Machine Learning Engineer Nanodegree

Capstone Project - Udacity

Thiago T. S. Gavioli

November, 2021

Predicting prices for Airbnb in Rio de Janeiro – Brazil

Definition

Project Overview

Booking an entire apartment, home, etc. during a trip is becoming increasingly common over time. Rather than book a room in a hotel, people have preferred entire places, whether by privacy or saving some money. In such context, Airbnb is nowadays one of the most important service for booking apartments, home, or any kind of place for staying during a trip or even a place to rent for living.

Whatever the reason, one important factor in this kind of transaction is the price. A host looks for aiming the best price which gives a good financial return and keeps the place booked most of the time, and the guests look for the best price considering the benefits.

That said, having a way of predicting the prices according to historical data is a valuable tool. For this purpose, evaluating the best options is the main goal of this project.

My personal motivation for this is because I am a heavy user of Airbnb service.

Problem Statement

Having some information about historical prices and features available, this project aims to evaluate the best ways of predicting prices for listings in Airbnb service located in Rio de Janeiro, one of the most important cities from Brazil. This way, we would be able to help either hosts or guests to optimize their decisions when it comes for price.

Metrics

Since our goal is to predict values (price) according to given features, the approach to this problem will be through regression models. Due to the nature of the approach, the chosen evaluation metric for the project is the R2 score, which tells us how well a prediction approximates the real data points [2].

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

R2-score formula

Analysis

Data Exploration

For this project we are going to use two datasets provided by Airbnb [1] with information from Rio de Janeiro.

The datasets are:

- calendar.csv
- listings.csv

Calendar contains mainly information about price and if the listing is available or not in a specific date. It is possible to relate this dataset to listings since it includes the listing ids.

It has 8.545.976 of rows and 7 columns. The date ranges from 2021-09-28 to 2022-09-29.

Some statistics, abnormalities and characteristics properly handled **from calendar dataset** are listed in the following lines.

- First lines of the dataset

```
#Loading and checking Calendar dataframe
df_calendar = pd.read_csv('calendar.csv')
df_calendar.head()
```

	listing_id	date	available	price	adjusted_price	minimum_nights	maximum_nights
0	1151855	2021-09-29	f	$500.00	$500.00	1.0	1125.0
1	858748	2021-09-28	t	$200.00	$200.00	5.0	1125.0
2	858748	2021-09-29	t	$200.00	$200.00	5.0	1125.0
3	858748	2021-09-30	t	$200.00	$200.00	5.0	1125.0
4	858748	2021-10-01	t	$130.00	$130.00	5.0	1125.0

- Missing values for price columns were dropped

```
[11]  1 #Checking null values
      2 df_calendar.isnull().sum()
```

```
listing_id          0
date                0
available           0
price             194
adjusted_price    194
minimum_nights   1113
maximum_nights   1113
dtype: int64
```

```
[12]  1 #Dropping NAs of price column
      2 df_calendar.dropna(subset=['price'], inplace=True)
```

- Turning price column into float to future analysis

```
[13]  1 #turning price into float
      2 def remove_symbol(price):
      3     """remove the symbols ['$', ','] in price"""
      4     if type(price) is str:
      5         return re.sub("[\$,]",'',price)
      6     return price
      7
      8 df_calendar['price'] = df_calendar.price.apply(remove_symbol)
      9 df_calendar['price'] = df_calendar['price'].astype(float)
```

- Checking for outliers based on Tukey test

```
[15]  1 def tukey_test(df, column_name):
      2   Q1 = df[column_name].quantile(0.25)
      3   Q3 = df[column_name].quantile(0.75)
      4
      5   IQR = Q3 - Q1
      6
      7   Max = Q3 + 1.5*IQR
      8   Min = Q1 - 1.5*IQR
      9
     10   return print('For {} column, the min value should be {} and the max value should be {}'.format(column_name, Min, Max))
```

```
[16]  1 tukey_test(df=df_calendar, column_name='maximum_nights')
```

```
For maximum_nights column, the min value should be -1462.5 and the max value should be 2677.5
```

```
[17]  1 tukey_test(df=df_calendar, column_name='price')
```

```
For price column, the min value should be -500.0 and the max value should be 1260.0
```

```
[18]  1 tukey_test(df=df_calendar, column_name='minimum_nights')
```

```
For minimum_nights column, the min value should be -1.0 and the max value should be 7.0
```

- Removing outliers

```
[19]  1 #Removing outliers from df_calendar
      2 df_calendar = df_calendar.query('maximum_nights < 2677')
      3 df_calendar = df_calendar.query('price < 1260')
      4 df_calendar = df_calendar.query('minimum_nights < 7')
```

Listings contains the detailed information about each place available in Airbnb. Details about the host, price, neighborhood, review, amenities, etc., can be found. Here are the features which will be used to predict prices for each place/listing.

The original dataset has 23.414 rows and 74 columns (features + price). The range of date is the same as presented in calendar dataset.

The raw dataset contains different types of data (int., float, object) which must be handled before being analyzed and used for modelling. Missing data, outliers and other problems will properly handle in the context of the problem.

Some statistics, abnormalities and characteristics properly handled **from listings dataset** are listed in the following lines.

- Part (since it is too big to show) of listings dataset

```
[29]  1 df.head(2)
```

	id	listing_url	scrape_id	last_scraped	name	description	neighborhood_overview	picture
0	17878	https://www.airbnb.com/rooms/17878	20210928134526	2021-09-28	Very Nice 2Br in Copacabana w. balcony, fast WiFi	Discounts for long term stays. Large b...	This is the one of the bests spots in Rio. Bec...	https://a0.muscache.com/pictures/65320518/3
1	24480	https://www.airbnb.com/rooms/24480	20210928134526	2021-09-29	Nice and cozy near Ipanema Beach, w/ home office	My studio is located in the best of Ipanema, t...	The beach, the lagoon, Ipanema is a great loca...	https://a0.muscache.com/pictures/11955612/b

- Dropping columns that at a first glance will not help our prediction model

```
[30]  1
      2 df.drop(['listing_url', 'scrape_id', 'last_scraped', 'name', 'description',
      3        'neighborhood_overview', 'picture_url', 'host_id', 'host_url',
      4        'host_name', 'host_since', 'host_location', 'host_about',
      5        'host_thumbnail_url', 'host_picture_url',
      6        'host_neighbourhood', 'host_listings_count',
      7        'host_total_listings_count', 'host_verifications',
      8        'minimum_nights', 'maximum_nights', 'minimum_minimum_nights',
      9        'maximum_minimum_nights', 'minimum_maximum_nights',
     10        'maximum_maximum_nights', 'minimum_nights_avg_ntm',
     11        'maximum_nights_avg_ntm', 'calendar_updated', 'calendar_last_scraped',
     12        'license', 'calculated_host_listings_count',
     13        'calculated_host_listings_count_entire_homes',
     14        'calculated_host_listings_count_private_rooms',
     15        'calculated_host_listings_count_shared_rooms', 'instant_bookable', 'has_availability'], axis=1, inplace=True)
```

- Remaining columns

```
[31]  1 df.columns
```

```
Index(['id', 'host_response_time', 'host_response_rate',
       'host_acceptance_rate', 'host_is_superhost', 'host_has_profile_pic',
       'host_identity_verified', 'neighbourhood', 'neighbourhood_cleansed',
       'neighbourhood_group_cleansed', 'latitude', 'longitude',
       'property_type', 'room_type', 'accommodates', 'bathrooms',
       'bathrooms_text', 'bedrooms', 'beds', 'amenities', 'price',
       'availability_30', 'availability_60', 'availability_90',
       'availability_365', 'number_of_reviews', 'number_of_reviews_ltm',
       'number_of_reviews_l30d', 'first_review', 'last_review',
       'review_scores_rating', 'review_scores_accuracy',
       'review_scores_cleanliness', 'review_scores_checkin',
       'review_scores_communication', 'review_scores_location',
       'review_scores_value', 'reviews_per_month'],
      dtype='object')
```

- Missing values (drop and fill in with mode in some cases)

```
[33]  1 #Dropping columns over 40% of missing values
      2 df.drop(['neighbourhood', 'neighbourhood_group_cleansed', 'bathrooms',], axis=1, inplace=True)
```

```
[34]  1 #For some columns we are going to fill in missing values with the most frequent value
      2 for column in ['host_response_time', 'host_is_superhost', 'host_has_profile_pic', 'host_identity_verified',
      3                'bathrooms_text', 'bedrooms', 'beds', 'first_review', 'last_review']:
      4                df[column].fillna(df[column].mode()[0], inplace=True)
```

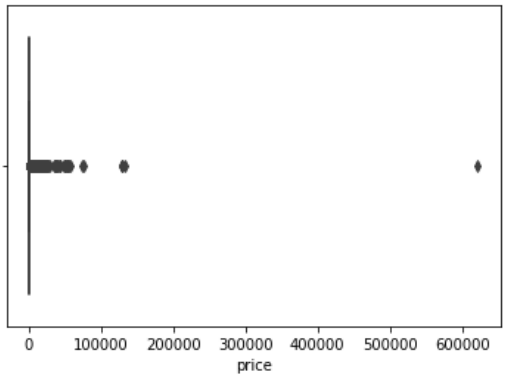
- Missing values (fill in with mean in some cases)

```
[37]  1 #Turning rate columns with % to numeric
      2 df['host_response_rate'] = pd.to_numeric(df['host_response_rate'].str.strip('%'))
      3 df['host_acceptance_rate'] = pd.to_numeric(df['host_acceptance_rate'].str.strip('%'))
      4
```

```
[38]  1 #For some columns we are going to fill in missing values with the mean value
      2 for column in ['host_response_rate', 'host_acceptance_rate', 'review_scores_rating',
      3                'review_scores_accuracy','review_scores_cleanliness', 'review_scores_checkin',
      4                'review_scores_communication', 'review_scores_location','review_scores_value',
      5                'reviews_per_month']:
      6                df[column].fillna(df[column].mean(), inplace=True)
```

- Price range on raw data

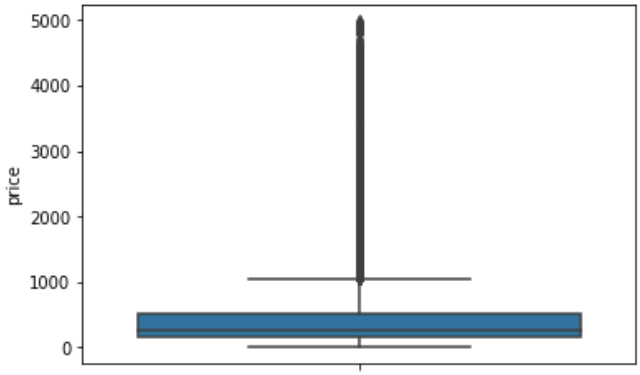
```
[41]  1 #Checking range of price
      2 ax = sns.boxplot(x=df['price'])
```



From the boxplot above we can see so many outliers in our data. To deal with it, we put a threshold of 5,000.

```
[43]  1 df = df[df['price']<5000]
```

```
[44]  1 #Checking new range of price
      2 ax = sns.boxplot(y=df['price'])
```



- Handling of the amenities column (exploring the most commons)

```
1 # Handle amenities - Explore most common amenities
2 amenities = []
3 for index, row in df.iterrows():
4     row['amenities']= row['amenities'].replace('[', '')
5     row['amenities']=row['amenities'].replace(']', '')
6     row['amenities']=row['amenities'].replace('"', '')
7     my_list= row['amenities'].split(',')
8     for n in my_list:
9         amenities.append(n)
10
11 amenities_count= Counter(amenities)
12 amenities_count.most_common()
```

```
[(' Wifi', 20206),
 (' Long term stays allowed', 19571),
 (' Essentials', 19347)
```

Checking the most common amenities we decided to keep the top 17.

```
[52]  1 amenities_new_cols =['amenities_Wifi|Wifi','amenities_Long term stays allowed|Long term stays allowed',
      2                      'amenities_Essentials|Essentials','amenities_Kitchen|Kitchen',
      3                      'amenities_Air conditioning|Air conditioning','amenities_Iron|Iron','amenities_Elevator|Elevator',
      4                      'amenities_Hot water|Hot water','amenities_Dedicated workspace|Dedicated workspace','amenities_TV|TV',
      5                      'amenities_Dishes and silverware|Dishes and silverware',
      6                      'amenities_Cooking basics|Cooking basics','amenities_Hangers|Hangers',
      7                      'amenities_Hair dryer|Hair dryer', 'amenities_Refrigerator|Refrigerator',
      8                      'amenities_Microwave|Microwave', 'amenities_Free parking on premises|Free parking on premises',]
      9
     10 for c in amenities_new_cols:
     11     c_col, c_desc= c.split('|')
     12     df[c_col]= df['amenities'].apply(lambda x: 1 if c_desc in x else 0)
```

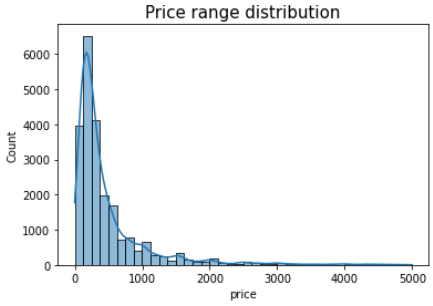
Exploratory visualization

In order to understand the data we are dealing with and want to model, it is very important to do a good exploratory data analysis. In this section we are going to visualize the distribution and relationship of some important variables present in our two datasets.

- Checking our target variable, price, distribution

```
1 #Price histogram
2 sns.histplot(data=df, x="price", bins=40, kde=True)
3 plt.title('Price range distribution', fontdict={'fontsize': 15})
```

```
Text(0.5, 1.0, 'Price range distribution')
```

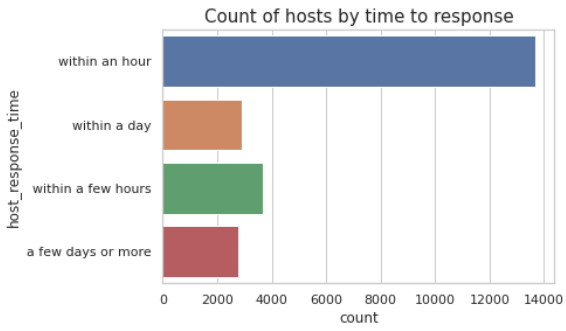


- Some hosts information

Count of hosts by time to response

```
1 #Plotting the quantity of hosts according to time to response groups
2 sns.set_theme(style="whitegrid")
3 sns.countplot(y='host_response_time', data=df)
4 plt.title('Count of hosts by time to response', fontdict={'fontsize': 15})
```

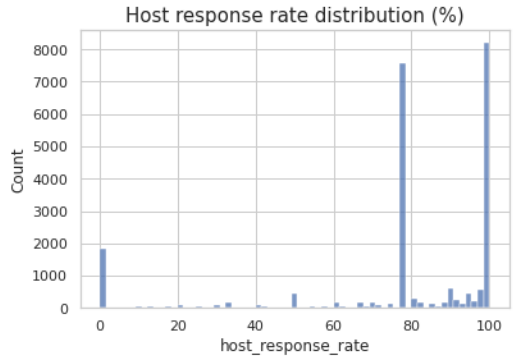
```
Text(0.5, 1.0, 'Count of hosts by time to response')
```



Host response rate distribution (%)

```
#Host response rate histogram
sns.histplot(data=df, x="host_response_rate")
plt.title('Host response rate distribution (%)', fontdict={'fontsize': 15})
```

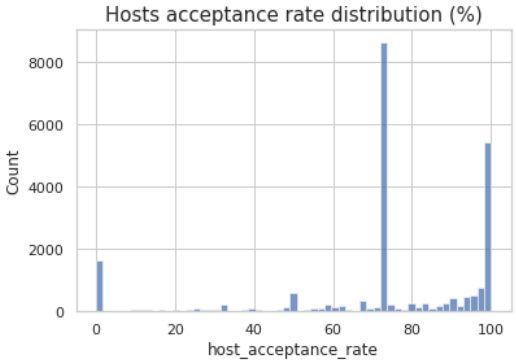
```
Text(0.5, 1.0, 'Host response rate distribution (%)')
```



Hosts acceptance rate distribution (%)

```
#Host acceptance rate histogram
sns.histplot(data=df, x="host_acceptance_rate")
plt.title('Hosts acceptance rate distribution (%)', fontdict={'fontsize': 15})
```

```
Text(0.5, 1.0, 'Hosts acceptance rate distribution (%)')
```



Hosts characteristics

```
fig, ax =plt.subplots(nrows=1, ncols=3, sharey=True, figsize=(10, 5))
fig.suptitle('Hosts characteristics', fontsize=16)

sns.countplot(df['host_is_superhost'], ax=ax[0])
sns.countplot(df['host_has_profile_pic'], ax=ax[1])
sns.countplot(df['host_identity_verified'], ax=ax[2])

fig.show()
plt.tight_layout(pad=5, w_pad=5)
```

Relationship between price and being a superhost or not

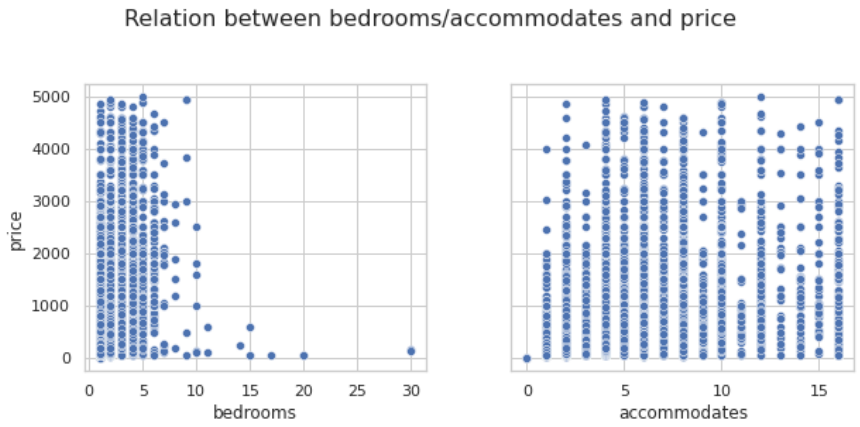
```
sns.barplot(data=df, x='host_is_superhost', y='price')
plt.title('Mean price by superhost or not', fontdict={'fontsize': 15})
```

```
Text(0.5, 1.0, 'Mean price by superhost or not')
```

From above we note that being a superhost is no guarantee of higher prices.

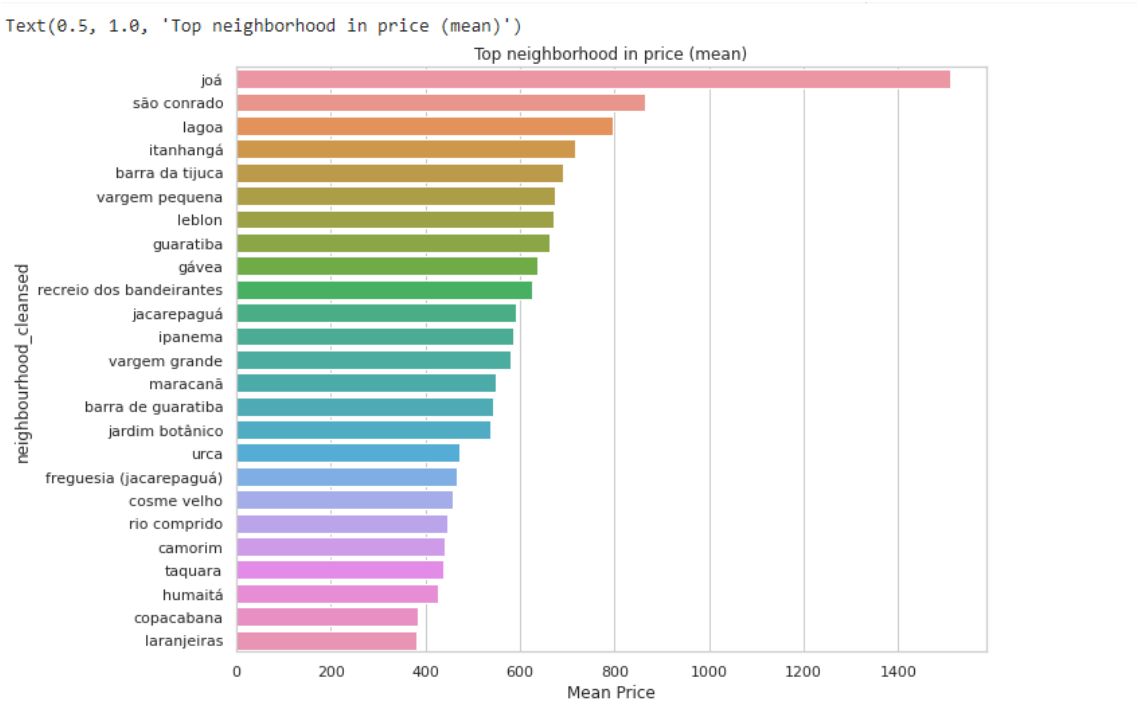
- Relation between bedrooms/accommodates and price

```
fig, ax =plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(10, 5))
fig.suptitle('Relation between bedrooms/accommodates and price', fontsize=16)

sns.scatterplot(data=df, x='bedrooms', y='price', ax=ax[0])
sns.scatterplot(data=df, x='accommodates', y='price', ax=ax[1])

fig.show()
plt.tight_layout(pad=5, w_pad=5)
```

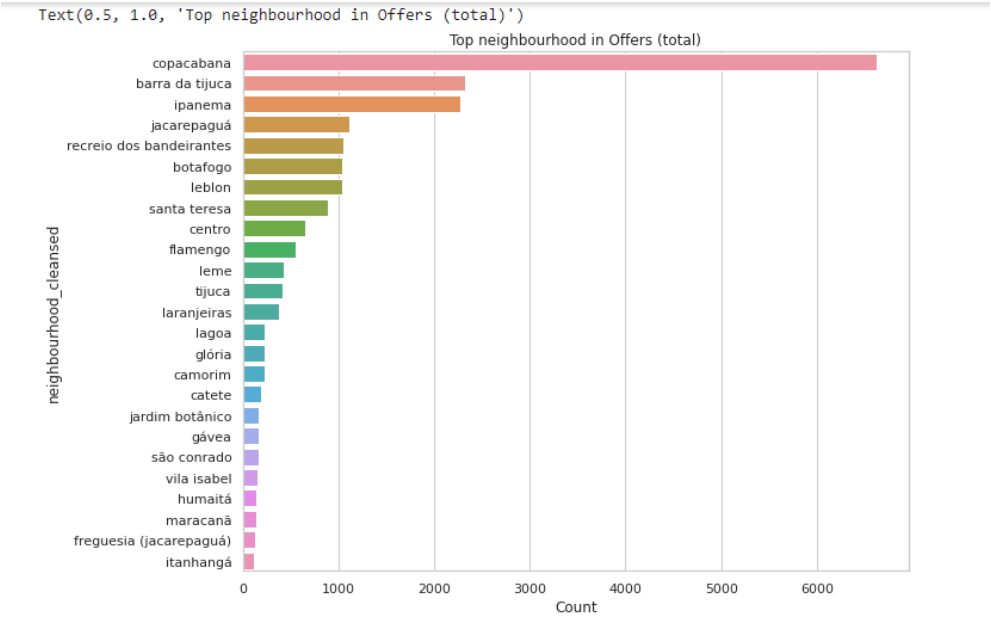


Checking the charts above we cannot see a clear relation between bedrooms/accommodates and increase or decrease in prices.

- Top n neighborhoods in price (only over 50 samples included)



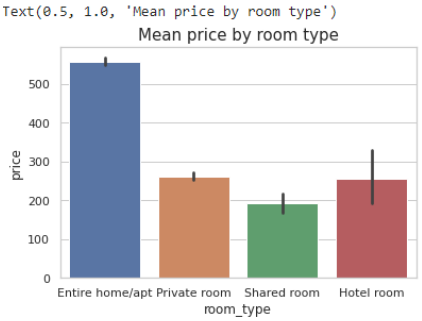
- Top n neighborhoods in offers (only over 50 samples included)



From the two last charts above we see that the neighborhood affects the price, and most of the offers available is contained in 10 to 15 neighborhoods. A highlight to Copacabana which has more than twice the total offers of the second place, Barra da Tijuca.

- Mean price by room type

```
1 sns.barplot(data=df, x='room_type', y='price')
2 plt.title('Mean price by room type', fontdict={'fontsize': 15})
```



Algorithms and Techniques

The proposed solution to this problem is applying machine learning regressor algorithms to predict price according to selected features.

First, we are going to handle the raw features in order to find the best inputs to each model evaluated. We will clean the dataset according to the necessity, analyze all the information we have through an exploratory data analysis, select the best features to fit the models and evaluate which model performs best.

The algorithms that will be analyzed are: SVR, Linear regression, Random Forest regressor and XGB. The evaluation metric abovementioned will be used to compare the models.

Benchmark

For the benchmark we are going to use a simple linear regression model from scikit-learn library.

Methodology

Data Preprocessing

We are going to work on our datasets to preprocess what is necessary in order to make them uniform and ready to our models.

- Map false and true columns to 1 or 0
- Extract only year from first and last review
- Drop unnecessary columns ['id', 'first_review', 'last_review']
- Keep only the rows which have a neighborhood that is present over 50 times in the dataset
- Get dummy variables from categorical features
- Split into train and test (70%/30%)

Implementation

I will start with a simple linear regression model, then trying different models and hyper-parameters to reach the best performance according to the metric picked out. It is important to mention that R2-score will be considered in our analysis only on test data.

- Benchmark model

Our benchmark, as defined previously, is going to be the linear regression model.

```
[196]  1 lm_model = LinearRegression(normalize=True) # Instantiate
       2 lm_model.fit(X_train, y_train) #Fit

       LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=True)

[197]  1 y_train_preds = lm_model.predict(X_train) #predict on train data
       2 y_test_preds = lm_model.predict(X_test)  #Predict on test data

[198]  1 print('The train R2 score is: {}, and the test R2 score is {}'.format(r2_score(y_train, y_train_preds), r2_score(y_test, y_test_preds)))

       The train R2 score is: 0.4091679151415173, and the test R2 score is 0.4244736105746165
```

As shown above, the R2-score for our benchmark model with default parameters is 0.4244.

- Random Forest Regressor

```
[199]  1 clf = RandomForestRegressor(max_depth=20, n_estimators=100)
```

```
[200]  1 clf.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=20, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
[201]  1 y_train_preds = clf.predict(X_train)
       2 y_test_preds = clf.predict(X_test)
```

```
[202]  1 print('The train R2 score is: {}, and the test R2 score is {}'.format(r2_score(y_train, y_train_preds), r2_score(y_test, y_test_preds)))
```

```
The train R2 score is: 0.9192443489379294, and the test R2 score is 0.4878837420036757
```

For our random forest regressor model we got a R2-score of 0.4878. Even though it is not as close to 1 as we wish, it is better than our benchmark model. Some improve can be seen.

- SVR

```
[207]  1 SVR = SVR('linear', C=1.0, epsilon=0.2)
```

```
[208]  1 SVR.fit(X_train,y_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.2, gamma='scale',
    kernel='linear', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
[209]  1 y_train_preds = SVR.predict(X_train)
       2 y_test_preds = SVR.predict(X_test)
```

```
[210]  1 train_score = r2_score(y_train, y_train_preds)
       2 test_score = r2_score(y_test, y_test_preds)
```

```
[211]  1 print('The train R2 score is: {}, and the test R2 score is {}'.format(train_score, test_score))
```

```
The train R2 score is: 0.3381551612996436, and the test R2 score is 0.3580549998382774
```

In the case of SVR we had a decrease in our evaluation metric, what is not good. We got a R2-score of 0.35, below our benchmark which is 0.4244.

- XGboost

```
#train XGBoost model
xgb = xgboost.XGBRegressor(n_estimators=50, learning_rate=0.05, subsample=0.75,
                           max_depth=7)
```

```
xgb.fit(X_train,y_train)
```

```
[23:06:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=7, min_child_weight=1, missing=None, n_estimators=50,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.75, verbosity=1)
```

```
y_train_preds = xgb.predict(X_train)
y_test_preds = xgb.predict(X_test)
```

```
print('The train R2 score is: {}, and the test R2 score is {}'.format(r2_score(y_train, y_train_preds), r2_score(y_test, y_test_preds)))
```

```
The train R2 score is: 0.6424865586477666, and the test R2 score is 0.5095390950617028
```

Implementing XGboost we got a performance of R2-score equals to 0.5095. It is better than our benchmark model, as well as the other models.

Refinement

Having the best model among the ones analyzed, XGboost, we are going to try to improve our result doing some refinement process on it. The chosen process is Gridsearchcv which makes, according to its documentation, an exhaustive search over specified parameter values for an estimator. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid [3].

We will vary the params n_estimators and learning rate.

```
[228]  params = {
           'xgb__n_estimators': [100, 150],
           'xgb__learning_rate': [0.08, 0.1]
       }
```

Implementing gridsearchcv

```
[229]  cv = GridSearchCV(xgb, param_grid=params, scoring='r2', verbose=3)
```

```
[230]  cv.fit(X_train, y_train)
```

```
[23:30:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    4.0s remaining:    0.0s
[CV]  xgb__learning_rate=0.08, xgb__n_estimators=100, score=0.501, total=   3.9s
[CV] xgb__learning_rate=0.08, xgb__n_estimators=100 ..................
[23:30:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    7.9s remaining:    0.0s
[CV]  xgb__learning_rate=0.08, xgb__n_estimators=100, score=0.499, total=   3.9s
[CV] xgb__learning_rate=0.08, xgb__n_estimators=100 ..................
[23:30:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[CV]  xgb__learning_rate=0.08, xgb__n_estimators=100, score=0.495, total=   3.9s
[CV] xgb__learning_rate=0.08, xgb__n_estimators=100 ..................
```

Checking the best params found and retraining our XGB model

```
[231] 1 #Checking the best params
      2 cv.best_params_
```

```
{'xgb__learning_rate': 0.08, 'xgb__n_estimators': 100}
```

```
[240] 1 xgb = xgboost.XGBRegressor(n_estimators=100, learning_rate=0.08, subsample=0.75,
      2                            max_depth=7)
```

```
[241] 1 xgb.fit(X_train,y_train)
```

```
[23:36:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.08, max_delta_step=0,
             max_depth=7, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=0.75, verbosity=1)
```

Predicting with best params

```
1 #Predict
2 y_train_preds = xgb.predict(X_train)
3 y_test_preds = xgb.predict(X_test)
```

```
[243] 1 print('The train R2 score is: {}, and the test R2 score is {}'.format(r2_score(y_train, y_train_preds), r2_score(y_test, y_test_preds)))
```

```
The train R2 score is: 0.7760108271808667, and the test R2 score is 0.5477201923097432
```

Implementing Gridsearchcv on XGboost, we found the best params according to the range set forth and got a performance of R2-score equals to 0.5477. It is our best result.

Results

Model evaluation and validation

A test dataset was used to evaluate the performance of our models, and the XGBoost got the best performance according to the metric chosen – R2score.

The best model has the following final architecture:

- XGBoost regressor
- N_estimators: 100
- Learning rate: 0.08
- Subsample: 0.75
- Max_depth: 7

In the end, our focus was to get a model with the high R2-score as possible. The R2-score ranges from 0 to 1, and trying different solutions with the features we have and the preprocessing steps adopted we got 0.5477 as our best solution.

Justification

According to the data available we designed a model able to predict prices for Airbnb better than our benchmark. That was the main problem to be solved and we can say that we did that. The best model is about 29% better in performance than the benchmark.

It is important to note that our results could be more satisfactory if we went deeper in every step adopted in this project, and furthermore, one step before: collecting better data.

Some improvements can guide next steps, for example avoiding so many missing data, since we had to fill in some Nan values with mode and mean, what can bias our results or/and get them worse. Varying more parameters of our regression models is another alternative in order to try to improve our results.

In any case, I understand that the work done in this project gave a good view of the data collected from Airbnb in Rio de Janeiro, Brazil. We understand the distribution of the main variables, their relationship with the target variable, price, and the models tested is a good parameter and point for further analysis.

References

[1] Datasets provided by Airbnb,

http://insideairbnb.com/get-the-data.html

[2] Interpretation of R2,

https://en.wikipedia.org/wiki/Coefficient_of_determination#:~:text=R2%20is%20a%20statistic,predictions%20perfectly%20fit%20the%20data.

[3] Gridsearchcv documentation

https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

---