Top Swiss city for young

Giovanni Sardelli

27 February 2020

Contents

1 Introduction

The Swiss territory is heterogeneous, each area has its own characteristics and specificities.It is sufficient to note what the official languages are to realize how varied the Swiss population is: german, french, italian, romansh. The differences between cantons are due to geographical and historical-cultural factors. Being placed on the mountain, on the shore of a lake or along the course of a river has certainly influenced the habits of the inhabitants; as well as, going through history over time, being subjected to the Habsburg or French domination is permeated in what have become the cultural traditions of the various areas.

Young people who intend to enroll in a university to complete their preparation often have to choose which city will welcome and prepare them for the future challenges that will arise. It is very important that the professional figure with whom they will enter the world of work one day will be solid and valid, for this reason they must carefully choose where to complete their studies. It's a pretty hard task! What we will do is to give support to these young in the choice by analyzing the capitals of the different swiss cantons to understand which set of cities suits them best. In the analysis we will use Foursquare through which we can explore the cities and extract 100 of the top locations. For each location we will consider their own category and we will try to understand how the locations are distributed on the Swiss territory according to their category to learn some hidden patterns. We will go in search of cities full of bookstores, coffee shops, pubs, parks, ethnic restaurants etc etc.. We will privilege the most attractive cities for young people, hoping in this way to help and support them in their difficult choice.

2 Dataset

The simplemaps website provides a quite updated dataset of the capitals of almost all countries (for more details refer to the listing 6 provided in the appendix). This project will only consider the cities of the Swiss cantons. Run the following python instructions to import it into your jupiter notebook:

```
import pandas as pd # library for data analsysis
import urllib.request
from io import BytesIO
from zipfile import ZipFile
from urllib.request import urlopen
print('libs imported!')

url = 'https://simplemaps.com/static/data/world-cities/' +
      'basic/simplemaps_worldcities_basicv1.6.zip'

hdr = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64) '
       + 'AppleWebKit/537.11 (KHTML, like Gecko) '
       + 'Chrome/23.0.1271.64 Safari/537.11',
       'Accept': 'text/html,application/xhtml+xml, application/xml;q=0.9,*/*;q=0.8',
       'Accept-Charset': 'ISO-8859-1,utf-8;q=0.7,*;q=0.3',
       'Accept-Encoding': 'none',
       'Accept-Language': 'en-US,en;q=0.8',
       'Connection': 'keep-alive'}

#The assembled request
request=urllib.request.Request(url, None, headers=hdr)

zipfile = ZipFile(BytesIO(urlopen(request).read()))
files_csv = [zipfile.open(file, 'r')
             for file in zipfile.namelist() if file.endswith(('.csv'))]
df_capitals = pd.concat([pd.read_csv(file_csv) for file_csv in files_csv])
zipfile.close()

print('\ndf_capitals shape: {}\n'.format(df_capitals.shape))
df_capitals.head(10)
```

Listing 1: dataset loading

The dataset under consideration is as follows:

	city	city-ascii	lat	lng	country
iso2	iso3	admin-name	capital	population	id
828	Kandahār	Kandahar	31.6100	65.6949	Afghanistan
AF	AFG	Kandahār	admin	715542	1004003059
4923	Ǫalāt	Qalat	32.1123	66.8868	Afghanistan
AF	AFG	Zābul	admin	12191.0	1004016690
6506	Sar-e Pul	Sar-e Pul	36.2154	65.9325	Afghanistan
AF	AFG	Sar-e Pul	admin	NaN	1004047427
3813	Pul-e Khumrī	Pul-e Khumri	35.9511	68.7011	Afghanistan
AF	AFG	Baghlān	admin	56369.0	1004123527
5141	Mamūd-e Rāqī	Mahmud-e Raqi	35.0167	69.3333	Afghanistan
AF	AFG	Kāpīsā	admin	7407.0	1004151943
2694	Ghaznī	Ghazni	33.5633	68.4178	Afghanistan
AF	AFG	Ghaznī	admin	141000.0	1004167490

There are 11 columns and 15,493 rows. Let's look at the columns:

- city: city name
- city ascii: city name in ascii encoding
- lat: city latitude
- lng: city longitude
- country: the name of city's country
- iso2: iso code of the country (two characters)
- iso3: iso code of the country (three characters)
- admi name: The name of the highest level administration region of the city
- capital: Blank string if not a capital, otherwise:
  - primary - country's capital
  - admin - first-level admin capital
  - minor - lower-level admin capital
- population: an estimate of the city's urban population
- id: a 10-digit unique id of the city

3 Methodology

The proposed solution consists of several steps at the end of which a set of preferred cities will be produced. Let's start analyzing the available dataset: in the previous paragraph we filtered only Swiss cities from the list, proceed by cleaning the dataset from incomplete data deleting rows with no reference about the amount of its population and lastly sort by population. The resulting set consists of twenty-eight cities (see table **??**).

In order to use Foursquare, we need to retrieve the geographic coordinates of each city: Google geocoders api are useful in this task. Below we provide an example of a function that can be used for their invocation: the only necessary data are the name of the city and the relative country of wich we want to find out the coordinates.

```
def geocoder_google(city= 'Geneva', country='Switzerland'):

    lat_lng_coords = None  # initialize your variable to None
    i = 0
    attempts = 3

    # loop until you get the coordinates
    while(lat_lng_coords is None and i < attempts):
        address = '{}, {}'.format(city, country)
        g = geocoder.google(address, key=GOOGLE_API_KEY)
        lat_lng_coords = g.latlng
        i = i + 1
        #print('city', city, lat_lng_coords)
    return lat_lng_coords
```

Listing 2: Google geocoder api

As shown in the listing above, due to the low reliability of the API, to prevent the lack of coordinates we have inserted a mechanism to make a maximum of three attempts in case of unavailability of the data.

	city	country	iso2	capital	population
0	Geneva	Switzerland	CH	admin	1240000.0
1	Zürich	Switzerland	CH	admin	1108000.0
3	Basel	Switzerland	CH	admin	830000.0
2	Bern	Switzerland	CH	primary	275329.0
4	Lausanne	Switzerland	CH	admin	265702.0
5	Lucerne	Switzerland	CH	admin	250000.0
6	Lugano	Switzerland	CH		105388.0
7	Biel/Bienne	Switzerland	CH		78708.0
8	Sankt Gallen	Switzerland	CH	admin	70572.0
9	Chur	Switzerland	CH	admin	38293.0
25	Schaffhausen	Switzerland	CH		33863.0
10	Fribourg	Switzerland	CH	admin	32827.0
11	Neuchâtel	Switzerland	CH	admin	31270.0
26	Sion	Switzerland	CH		28045.0
12	Zug	Switzerland	CH	admin	23435.0
27	Frauenfeld	Switzerland	CH		21979.0
28	Bellinzona	Switzerland	CH		16572.0
29	Aarau	Switzerland	CH		15501.0
30	Herisau	Switzerland	CH		15438.0
13	Solothurn	Switzerland	CH		14853.0
14	Schwyz	Switzerland	CH	admin	14177.0
15	Liestal	Switzerland	CH	admin	12832.0
31	Delémont	Switzerland	CH		11315.0
16	Sarnen	Switzerland	CH	admin	9410.0
17	Altdorf	Switzerland	CH	admin	8678.0
18	Stans	Switzerland	CH	admin	7475.0
19	Glarus	Switzerland	CH	admin	5681.0
20	Appenzell	Switzerland	CH	admin	5649.0

Table 1: sorted dateset of swiss city table

```
lambda_geocoder_google = lambda x: geocoder_google(x['city'], x['country'])
df_city_sorted['coords'] = df_city_sorted.apply(lambda_geocoder_google, axis=1)
```

Listing 3: Google geocoder api invocation

All the necessary data are now available: proceed with the exploration of cities of interest to recover the first 100 venues. Foursquare provides a large set of data for each venue, what we will consider are the venue name, its geographic coordinates and its category.

```
# explore first n venues near the city, coords=[x, y]
def foursquare_explore_venues(cities, coordinates, radius=500):

```

```
    venues_list=[]
    for city, coords in zip(cities, coordinates):
        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret
    ={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            coords[0],
            coords[1],
            radius,
            LIMIT)

        # make the GET request
        results = requests.get(url).json()["response"]['groups'][0]['items']

        # return only relevant information for each nearby venue
        venues_list.append([(
            city,
            coords[0],
            coords[1],
            v['venue']['name'],
            v['venue']['location']['lat'],
            v['venue']['location']['lng'],
            v['venue']['categories'][0]['name']) for v in results])

    nearby_venues =
    pd.DataFrame([item for venue_list in venues_list for item in venue_list])

    nearby_venues.columns = ['City Name',
                             'City Latitude',
                             'City Longitude',
                             'Venue',
                             'Venue Latitude',
                             'Venue Longitude',
                             'Venue Category']

    return(nearby_venues)

df_venues = foursquare_explore_venues(df_city_sorted['city'], df_city_sorted['coords'])
df_venues.head()
```

Listing 4: Foursquare api invocation

The following table shows an extract of the enriched dataset after the exploration of venues carried out with the Foursquare api.

City Name	Venue	Venue Latitude	Venue Longitude	Venue Category
Geneva	Mövenpick Boutique	46.204839	6.145769	Ice Cream Shop
Geneva	Payot	46.203727	6.144584	Bookstore
Geneva	Brasserie Lipp	46.203286	6.144775	French Restaurant

Table 2: extract of the enriched dataset

Let's check how many venues were returned for each city and exclude from the dataset those cities for which Foursquare does not have a sufficient number of data: we only consider valid the cities for which at least 30 venues have been returned (table **??**).

The number of cities to be analyzed has been reduced to thirteen: the choice has narrowed a lot, but this is not enough to adequately support in the final decision. Let's check for these cities how many unique categories of places are extracted. Since we have explored no more

City Name	Count
Aarau	53
Basel	65
Bern	70
Biel/Bienne	44
Geneva	100
Lausanne	100
Lucerne	100
Lugano	54
Neuchâtel	31
Sankt Gallen	47
Schaffhausen	40
Solothurn	31
Zürich	100

Table 3: Count of venues extracted by Foursquare

than 100 venues per city, the number of categories found is high: 135.

```
print('There are {} categories.'.format(len(df_venues['Venue Category'].unique())))
#There are 135 uniques categories.
```

Listing 5: number of uniques venue categories

Let's start to cross the data. It would be interesting to know how many venues are associated with each category: let's calculate the frequency!

```
# extract the city name
df_city = df_venues[['City Name']]
df_dummies = pd.get_dummies(df_venues[['Venue Category']], prefix="", prefix_sep="")

df_analysis = # concatenate city name column to the dummy venues categories dataframe
pd.concat([df_city, df_dummies], sort=False, axis=1)

df_means_venues_categories = df_analysis.groupby('City Name').mean().reset_index()
print('df_means_venues_categories shape:', df_means_venues_categories.shape)
```

Listing 6: venues categories mean

City Name	Aarau	Basel	Bern	Biel/Bienne	Geneva	Lausanne	Lucerne	Lugano	Neuchâtel	Sankt Gallen	Schaffhausen	Solothurn	Zürich
Accessories Store	0	0	0	0	0.01	0.01	0	0	0	0	0	0	0
American Restaurant	0	0	0	0	0	0.01	0	0	0	0.0212766	0	0	0
Argentinian Restaurant	0	0	0	0	0	0	0	0	0	0	0	0	0.01
Art Gallery	0.0188679	0	0	0	0.01	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Turkish Restaurant	0	0.0153846	0	0	0	0.01	0	0	0	0	0	0	0
Vegetarian / Vegan Restaurant	0	0	0.0142857	0	0	0.01	0.01	0.0185185	0	0	0	0	0.04
Waterfront	0	0	0	0	0.01	0	0	0	0	0	0	0	0
Wine Bar	0	0.0153846	0	0	0.02	0.01	0	0	0	0	0	0	0.02
Wine Shop	0	0	0	0	0.01	0	0	0	0	0	0	0	0

135 rows × 13 columns

Now let's create the new dataframe and display the top 30 venues category for each city. The dataset built in this way will be used to increase our knowledge: let's try to have it processed by the k-means algorithm and take a look at the results.

```
def get_top_categories_venues(row, num_top_venues):
    return row.iloc[1:].sort_values(ascending=False).index.values[0:num_top_venues]

# create columns according to number of top venues
columns = get_most_common_venue_columns(groupby_column='City Name', num_top_venues=
    num_top_venues)

# create a new dataframe
df_clustering_venues = pd.DataFrame(columns=columns)
df_clustering_venues['City Name'] = df_means_venues_categories['City Name']

for ind in np.arange(df_means_venues_categories.shape[0]):
    df_clustering_venues.iloc[ind, 1:] = get_top_categories_venues(
    df_means_venues_categories.iloc[ind, :], num_top_venues)

print('df_clustering_venues shape:', df_clustering_venues.shape)

df_clustering_venues.T.head()
```

Listing 7: top 30 venues categories

For simplicity let's imagine we want to get only three clusters, let's proceed with the removal of the city column from the dataset and run k-means clustering.

```
kclusters = 3
df = df_means_venues_categories.drop('City Name', 1)
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(df)

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

# add clustering labels
df_clustering = df_clustering.drop(['Cluster Labels'], axis=1, errors='ignore')
df_clustering.insert(0, 'Cluster Labels', kmeans.labels_)

# clone initial dataset
df_merge = df_ch_it[:]
df_merge.rename(columns = {'city' : 'City Name'}, inplace=True)

# filter the city of interest
my_cities = df_city_sorted['city']
df_merge = df_merge[df_merge['City Name'].isin(my_cities)]

# merge city dataset with venue category clusters
df_merged = df_merge.join(df_clustering.set_index('City Name'), on='City Name', how='
    inner')

print('df_merged shape:', df_merged.shape)
df_merged.head()
```

Listing 8: k-means

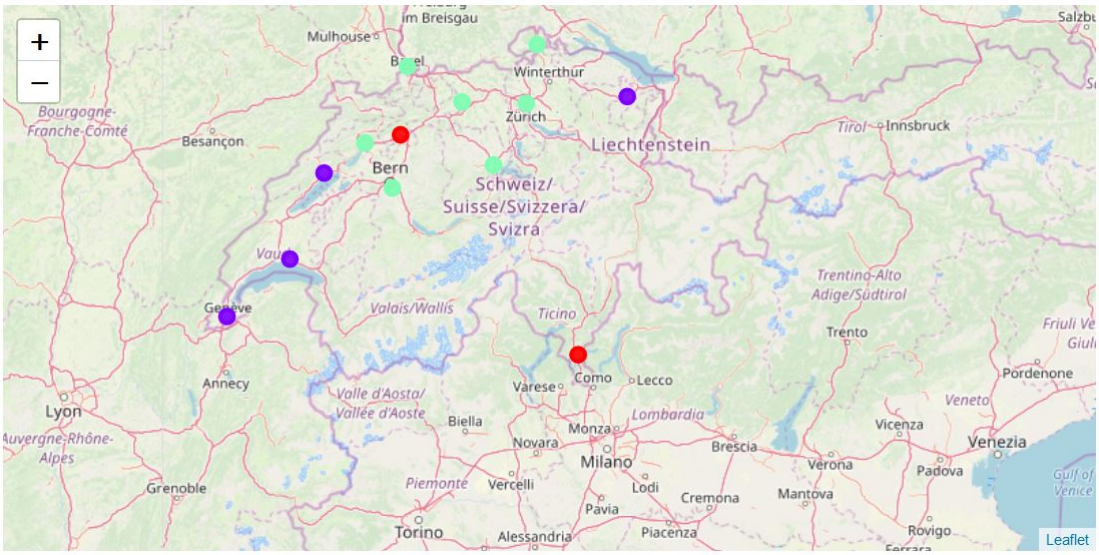
4 Results

The k-means execution produced the three clusters:

cluster 1	cluster 2	cluster 3
Zürich	Geneva	Lugano
Bern	Lausanne	Solothurn
Basel	Sankt Gallen	
Lucerne	Neuchâtel	
Biel/Bienne		
Schaffhausen		
Aarau		

Table 4: cluster

For each of the cities the 30 most relevant venues categories have been considered: showing them in tabular form is not helpful. So let's try to represent the clusters generated using folium: a python library for geospatial data visualization. Below is the list necessary to generate the map. Each city was represented by a circle colored with the color of the cluster it belongs to.




```
# create map
location=[df_merged.mean()['lat'], df_merged.mean()['lng']]
map_clusters = folium.Map(location=location, zoom_start=6.5)
    .add_to(folium.Figure(width=800, height=600))

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(df_merged['lat'], df_merged['lng'], df_merged['City
    Name'], df_merged['Cluster Labels']):
```

```
    label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
    folium.CircleMarker(
        [lat, lon]
        , radius=5
        , popup=label
        , color=rainbow[cluster-1]
        , fill=True
        , fill_color=rainbow[cluster-1]
        , fill_opacity=0.9
    ).add_to(map_clusters)

map_clusters
```

Listing 9: map of clustered city

The representation chosen helps us to understand how our cities of interest have been grouped, but does not provide us with any information about the clusters generated. We represent the categories of city venues as they have been grouped by the k-means algorithm using word clouds. Let's concatenate in one string all the categories of the extracted venues for the cluster of cities.

```
def get_cluster_words (df, cities):
    cluster_words = ''
    for index, row  in df[df['City Name'].isin(cities)][['City Name', 'Venue Category']]
     .iterrows():
        cluster_words = cluster_words + ' ' + row['City Name'] + ' ' + row['Venue
    Category']
    return cluster_words
cluster1_words_cloud= get_cluster_words(df_venues, cluster1['City Name'])
```

Listing 10: word clouds

This type of representation is much clearer and will provide more support in the final decision. As you can see in the following table, two word clouds images were generated for each cluster: the first one includes also the name of cities grouped into the cluster, in the second one the names of the cities have not been included to avoid "noise". Based on your interests, choose the word cloud you prefer, based on your choice, you can see in the previous table which cities are included in the cluster.

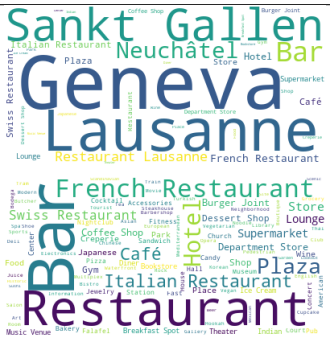
cluster 1	cluster 2	cluster 3
		

Figure 1: word clouds

5 Discussion

A future job could be to extend the selection to foreign cities to support more enterprising young people who do not exclude the possibility of moving outside the homeland.

6 Conclusion

I hope this work has been helpful and supportive in your choices. I conclude with a famous quote: “I always thought something was fundamentally wrong with the universe” [1]

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy*. San Val, 1995.

Appendices

The dataset of world city includes data on 248 countries. The included countries are as follows:


```
{
    "AD": "Andorra",
    "AE": "United Arab Emirates",
    "AF": "Afghanistan",
    "AG": "Antigua And Barbuda",
    "AI": "Anguilla",
    "AL": "Albania",
    "AM": "Armenia",
    "AO": "Angola",
    "AR": "Argentina",
    "AS": "American Samoa",
    "AT": "Austria",
    "AU": "Australia",
    "AW": "Aruba",
    "AZ": "Azerbaijan",
    "BA": "Bosnia And Herzegovina",
    "BB": "Barbados",
    "BD": "Bangladesh",
    "BE": "Belgium",
    "BF": "Burkina Faso",
    "BG": "Bulgaria",
    "BH": "Bahrain",
    "BI": "Burundi",
    "BJ": "Benin",
    "BL": "Saint Barthelemy",
    "BM": "Bermuda",
    "BN": "Brunei",
    "BO": "Bolivia",
    "BR": "Brazil",
    "BS": "Bahamas, The",
    "BT": "Bhutan",
    "BW": "Botswana",
    "BY": "Belarus",
    "BZ": "Belize",
    "CA": "Canada",
    "CC": "Cocos (Keeling) Islands",
    "CD": "Congo (Kinshasa)",
    "CF": "Central African Republic",
    "CG": "Congo (Brazzaville)",
    "CH": "Switzerland",
    "CI": "C te  D Ivoire  (Ivory Coast)",
    "CK": "Cook Islands",
    "CL": "Chile",
    "CM": "Cameroon",
    "CN": "China",
    "CO": "Colombia",
    "CR": "Costa Rica",
    "CU": "Cuba",
    "CV": "Cabo Verde",
    "CW": "Cura ao",
    "CX": "Christmas Island",
    "CY": "Cyprus",
    "CZ": "Czechia",
    "DE": "Germany",
    "DJ": "Djibouti",
    "DK": "Denmark",
    "DM": "Dominica",
    "DO": "Dominican Republic",
    "DZ": "Algeria",
    "EC": "Ecuador",
    "EE": "Estonia",
```

```
    "EG": "Egypt",
    "EH": "Western Sahara",
    "ER": "Eritrea",
    "ES": "Spain",
    "ET": "Ethiopia",
    "FI": "Finland",
    "FJ": "Fiji",
    "FK": "Falkland Islands (Islas Malvinas)",
    "FM": "Micronesia, Federated States Of",
    "FO": "Faroe Islands",
    "FR": "France",
    "GA": "Gabon",
    "GB": "United Kingdom",
    "GD": "Grenada",
    "GE": "Georgia",
    "GF": "French Guiana",
    "GG": "Guernsey",
    "GH": "Ghana",
    "GI": "Gibraltar",
    "GL": "Greenland",
    "GM": "Gambia, The",
    "GN": "Guinea",
    "GP": "Guadeloupe",
    "GQ": "Equatorial Guinea",
    "GR": "Greece",
    "GS": "South Georgia And South Sandwich Islands",
    "GT": "Guatemala",
    "GU": "Guam",
    "GW": "Guinea-Bissau",
    "GY": "Guyana",
    "HK": "Hong Kong",
    "HN": "Honduras",
    "HR": "Croatia",
    "HT": "Haiti",
    "HU": "Hungary",
    "ID": "Indonesia",
    "IE": "Ireland",
    "IL": "Israel",
    "IM": "Isle Of Man",
    "IN": "India",
    "IO": "British Indian Ocean Territory",
    "IQ": "Iraq",
    "IR": "Iran",
    "IS": "Iceland",
    "IT": "Italy",
    "JE": "Jersey",
    "JM": "Jamaica",
    "JO": "Jordan",
    "JP": "Japan",
    "KE": "Kenya",
    "KG": "Kyrgyzstan",
    "KH": "Cambodia",
    "KI": "Kiribati",
    "KM": "Comoros",
    "KN": "Saint Kitts And Nevis",
    "KP": "Korea, North",
    "KR": "Korea, South",
    "KW": "Kuwait",
    "KY": "Cayman Islands",
    "KZ": "Kazakhstan",
    "LA": "Laos",
    "LB": "Lebanon",
    "LC": "Saint Lucia",
    "LI": "Liechtenstein",
    "LK": "Sri Lanka",
    "LR": "Liberia",
```

```
    "LS": "Lesotho",
    "LT": "Lithuania",
    "LU": "Luxembourg",
    "LV": "Latvia",
    "LY": "Libya",
    "MA": "Morocco",
    "MC": "Monaco",
    "MD": "Moldova",
    "ME": "Montenegro",
    "MF": "Saint Martin",
    "MG": "Madagascar",
    "MH": "Marshall Islands",
    "MK": "Macedonia",
    "ML": "Mali",
    "MM": "Burma",
    "MN": "Mongolia",
    "MO": "Macau",
    "MP": "Northern Mariana Islands",
    "MQ": "Martinique",
    "MR": "Mauritania",
    "MS": "Montserrat",
    "MT": "Malta",
    "MU": "Mauritius",
    "MV": "Maldives",
    "MW": "Malawi",
    "MX": "Mexico",
    "MY": "Malaysia",
    "MZ": "Mozambique",
    "NA": "Namibia",
    "NC": "New Caledonia",
    "NE": "Niger",
    "NF": "Norfolk Island",
    "NG": "Nigeria",
    "NI": "Nicaragua",
    "NL": "Netherlands",
    "NO": "Norway",
    "NP": "Nepal",
    "NR": "Nauru",
    "NU": "Niue",
    "NZ": "New Zealand",
    "OM": "Oman",
    "PA": "Panama",
    "PE": "Peru",
    "PF": "French Polynesia",
    "PG": "Papua New Guinea",
    "PH": "Philippines",
    "PK": "Pakistan",
    "PL": "Poland",
    "PM": "Saint Pierre And Miquelon",
    "PN": "Pitcairn Islands",
    "PR": "Puerto Rico",
    "PT": "Portugal",
    "PW": "Palau",
    "PY": "Paraguay",
    "QA": "Qatar",
    "QZ": "Akrotiri",
    "RE": "Reunion",
    "RO": "Romania",
    "RS": "Serbia",
    "RU": "Russia",
    "RW": "Rwanda",
    "SA": "Saudi Arabia",
    "SB": "Solomon Islands",
    "SC": "Seychelles",
    "SD": "Sudan",
    "SE": "Sweden",
```

```
    "SG": "Singapore",
    "SH": "Saint Helena, Ascension, And Tristan Da Cunha",
    "SI": "Slovenia",
    "SK": "Slovakia",
    "SL": "Sierra Leone",
    "SM": "San Marino",
    "SN": "Senegal",
    "SO": "Somalia",
    "SR": "Suriname",
    "SS": "South Sudan",
    "ST": "Sao Tome And Principe",
    "SV": "El Salvador",
    "SX": "Sint Maarten",
    "SY": "Syria",
    "SZ": "Swaziland",
    "TC": "Turks And Caicos Islands",
    "TD": "Chad",
    "TF": "French Southern And Antarctic Lands",
    "TG": "Togo",
    "TH": "Thailand",
    "TJ": "Tajikistan",
    "TK": "Tokelau",
    "TL": "Timor-Leste",
    "TM": "Turkmenistan",
    "TN": "Tunisia",
    "TO": "Tonga",
    "TR": "Turkey",
    "TT": "Trinidad And Tobago",
    "TV": "Tuvalu",
    "TW": "Taiwan",
    "TZ": "Tanzania",
    "UA": "Ukraine",
    "UG": "Uganda",
    "US": "United States",
    "UY": "Uruguay",
    "UZ": "Uzbekistan",
    "VC": "Saint Vincent And The Grenadines",
    "VE": "Venezuela",
    "VG": "Virgin Islands, British",
    "VI": "Virgin Islands, U.S.",
    "VN": "Vietnam",
    "VU": "Vanuatu",
    "WF": "Wallis And Futuna",
    "WS": "Samoa",
    "XD": "Dhekelia",
    "XG": "Gaza Strip",
    "XK": "Kosovo",
    "XP": "Paracel Islands",
    "XR": "Svalbard",
    "XS": "Spratly Islands",
    "XW": "West Bank",
    "YE": "Yemen",
    "YT": "Mayotte",
    "ZA": "South Africa",
    "ZM": "Zambia",
    "ZW": "Zimbabwe"
}
```

Listing 11: countries of world city database