Coursera IBM Data Science Certification Final Project - Amsterdam Housing Market

Martino Pitruzzella

April 13, 2021

Abstract

We run linear regression models on the Amsterdam housing data, we then try to improve the models using venues location data we extract from the Foursquare API, in particular we run a clustering algorithm on the Foursquare data and use the results in the linear regression model. We try two ways to improve the r2 score of the linear regression model and find out they both give significant improvement.

Keywords: Linear regression, Clustering, Housing data, Amsterdam, Foursquare

Contents

1 Introduction

1.1 Background

Amsterdam is the capital of the Netherlands, It is a well known tourist destination, receiving more than 40 millions visitors a year, with a large number of cultural attractions per capita. Amsterdam is also well known for its rising housing prices: since 2015, prices have steadily increased every year, with an increase of about 80% between 2014 and 2020. In this capstone project we will explore the Amsterdam housing market, with a particular focus on apartment prices. We will use linear regression models on the Amsterdam housing data to predict the price from house features such as surface area.

1.2 Business problem

We will then combine the house price dataset with Foursquare location data, setting out to answer the following business question: Can we improve house price predictions using Foursquare location data?. In answering this question we will run a clustering algorithm on Foursquare location data to distinguish areas with similar features. We then use Amsterdam's postcodes and try to use the cluster-labels to

improve the linear regression prediction. We then use the Foursquare data in other ways to try to reach the same goal, namely, we will run the linear regression on the main cluster only, give that we will find out there is one cluster significantly bigger than the others.

1.3 Interest

Whether you are a stakeholder looking to invest in the Amsterdam housing market, a single buyer looking to check if an asking price is convenient or not or seller looking to set a competitive yet rewardable price for the house you want to sell, the results of this project will surely be of interest to you.

2 Data

2.1 Data sources

Amsterdam is administratively divided into 8 large districts or boroughs. Each district is then subdivided into neighbourhoods, with 107 neighbourhoods in total. Alternatively, Amsterdam can be divided into postcode areas. Due to the difficulty to find consistent data on the neighbourhoods, we have decided to run a clustering algorithm on the Foursquare data using the postcode areas as geolocations. We use 72 postcode areas, and for each postcode we extracted its approximate centre location from Google maps. Due to its geographically separate location we have excluded Amsterdam Zuid-Oost from our study.

For the housing market data, we have gathered and used data from funda.nl, the main website for housing in the Netherlands and association of housing agents. As funda.nl has a protection mechanism and it is not allowed to scrape data from the site, we have used only the data present in the main pages for the Amsterdam listing, this includes only: address, postal code, m2 of living area, number of rooms and price. We will see that even with this limitation, we are still able to make reasonable predictions with this limited feature count.

At the time of data collection (end December 2020) there were about 2500 apartments for sale in Amsterdam and about 500 houses. Since houses have an additional feature (plot size) and apartments do not, we have decided to focus on apartment data only. We have then combined funda dataset with the Foursquare one in order to try to answer the business question and find out if it is possible to improve the prediction by combining the housing data with the Foursquare data.

2.2 Data cleaning

After having parsed the housing data from the html pages and populated a pandas dataframe with it we proceed with the data cleaning. We have 2859 items and 6 features:

```
In [6]: df_houses_raw.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2859 entries, 0 to 2858
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Address      2859 non-null   object
 1   Postcode     2859 non-null   object
 2   Living_area  2859 non-null   int64 
 3   Plot_size    2859 non-null   int64 
 4   Rooms        2859 non-null   int64 
 5   Price        2859 non-null   int64 
dtypes: int64(4), object(2)
memory usage: 134.1+ KB
```

We remove the items where the price is not available and remove 4 duplicates. We then create a feature-limited dataframe for modeling purposes. We are left with 2844 items and 3 features:

In [11]:
```
cdf = df_houses[['Living_area','Rooms','Price']]
cdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2844 entries, 0 to 2858
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Living_area  2844 non-null   int64
 1   Rooms        2844 non-null   int64
 2   Price        2844 non-null   int64
dtypes: int64(3)
memory usage: 88.9 KB
```

Regarding the Foursquare data, we followed along the lines of the previous project in this final course of the certification and will not need to make much data cleaning.

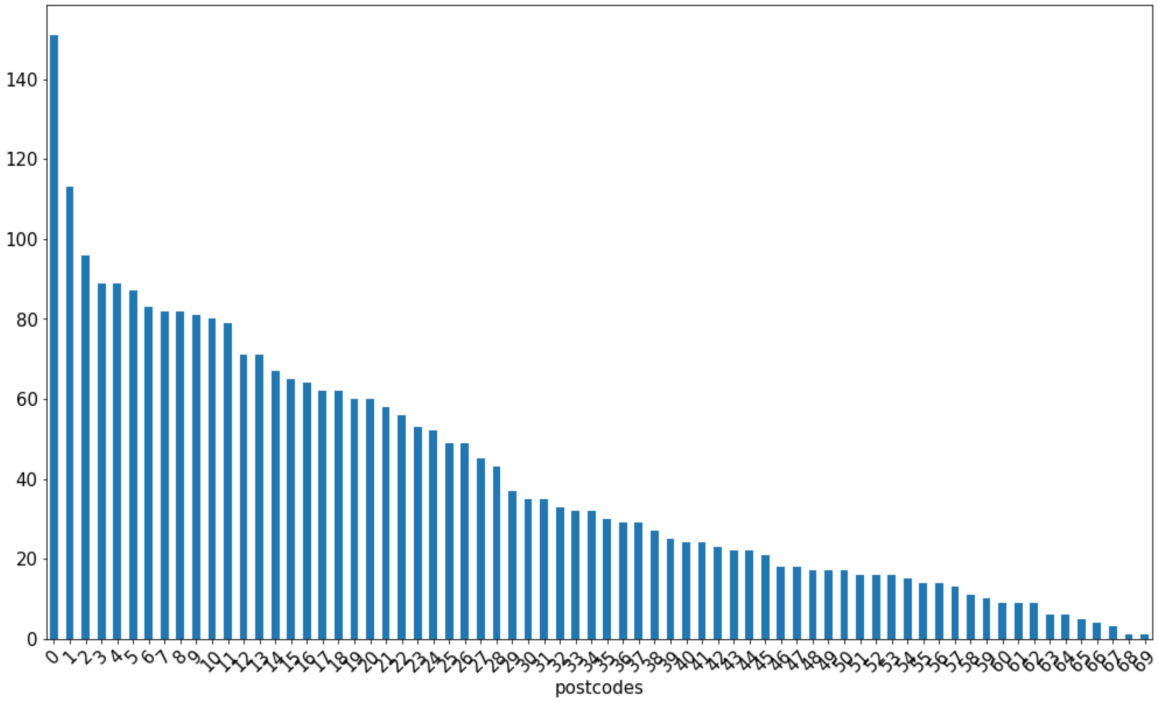
2.3 Exploratory data analysis and features engineering

We start by exploring the housing data with some summary statistics. We can see that the mean living area is approximately 100m2, going from as low as 10m2 to as high as 500m2. The number of rooms ranges from 1 to 16.

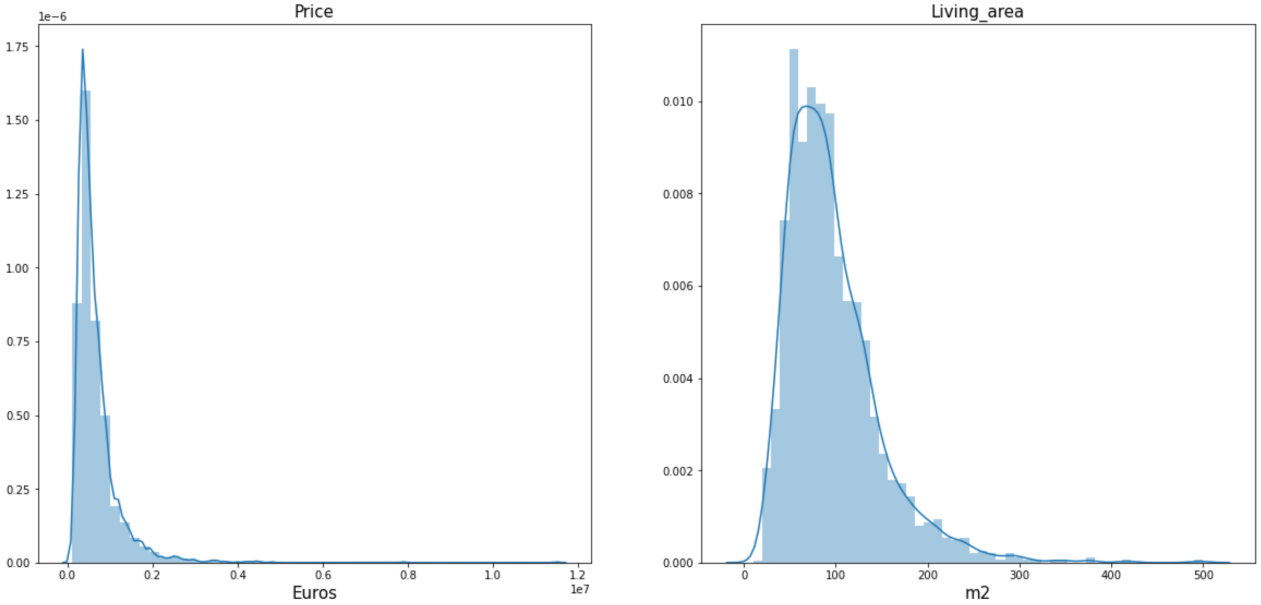
Out[12]:

	Living_area	Rooms	Price
count	2844.0	2844.0	2844.0
mean	97.0	3.0	663093.0
std	51.0	1.0	547083.0
min	10.0	1.0	100000.0
25%	60.0	3.0	350000.0
50%	86.0	3.0	500000.0
75%	120.0	4.0	775947.0
max	500.0	16.0	11500000.0

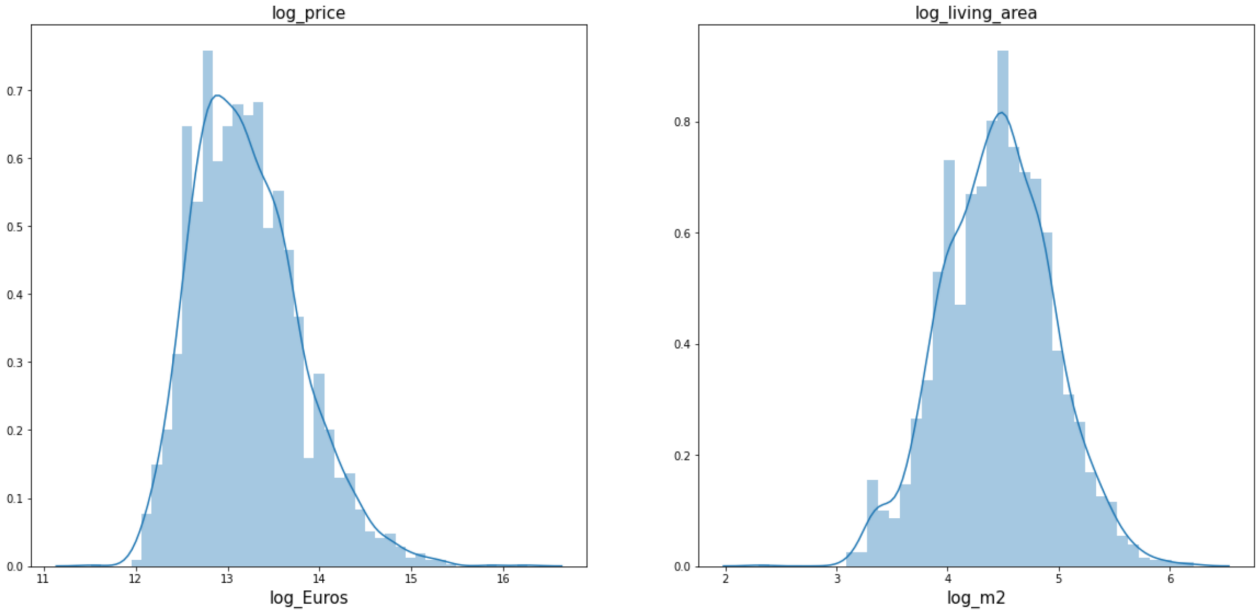
With the following graphics we inspect the dataframe. In particular, we plot the number of apartments for sale for each postcode in descending order. From this we can see that the data has a nice power-law shape. Two postcodes had as many as 151 and 113 apartments for sale (the area near to the ARTIS zoo and the area in Amsterdam Noord close the Eye Filmmuseum). We notice that we have 70 different postcodes represented in the data set, so almost all of the 72 in total in Amsterdam.



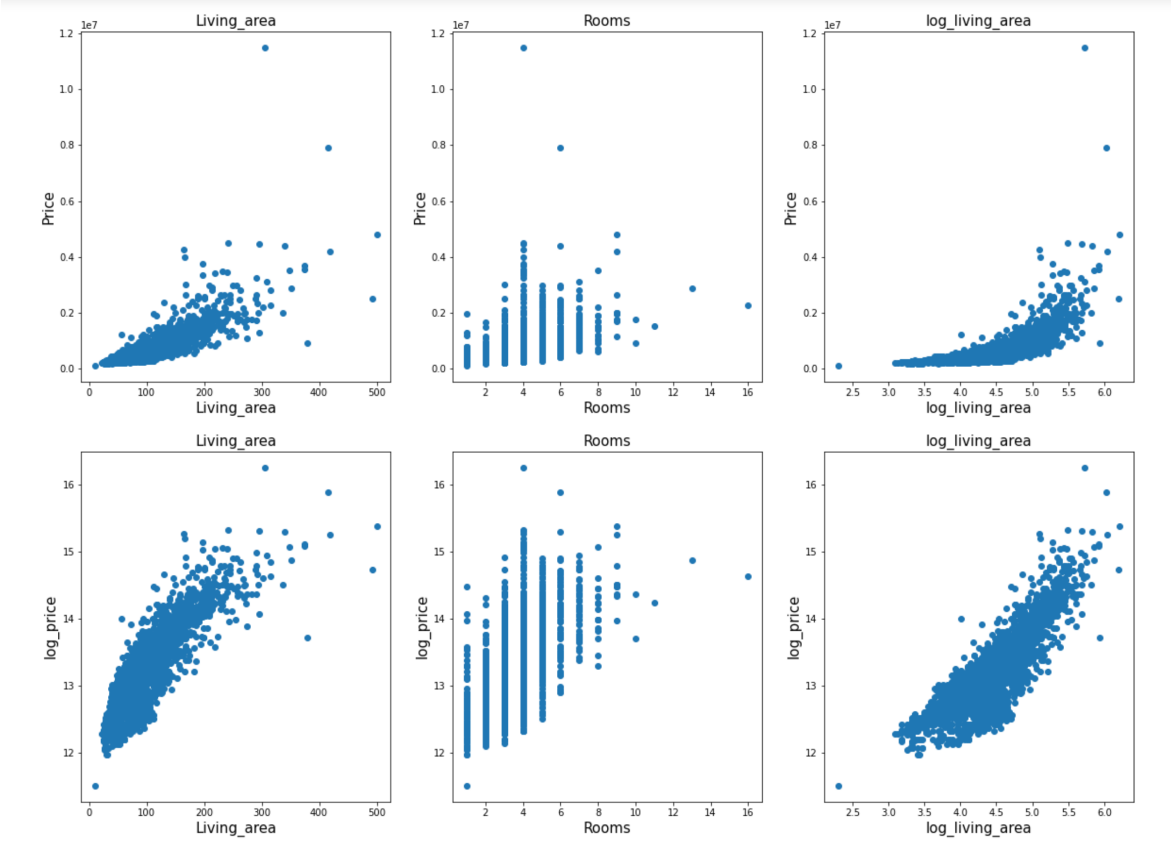
Next, we explore the features Living area and Price, by inspecting the distribution plot of each feature we can see that both features have a highly left-skewed distribution.



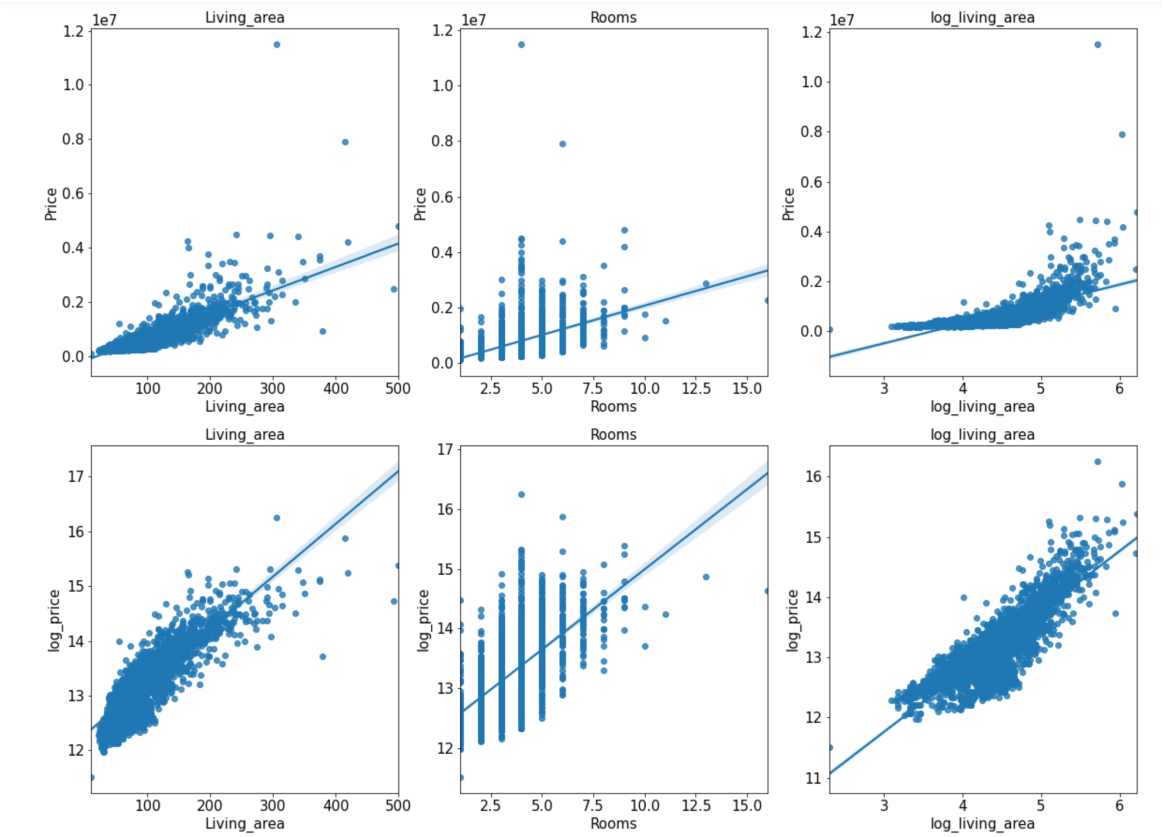
Since the algorithm prefers more normally distributed features, we log-transform these two. We can see that the distributions of the transformed features are now roughly normal, we will keep the original features for the data exploration but discard them in the model.



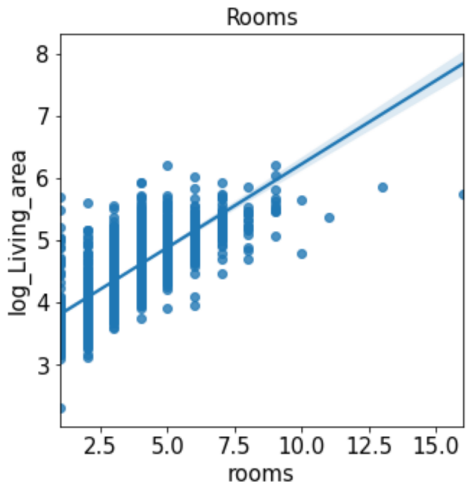
Next we scatter plot the data. Since we want to predict the log price we plot the log price against the log living area and the log price against the number of rooms. As expected from previous research on this kind of problem, we can see that there is a roughly linear relationship between both: log living area and log price and number of rooms and log price. Also we notice that there are some outliers, both in the price, in the living area and in the number of rooms.



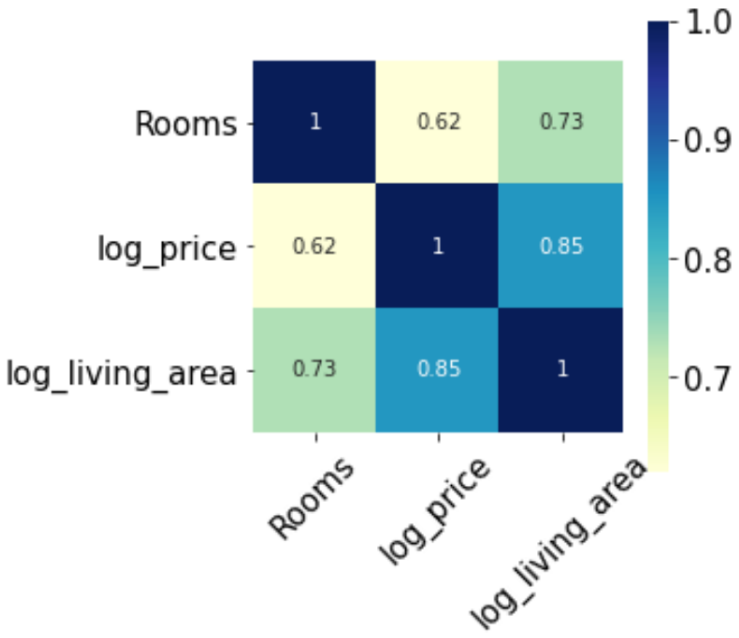
We further inspect the linear dependancy by making a seaborn regression plot:



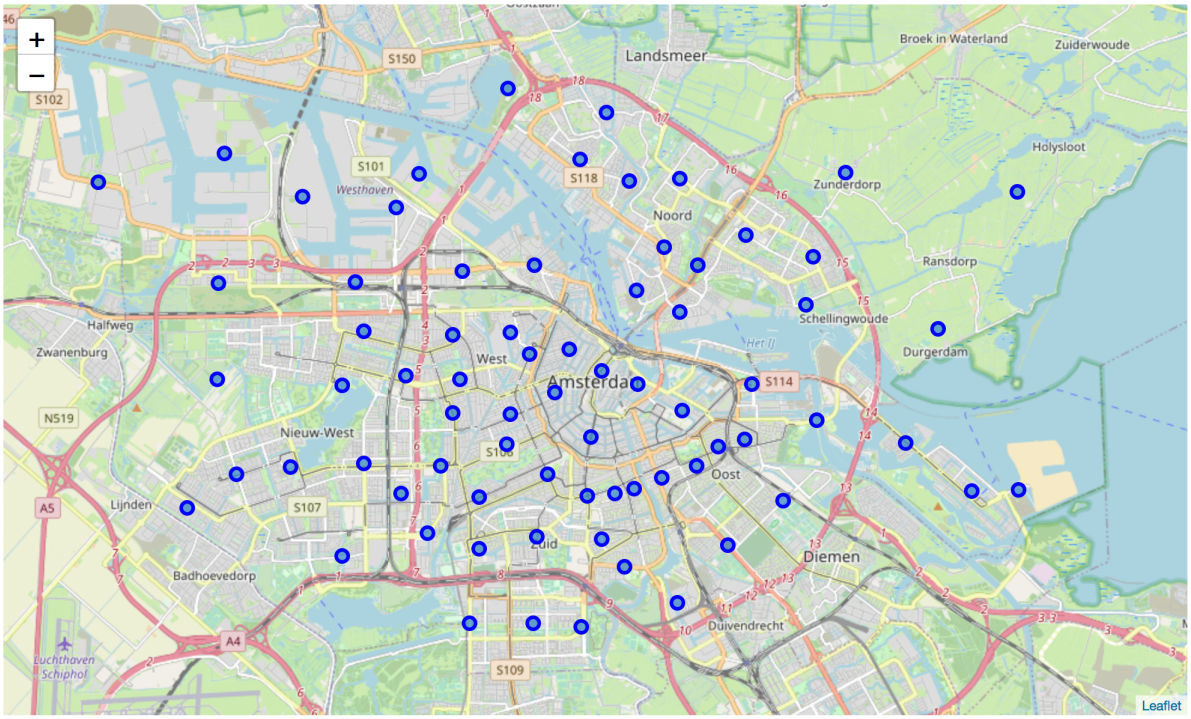
Since we want to run a linear regression we want to make sure there are no features mutually linearly dependent, so we make a regression plot of the number of rooms and the log living area. We can see that there seem to be some kind of linearly dependency between the two features.



This is further confirmed by examining the correlation coefficients, we can see that the highest correlation is between log living area and log price (0.85), rooms has also a good correlation with log price (0.62) but the two features have a notable correaltion between themselves (0.73). We expect that we will want to remove the rooms feature from the regression model.



Next we consider the Foursquare data. We need to use the coordinates of the approximate center of each postcode area, since they are only 72 areas we have gathered the coordinates manually from google maps instead of using the google API, and saved them in a cvs file. We then display the coordinate points in a Amsterdam map with the use of folium library.

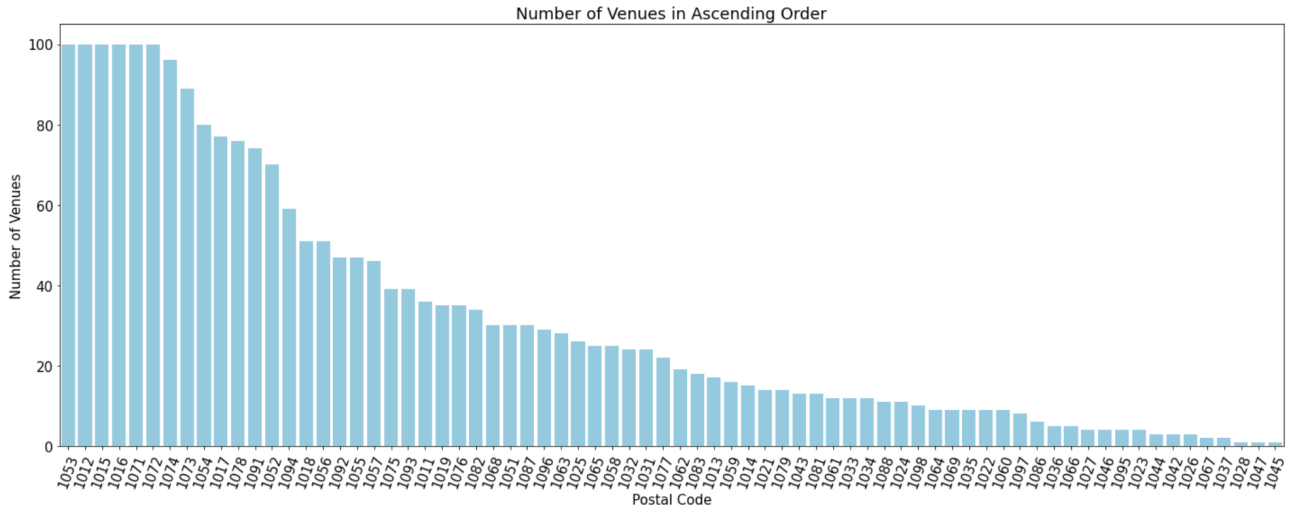


We then extract the venues of Amsterdam per postcode area from the Foursquare API with the use of the two functions provided by the course. The result is a dataframe with 2266 entries and 7 columns. We can see there are 281 unique venue type categories.

```
In [42]: Amsterdam_venues.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2266 entries, 0 to 2265
Data columns (total 7 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Postal Code                 2266 non-null   int64
 1   Postal code area Latitude   2266 non-null   float64
 2   Postal code area Longitude  2266 non-null   float64
 3   Venue                       2266 non-null   object
 4   Venue Latitude              2266 non-null   float64
 5   Venue Longitude             2266 non-null   float64
 6   Venue Category              2266 non-null   object
dtypes: float64(4), int64(1), object(2)
memory usage: 124.0+ KB
```

Next we plot the number of venues per post code area in descending order, we can see that for 6 areas we have attained the maximum number of venues of 100 allowed by the API, also in this case we see we have some kind of power law shape.



3 Methodology and Analysis

3.1 Linear regression on the housing data

We now build three linear regression models, one with each of the two features and one with both and compare the results. The first step is to split the data into train and test set.

```
In [24]: X = cdf[['log_living_area','Rooms']]

         y = cdf['log_price']

         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)
```

```
Train set: (2275, 2) (2275,)
Test set: (569, 2) (569,)
```

Next we built and fit the three models.

In [25]:
```
from sklearn import linear_model
regr1 = linear_model.LinearRegression()
regr2 = linear_model.LinearRegression()
regr3 = linear_model.LinearRegression()
regr1.fit(X_train[['log_living_area']], y_train)
regr2.fit(X_train[['Rooms']], y_train)
regr3.fit(X_train, y_train)
```

And we compare their mean squared errors and r2 scores. We can see that, as expected, the regression on rooms feature alone performs badly, also there is no improvement in including this feature next to the log living area feature. The best model is indeed the one on the log living area feature only, which has a r2 score of 0.7294 in what follows we will exclude the feature rooms from the models.

In [26]:
```
y_pred1 = regr1.predict(X_test[['log_living_area']])
y_pred2 = regr2.predict(X_test[['Rooms']])
y_pred3 = regr3.predict(X_test)

from sklearn.metrics import r2_score

print('Mean squared error model 1: %.4f' % np.mean((y_pred1 - y_test) ** 2))

print('r2 score model 1 : %.4f' % r2_score(y_test,y_pred1))

print('Mean squared error model 2: %.4f' % np.mean((y_pred2 - y_test) ** 2))

print('r2 score model 2: %.4f' % r2_score(y_test,y_pred2))

print('Mean squared error model 3: %.4f' % np.mean((y_pred3 - y_test) ** 2))

print('r2 score model 3: %.4f' % r2_score(y_test,y_pred3))
```

```
Mean squared error model 1: 0.0889
r2 score model 1 : 0.7294
Mean squared error model 2: 0.1904
r2 score model 2: 0.4205
Mean squared error model 3: 0.0889
r2 score model 3: 0.7293
```

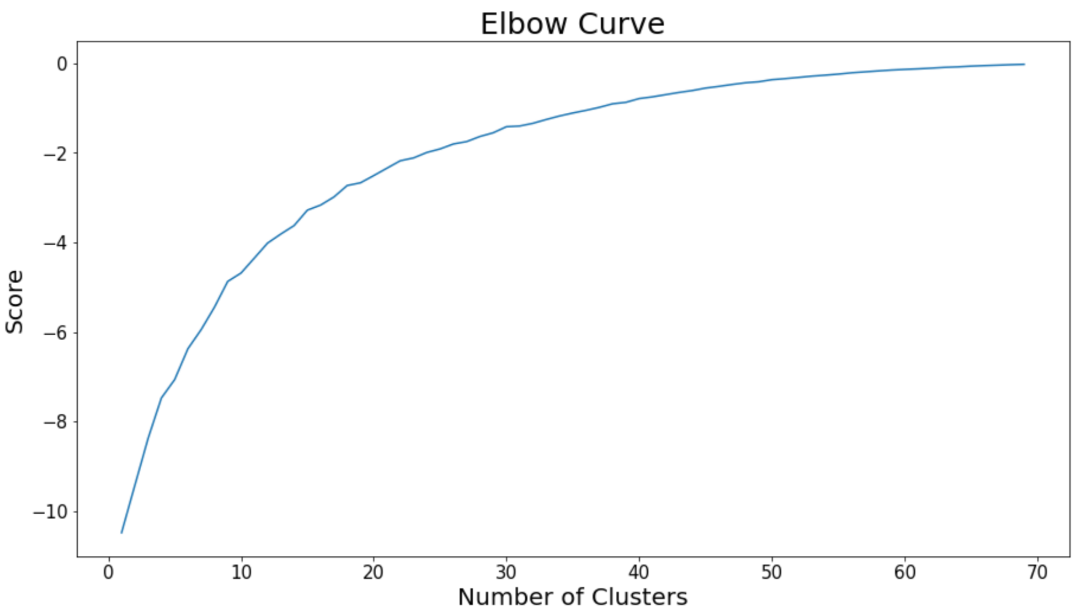
Lastly we draw a predicted versus actual prices plot, we can see that the model performs very good on roughly the first half of the range but, as commonly seen, fails to predict properly the price of the outliers.

3.2 Clustering on the Foursquare data

Next, In this section, we run a kmeans clustering algorithm the data that we have gathered from the Foursquare API. We first need to prepare the data for so that it can be fed to the algorithm, for this we follow along the steps of the previous project in this course. The first step is to one-hot-encode the categories. Next, we exctract the 10 top venues and their relative frequencies in the dataset. For example, for the first postcode area we have the following top 10 categories.

```
----1011----
                      venue  freq
0                    Hostel  0.10
1                     Hotel  0.10
2                       Bar  0.08
3             Grocery Store  0.05
4               Coffee Shop  0.05
5               Cocktail Bar  0.05
6            Breakfast Spot  0.03
7  Bike Rental / Bike Share  0.03
8         Convenience Store  0.03
9  Mediterranean Restaurant  0.03
```

We are now ready to run the kmeans clustering algorithm, in order to do this, we need to choose the number of clusters, for this reason we run the algorithm with the number of clusters in the range from 1 to 70 and inspect the elbow curve.

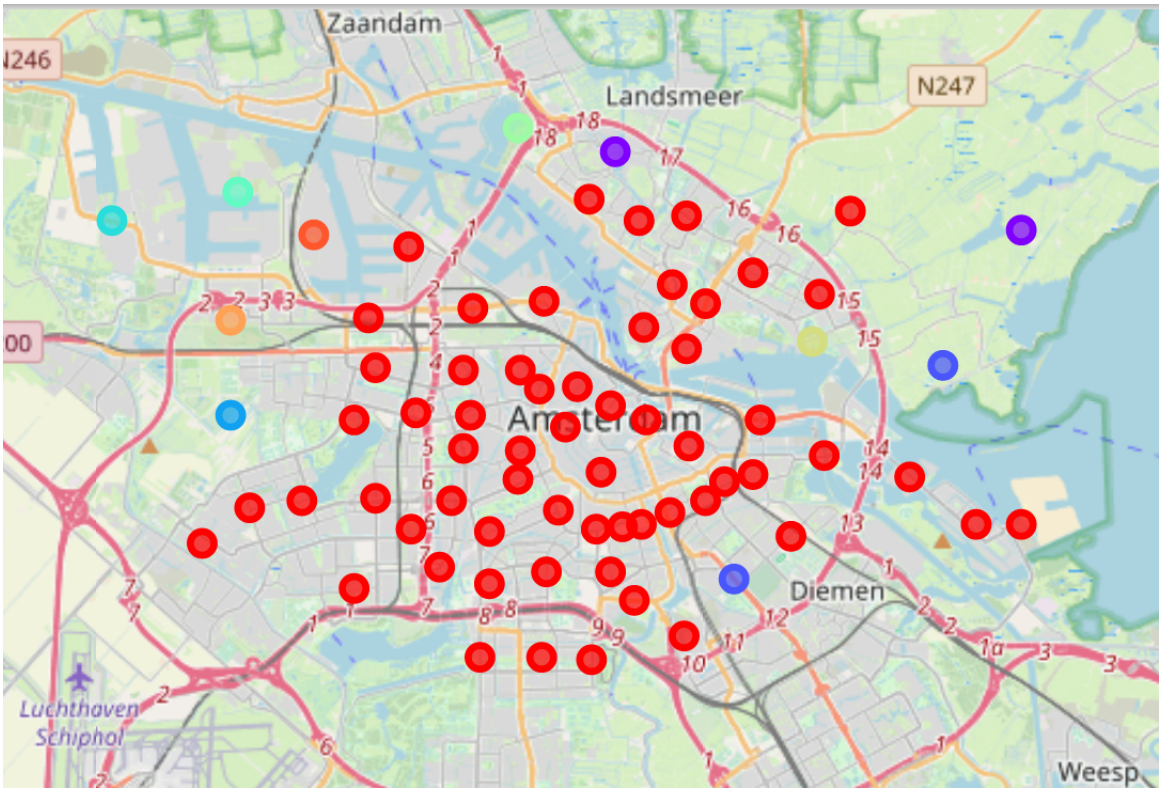


We can see that the elbow curve does not have a clear cusp point, this doesn't mean that the data can not be clustered, it means that we need to be more careful in choosing the right number of clusters. We will at first make an attempt with 10 clusters and see if our linear regression model improves with this number of clusters, after that we will further explore all the range of cluster numbers. We can see that when we divide in 10 clusters most of them fall in one main big cluster.

```
# run k-means clustering
kclusters = 10
kmeans10 = KMeans(n_clusters=kclusters, random_state=0).fit(Amsterdam_grouped_clustering)

# check cluster labels generated for each row in the dataframe
A = kmeans10.labels_[0:]

print(sorted(A))
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 3, 4, 5, 6, 7, 8, 9]
```

After some further cleaning and rearranging the data we display the map with the post code areas belonging to the different clusters.



We can see that the clustering algorithm has done a great job in differentiating between areas in the center and areas in the outskirts of town, this is noticeable because the clustering is based only on the venues data and not on geographical data.

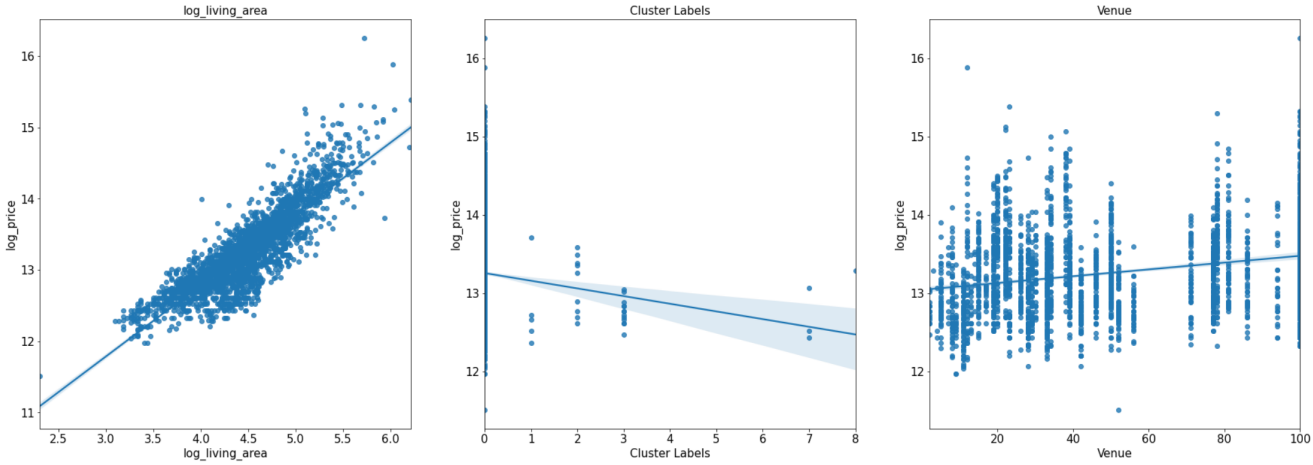
3.3 Linear regression on the combined data

In this section we finally merge the two datasets and see if we can improve our linear regression housing price prediction by using the two datasets jointly. Clearly we do not expect to be able to use the cluster labels directly in the regression as the actual cluster label numbers are assigned randomly by the algorithm.

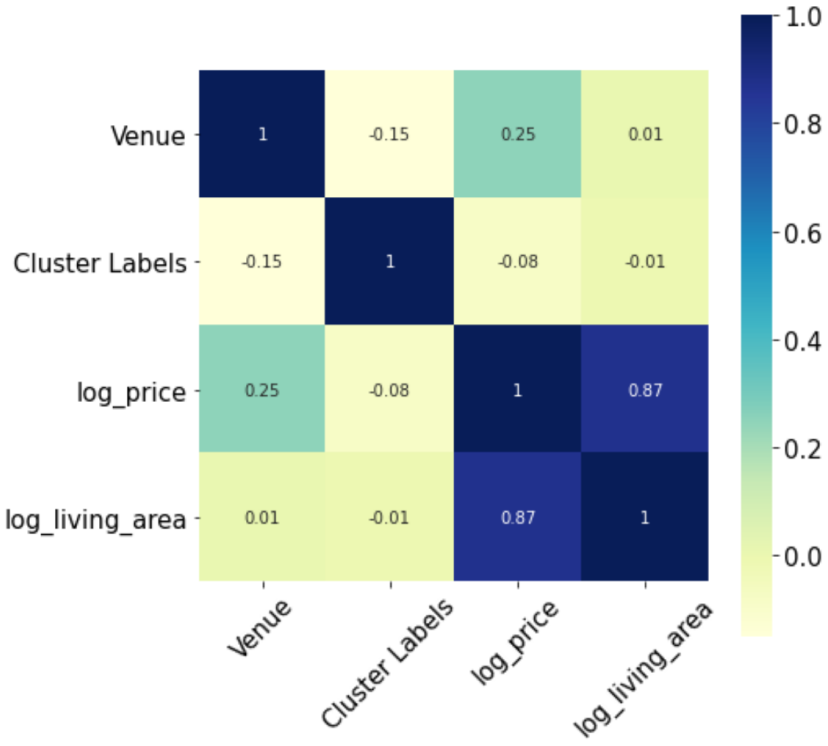
```
In [70]: Amsterdam_combined.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2733 entries, 0 to 2732
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Postcode         2733 non-null   int64  
 1   Venue            2733 non-null   int64  
 2   Cluster Labels   2733 non-null   float64
 3   log_price        2733 non-null   float64
 4   log_living_area  2733 non-null   float64
dtypes: float64(3), int64(2)
memory usage: 128.1 KB
```

We explore the data with a seaborn regression plot.



And we inspect the correlation coefficients, as expected, cluster labels have little correlation with the log price while the number of venues has a better score. We can see that the feature number of venues has a not very high but not negligible correlation with price, we notice also that the number of venues has almost no correlation with the living space (not surprisingly!). It is probably this combination that will allow the number of venues feature to improve the model. We remind that the number of venues comes solely from the Foursquare data and not from the clustering algorithm.



Then we run the linear regressions with the combined data.

```
In [82]: X = Amsterdam_combined[['log_living_area','Venue','Cluster Labels']]

         y = Amsterdam_combined['log_price']

         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
         print ('Train set:', X_train.shape,  y_train.shape)
         print ('Test set:', X_test.shape,  y_test.shape)

         Train set: (2092, 3) (2092,)
         Test set: (523, 3) (523,)
```

And create 5 different models:

```
In [84]: from sklearn import linear_model
         regr0 = linear_model.LinearRegression()
         regr1 = linear_model.LinearRegression()
         regr2 = linear_model.LinearRegression()
         regr3 = linear_model.LinearRegression()
         regr4 = linear_model.LinearRegression()
         regr5 = linear_model.LinearRegression()

         regr0.fit(X_train[['log_living_area']], y_train)
         regr1.fit(X_train[['log_living_area','Venue']], y_train)
         regr2.fit(X_train[['log_living_area','Cluster Labels']], y_train)
         regr3.fit(X_train[['Cluster Labels']], y_train)
         regr4.fit(X_train[['Venue']], y_train)
         regr5.fit(X_train, y_train)
```

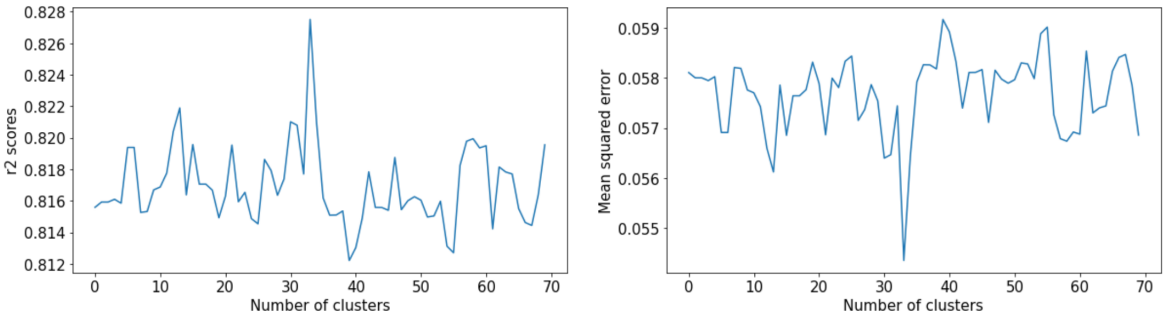
We can see that we have a significant improvement in the r2 score when we make the regression on the features living area and number of venues only, from about 0.74 to about 0.80, while, as expected, there is little improvement when further adding the cluster labels.

```
r2 score model 0 : 0.7450
Residual sum of squares model 1: 0.0863

r2 score model 1 : 0.8049
Residual sum of squares model 1: 0.0660

r2 score model 2: 0.7488
Residual sum of squares model 2: 0.0850

r2 score model 3: -0.0345
Residual sum of squares model 3: 0.3500

r2 score model 4: 0.0436
Residual sum of squares model 4: 0.3236

r2 score model 5: 0.8056
Residual sum of squares model 5: 0.0658
```

For completeness we check the p values and statistical significance of the features in the regression, we can see that the p values are all very small but they also indicate that the features to be selected are definitely living area and venue.

```
In [86]: import statsmodels.api as sm
         mod = sm.OLS(y_train,X_train)
         fii = mod.fit()
         p_values = fii.summary2().tables[1]['P>|t|']
         p_values

Out[86]: log_living_area    0.000000e+00
         Venue              7.403189e-44
         Cluster Labels     4.481706e-01
         Name: P>|t|, dtype: float64
```

We have made a regression for the combined data in the case of 10 clusters, we want now to explore the dependancy of the r2 score and of the mean squared error on the number of clusters. We can see that as expected there is almost no variation in the r2 scores and error with respect to the number of clusters.

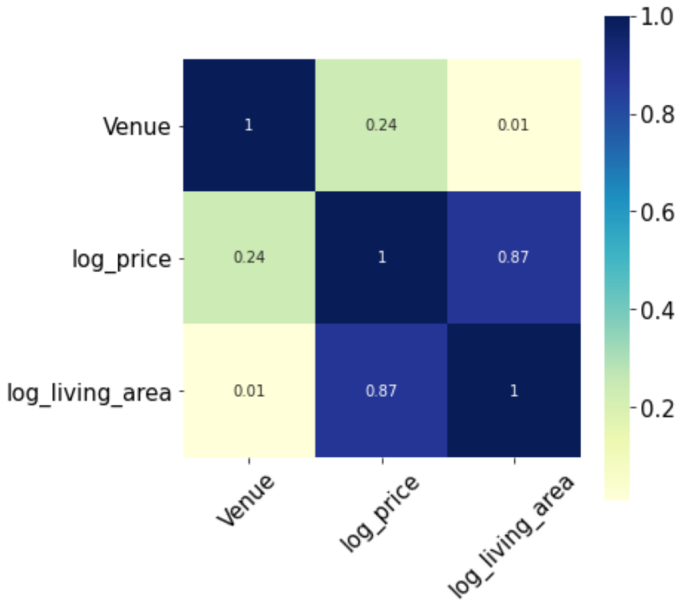


3.4 Linear regression on the main cluster

In this section we try a different approach and try to use the cluster labels in a different way: we run the linear regression on the bigger cluster only, we can expect to be able to make a better estimate at the expense of removing some data and use the data in that specific cluster only. We will see that this will increase the r2 score and given that the number of clusters is not too high we will not have to discard too much data. We will try this in the case of 10 clusters first, and then we explore all the range of clusters. We start by checking the size of the clusters we have in the case of 10 clusters, we notice that most entries fall into one cluster only, we also notice that during the merging with the clusters were reduced to 5 (probably because few areas had no houses for sell).

```
In [93]: n_by_cluster

Out[93]: Cluster Labels
         0.0    2583
         4.0      13
         6.0       9
         7.0       3
         8.0       9
```

We then extract the label of the main cluster and check the correlation coefficients on the main cluster. We can see that the correlation coefficients have not changed much if at all.



And run the linear regression on the main cluster. We can see that the r2 score of the regression on the living area feature alone has improved from around 0.76 to around 0.77, however if we add the number of venues feature we have a better r2 score compared to the first attempt we made, the improvement is approximately 0.02, from about 0.80 to 0.82.

In [102]:
```
y_pred3 = regr3.predict(X_test[['log_living_area']])
y_pred4 = regr4.predict(X_test[['log_living_area','Venue']])

from sklearn.metrics import r2_score

print('r2 score model 3: %.4f' % r2_score(y_test,y_pred3))
print('Residual sum of squares model 3: %.4f' % np.mean((y_pred3 - y_test) ** 2))
print('')

print('r2 score model 4: %.4f' % r2_score(y_test,y_pred4))
print('Residual sum of squares model 4: %.4f' % np.mean((y_pred4 - y_test) ** 2))
print('')
```

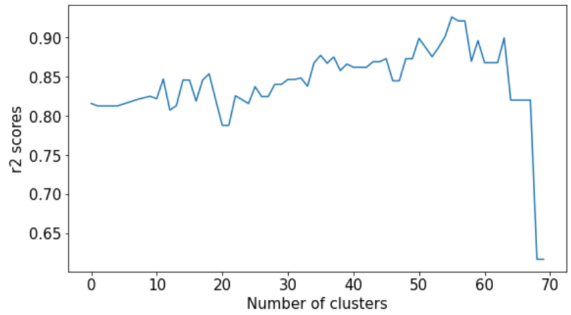
```
r2 score model 3: 0.7691
Residual sum of squares model 3: 0.0830

r2 score model 4: 0.8257
Residual sum of squares model 4: 0.0626
```

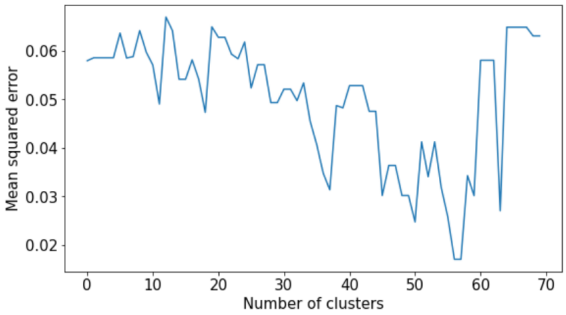
For completeness we check again the p values.

In [103]:
```
import statsmodels.api as sm
mod = sm.OLS(y_train,X_train)
fii = mod.fit()
p_values = fii.summary2().tables[1]['P>|t|']
p_values
```

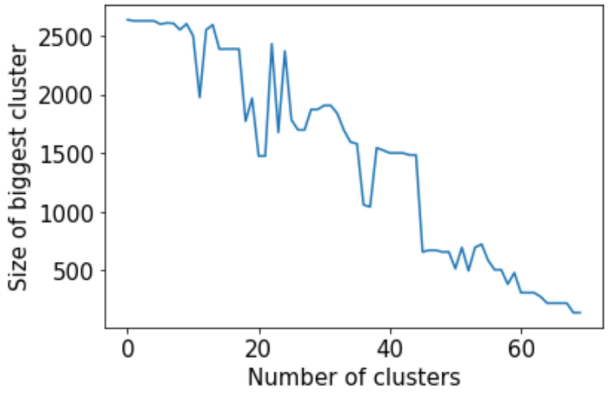
Out[103]:
```
log_living_area    0.000000e+00
Venue              2.729322e-41
Name: P>|t|, dtype: float64
```

Lastly we inspect the dependency of the r2 score for the regression on the main cluster respect to the number of clusters.

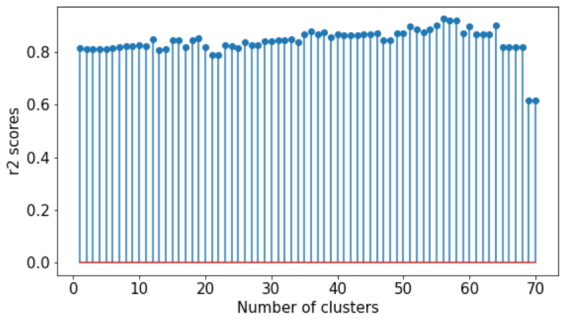


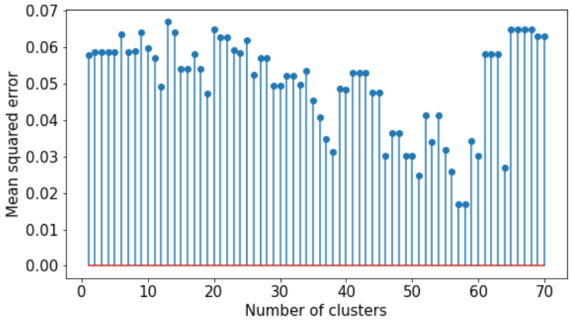


And we visualize also how the size of the main cluster reduces as we increase the number of clusters.

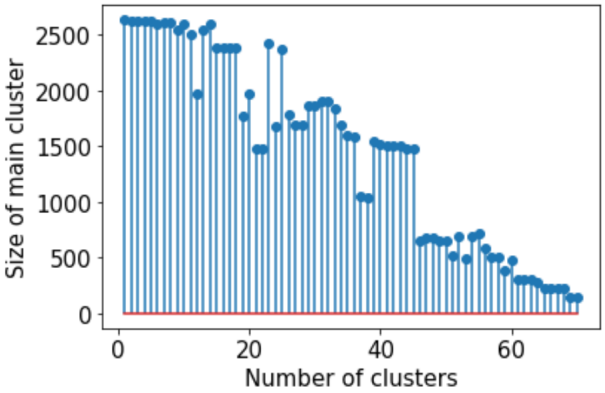


We make a stem plot to better inspect the result.





And of the main cluster size.



We can see that above 45 clusters the main cluster becomes significantly smaller while it stays almost constant for the first 10 cluster numbers, also we can see that the r2 score stays above 0.81 and does not increase much before 35 clusters. We can see that, while there is some improvement when running the linear regression on the main cluster, when increasing the number of clusters the improvement is not

very high. For higher number of clusters, above 20 or 30, there is more improvement but we probably run into overfitting issues as the size of the data becomes too small. Combining these plots we see that this technique seems to bring significant improvement to the model and the number of clusters to choose is in the range below 10.

4 Conclusions

We have tried to improve the performance of the linear regression with the use of the Foursquare data and the clustering algorithm on this data. We can see that based on the feature log living area only, the regression performs good with a r2 score of almost 0.73, this can be significantly improved to 0.80 by adding the number of venues in that area, feature which we have extracted from the Foursquare data. We have seen that, as expected, there is no correlation and no improvement in adding the raw cluster labels to the model. However, if we run the linear regression on the biggest cluster (in the case of 10 clusters) we have a significant improvement in the r2 score to more than 0.82, by doing this we have to discard some data, which however accounts to less than 2% of the data.

We can conclude that it is indeed possible to improve the performance of the linear regression by adding the Foursquare data, we have found two ways to do this, the best models among the ones we have run, are the simple ones which uses the log living area and number of venues features only.

We have explored some ways to improve the linear regression model, future research might include using the cluster labels and the Foursquare data in different ways, for example to use a composite model that runs different models on different clusters.

Acknowledgements I would like to thank Andrew Brown, Matt Debondt, Stella Namio and Giulia Maraventano for their input, suggestions and advice.