CSU44061 Final Assignment: Feasibility of prediction for listings of Airbnb review scores

Name:Tianze Zhang
Student Number: 19334401

Contents

1 Introduction

In this project, the feasibility of prediction for 7 different Airbnb review scores is investigated. In order to draw to a conclusion, some necessary steps are taken, this includes exploring the linearity and distribution of the dependant variables, standardizing and normalizing the dataset, encode textual features using label encoding and one hot encoding, filling missing values, feature selection, model training and hyperparameter tuning, evaluating the model by comparing to the baseline and evaluate their metrics (MSE and R2), and much more. A step-by-step approach of this project will be presented and discussed in the upcoming sections.

2 Feature Engineering

2.1 Data exploration and pre-processing

The dataset is achieved from the Airbnb web site at http://insideairbnb.com/get-the-data/, under" Dublin, Leinster, Ireland". The two datasets listings.csv and reviews.csv are downloaded and imported as Pandas DataFrames.

The listings data has 75 feature columns and 7567 samples, and the reviews data has 6 feature columns and 243184 samples. The relation between listings and reviews is one to many, this means for each listing there can be multiple reviews. Hence the two datasets are merged into one, by matching each review to its associated listing, which is done by matching the reviews listings_id to the corresponding ids in the listings file.

After merging the dataset, 13 of features considered no use for this project, including host name and ID number, URLs to the listings or photographs, and data scraping info. Those features are dropped directly. This reduced the dataset to 65 feature columns.

The merged dataset now consisted of 20 integer features, 14 float features and also 30 object features, since object features is not applicable to use directly as inputs when training the model, some conversion to integer/matrix is needed. Firstly, features that are only have two categories, are converted into 0 and 1. All features that are in the form of dates are converted to float. There are also feature such as host_response_rate and host_acceptance_rate that are numeric but with a symbol at the end, those are converted to float by removing the '%' symbol and dividing by 100. Similar operations are done to the price feature, which consist of values combined with comma and '$'. Those symbols are simply removed and then the price is converted to float. Finally, the host_response_time is ranked based on 4 categories: within an hour, within a few hours, within a day and a few days or more. By common sense the shorter the response time the better the score, therefore, within an hour is associated to the highest integer etc. Now only object feature that are textual description are left which cannot be easily categorized. There are also some NaN and empty values in the dataset. In the next two subsections, methods dealing with those two parts will be described.

2.1.1 Text analysis using label Encoding and One Hot Encoding

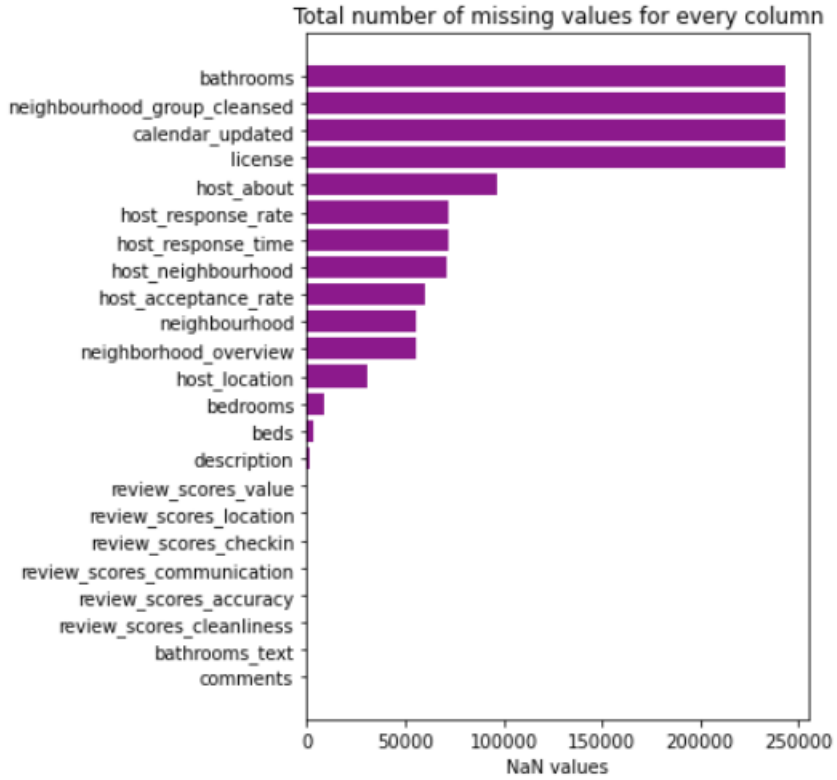
Both label encoding and one hot encoding are designed to convert columns to numerical columns. However, label encoding turns a bag of tokens into a single value, while one hot encoding turns to a feature vector based on the unique catogories. One hot encoding usually creates less bias than label encoding since the machine learning model is likely to treat a single value as ranking based on its value. However, due to memory constraints, it is not

possible to encode every text feature using one hot encoding. Therefore, only one feature, the comments feature is one hot encoded.

Since the comments feature contains non-English characters, all those non-English characters are translated using the googletrans package in python. All comments excluding emojis and empty values are then converted into feature vectors using one hot encoding and added to our exciting data frame. The original comments feature which is textual are dropped.

2.1.2 Fill out missing values

The number of Nan values in the dataset is discovered as picture 2.1. As in the image, the features bathrooms, calendar_updated, license and neighbourhood_group_cleansed are completely empty. Those columns are dropped directly. For host_response_rate, host_response_time the assumption is made that when the value is empty it means the host never response, which is filled with 0. Similar logic is applied for host_acceptance_rate, bedrooms and beds. The others are binary categorical features are filled with a vector of 0.

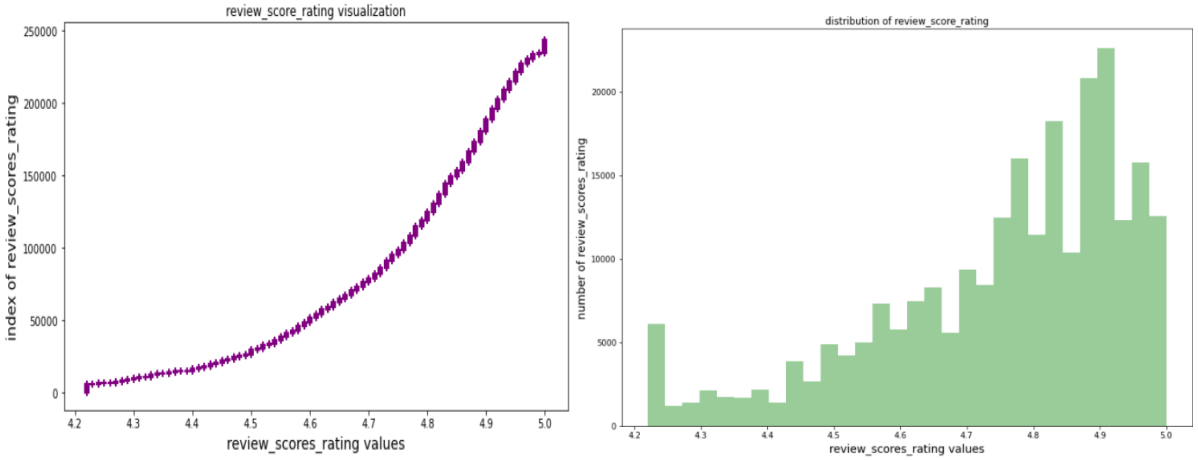


Picture 2.1: the image shows the number of missing values per feature, where the x axis is the number of Nan values and y axis the features.

2.2 Target feature visualization and model selection

The project is required to investigate the feasibility of prediction of review scores for each listing (each individual property). This leads to 7 different targeted values or dependant variable. Since all the 7 scores consists of float values with two decimals from 4-5, the assumption that they are continuous data are made. Therefore, since the data is continuous, regression is chosen instead of classification. The distribution of each target variable are plotted. In picture 2.2 and 2.3 shows an example of review_score_rating's distribution. As in the figure, it is clear that review_score_rating is linearly distributed, since more properties are associated to higher ratings. The other 6 targeted values also follows a similar distribution.

Therefore, algorithms in the linear regression family lasso regression and ridge regression are chosen for this project.

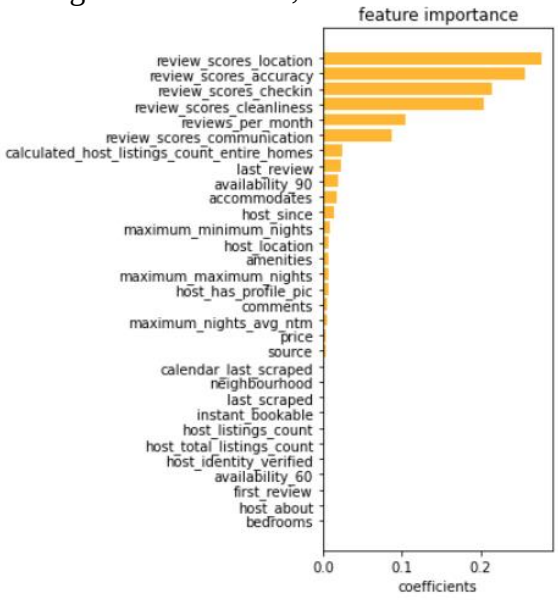


Picture 2.2,2.3: shows the distribution of review score ratings, with the x axis review score ratings and y axis the number of properties associated

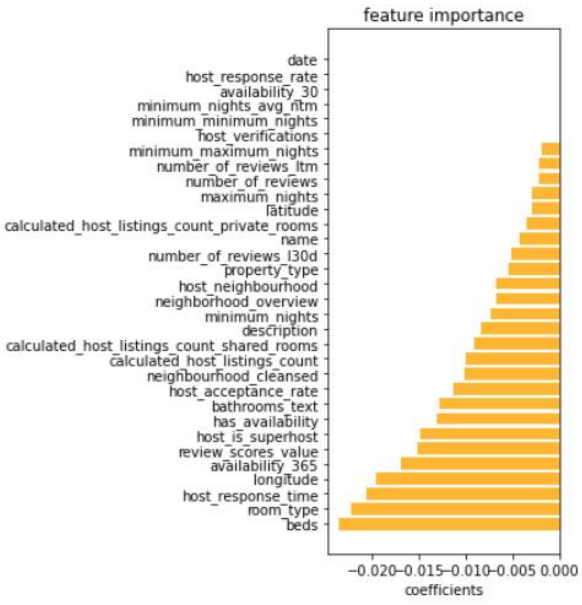
2.3 Feature importance

2.3.1 Lasso Regression

For Lasso, features can be ranked by the coefficient used on the model parameters. The features all 7 scores are plotted against the features coefficient. Picture 2.4 and 2.5 shows an example feature importance using lasso regression with review_scores_ratings as target variable. The parameter C for the model is get using 5-fold cross validation. As in the figure there are many features with a coefficient of 0, those features has no impact on the predictions, and they are dropped. The dominate positive features are review score for location, accuracy, checkin and cleanliness, those create a positively linear effect to the prediction, for example, higher location score indicates a higher rating. Whereas beds, room type, host response time and longitude are the major negative coefficients, which indicates the higher those values, the smaller the rating's score.



Picture 2.4: Feature of Importance lasso. Features with positive importance, where the x axis indicates coefficient and the y axis are the features name.

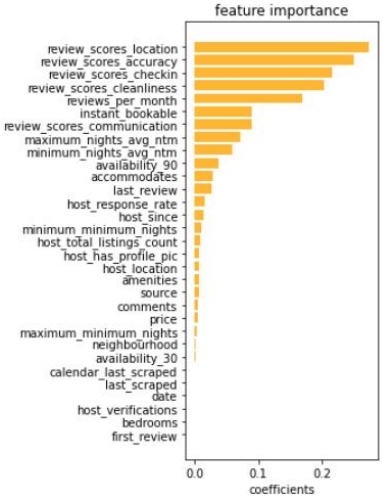


Picture 2.5: negative features importance

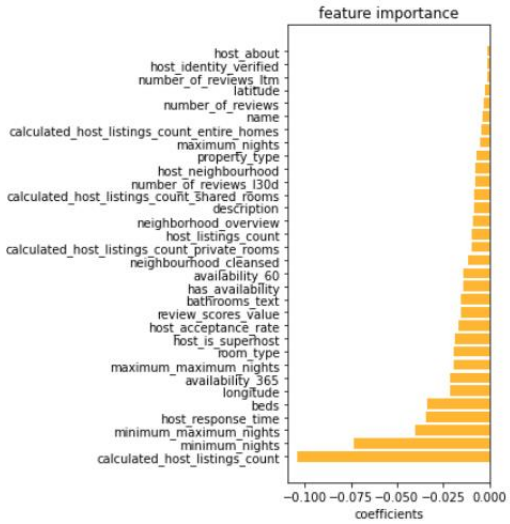
2.3.2 Ridge Regression

The same methodology is applied again but this time using ridge regression. For each of the 7 target values as label, the features are plotted against the coefficient. Picture 2.6 and 2.7 shows an example feature importance using ridge regression with review scores ratings as target variable. As in the figure, the most dominant positively impact features remains the same as lasso regression. However, the dominated negative impacted feature changed to calculated host listings count, minimum and maximum nights allowed. However, there is a common dominated negatively impacted feature for both algorithms, which is the host response time.

From the results from feature engineering using each model, there is an indication of highly rated listings tend to have more reviews, a high location, accuracy, cleanliness score and a short response time from host. When training the model only 12 features: the 6 top positive and 6 top negative features are preserved to prevent overfitting and they are selected based on feature importance from both models.



Picture 2.6: Feature of Importance ridge. Features with positive importance, where the x axis indicates coefficient and the y axis are the features name.



Picture 2.7: negative features importance

3 Machine Learning methodology

The models are trained using all 7 different review scores in the listing as targeted values and undergo the same process of hypermeter tuning. In this section, the section will continually use review_score_rating as an example. Firstly, the data is standardized using a standard scaler. It is then divided into training and test data with an 80/20 train test split. Hypermeter tuning is conducted using 5-fold cross validation on the entire training set and then testing on the test data that has been trained on the entire training data. The performance metrics used for cross validation is Mean squared error (MSE) which will be further explained in the evaluation section. Finally, the mean and standard deviation will be returned and used for further evaluation. The aim is to find the most optimal value for the penalty parameter C which minimizes underfitting and overfitting, so that the most accurate prediction will be produced. A dummy regressor is selected as baseline model which predicts the mean value every time. All models are compared to the baseline.

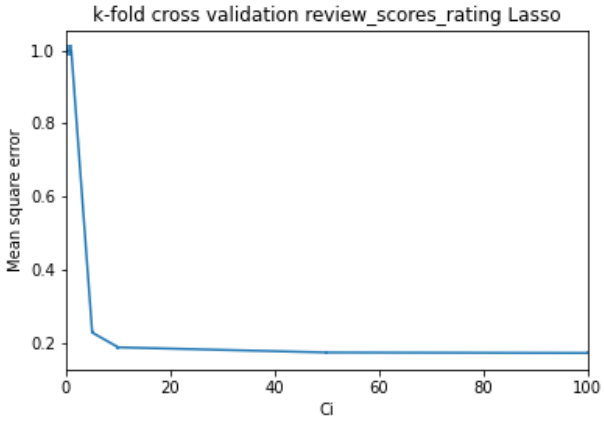
3.1 Lasso Regression Model

Lasso or least absolute shrinkage and selection operation is a form of linear regression method. It uses L1 regularization technique $R(\theta) = 1/C*\sum nj\ |\theta j|$, it changes the cost function and adds a penalty to linear regressor. Lasso regression penalizes less important features assigning it a coefficient to 0, hence eliminate useless features. (G.Serrano, 2021)

The penalty parameter C is needed to be hypermeter tuned, which is required to calculate alpha. Alpha is 1/C which is a constant multiplied by the L1 term. The higher the C value, the smaller the alpha and it gets closer and closer to 0, which also means a smaller L1 and L1 also approaches to 0. Hence really large values of C leads to lasso becoming a linear regression model.

3.2 Lasso Hyperparameter Tuning

Hyperparameter Tuning is conducted using 5-fold cross validation on the entire training set and then testing on the test data that has been trained on the entire training data. The performance metrics used for cross validation is Mean squared error (MSE). Finally, the mean and standard deviation will be returned and used for further evaluation. Picture 3.0 shows the result of Hyperparameter tuning performed over a range of C = 0.1 to C = 100 in steps of multiplication by 10 plotted against the MSE. This is a rule of thumb on how to choose C, which ensure a wider range of C is chosen, hence more accurate result.



Picture 3.0: shows the mean and standard deviation cross validation scores for a wide range of C values

As in figure 3.0, it is clear that when C is really small, the prediction is really inaccurate (High MSE) due to underfitting. However, when C increases, the model performs better which the mean square error decreases. The model is mostly stable and predicts most accurate for C>10, since the MSE doesn't change to much when C goes above 10, therefore a value of C=10 is chosen instead of a higher C value to avoid overfitting.

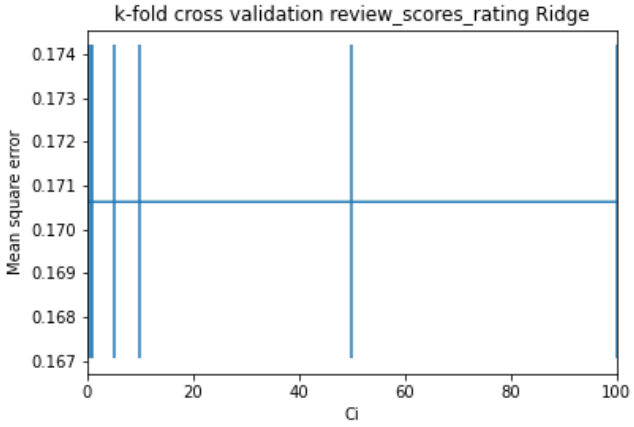
3.3 Ridge Regression Model

In contrast to lasso regression, ridge regression adds a L2 penalty to linear regression's cost function. Ridge regression's cost function is $J(\theta) = 1m\sum mi=1(h\theta(x(i))-y(i))2 + \theta T\theta/C$ where $\theta T\theta$ is L2. L1 regularization penalizes the sum of absolute values of the weights, whereas L2 regularization penalizes the sum of squares of the weights. This means ridge regression has smaller effect when dealing with small C values.

Similarly, ridge's C also needs to be tuned, and it is tuned in the using same methodology as lasso (see Section 2.1).

3.4 Ridge Hyperparameter Tuning

5-Fold cross validation is applied with the same range of C values as lasso. (See Section 2.2).



Picture 3.1: shows the mean and standard deviation cross validation scores for a wide range of C values

As Figure 3.1, It is clear that the C value is harder to choose since the MSE are mostly constant. Since small C values doesn't affect ridge regression as much as lasso, simpler model, hence C = 1, a small value of C is chosen to prevent overfitting.

4 Evaluation

4.1 Mean Squared Error

Mean Squared Error or MSE is if an estimator used to measure the squared difference between the estimated value and the actual value. Hence the equation 1/m∑mi=1(θTx^(i) – y^(i)) ^2. This means the smaller the MSE, the shorter distance between the predicted and actual results, the lesser the error, which means a better model. In this project, all MSE of running different models on different review scores are recorded and presented in the table below:

MSE table of the 7 review scores run by different models:

	rating	accuracy	cleanliness	checkin	communication	location	value
baseline	1.0032	0.9961	0.9913	0.9956	0.9969	1.0004	0.9999
lasso	0.1953	0.2999	0.2397	0.2991	0.5963	0.2227	0.8156
ridge	0.1793	0.2756	0.2203	0.2562	0.4018	0.1939	0.8050

Based on the table, it is clear that both lasso and ridge outperforms the baseline that always predicts the mean. For example, the baseline's MSE are all >0.9, where lasso and ridge's MSE are in the range of 0.17-0.82. Ridge and Lasso performance are mostly similar, but Ridge outperforming Lasso a little bit, with mostly a difference of MSE of 0.01(rating as example 0.1953 vs 0.1793). The MSE is large at communication and value, especially value

with a MSE of 0.817. This is only around 0.1 less than the baseline, which makes communication and value less feasible for prediction than others. Now R^2 will be investigated to further prove on this.

4.2 R Squared

R-squared (R2) is a statistical measure that measures the proportion of the variance for a dependent variable that's explained by an independent variable or features in a regression model. In simple terms, R2 measures how well the data fits the regression model. For example, if the R2 value is 0.5, half of the predictions can be explained by the model, while 50% cannot.

R2 table of the 7 review scores run by different models:

	rating	accuracy	cleanliness	checkin	communication	location	value
baseline	-2.626	-1.4968	-3.782	-2.03	-7.108	-9.807	-6.1487
lasso	0.8054	0.6989	0.7582	0.6995	0.5687	0.7781	0.1843
ridge	0.8212	0.7234	0.7778	0.7427	0.4295	0.8069	0.1950

Based on the table, it is clear that both lasso and ridge outperforms the baseline, hence they explains the feature better than the baseline. Since the baseline is always predicting the mean, it is clear that the data has no correlation with the algorithm itself, then giving a negative value. Similar to the MSE, ridge performance always slightly better than lasso, which means ridge explains more of the predicted data better (rating as example is 0.8212 vs 0.8054). However, communication and value in the case also provides a different result than others, especially value which only gives an R2 of 0.1843 and 0.1950 respectively, this mean more than 81% of the review_scores_value data cannot be explained by the two models, since according to the MSE 0.8156 and 0.8050, the conclusion is drawn that review_score_value not feasible for prediction. Communication on the other hand, only gives an R2 of 0.5687 and 0.4295 respectively, this mean greater or equal to 50% of the data cannot be explained well by lasso and ridge, with MSE 0.5963 and 0.4018, are less than other review score accuracy, making review_score_ communication also not feasible for prediction. Finally, other review score's MSE remains in a range of 0.1–0.3 with a R2 range of 0.69-0.83, which are much more feasible for prediction.

5 Conclusion

Most of the review scores are feasible to predict. All of the review scores performs better than the baseline. However, review_scores_value is not feasible for prediction because of a MSE and R2 of 0.8156, 0.1843 on lasso and 0.8050, 0.1950 on ridge respectively, which makes it performing almost the same as the baseline. review_scores_communication is also possibly not feasible to predict with a MSE and R2 of 0.5963, 0.5687 on lasso and 0.4018,0.4295 on ridge respectively, comparing to other scores all have a MSE of ~0.2 and R2 of ~0.7, which makes review_scores_communication not feasible to predict.

6 References

G.Serrano, L. (2021). *grokking Machine Learning.* NY: Manning Publications.

Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn,Keras & TensorFlow.* CA: O'Reilly Media Inc.

LEITH, D. (2022, 1 4). *Blackboard.* Retrieved from Blackboard Slides: https://tcd.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_77283_1&content_id=_2298101_1&mode=reset

7 Appendix

```
%pip install googletrans==3.1.0a0
%pip install langdetect
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from googletrans import Translator
from langdetect import detect

reviews.head()
data = pd.merge(reviews,listings,how="left",left_on='listing_id', right_on='id')
data.head()

data=data.drop(["listing_id","id_x","reviewer_id","reviewer_name","id_y","scrape_id","listi
ng_url","picture_url","host_id","host_url","host_name","host_thumbnail_url","host_picture
_url"],axis=1)

missing_data = data.isnull().sum(axis=0).reset_index()
missing_data.columns = ['names', 'counts']
missing_data = missing_data.sort_values(by='counts')
missing_data = missing_data[missing_data.counts != 0]

fig, axes = plt.subplots(figsize=(5,7))
axes.barh(np.arange(missing_data.shape[0]), missing_data.counts.values,
alpha=0.9,color='purple')
axes.set_title("Total number of missing values for every column")
axes.set_yticks(np.arange(missing_data.shape[0]))
axes.set_yticklabels(missing_data.names.values, rotation='horizontal')
plt.xlabel('NaN values', fontsize=10)
plt.savefig('missing_val.png', facecolor='w', transparent=False)
plt.show()

data=data.drop(["bathrooms","calendar_updated","license","neighbourhood_group_cleans
ed"],axis=1)
data_type =data.dtypes.reset_index()
data_type.columns = ["Count", "Data Type"]
```

```
data_type.groupby("Data Type").aggregate('count').reset_index()
data_type.loc[data_type['Data Type'] == 'object']
data['host_is_superhost']=data['host_is_superhost'].replace(['f','t'],[0,1])
data['host_has_profile_pic'] = data['host_has_profile_pic'].replace(['f','t'],[0,1])
data['host_identity_verified'] = data['host_identity_verified'].replace(['f','t'],[0,1])
data['has_availability'] = data['has_availability'].replace(['f','t'],[0,1])
data['instant_bookable'] = data['instant_bookable'].replace(['f','t'],[0,1])
data['source'] = data['source'].replace(['city scrape','previous scrape'],[0,1])

data['date'] = pd.to_datetime(data['date'])
data['last_scraped']=pd.to_datetime(data['last_scraped'])
data['host_since'] = pd.to_datetime(data['host_since'])
data['calendar_last_scraped']=pd.to_datetime(data['calendar_last_scraped'])
data['first_review']=pd.to_datetime(data['first_review'])
data['last_review']=pd.to_datetime(data['last_review'])

data['host_response_time'] = data['host_response_time'].map({'within an hour': 4, 'within a
few hours': 3,'within a day':2,'a few days or more':1})
data['host_response_rate'] = data['host_response_rate'].str.rstrip('%').astype('float') / 100.0
data['host_acceptance_rate'] = data['host_acceptance_rate'].str.rstrip('%').astype('float') /
100.0
data['price'] = data['price'].str.replace(',', '').str.replace('$', '').astype(float)
#use LabelEncoder to deal with non numerical data

for feature in data.columns:
    if feature == 'comments':
        continue
    if data[feature].dtype=='object':
        label = LabelEncoder()
        label.fit(list(data[feature].values))
        data[feature] = label.transform(list(data[feature].values))
data.head(1)

def det(x):
    try:
        lang = detect(x)
    except:
        lang = 'Other'
    return lang

    translator=Translator()

    data['comments'] = data['comments'].apply(lambda x: translator.translate(x,
dest='en').text if det(x)!='en' else x)

    data1 = data[:100000]
    label = LabelEncoder()
```

```
    label.fit(list(data1['comments'].values))
    data['comments'][:100000] = label.transform(list(data1['comments'].values))
    data2 = data[100000:200000]
    label = LabelEncoder()
    label.fit(list(data2['comments'].values))
    data['comments'][100000:200000] = label.transform(list(data2['comments'].values))
    data3 = data[200000:243183]
    label = LabelEncoder()
    label.fit(list(data3['comments'].values))
    data['comments'][200000:243183] = label.transform(list(data3['comments'].values))
    data = data.fillna(0)

    label = LabelEncoder()
    label.fit(list(data['comments'].values))
    integer_encoded = label.transform(list(data['comments'].values))

    onehot_encoder = OneHotEncoder(sparse=False)
    integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
    onehot_encoded = onehot_encoder.fit_transform(integer_encoded)

    df=pd.DataFrame(onehot_encoded,columns=['comment_'+str(i) for i in
range(0,len(onehot_encoded[0]))])
    data.to_pickle("cleaned_data.pkl")

    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plot
    from sklearn.linear_model import LassoCV
    from sklearn.linear_model import RidgeCV
    from sklearn.linear_model import Lasso
    from sklearn.linear_model import Ridge
    from sklearn.model_selection import train_test_split
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import classification_report
    from sklearn.metrics import roc_curve
    from sklearn.preprocessing import StandardScaler
    from sklearn.dummy import DummyRegressor
    from sklearn.metrics import mean_squared_error
    from sklearn.metrics import r2_score
    from sklearn.model_selection import KFold
    import seaborn as sns
    import warnings

    df = pd.read_pickle("./cleaned_data.pkl")
    warnings.filterwarnings('ignore')
```

```
   df['review_scores_rating'].loc[df['review_scores_rating']>np.percentile(df['review_scores
_rating'].values, 98)] = np.percentile(df['review_scores_rating'].values, 98)
   df['review_scores_rating'].loc[df['review_scores_rating']<np.percentile(df['review_scores
_rating'].values, 2)] = np.percentile(df['review_scores_rating'], 2)

   plot.figure(figsize=(13,9))
   sns.distplot(df.review_scores_rating.values,kde=False,color='green', bins=30)
   plot.xlabel('review_scores_rating values', fontsize=14)
   plot.ylabel('number of review_scores_rating', fontsize=14)
   plot.title("distribution of review_score_rating")
   plot.savefig('dis_val.png', facecolor='w', transparent=False)
   plot.show()

   plot.figure(figsize=(10,6))
   x, y = pd.Series(np.sort(df['review_scores_rating'].values), name="review_scores_rating"),
pd.Series(range(len(df['review_scores_rating'])), name="index of review_scores_rating
values")
   sns.regplot(x=x, y=y, marker="+",fit_reg=False,color='purple')
   plot.title("review_score_rating visualization")
   plot.xlabel('review_scores_rating values', fontsize=14)
   plot.ylabel('index of review_scores_rating', fontsize=14)
   plot.savefig('vis_val.png', facecolor='w', transparent=False)
   plot.show()

   #fit datetime to standard scaler
   df['date']=df['date'].values.astype("float64")
   df['last_scraped']=df['last_scraped'].values.astype("float64")
   df['host_since'] = df['host_since'].values.astype("float64")
   df['calendar_last_scraped']=df['calendar_last_scraped'].values.astype("float64")
   df['first_review']=df[('first_review')].values.astype("float64")
   df['last_review']=df[('last_review')].values.astype("float64")

   column_keys = df.keys()
   scaler = StandardScaler()
   review_scores_rating = df.pop('review_scores_rating')

   df = scaler.fit_transform(df)

   df = pd.DataFrame(np.column_stack((df, review_scores_rating)), columns=column_keys)
   df = df.sample(frac=1).reset_index(drop=True)

   columns = [col for col in df.columns
           if col not in ['review_scores_rating'] ]
   lbls = []
   vals = []
   for column in columns:
      lbls.append(column)
```

```
        vals.append(np.corrcoef(df[column].values, df['review_scores_rating'].values)[0,1])

    correlation_data = pd.DataFrame({'lbls':lbls, 'vals':vals})
    correlation_data = correlation_data.sort_values(by='vals')
    fig, axes = plot.subplots(figsize=(11,17))
    axes.barh(np.arange(correlation_data.shape[0]), correlation_data.vals.values,
alpha=0.8,color='orange')
    axes.set_title("Coefficient of every column with review_scores_rating")
    axes.set_yticks(np.arange(correlation_data.shape[0]))
    axes.set_yticklabels(correlation_data.lbls.values, rotation='horizontal')
    plt.xlabel('coefficients', fontsize=10)
    plot.savefig('corr_val1.png', facecolor='w', transparent=False)
    plot.show()

    # define dataset
    y = df['review_scores_rating']
    X = df.drop(['review_scores_rating'], axis=1)
    # define the model
    model = LassoCV(cv=5)
    # fit the model
    model.fit(X, y)
    # get importance
    importance = model.coef_


    feas = pd.DataFrame()
    feas['features'] = X.keys()
    feas['importance'] = importance
    feas = feas.sort_values(by='importance')


    feas1 = feas[0:32]
    feas2 = feas[32:64]

    plot.rc('font', size=10);
    fig, axes = plot.subplots(figsize=(3,7))
    axes.barh(np.arange(feas1.shape[0]), feas1.importance.values, alpha=0.8,color='orange')
    axes.set_title("feature importance")
    axes.set_yticks(np.arange(feas1.shape[0]))
    axes.set_yticklabels(feas1.features.values, rotation='horizontal')
    plot.xlabel('coefficients', fontsize=10)
    plot.savefig('las_val1.png', facecolor='w', transparent=False)

    plot.show()

    fig, axes = plot.subplots(figsize=(3,7))
    axes.barh(np.arange(feas2.shape[0]), feas2.importance.values, alpha=0.8,color='orange')
```

```
axes.set_title("feature importance")
axes.set_yticks(np.arange(feas2.shape[0]))
axes.set_yticklabels(feas2.features.values, rotation='horizontal')
plot.xlabel('coefficients', fontsize=10)
plot.savefig('las_val2.png', facecolor='w', transparent=False)
plot.show()

# define dataset
y = df['review_scores_rating']
X = df.drop(['review_scores_rating'], axis=1)
# define the model
model = RidgeCV(cv=5)
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_


feas = pd.DataFrame()
feas['features'] = X.keys()
feas['importance'] = importance
feas = feas.sort_values(by='importance')


feas3 = feas[0:32]
feas4 = feas[32:64]

plot.rc('font', size=10);
fig, axes = plot.subplots(figsize=(3,7))
axes.barh(np.arange(feas3.shape[0]), feas3.importance.values, alpha=0.8,color='orange')
axes.set_title("feature importance")
axes.set_yticks(np.arange(feas3.shape[0]))
axes.set_yticklabels(feas3.features.values, rotation='horizontal')
plot.xlabel('coefficients', fontsize=10)
plot.savefig('rig_val1.png', facecolor='w', transparent=False)

plot.show()

fig, axes = plot.subplots(figsize=(3,7))
axes.barh(np.arange(feas4.shape[0]), feas4.importance.values, alpha=0.8,color='orange')
axes.set_title("feature importance")
axes.set_yticks(np.arange(feas4.shape[0]))
axes.set_yticklabels(feas4.features.values, rotation='horizontal')
plot.xlabel('coefficients', fontsize=10)
plot.savefig('rig_val2.png', facecolor='w', transparent=False)
plot.show()
```

```
feas_neg=feas1.head(6)
neg = feas_neg['features'].values.tolist()
feas_pos=feas2.tail(6)
pos = feas_pos['features'].values.tolist()

y = df['review_scores_rating']
X = df[neg+pos]
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size = 0.2,stratify=y)

dummy = DummyRegressor(strategy='mean').fit(Xtrain, ytrain)
ydummy = dummy.predict(Xtest)
mean_squared_error(ytest, ydummy)

dummy_r2 = r2_score(y_true=ytest, y_pred=ydummy)
print(dummy_r2)

x_ax = range(len(ytest))
plot.scatter(x_ax, ytest, s=5, color="blue", label="original")
plot.plot(x_ax, ydummy, lw=0.8, color="red", label="predicted")
plot.legend()
plot.show()

mean_error = []
std_error = []
C_range = [0.1, 0.5, 1, 5, 10, 50, 100]
for C in C_range:
   model = Lasso(alpha=1/C)
   temp = []
   kf = KFold(n_splits=5)
   for train, test in kf.split(Xtrain, ytrain):

      model.fit(Xtrain.iloc[train], ytrain.iloc[train])
      ypred = model.predict(Xtrain.iloc[train])
      temp.append(mean_squared_error(ytrain.iloc[train], ypred))
   mean_error.append(np.array(temp).mean())
   std_error.append(np.array(temp).std())
plot.errorbar(C_range, mean_error, yerr=std_error)
plot.xlabel("Ci")
plot.ylabel("Mean square error")
plot.title("k-fold cross validation review_scores_rating Lasso")
plot.xlim((0, 100))
plot.savefig('lasCV_val.png', facecolor='w', transparent=False)
plot.show()

model1 = Lasso(alpha=1/10).fit(Xtrain,ytrain)
y_pred = model1.predict(Xtest)
mean_squared_error(ytest, y_pred)
```

```
las_r2 = r2_score(y_true=ytest, y_pred=y_pred)
print(las_r2)

x_ax = range(len(ytest))
plot.scatter(x_ax, ytest, s=5, color="blue", label="original")
plot.plot(x_ax, y_pred, lw=0.8, color="red", label="predicted")
plot.legend()
plot.show()

mean_error = []
std_error = []
C_range = [0.1, 0.5, 1, 5, 10, 50, 100]
for C in C_range:
   model = Ridge(alpha=1/C)
   temp = []
   kf = KFold(n_splits=5)
   for train, test in kf.split(Xtrain, ytrain):

      model.fit(Xtrain.iloc[train], ytrain.iloc[train])
      ypred = model.predict(Xtrain.iloc[train])
      temp.append(mean_squared_error(ytrain.iloc[train], ypred))
   mean_error.append(np.array(temp).mean())
   std_error.append(np.array(temp).std())
plot.errorbar(C_range, mean_error, yerr=std_error)
plot.xlabel("Ci")
plot.ylabel("Mean square error")
plot.title("k-fold cross validation review_scores_rating Ridge")
plot.xlim((0, 100))
plot.savefig('rigCV_val.png', facecolor='w', transparent=False)
plot.show()

model2 = Ridge(alpha=1).fit(Xtrain,ytrain)
y_pred1 = model2.predict(Xtest)
mean_squared_error(ytest, y_pred1)

ridge_r2 = r2_score(y_true=ytest, y_pred=y_pred1)
print(ridge_r2)

x_ax = range(len(ytest))
plot.scatter(x_ax, ytest, s=5, color="blue", label="original")
plot.plot(x_ax, y_pred1, lw=0.8, color="red", label="predicted")
plot.legend()
plot.show()
```