Chicago airbnb : Predictive modelling

September 2022

1 Introduction

Airbnb, as in "Air Bed and Breakfast," is a service that lets property owners rent out their spaces to travelers looking for a place to stay. Travelers can rent a space for multiple people to share, a shared space with private rooms, or the entire property for themselves. It is world-wide service. And quite a practical service as it offers multiple choices for finding housing during your vacation or your short term residence with great prices.

As part of this Machine Learning project, we chose to build a predictive model that can output Airbnb prices for a specific area. To do so, we will focus on the city of Chicago, one of the United States' biggest cities and the most populous city of the state of Illinois. Our aim is to train a model using a pre-existing data and test it using new one.

In this rapport, we will first understand the data and formulate the ML problem. Then, we will talk about feature engineering and identify our features and labels. In the next step we will use two ML methods (models) to fit our data, and then compare them in terms of performance. The last part would be to formulate a conclusion on which we will choose the better model.

2 Problem Formulation and Understanding the data

In order to build a predictive model, we need a comprehensive data that includes multiple variables that could potentially influence the price tags of the Airbnb rental houses. To that extent, we were able to find a data that holds many features. The data can be found here Airbnb data.

There are some columns in the data that are irrelevant to our model, such as the name of the host, ID ..., so we will drop them from the beginning. Here are the columns we are going to use to build our model.

1. id, minimum nights, number of reviews,reviews per month, price: numeric
2. neighbourhood : Qualitative
3. room type : categorical

Each row of data contains information about a booking at a chicago Airbnb. The information includes the neighbourhood of the housing, the reviews previously given, type of the room and the price.

3 Feature engineering

Now that we have a clear idea of our data, we need to transform raw data into features we can work with to build the predictive model.

Now, the ML problem will be trying to use this data to predict the prices of Airbnb housing.

3.1 Feature Selection:

In this step, we essentially analyze, judge, and rank various features to determine which features are irrelevant, or redundant. We used the seaborn library to plot the linear correlations between all numeric variables to see how they relate to each other. See Figure 1. We can see a clear correlation between the two features reviews per month and number of reviews. So we will remove one and keep the other.

3.2 Feature Creation:

In this step we will try to manipulate the two columns Neighborhood and room type. They are categorical variables, and we need to convert them into numeric by creating a substitution that makes sense statistically.

3.2.1 Neighborhood:

This column contains neighborhood names in which the Airbnb listing takes place. This is significant when predicting the price, because there are expensive neighborhoods as well as cheap ones. We can't simply just encode categorical predictors as dummy numerical values, meaning that we label the first neighborhood as 1, the second as 2,...and so on. We need to find a label for this column that makes sense machine learning wise. The way we thought best to do is replace the neighborhood names by their safety scores. It makes sense because a safe neighborhood would be more expensive than a neighborhood with higher crime rate for example. The data we used for this is the official ratings of the state of Illinois. You can find it here safety scores. See also Figure 2.

3.2.2 Room type:

The column room type holds categorical variables. Since we only have four different types of room (Hotel room, private room, shared room and entire home/apt), we could use One-hot encoding. A commonly used technique in Machine Learning. Basically, four different columns will be created, one for each type and each of the columns will have binary values (0 or 1).

4 Features and labels

We can now state clearly our features and the label.
Features:

1. neighbourhood score, minimum nights, number of reviews : numeric
2. Hotel room, shared room, Entire home, private room : binary

Label: price (numeric)

See Figure 3 for a sample of the final data.

Since we have features and a label. This a Surpervised Machine learning problem.

5 ML methods

Before discussing the ML methods, we will first talk about the loss function we will use for both models, and the way in which we obtained the training and validation sets from the data.

Loss function

We used mean squared error as our loss function for the following reasons. MSE is sensitive towards outliers. MSE is also best used with regression as our target is heavily conditioned on the input.

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

Construction of validation and training data

Conforming with the standard practice, and setting the test data apart (15 %) we split our data into two sets one for training and one for validation with a ratio of 75 percent and 25 percent. To verify that the choice wasn't at all arbitrary and could potantialy give us a false sense of the training and validation errors, we performed a 10 fold cross validation and calculated the training and validation errors each time.

5.1 Polynomial regression

5.1.1 Choice justification

Since this a price prediction problem, which is a continuous variable, regression is the best way to proceed. From Appendix 1 (pair plots), we noticed lack of linearity between the features and the label. So, linear regression was not the best fit for this. On the other hand, there is polynomial relation between the safety score and the price. Furthermore, polynomial regression provides the best approximation of the relationship between dependent and independent variables. In our case, for example neighborhood score is independent of minimum nights... For the stated reasons, we opted for a polynomial regression with degree 2. The second degree is the best one because with higher degrees there is clear overfitting as well as higher time complexity.

5.1.2 Performance

Please refer to the code section Method 1 : Polynomial regression for analysis and plots.

Using cross validation, we plotted the training and validation errors for each fold. We noticed that the errors are close to each other for every split in the cross validation process. So, choosing a split randomly would give us a sense of the true performance of our model. The errors are relatively low and close to each other, with the validation error slightly bigger which would support the fact that the model was a good fit.

5.2 MLP regression

5.2.1 Choice justification

Following the same reasoning in the choice justification for polynomial regression, we identified that the problem we are working on is price prediction and therefore regression was our best bet.

Now, deep learning is proven to be useful and reliable in a wide range of problems. There are three main classes of artificial neural networks. From these classes, we decided to choose MLP because it is highly suitable for regression prediction problems where a real-valued quantity is predicted given a set of inputs. Also, MLP offers flexibility and reliability which are two of the most important aspects to look for in a model.

5.2.2 Choosing the number of layers

Please refer to the code section : Method 2 : MLP (Multi-layer Perceptron) Regression:

MLP requires as a parameter, the number of (hidden) layers. To make a significant choice, we setting the number of neurons to be 15, and then plotted the training and validation errors using a different number of layers each time. We could notice from the graph that some layers give smaller errors than others. To validate our choice, we calculated the difference between training and validation errors for each layer because we want the errors to be relatively close to each other. Using two layers would certainly produce errors that are small and close to each other.

6 Comparing the models

Please refer to the code section : Comparing the two models:

In order to compare the two models, we will rely on three factors or criteria: Training and validation errors, time complexity and Akaike's Information Criterion (AIC).

6.1 Training and validation errors

We used 10-Fold cross validation to plot the training and validation errors for both models. See Figure 4. For the training errors, we notice the polynomial regression performs better because it is lower than that of MLP at every fold used. They also, take exactly the same shape.

For the validation errors, for the most part, both models take similar values except for the last fold (MLP performs better).

→ Based on training and validation errors : MLP is better.

6.2 Time complexity

We calculated the time it takes each model to fit the data. We got the following results : MLP takes around 2.5 seconds, on the other hand, Polynomial regression takes only 0.0033 seconds which a tiny fraction of the time it takes MLP.

This makes sense, because MLP has hidden layers and a much more complex network. Hence, it takes more time to perform calculations.

→ Based on time complexity : Polynomial regression is better.

6.3 The Akaike information criterion

The Akaike information criterion (AIC) is a mathematical method for evaluating how well a model fits the data it was generated from. In statistics, AIC is used to compare different possible models and determine which one is the best fit for the data. A lower AIC means a better model.

We calculated the AIC for both models. The AIC for MLP is relatively large in comparison to Polynomial regression which has a negative AIC. That means the polynomial regression preforms better and there is, statistically speaking, a lower degree of information loss.

→ Based on AIC : Polynomial regression is by far better than MLP.

Conclusion for the comparaison

In regard to validation errors, MLP is slightly better. The difference is not very noticeable. So, for this factor, both models could be considered equal in terms of performance. However, taking into account time complexity and AIC, polynomial regression is definitely better than MLP.

The previous conclusion makes sense since we have a relatively small sample of data (around 4000 rows) as well as a small number of features (7 features used for training the models). MLP would perform better with more data. In fact, Deep learning has become widely used in the past ten years because of the enormous datasets available in almost every domain right now. The world has become more digitalized and therefore Deep learning algorithms were able to construct complex neural networks and give much more accurate results.

Test sets and test errors

Now that we identified polynomial regression as the better model, we need to test it using the testing set.

The test set was never used for training the model nor for validation. We constructed this set by taking 15 % of the original because we don't a have a large sample (4000 rows).

Please refer to the code section : Construction of the train, validation and test sets:

Using the test set, the testing error was: 0.4368053210067403.

7 Conclusion

The goal of this study was to train a model to predict Airbnb prices as accurate as possible given a certain amount of data. We used two models, polynomial regression and MLP. they both produced reliable results, but in terms of performance, polynomial regression was slightly better.However, we believe that if provided with additional data (samples and features) to train the model, the results could improve significantly using MLP.

Refrences :

1) Dataset link : Airbnb data

2) Safety scores for chicago neighborhoods : safety scores

3) Converting categorical variables into numeric blog : blog link

4) K-fold cross validation : link

5) How to create pair plots : link

6) When to use Neural networks : link

Figure 1: Pair plots

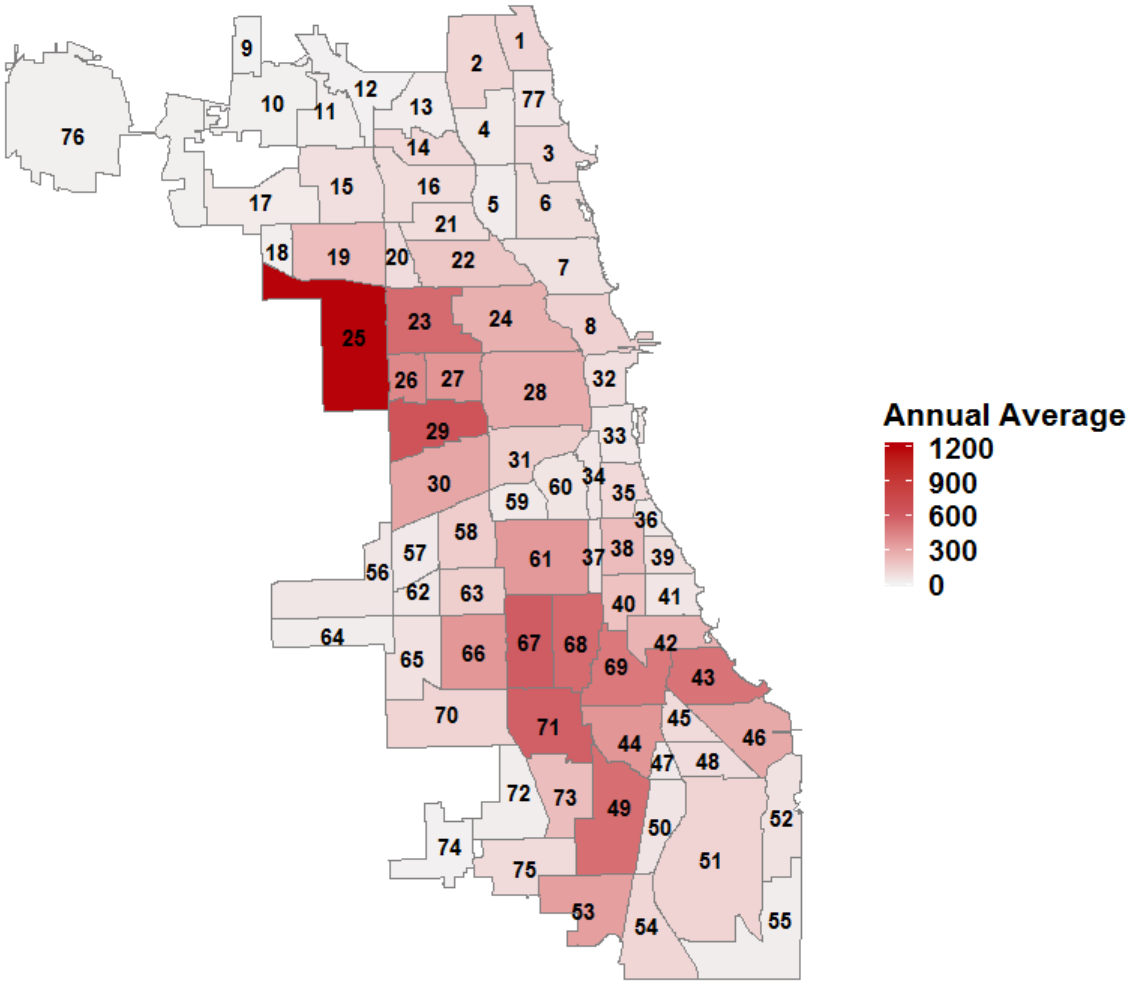


Figure 2: Safety score of Chicago neighborhoods

	neighbourhood	neighbourhood_score	Entire home/apt	Hotel room	Private room	Shared room	minimum_nights	number_of_reviews	price
221	Bridgeport	73	1	0	0	0	2	50	76
2918	Logan Square	65	1	0	0	0	1	279	90
343	Woodlawn	52	0	0	1	0	1	7	52
2799	West Garfield Park	23	1	0	0	0	1	324	39
2456	Edgewater	73	0	0	1	0	1	50	66
335	Lincoln Park	90	1	0	0	0	4	0	153
2413	Logan Square	65	1	0	0	0	2	121	63
1133	Near North Side	73	1	0	0	0	3	40	171
2222	Hyde Park	61	1	0	0	0	1	6	70
2605	Humboldt Park	45	1	0	0	0	3	3	216

Figure 3: Final data

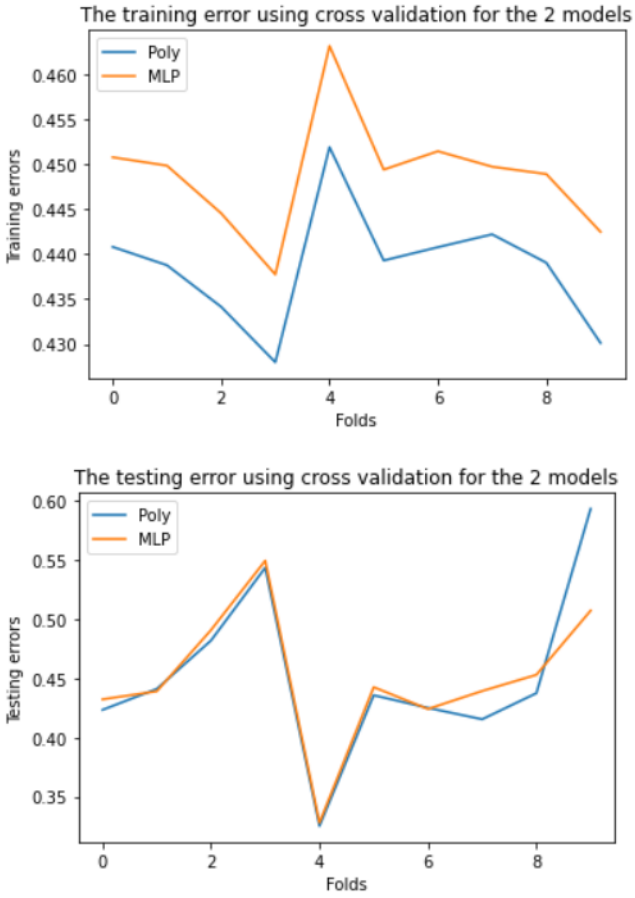


Figure 4: Training and validation errors for each model

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_validate
from sklearn.linear_model import Ridge
from sklearn.neural_network import MLPRegressor
import timeit
```

Understanding and cleaning the data (Feature engineering)

In [2]:

```
# store the data into a pandas dataframe
df = pd.read_csv('airbnb_chicago.csv')


dff = df.drop(columns=['name','host_id','neighbourhood_group', 'host_name','latitude
                'longitude', 'calculated_host_listings_count', 'last_review', 'avail

# take a look
dff.head(100)
```

Out[2]:

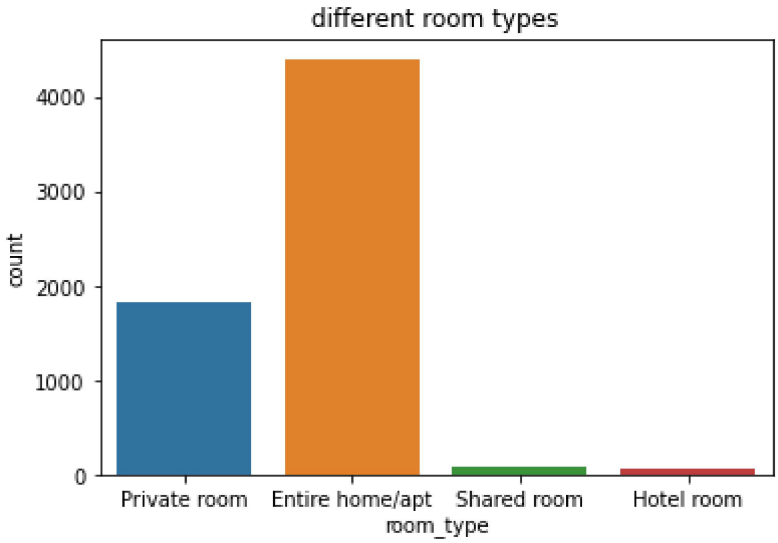
	id	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_per_mc
0	2384	Hyde Park	Private room	60	2	178	
1	4505	South Lawndale	Entire home/apt	105	2	395	
2	7126	West Town	Entire home/apt	60	2	384	
3	9811	Lincoln Park	Entire home/apt	65	4	49	
4	10610	Hyde Park	Private room	21	1	44	
...	...	...	...	...	...	...	
95	903996	Logan Square	Private room	110	2	145	
96	929914	Lincoln Park	Entire home/apt	115	2	311	
97	931105	East Garfield Park	Private room	101	3	23	
98	949514	West Town	Private room	50	3	133	
99	960326	West Town	Entire home/apt	146	1	274	

100 rows × 7 columns

Visualisation

Here we will visualise the datapoint "room type". Let us visualise the distribution of the different room_types.

In [3]:
```
sns.countplot(x = "room_type", data = dff)
plt.title("different room types")
plt.show()
```



In [4]:
```
# room_type and neighbourhood are categorical variables. We need to quantify them.
# room_type
room_labels = pd.get_dummies(dff['room_type'])

final_data = pd.concat([dff, room_labels], axis=1)#dff.drop(columns=['room_type'])

#  Neighbourhood
safety_score = pd.read_csv('score.txt')
final = final_data.merge(safety_score,on=["neighbourhood"])

airbnb_data = final.sample(random_state=1, frac=1).reset_index()
airbnb_data.pop("index")

# rearange columns
airbnb = airbnb_data.reindex(columns= ['id', 'neighbourhood', 'neighbourhood_score',
                                       'Shared room', 'minimum_nights', 'number_of_

airbnb.dropna()

# This the final data cleaned
airbnb.head(100)
```

Out[4]:

	id	neighbourhood	neighbourhood_score	Entire home/apt	Hotel room	Private room	Shared room	minimum_r
0	39754872	Near North Side	73	1	0	0	0	
1	14382622	Woodlawn	52	1	0	0	0	
2	18769456	Woodlawn	52	1	0	0	0	
3	35459852	Douglas	51	1	0	0	0	

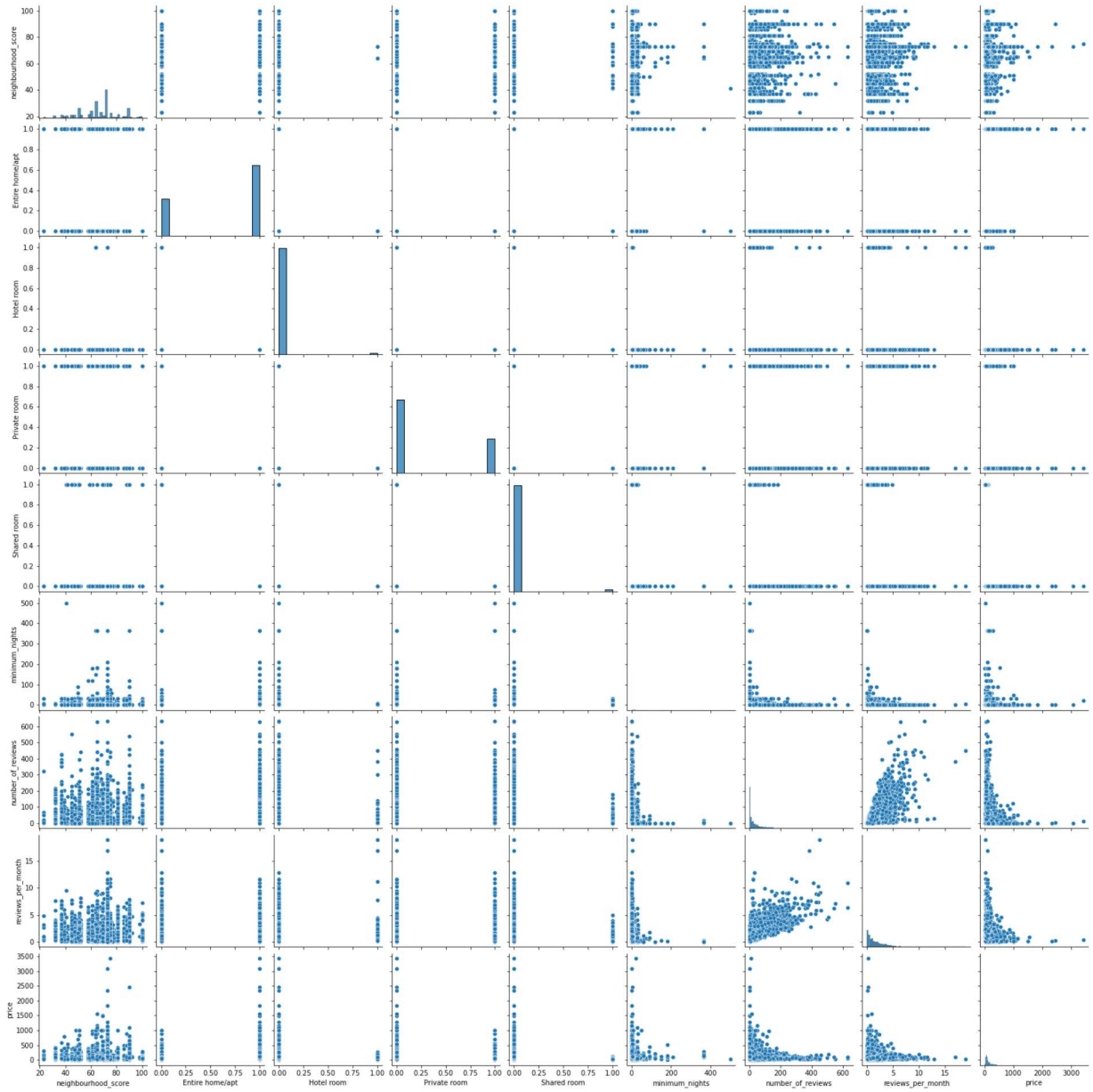
	id	neighbourhood	neighbourhood_score	Entire home/apt	Hotel room	Private room	Shared room	minimum_r
4	15867158	Near North Side	73	1	0	0	0	
...	...	...	...	...	...	...	...	
95	44433063	Lincoln Park	90	1	0	0	0	
96	40753273	Near North Side	73	0	0	1	0	
97	25481520	Jefferson Park	88	1	0	0	0	
98	41748592	South Shore	50	0	0	1	0	
99	43148081	Near North Side	73	1	0	0	0	

100 rows × 11 columns

Pair plots

In [4]:
```
# create pairs plot for all numeric variables to study the coorelations
# between the features and the label (price)
sns.pairplot(airbnb.drop(columns=['id']))
```

Out[4]:
```
<seaborn.axisgrid.PairGrid at 0x198670c9a00>
```



In [5]:
```
# There is a clear coorelation between the two features number_of_reviews and review
# We will not use one of them as it would be statisticall insignificant to keep them
```

In [5]:
```
# Scaling the price.
#airbnb['price'] = airbnb['price'].apply(lambda x: x/mean_price)
a = airbnb[airbnb['price'] > 500].index
airbnb.drop(a, inplace=True)
```

In [6]:
```
b = airbnb[airbnb['number_of_reviews'] == 0].index
airbnb.drop(b , inplace=True)
mean_price = airbnb["price"].mean()
airbnb['price'] = airbnb['price'].apply(lambda x: x/mean_price)
# create the feature list and the label list
X = airbnb.drop(columns = ['id', 'price', 'reviews_per_month','neighbourhood']).to_n
y = airbnb['price'].to_numpy()
```

Construction of the train, validation and test sets.

In [14]:
```
# Split the data into train and test sets.
X_model, X_test, y_model, y_test = train_test_split(X, y, test_size=0.15)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_model, y_model, test_size=0.25)
```

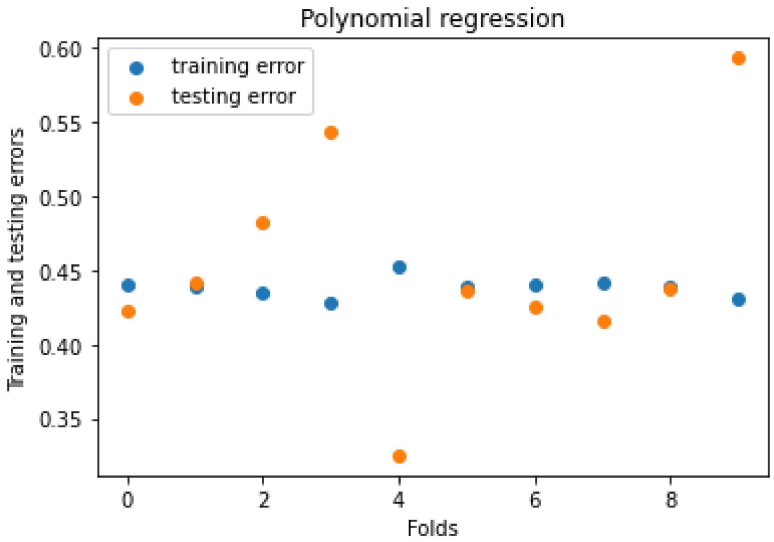
In [15]:
```
# Cross validation
def cross_validation(model):
    cv_results = cross_validate(model, X, y,
                                cv=10, scoring="neg_mean_squared_error",
                                return_train_score=True,
                                return_estimator=True)
    train_error = -cv_results["train_score"]
    tesr_error = -cv_results["test_score"]
    return train_error, tesr_error
```

ML methods

Method 1 : Polynomial regression

In [16]:
```
# Cross validation to test the model on diffrent sets
# we will use the Ridge regularization to prevent overfitting
linear_reg = make_pipeline(PolynomialFeatures(degree=2),Ridge(alpha=0.2))

sets = [0,1,2,3,4,5,6,7,8,9]
train_error, test_error = cross_validation(linear_reg)
plt.scatter(sets, train_error)
plt.scatter(sets, test_error)
plt.xlabel("Folds")
plt.ylabel("Training and testing errors")
plt.legend(["training error", "testing error"], loc ="upper left")
plt.title("Polynomial regression")
```

Out[16]:
```
Text(0.5, 1.0, 'Polynomial regression')
```



In [17]:
```
# polynomial regression
poly = PolynomialFeatures(2)
poly_features = poly.fit_transform(X_train)
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y_train)
y_pred_training = poly_reg_model.predict(poly_features)
y_pred_testing = poly_reg_model.predict(poly.fit_transform(X_val))
training_error = mean_squared_error(y_train, y_pred_training)
testing_error = mean_squared_error(y_val, y_pred_testing)
```

```
print("Training error: ", training_error)
print("Testing erro: ", testing_error)
```

```
Training error:  0.4452371786022417
Testing erro:  0.4105897124642374
```

The training and testing errors are low and very close to each other. Hence, polynomial regression (degree 2) is a good model.

Method 2 : MLP (Multi-layer Perceptron) Regression

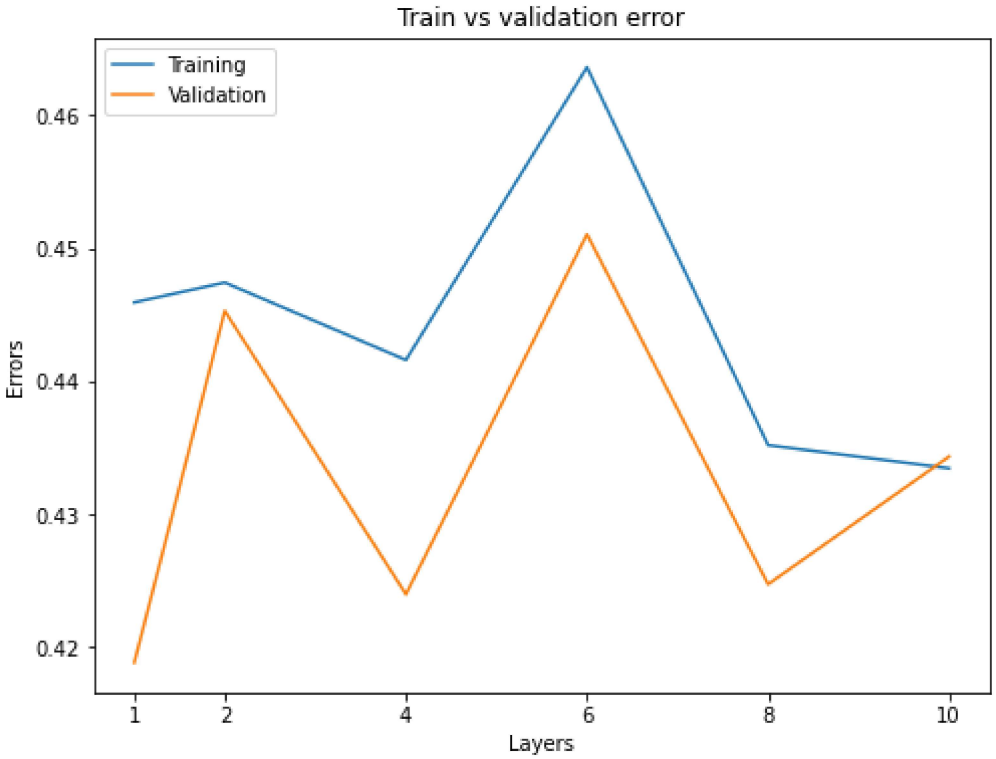
First, Let us choose the number of layers.

In [18]:

```
layers = [1,2,4,6,8,10]    # number of hidden layers
num_neurons = 15  # number of neurons in each layer

tr_errors_deep = []
val_errors_deep = []

for i, j in enumerate(layers):
    hidden_layer_sizes = tuple([num_neurons]*j)
    deep_reg = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,random_state=42, m
    deep_reg.fit(X_train, y_train)

    y_pred_train = deep_reg.predict(X_train)
    y_pred_val = deep_reg.predict(X_val)

    train_error = mean_squared_error(y_train, y_pred_train)
    val_error = mean_squared_error(y_val, y_pred_val)

    tr_errors_deep.append(train_error)
    val_errors_deep.append(val_error)
```

In [19]:

```
plt.figure(figsize=(8, 6))

plt.plot(layers, tr_errors_deep, label = 'Training')
plt.plot(layers, val_errors_deep,label = 'Validation')
plt.xticks(layers)
plt.legend(loc = 'upper left')

plt.xlabel('Layers')
plt.ylabel('Errors')
plt.title('Train vs validation error')
plt.show()
```



In [20]:

```
# create a table to compare the training and validation errors for MLPs with differe
errors = {"num_hidden_layers":layers,
          "mlp_train_errors":tr_errors_deep,
          "mlp_val_errors":val_errors_deep,
          "diffrence of errors": abs(np.array(tr_errors_deep) - np.array(val_errors_
         }
pd.DataFrame(errors)
```

Out[20]:

	num_hidden_layers	mlp_train_errors	mlp_val_errors	diffrence of errors
0	1	0.445917	0.418820	0.027097
1	2	0.447410	0.445271	0.002139
2	4	0.441586	0.423955	0.017631
3	6	0.463610	0.451040	0.012569
4	8	0.435178	0.424708	0.010471
5	10	0.433435	0.434319	0.000884

we choose layer = 2

In [21]:

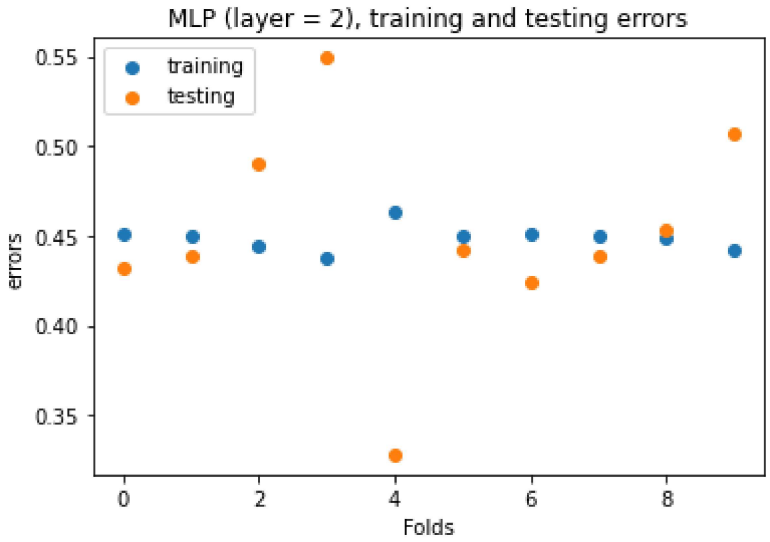
```
# Cross validation to test the second model on diffrent sets
mlp_deep_regr = make_pipeline(MLPRegressor(hidden_layer_sizes=2,random_state=42, max

train_mlp, test_mlp = cross_validation(mlp_deep_regr)
```

In [22]:

```
plt.scatter(sets, train_mlp)
plt.scatter(sets, test_mlp)
plt.xlabel("Folds")
plt.ylabel("errors")
plt.legend(["training", "testing"], loc ="upper left")
plt.title("MLP (layer = 2), training and testing errors")
```

```
Text(0.5, 1.0, 'MLP (layer = 2), training and testing errors')
```

Out[22]:

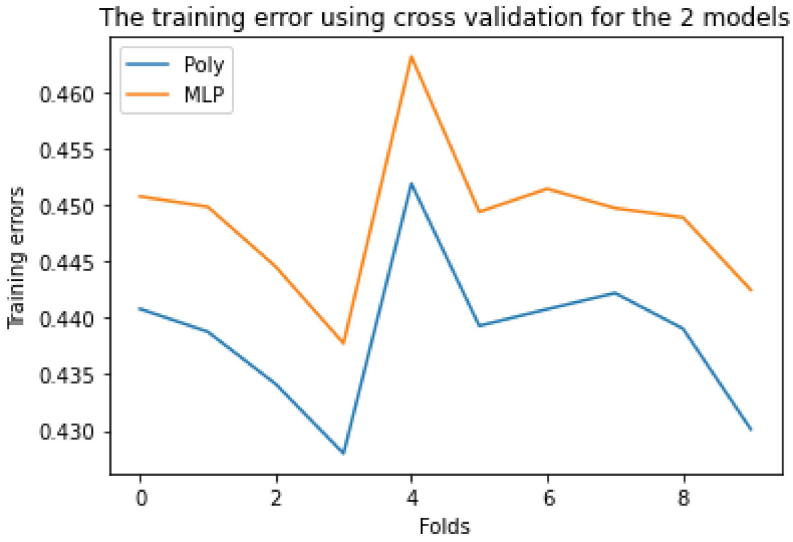


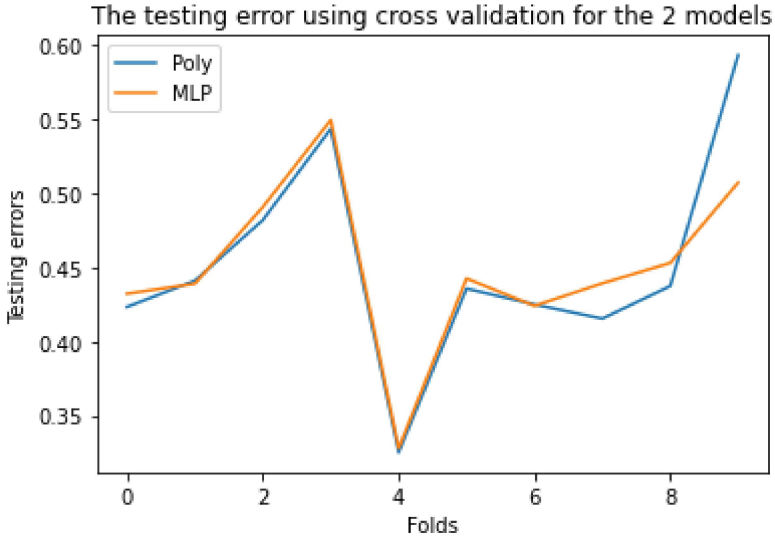
In some cases, the training error is bigger tha the tesing error. This can be explained by the fact that the model handles for this particular split od data "hard cases" to predict in the training set, while having some "Soft cases" in the testing set.

Comparing the two models

In [19]:

```
# Plot the training and testing errors for both models

mlp_train_errors, mlp_test_errors = cross_validation(mlp_deep_regr)
poly_train_errors, poly_test_errors = cross_validation(linear_reg)

sets = [0,1,2,3,4,5,6,7,8,9]
plt.plot(sets, poly_train_errors)
plt.plot(sets, mlp_train_errors)
plt.xlabel("Folds")
plt.ylabel("Training errors")
plt.legend(["Poly", "MLP"], loc ="upper left")
plt.title("The training error using cross validation for the 2 models")
plt.show()

plt.plot(sets, poly_test_errors)
plt.plot(sets, mlp_test_errors)
plt.xlabel("Folds")
plt.ylabel("Testing errors")
plt.legend(["Poly", "MLP"], loc ="upper left")
plt.title("The testing error using cross validation for the 2 models")
plt.show()
```





time complexity

In [18]:

```
# Compute the time it takes for both models.

t1_mlp = timeit.default_timer()
mlp = MLPRegressor(hidden_layer_sizes=2,random_state=42, max_iter=1000)
mlp.fit(X_train, y_train)
t2_mlp = timeit.default_timer()


t1_poly = timeit.default_timer()
poly = PolynomialFeatures(2)
poly_features = poly.fit_transform(X_train)
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y_train)
t2_poly = timeit.default_timer()


print('time of execution for the mlp model ', t2_mlp - t1_mlp)
print('time of execution for the plonomial model ', t2_poly - t1_poly)
```

```
time of execution for the mlp model  2.2433163999999977
time of execution for the plonomial model  0.0033956999999986692
```

Akaike's Information Criterion (AIC) by (Akaike 1969)

In [47]:

```
def AIC(nbr_of_params,ntrain,MSE):
    return ntrain*np.log(MSE) + 2*nbr_of_params

poly = PolynomialFeatures(2)
poly_features = poly.fit_transform(X_train)
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y_train)
y_pred_training = poly_reg_model.predict(poly_features)
MSE_poly = mean_squared_error(y_train, y_pred_training)


mlp = MLPRegressor(hidden_layer_sizes=2,random_state=42, max_iter=1000)
mlp.fit(X_train, y_train)
y_pred_training = mlp.predict(X_train)
MSE_mlp = mean_squared_error(y_train, y_pred_training)
mlp_params = (len(X_train)*15 + 15*15 + 15*1) + (15 + 15 + 1) # 15 is the number of

# the polynomial model
print("the AIC for the mlp model is :", AIC(mlp_params,len(X_train), MSE_mlp))
print("the AIC for the polynomial model is :", AIC(3,len(X_train), MSE_poly))
```

```
the AIC for the mlp model is : 71526.13551323663
the AIC for the polynomial model is : -2032.5138724875526
```

Testing the better model on the test set.

In [23]:
```
#
poly = PolynomialFeatures(2)
poly_features = poly.fit_transform(X_test)
poly_reg_model = LinearRegression()
poly_reg_model.fit(poly_features, y_test)
y_pred_test = poly_reg_model.predict(poly_features)
MSE_poly = mean_squared_error(y_test, y_pred_test)
```

In [24]:
```
MSE_poly
```

Out[24]:
```
0.4368053210067403
```

In [ ]: