WEEK 2&3

Data processing (following the steps of MLsteps-Python.pdf)

Data collecting & cleaning

GeoNames is main data set which we will use to predict the location of the input city name. We do the first check about this data set.

```
     latitude      longitude
```

count 1.106199e+07 1.106199e+07

mean 2.807406e+01 1.508189e+01

std 2.405836e+01 7.962589e+01

min -9.000000e+01 -1.799836e+02

25% 1.600928e+01 -7.173488e+01

50% 3.288333e+01 1.885294e+01

75% 4.434470e+01 8.174773e+01

max 9.000000e+01 1.800000e+02

Total amount: 11061987

Deficiency amount:

country code 13767

asciiname 115

longitude 0

latitude 0

name 0

dtype: int64

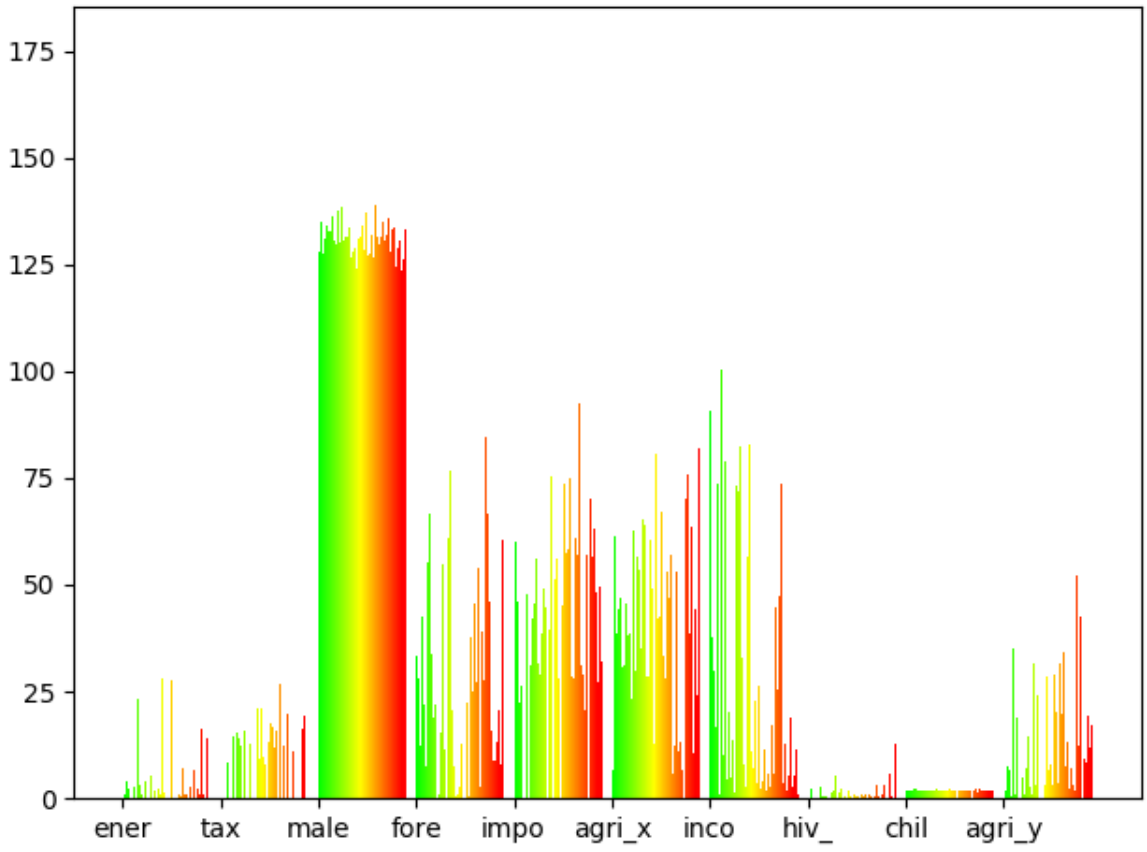
According to the rough check, only *0.1%* city lost their country code, and almost each city has its asciiname. So we can ignore these lost their cc or asciiname city to avoid the effection cased by them.

Besides our main predicting goal, the geography data set is used widely. So we choose some data set about many different kinds of countries in the world to analyze and statistics.

- **agriculture_GDP.csv**
- **agriculture_land.csv**

- **hiv_adults.csv**
- **children_per_woman.csv**
- **energy_production.csv**
- **income_pre_person.csv**
- **...**

We choose these data set from different aspects, so the analysis may be more diverse.



distribution of the variables

These data set have the common feature that the columns index are the years and the row index are the country name. So we may use the lastest year data in each data set and join them to the **country_location.csv** to get the new data set.

There many deficiency in these dataset ,and the country are only around 200, so we can't ignore them, after discussion, we decide to use lastest and existed data to fill the blank after it and fill ***0*** to the whole blank line.

> code latitude longitude ... chi agr_y gdp
>
> 12 AR -38.416097 -63.616672 ... 1.863 7.503740 -0.026220
>
> 13 AT 47.516231 14.550072 ... 1.828 1.528136 -4.135938
>
> 14 AU -25.274398 133.775136 ... 1.874 2.365473 -0.646062
>
> 15 AZ 40.143105 47.576927 ... 1.824 6.648044 7.054164
>
> 16 BA 43.915886 17.679076 ... 1.798 7.838391 -2.742990

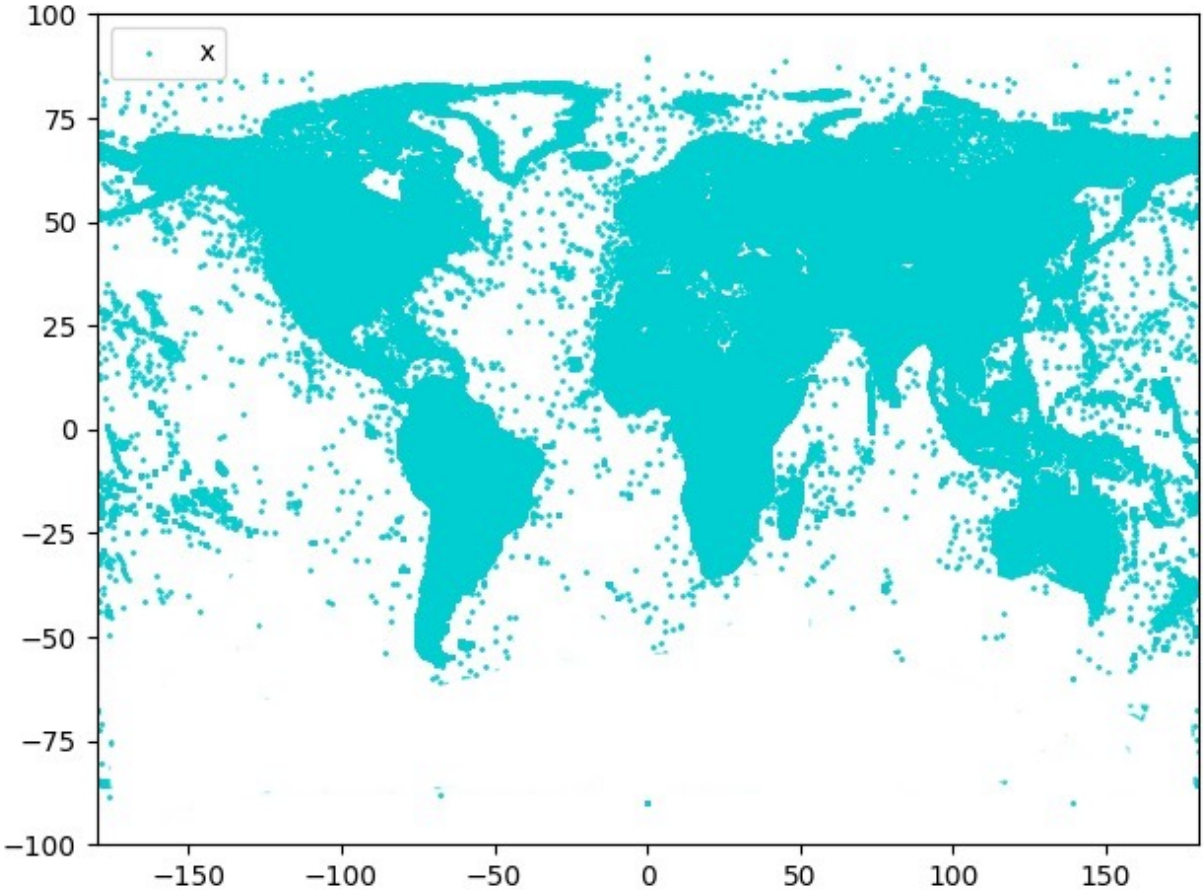
17	BB	13.193887	-59.543198	...	1.926	3.027063	-5.497907
18	BD	23.684994	90.356331	...	1.772	18.728447	4.625103	.....

the brief of the gapminder dataset

- geonameid : integer id of record in geonames database
- name : name of geographical point (utf8) varchar(200)
- asciiname : name of geographical point in plain ascii characters, varchar(200)
- alternatenames : alternatenames, comma separated, ascii names automatically transliterated, convenience attribute from alternatename table, varchar(10000)
- latitude : latitude in decimal degrees (wgs84)
- longitude : longitude in decimal degrees (wgs84)
- feature class : see http://www.geonames.org/export/codes.html, char(1)
- feature code : see http://www.geonames.org/export/codes.html, varchar(10)
- country code : ISO-3166 2-letter country code, 2 characters
- cc2 : alternate country codes, comma separated, ISO-3166 2-letter country code, 200 characters
- admin1 code : fipscode (subject to change to iso code), see exceptions below, see file admin1Codes.txt for display names of this code; varchar(20)
- admin2 code : code for the second administrative division, a county in the US, see file admin2Codes.txt; varchar(80)
- admin3 code : code for third level administrative division, varchar(20)
- admin4 code : code for fourth level administrative division, varchar(20)
- population : bigint (8 byte int)
- elevation : in meters, integer
- dem : digital elevation model, srtm3 or gtopo30, average elevation of 3''x3'' (ca 90mx90m) or 30''x30'' (ca 900mx900m) area in meters, integer. srtm processed by cgiar/ciat.
- timezone : the iana timezone id (see file timeZone.txt) varchar(40)
- modification date : date of last modification in yyyy-MM-dd format

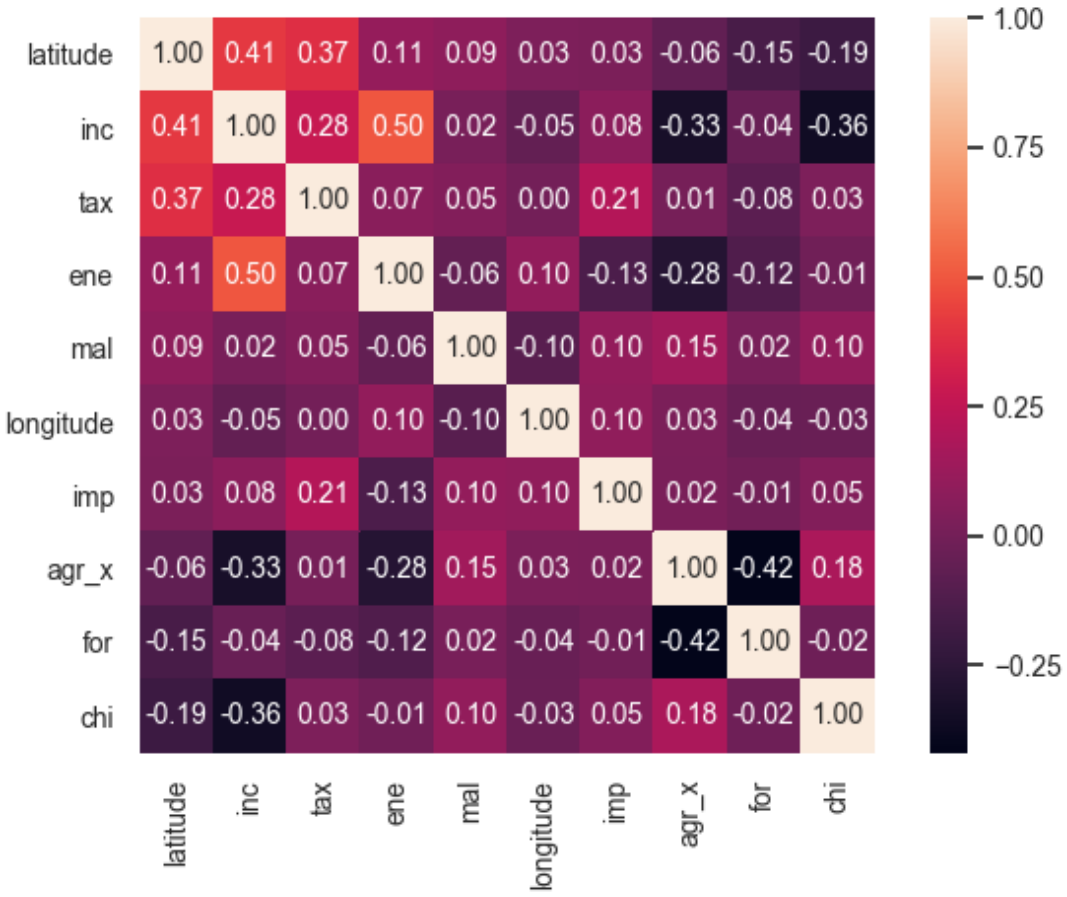
the columns of the geoname dataset

This data set's valuable columns are asciiname, latitude, longitude and country code. But asciiname and country code are all strings, we can't visualize it, so we just visilize the almost city distribution.



Data clustering

Before machine learning, we need to cluster these city by so we choose K-means model to do this job. Considering the huge difference between different country, like location, culture, population, language, GDP and so on, so we use the gapminderto do the K-means method, and we need use the ***PCA*** method to do dimension reduction analysis.



top 10 relation feature

```
data = normalize(np.array(netArr),axis=0)
pca = PCA(n_components=2)
pca.fit(data)
afterData = pca.fit_transform(data)
```

After dimension reduction, we need to decide the value of ***K*** , we use ***sum of the squared errors*** and ***Silhouette analysis*** to decide the accurate value. Here are math theories of these two method

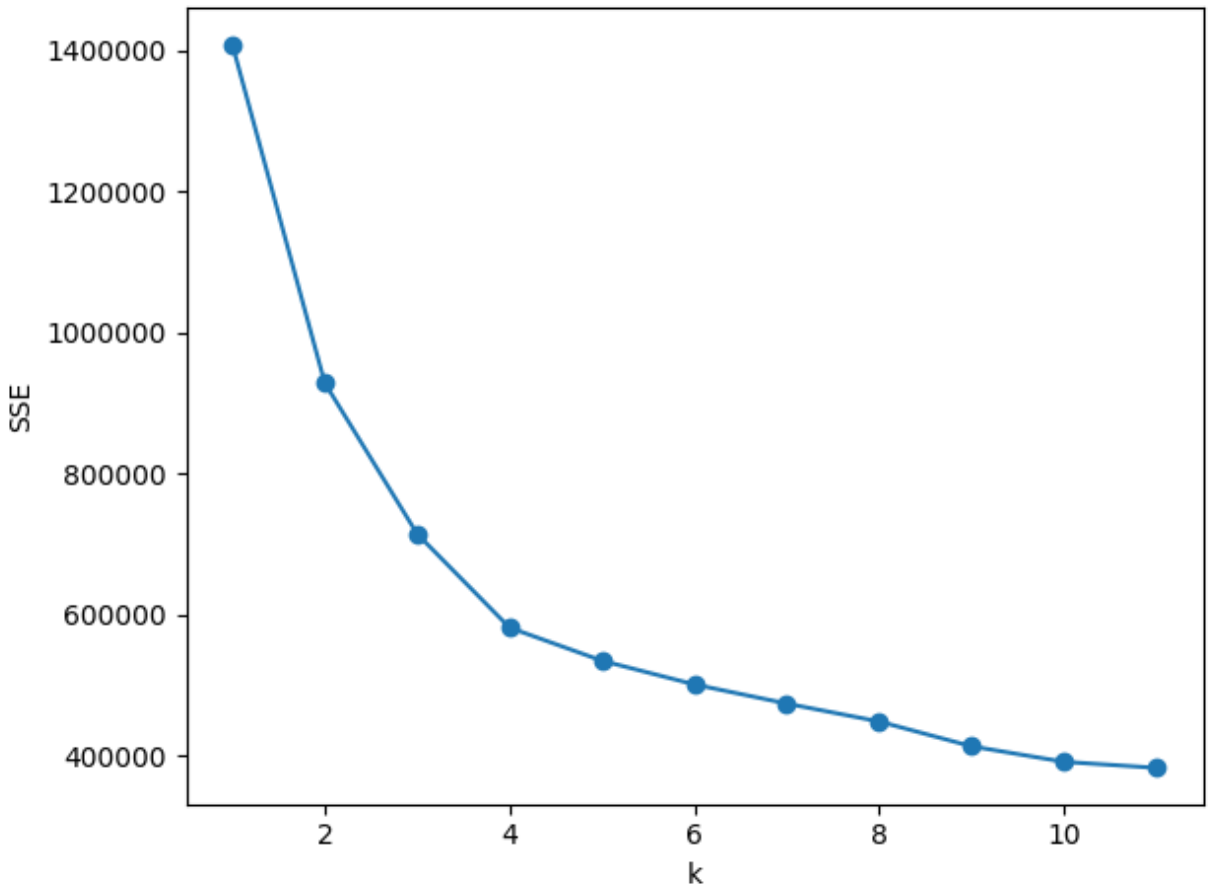
sum of the squared errors

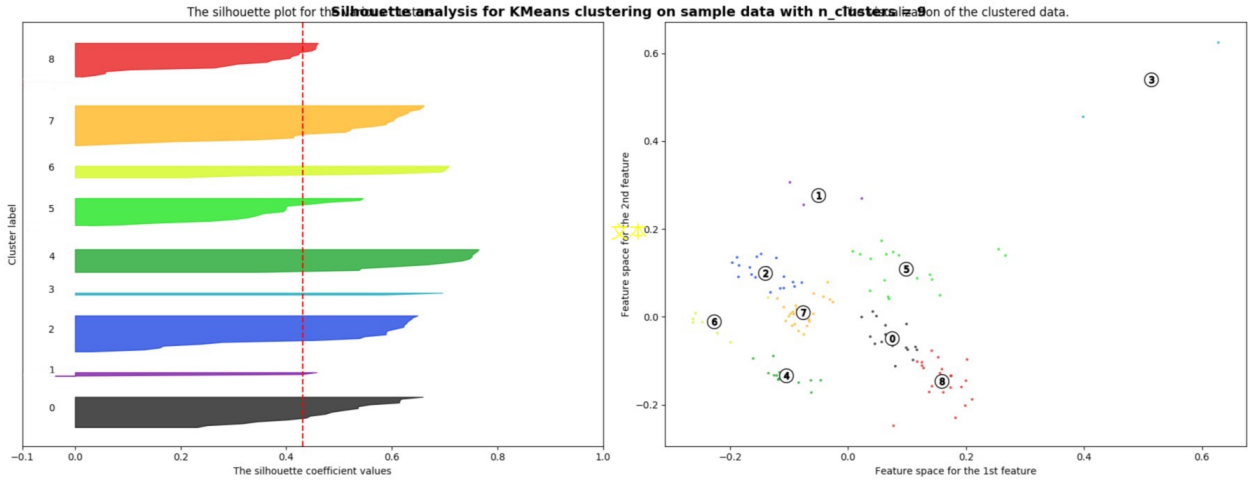
$$SSE = \sum_{i=1}^{K} \sum_{p \in Ci} |p - mi|^2$$

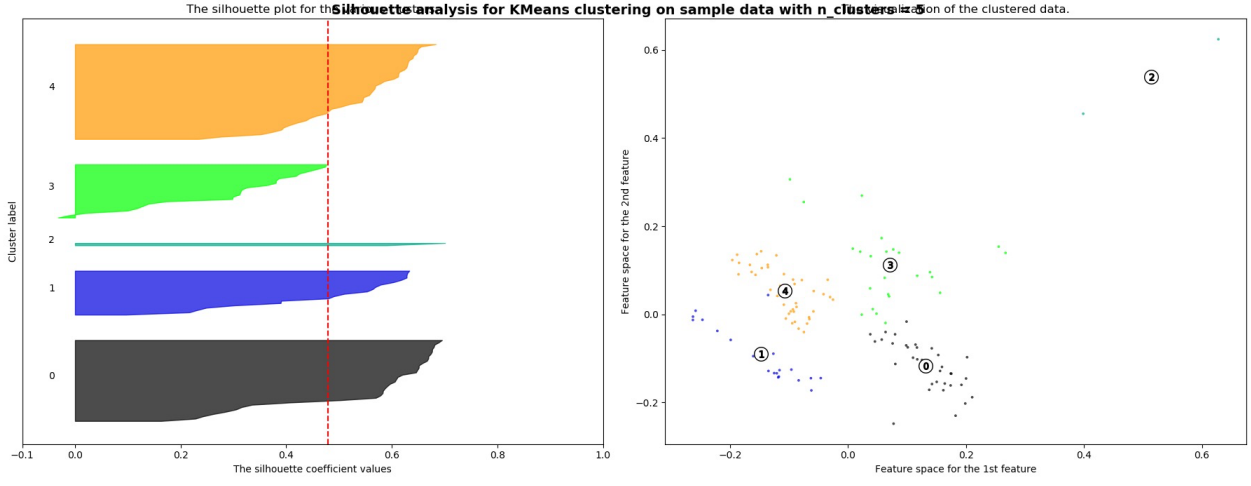
Silhouette analysis

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \qquad s(x) = \begin{cases} 1 - \dfrac{a(i)}{c(i)}, & a(i) < b(i) \\ 0, & a(i) = b(i) \\ \dfrac{a(i)}{c(i)} - 1, & a(i) > b(i) \end{cases}$$

Here are our result pictures.


SSE
k
1400000
1200000
1000000
800000
600000
400000
2
4
6
8
10


Silhouette analysis for KMeans clustering on sample data with n_clusters = 9
Cluster label
The silhouette coefficient values
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
Cluster label
The silhouette coefficient values
Feature space for the 2nd feature
Feature space for the 1st feature

> For n_clusters = 2 The average silhouette_score is : 0.3905449300354942
>
> For n_clusters = 3 The average silhouette_score is : 0.4176882525682744
>
> For n_clusters = 4 The average silhouette_score is : 0.4461314443682128
>
> For n_clusters = 5 The average silhouette_score is : 0.47883341308085464
>
> For n_clusters = 6 The average silhouette_score is : 0.4165650982422084
>
> For n_clusters = 7 The average silhouette_score is : 0.38732557724848593
>
> For n_clusters = 8 The average silhouette_score is : 0.42761880928564716
>
> For n_clusters = 9 The average silhouette_score is : 0.4719416932136283

We' ve got one ***SSE*** picture and 9 ***Silhouette*** pictures, after comparing, ***K=5*** and ***K=9*** can be choosn. But 5 may be not enough for classify these area, so we choose ***K=9*** as our value.

```
nClusters = 9
kmeans = KMeans(n_clusters=nClusters, random_state=0).fit(afterData)
clusteredArr = []
for i in range(0, nClusters):
    clusteredArr.append([])
id = 0
for country in netDict:
    clusteredTo = kmeans.predict([afterData[id]])[0]
    clusteredArr[clusteredTo].append(country)
# print("%s in Cluster %s" % (country, kmeans.predict([afterData[id]])))
    id += 1
```

After clustering , in order to be compatible with *Geonames* and *Gapminder*, we use the ***country code*** to describe these country.

> EastAsia CN HK JP KP KR LA MO TW VN
>
> S&SEAsia BD BT BN CC ID IN KH LK MM MV MY NP PH SG TH TL
>
> EnUsAuNz AU CA CX FK IM IO NZ US VG VI
>
> Latinos AG AI AR AW BB BL BO BR BZ CL CO CR CU CW DM DO EC ES GB GD GI GN GQ GT GY HN HT JM MX NI PA PE PR PT PY SR ST SV TT UY VE
>
> Arabics AE AF BH DZ EG EH IL IQ IR JO KG KW KZ LB LY OM PK PS QA SA SY TJ TM UZ YE
>
> WEurope AD AL AT BE CH DE DK FI FO FR GL GR HR IE IS IT LI LU MC MT NL NO RE RO SE SM VA
>
> EEurope AM AZ BA BG BA BY CY CZ EE GE HU LT LV MD ME MK MN PL RS RU SI SK UA XK
>
> Oceania AS BM CK FJ FM KI NR PG PW TK TO TV WS
>
> SSAfrica AO BF BI BJ BW CD CF CG CI CM CV DJ ER ET GA GH GM GW KE KM LR LS MA MG ML MR MU MW MZ NA NE NG RW SC SD SL SN SO SS SZ TD TG TN TZ UG ZA ZM ZW

Here are nine area which we clustered , so next step we need to normalize our city name to tensor.

Data normalize

To represent a single letter, we use a “one-hot vector” of size `<1 x n_letters>`. A one-hot vector is filled with 0s except for a 1 at index of the current letter, e.g. `"b" = <0 1 0 0 0 ...>`.

To make a word we join a bunch of those into a 2D matrix `<line_length x 1 x n_letters>`.

That extra 1 dimension is because PyTorch assumes everything is in batches - we’re just using a batch size of 1 here.

> "bad" may be converted to tensor like:
>
> a b c d e f ... z
>
> 0 1 0 0 0 0 ... 0
>
> 1 0 0 0 0 0 ... 0
>
> 0 0 0 1 0 0 ... 0

So we can use ***one-hot vector*** to convert our city name

```
import torch

all_letters = "abcdefghijklmnopqrstuvwxyz'`"
n_letters=len(all_letters)

def letterToIndex(letter):
 return all_letters.find(letter)

def letterToTensor(letter):
 tensor = torch.zeros(1, n_letters)
 tensor[0][letterToIndex(letter)] = 1
 return tensor

def lineToTensor(line):
 tensor = torch.zeros(len(line), 1, n_letters)
 for li, letter in enumerate(line):
     tensor[li][0][letterToIndex(letter)] = 1
 return tensor
```

Next step, we will build our recurrent neural network to analysis the data classification.