Data Science IBM Specialization

Coursera

CITIES CLASSIFICATION

Analysis And Clustering
Of The Cities Around The World

Lucas Daniel Tomasini

November 21, 2018

Content Table

Introduction – Business Problem 3
Data Set and Cleaning...3

Introduction – Business Problem

The problem that we are interested in here is the clustering of different cities around the world that share similar characteristics. More specifically, it is about categorizing cities according to different types of tourist activities that can be carried out there. For example, there may be cities full of historic sites, cities surrounded by natural landscapes (mountains, rivers), cities with adventure tourism activities (mountaineering, skiing, snorkeling), cities with a lot of musical culture, cities with plenty of paradisiacal beaches or cities with great nightlife.

This study is meant to help tourism agencies to build not only standard travel packs, but also especially custom-made packages adapted to the needs and requirements of the customer.

Data Set and Cleaning

For this study, a sample of 53 cities with varied characteristics was selected. The first step was to find the most relevant venues belonging to a given category in every city. To that end the Foursquare API was used with the explore functionality. One request to that end-point was run per city and per category, with the following parameters:

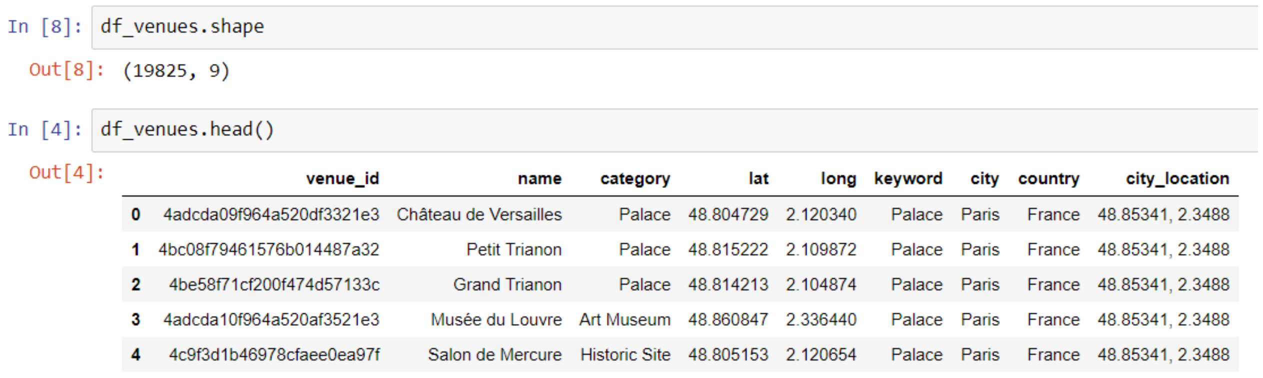
> *near*: a string naming a place in the world (e.g. Berlin, Germany). We used this parameter instead of ll (latitude, longitude)
>
> *query*: a term to be searched against a venue's tips, category, etc. For example, "Historic Site", or "Beach". Here we specify the category of the venues we want to search.
>
> *time = "any"* & *day ="'any"*: we passed "any" to retrieve results for any time of day and any day of the week. Omitting these parameters returns results targeted to the current time of day or the current day of week. And we want this analysis not to be time dependent.

We did not specify a radius, so as a suggested one be used based on the density of venues in the area, because we do not know in advance how large the city and its surroundings can be.

After doing a little research on the existing categories in the Foursquare platform, we came up with the following 25 list items for this study:

> Palace – Church – History Museum – Art Museum – Art Gallery – Theater – Restaurant – Dancing – Beer Bar – Wine Bar – Cocktail – Trail – Mountain – Lake – Beach – Park – Concert Hall – Music & Shows – Stadium – Ski Area – National Park – River – Gourmet – Historic Site – Snorkeling

We executed a total of 53 x 25 = 1325 requests to API to get the most relevant venues per city per category. We stored all this information into a data frame, getting a total of 19825 venues, as shown below:

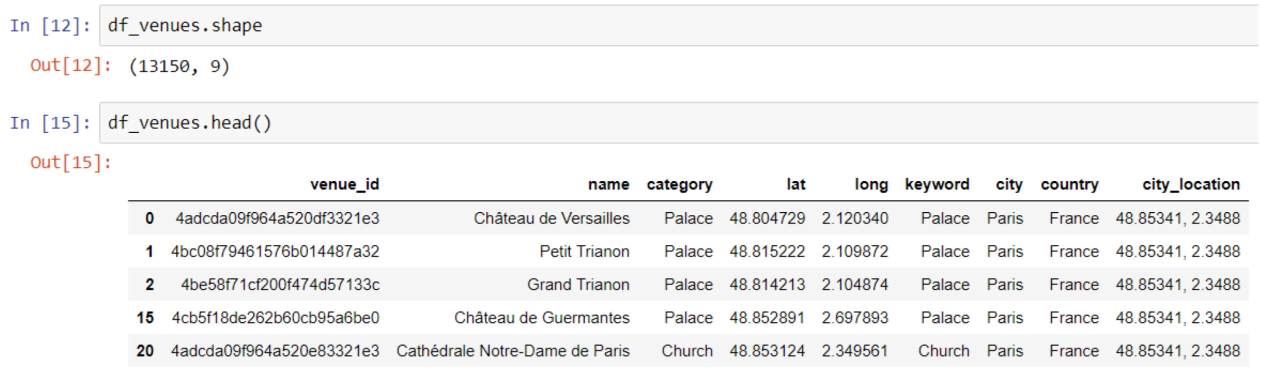


```
In [8]: df_venues.shape
Out[8]: (19825, 9)
```

```
In [4]: df_venues.head()
Out[4]:
```

	venue_id	name	category	lat	long	keyword	city	country	city_location
0	4adcda09f964a520df3321e3	Château de Versailles	Palace	48.804729	2.120340	Palace	Paris	France	48.85341, 2.3488
1	4bc08f79461576b014487a32	Petit Trianon	Palace	48.815222	2.109872	Palace	Paris	France	48.85341, 2.3488
2	4be58f71cf200f474d57133c	Grand Trianon	Palace	48.814213	2.104874	Palace	Paris	France	48.85341, 2.3488
3	4adcda10f964a520af3521e3	Musée du Louvre	Art Museum	48.860847	2.336440	Palace	Paris	France	48.85341, 2.3488
4	4c9f3d1b46978cfaee0ea97f	Salon de Mercure	Historic Site	48.805153	2.120654	Palace	Paris	France	48.85341, 2.3488

As you can see from this table, the keyword column is the query term used for the search and the category column is the actual category of the venue as it appears in Foursquare. We want to keep only the venues whose categories match or be related to the search keyword (e.g. we do not want a historic site when asked for a palace, because historic sites will be collected with another keyword: "Historic Site"). After this filtering we keep 13150 venues:



```
In [12]: df_venues.shape
Out[12]: (13150, 9)
```

```
In [15]: df_venues.head()
Out[15]:
```

	venue_id	name	category	lat	long	keyword	city	country	city_location
0	4adcda09f964a520df3321e3	Château de Versailles	Palace	48.804729	2.120340	Palace	Paris	France	48.85341, 2.3488
1	4bc08f79461576b014487a32	Petit Trianon	Palace	48.815222	2.109872	Palace	Paris	France	48.85341, 2.3488
2	4be58f71cf200f474d57133c	Grand Trianon	Palace	48.814213	2.104874	Palace	Paris	France	48.85341, 2.3488
15	4cb5f18de262b60cb95a6be0	Château de Guermantes	Palace	48.852891	2.697893	Palace	Paris	France	48.85341, 2.3488
20	4adcda09f964a520e83321e3	Cathédrale Notre-Dame de Paris	Church	48.853124	2.349561	Church	Paris	France	48.85341, 2.3488

We need to be able to compare the importance of the different categories between each city. That is why we need a set of parameters that can help us determine how important a venue of a given category is compared to others from the same category. The Foursquare API has an endpoint named *venues* that gives us the full details about a venue including rating, likes, tips, etc. The second step in the data collection is to obtain this information for every venue from the previous step. One request per venue was made to *venues* endpoint with the following parameters:

venue_id: ID of the venue to retrieve

The information extracted from the response was:

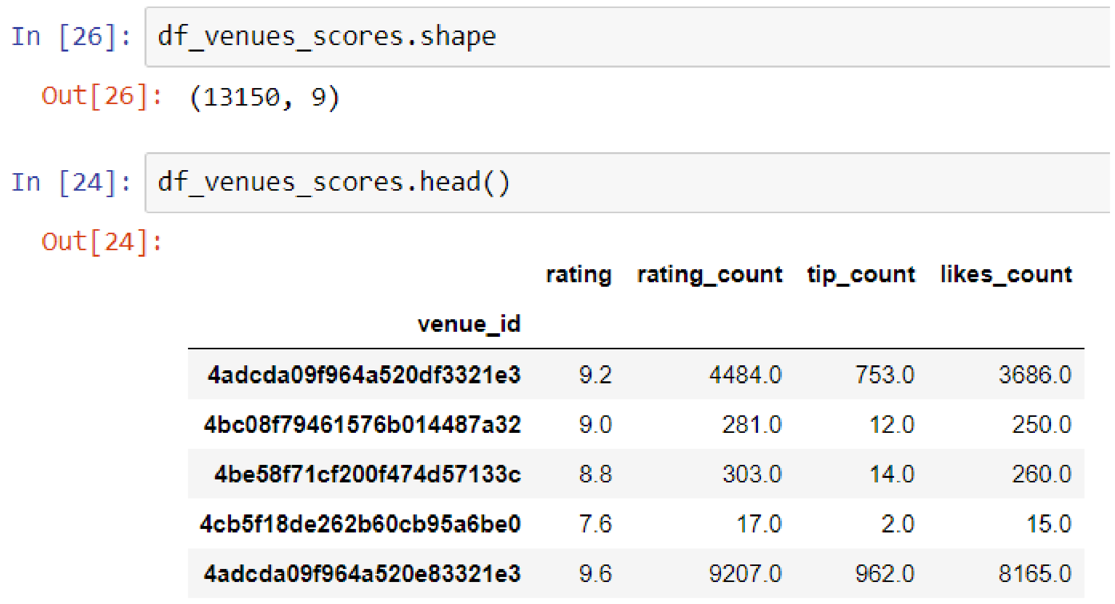
rating: numerical rating of the venue (0 through 10). Not all venues will have a rating.

ratingSignals: The count of users who have rated this this venue

likes: The count of users who have liked this venue

tipCount: Contains the total count of tips.

This is a premium request, therefore several days and five different accounts were needed to execute all of them. The collected data was store in a second data frame:

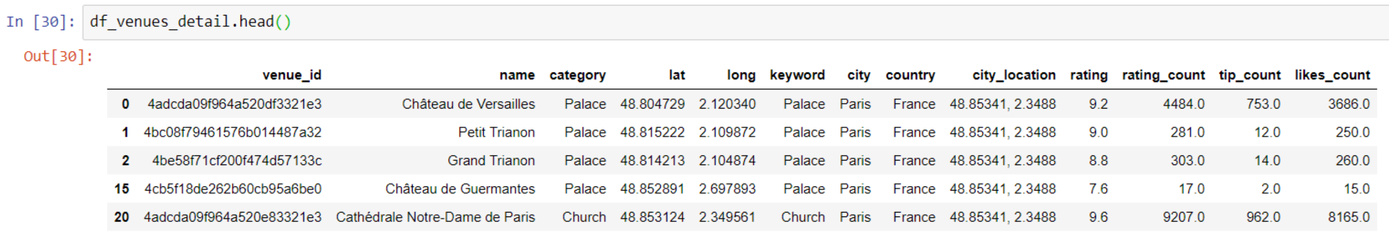


```
In [26]: df_venues_scores.shape
Out[26]: (13150, 9)
```

```
In [24]: df_venues_scores.head()
Out[24]:
```

venue_id	rating	rating_count	tip_count	likes_count
4adcda09f964a520df3321e3	9.2	4484.0	753.0	3686.0
4bc08f79461576b014487a32	9.0	281.0	12.0	250.0
4be58f71cf200f474d57133c	8.8	303.0	14.0	260.0
4cb5f18de262b60cb95a6be0	7.6	17.0	2.0	15.0
4adcda09f964a520e83321e3	9.6	9207.0	962.0	8165.0

The third step was to join the two data frames into one, getting a complete list of venues with the details. Then we removed all the venues which had not a rating:



```
In [30]: df_venues_detail.head()
Out[30]:
```

	venue_id	name	category	lat	long	keyword	city	country	city_location	rating	rating_count	tip_count	likes_count
0	4adcda09f964a520df3321e3	Château de Versailles	Palace	48.804729	2.120340	Palace	Paris	France	48.85341, 2.3488	9.2	4484.0	753.0	3686.0
1	4bc08f79461576b014487a32	Petit Trianon	Palace	48.815222	2.109872	Palace	Paris	France	48.85341, 2.3488	9.0	281.0	12.0	250.0
2	4be58f71cf200f474d57133c	Grand Trianon	Palace	48.814213	2.104874	Palace	Paris	France	48.85341, 2.3488	8.8	303.0	14.0	260.0
15	4cb5f18de262b60cb95a6be0	Château de Guermantes	Palace	48.852891	2.697893	Palace	Paris	France	48.85341, 2.3488	7.6	17.0	2.0	15.0
20	4adcda09f964a520e83321e3	Cathédrale Notre-Dame de Paris	Church	48.853124	2.349561	Church	Paris	France	48.85341, 2.3488	9.6	9207.0	962.0	8165.0

The *df_venues_detail* data frame will be used as the input data for the next section, where we will analyze and transform this data in a way that best suits the subsequent analysis.