Double-click (or enter) to edit

written material

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

The Ulta Beauty Problem

our work entails designing and delivering a business intelligence application that serves a major retail enterprise. The system ....

first, install the plotly visualization library.

```
!pip install plotly-geo
```

```
Collecting plotly-geo
  Downloading plotly_geo-1.0.0-py3-none-any.whl (23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 45.8 MB/s eta 0:00:00
Installing collected packages: plotly-geo
Successfully installed plotly-geo-1.0.0
```

In this cell, we can see the package installer for Python called 'pip'. The 'install' command grabs the specified package from the Python Package Index and installs it on your computer. 'plotly-geo' is the name of the package being installed. It allows you to create maps and plot geographic data.

our system depends on the use of the pandas and numpy libraries.

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

'import pandas as pd' makes the pandas module availale for us in the Python script and the 'as pd' part is an alias that enables you to refer to the 'pandas' module as 'pd' in the code. 'numpy' is a numerical computing librar for Python, and 'import numpy' makes the NumPy module usable in Python script.

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

The first line assigns a URL to variable 'url', and provides raw content directory a file listed on GitHub. The second line does similar but the variable is 'url_m' and it points toward a different CSV file named "matrix.csv".

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

The white line of code utilizes the Pandas library in Python to read the content from a CSV and creates a Panda Dataframe. It first reads the data from the CSV file and assings the result to the variable 'df_m', which is stored to Pandas Dataframe.

```
df_m
```

	City	1	2	3	4	5	6	7	8	9	...	32	33	34	35	36	37	38	39	40	41
0	Birmingham	8285	5343	6738	6635	5658	8118	4311	8535	3436	...	1340	6923	3082	5617	3555	1341	1756	7598	1509	1861
1	Montgomery	1287	6585	8300	8874	8208	5363	3552	3387	2765	...	4424	8813	6655	3986	2805	4601	4449	5727	2315	8822
2	Mobile	8035	5569	9492	5905	5024	1107	6937	5580	8044	...	5430	1601	9145	1493	9807	2652	9296	2815	4886	7458
3	Huntsville	6280	2841	3399	5448	6173	5451	7488	9981	5236	...	9169	7829	6879	4166	7935	2605	9982	3338	9116	3875
4	Tuscaloosa	4079	1066	3923	4177	4277	4219	9436	8160	4302	...	1556	5533	1884	2088	3657	2158	4469	2513	8135	6963

'df_m' displays the contents of the Pandas Dataframe. In this data structure we see that each row represents a city, and each column represents a specific measurement associated with that city.

7	Auburn	4326	2659	6928	4656	1828	5199	5331	6294	3076	...	6128	3737	7785	3281	4387	6890	2833	5083	9707	2116

```
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')
```

12	Vestavia Hills	9471	9142	4419	3846	2016	5069	4853	6336	9062	...	4613	2942	7408	9484	5142	9619	9601	8099	1391	6276

'df_m.columns' is used for accessing the column labels of 'df_m' and returns an index object containing the labels. The 42 columns in the DataFrame are 42 dimensions, with each row corresonding to a city and each column corresponding to a measurement.

list all cities in 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 first line of code extracts a column from 'df_m" and returns a series containing the values of the 'City' column. Each value corresponds to the name of a city.

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
```

The first code displays the data types of each column in 'df_m'. '.dtypes' returns a Panda Series containing the data types of each column and results with a series where the index is the column names and the values are the corresponding data types.

Quantiles for each display, all stores

Double-click (or enter) to edit

```
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	...	15	16	17	18	19	20	21	22	23	24
0.25	3082.0	3633.0	2236.0	3473.0	3657.0	4628.0	4254.0	3588.0	3704.0	3451.0	...	3449.0	4246.0	4375.0	3217.0	4259.0	2468.0	3646.0	3171.0	3668.0	3027.0
0.50	5343.0	5431.0	5311.0	5771.0	5131.0	7588.0	5156.0	5331.0	6589.0	5875.0	...	6478.0	5944.0	6315.0	5341.0	6472.0	5472.0	5779.0	5995.0	5458.0	4686.0
0.75	7242.0	8074.0	7508.0	7935.0	7490.0	9145.0	6840.0	7606.0	8221.0	7783.0	...	7437.0	8331.0	8436.0	8472.0	8389.0	7877.0	8373.0	7693.0	8433.0	6576.0

3 rows × 25 columns

The first code calculates the percentiles for the numeric columns of 'df_m' alongside the specified axis, which is axis=1. 'df_3' contains the specified quantiles.

per store, the quartile values

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

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

The first code obtains the column names after transposing 'df_3'. 'Index' object '1' contains the column names of the transposed DataFrame. The column names are the quantiles (0.25, 0.5 and 0.75).

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

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

This code calculates the mean of each column transposed from 'df_3.T' The output is a Panda Series representing the mean of each quantile across all cities. Provides insights into the average values at different percentiles across all cities.

define the global quartile boundary, per q

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

```
3535.24
```

This code functions similarly to the prior, but outputs a value representing the mean of the values at the 25th percentile.

Double-click (or enter) to edit

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

```
5826.36
```

This code functions similarly to the prior, but outputs a value representing the mean of the values at the 50th percentile.

Double-click (or enter) to edit

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

```
7953.0
```

This code functions similarly to the prior, but outputs a value representing the mean of the values at the 75th percentile.

Double-click (or enter) to edit

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

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

The output is 'kk' where each index corresponds to a quantile, and each value represents the mean value for that quantile across all cities. For example, the mean value for the 25th percentile is 3535.24.

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
```

This code provides the percentage of values in each row of the numeric columns in 'df_m' that are less than or equal to the 25th percentile.

```
la = df_m['25qt'] = round(((df_m.iloc[:, 1:] <= kk[0.25]).sum(axis=1) / df_m.shape[1]) * 100,1)
ll = df_m['50qt'] = round(((df_m.iloc[:, 1:] <= kk[0.50]).sum(axis=1) / df_m.shape[1]) * 100,1)
lll = df_m['75qt'] = round(((df_m.iloc[:, 1:] <= kk[0.75]).sum(axis=1) / df_m.shape[1]) * 100,1)
print(la, ll, lll)
```

```
18    28.6
19    19.0
20    28.6
21    23.8
22    33.3
23    19.0
24    33.3
dtype: float64 0     55.8
1     55.8
2     60.5
3     51.2
4     60.5
5     34.9
6     55.8
7     51.2
8     46.5
9     48.8
10    48.8
11    41.9
12    53.5
13    44.2
14    48.8
15    41.9
16    46.5
```

```
10    63.6
11    68.2
12    70.5
13    75.0
14    75.0
15    84.1
16    70.5
17    72.7
18    72.7
19    68.2
20    75.0
21    72.7
22    75.0
23    70.5
24    86.4
dtype: float64
```

This code calculates and assigns 3 new columns to 'df_m'. These new columns represent the percentage of values in each city that are less than or equal to the quantiles. The print code will print the Series Objects to the console.

```
# df_m
```

This is a comment, which are used to include notes.

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

	City	25qt	50qt	75qt
0	Birmingham	28.6	55.8	77.3
1	Montgomery	21.4	55.8	70.5
2	Mobile	38.1	60.5	79.5
3	Huntsville	26.2	51.2	77.3
4	Tuscaloosa	21.4	60.5	79.5
5	Hoover	16.7	34.9	59.1
6	Dothan	19.0	55.8	90.9
7	Auburn	23.8	51.2	79.5
8	Decatur	21.4	46.5	70.5
9	Madison	28.6	48.8	75.0
10	Florence	26.2	48.8	63.6
11	Gadsden	19.0	41.9	68.2
12	Vestavia Hills	26.2	53.5	70.5
13	Prattville	23.8	44.2	75.0
14	Phenix City	28.6	48.8	75.0
15	Alabaster	28.6	41.9	84.1
16	Bessemer	14.3	46.5	70.5
17	Enterprise	19.0	41.9	72.7
18	Opelika	28.6	55.8	72.7
19	Homewood	19.0	41.9	68.2
20	Northport	28.6	53.5	75.0
21	Pelham	23.8	51.2	72.7
22	Trussville	33.3	48.8	75.0
23	Mountain Brook	19.0	53.5	70.5
24	Fairhope	33.3	67.4	86.4

'end_set' is a list containing the cities and the quantiles. [end_set] is a syntax used to select columns from 'df_m' based on the column names provided in 'end_set' list. The result will have the columns and quantiles and all other columns from the original DataFrame will be excluded.

create a choropleth for each store

```
#choropleth:
import pandas as pd

# Create a sample dataframe
data = {'City': ['Birmingham', 'Montgomery', 'Mobile', 'Huntsville', 'Tuscaloosa', 'Hoover', 'Dothan', 'Auburn', 'Decatur', 'Madison', 'Florence', 'Gadsden', 'Vestavia Hills', 'Prattville', 'Phenix City', 'Alabaster', 'Bessemer', 'Enterprise', 'Opelika', 'Homewood', 'Northport', 'Pelham', 'Trussville', 'Mountai
        'Zip Code': ['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']}

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  28.6  55.8  77.3  35201
1   2805  4601  4449  5727  2315  8822  21.4  55.8  70.5  36101
2   9807  2652  9296  2815  4886  7458  38.1  60.5  79.5  36601
3   7935  2605  9982  3338  9116  3875  26.2  51.2  77.3  35801
4   3657  2158  4469  2513  8135  6963  21.4  60.5  79.5  35401
5   9748  7224  4628  8107  6143  1671  16.7  34.9  59.1  35216
6   5650  4400  7842  4006  9335  3571  19.0  55.8  90.9  36301
7   4387  6890  2833  5083  9707  2116  23.8  51.2  79.5  36830
8   9305  6509  6848  5408  3707  8744  21.4  46.5  70.5  35601
9   1746  4470  7054  6573  3556  1374  28.6  48.8  75.0  35756
10  5929  1123  7306  8746  4000  6943  26.2  48.8  63.6  35630
11  2549  5175  5997  9608  7230  9731  19.0  41.9  68.2  35901
12  5142  9619  9601  8099  1391  6276  26.2  53.5  70.5  35216
13  1591  4401  3457  4245  4341  2573  23.8  44.2  75.0  36066
14  3520  7654  6845  7738  3828  1202  28.6  48.8  75.0  36867
15  2479  9673  7478  7207  7006  3523  28.6  41.9  84.1  35007
16  4810  7641  5365  3545  6812  9483  14.3  46.5  70.5  35020
17  3461  2640  4375  8634  4917  2830  19.0  41.9  72.7  36330
18  5191  9304  2720  3100  3912  1548  28.6  55.8  72.7  36801
19  8787  5459  8389  5242  2224  6025  19.0  41.9  68.2  35209
20  6947  5401  6681  9018  1668  8307  28.6  53.5  75.0  35473
21  2777  4045  7309  4745  4284  2640  23.8  51.2  72.7  35124
22  1650  9470  6356  4700  3344  8743  33.3  48.8  75.0  35173
23  5765  3653  5198  9266  4945  3935  19.0  53.5  70.5  35213
24  3457  4808  7227  5482  6355  4553  33.3  67.4  86.4  36532

[25 rows x 46 columns]
```

This code is to create a choropleth map, inwhich areas are shaded or patterend in proportion to the value being represented. Creates a DataFrane 'df' with cities and zipcodes based on the given data.

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')
```

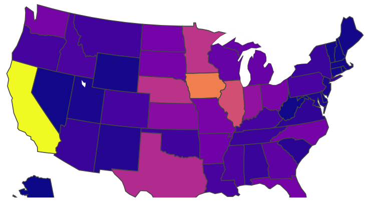
This code returns and Index object containign the column names from 'df_m'. The output represents the cities and zip codes.

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

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

# Create choropleth map
fig = px.choropleth(df_demo, locations='code', locationmode='USA-states', color='total exports', scope='usa')

# Show map
fig.show()
```



This code uses the Plotly Express library and creates a choropleth map based on agricultural export data. 'df_demo' reads a CSV file from a URL containing data related to US agricultural exports. The color intensity represents the total exports for each state.

```
df_demo
```

3	AR	Arkansas	state	3586.02	53.2	29.4	562.9	3.53	2.2	4.7	6.88	4.4	7.1	11.45	69.5	114.5	665.44
4	CA	California	state	16472.88	228.7	11.1	225.4	929.95	2791.8	5944.6	8736.40	803.2	1303.5	2106.79	34.6	249.3	1064.95
5	CO	Colorado	state	1851.33	261.4	66.0	14.0	71.94	5.7	12.2	17.99	45.1	73.2	118.27	183.2	400.5	0.00
6	CT	Connecticut	state	259.62	1.1	0.1	6.9	9.49	4.2	8.9	13.10	4.3	6.9	11.16	0.0	0.0	0.00
7	DE	Delaware	state	282.19	0.4	0.6	114.7	2.30	0.5	1.0	1.53	7.6	12.4	20.03	26.9	22.9	0.00
8	FL	Florida	state	3764.09	42.6	0.9	56.9	66.31	438.2	933.1	1371.36	171.9	279.0	450.86	3.5	1.8	78.24
9	GA	Georgia	state	2860.84	31.0	18.9	630.4	38.38	74.6	158.9	233.51	59.0	95.8	154.77	57.8	65.4	1154.07
10	HI	Hawaii	state	401.84	4.0	0.7	1.3	1.16	17.7	37.8	55.51	9.5	15.4	24.83	0.0	0.0	0.00
11	ID	Idaho	state	2078.89	119.8	0.0	2.4	294.60	6.9	14.7	21.64	121.7	197.5	319.19	24.0	568.2	0.00
12	IL	Illinois	state	8709.48	53.7	394.0	14.0	45.82	4.0	8.5	12.53	15.2	24.7	39.95	2228.5	223.8	0.00
13	IN	Indiana	state	5050.23	21.9	341.9	165.6	89.70	4.1	8.8	12.98	14.4	23.4	37.89	1123.2	114.0	0.00
14	IA	Iowa	state	11273.76	289.8	1895.6	155.6	107.00	1.0	2.2	3.24	2.7	4.4	7.10	2529.8	3.1	0.00
15	KS	Kansas	state	4589.01	659.3	179.4	6.4	65.45	1.0	2.1	3.11	3.6	5.8	9.32	457.3	1426.5	43.98
16	KY	Kentucky	state	1889.15	54.8	34.2	151.3	28.27	2.1	4.5	6.60	0.0	0.0	0.00	179.1	149.3	0.00
17	LA	Louisiana	state	1914.23	19.8	0.8	77.2	6.02	5.7	12.1	17.83	6.6	10.7	17.25	91.4	78.7	280.42
18	ME	Maine	state	278.37	1.4	0.5	10.4	16.18	16.6	35.4	52.01	24.0	38.9	62.90	0.0	0.0	0.00
19	MD	Maryland	state	692.75	5.6	3.1	127.0	24.81	4.1	8.8	12.90	7.8	12.6	20.43	54.1	55.8	0.00
20	MA	Massachusetts	state	248.65	0.6	0.5	0.6	5.81	25.8	55.0	80.83	8.1	13.1	21.13	0.0	0.0	0.00
21	MI	Michigan	state	3164.16	37.7	118.1	32.6	214.82	82.3	175.3	257.69	72.4	117.5	189.96	381.5	247.0	0.00
22	MN	Minnesota	state	7192.33	112.3	740.4	189.2	218.05	2.5	5.4	7.91	45.9	74.5	120.37	1264.3	538.1	0.00
23	MS	Mississippi	state	2170.80	12.8	30.4	370.8	5.45	5.4	11.6	17.04	10.6	17.2	27.87	110.0	102.2	494.75

'df_demo' displays the contents of its DataFrame, showing the rows and columns and the data in each cell.

```
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')
```

This code specifies to retrieve the column names of 'df_demo'. The columns retrieved include 'code', 'state', 'category', etc.

map demo #2: state of AL

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

import pandas as pd
df_us = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-16.csv",
                   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()
```

This code uses Plotly Express to create and display a choropleth map of the US based on county-level unemployment data. The data is retrieved from the URL using the 'json.load' function. A CSV file containing the unemployment data is read, and the 'fips' column is treated as a string.

```
df_us.columns
```

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

This code retrieves the column names of 'df_us'. '.columns' returns an Index object containing the column labels. The columns include 'fips' and 'unemp', representing the unique identifies for US counties and the unemployment rate data for ecah county respectively.

```
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

'df_us' is used to display the contents of its DataFrame. It shoes the rows and columns including the data in each cell.

documentation here, with more discusssion here, and specifially to do counties, here

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
```

This code defines a list names 'al_fips'. This list contains dictionaries representing counties in Alabama and their fips. 'len(al_fips)' calculates and returns the number of dictionaries in the list.

```
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')
```

This code retrieves the column names from 'df_m'. The columns are labeled as city and their numbers.

```
df_m
```

	City	1	2	3	4	5	6	7	8	9	...	36	37	38	39	40	41	25qt	50qt	75qt	zip
0	Birmingham	8285	5343	6738	6635	5658	8118	4311	8535	3436	...	3555	1341	1756	7598	1509	1861	28.6	55.8	77.3	35201
1	Montgomery	1287	6585	8300	8874	8208	5363	3552	3387	2765	...	2805	4601	4449	5727	2315	8822	21.4	55.8	70.5	36101
2	Mobile	8035	5569	9492	5905	5024	1107	6937	5580	8044	...	9807	2652	9296	2815	4886	7458	38.1	60.5	79.5	36601
3	Huntsville	6280	2841	3399	5448	6173	5451	7488	9981	5236	...	7935	2605	9982	3338	9116	3875	26.2	51.2	77.3	35801
4	Tuscaloosa	4079	1066	3923	4177	4277	4219	9436	8160	4302	...	3657	2158	4469	2513	8135	6963	21.4	60.5	79.5	35401
5	Hoover	9741	7377	9410	9790	8864	2522	5347	9145	8402	...	9748	7224	4628	8107	6143	1671	16.7	34.9	59.1	35216
6	Dothan	7646	2060	4911	4976	7851	4277	7423	6183	6641	...	5650	4400	7842	4006	9335	3571	19.0	55.8	90.9	36301
7	Auburn	4326	2659	6928	4656	1828	5199	5331	6294	3076	...	4387	6890	2833	5083	9707	2116	23.8	51.2	79.5	36830
8	Decatur	3786	2891	8124	2469	3704	3623	2409	8287	2032	...	9305	6509	6848	5408	3707	8744	21.4	46.5	70.5	35601
9	Madison	1934	3628	9190	3275	9344	5778	1256	3523	1781	...	1746	4470	7054	6573	3556	1374	28.6	48.8	75.0	35756
10	Florence	8017	3187	1128	4706	9962	7547	4440	4530	9569	...	5929	1123	7306	8746	4000	6943	26.2	48.8	63.6	35630
11	Gadsden	2290	6402	8598	7547	5158	9731	8038	4435	7357	...	2549	5175	5997	9608	7230	9731	19.0	41.9	68.2	35901
12	Vestavia Hills	9471	9142	4419	3846	2016	5069	4853	6336	9062	...	5142	9619	9601	8099	1391	6276	26.2	53.5	70.5	35216
13	Prattville	6039	8003	6180	4610	3548	7115	6720	8512	9954	...	1591	4401	3457	4245	4341	2573	23.8	44.2	75.0	36066
14	Phenix City	8788	8269	6838	2863	6753	6608	4048	8774	4513	...	3520	7654	6845	7738	3828	1202	28.6	48.8	75.0	36867
15	Alabaster	1733	9767	3274	7125	7437	5748	5399	6513	3038	...	2479	9673	7478	7207	7006	3523	28.6	41.9	84.1	35007
16	Bessemer	6559	2453	1578	5158	3058	8075	7066	8530	8346	...	4810	7641	5365	3545	6812	9483	14.3	46.5	70.5	35020
17	Enterprise	8436	7800	7234	5063	4274	1948	7887	6647	1320	...	3461	2640	4375	8634	4917	2830	19.0	41.9	72.7	36330
18	Opelika	9998	8953	7923	6176	4369	9503	2126	1816	9224	...	5191	9304	2720	3100	3912	1548	28.6	55.8	72.7	36801
19	Homewood	2373	7188	9880	9236	5969	9998	8703	8440	4643	...	8787	5459	8389	5242	2224	6025	19.0	41.9	68.2	35209
20	Northport	3536	9231	8651	6374	4842	5704	8484	6322	2012	...	6947	5401	6681	9018	1668	8307	28.6	53.5	75.0	35473
21	Pelham	6830	3736	2734	6443	8494	6206	7290	8518	6176	...	2777	4045	7309	4745	4284	2640	23.8	51.2	72.7	35124
22	Trussville	2794	8273	9174	2850	8351	3978	5995	4632	7693	...	1650	9470	6356	4700	3344	8743	33.3	48.8	75.0	35173
23	Mountain Brook	8433	9368	2141	2357	6566	1482	4787	3900	6615	...	5765	3653	5198	9266	4945	3935	19.0	53.5	70.5	35213
24	Fairhope	8114	1464	2811	3090	4686	7995	7676	1304	7332	...	3457	4808	7227	5482	6355	4553	33.3	67.4	86.4	36532

25 rows × 46 columns

As used before, this code displays the contents of the DataFrame known as 'df_m'. It is the most up to date version of the table including the cities, the quartiles, and the zip to each city.

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

```
25
```

This code retrieves the rows in 'df_m'. '.shape' returns a tuple that shows the dimensions of the DataFrame. In this case, the value is 25 which indicates the 25 rows in the DataFrame.

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
```

The first line prints the list 'al_fips' using the 'len()' function. The output is the number of dictionaries in the list 'al_fips'. The second code creates a DataFrame named 'df_counties' from the list of dictionaries 'al_fips'. The last code returns the number of elements, which is the total number of rows and columns.

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

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

The code displays the column names of 'df_counties'. The columns are labled 'county' and 'fips code'.

df_m: all display data, per store

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

```
25
```

Retrieves the number of rows in 'df_m'. The value is 25, indicating there are 25 rows.

fips codes per county

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

```
25
```

This code functions similarly to the last but instead retrieves the number of rows in the 'df_counties' dataframe.

Double-click (or enter) to edit

```
df_counties.columns
```

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

The first code retrieves the column names from 'df_counties'. The output represents these column names as an index object. The columns are labeled 'county' and 'fips code'.

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	40	41	25qt	50qt	75qt	zip	County	FIPS Code
0	Birmingham	8285	5343	6738	6635	5658	8118	4311	8535	3436	...	1756	7598	1509	1861	28.6	55.8	77.3	35201	Autauga	01001
1	Montgomery	1287	6585	8300	8874	8208	5363	3552	3387	2765	...	4449	5727	2315	8822	21.4	55.8	70.5	36101	Baldwin	01003
2	Mobile	8035	5569	9492	5905	5024	1107	6937	5580	8044	...	9296	2815	4886	7458	38.1	60.5	79.5	36601	Barbour	01005
3	Huntsville	6280	2841	3399	5448	6173	5451	7488	9981	5236	...	9982	3338	9116	3875	26.2	51.2	77.3	35801	Bibb	01007
4	Tuscaloosa	4079	1066	3923	4177	4277	4219	9436	8160	4302	...	4469	2513	8135	6963	21.4	60.5	79.5	35401	Blount	01009

5 rows × 48 columns

This code merges 'df_m' and 'df_counties' along the columns (axis=1). The new data frame is 'merged_df'. The result contains both city and county information into one new dataframe.

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')
```

Double-click (or enter) to edit

This code retrieves the columnn names from 'merged_df', which is a new dataframe that was made merging 'df_m' and 'df_counties'.

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()
```

'px.choropleth' is a function that creates a choropleth map using plotly express, using 'merged_df' as the data. It then assigns specific colors to the quartiles and sets the city names and hover data. As a result, this map visualizes the data and uses color intensity to represent values in the '25qt' column.

```
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-fips.json')
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'] in target_states]

# Load the sample data for Alabama's counties
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-16.csv', dtype={'fips': str})

# 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()
```

'requests.get' retrives GeoJSON data for US counties from a URL, to which ('target_states = ['01']') filters the data to only include Alabama's counties. A choropleth map is made, which like the last, uses color to display the unemployment rates. Darker colors indicate higher rates.