written material

going to grab this data from gh:
https://raw.githubusercontent.com/stefanbund/py3100/main/ProductList_118.csv

```
pip install plotly-geo
```

```
Collecting plotly-geo
  Downloading plotly_geo-1.0.0-py3-none-any.whl (23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23
Installing collected packages: plotly-geo
Successfully installed plotly-geo-1.0.0
```

our system depends on the panda and numpy library

```
import pandas as pd
import numpy as np
```

```
url ='https://raw.githubusercontent.com/stefanbund/py3100/main/ProductList_118.csv'
url_m ='https://raw.githubusercontent.com/stefanbund/py3100/main/matrix.csv'
```

```
df_m = pd.read_csv(url_m) #make a pandas dataframe
```

```
df_m
```

	City	1	2	3	4	5	6	7	8	9	...	32	33
0	Birmingham	8285	5343	6738	6635	5658	8118	4311	8535	3436	...	1340	6923
1	Montgomery	1287	6585	8300	8874	8208	5363	3552	3387	2765	...	4424	8813
2	Mobile	8035	5569	9492	5905	5024	1107	6937	5580	8044	...	5430	1601
3	Huntsville	6280	2841	3399	5448	6173	5451	7488	9981	5236	...	9169	7829
4	Tuscaloosa	4079	1066	3923	4177	4277	4219	9436	8160	4302	...	1556	5533
5	Hoover	9741	7377	9410	9790	8864	2522	5347	9145	8402	...	6031	7673
6	Dothan	7646	2060	4911	4976	7851	4277	7423	6183	6641	...	8253	1565
7	Auburn	4326	2659	6928	4656	1828	5199	5331	6294	3076	...	6128	3737
8	Decatur	3786	2891	8124	2469	3704	3623	2409	8287	2032	...	6622	9742
9	Madison	1934	3628	9190	3275	9344	5778	1256	3523	1781	...	6619	6128
10	Florence	8017	3187	1128	4706	9962	7547	4440	4530	9569	...	8306	1392
11	Gadsden	2290	6402	8598	7547	5158	9731	8038	4435	7357	...	4488	3591
12	Vestavia Hills	9471	9142	4419	3846	2016	5069	4853	6336	9062	...	4613	2942
13	Prattville	6039	8003	6180	4610	3548	7115	6720	8512	9954	...	8225	7278
14	Phenix City	8788	8269	6838	2863	6753	6608	4048	8774	4513	...	5704	8720
15	Alabaster	1733	9767	3274	7125	7437	5748	5399	6513	3038	...	7351	9503
16	Bessemer	6559	2453	1578	5158	3058	8075	7066	8530	8346	...	8921	3517
17	Enterprise	8436	7800	7234	5063	4274	1948	7887	6647	1320	...	4840	6309

```
df_m.columns #dimensionality of the matrix
```

```
Index(['City', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
       '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36',
       '37', '38', '39', '40', '41'],
      dtype='object')
```

df_m.columns retrieves the column labels or names of the DataFrame df_m. It provides a list of the names of the columns in the DataFrame, revealing the dimensionality or structure of the data matrix represented by df_m.

list all citys is the matrix dataframe

```
df_m['City'] #explore a Series inside the dataframe
```

```
0         Birmingham
1         Montgomery
2             Mobile
3         Huntsville
4         Tuscaloosa
5             Hoover
6             Dothan
7             Auburn
8            Decatur
9            Madison
10          Florence
11           Gadsden
12    Vestavia Hills
13        Prattville
14       Phenix City
15         Alabaster
16          Bessemer
17        Enterprise
18           Opelika
19          Homewood
20         Northport
21            Pelham
22        Trussville
23    Mountain Brook
24          Fairhope
Name: City, dtype: object
```

The code df_m['City'] extracts the column labeled 'City' from the DataFrame df_m, returning a pandas Series object containing the data in that specific column to list all the cities

investigate quartile as an analytic tool

```
df_m.dtypes
# df_m.columns
```

```
City    object
1        int64
2        int64
3        int64
4        int64
5        int64
6        int64
7        int64
8        int64
9        int64
10       int64
11       int64
12       int64
13       int64
14       int64
15       int64
```

```
16        int64
17        int64
18        int64
19        int64
20        int64
21        int64
22        int64
23        int64
24        int64
25        int64
26        int64
27        int64
28        int64
29        int64
30        int64
31        int64
32        int64
33        int64
34        int64
35        int64
36        int64
37        int64
38        int64
39        int64
40        int64
41        int64
dtype: object
```

df_m.dtypes returns a pandas Series with the data types of each column in the DataFrame df_m. It provides information about whether each column contains numerical or categorical data, and the specific data type of each.

Quantiles for each display, all stores

```
df_3 = df_m.quantile([0.25, 0.5, 0.75], numeric_only=True, axis=1)
df_3
```

	0	1	2	3	4	5	6	7	8	9	...	
0.25	3082.0	3633.0	2236.0	3473.0	3657.0	4628.0	4254.0	3588.0	3704.0	3451.0	...	3
0.50	5343.0	5431.0	5311.0	5771.0	5131.0	7588.0	5156.0	5331.0	6589.0	5875.0	...	6
0.75	7242.0	8074.0	7508.0	7935.0	7490.0	9145.0	6840.0	7606.0	8221.0	7783.0	...	7

3 rows × 25 columns

df_m.quantile([0.25, 0.5, 0.75], numeric_only=True, axis=1) computes the specified quantiles (in this case, the 25th, 50th, and 75th percentiles) along the columns (axis=1) of the DataFrame df_m.

The numeric_only=True parameter ensures that only numeric columns are considered for the calculation

per store, the quartile values

```
l = df_3.T.columns  #transpose, T
l
```

```
Float64Index([0.25, 0.5, 0.75], dtype='float64')
```

l = df_3.T.columns transposes the DataFrame df_3 using the .T attribute, and then retrieves the column labels using .columns. This operation effectively switches the rows and columns in df_3

```
df_3.T.mean()
```

```
0.25    3535.24
0.50    5826.36
0.75    7953.00
dtype: float64
```

df_3.T.mean() calculates the mean (average) for each column in the transposed DataFrame df_3. The .T attribute is used to transpose the DataFrame, and then .mean() computes the mean along the rows, providing the average value for each original column in df_3

define the global quartile boundary, per q

```
df_3.T[0.25].mean()
```

```
3535.24
```

```
df_3.T[0.5].mean()
```

```
5826.36
```

```
df_3.T[0.75].mean()
```

```
7953.0
```

```
kk = df_3.T.mean()
kk #series
```

```
0.25    3535.24
0.50    5826.36
0.75    7953.00
dtype: float64
```

kk = df_3.T.mean() calculates the mean for each column in the transposed DataFrame df_3 and assigns the result to the variable kk

what percentage of displays are at or below the 25th quartile, per store? exercise

```
# n =
((df_m.iloc[:, 1:] <= kk[0.25]).sum(axis=1) / df_m.shape[1]) * 100
# print(round(n))
```

```
0     28.571429
1     21.428571
2     38.095238
3     26.190476
4     21.428571
5     16.666667
6     19.047619
7     23.809524
8     21.428571
9     28.571429
10    26.190476
11    19.047619
12    26.190476
13    23.809524
14    28.571429
15    28.571429
16    14.285714
17    19.047619
18    28.571429
19    19.047619
20    28.571429
21    23.809524
22    33.333333
23    19.047619
24    33.333333
dtype: float64
```

calculates a percentage based on how many values in each row of the DataFrame df_m are less than or equal to the 25th percentile value (kk[0.25])

```
la = df_m['25qt'] = round(((df_m.iloc[:, 1:] <= kk[0.25]).sum(axis=1) / df_m.shape[1]
ll = df_m['50qt'] = round(((df_m.iloc[:, 1:] <= kk[0.50]).sum(axis=1) / df_m.shape[1]
lll = df_m['75qt'] = round(((df_m.iloc[:, 1:] <= kk[0.75]).sum(axis=1) / df_m.shape[1
print(la, ll, lll)
```

```
0     33.3
```

```
1     26.7
2     42.2
3     31.1
4     26.7
5     22.2
6     24.4
7     28.9
8     26.7
9     33.3
10    31.1
11    24.4
12    31.1
13    28.9
14    33.3
15    33.3
16    20.0
17    24.4
18    33.3
19    24.4
20    33.3
21    28.9
22    37.8
23    24.4
24    37.8
dtype: float64 0     57.8
1     57.8
2     62.2
3     53.3
4     62.2
5     37.8
6     57.8
7     53.3
8     48.9
9     51.1
10    51.1
11    44.4
12    55.6
13    46.7
14    51.1
15    44.4
16    48.9
17    44.4
18    57.8
19    44.4
20    55.6
21    53.3
22    51.1
23    55.6
24    68.9
dtype: float64 0     77.8
1     71.1
2     80.0
3     77.8
4     80.0
5     60.0
6     91.1
```

calculates and assigns quartile percentages to new columns ('25qt', '50qt', '75qt') in the DataFrame df_m based on conditions related to the 25th, 50th, and 75th percentiles. The resulting percentages are rounded to one decimal place and then printed

```
# df_m
```

```
end_set = ['City','25qt','50qt','75qt']
df_m[end_set]
```

	City	25qt	50qt	75qt
0	Birmingham	33.3	57.8	77.8
1	Montgomery	26.7	57.8	71.1
7	Auburn	28.9	53.3	80.0

df_m[end_set] selects and retrieves a subset of columns from the DataFrame df_m containing the specified columns in the list end_set ('City', '25qt', '50qt', '75qt'). This operation creates a new DataFrame that includes only the specified columns, allowing you to focus on and analyze the selected variables.

create a choropleth for each store

```
#choropleth:
import pandas as pd

# Create a sample dataframe
data = {'City': ['Birmingham', 'Montgomery', 'Mobile', 'Huntsville', 'Tuscaloosa', 'H
        'Zip Code': ['35201','36101','36601','35801','35401','35216','36301','36830',

df = pd.DataFrame(data)

# Create a list of zip codes
zip_codes = ['35201', '36101', '36601', '35801', '35401', '35216',
             '36301', '36830', '35601', '35756', '35630', '35901',
             '35216', '36066', '36867', '35007', '35020',
             '36330', 36801, 35209, 35473, 35124, 35173, 35213, 36532]

# Add the list of zip codes as a new column to the dataframe
# df = df.assign(Zip_Codes=zip_codes)
df_m = df_m.assign(zip=zip_codes)


print(df_m)
```

```
              City     1     2     3     4     5     6     7     8     9  ...  \
0       Birmingham  8285  5343  6738  6635  5658  8118  4311  8535  3436  ...
1       Montgomery  1287  6585  8300  8874  8208  5363  3552  3387  2765  ...
2           Mobile  8035  5569  9492  5905  5024  1107  6937  5580  8044  ...
3       Huntsville  6280  2841  3399  5448  6173  5451  7488  9981  5236  ...
4       Tuscaloosa  4079  1066  3923  4177  4277  4219  9436  8160  4302  ...
5           Hoover  9741  7377  9410  9790  8864  2522  5347  9145  8402  ...
6           Dothan  7646  2060  4911  4976  7851  4277  7423  6183  6641  ...
7           Auburn  4326  2659  6928  4656  1828  5199  5331  6294  3076  ...
8          Decatur  3786  2891  8124  2469  3704  3623  2409  8287  2032  ...
9          Madison  1934  3628  9190  3275  9344  5778  1256  3523  1781  ...
10        Florence  8017  3187  1128  4706  9962  7547  4440  4530  9569  ...
11         Gadsden  2290  6402  8598  7547  5158  9731  8038  4435  7357  ...
12  Vestavia Hills  9471  9142  4419  3846  2016  5069  4853  6336  9062  ...
13      Prattville  6039  8003  6180  4610  3548  7115  6720  8512  9954  ...
```

```
14     Phenix City  8788  8269  6838  2863  6753  6608  4048  8774  4513  ...
15       Alabaster  1733  9767  3274  7125  7437  5748  5399  6513  3038  ...
16        Bessemer  6559  2453  1578  5158  3058  8075  7066  8530  8346  ...
17      Enterprise  8436  7800  7234  5063  4274  1948  7887  6647  1320  ...
18         Opelika  9998  8953  7923  6176  4369  9503  2126  1816  9224  ...
19        Homewood  2373  7188  9880  9236  5969  9998  8703  8440  4643  ...
20       Northport  3536  9231  8651  6374  4842  5704  8484  6322  2012  ...
21          Pelham  6830  3736  2734  6443  8494  6206  7290  8518  6176  ...
22      Trussville  2794  8273  9174  2850  8351  3978  5995  4632  7693  ...
23  Mountain Brook  8433  9368  2141  2357  6566  1482  4787  3900  6615  ...
24        Fairhope  8114  1464  2811  3090  4686  7995  7676  1304  7332  ...

      36    37    38    39    40    41  25qt  50qt  75qt    zip
0   3555  1341  1756  7598  1509  1861  33.3  57.8  77.8  35201
1   2805  4601  4449  5727  2315  8822  26.7  57.8  71.1  36101
2   9807  2652  9296  2815  4886  7458  42.2  62.2  80.0  36601
3   7935  2605  9982  3338  9116  3875  31.1  53.3  77.8  35801
4   3657  2158  4469  2513  8135  6963  26.7  62.2  80.0  35401
5   9748  7224  4628  8107  6143  1671  22.2  37.8  60.0  35216
6   5650  4400  7842  4006  9335  3571  24.4  57.8  91.1  36301
7   4387  6890  2833  5083  9707  2116  28.9  53.3  80.0  36830
8   9305  6509  6848  5408  3707  8744  26.7  48.9  71.1  35601
9   1746  4470  7054  6573  3556  1374  33.3  51.1  75.6  35756
10  5929  1123  7306  8746  4000  6943  31.1  51.1  64.4  35630
11  2549  5175  5997  9608  7230  9731  24.4  44.4  68.9  35901
12  5142  9619  9601  8099  1391  6276  31.1  55.6  71.1  35216
13  1591  4401  3457  4245  4341  2573  28.9  46.7  75.6  36066
14  3520  7654  6845  7738  3828  1202  33.3  51.1  75.6  36867
15  2479  9673  7478  7207  7006  3523  33.3  44.4  84.4  35007
16  4810  7641  5365  3545  6812  9483  20.0  48.9  71.1  35020
17  3461  2640  4375  8634  4917  2830  24.4  44.4  73.3  36330
18  5191  9304  2720  3100  3912  1548  33.3  57.8  73.3  36801
19  8787  5459  8389  5242  2224  6025  24.4  44.4  68.9  35209
20  6947  5401  6681  9018  1668  8307  33.3  55.6  75.6  35473
21  2777  4045  7309  4745  4284  2640  28.9  53.3  73.3  35124
22  1650  9470  6356  4700  3344  8743  37.8  51.1  75.6  35173
23  5765  3653  5198  9266  4945  3935  24.4  55.6  71.1  35213
24  3457  4808  7227  5482  6355  4553  37.8  68.9  86.7  36532

[25 rows x 46 columns]
```

creates a DataFrame named df with columns 'City' and 'Zip Code' containing information about cities and their corresponding zip codes. It then assigns a list of zip codes to a new column named 'zip' in the DataFrame df_m, which could be useful for further analysis or visualization.

experiment with chloropleths

```
df_m.columns
```

```
Index(['City', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
       '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36',
```

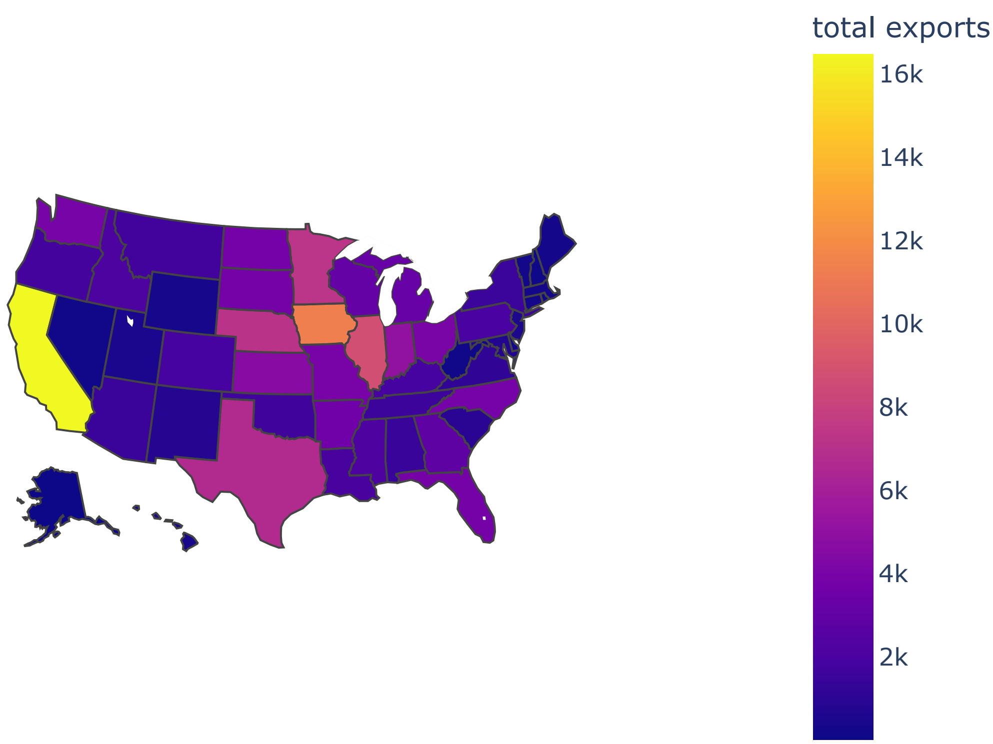
```
           '37', '38', '39', '40', '41', '25qt', '50qt', '75qt', 'zip'],
          dtype='object')
```

```
import plotly.express as px
import pandas as pd

# Load data
df_demo = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_

# Create choropleth map
fig = px.choropleth(df_demo, locations='code', locationmode='USA-states', color='tota

# Show map
fig.show()
```



uses Plotly Express to create a choropleth map visualizing total agricultural exports for each U.S. state. It loads the data from a CSV file and generates an interactive map where the color intensity represents the total exports, providing a visual representation of the geographical distribution of agricultural exports across the United States

```
df_demo
```

	code	state	category	total exports	beef	pork	poultry	dairy	fruits fresh	fr
0	AL	Alabama	state	1390.63	34.4	10.6	481.0	4.06	8.0	
1	AK	Alaska	state	13.31	0.2	0.1	0.0	0.19	0.0	
2	AZ	Arizona	state	1463.17	71.3	17.9	0.0	105.48	19.3	
3	AR	Arkansas	state	3586.02	53.2	29.4	562.9	3.53	2.2	
4	CA	California	state	16472.88	228.7	11.1	225.4	929.95	2791.8	5
5	CO	Colorado	state	1851.33	261.4	66.0	14.0	71.94	5.7	
6	CT	Connecticut	state	259.62	1.1	0.1	6.9	9.49	4.2	
7	DE	Delaware	state	282.19	0.4	0.6	114.7	2.30	0.5	
8	FL	Florida	state	3764.09	42.6	0.9	56.9	66.31	438.2	
9	GA	Georgia	state	2860.84	31.0	18.9	630.4	38.38	74.6	
10	HI	Hawaii	state	401.84	4.0	0.7	1.3	1.16	17.7	
11	ID	Idaho	state	2078.89	119.8	0.0	2.4	294.60	6.9	
12	IL	Illinois	state	8709.48	53.7	394.0	14.0	45.82	4.0	
13	IN	Indiana	state	5050.23	21.9	341.9	165.6	89.70	4.1	
14	IA	Iowa	state	11273.76	289.8	1895.6	155.6	107.00	1.0	
15	KS	Kansas	state	4589.01	659.3	179.4	6.4	65.45	1.0	
16	KY	Kentucky	state	1889.15	54.8	34.2	151.3	28.27	2.1	
17	LA	Louisiana	state	1914.23	19.8	0.8	77.2	6.02	5.7	
18	ME	Maine	state	278.37	1.4	0.5	10.4	16.18	16.6	
19	MD	Maryland	state	692.75	5.6	3.1	127.0	24.81	4.1	
20	MA	Massachusetts	state	248.65	0.6	0.5	0.6	5.81	25.8	
21	MI	Michigan	state	3164.16	37.7	118.1	32.6	214.82	82.3	
22	MN	Minnesota	state	7192.33	112.3	740.4	189.2	218.05	2.5	

this provides a tabular representation of the dataset

```
df_demo.columns
```

```
Index(['code', 'state', 'category', 'total exports', 'beef', 'pork', 'poultry',
       'dairy', 'fruits fresh', 'fruits proc', 'total fruits', 'veggies fresh',
       'veggies proc', 'total veggies', 'corn', 'wheat', 'cotton'],
      dtype='object')
```

28	NH	New	state	73.06	0.6	0.2	0.8	7.46	2.6

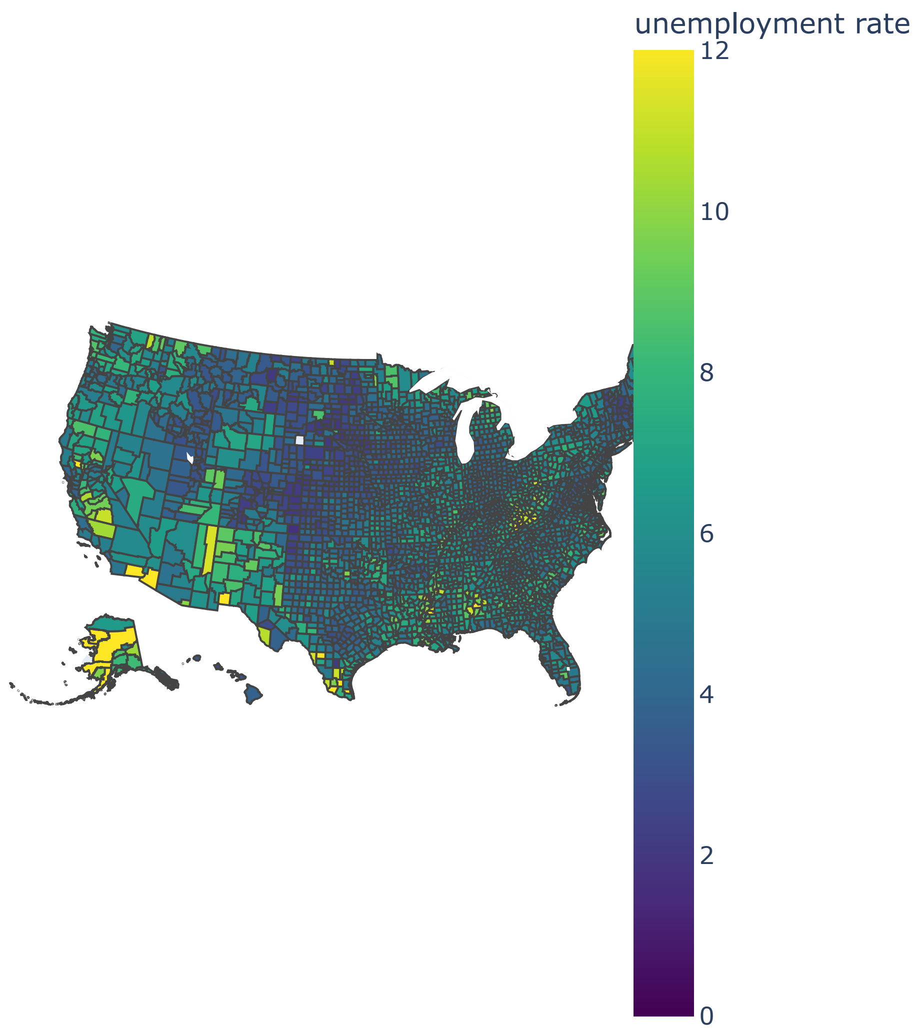
map demo #2: state of AL

```
from urllib.request import urlopen
import json
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counti
    counties = json.load(response)

import pandas as pd
df_us = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-un
                   dtype={"fips": str})

import plotly.express as px

fig = px.choropleth(df_us, geojson=counties, locations='fips', color='unemp',
                           color_continuous_scale="Viridis",
                           range_color=(0, 12),
                           scope="usa",
                           labels={'unemp':'unemployment rate'})
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```



fetches GeoJSON data of U.S. counties from a URL using urlopen and json.load, then loads unemployment data from another URL into a pandas DataFrame (df_us). it creates a choropleth map using Plotly Express, visualizing the unemployment rates across U.S. counties with specified color scales, ranges, and layout settings. The resulting interactive map provides a geographical representation of unemployment rates in the United States

```
df_us.columns
```

```
Index(['fips', 'unemp'], dtype='object')
```

```
df_us
```

	fips	unemp
0	01001	5.3
1	01003	5.4
2	01005	8.6
3	01007	6.6
4	01009	5.5
...	...	...
3214	72145	13.9
3215	72147	10.6
3216	72149	20.2
3217	72151	16.9
3218	72153	18.8

3219 rows × 2 columns

county list for ulta stores in Alabama, by FIPS code

```
al_fips =[
    {'County': 'Autauga', 'FIPS Code': '01001'},
    {'County': 'Baldwin', 'FIPS Code': '01003'},
    {'County': 'Barbour', 'FIPS Code': '01005'},
    {'County': 'Bibb', 'FIPS Code': '01007'},
    {'County': 'Blount', 'FIPS Code': '01009'},
    {'County': 'Bullock', 'FIPS Code': '01011'},
    {'County': 'Butler', 'FIPS Code': '01013'},
    {'County': 'Calhoun', 'FIPS Code': '01015'},
```

```
    {'County': 'Chambers', 'FIPS Code': '01017'},
    {'County': 'Cherokee', 'FIPS Code': '01019'},
    {'County': 'Chilton', 'FIPS Code': '01021'},
    {'County': 'Choctaw', 'FIPS Code': '01023'},
    {'County': 'Clarke', 'FIPS Code': '01025'},
    {'County': 'Clay', 'FIPS Code': '01027'},
    {'County': 'Cleburne', 'FIPS Code': '01029'},
    {'County': 'Coffee', 'FIPS Code': '01031'},
    {'County': 'Colbert', 'FIPS Code': '01033'},
    {'County': 'Conecuh', 'FIPS Code': '01035'},
    {'County':'Greene', 'FIPS Code' : '28073'},
    {'County':'Hale', 'FIPS Code' : '28065'},
    {'County':'Henry','FIPS Code' : '28067'},
    {'County':'Houston', 'FIPS Code' : '28069'},
    {'County':'Jackson', 'FIPS Code' : '28071'},
    {'County':'Jefferson', 'FIPS Code' : '28073'},
    {'County':'Lamar', 'FIPS Code' : '28073'}]
len(al_fips)
```

```
25
```

```
defines a list named al_fips containing
dictionaries where each dictionary represe
county in Alabama along with its correspon
(Federal Information Processing Standards)
The list includes information for 27 count
len(al_fips) statement at the end returns
number of elements (counties) in the list,
27
```

defines a list named al_fips containing dictionaries where each dictionary represents a county in Alabama along with its corresponding FIPS (Federal Information Processing Standards) code. The list includes information for 27 counties. The len(al_fips) statement at the end returns the number of elements (counties) in the list, which is 27

```
df_m.columns
```

```
Index(['City', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
       '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36',
       '37', '38', '39', '40', '41', '25qt', '50qt', '75qt', 'zip'],
      dtype='object')
```

```
df_m
```

	City	1	2	3	4	5	6	7	8	9	...	36	37
0	Birmingham	8285	5343	6738	6635	5658	8118	4311	8535	3436	...	3555	1341
1	Montgomery	1287	6585	8300	8874	8208	5363	3552	3387	2765	...	2805	4601
2	Mobile	8035	5569	9492	5905	5024	1107	6937	5580	8044	...	9807	2652
3	Huntsville	6280	2841	3399	5448	6173	5451	7488	9981	5236	...	7935	2605
4	Tuscaloosa	4079	1066	3923	4177	4277	4219	9436	8160	4302	...	3657	2158
5	Hoover	9741	7377	9410	9790	8864	2522	5347	9145	8402	...	9748	7224
6	Dothan	7646	2060	4911	4976	7851	4277	7423	6183	6641	...	5650	4400
7	Auburn	4326	2659	6928	4656	1828	5199	5331	6294	3076	...	4387	6890
8	Decatur	3786	2891	8124	2469	3704	3623	2409	8287	2032	...	9305	6509
9	Madison	1934	3628	9190	3275	9344	5778	1256	3523	1781	...	1746	4470
10	Florence	8017	3187	1128	4706	9962	7547	4440	4530	9569	...	5929	1123
11	Gadsden	2290	6402	8598	7547	5158	9731	8038	4435	7357	...	2549	5175
12	Vestavia Hills	9471	9142	4419	3846	2016	5069	4853	6336	9062	...	5142	9619
13	Prattville	6039	8003	6180	4610	3548	7115	6720	8512	9954	...	1591	4401
14	Phenix City	8788	8269	6838	2863	6753	6608	4048	8774	4513	...	3520	7654
15	Alabaster	1733	9767	3274	7125	7437	5748	5399	6513	3038	...	2479	9673
16	Bessemer	6559	2453	1578	5158	3058	8075	7066	8530	8346	...	4810	7641
17	Enterprise	8436	7800	7234	5063	4274	1948	7887	6647	1320	...	3461	2640
18	Opelika	9998	8953	7923	6176	4369	9503	2126	1816	9224	...	5191	9304
19	Homewood	2373	7188	9880	9236	5969	9998	8703	8440	4643	...	8787	5459
20	Northport	3536	9231	8651	6374	4842	5704	8484	6322	2012	...	6947	5401
21	Pelham	6830	3736	2734	6443	8494	6206	7290	8518	6176	...	2777	4045
22	Trussville	2794	8273	9174	2850	8351	3978	5995	4632	7693	...	1650	9470
23	Mountain Brook	8433	9368	2141	2357	6566	1482	4787	3900	6615	...	5765	3653

```
df_m.shape[0]
```

```
25
```

transform al_fips, the list of county fps codes, into a pandas dataframe

```
print(len(al_fips))
df_counties = pd.DataFrame(al_fips)
df_counties.size
```

```
25
50
```

```
print(df_counties.columns)
```

```
Index(['County', 'FIPS Code'], dtype='object')
```

```
df_m.shape[0]
```

```
25
```

```
df_counties.shape[0]
```

```
25
```

```
df_counties.columns
```

```
Index(['County', 'FIPS Code'], dtype='object')
```

merge the county fips codes with the stores sales results (df_m)

```
merged_df = pd.concat([df_m, df_counties], axis=1)
merged_df.head()
```

	City	1	2	3	4	5	6	7	8	9	...	38	39
0	Birmingham	8285	5343	6738	6635	5658	8118	4311	8535	3436	...	1756	7598
1	Montgomery	1287	6585	8300	8874	8208	5363	3552	3387	2765	...	4449	5727
2	Mobile	8035	5569	9492	5905	5024	1107	6937	5580	8044	...	9296	2815
3	Huntsville	6280	2841	3399	5448	6173	5451	7488	9981	5236	...	9982	3338
4	Tuscaloosa	4079	1066	3923	4177	4277	4219	9436	8160	4302	...	4469	2513

5 rows × 48 columns

use the merged_df as data source for the choropleth

```
merged_df.columns
```

```
Index(['City', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
       '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36',
       '37', '38', '39', '40', '41', '25qt', '50qt', '75qt', 'zip', 'County',
       'FIPS Code'],
      dtype='object')
```

use the plotly api, feed it the merged_df information to do a map, with encoded quantile values

```
import plotly.express as px

fig = px.choropleth(merged_df, geojson=counties, locations='FIPS Code', color='25qt
                           color_continuous_scale="Viridis",
                           range_color=(0, 38),
                           scope="usa",
                    hover_name="City",
                    hover_data=["City"],
                           labels={'25qt':'percentage displays under 25th qt'}  #
                          )
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

This code, using Plotly Express, creates a choropleth map visualizing data from the merged_df. It maps the '25qt' column onto U.S. counties specified by the 'FIPS Code', with a Viridis color scale, and displays additional information on hover, including city names. The layout is adjusted to have zero margins, and the resulting map is displayed interactively.

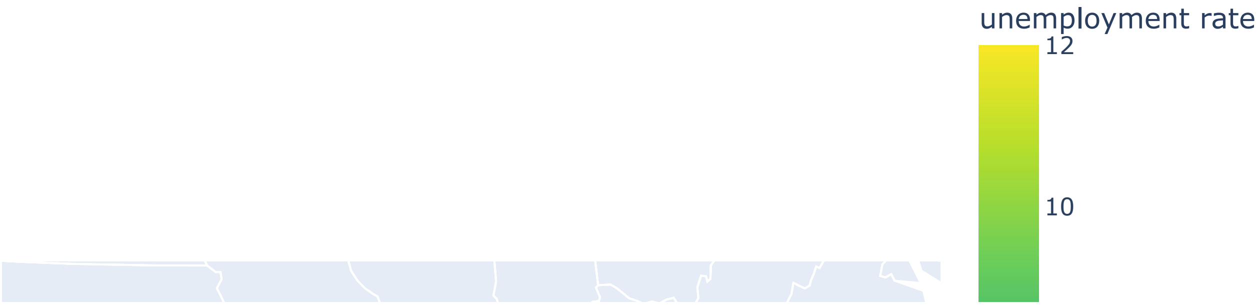
```
import plotly.express as px
import requests
import json
import pandas as pd

# Load the geojson data for Alabama's counties
r = requests.get('https://raw.githubusercontent.com/plotly/datasets/master/geojson-
counties = json.loads(r.text)

# Filter the geojson data to only include Alabama's counties
target_states = ['01']
counties['features'] = [f for f in counties['features'] if f['properties']['STATE']

# Load the sample data for Alabama's counties
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/fips-une

# Create the choropleth map
fig = px.choropleth(df, geojson=counties, locations='fips', color='unemp',
                    color_continuous_scale='Viridis', range_color=(0, 12),
                    scope='usa', labels={'unemp': 'unemployment rate'})
fig.update_layout(margin={'r': 0, 't': 0, 'l': 0, 'b': 0})
fig.show()
```



This code uses Plotly Express along with requests, json, and pandas to create a choropleth map of unemployment rates for Alabama's counties. It fetches GeoJSON data for U.S. counties, filters it to include only Alabama, loads unemployment data from a CSV file, and generates an interactive choropleth map visualizing unemployment rates across different counties in Alabama.