The Rules Behind Housing Prices in the Neighborhoods of Munich City

- An applied data science project with machining learning by Nan Chen

Description: this project demo shows how to apply simple approaches in data science to obtain insights into some data as common as the housing prices in Munich. The methods adopted by the project include python libraries `numpy`, `pandas` and `lxml` to extract and process data from online source, and `geopy` to transfer address into geographical coordinates. The data visualization part is performed by `matplotlib` for plotting charts, `seaborn` to reveal variable correlations, as well as drawing maps by `folium` plus detailed features by `requests` and `json`. As next step, the location provider `Foursquare` is used to search for the nearby venues. After making a better generalization based on the original venue categories provided by `Foursquare API`, they can be processed by `KMeans` for neighborhood clustering. In this way, all neighborhoods are clustered by the machine by their shared interesting characters. As the final step, the counting numbers of venue categories are fed into `sklearn` to formulate an insightful price model. The entire data set is divided into a train group for developing model and a test group for checking result. Different models (*Simple Linear Regression* vs. *Multiple Linear Regression* vs. *Multiple-variable Polynomial Fit*) are attempted to select the best. Finally, the new price models can be proved by feeding it with some new data, in which the estimated prices are found to be meaningful with tolerable errors compared to the reality.

1. Introduction

The data description about the Munich city neighborhoods (districts or “Bezirk” in German) can be found from wiki [1]. The tabular information can be extracted by `pd.read_html` and processed into a district data frame. From the area and population numbers we can calculate the population density. The housing prices of Munich by districts are listed on a local real-estate website [2]. Their neighborhood names are mostly consistent, yet not exactly the same. Thus, a treatment has to be made to align the price data to the district data. I dropped accidentally three rows of “unofficial” districts, but they will be still made use of later in the project to testify my price model.

The population density and the prices can be combined together as the following plot:

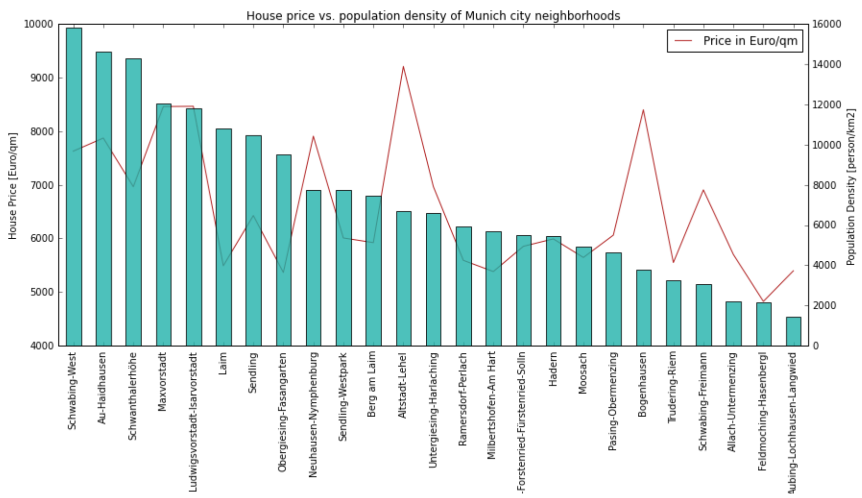

Fig.1: Population density and housing prices per neighborhood of Munich city

This data set will be used as the starting point of analysis throughout the whole project.

2. A first glance of data

Python `numpy` and `pandas` can do basic data analysis in very fast way. An intuitive thought about if any correlation already exists between area, population, population density and price, can be easily realized by using `df.corr`. This returns us correlation coefficients between [-1, 1] while 1 stands for absolute positive correlation and -1 for absolute negative correlation. The table looks like this:

	Price in Euro/qm	Area in km2	Population	Population Density
Price in Euro/qm	1.000000	-0.407529	-0.124438	0.462490
Area in km2	-0.407529	1.000000	0.457620	-0.811897
Population	-0.124438	0.457620	1.000000	-0.215528
Population Density	0.462490	-0.811897	-0.215528	1.000000

Fig.2: Correlation coefficients between basic variables

As for the target variable "price", we can see that the "population density" has certain positive correlation while "area" has certain negative correlation, although both not strong. These relations can be visualized by `seaborn.regplot` as Fig.3. This makes sense to our common sense that the denser people live in a certain region, the more centered this region may locate and thus higher price it has. The city suburbs have affluent areas where the housing prices start to fell. Notice that "population" has almost no correlation to "price" because of its ambiguous meaning: either the housing price is affordable for many people, or the region is so popular to these people despite of its high price.

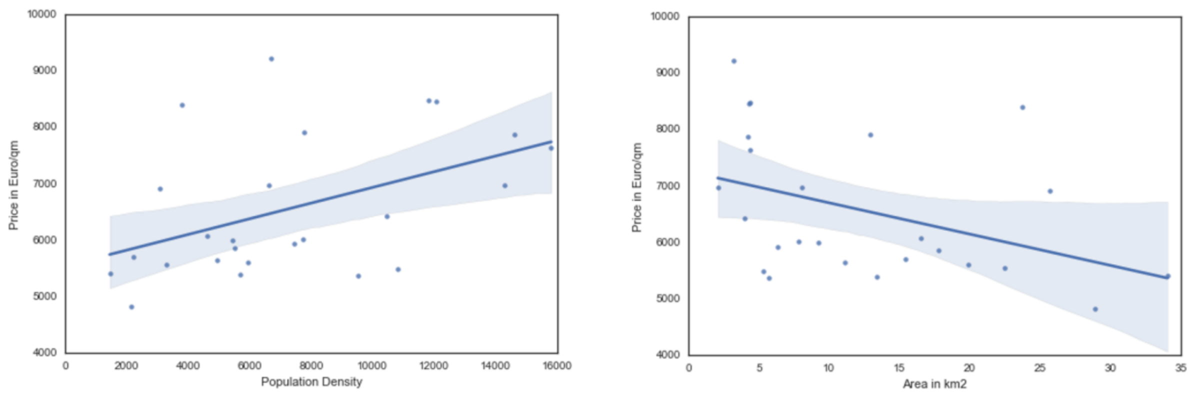

Fig.3: Linear regression diagrams of population density vs. price and area vs. price

By taking a second look at Fig.1, we can now notice that the price curve is indeed gradually decreasing from left to right as the order of population density. However, there are several expensive neighborhoods as outliners, e.g. "Altstadt-Lehr" (old city), "Neuhausen-Nymphenburg" and "Bogenhausen" (two famous rich-people districts) as the top three. This fact suggests us that it is necessary to find out further data variables if we aim to reveal the rules behind the price.

3. Neighborhoods on the map

Before including more data into our discussion, let us firstly plot our neighborhoods on a map to see how they are distributed among the city. This can be easily done by using `Folium`. In addition, I found the geo.json from [4] to enrich the map content with details such as district borders. All neighborhoods are illustrated at their geographic centers, see Fig.4.

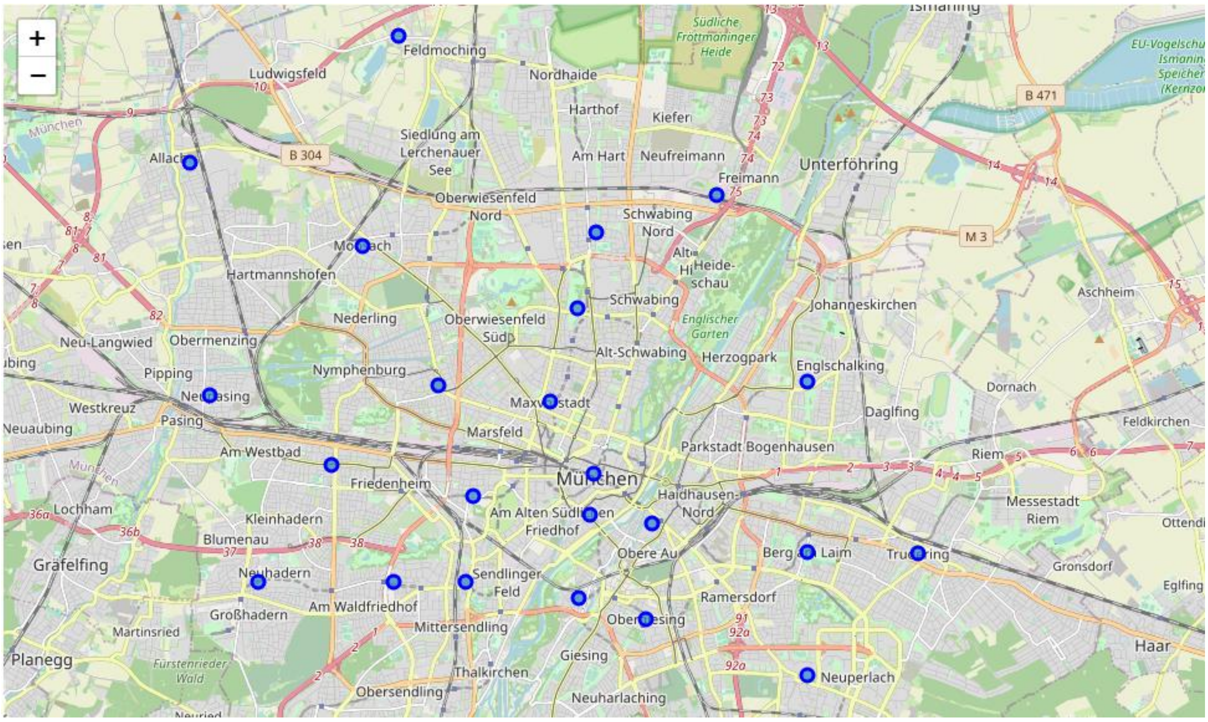

Fig.4: Munich city map with all neighborhoods

4. Explore the venues of neighborhoods

Similar to the Coursera applied science capstone tutorial part, this project also uses `Foursquare API` to search for the nearby venues within each neighborhood. The searching radius is set to 1700m to fit with the mean area value ($A = \pi r^2$) per neighborhood obtained before, and the limit of venues is set to be large enough to find as many venues as possible. A disadvantage regarding the venue categories returned by `Foursquare` lies in that there are too many different categories which are essentially the same, such as American Restaurant, Asian Restaurant, German Restaurant, etc. are all “Restaurant”, or “Pub”, “Bar”, “Nightclub”, etc. are all “Nightpub”. This would lead to two main problems later for data analysis:

(1) The number of venues in each category becomes rare and the number of category types becomes too large;

(2) The machine is unable to treat all “Restaurants”, “Pubs” together, but to let them stand separately against each other.

To overcome this drawback, I decided to make a further generation of venue categories into a total of 15 venue categories, and they are: “Restaurant”, “Cafe”, “Shopping”, “Supermarket”, “SportFitness”, “Nightpub”, “Hotel”, “Snack”, “Culture”, “Landmark”, “Pharmacy”, “Park”, “PublicTransport”, “Bakery” and “Bank”. In addition, all the rest venue categories with less than five venues will be omitted. Obviously, this kind of generation imposes my own tastes and may be affected by my own life experience in Germany. Nevertheless, I believe (and have tested) that these new venue categories are much better than the original ones. The resulting venue categories are shown by `df.value_counts()` in Fig. 5.

```
Restaurant       642
Cafe             199
Shopping         159
Supermarket      126
SportFitness     116
Nightpub         112
Hotel             92
Snack             70
Culture           67
Landmark          67
Pharmacy          65
Park              64
PublicTranport    62
Bakery            60
Bank              16
Name: VenueCategory, dtype: int64
```

Fig.5: The new venue categories after my generalization

We can further put all the venues into the previous city map by embedding `Folium.plugins` to get an impression on where are they exactly located, see Fig.6.

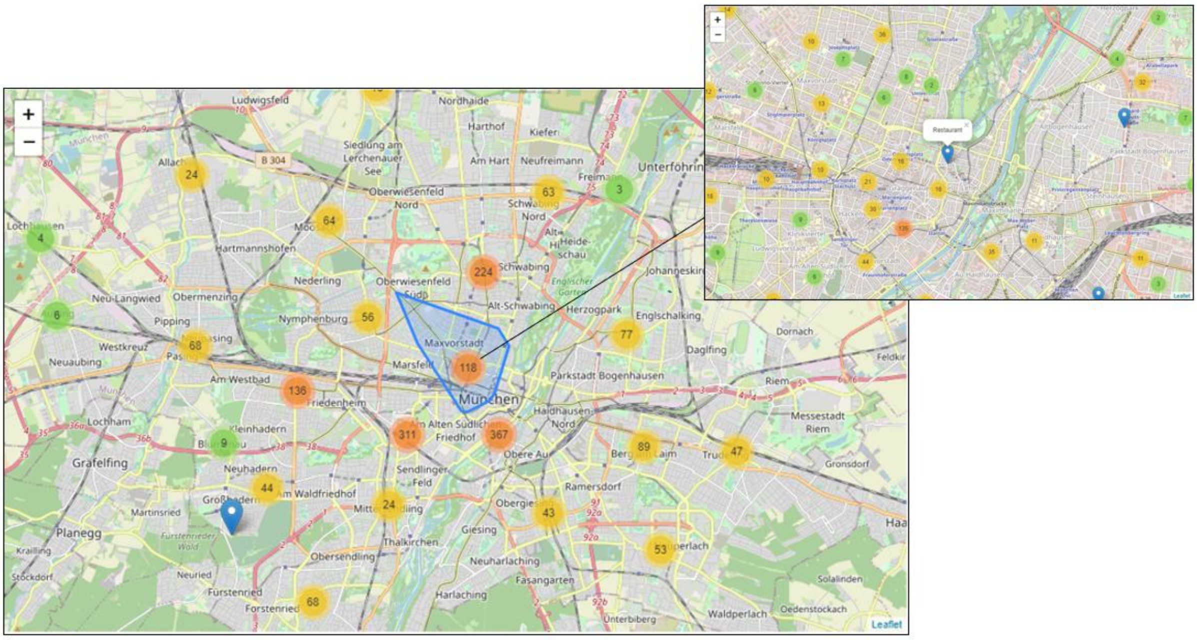

Fig.6: The venues on the Munich city map (venue categories appear as labels by zoom-in)

5. Neighborhood clustering

The data frame containing the neighborhoods, venues and venue categories can be transformed into another data frame with focus on the most common venues for each neighborhood. This new data frame can be used as input by `KMeans` to make a neighborhood clustering. The selection on the number of clusters can be verified by the “elbow” method from `yellowbrick.cluster`, which generates the plot shown in Fig.7:

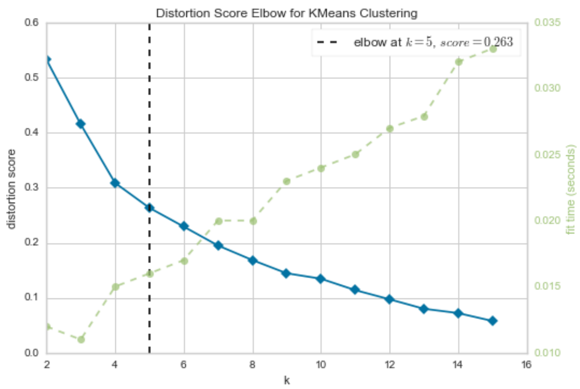

Fig.7: The “elbow” method to select the number of clusters for K-means

Thus, the number of clusters is set to 5. Fig.8 shows the clustering result conducted by `KMeans`.

Neighborhood	Price in Euro/qm	Area in km2	Population	Population Density	Latitude	Longitude	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
Schwabing-West	7628.310	4.36	68935	15810.779817	48.168270	11.569873	0	Restaurant	SportFitness	Cafe	Nightpub	Culture
Neuhausen-Nymphenburg	7907.295	12.91	100213	7762.432223	48.154221	11.531517	0	Restaurant	Cafe	SportFitness	Snack	Nightpub
Obergiesing-Fasangarten	5364.330	5.72	54498	9527.622378	48.111156	11.588909	0	Restaurant	SportFitness	Cafe	Nightpub	Supermarket
Sendling	6426.750	3.94	41256	10471.065990	48.118011	11.539083	0	Restaurant	Cafe	Nightpub	SportFitness	Shopping
Untergiesing-Harlaching	6966.730	8.06	53243	6605.831266	48.114964	11.570189	0	Restaurant	Cafe	Nightpub	Snack	Park
Schwanthalerhöhe	6964.680	2.07	29611	14304.830918	48.133781	11.541057	0	Restaurant	Cafe	Nightpub	Hotel	SportFitness
Aubing-Lochhausen-Langwied	5396.160	34.06	49072	1440.751615	48.165058	11.400222	1	Restaurant	PublicTranport	Supermarket	Pharmacy	Hotel
Feldmoching-Hasenbergl	4824.410	28.94	62069	2144.747754	48.218460	11.520409	1	Restaurant	Supermarket	Cafe	PublicTranport	Bakery
Hadern	5991.380	9.22	50165	5440.889371	48.118065	11.481842	1	Restaurant	PublicTranport	Supermarket	Cafe	Bank
Trudering-Riem	5549.670	22.45	73479	3273.006682	48.123177	11.664078	1	Restaurant	Supermarket	PublicTranport	Hotel	Shopping
Maxvorstadt	8460.265	4.30	51834	12054.418605	48.151093	11.562418	2	Restaurant	Cafe	Nightpub	Landmark	Hotel
Au-Haidhausen	7872.340	4.22	61654	14609.952607	48.128754	11.590536	2	Restaurant	Cafe	Nightpub	Culture	Shopping
Altstadt-Lehel	9208.190	3.15	21126	6706.666667	48.137829	11.574582	2	Cafe	Restaurant	Shopping	Landmark	Nightpub
Ludwigsvorstadt-Isarvorstadt	8464.500	4.40	51933	11802.954545	48.130341	11.573366	2	Cafe	Restaurant	Shopping	Snack	Nightpub
Berg am Laim	5921.690	6.31	47000	7448.494453	48.123482	11.633451	3	Restaurant	Supermarket	PublicTranport	SportFitness	Shopping
Ramersdorf-Perlach	5590.750	19.90	117918	5925.527638	48.100895	11.633371	3	Restaurant	Shopping	Supermarket	Cafe	SportFitness
Thalkirchen-Obersendling-Forstenried-Fürstenri...	5852.460	17.76	97689	5500.506757	48.084213	11.508051	3	Restaurant	Shopping	Supermarket	SportFitness	PublicTranport
Pasing-Obermenzing	6061.300	16.50	76348	4627.151515	48.152363	11.468434	3	Restaurant	SportFitness	Supermarket	Cafe	PublicTranport
Bogenhausen	8399.890	23.71	90025	3796.921130	48.154781	11.633484	3	Restaurant	Supermarket	Park	SportFitness	Cafe
Allach-Untermenzing	5699.770	15.45	34277	2218.576052	48.195156	11.462974	4	Restaurant	Shopping	Hotel	Supermarket	SportFitness
Schwabing-Freimann	6905.720	25.67	78657	3064.160499	48.189278	11.608582	4	Restaurant	Shopping	Hotel	SportFitness	Snack
Laim	5489.510	5.29	57111	10796.030246	48.139549	11.502166	4	Restaurant	Supermarket	Shopping	SportFitness	PublicTranport
Sendling-Westpark	6008.750	7.81	60498	7746.222791	48.118031	11.519333	4	Restaurant	Supermarket	SportFitness	Shopping	Park
Moosach	5643.690	11.09	54872	4947.880974	48.179893	11.510571	4	Restaurant	Shopping	Supermarket	Pharmacy	Bakery
Milbertshofen-Am Hart	5381.745	13.42	76559	5704.843517	48.182384	11.575043	4	Restaurant	Shopping	Hotel	SportFitness	Nightpub

Fig.8: The result of neighborhood clustering conducted by `KMeans` (column = “Cluster Labels”)

Now let us take a close look at each cluster one by one to understand why the machine believes the neighborhoods should be clustered like this. To quickly visualize the similarities, we can apply `df.describe()` and `df.describe(include=[object])` to get good summarized information for number and non-number values. Among all values returned I find the `mean` of price and the `top` of venues to be mostly valuable, and thus summarize these values into the following table (Fig.9).

Number of Clusters	Neighborhood (in top)	Mean Price in Euro/qm	1st common venue	2nd common venue	3rd common venue	4th common venue	5th common venue
2	Maxvorstadt	8501.32	Restaurant	Restaurant	Shopping	Landmark	Nightpub
0	Schwanthalerhöhe	6876.35	Restaurant	Cafe	Nightpub	Snack	Park
3	Berg am Laim	6365.22	Restaurant	Supermarket	Supermarket	SportFitness	PublicTranport
4	Milbertshofen a. H.	5854.86	Restaurant	Shopping	Hotel	SportFitness	Snack
1	Aubing-L.-Langwied	5440.40	Restaurant	Supermarket	Supermarket	Pharmacy	Bank

Fig.9: Typical neighborhoods by cluster with their mean prices and top five common venues

Alone from this table we can already come to understand that the values “Landmark”, “Nightpub”, “Park” and “Cafe” could be positive correlation factors to the price, whereas the values “Supermarket” and “Pharmacy” are likely to be negative correlation factors. The clustering decided by the machine is essentially a reflection of the differences in the mostly common venue categories.

The five clusters of neighborhoods can be also plotted on the city map as Fig.10. Here we can see that the clustering made by the machine seems quite logical, since the clusters are nearly distributed according to their radius distance to the city center. Having lived in Munich for five years during my study, I can also affirm from my life experience that this clustering result is consistent with my impressions there and thus meets my satisfaction.

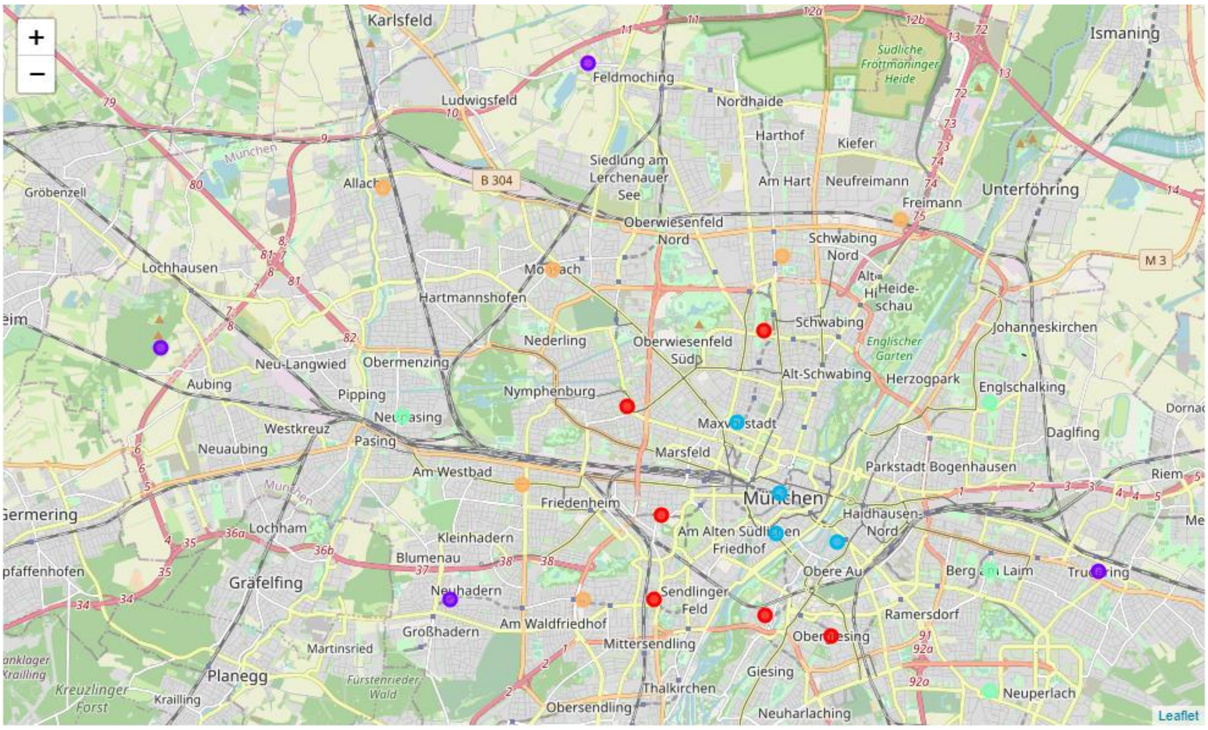

Fig.10: Result of neighborhood clustering by `KMeans` on the city map

(blue: 2; red: 0; green: 3; brown: 4; purple: 1)

6. Develop some price estimation models (SLR, MLR and MPF)

The data frame previously used to generate Fig.8 and Fig.9 already contain all venue categories which have correlations to the price. In order to make them as our input values for a price model, they still have to be converted from string into numerical values. This can be done by conducting either `df.groupby('Neighborhood').sum()` or `.mean()` to get some scores based on the occurring frequency of venue categories. I decided to use `sum()` to make the values look more like scores, which results in the new data frame as shown in Fig.11.

	Neighborhood	Price in Euro/qm	Area in km2	Population	Population Density	Bakery	Bank	Cafe	Culture	Hotel	Landmark	Nightpub	Park	Pharmacy	PublicTranport
0	Schwabing-West	7628.310	4.36	68935	15810.779817	3	0	8	6	1	3	6	5	0	0
1	Au-Haidhausen	7872.340	4.22	61654	14609.952607	1	0	18	8	5	7	10	3	0	0
2	Schwanthalerhöhe	6964.680	2.07	29611	14304.830918	0	0	14	5	9	1	9	2	0	0
3	Maxvorstadt	8460.265	4.30	51834	12054.418605	4	0	18	5	5	8	10	4	0	0
4	Ludwigsvorstadt-Isarvorstadt	8464.500	4.40	51933	11802.954545	0	0	25	7	3	7	8	4	1	0
5	Laim	5489.510	5.29	57111	10796.030246	6	3	3	3	5	3	4	1	6	6
6	Sendling	6426.750	3.94	41256	10471.065990	3	0	12	5	2	0	10	3	3	0
7	Obergiesing-Fasangarten	5364.330	5.72	54498	9527.622378	2	0	8	1	3	5	7	2	4	0
8	Neuhausen-Nymphenburg	7907.295	12.91	100213	7762.432223	4	0	9	3	4	5	5	2	3	1
9	Sendling-Westpark	6008.750	7.81	60498	7746.222791	4	2	5	6	5	3	2	8	5	4
10	Berg am Laim	5921.690	6.31	47000	7448.494453	4	1	1	0	3	0	2	1	4	5
11	Altstadt-Lehel	9208.190	3.15	21126	6706.666667	0	0	20	5	5	13	11	3	1	0
12	Untergiesing-Harlaching	6966.730	8.06	53243	6605.831266	1	0	17	1	2	2	12	6	1	0
13	Ramersdorf-Perlach	5590.750	19.90	117918	5925.527638	2	0	5	1	1	1	0	3	3	3
14	Milbertshofen-Am Hart	5381.745	13.42	76559	5704.843517	3	1	7	3	10	1	8	3	6	0
15	Thalkirchen-Obersendling-Forstenried-Fürstenri...	5852.460	17.76	97689	5500.506757	4	3	3	2	3	0	1	0	4	4
16	Hadern	5991.380	9.22	50165	5440.889371	1	3	3	0	1	2	0	0	2	10
17	Moosach	5643.690	11.09	54872	4947.880974	5	0	1	1	4	1	2	0	6	3
18	Pasing-Obermenzing	6061.300	16.50	76348	4627.151515	1	0	7	2	1	1	0	2	4	6
19	Bogenhausen	8399.890	23.71	90025	3796.921130	2	2	6	1	4	3	3	7	5	4
20	Trudering-Riem	5549.670	22.45	73479	3273.006682	4	0	3	0	6	0	1	0	3	7
21	Schwabing-Freimann	6905.720	25.67	78657	3064.160499	1	1	3	2	6	0	0	2	1	2
22	Allach-Untermenzing	5699.770	15.45	34277	2218.576052	2	0	0	0	3	0	0	2	2	2
23	Feldmoching-Hasenbergl	4824.410	28.94	62069	2144.747754	2	0	3	0	0	1	1	1	0	2
24	Aubing-Lochhausen-Langwied	5396.160	34.06	49072	1440.751615	1	0	0	0	1	0	0	0	1	3

Fig.11: Data frame containing the sum of venue categories as “scores” of neighborhoods.

Let us check again the correlation coefficients of new variables (Fig.12):

	Price in Euro/qm	Area in km2	Population	Population Density	Bakery	Bank	Cafe	Culture	Hotel	Landmark	Nightpub	Park	Pharmacy
Price in Euro/qm	1.000000	-0.407529	-0.124438	0.462490	-0.333092	-0.186172	0.759212	0.621016	0.166855	0.748047	0.586076	0.499297	-0.421073

PublicTranport	Restaurant	Shopping	Snack	SportFitness	Supermarket
-0.435510	0.401825	0.111006	0.474879	-0.106851	-0.460902

Fig.12: Correlation coefficients of all input variables of price

We can then put all venue variables into four different groups. I selected my threshold value to determine whether the variable has a correlation with price to be 0.46, the value of population density, with the hope that my later price function based on the variables beyond this scale will have better performance than just using "PopulationDensity".

	Correlation exists (abs. value >0.46)	**Correlation weak (abs. value <0.46)**
Positive	Cafe (0.759) Landmark (0.748) Culture (0.621) Nightpub (0.586) Park (0.499) Snack (0.475)	Restaurant (0.402) Hotel (0.167) Shopping (0.111) Population Density (0.46)
Negative	Supermarket (-0.461) PublicTransport (-0.436) Pharmacy (-0.421)	Bakery (-0.333) Bank (-0.186) SportFitness (-0.107)

Fig.13: Correlation variables in four groups (threshold = 0.46 as the value of "Population Density")

Therefore, we get our columns of interest as:

```
cols_of_interest = ['Cafe', 'Landmark', 'Culture', 'Nightpub',
'Park', 'Snack', 'PublicTranport', 'Supermarket', 'Pharmacy']
```

You may notice that although "Restaurant" is literally *everywhere* in all neighborhoods, it will be just taken out of the price function (just because it is *everywhere*). For the rest variables we can also ask ourselves *why a certain variable should be with positive, negative or without correlation*. I think they are understandable so that I will spare some space here without making further discussion on each specifically.

Now, let us concentrate on developing a suitable price model based on these variables of interest. Before doing that, I would like to divide my whole data set into two groups: training group for developing model and test group for checking model. This is done by using `train_test_split` in the module `sklearn.model_selection`. I found the best proportion should be set to 0.20 (meaning 20% x 25 = 5 test samples and 20 training samples).

a. Single Linear Regression (SLR)

As a first attempt, I selected the strongest correlated variable "Cafe (0.759)" to generate a SLR model. The good thing about this variable is that it *almost* appears in every neighborhood (except "Allach-Untermenzing" and "Aubing-L.-L."). I found the R^2 score of this model to be around 0.58 (=58% chance of correctness; not bad but also not outstanding). However, when I used:

```
Rcross = cross_val_score(lre, x_data[['Cafe']], y_data, cv=3)
```

to test this model with four portions of all data samples simultaneously, I got these R^2 scores:

```
array([0.51885359, 0.66583818, 0.1493539 ])
```

with a bad score. This infers that the last data portion must be the one containing the two neighborhoods without any "Cafe".

Although not satisfied with this model, I still let it predict the price values for all neighborhoods and record the them into the table together with other models (see later Fig.15).

We can use R^2 score to quantify the quality of SLR model for the train, test and entire groups:

```
SLR Model:

The R-square of the train group is:  0.570714465991117
The R-square of the test group is:  0.5890650298731998
The R-square of the entire group is:  0.5760722720061815
```

b. Multiple Linear Regression (MLR)

This time, I introduced all variables of interest into a MLR model. The quality of this MLR model can be illustrated by the distribution curves shown in Fig. 14:

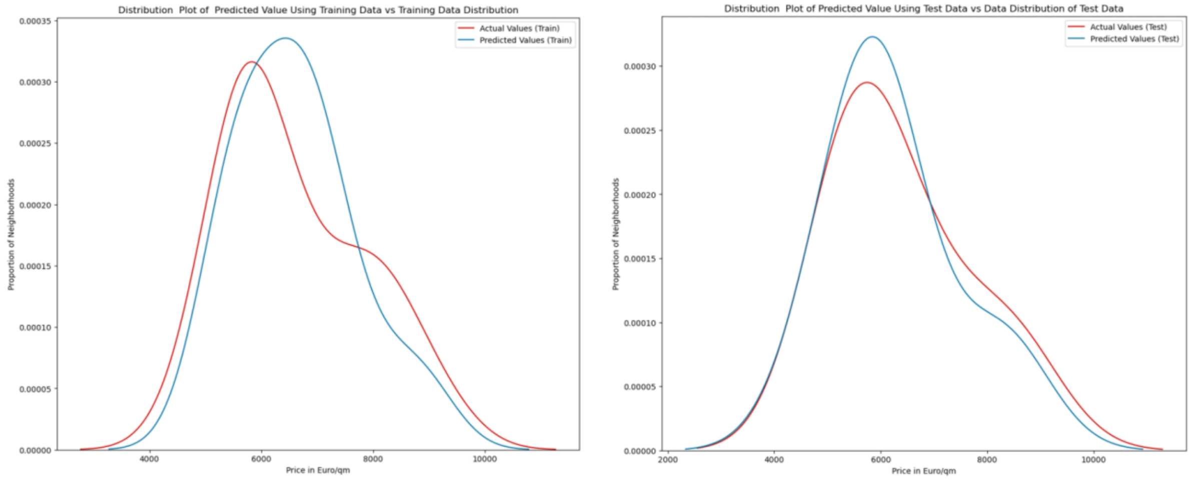

Fig.14: The distribution curves of MLR for the train samples (left) and the test samples (right)

Red: actual price values; Blue: Predicted price values

Although the predicted values for the train group does not 100% follow the actual values (underfitting), the predicted result for the test group is quite good. More importantly, the predicted curve is convergent at the left and right ends. This prevents our MLR model from predicting extraordinarily high price values or negative price values. The underfitting characteristic of this MLR model allows a robust response for the higher price range (8000-10,000 Euro/qm), yet with a compromise that the model does not want to response too rashly to the factors causing higher prices. The predicted result is also listed later in Fig.15.

We can still use R^2 score to quantify the quality of our MLR model:

```
MLR Model:

The R-square of the train group is:  0.731604826512386
The R-square of the test group is:  0.678830788649489
The R-square of the entire group is:  0.7230274922061546
```

c. Multivariate Polynomial Fit (MPF)

Now we can simply turn the number of degrees for the `PolynomialFeatures` in `sklearn.preprocessing` to 2 to switch our model from MLR into MPF. We can check the predicted curves again as shown in Fig.15.

Now it is clear that the Polynomial Fit is able to fit all the training samples perfectly. But this perfect match has its cost: the predicted curve for the test group begin to shift as a whole towards the higher price ranges. In the highest value range (above 8000 Euro/qm), the predicted curve also has to drop more radically to ensure its convergence. In short, the predicted values by MPF will have overpredictions for the 6000-8000 Euro/qm range, and underprediction for the above 8000 Euro/qm range.

We can further change the number of degrees to more than 2, and check the new behaviour of higher-order Polynomial Fit. I found that all higher-order models have worse predicted curves for the test group because of their “over-bonding” to the training samples.

We can use R^2 score to quantify the quality of our MPF model. The R^2 score for the test group indeed decreases (to 0.537) compared to the MLR model (= 0.679):

```
MPF Model:
The R-square of the train group is:  1.0
The R-square of the test group is:  0.5367417524084587
The R-square of the entire group is:  0.9139986956697428
```

Therefore, we have successfully generated three different price models: Single Linear Regression (SLR), Multiple Linear Regression (MLR) and a Multivariate Polynomial Fit (MPF). All three models

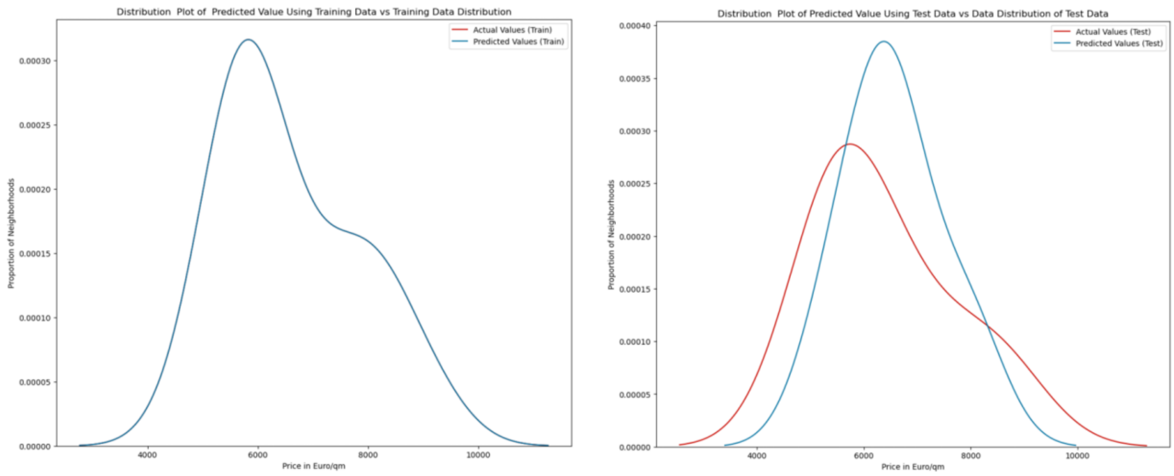

Fig.15: Distribution curves of MPF for the train samples (left) and the test samples (right)

Red: actual price values; Blue: Predicted price values

Have their own predicted values, which are summarized in the table as Fig.16. The errors made by three different models are consistent with what we have discussed above from the distribution curves and R^2 scores.

ID	Neighborhood	Price in Euro/qm	SLR	Error_SLR	MLR	Error_MLR	MPF	Error_MPF
0	Schwabing-West	7628	6604	-0.13	6954	-0.09	7628	0.00
1	Au-Haidhausen	7872	8027	0.02	7864	0.00	7872	0.00
2	Schwanthalerhöhe	6965	7458	0.07	6633	-0.05	6965	0.00
3	Maxvorstadt	8460	8027	-0.05	8273	-0.02	7952	-0.06
4	Ludwigsvorstadt-Isarvorstadt	8465	9023	0.07	8532	0.01	8465	0.00
5	Laim	5490	5892	0.07	5940	0.08	5490	0.00
6	Sendling	6427	7173	0.12	6571	0.02	6427	0.00
7	Obergiesing-Fasangarten	5364	6604	0.23	6402	0.19	5364	0.00
8	Neuhausen-Nymphenburg	7907	6746	-0.15	7047	-0.11	7907	0.00
9	Sendling-Westpark	6009	6256	0.04	7124	0.19	6009	0.00
10	Berg am Laim	5922	5749	-0.03	5033	-0.15	5922	0.00
11	Altstadt-Lehel	9208	8156	-0.11	9024	-0.02	9208	0.00
12	Untergiesing-Harlaching	6967	7776	0.12	7443	0.07	6967	0.00
13	Ramersdorf-Perlach	5591	6256	0.12	5948	0.06	5412	-0.03
14	Milbertshofen-Am Hart	5382	6509	0.21	6157	0.14	6797	0.26
15	Thalkirchen-Obersendling-F…	5852	6002	0.03	5453	-0.07	5852	0.00
16	Hadern	5991	6002	0.00	6268	0.05	5991	0.00
17	Moosach	5644	5467	-0.03	4979	-0.12	6339	0.12
18	Pasing-Obermenzing	6061	6318	0.04	6387	0.05	6061	0.00
19	Bogenhausen	8400	6176	-0.26	7077	-0.16	8400	0.00
20	Trudering-Riem	5550	5751	0.04	5259	-0.05	5550	0.00
21	Schwabing-Freimann	6906	5751	-0.17	5922	-0.14	6308	-0.09
22	Allach-Untermenzing	5700	5325	-0.07	5668	-0.01	5700	0.00
23	Feldmoching-Hasenbergl	4824	5751	0.19	5715	0.18	4824	0.00
24	Aubing-Lochhausen-Langwied	5396	5325	-0.01	5606	0.04	5396	0.00

Fig.15: A summary of predicted price values and errors in percentage made by SLR, MLR and MPF

7. Final examination of price models with a few new test data

Before in Chapter 1, I mentioned there are three additional rows of “unofficial districts” from the online source [2]. Now as a final check, we can import these three new neighborhoods to let our price models to calculate their predicted price.

The whole procedure is the same as before. In addition, the three models are already available. We only need to request the venues by `Foursquare API` again, and put the new counting values of venue categories into the price model.

The predicted prices are summarized in Fig.16.

ID	Neighborhood	Price in Euro/qm	SLR	Error_SLR	MLR	Error_MLR	MPF	Error_MPF
0	Alte Heide-Hirschau	6549	6174	-0.06	6937	0.06	8544	0.30
1	Daglfing	6305	6043	-0.04	6116	-0.03	5620	-0.11
2	Oberföhring-Englschalking	5538	5781	0.04	6358	0.15	6907	0.25

Fig.16: predicted price values and errors in percentage for the new data made by SLR, MLR and MPF

We can see that SLR and MLR both have done impressive good work for the new data set. The SLR method may achieve its outstanding result by chance, since as shown in the following distribution plot, its prediction is in a conservative manner, which happens to have more predicted values within the mostly common range 5500-6500 Euro/qm. Its three max. errors in Fig.15 are due to this kind of conservation.

Therefore, we can finally draw to the conclusion that the MLR model is the best price prediction model. It is capable of generating predicted values within +/-15% error range. Furthermore, its behavior has a well balance between robustness and response sensitivity.

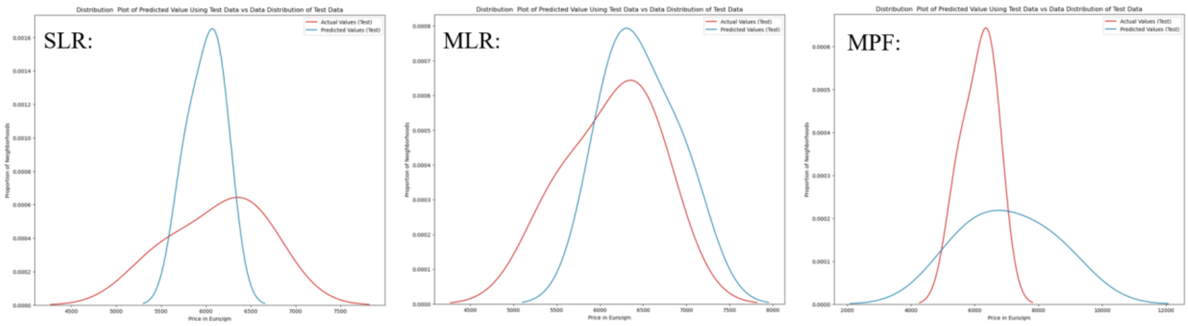

Fig.17: Distribution curves of MPF for the new data samples (left: SLR; middle: MLR; right: MPF)

Red: actual price values; Blue: Predicted price values

8. Conclusion

This applied data science project demo summarizes nearly all the tools I have learned from the applied data science specification series. Despite of some drawbacks of venue data provided by Foursquare (e.g. its content is strongly biased to tourists' interest instead of a full description of neighborhoods), the information can be still processed in a scientific way to make neighborhood clustering and even to develop a successful price prediction model.

The beaty of data science lies in its strength in dealing with common data to reveal its underlying secrets. I am looking forward to all other applications in future with the tools I have learned here. So, keep going!

Appendix

[1] wiki website of Munich districts: https://de.wikipedia.org/wiki/Stadtbezirke_M%C3%BCnchens

[2] an online source about the Munich housing prices per district: https://suedbayerische-immobilien.de/Immobilienpreise-Muenchen

[3] Geo.json containing geographical data of counties and districts in Germany https://github.com/isellsoap/deutschlandGeoJSON/blob/master/3_regierungsbezirke/1_sehr_hoch.geo.json