Relocation to Germany - Report

IBM Data Science Certification - Capstone Project

Introduction

Relocation to another country, to find new job opportunities, can be a real challenge for anyone.
In fact, once you decided to move away from your country you have many choices to make and to find the best cities, or the best candidate cities for relocation is a tough problem.

Our client, in this example project, is Andrea, who is an employee in a IT Consulting company in Italy but wants to find better job opportunities in Germany, where he knows he's going to be much better paid.

However, things are not so simple.
In fact, not all german cities are the same and some are clearly worse than others when you take in consideration average salary, unemployment rate, etc.

So, to tackle this problem, we talked to Andrea and stated clearly what are his major criteria in order to pick one city over another.

Let's list them here:

- A big German city is preferrable, and has to have at least 100'000 citizens
- Low unemployment rate
- High wealth indexes, such as GDP and average net income per employee
- Many Italian restaurants and cafè close to the center of the city (max 20' away)

Also, we ask him to assign a weight to each of these requirements.

Requirement	Weight
City population	0,7
Unemployment Rate	0,9
Wealth Index	0,8
Number of italian Restaurants and cafès	0,5

Now all we have to do is gather the relevant data, analyze what we obtain and decide which are the best candidate cities to relocate in Germany, based on these criteria.

Data

Following the business requirements stated above, it's clear that the relevant data will be:

- Population Data
- Wealth Index Data
- Unemployment Rate Data
- City venues Data

1. Population Data

   Since we want to obtain the population of the biggest german cities, and these cities in order to be candidates have to have at least 100'000 citizens, we decided to use this Wikipedia page to scrape the relative HTML table:
   https://en.wikipedia.org/wiki/List_of_cities_in_Germany_by_population

   Let's take a snapshot of the table:

2015 rank	City	State	2015 estimate	2011 census	Change	2015 land area	2015 population density	Location
1	***Berlin***	Berlin	3,520,031	3,292,365	+6.91%	891.68 km^2 344.28 sq mi	3,948/km^2 10,230/sq mi	52°31′N 13°23′E
2	***Hamburg***	Hamburg	1,787,408	1,706,696	+4.73%	755.3 km^2 291.6 sq mi	2,366/km^2 6,130/sq mi	53°33′N 10°0′E
3	***Munich*** (München)	Bavaria	1,450,381	1,348,335	+7.57%	310.7 km^2 120.0 sq mi	4,668/km^2 12,090/sq mi	48°8′N 11°34′E
4	*Cologne* (Köln)	North Rhine-Westphalia	1,060,582	1,005,775	+5.45%	405.02 km^2 156.38 sq mi	2,619/km^2 6,780/sq mi	50°56′N 6°57′E
5	*Frankfurt am Main*	Hesse	732,688	667,925	+9.70%	248.31 km^2 95.87 sq mi	2,951/km^2 7,640/sq mi	50°7′N 8°41′E
6	***Stuttgart***	Baden-Württemberg	623,738	585,890	+6.46%	207.35 km^2 80.06 sq mi	3,008/km^2 7,790/sq mi	48°47′N 9°11′E
7	**Düsseldorf**	North Rhine-Westphalia	612,178	586,291	+4.42%	217.41 km^2 83.94 sq mi	2,816/km^2 7,290/sq mi	51°14′N 6°47′E

   We will use the *'City'*, *'State'* and *'2015 estimate'* columns from this table and save this in a *pandas Dataframe.*

2. Wealth Index Data

   For this data, we've obtained the GDP (Gross Domestic Product) indicator, for the best 107 cities in Germany, and also the list of German state by Household income per year. So, after a discussion with Andrea, we decided to leave both of these 2 parameters.
   We retrieved the GDP for city from this Wikipedia page:
   https://en.wikipedia.org/wiki/List_of_German_cities_by_GDP

List of German cities by GDP

From Wikipedia, the free encyclopedia

The following article sorts the 107 urban districts (*Kreisfreie Städte* – cities that constitute districts in their own right) and the metropolitan districts of Hanover, Aachen and Saarbrücken by their gross domestic product in the year 2016. Most figures are from the Federal Statistical Office of Germany; figures from other sources are otherwise referenced. The GDP of German cities are shown in EUR€.[1]

Rank	City	State	Gross Domestic Product in million €	Gross Domestic Product per capita in €	Gross Domestic Product per employee in €
1	Berlin	Berlin	130.537	36,798	68,906
2	Hamburg	Hamburg	112.959	62,793	92,163
3	Munich	Bavaria	109.571	75,186	100,776
4	Frankfurt am Main	Hesse	66.917	91,099	97,178
5	Cologne	North Rhine-Westphalia	63.463	59,407	85,127
6	Stuttgart	Baden-Württemberg	51.571	82,397	99,311
7	Hannover Region	Lower Saxony	49.578	43,240	73,788
8	Düsseldorf	North Rhine-Westphalia	48.783	79,619	93,054
9	Nuremberg	Bavaria	28.130	55,071	72,379
10	Bremen	Bremen	28.108	50,052	78,738
11	Essen	North Rhine-Westphalia	24.196	41,512	73,327

and we extracted the columns *'City'* and '*Gross Domestic Product per employee in €*'.

While we retrieved the Household income per state from this page: https://en.wikipedia.org/wiki/List_of_German_states_by_household_income

List of German states by household income

From Wikipedia, the free encyclopedia

This is a **list of German states by household income per capita** in 2016 according to th

Rank	State	Household income per capita (in EUR€)
1	Hamburg	24,421
2	Bavaria	24,026
3	Baden-Württemberg	23,947
4	Hesse	22,454
5	Rhineland-Palatinate	22,240
6	Schleswig-Holstein	22,217
7	North Rhine-Westphalia	21,614
8	Lower Saxony	21,045
9	Bremen	20,724
10	Saarland	20,536
11	Berlin	19,719
12	Brandenburg	19,431
13	Saxony	19,191
14	Thuringia	18,951

3. Unemployment Rate

As for the Household income data, the unemployment rate data was also retrieved by State, using the following Wikipedia page:
https://en.wikipedia.org/wiki/List_of_German_states_by_unemployment_rate

List of German states by unemployment rate

From Wikipedia, the free encyclopedia

This is a **list of German states by unemployment rate** as of July 2019 according to the Fe

Rank	States	unemployment rate (July 2019)[1]
1	Bremen	10.2%
2	Berlin	8.0%
3	Saxony-Anhalt	7.0%
4	Mecklenburg-Vorpommern	6.7%
5	North Rhine-Westphalia	6.6%
6	Saarland	6.4%
7	Hamburg	6.3%
	East Germany	6.3%
8	Brandenburg	5.7%
9	Saxony	5.3%
10	Thuringia	5.2%
11	Lower Saxony	5.1%
11	Schleswig-Holstein	5.1%
	West Germany	4.7%
13	Hesse	4.5%

When we extracted this data, we also had to convert the ‘object’ data type for column ‘Unemployment Rate’ to a float type.

4. City venues

To obtain the city venues of interest for our client, we used the Foursquare API, and its *explore* endpoint.
More specifically, for each city in our Dataframe we obtained the geographical data regarding Latitude and Longitude using the *geopy.geocoders* module, and then we made several API calls to obtain the list of all the venues of a city within 1500 meters from its center.
From these list of venues, we ignored those who weren’t of interest of our client and we counted (for each city) the number of italian restaurants and cafès.

So we’ve built a Dataframe that contained this data, let’s view the first rows of the dataset:

City	Num of IT Restaurants	Num of Cafès
Berlin	4	9
Hamburg	8	6
Munich	9	24
Nuremberg	3	9
Augsburg	5	5

Finally, we merged all the data retrieved up to this moment, in a single pandas Dataframe, that looked like this (first 5 rows shown):

	City	State	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès
0	Berlin	Berlin	3520031	68906	8.0	19719	4	9
1	Hamburg	Hamburg	1787408	92163	6.3	24421	8	6
2	Munich	Bavaria	1450381	100776	2.7	24026	9	24
3	Nuremberg	Bavaria	509975	72379	2.7	24026	3	9
4	Augsburg	Bavaria	286374	72062	2.7	24026	5	5

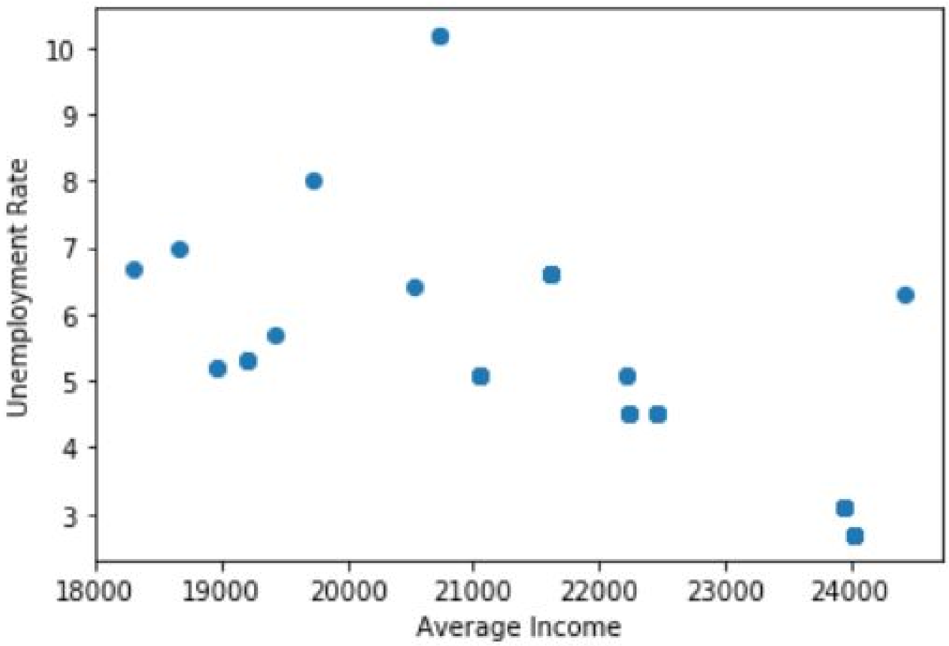
Methodology

Now that we have gathered all the relevant data for this project, it's time to describe how we can use it!

1. Exploratory Data Analysis on Employment and Average income

    Since we retrieved some important economic feature, such as unemployment rate and the average household income per employee, our client Andrea wants to know if there's a relationship between these 2 features.

    Let's plot the data points in a scatter plot:



    Although there are some outliers clearly shown in the above graph, we can see a trend that tells us how the lower is the unemployment rate, the higher is the average income per city, showing a negative linear relationship between the two features.

2. **Refining the Requirements Matrix**

In the Introduction section, discussing the business requirements and the criteria chosen by our client, we defined a Matrix where each criteria was associated with a weight.

Let's show this matrix again:

Requirement	Weight
City population	0,7
Unemployment Rate	0,9
Wealth Index	0,8
Number of italian Restaurants and cafès	0,5

However, not each one of these feature has a 'positive' correlation with its criteria, in order to define a 'Score' that represents which cities are best candidates to relocate to.

In fact, for example, while the Wealth Index is a positive factor (the higher the better), the Unemployment Rate is a negative one (the **lower** the better).

So let's build a correlated weight matrix:

Requirement	Correlated Weight
City population	0,7
Unemployment Rate	- 0,9
Wealth Index	0,8
Number of italian Restaurants and cafès	0,5

3. **Normalizing the features**

Let's take another look at the data we retrieved:

	City	State	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès
0	Berlin	Berlin	3520031	68906	8.0	19719	4	9
1	Hamburg	Hamburg	1787408	92163	6.3	24421	8	6
2	Munich	Bavaria	1450381	100776	2.7	24026	9	24
3	Nuremberg	Bavaria	509975	72379	2.7	24026	3	9
4	Augsburg	Bavaria	286374	72062	2.7	24026	5	5

As we can clearly see, the data is not standardized. The population column contains

values way above those about the unemployment rate, for example.
So, running any example of statistical testing on this data would prove useless.

In order to avoid this problem, we used the Simple Feature Scaling approach to normalize the data.

We define the Simple Feature Scaling approach to calculate the new values as following:

new_value = old_value / max_value

In this way all the values in our Dataframe will now range from 0 to 1.

Let’s have a look at the normalized data:

```
normalized_df.head()
```

	City	State	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès
0	Berlin	Berlin	1.000000	0.404815	0.784314	0.807461	0.266667	0.375000
1	Hamburg	Hamburg	0.507782	0.541447	0.617647	1.000000	0.533333	0.250000
2	Munich	Bavaria	0.412036	0.592048	0.264706	0.983825	0.600000	1.000000
3	Nuremberg	Bavaria	0.144878	0.425219	0.264706	0.983825	0.200000	0.375000
4	Augsburg	Bavaria	0.081356	0.423356	0.264706	0.983825	0.333333	0.208333

4. **Defining a Score Index**

Now we can combine points 2. and 3. and define a Score index multiplying each feature for its relative correlated weight.
So, we add a new column to the Dataframe, called ‘Score’, and we will compute the score for each row in the following way:

```
normalized_df['Score'] = normalized_df['Population']*0.7 + normalized_df['Num of IT Restaurants']*0.5 + \
                         normalized_df['Num of Cafès']*0.5 - normalized_df['Unemployment']*0.9 + \
                         normalized_df['GDP']*0.8 + normalized_df['Avg Income']*0.8
```

The Dataframe will now appear as this:

	City	State	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès	Score
0	Berlin	Berlin	1.000000	0.404815	0.784314	0.807461	0.266667	0.375000	1.284772
1	Hamburg	Hamburg	0.507782	0.541447	0.617647	1.000000	0.533333	0.250000	1.424390
2	Munich	Bavaria	0.412036	0.592048	0.264706	0.983825	0.600000	1.000000	2.110889
3	Nuremberg	Bavaria	0.144878	0.425219	0.264706	0.983825	0.200000	0.375000	1.277914
4	Augsburg	Bavaria	0.081356	0.423356	0.264706	0.983825	0.333333	0.208333	1.215292

Let's describe accurately this data, using the *describe* method for the Dataframe:

```
normalized_df['Score'].describe()
```

```
count    68.000000
mean      0.940990
std       0.378531
min       0.115034
25%       0.646974
50%       0.839148
75%       1.228434
max       2.110889
Name: Score, dtype: float64
```

We can sort the Dataframe by score value, and obtain this:

```
normalized_df.sort_values('Score', ascending=False)
```

	City	State	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès	Score
2	Munich	Bavaria	0.412036	0.592048	0.264706	0.983825	0.600000	1.000000	2.110889
38	Stuttgart	Baden-Württemberg	0.177197	0.583441	0.303922	0.980590	1.000000	0.750000	1.976734
6	Ingolstadt	Bavaria	0.037624	0.797363	0.264706	0.983825	0.466667	0.375000	1.633886
33	Frankfurt am Main	Hesse	0.208148	0.570910	0.441176	0.919455	0.466667	0.875000	1.611770
40	Mannheim	Baden-Württemberg	0.086869	0.475983	0.303922	0.980590	0.400000	0.625000	1.465038
...	...	...	...	...	...	...	...	...	...
30	Herne	North Rhine-Westphalia	0.044275	0.340526	0.647059	0.885058	0.133333	0.000000	0.495774
31	Bottrop	North Rhine-Westphalia	0.033279	0.306887	0.647059	0.885058	0.000000	0.166667	0.477831
32	Remscheid	North Rhine-Westphalia	0.031107	0.379806	0.647059	0.885058	0.000000	0.041667	0.472146
49	Bremen	Bremen	0.158369	0.462577	1.000000	0.848614	0.200000	0.208333	0.463978
50	Bremerhaven	Bremen	0.032393	0.339751	1.000000	0.848614	0.000000	0.083333	0.115034

5. **Clustering the German cities using Unsupervised Machine Learning**

Without taking in consideration, for the moment, the Score index, let's use the following parameters to build a famous Machine Learning model for clustering, the KMeans model.

Just to illustrate how it works, let's say we want to divide the data in 7 clusters. Now we'll see how simple it is to run such a model.

```python
from sklearn.cluster import KMeans
```

```python
# set number of clusters
clusters = 7

# run k-means clustering
kmeans = KMeans(n_clusters=clusters, random_state=0).fit(df_for_clustering)

# check cluster labels generated for each row in the dataframe
kmeans.labels_[0:10]
```

```
array([4, 1, 2, 0, 0, 0, 6, 0, 0, 0])
```

As we can see, once we have defined the number of clusters, we can just pass this value as input parameter for the KMeans model, along with the data needed to train the model.
In this case, the Dataframe *df_for_clustering* is composed in this way:

City	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès
Berlin	1.000000	0.404815	0.784314	0.807461	0.266667	0.375000
Hamburg	0.507782	0.541447	0.617647	1.000000	0.533333	0.250000
Munich	0.412036	0.592048	0.264706	0.983825	0.600000	1.000000
Nuremberg	0.144878	0.425219	0.264706	0.983825	0.200000	0.375000
Augsburg	0.081356	0.423356	0.264706	0.983825	0.333333	0.208333

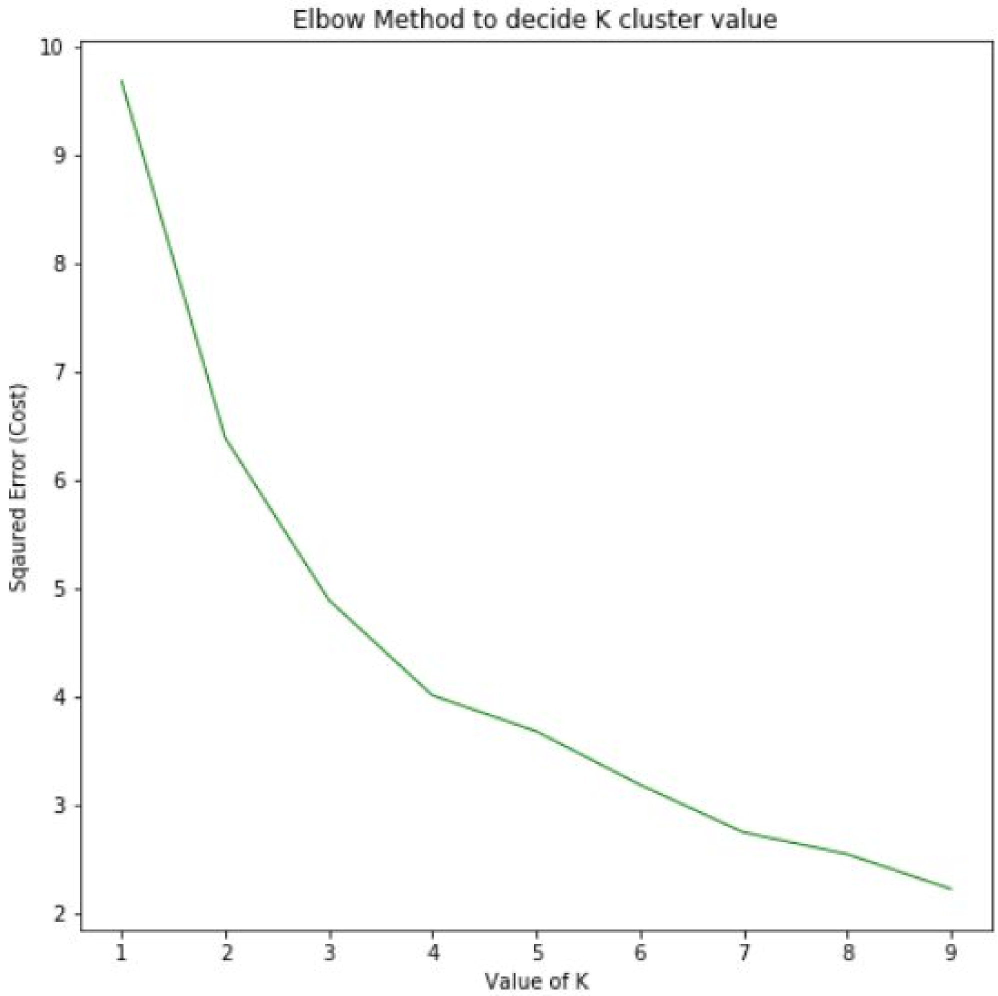
As we can notice, there is no more a 'State' column, nor the 'Score' column, and 'City' has become the index of the table (not a column anymore).

Once invoked, the 'fit' method clusters every row in the Dataframe into one of the 7 clusters.
If we take a look at the output, we can see how the algorithm clustered the first 10 cities in the dataset.

However, one of the most characteristic problems of the KMeans clustering is to find the appropriate value for the *n_clusters* parameter.
In the above example we tried '7', but we need to find a value that minimize the squared error for clustering, while being enough small to have a meaningful meaning for the problem (i.e. having 68 different clusters for 68 cities doesn't provide any value).

To address this issue, we train the model using 10 different values for *n_clusters* starting from 1 to 10.

Then we plot the squared errors of each run against the value of K and use the Elbow method for picking the best value of K:



As we can see from the image, it's not immediate to pick the best value.
We can see that both 4 and 7 are good values (elbow points), but since having 7 clusters, for this given problem, would not provide any more meaningful information, we will stick with K = *n_clusters* = 4.

Now let's run the model again with 4 clusters, and let's add to the Dataframe a column 'Cluster Labels' that tell us, for each city, the cluster it has fallen into.

```
OPTIMAL_CLUSTERS = 4

# run k-means clustering
kmeans = KMeans(n_clusters=OPTIMAL_CLUSTERS, random_state=0).fit(df_for_clustering)

df_for_clustering.insert(0, 'Cluster Labels', kmeans.labels_)
df_for_clustering.reset_index(inplace=True)
df_for_clustering.head()
```

	City	Cluster Labels	Population	GDP	Unemployment	Avg Income	Num of IT Restaurants	Num of Cafès
0	Berlin	2	1.000000	0.404815	0.784314	0.807461	0.266667	0.375000
1	Hamburg	2	0.507782	0.541447	0.617647	1.000000	0.533333	0.250000
2	Munich	3	0.412036	0.592048	0.264706	0.983825	0.600000	1.000000
3	Nuremberg	1	0.144878	0.425219	0.264706	0.983825	0.200000	0.375000
4	Augsburg	1	0.081356	0.423356	0.264706	0.983825	0.333333	0.208333

As we can notice, Berlin and Hamburg have fallen into Cluster 2, while Munich in Cluster 3 and Nuremberg and Augsburg in Cluster 1.

Results

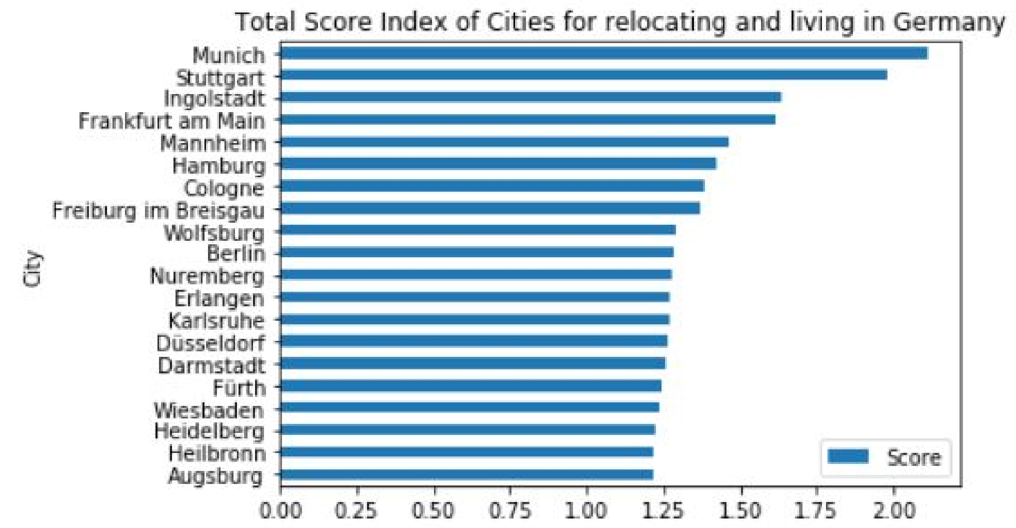
Let's make a small recap.

In the methodology section, we've calculated the score index of each candidate city, and the cluster into each city has fallen.

Let's analyze these results more thoroughly.

1. Score Index Parameter

   Let's build a bar chart to better show the results obtained calculating the Score Index Parameter:



   Only the best 20 cities are shown, just to read the chart easier.

   Munich is, predictively enough, on top of the chart and is the only city to have a index above 2.
   Another very good city to relocate to, based on Andrea's criteria, is Stuttgart, which comes just after Munich, in the 2nd place.
   Ingolstadt and Frankfurt am Main are very close by, with the first being 3rd and the latter 4th on the chart.
   Berlin is considerably away from the top cities, being only number 10 in this list, with a Score index slightly above 1.25

2. KMeans Clustering

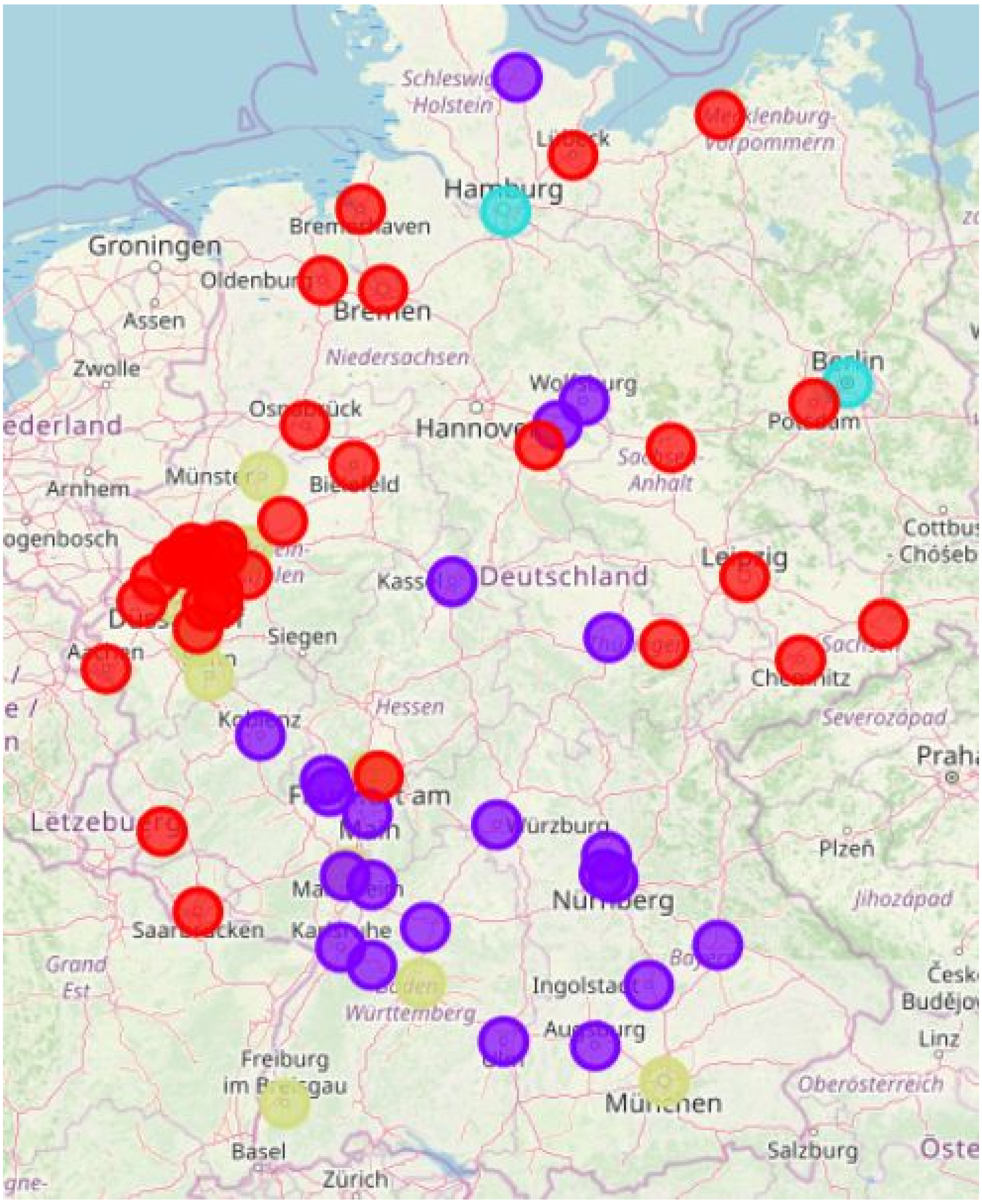
   Using the KMeans Clustering model, with 4 as the number of clusters, we’ve split the cities into the following 4 categories.

   Splitted cities:

Cluster Labels	City
0	['Essen', 'Duisburg', 'Bochum', 'Wuppertal', 'Bielefeld', 'Gelsenkirchen', 'Mönchengladbach', 'Aachen', 'Krefeld', 'Oberhausen', 'Hagen', 'Hamm', 'Mülheim an der Ruhr', 'Leverkusen', 'Solingen', 'Herne', 'Bottrop', 'Remscheid', 'Offenbach am Main', 'Leipzig', 'Dresden', 'Chemnitz', 'Bremen', 'Bremerhaven', 'Oldenburg', 'Osnabrück', 'Salzgitter', 'Lübeck', 'Magdeburg', 'Jena', 'Trier', 'Rostock', 'Saarbrücken', 'Potsdam']
1	['Nuremberg', 'Augsburg', 'Regensburg', 'Ingolstadt', 'Würzburg', 'Fürth', 'Erlangen', 'Wiesbaden', 'Kassel', 'Darmstadt', 'Karlsruhe', 'Heidelberg', 'Ulm', 'Heilbronn', 'Pforzheim', 'Braunschweig', 'Wolfsburg', 'Kiel', 'Erfurt', 'Mainz', 'Ludwigshafen am Rhein', 'Koblenz']
2	['Berlin', 'Hamburg']
3	['Munich', 'Cologne', 'Düsseldorf', 'Dortmund', 'Bonn', 'Münster', 'Frankfurt am Main', 'Stuttgart', 'Mannheim', 'Freiburg im Breisgau']

   Using the *folium* module, we’ve built a map of Germany showing each cluster with a different color.

   Cluster 0 : Red
   Cluster 1: Purple
   Cluster 2: Teal
   Cluster 3: Yellow



3. Comparing the results

   Comparing the bar chart and the map, we can definitely see a relationship between the 2 results.
   For example, as many as 7 cities out of the 10 in the yellow cluster appear also in the best 9 cities calculated using the score index!
   That's a great indicator that both methods solved well this particular problem, even using different approaches.

Discussion

Evaluating the best cities to relocate to we considered only the requirements given to us by our client, Andrea.
In order to make even more precise considerations, we could use and elaborate more data. For example, we could use the fact that he’s an IT consultant and we would have consider the unemployment rate and the average net income for that specific job only.
However this data was not easy to gather, and we didn’t find any real and complete source on the Web.

Moreover, for future direction, we can decide to use other clustering models other than the KMeans approach we’ve taken here.

Conclusion

The business problem was very challenging.
Let's make a recap of what we accomplished.
Firstly we gathered some data related to the biggest german cities with > 100'000 citizens from Wikipedia; this data included exact population, GDP per state, Average Employer Net Income per year and Unemployment Rate.
Then we obtained the number of venues of interest for our client for each city, using the Foursquare API.
We grouped these observations in one simple Dataframe, and then we normalized the data using the Simple Feature Scaling approach.
Then we used 2 methods to predict the best cities in which Andrea can relocate to:

- The Score Index
- KMeans Clustering model

Both methods showed that Munich and Stuttgart are the best cities, since they have the highest Score Index and they are grouped together in the same cluster by the KMeans model.