.ylabel("Review Rating", fontsize=13)
    plt.xlabel("Continuous Features", fontsize=13)
    plt.title("Remaining non-dependent continuous features", fontsize=16)
    plt.legend(handles=[city_center_vs_review_scores_rating,
                        clean_comfortable_vs_review_scores_rating,
                        definitely_recommend_vs_review_scores_rating,
                        definitely_stay_vs_review_scores_rating,
                        gave_us_vs_review_scores_rating,
                        great_host_vs_review_scores_rating,
                        great_location_vs_review_scores_rating,
                        great_place_vs_review_scores_rating,
                        great_stay_vs_review_scores_rating,
                        highly_recommend_vs_review_scores_rating,
                        location_great_vs_review_scores_rating,
                        minute_walk_vs_review_scores_rating,
                        place_great_vs_review_scores_rating,
                        place_stay_vs_review_scores_rating,
                        recommend_place_vs_review_scores_rating,
                        temple_bar_vs_review_scores_rating,
                        walking_distance_vs_review_scores_rating,
                        would_definitely_vs_review_scores_rating,
```

```
                        would_highly_vs_review_scores_rating,
                        would_recommend_vs_review_scores_rating,
                        latitude_vs_review_scores_rating,
                        accommodates_vs_review_scores_rating,
                        bedrooms_vs_review_scores_rating,
                        beds_vs_review_scores_rating,
                        availability_30_vs_review_scores_rating,
                        number_of_reviews_l30d_vs_review_scores_rating,
                        number_of_bathrooms_vs_review_scores_rating,
                        host_since_vs_review_scores_rating,
                        host_response_rate_vs_review_scores_rating,
                        host_listings_count_vs_review_scores_rating,
                        host_acceptance_rate_vs_review_scores_rating,
                        host_total_listings_count_vs_review_scores_rating,
                        price_vs_review_scores_rating,
                        minimum_nights_vs_review_scores_rating,
                        maximum_nights_vs_review_scores_rating,
                        minimum_minimum_nights_vs_review_scores_rating,
                        maximum_minimum_nights_vs_review_scores_rating,
                        minimum_maximum_nights_vs_review_scores_rating,
                        maximum_maximum_nights_vs_review_scores_rating,
                        minimum_nights_avg_ntm_vs_review_scores_rating,
                        maximum_nights_avg_ntm_vs_review_scores_rating,
                        availability_60_vs_review_scores_rating,
                        availability_90_vs_review_scores_rating,
                        availability_365_vs_review_scores_rating,
                        number_of_reviews_ltm_vs_review_scores_rating,
                        first_review_vs_review_scores_rating,
                        last_review_vs_review_scores_rating,
                        calculated_host_listings_count_vs_
                        review_scores_rating,
                        calculated_host_listings_count_entire_homes_vs_
                        review_scores_rating,
                        calculated_host_listings_count_private_rooms_vs_
                        review_scores_rating,
                        calculated_host_listings_count_shared_rooms_vs_
                        review_scores_rating,
                        calculated_host_listings_count_shared_rooms_vs_
                        review_scores_rating,
                        reviews_per_month_vs_review_scores_rating],
                        title='Legend for non-dependent features',
            bbox_to_anchor=(1.01, 1.05), loc='upper left',
            fontsize=8.5, title_fontsize=12, ncol=2)
plt.show()
plt.rcParams["figure.constrained_layout.use"] = True
plt.figure(figsize=(50, 30), dpi=80, tight_layout=True)
plt.subplot(3, 2, 1)
# Find correlated features for review_scores_accuracy
y = review_scores_accuracy
select_features_from_correlation(X, y, continuousFeatures, "upper
right", "Continuous", "review_scores_accuracy")
plt.subplot(3, 2, 2)
# Find correlated features for review_scores_cleanliness
y = review_scores_cleanliness
```

```
    select_features_from_correlation(X, y, continuousFeatures, "upper
    right", "Continuous",
          "review_scores_cleanliness")
    plt.subplot(3, 2, 3)
    # Find correlated features for review_scores_checkin
    y = review_scores_checkin
    select_features_from_correlation(X, y, continuousFeatures, "upper
    right", "Continuous", "review_scores_checkin")
    plt.subplot(3, 2, 4)
    # Find correlated features for review_scores_communication
    y = review_scores_communication
    select_features_from_correlation(X, y, continuousFeatures, "upper
    right", "Continuous",
          "review_scores_communication")
    plt.subplot(3, 2, 5)
    # Find correlated features for review_scores_location
    y = review_scores_location
    select_features_from_correlation(X, y, continuousFeatures, "upper
    right", "Continuous", "review_scores_location")
    plt.subplot(3, 2, 6)
    # Find correlated features for review_scores_value
    y = review_scores_value
    select_features_from_correlation(X, y, continuousFeatures, "upper
    right", "Continuous", "review_scores_value")
    plt.show()
```

1.4 gaussian_kernel.py (inspired from lectures)

```
import numpy as np


def gaussian_kernel1(distances):
    weights = np.exp((1 * (distances ** 2)))
    return weights / np.sum(weights)


def gaussian_kernel5(distances):
    weights = np.exp((5 * (distances ** 2)))
    return weights / np.sum(weights)


def gaussian_kernel10(distances):
    weights = np.exp((10 * (distances ** 2)))
    return weights / np.sum(weights)


def gaussian_kernel30(distances):
    weights = np.exp((-30 * (distances ** 2)))
    return weights / np.sum(weights)


def gaussian_kernel50(distances):
    weights = np.exp((-50 * (distances ** 2)))
```

```
    return weights / np.sum(weights)


def gaussian_kernel100(distances):
    weights = np.exp((-100 * (distances ** 2)))
    return weights / np.sum(weights)


def gaussian_kernel150(distances):
    weights = np.exp((-150 * (distances ** 2)))
    return weights / np.sum(weights)
```

1.5 kNN.py

```
import matplotlib
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
from gaussian_kernel import *

# Comment the line below if you are not using an M1 (ARM-based) machine
matplotlib.use('TkAgg')



# Step 1: Selecting k for kNN regressor

# Selecting range of k values for the kNN model
# Conclusion from this function: Best range is between 1 and 100
def select_k_range(X, y):
    plt.rcParams["figure.constrained_layout.use"] = True
    mean_error = []
    std_error = []
    k_range = [1, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250]
    for k in k_range:
        model = KNeighborsRegressor(n_neighbors=k)
        scores = cross_val_score(model, X, y, cv=5,
        scoring="neg_mean_squared_error")
        mean_error.append(abs(np.array(scores).mean()))
        std_error.append(np.array(scores).std())
    plt.errorbar(k_range, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("k")
    plt.ylabel("Mean squared error")
    plt.title("k vs Mean squared error (Selecting range of k)")
    plt.show()


# 5-fold Cross validation on range of k values selected previously (1 to
100)
# Conclusion from this function: Best value for k is 50
def choose_k_using_CV(X, Y):
    mean_error = []
    std_error = []
    k_range = [1, 25, 50, 75, 100]
```

```
    for k in k_range:
        model = KNeighborsRegressor(n_neighbors=k)
        scores = cross_val_score(model, X, Y, cv=5,
        scoring="neg_mean_squared_error")
        mean_error.append(abs(np.array(scores).mean()))
        std_error.append(np.array(scores).std())
    plt.errorbar(k_range, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("k");
    plt.ylabel("Mean squared error")
    plt.title("k vs Mean squared error (performing 5-fold
    cross-validation)")
    plt.show()


# Step 2: Selecting gamma (for weights)

# Selecting range of gamma values for the kNN model
# Conclusion from this function: The best range to use CV for gamma is
less than 10.
def select_kNN_gamma_range_for_CV(X, Y):
    mean_error = []
    std_error = []
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel10)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel30)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel100)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel150)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    plt.errorbar([10, 30, 100, 150], mean_error, yerr=std_error,
    linewidth=3)
    plt.xlabel("Gamma")
    plt.ylabel("Mean squared error")
    plt.title("Gamma vs Mean squared error (Selecting range of gamma)")
    plt.show()


# 5-fold Cross validation on range of gamma values selected previously
(less than 10)
# Conclusion from this function: The best value for gamma is 1.
def choose_kNN_gamma_using_CV(X, Y):
    mean_error = []
```

```
    std_error = []
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel1)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel5)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    model = KNeighborsRegressor(n_neighbors=50, weights=gaussian_kernel10)
    scores = cross_val_score(model, X, Y, cv=5,
    scoring="neg_mean_squared_error")
    mean_error.append(abs(np.array(scores).mean()))
    std_error.append(np.array(scores).std())
    plt.errorbar([1, 5, 10], mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("Gamma")
    plt.ylabel("Mean squared error")
    plt.title("Gamma vs Mean squared error (Performing 5-fold
    cross-validation)")
    plt.show()


# kNN model with chosen hyperparameters k=50 & gamma=1 selected via 5-fold
cross-validation
def kNN(x_train, y_train):
    model_knn = KNeighborsRegressor(n_neighbors=50,
    weights=gaussian_kernel1).fit(x_train, y_train)
    return model_knn
```

1.6 lasso.py

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score

# Comment the line below if you are not using an M1 (ARM-based) machine
from sklearn.preprocessing import PolynomialFeatures

matplotlib.use('TkAgg')



# Step 1: Selecting C for lasso model

# Selecting range of C values for the lasso model
# Conclusion from this function: Best range is between 100-300
def select_c_range(X, y):
    plt.rcParams["figure.constrained_layout.use"] = True
    mean_error = []
    std_error = []
```

```
    c_range = [1, 100, 200, 300, 400, 500, 600, 700, 800]
    for c in c_range:
        model = Lasso(alpha=1 / (2 * c))
        scores = cross_val_score(model, X, y, cv=5,
        scoring="neg_mean_squared_error")
        mean_error.append(abs(np.array(scores).mean()))
        std_error.append(np.array(scores).std())
    plt.errorbar(c_range, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("C")
    plt.ylabel("Mean squared error")
    plt.title("C vs Mean squared error (Selecting range for k)")
    plt.show()


# 5-fold Cross validation on range of k values selected previously
(100-300)
# Conclusion from this function: Best value for C is 300
def choose_c_using_CV(X, Y):
    mean_error = []
    std_error = []
    c_range = [100, 125, 150, 175, 200, 225, 250, 275, 300]
    for c in c_range:
        model = Lasso(alpha=1 / (2 * c))
        scores = cross_val_score(model, X, Y, cv=5,
        scoring="neg_mean_squared_error")
        mean_error.append(abs(np.array(scores).mean()))
        std_error.append(np.array(scores).std())
    plt.errorbar(c_range, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("C")
    plt.ylabel("Mean squared error")
    plt.title("C vs Mean squared error (performing 5-fold
    cross-validation)")
    plt.show()


# Step 2: Selecting q for polynomial features

# 5-fold Cross validation on range of q values for polynomial features
# Conclusion from this function: Best value for q is
def choose_q_using_CV(X, Y):
    mean_error = []
    std_error = []
    q_range = [1, 2, 3, 4]
    for q in q_range:
        Xpolynomial = PolynomialFeatures(q).fit_transform(X)
        model = Lasso(alpha=1 / (2 * 300))
        scores = cross_val_score(model, Xpolynomial, Y, cv=5,
        scoring="neg_mean_squared_error")
        mean_error.append(abs(np.array(scores).mean()))
        std_error.append(np.array(scores).std())
    plt.errorbar(q_range, mean_error, yerr=std_error, linewidth=3)
    plt.xlabel("q")
    plt.ylabel("Mean squared error")
    plt.title("q vs Mean squared error (performing 5-fold
    cross-validation)")
```

```python
    plt.show()


# Lasso regression model chosen hyperparameters C=300 & q=1 selected via 5-fold cross-validation
def lassoRegression(x_train, y_train):
    model = Lasso(alpha=(1 / (2 * 300))).fit(x_train, y_train)
    return model
```