Lecture 6

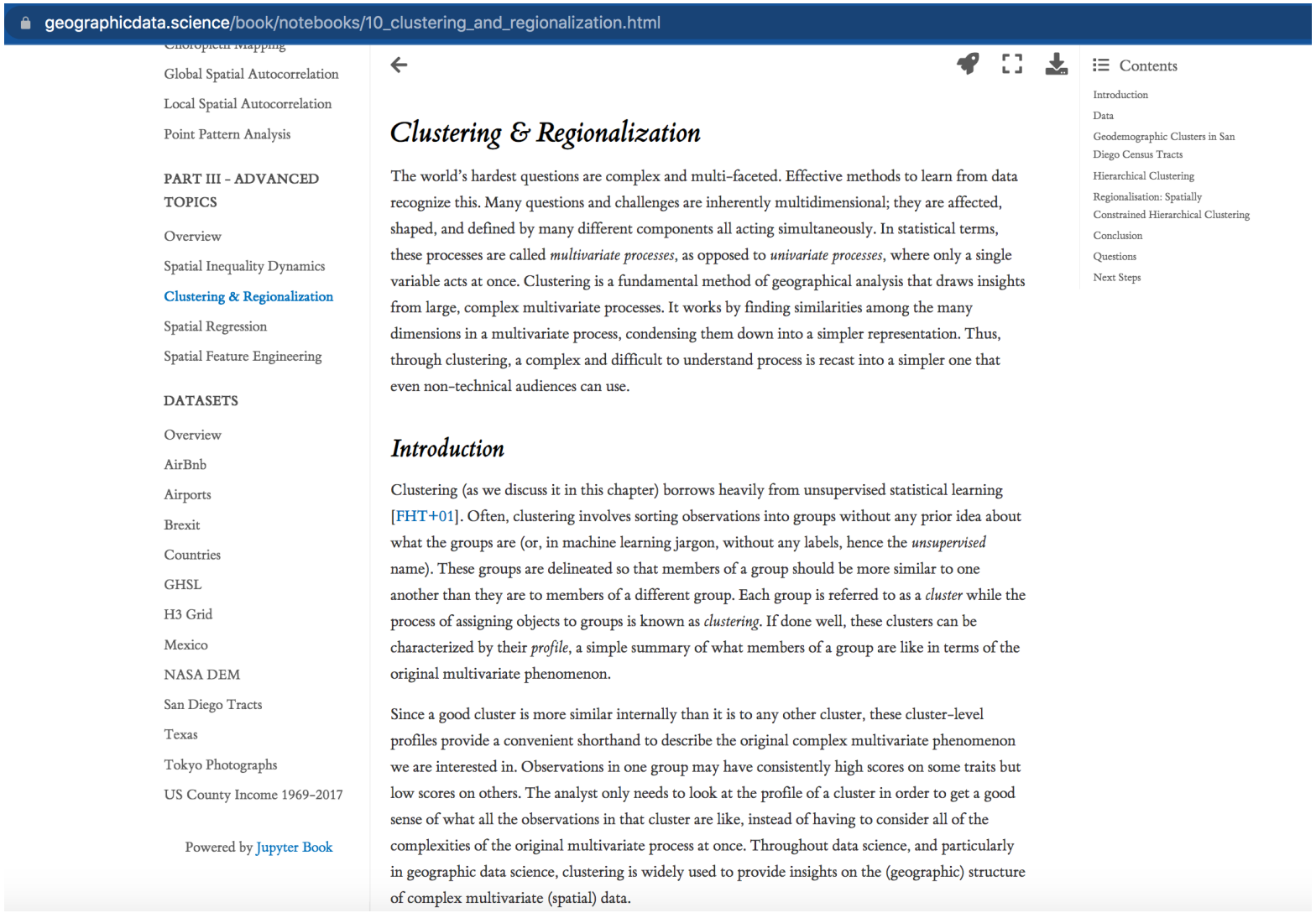
# K-means Clustering and Census Data

- cenpy_api.ipynb (Lecture 5)
- sandiego_tracts_cleaning.ipynb (Lecture 5)
- Clustering_SanDiegoHouseValues.ipynb (Lecture 6)
- Clustering_FultonGA_FoodStamps.ipynb (Lecture 6)

# Today

- Summary from last Class
- Participation Slides
- Clustering_SanDiegoHouseValues.ipynb
- Clustering_FultonGA_FoodStamps.ipynb

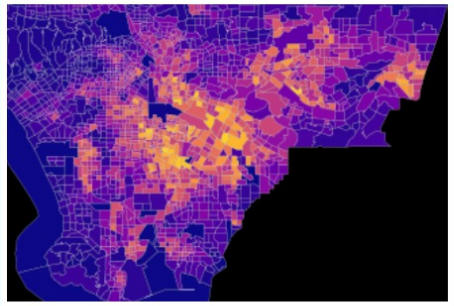
Book chapter for Lectures 5 and 6:



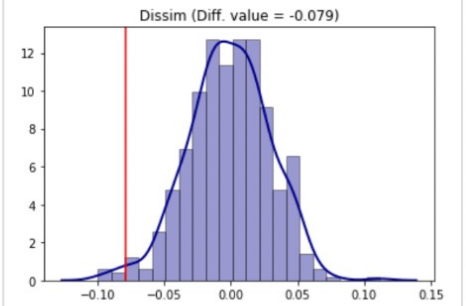

https://geographicdata.science/book/notebooks/10_clustering_and_regionalization.html

# Cenpy

Cenpy (pronounced sen-pie) is a package that automatically discovers US Census Bureau API endpoints and exposes them to Python in a consistent fashion. It also provides easy-to-use access to certain well-used data products, like the American Community Survey (ACS) and 2010 Decennial Census. To get started, check out one of the case studies shown below.



Getting Data using cenpy




Segregation in Time and Space with cenpy and pysal

A Road to Frictionless Urban Data Science: osmnx and cenpy


Cenpy is easiest to install using `conda`, a commonly-used package manager for scientific python. First, install Anaconda.

Then, `cenpy` is available on the `conda-forge` channel. You can install this using the *Anaconda Prompt*, or from within the Anaconda Navigator program. If you want to install the package from within the Anaconda Prompt, you can use:

```
conda install -c conda-forge cenpy
```

Alternatively, you can install cenpy *via* `pip`, the python package manager, if you have installed `geopandas` and `rtree`:

```
pip install cenpy
```

https://cenpy-devs.github.io/cenpy/index.html

Cenpy Products

# http://cenpy-devs.github.io/cenpy/api.html

A product that integrates the data & geographic APIs for the 5-year 2013-2017 ACSs.

| | |
|---|---|
| `cenpy.products.ACS` | The American Community Survey (5-year vintages) from the Census Bueau |

A product that integrates the data & geographic APIs for the 2010 Census.

| | |
|---|---|
| `cenpy.products.Decennial2010` | The 2010 Decennial Census from the Census Bueau |

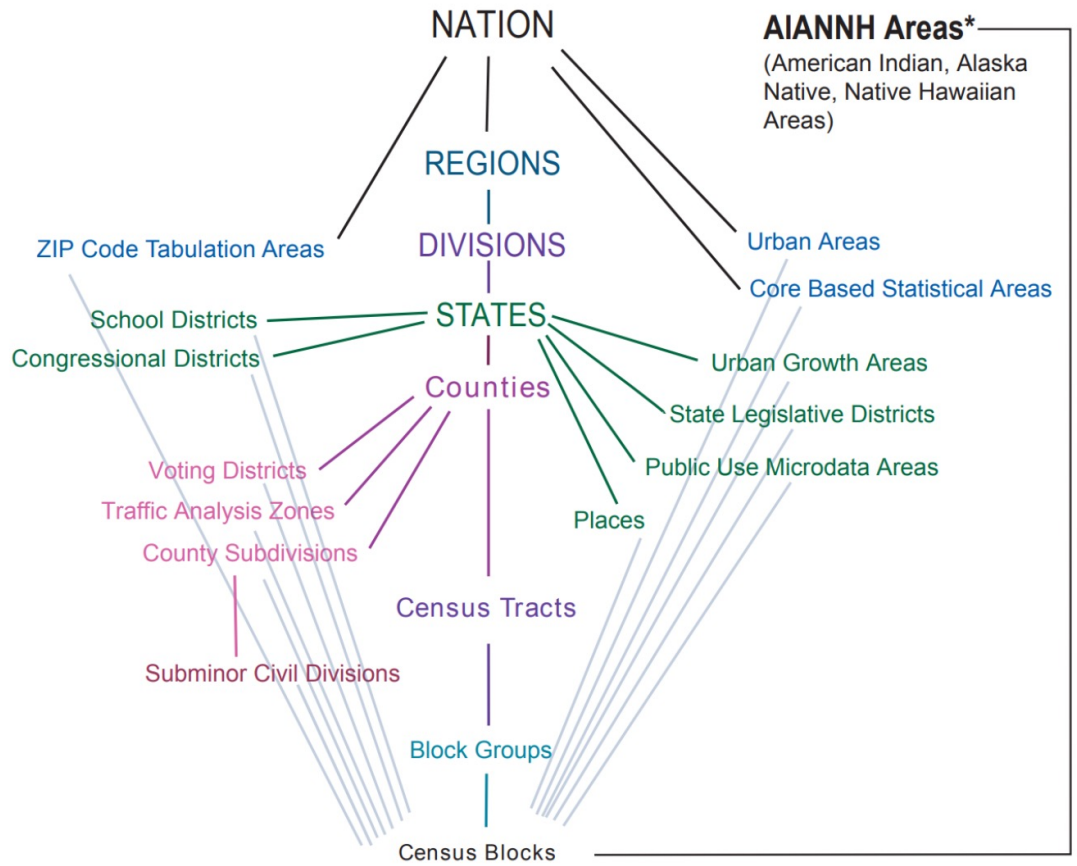
# Cenpy 1.0.0

Cenpy started as an interface to explore and query the US Census API and return Pandas Dataframes. This was mainly intended as a wrapper over the basic functionality provided by the census bureau. I was initially inspired by `acs.R` in its functionality and structure. In addition to `cenpy`, a few other census packages exist out there in the Python ecosystem, such as:

- datamade/census - "a python wrapper for the US Census API"
- jtleider/censusdata - "download data from Census API"

Others use `requests` raw on the Census API to extract the data they want.

Most packages (including `cenpy` itself) involved a very stilted/specific API query due to the way the census API worked. Basically, it's difficult to construct an efficiently query against the census API without knowing the so-called "geographic hierarchy" in which your query fell:



1- Decide the Data set and Variable:

2- Decide Geography

3-Make an API Call
Select Table Name (for ACS start with B)
https://data.census.gov/cedsci/
then google it and find variable name
https://api.census.gov/data/2017/acs/acs5/variables.html

# cenpy-api.ipynb

```
dectest = products.Decennial2010()
```

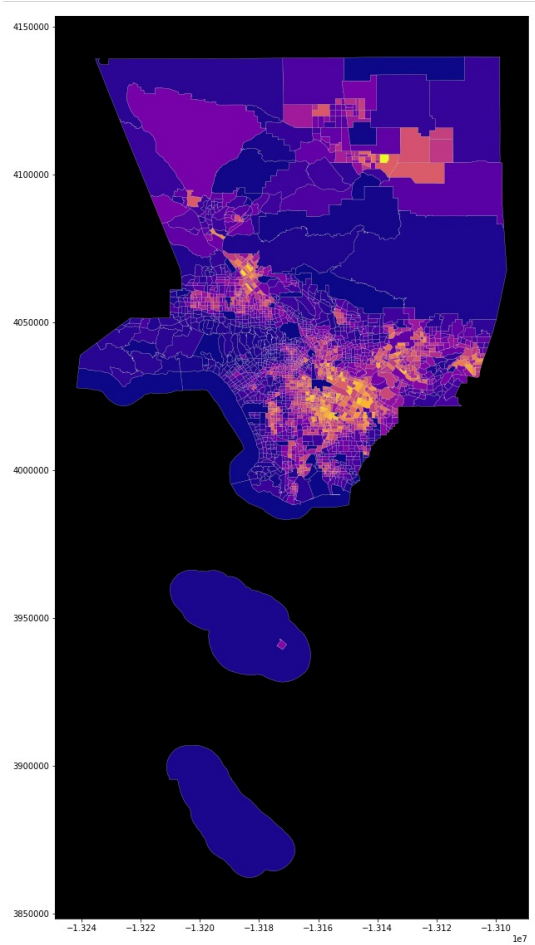
Now, since we don't need to worry about entering geo-in-geo structures for our queries, we can request race data for all the tracts in Los Angeles County using the following method:

```
la = dectest.from_county('Los Angeles, CA',
                         level='tract',
                         variables=['^P004'])
```

hispanic

And, making a pretty plot of the Hispanic population in LA:

```
f,ax = plt.subplots(1,1,figsize=(20,20))
la.plot('P004003', edgecolor='white', cmap='plasma', ax=ax, linewidth=.2)
ax.set_facecolor('k')
```



Let’s see the participation slides:

https://docs.google.com/presentation/d/1IxmXNjFCS1TXb7FQMB2TByHb8zHVyMBHzSzfWsuTwGU/edit?usp=sharing

# sandiego_tracts_cleaning.ipynb

```
import contextily
import geopandas
import cenpy

acs = cenpy.products.ACS(2017)
```

## Download Data

- Set variables to download

```
vars_to_download = {
    "B25077_001E": "median_house_value",  # Median house value
    "B02001_002E": "total_pop_white",     # Total white population
    "B01003_001E": "total_pop",           # Total population
    "B25003_003E": "total_rented",        # Total rented occupied
    "B25001_001E": "total_housing_units", # Total housing units
    "B09019_006E": "hh_female",           # Female households
    "B09019_001E": "hh_total",            # Total households
    "B15003_002E": "total_bachelor",      # Total w/ Bachelor degree
    "B25018_001E": "median_no_rooms",     # Median number of rooms
    "B19083_001E": "income_gini",         # Gini index of income inequality
    "B01002_001E": "median_age",          # Median age
    "B08303_001E": "tt_work",              # Aggregate travel time to work
    "B19013_001E": "median_hh_income"     # Median household income
}
vars_to_download_l = list(vars_to_download.keys())
```

- Download geometries and attributes

```
%%time
db = acs.from_msa("San Diego, CA",
                  level="tract",
                  variables=vars_to_download_l
                 )
```

## Process data

While the ACS comes with a large number of attributes, we are not limited to the original variables at hand; we can construct additional variables. This is particularly useful when we want to compare areas that are not very similar in some structural characteristic, such as area or population. For example, a quick look into the variable names shows most variables are counts. For tracts of different sizes, these variables will mainly reflect their overall population, rather than provide direct information about the variables itself. To get around this, we will cast many of these count variables to rates, and use them in addition to a subset of the original variables.

- Replace missing values with columns mean

In [33]:
```
filler = lambda col: col.fillna(col.mean())
db.loc[:, vars_to_download] = db.loc[:, vars_to_download]\
                                .apply(filler)
```

- Replace variable codes with short names

In [34]:
```
db = db.rename(columns=vars_to_download)
```

- Calculate area in Sq.Km (we use the Conus Albers CRS)

In [35]:
```
db["area_sqm"] = db.to_crs(epsg=5070).area / 1e6
```

- Percentage of renter occupied units

In [36]:
```
db["pct_rented"] = db["total_rented"] / \
                   (db["total_housing_units"] + \
                    (db["total_housing_units"]==0) * 1
                   )
```

- Percentage of female households

In [37]:
```
db["pct_hh_female"] = db["hh_female"] / \
                      (db["hh_total"] + \
                       (db["hh_total"]==0) * 1
                      )
```

- Dataset

[47]:
```
! rm -f sandiego_tracts.gpkg
db.to_file("sandiego_tracts.gpkg", driver="GPKG")
```

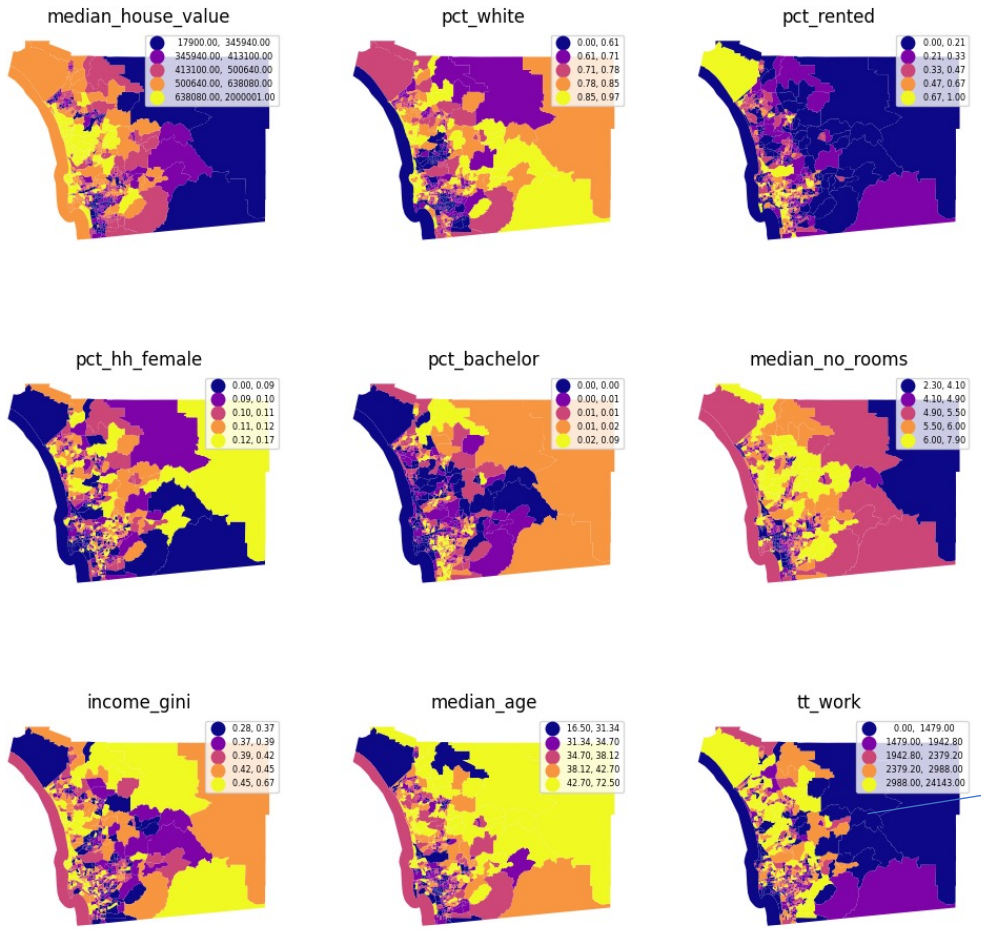
- Metadata

[44]:
```
! rm -f sandiego_tracts_varnames.json
var_names.to_json("sandiego_tracts_varnames.json")
```

See: https://www.gis-blog.com/geopackage-vs-shapefile/

# Clustering_SanDiegoHouseValues.ipynb

```python
f, axs = plt.subplots(nrows=3, ncols=3, figsize=(12, 12))
# Make the axes accessible with single indexing
axs = axs.flatten()
# Start a loop over all the variables of interest
for i, col in enumerate(cluster_variables):
    # select the axis where the map will go
    ax = axs[i]
    # Plot the map
    db.plot(column=col, ax=ax, scheme='Quantiles',
            linewidth=0, cmap='plasma',legend=True,legend_kwds={'fontsize':'xx-small'})
    # Remove axis clutter
    ax.set_axis_off()
    # Set the axis title to the name of variable being plotted
    ax.set_title(col)
    plt.savefig('msps_variables.png')
# Display the figure
plt.show()
```




Quantiles are statistical measures that divide a dataset into equal-sized groups or intervals.

They are used to describe the distribution of a dataset and provide information about the spread and central tendency of the data.

For example, the median is a commonly used quantile that divides a dataset into two equal halves

In the example from the left the first quantile (Q1) is the value below which 20% of the data falls, while the third quartile (Q3) is the value below which 60% of the data falls.

This is the number of households with Travel Time to work Data no the travel time. It is important to make sense of The variables to be plotted.

In probability theory, a **probability density function** (**PDF**), or **density** of a continuous random variable, is a function whose value at any given sample can be interpreted as providing a *relative likelihood* that the value of the random variable would equal that sample

$$\Pr[a \leq X \leq b] = \int_a^b f_X(x)\,dx.$$

Probability Density curves have the following properties:

- The area under the curve always adds up to 1.
- The curve is nonnegative

This plot shows the Probability Density Function and the scatter plots of variables


```
_=seaborn.pairplot(db[cluster_variables], kind='reg', diag_kind='kde')
plt.show()
```

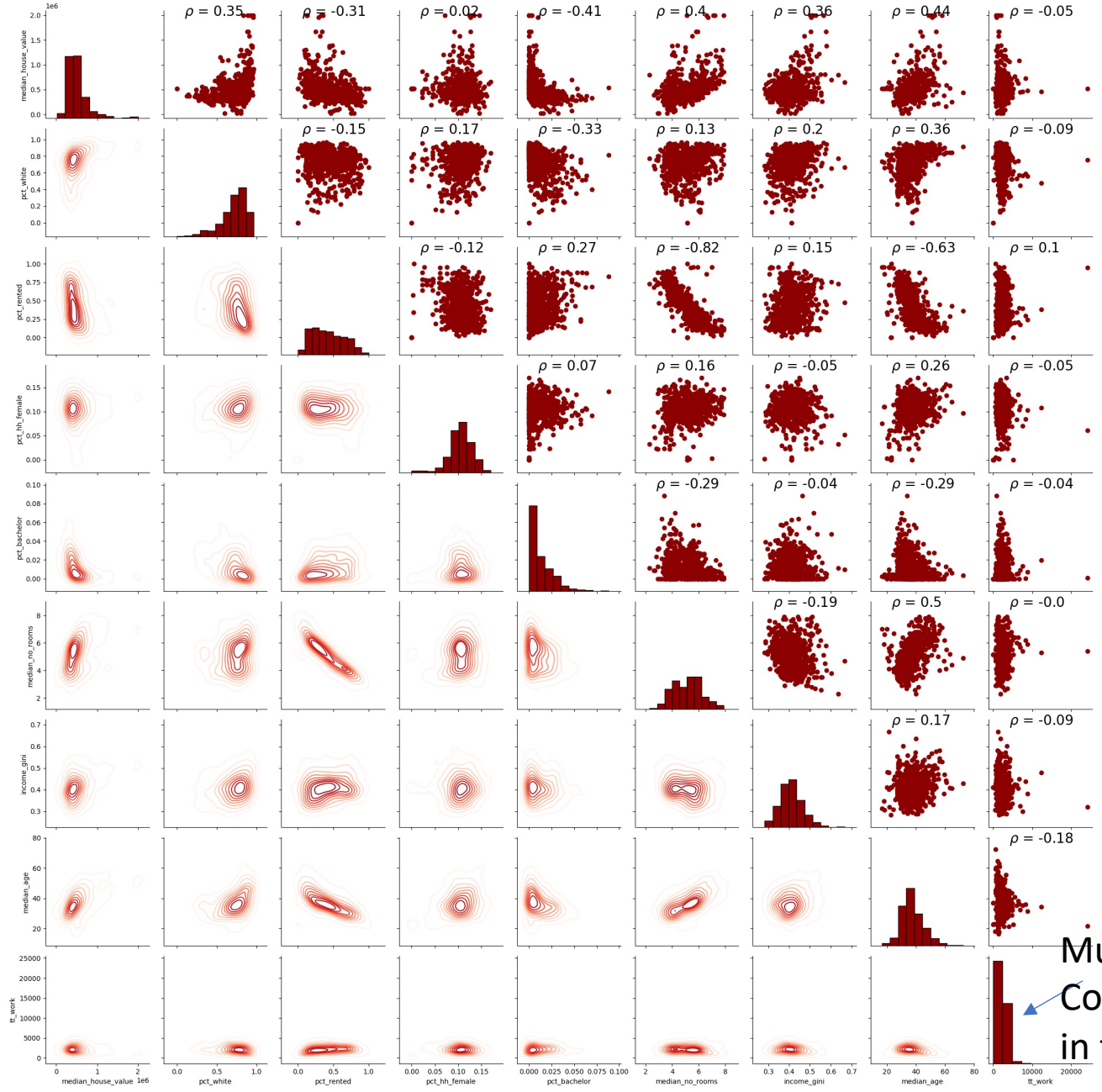
HH value negative correlation with % rented
HH value positive correlation with Median number of rooms

% rented negative correlation with # rooms and median age

```python
# Function to calculate correlation coefficient between two arrays
def corr(x, y, **kwargs):
    # Calculate the value
    coef = np.corrcoef(x, y)[0][1]
    # Make the label
    label = r'$\rho$ = ' + str(round(coef, 2))

    # Add the label to the plot
    ax = plt.gca()
    ax.annotate(label, xy = (0.2, 0.95), size = 20, xycoords = ax.transAxes)


grid = seaborn.PairGrid(db[cluster_variables])
# Map the plots to the locations
grid = grid.map_upper(plt.scatter, color = 'darkred')
grid = grid.map_upper(corr)
grid = grid.map_lower(seaborn.kdeplot, cmap = 'Reds')
grid = grid.map_diag(plt.hist, bins = 10, edgecolor =  'k', color = 'darkred');
plt.savefig('PairGrid.png')
plt.show()
```

$$\rho = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Source:
https://towardsdatascience.com/visualizing-data-with-pair-plots-in-python-f228cf529166




Must be Corrected in textbook

- Clustering is a method of data analysis that draws insights from large, complex multivariate processes.

- It works by finding similarities among the many dimensions in a multivariate process, condensing them down into a simpler representation. Thus, through clustering, we seek to reduce the complexity of the data

- Often, clustering involves sorting observations into groups. For these groups to be meaningful, members of a group should be more similar to one another than they are to members of a different group.

- Each group is referred to as a *cluster* while the process of assigning objects to groups is known as *clustering*. Since a good cluster is more similar internally than it is to any other cluster, these cluster-level profiles provide a convenient shorthand to describe the original complex multivariate process.

# K-means Clustering

Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_n$), where each observation is a *d*-dimensional real vector, *k*-means clustering aims to partition the *n* observations into *k* (≤ *n*) sets $\mathbf{S} = \{S_1, S_2, ..., S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance). Formally, the objective is to find:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

Vector X contains the variables of element i
Each Set $S_i$ has a list of elements close to their mean $\boldsymbol{\mu}_i$

where $\boldsymbol{\mu}_i$ is the mean of points in $S_i$.

**This is equivalent to minimizing the pairwise squared deviations of points in the same cluster**

**This is equivalent to maximizing the sum of squared deviations between points in *different* clusters (between-cluster sum of squares, BCSS)**

## K-means

K-means is probably the most widely used approach to cluster a dataset. The algorithm groups observations into a prespecified number of clusters so that that each observation is closer to the mean of its own cluster than it is to the mean of any other cluster.

The k-means problem is solved by iterating between an assignment step and an update step. First, all observations are randomly assigned one of the $k$ labels.

Next, the multivariate mean over all covariates is calculated for each of the clusters. Then, each observation is reassigned to the cluster with the closest mean.

If the observation is already assigned to the cluster whose mean it is closest to, the observation remains in that cluster. This assignment-update process continues until no further reassignments are necessary.

The nature of this algorithm requires us to select the number of clusters we want to create. The right number of clusters is unknown in practice. For illustration, we will use $k = 5$ in the `KMeans` implementation from `scikit-learn`. To proceed, we first create a `KMeans` clusterer:

```
# Initialise KMeans instance
kmeans = KMeans(n_clusters=5)
```

Next, we call the `fit` method to actually apply the k-means algorithm to our data:

```
# Set the seed for reproducibility
numpy.random.seed(1234)
# Run K-Means algorithm
k5cls = kmeans.fit(db[cluster_variables])
```

Now that the clusters have been assigned, we can examine the label vector, which records the cluster to which each observation is assigned:

```
k5cls.labels_
```

```
array([0, 4, 2, 3, 3, 0, 1, 3, 0, 2, 4, 4, 4, 2, 4, 4, 4, 4, 0, 4, 1, 4,
       4, 4, 4, 0, 0, 0, 4, 0, 0, 0, 1, 4, 4, 0, 4, 2, 0, 0, 4, 4, 0, 0,
       2, 0, 4, 0, 4, 0, 0, 4, 4, 0, 0, 0, 4, 2, 0, 4, 4, 0, 2, 0, 4, 4,
       0, 0, 4, 2, 0, 0, 0, 4, 1, 4, 0, 2, 3, 4, 2, 0, 4, 0, 1, 1, 2, 2,
       4, 4, 4, 4, 0, 4, 2, 0, 0, 0, 0, 2, 4, 4, 0, 4, 4, 0, 0, 0, 0, 2,
       0, 4, 2, 0, 0, 0, 2, 2, 4, 4, 0, 4, 0, 4, 4, 4, 4, 4, 4, 4, 1, 0,
       4, 4, 0, 0, 4, 0, 0, 0, 4, 4, 4, 4, 0, 2, 0, 2, 0, 0, 0, 4, 2, 0,
       4, 2, 4, 4, 0, 4, 1, 2, 2, 4, 4, 2, 0, 4, 4, 4, 4, 4, 1, 3, 4, 4,
       4, 3, 4, 0, 4, 2, 0, 0, 0, 4, 4, 4, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0,
       0, 4, 0, 4, 4, 0, 0, 4, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4,
       4, 4, 4, 4, 0, 4, 4, 2, 2, 0, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 0, 2,
       0, 0, 0, 4, 4, 0, 2, 2, 1, 2, 2, 4, 0, 2, 2, 0, 4, 3, 1, 4, 1, 4,
       2, 4, 0, 4, 4, 0, 2, 4, 4, 4, 4, 4, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
       4, 2, 0, 0, 0, 4, 0, 0, 0, 2, 0, 0, 0, 4, 0, 0, 0, 0, 4, 4, 4, 0,
       4, 4, 4, 0, 4, 0, 0, 2, 4, 0, 2, 1, 4, 4, 2, 4, 4, 0, 1, 4, 2, 0,
       4, 0, 2, 4, 0, 4, 1, 4, 4, 0, 4, 0, 4, 4, 2, 4, 4, 4, 0, 4, 2, 2,
       0, 1, 4, 0, 0, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 4, 4, 0, 2,
       2, 4, 0, 0, 0, 0, 0, 0, 2, 0, 0, 4, 2, 4, 0, 4, 4, 0, 4, 2, 0, 0,
       0, 4, 4, 4, 0, 4, 0, 2, 2, 1, 2, 2, 0, 1, 2, 0, 2, 4, 2, 2, 4, 2,
       2, 4, 0, 4, 4, 4, 0, 2, 2, 0, 0, 1, 0, 0, 4, 0, 2, 0, 4, 4, 2, 0,
       4, 0, 0, 4, 0, 2, 0, 4, 0, 4, 4, 0, 0, 4, 0, 0, 4, 0, 0, 4, 4, 4,
       4, 2, 4, 0, 4, 0, 0, 0, 4, 4, 0, 2, 4, 0, 2, 4, 0, 2, 2, 0, 2, 2,
       0, 0, 3, 2, 1, 4, 2, 4, 4, 4, 2, 4, 1, 1, 2, 2, 2, 2, 4, 4, 4, 2,
       4, 4, 4, 0, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 0, 4, 0, 4, 0, 0, 4, 4,
       0, 0, 4, 0, 0, 4, 4, 0, 4, 0, 0, 0, 4, 0, 2, 4, 0, 4, 0, 4, 4, 4,
       0, 0, 4, 4, 0, 0, 0, 4, 4, 4, 4, 4, 2, 4, 4, 4, 2, 4, 2, 2, 4, 1, 1,
       4, 2, 3, 2, 2, 2, 4, 2, 2, 4, 2, 4, 0, 4, 4, 2, 4, 4, 0, 4, 4, 4,
       0, 1, 2, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 4, 2, 0, 4, 4, 0, 4, 0, 4, 4], dtype=int32)
```

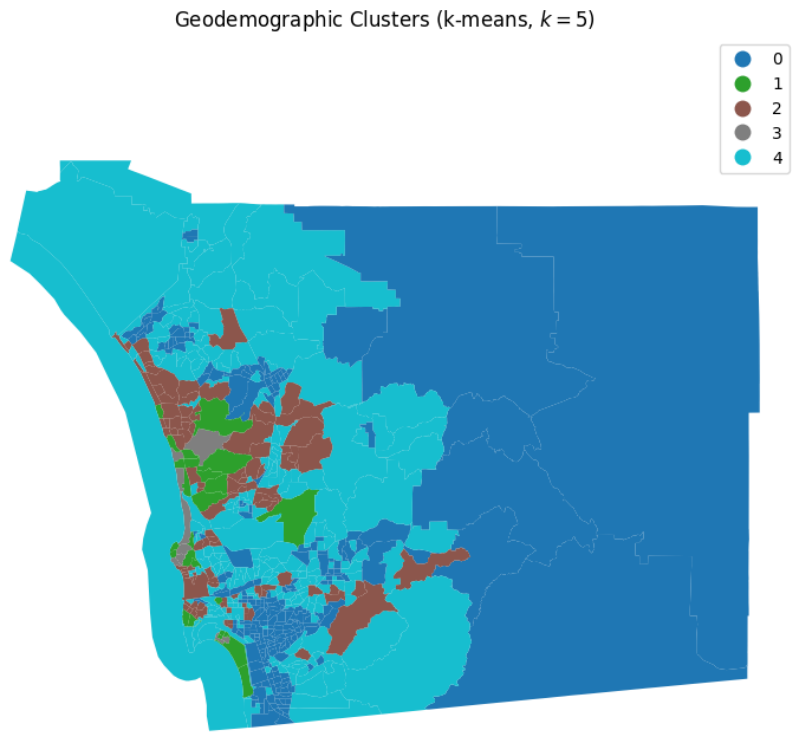
The integer labels should be viewed as denoting membership only — the numerical differences between the values for the labels are meaningless. The profiles of the various clusters must be further explored by looking at the values of each dimension.

But, before we do that, let's make a map.

### Spatial Distribution of Clusters

Having obtained the cluster labels, we can display the spatial distribution of the clusters by using the labels as the categories in a choropleth map. This allows us to quickly grasp any sort of spatial pattern the clusters might have. Since clusters represent areas with similar characteristics, mapping their labels allows to see to what extent similar areas tend to have similar locations. Thus, this gives us one map that incorporates the information of from all nine covariates.

```
# Assign labels into a column
db['k5cls'] = k5cls.labels_
# Setup figure and ax
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot unique values choropleth including a legend and with no boundary lines
db.plot(column='k5cls', categorical=True, legend=True, linewidth=0, ax=ax)
# Remove axis
ax.set_axis_off()
# Keep axes proportionate
plt.axis('equal')
# Add title
plt.title(r'Geodemographic Clusters (k-means, $k=5$)')
# Display the map
plt.show()
```



### Statistical Analysis of the Cluster Map

To complement the geovisualization of the clusters, we can explore the statistical properties of the cluster map. This process allows us to delve into what observations are part of each cluster and what their characteristics are.

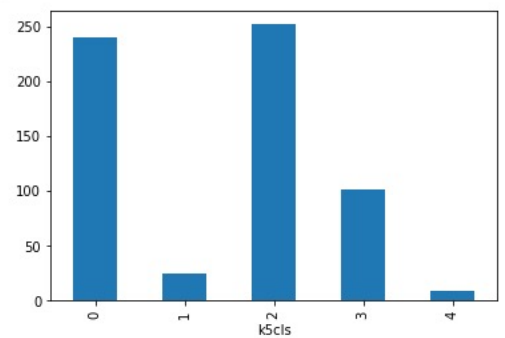
This gives us the profile of each cluster so we can interpret the meaning of the labels we've obtained. We can start, for example, by considering cardinality, or the count of observations in each cluster:

```
# Group data table by cluster label and count observations
k5sizes = db.groupby('k5cls').size()
k5sizes
```

```
k5cls
0    240
1     25
2    252
3    102
4      9
dtype: int64
```

And we can get a visual representation of cardinality as well:

```
_ = k5sizes.plot.bar()
```



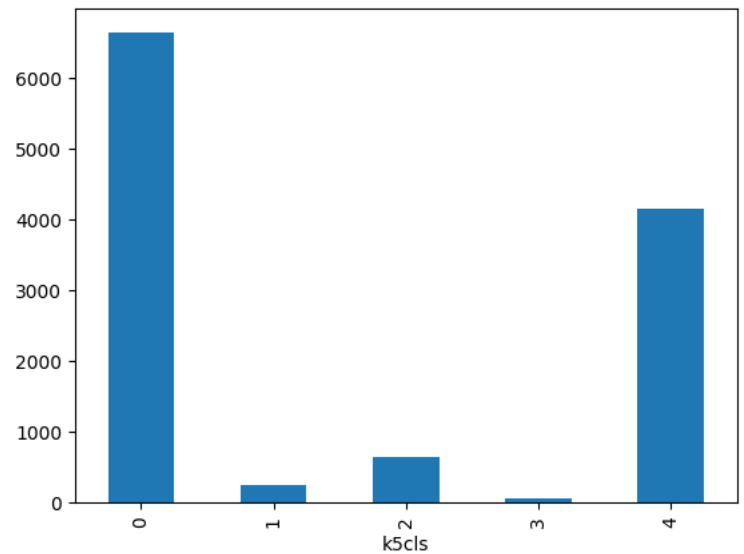

There are substantial differences in the sizes of the five clusters, with two very large clusters (0, 2), one medium sized cluster (3), and two small clusters (1, 4). Cluster 2 is the largest when measured by the number of assigned tracts. This confirms our intuition from the map above, where we got the visual impression that tracts in cluster 2 seemed to have the largest area. Let's see if this is the case. To do so we can use the `dissolve` operation in `geopandas`, which combines all tracts belonging to each cluster into a single polygon object. After we have dissolved all the members of the clusters, we report the total land area of the cluster:

```
# Dissolve areas by Cluster, aggregate by summing, and keep column for area
areas = db.dissolve(by='k5cls', aggfunc='sum')['area_sqkm']
areas
```

```
k5cls
0    6636.452408
1     245.344545
2     639.170183
3      63.002873
4    4137.802631
Name: area_sqkm, dtype: float64
```

And, to show this visually:

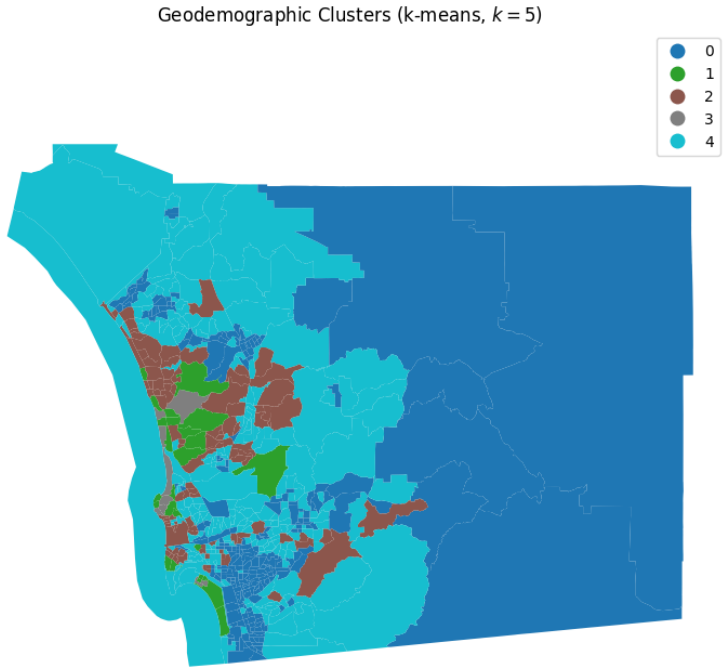
```
areas.plot.bar()
plt.show()
```




Our visual impression is confirmed: cluster 0 contains tracts that together comprise 6,636 square kilometers (approximately 2562 square miles), which accounts for over half of the total land area in the county:

```
areas[0]/areas.sum()
```

```
0.5661645735975523
```



]:
```
# Group table by cluster label, keep the variables used
# for clustering, and obtain their mean
k5means = db.groupby('k5cls')[cluster_variables].mean()
k5means.T.round(2)
```

]:

| k5cls | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| median_house_value | 326728.97 | 1168004.00 | 736881.37 | 1876867.22 | 500787.57 |
| pct_white | 0.66 | 0.84 | 0.81 | 0.91 | 0.72 |
| pct_rented | 0.51 | 0.29 | 0.33 | 0.25 | 0.41 |
| pct_hh_female | 0.11 | 0.11 | 0.10 | 0.12 | 0.10 |
| pct_bachelor | 0.02 | 0.00 | 0.00 | 0.00 | 0.01 |
| median_no_rooms | 4.63 | 6.02 | 5.73 | 6.42 | 5.29 |
| income_gini | 0.40 | 0.47 | 0.43 | 0.52 | 0.40 |
| median_age | 34.25 | 44.44 | 41.60 | 50.54 | 37.21 |
| tt_work | 2187.75 | 2182.00 | 2336.28 | 1237.78 | 2539.43 |

We see that cluster 3, for example, is composed of tracts that have the highest average `median_house_value`, and also the highest level of inequality (`income_gini`); and cluster 3 contains an older population (`median_age`) who tend to live in housing units with more rooms (`median_no_rooms`). Average values, however, can hide a great deal of detail and, in some cases, give wrong impressions about the type of data distribution they represent. To obtain more detailed profiles, we can use the `describe` command in `pandas`, after grouping our observations by their clusters:

We can have a descriptive summary but it is harder to read

```
: 1 # Group table by cluster label, keep the variables used
  2 # for clustering, and obtain their descriptive summary
  3 k5desc = db.groupby('k5cls')[cluster_variables].describe()
  4 # Loop over each cluster and print a table with descriptives
  5 for cluster in k5desc.T:
  6     print('\n\t---------\n\tCluster %i'%cluster)
  7     print(k5desc.T[cluster].unstack())
```

```
        Cluster 0
                        count           mean           std         min  \
median_house_value      240.0  500787.574698  57295.975694  414100.000
pct_white               240.0       0.722390      0.149383       0.000
pct_rented              240.0       0.405979      0.223881       0.000
pct_hh_female           240.0       0.101796      0.030586       0.000
pct_bachelor            240.0       0.010111      0.010658       0.000
median_no_rooms         240.0       5.294402      0.957028       2.800
income_gini             240.0       0.400898      0.048220       0.283
median_age              240.0      37.210522      7.389658      16.500
tt_work                 240.0    2539.429167   1911.989514       0.000

                              25%            50%            75%            max
median_house_value  453750.000000  496850.000000  539675.000000  617000.000000
pct_white                0.667674       0.754861       0.823160       0.946018
pct_rented               0.225816       0.373094       0.541326       1.000000
pct_hh_female            0.091103       0.107624       0.119297       0.166396
pct_bachelor             0.002472       0.006677       0.015632       0.088600
median_no_rooms          4.700000       5.400000       6.000000       7.500000
income_gini              0.369650       0.403850       0.428775       0.553800
```

However, this approach quickly gets out of hand: more detailed profiles can simply return to an unwieldy mess of numbers. A better approach to constructing cluster profiles is be to draw the distributions of cluster members' data. To do this we need to "tidy up" the dataset. A tidy dataset (Wickham, 2014) is one where every row is an observation, and every column is a variable. Thus, a few steps are required to tidy up our labelled data:

```
# Index db on cluster ID
tidy_db = db.set_index('k5cls')
# Keep only variables used for clustering
tidy_db = tidy_db[cluster_variables]
# Stack column names into a column, obtaining
# a "long" version of the dataset
tidy_db = tidy_db.stack()
# Take indices into proper columns
tidy_db = tidy_db.reset_index()
# Rename column names
tidy_db = tidy_db.rename(columns={
                         'level_1': 'Attribute',
                         0: 'Values'})
# Check out result
tidy_db.head(20)
```

| | k5cls | Attribute | Values |
|---|---|---|---|
| 0 | 0 | median_house_value | 361900.000000 |
| 1 | 0 | pct_white | 0.768073 |
| 2 | 0 | pct_rented | 0.290393 |
| 3 | 0 | pct_hh_female | 0.098463 |
| 4 | 0 | pct_bachelor | 0.022244 |
| 5 | 0 | median_no_rooms | 5.800000 |
| 6 | 0 | income_gini | 0.360800 |
| 7 | 0 | median_age | 38.200000 |
| 8 | 0 | tt_work | 1362.000000 |
| 9 | 4 | median_house_value | 452900.000000 |
| 10 | 4 | pct_white | 0.814867 |
| 11 | 4 | pct_rented | 0.469031 |
| 12 | 4 | pct_hh_female | 0.110593 |
| 13 | 4 | pct_bachelor | 0.000000 |
| 14 | 4 | median_no_rooms | 5.300000 |
| 15 | 4 | income_gini | 0.397900 |
| 16 | 4 | median_age | 38.600000 |
| 17 | 4 | tt_work | 2175.000000 |
| 18 | 2 | median_house_value | 803900.000000 |
| 19 | 2 | pct_white | 0.863097 |

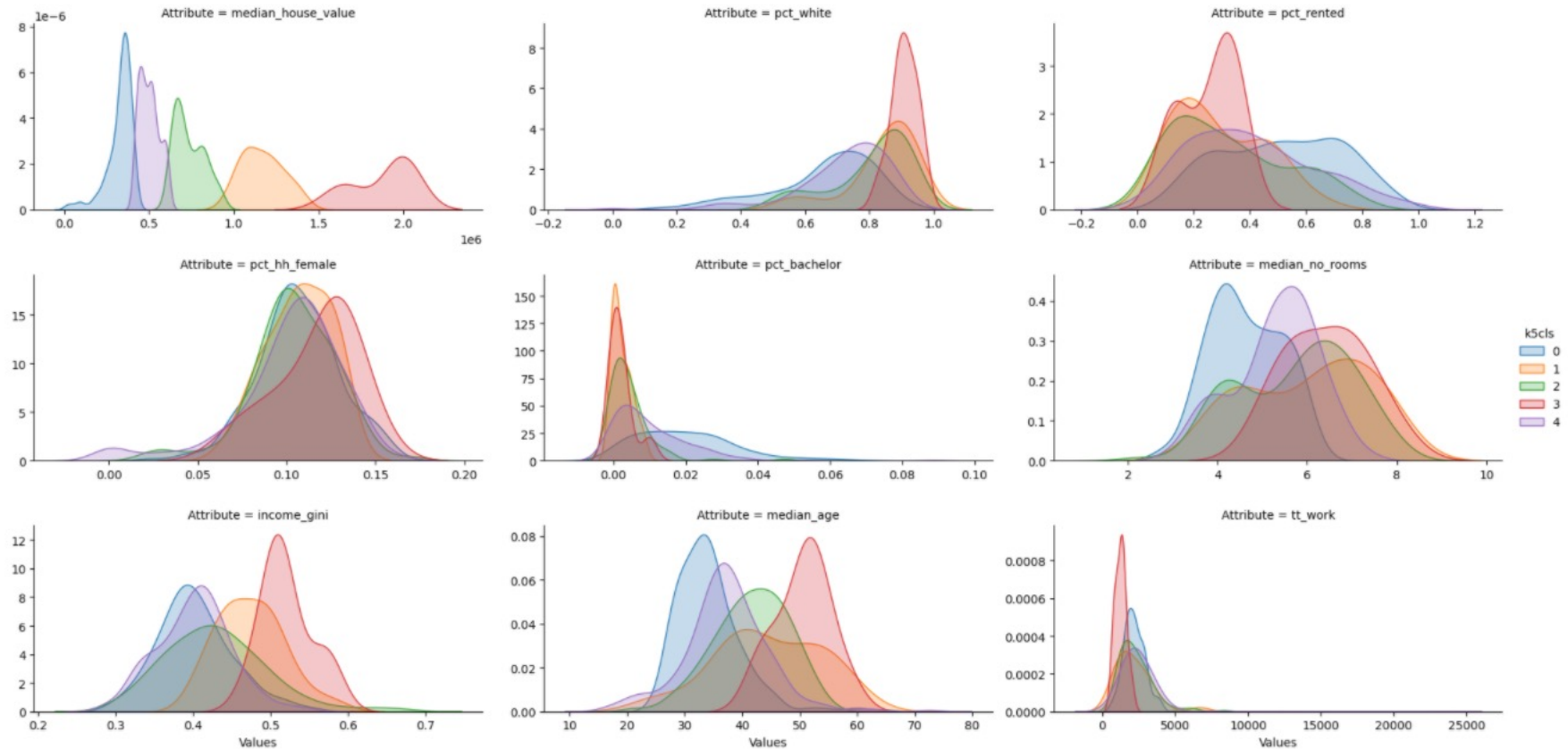
```
len(tidy_db)
```

```
5652
```

Note: For each census tract it shows the cluster number and the values of the 9 variables.

It has 5652 rows, 9*628

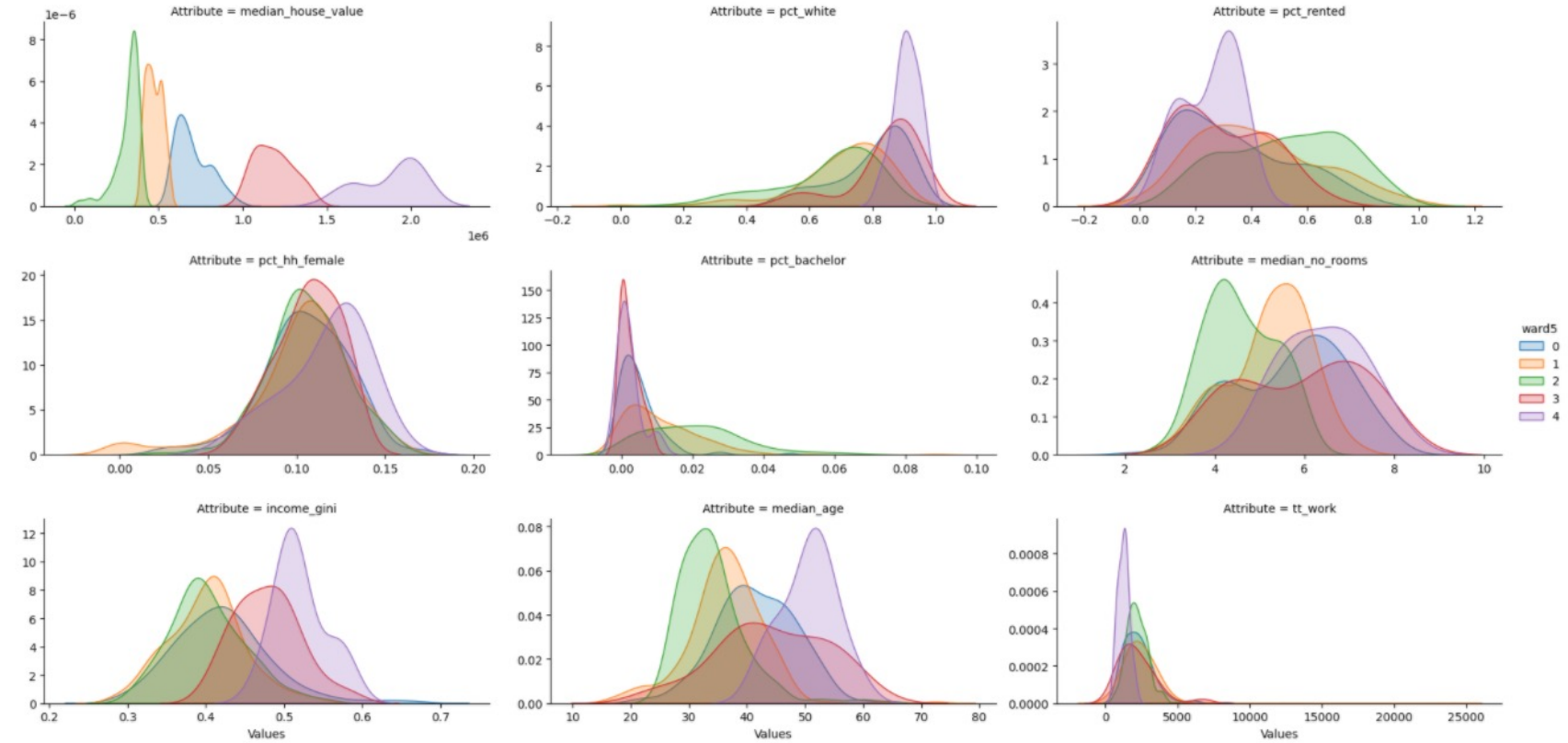
```
# Setup the facets
facets = seaborn.FacetGrid(data=tidy_db, col='Attribute', hue='k5cls', \
                  sharey=False, sharex=False, aspect=2, col_wrap=3)
# Build the plot from `sns.kdeplot`
_ = facets.map(seaborn.kdeplot, 'Values', shade=True).add_legend()
plt.show()
```



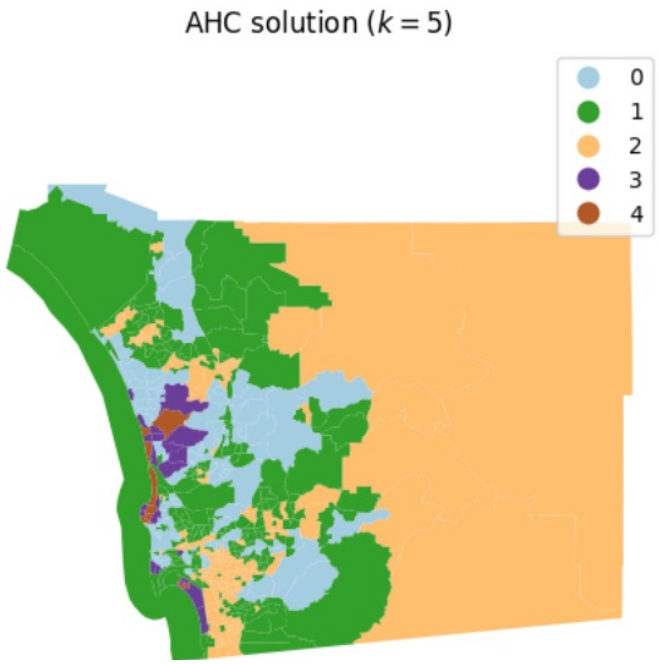

This allows us to see that, while some attributes such as the percentage of female households ( `pct_hh_female` ) display largely the same distribution for each cluster, others paint a much more divided picture (e.g. `median_house_value` ). Taken altogether, these graphs allow us to start delving into the multidimensional complexity of each cluster and the types of areas behind them.

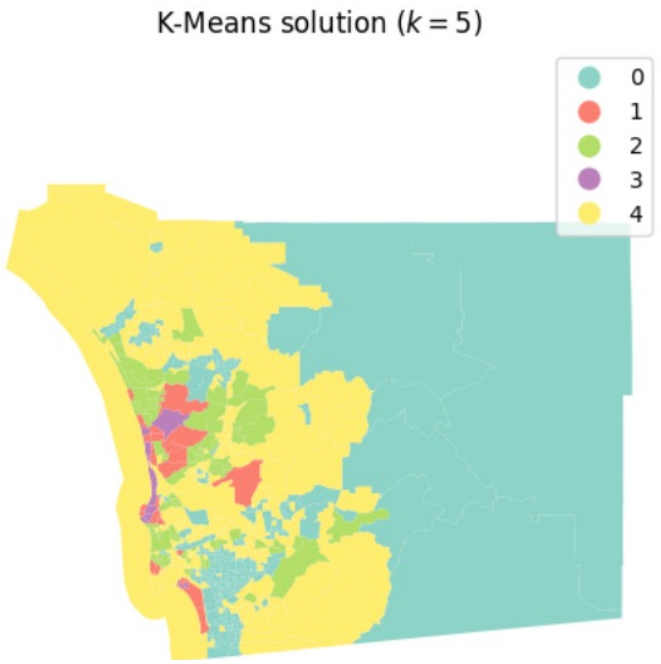
# Other Clustering Algorithm (Agglomerative Hierarchical Clustering)

```
# Setup the facets
facets = seaborn.FacetGrid(data=tidy_db, col='Attribute', hue='ward5', \
                  sharey=False, sharex=False, aspect=2, col_wrap=3)
# Build the plot as a `sns.kdeplot`
_ = facets.map(seaborn.kdeplot, 'Values', shade=True).add_legend()
plt.show()
```



# A few members (tracts) change cluster but the overall property of the clusters remain




```
ward5
0    141
1    233
2    222
3     23
4      9
dtype: int64
```

| ward5 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| median_house_value | 703765.957 | 473097.931 | 316161.712 | 1184173.913 | 1876867.222 |
| pct_white | 0.799 | 0.706 | 0.656 | 0.842 | 0.909 |
| pct_rented | 0.327 | 0.420 | 0.523 | 0.293 | 0.251 |
| pct_hh_female | 0.105 | 0.102 | 0.106 | 0.107 | 0.117 |
| pct_bachelor | 0.005 | 0.011 | 0.021 | 0.002 | 0.002 |
| median_no_rooms | 5.695 | 5.222 | 4.575 | 5.939 | 6.422 |
| income_gini | 0.421 | 0.401 | 0.403 | 0.478 | 0.520 |
| median_age | 41.535 | 36.389 | 34.062 | 44.261 | 50.544 |
| tt_work | 2305.206 | 2556.399 | 2172.563 | 2201.913 | 1237.778 |

```
k5cls
0    252
1     25
2    102
3      9
4    240
dtype: int64
```

| k5cls | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| median_house_value | 326728.97 | 1168004.00 | 736881.37 | 1876867.22 | 500787.57 |
| pct_white | 0.66 | 0.84 | 0.81 | 0.91 | 0.72 |
| pct_rented | 0.51 | 0.29 | 0.33 | 0.25 | 0.41 |
| pct_hh_female | 0.11 | 0.11 | 0.10 | 0.12 | 0.10 |
| pct_bachelor | 0.02 | 0.00 | 0.00 | 0.00 | 0.01 |
| median_no_rooms | 4.63 | 6.02 | 5.73 | 6.42 | 5.29 |
| income_gini | 0.40 | 0.47 | 0.43 | 0.52 | 0.40 |
| median_age | 34.25 | 44.44 | 41.60 | 50.54 | 37.21 |
| tt_work | 2187.75 | 2182.00 | 2336.28 | 1237.78 | 2539.43 |

Cluster 0 (2 AHC) has lowest house value: 326k, lowest median age 34 and smaller number of rooms 4.63 and 51% rented houses.

Cluster 4 (1 AHC) has the second less expensive house value median of 500k, longest average travel time to work of 42min and 41% rentals.

Cluster 2 (0 AHC) has intermediate values of the data set, in term of of median house value 736k, age (41.60) , number of rooms (5.73) and rental % (33%)

Cluster 1 (3 AHC) is the second most expensive in house value (1168k) with 29% renters 6 rooms and median age 44

Cluster 3 (4 AHC) groups the 9 most expensive tracts with median house value of 1876k, 25% Rented, 6.42 rooms, median age 50, largest gini 0.52 and shortest travel time to work of 20.6 min

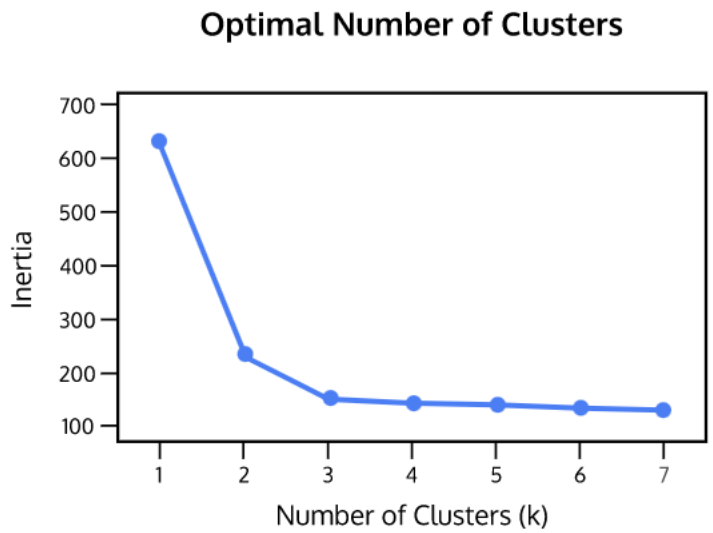
# How to quantitatively evaluate the partition in K  clusters?

- K-Means Inertia (elbow method)
- Silhouette Coefficient or silhouette score

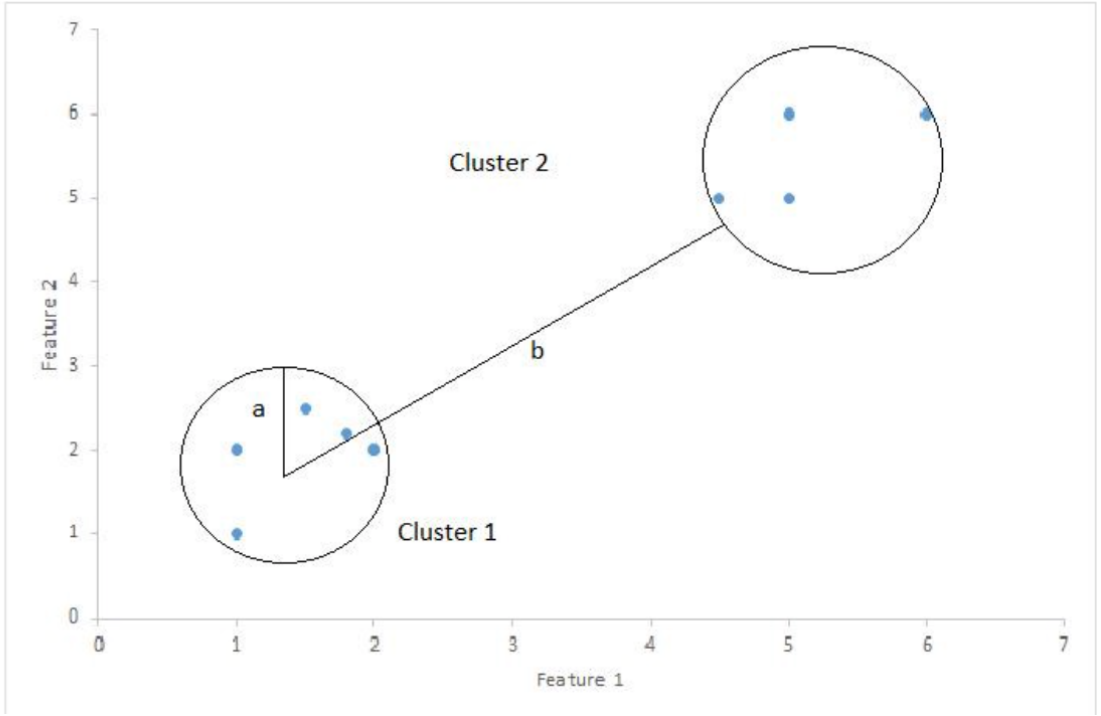
**K-Means: Inertia**

*Inertia* measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster.

To find the optimal K for a dataset, use the *Elbow method*; find the point where the decrease in inertia begins to slow. K=3 is the "elbow" of this graph.



**Silhouette Coefficient or silhouette score** is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1. 1: Means clusters are well apart from each other and clearly distinguished




Silhouette Score = (b-a)/max(a,b)

where

a= average intra-cluster distance i.e the average distance between each point within a cluster.

b= average inter-cluster distance i.e the average distance between all clusters.

Evaluation Criteria measure the average intra-cluster distance of a given clustering result vs. the inter cluster distance

The Silhouette value is in the range **[-1,1]**

# Criteria to decide the number of clusters

1. Check the values of K-inertia and Silhouette Score

2. Decide the clusters based on a story, what did you learn? How can you Identify each cluster distinctively.

# Data Science Story 2 (Food Stamps in GA)

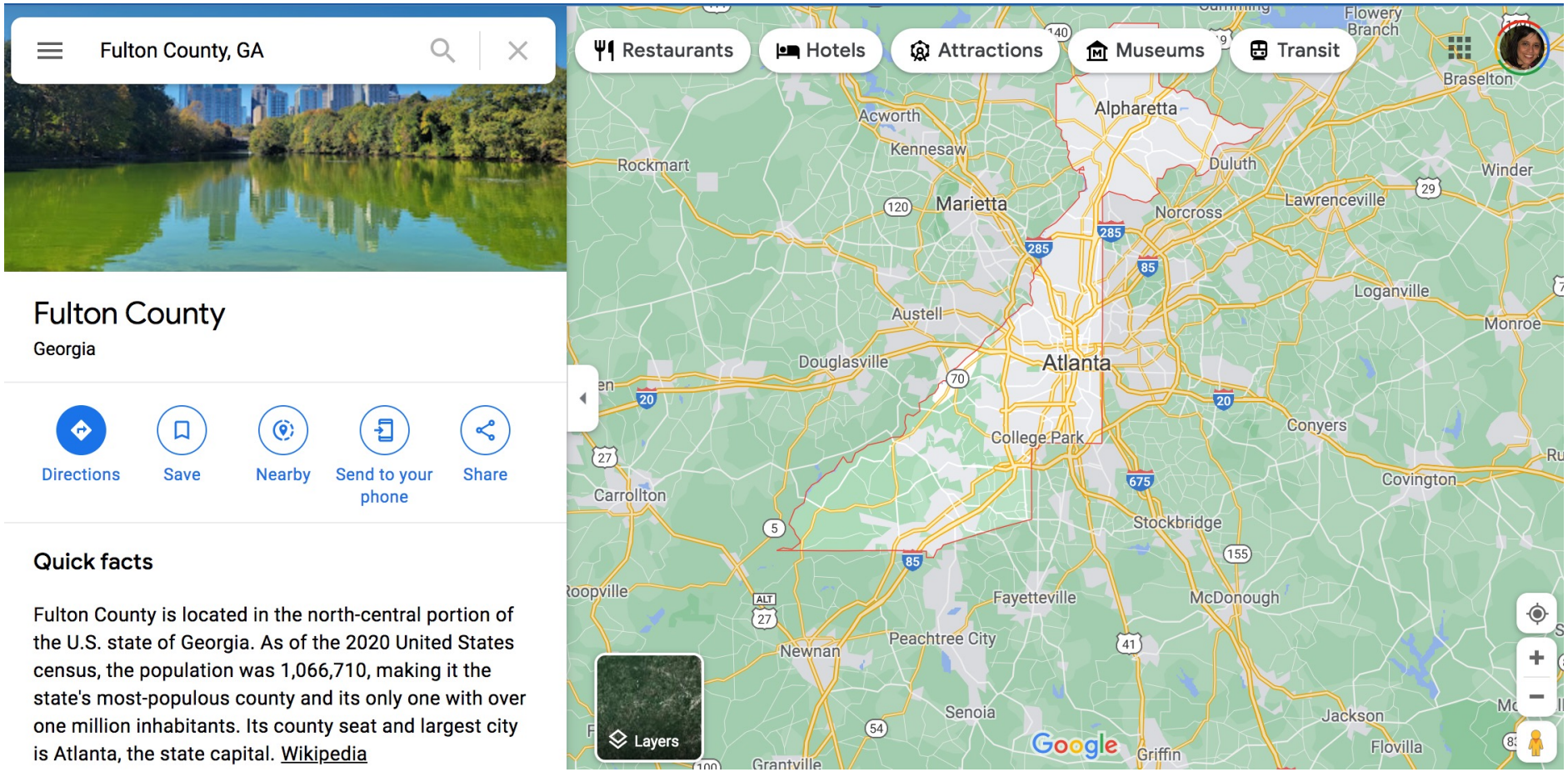
Clusterring_FultonGA_FoodStamps

```
vars_to_download = {
    "B02001_002E": "total_pop_white",      # Total white population
    "B02001_003E": "total_pop_black",      #Total black population
    "B01003_001E": "total_pop",            # Total population
    "B09019_001E": "hh_total",             # Total households
    "B15003_002E": "total_no_bachelor",       # Total w/o schooling completed
    "B01002_001E": "median_age",           # Median age
    "B19013_001E": "median_hh_income",      # Median household income
    "B19058_001E": "SNAP_hh",               # Households receiving Food Stamps/SNAP
    "B08015_001E": "access_to_vehicle"    # Workers over age 16 that drove alone to work by car, van, truck
}
vars_to_download_l = list(vars_to_download.keys())
```

http://cenpy-devs.github.io/cenpy/generated/cenpy.products.ACS.html

B15003_002E Estimate!!Total!!No schooling completed

```python
# Extracting census variables from Fulton County, GA
db = acs.from_county("Fulton, GA",
                     level="tract",
                     variables=vars_to_download_1
                    )
```



```
1 db.head()
```

| | GEOID | geometry | B01002_001E | B01003_001E | B02001_002E | B02001_003E | B08015_001E | B09019_001E | B15003_002E | B19013_001E | B19058_001E |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 13121007703 | POLYGON ((-9408543.140 3988685.680, -9408541.1... | 38.1 | 4403.0 | 42.0 | 4314.0 | 1155.0 | 4403.0 | 23.0 | 42150.0 | 1518.0 |
| **1** | 13121007807 | POLYGON ((-9408051.660 3997979.050, -9408034.2... | 27.5 | 3564.0 | 60.0 | 3238.0 | 445.0 | 3564.0 | 24.0 | 21912.0 | 996.0 |
| **2** | 13121010508 | POLYGON ((-9406046.240 3974350.080, -9405997.5... | 36.5 | 3503.0 | 171.0 | 3273.0 | 1230.0 | 3503.0 | 16.0 | 46983.0 | 1186.0 |
| **3** | 13121008302 | POLYGON ((-9402394.850 3996615.480, -9402284.8... | 49.3 | 1653.0 | 21.0 | 1622.0 | 385.0 | 1653.0 | 0.0 | 28949.0 | 671.0 |
| **4** | 13121006601 | POLYGON ((-9398620.670 3988652.630, -9398605.4... | 35.4 | 2087.0 | 329.0 | 1715.0 | 615.0 | 2087.0 | 26.0 | 36250.0 | 764.0 |

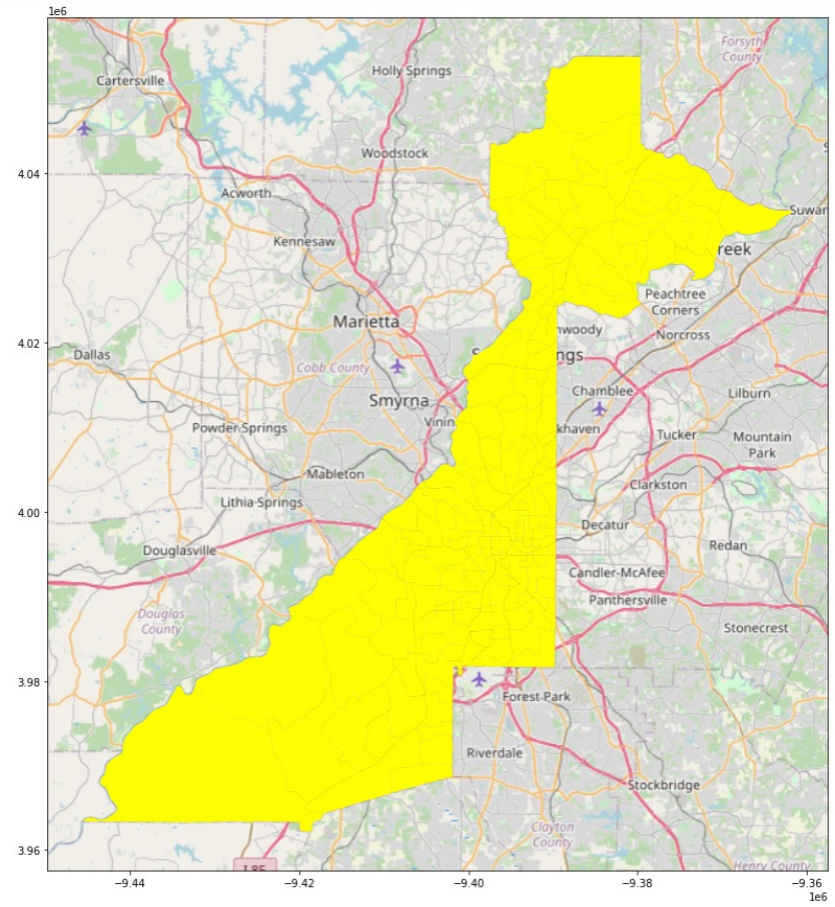
```
var_names = acs.variables\
               .reindex(vars_to_download)\
               [["label", "concept"]]\
               .reset_index()\
               .rename(columns={"index": "var_id"})
var_names["short_name"] = var_names["var_id"].map(vars_to_download)
```

:
```
db.head()
```

:

| cess_to_vehicle | hh_total | total_bachelor | median_hh_income | SNAP_hh | NAME | state | county | tract | area_sqm | pct_bachelor | pct_black | pct_white | pct_SNAP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1155.0 | 4403.0 | 23.0 | 42150.0 | 1518.0 | Census Tract 77.03, Fulton County, Georgia | 13 | 121 | 007703 | 4.531670 | 0.005224 | 0.979787 | 0.009539 | 0.344765 |
| 445.0 | 3564.0 | 24.0 | 21912.0 | 996.0 | Census Tract 78.07, Fulton County, Georgia | 13 | 121 | 007807 | 2.898744 | 0.006734 | 0.908530 | 0.016835 | 0.279461 |
| 1230.0 | 3503.0 | 16.0 | 46983.0 | 1186.0 | Census Tract 105.08, Fulton County, Georgia | 13 | 121 | 010508 | 3.906751 | 0.004568 | 0.934342 | 0.048815 | 0.338567 |
| 385.0 | 1653.0 | 0.0 | 28949.0 | 671.0 | Census Tract 83.02, Fulton County, Georgia | 13 | 121 | 008302 | 1.890469 | 0.000000 | 0.981246 | 0.012704 | 0.405929 |
| 615.0 | 2087.0 | 26.0 | 36250.0 | 764.0 | Census Tract 66.01, Fulton County, Georgia | 13 | 121 | 006601 | 1.954144 | 0.012458 | 0.821754 | 0.157643 | 0.366076 |

```python
# Visualizing the food desert (my area of analysis)

ax = db.plot(figsize=(15, 15), alpha=0.2, color="k")
db.plot(ax=ax, color="yellow")
contextily.add_basemap(ax, url=contextily.sources.OSM_A);

```




## Basic statistics of the Data

1e6

```
44]: 1 db.describe()
```

44]:

| | median_age | total_pop | total_pop_white | total_pop_black | access_to_vehicle | hh_total | total_bachelor | median_hh_income | SNAP_hh | area_s |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 204.000000 | 204.000000 | 204.000000 | 204.000000 | 204.000000 | 204.000000 | 204.000000 | 204.000000 | 204.000000 | 204.0000 |
| mean | 36.241872 | 4953.039216 | 2229.887255 | 2186.039216 | 1851.243781 | 4953.039216 | 24.602941 | 66844.262376 | 1920.833333 | 6.7833 |
| std | 6.990583 | 2972.071006 | 2155.574353 | 2604.308540 | 1254.172915 | 2972.071006 | 30.511472 | 42419.178750 | 1098.389713 | 15.6970 |
| min | 12.400000 | 0.000000 | 0.000000 | 0.000000 | 200.000000 | 0.000000 | 0.000000 | 9815.000000 | 0.000000 | 0.0905 |
| 25% | 32.400000 | 2590.500000 | 246.000000 | 545.000000 | 821.250000 | 2590.500000 | 0.000000 | 31153.250000 | 1091.250000 | 1.4674 |
| 50% | 35.500000 | 4463.000000 | 1775.000000 | 1364.000000 | 1695.000000 | 4463.000000 | 16.000000 | 57066.500000 | 1834.000000 | 3.3363 |
| 75% | 40.300000 | 6091.500000 | 3832.500000 | 2488.000000 | 2502.500000 | 6091.500000 | 33.250000 | 93487.000000 | 2565.500000 | 6.8530 |
| max | 67.900000 | 17958.000000 | 12255.000000 | 16075.000000 | 6555.000000 | 17958.000000 | 196.000000 | 200179.000000 | 6228.000000 | 185.1171 |

```
db.to_csv('fulton.csv', index=False)
```

```
! rm -f atlanta_tracts.gpkg
db.to_file("atlanta.gpkg", driver="GPKG")
```

## Kmeans Clustering (Elbow Method and Silhouette Scores)

```
from esda.moran import Moran
import libpysal.weights.set_operations as Wsets
from libpysal.weights import Queen, KNN
import seaborn
import pandas as pd
import geopandas
import numpy
from sklearn.cluster import KMeans, AgglomerativeClustering
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('fulton.csv')
df.head()
```

| | GEOID | geometry | median_age | total_pop | total_pop_white | total_pop_black | access_to_vehicle | hh_total | total_bachelor | median_hh_income |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 13121007703 | POLYGON ((-9408543.140000001 3988685.68, -9408... | 38.1 | 4403.0 | 42.0 | 4314.0 | 1155.0 | 4403.0 | 23.0 | 42150.0 |
| **1** | 13121007807 | POLYGON ((-9408051.66 3997979.05, -9408034.289... | 27.5 | 3564.0 | 60.0 | 3238.0 | 445.0 | 3564.0 | 24.0 | 21912.0 |
| **2** | 13121010508 | POLYGON ((-9406046.24 3974350.08, -9405997.59 ... | 36.5 | 3503.0 | 171.0 | 3273.0 | 1230.0 | 3503.0 | 16.0 | 46983.0 |

```
# Selecting my cluster variables
df_Short = df[['pct_white', 'pct_black', 'pct_bachelor', 'pct_SNAP', 'median_age', 'median_hh_income']]
```

```
import sklearn.cluster as cluster
```

```
K=range(1,12)
wss = []
for k in K:
    kmeans=cluster.KMeans(n_clusters=k,init="k-means++")
    kmeans=kmeans.fit(df_Short)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)
```

Two steps: Instantiate the method (line 4) and call it (line 5)

inertia_ is an output of Kmeans

```
mycenters = pd.DataFrame({'Clusters' : K, 'WSS' : wss})
mycenters
```

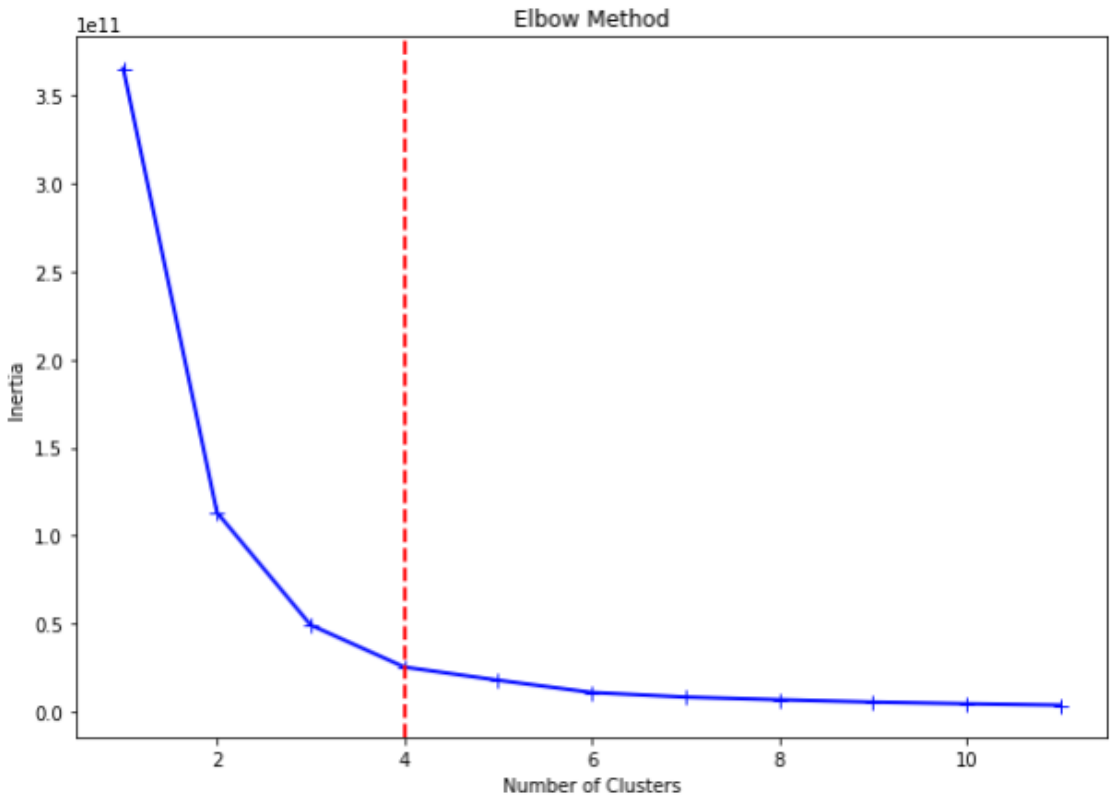
| | Clusters | WSS |
|---|---|---|
| 0 | 1 | 3.652755e+11 |
| 1 | 2 | 1.131231e+11 |
| 2 | 3 | 4.917238e+10 |
| 3 | 4 | 2.530135e+10 |
| 4 | 5 | 1.783359e+10 |
| 5 | 6 | 1.088108e+10 |
| 6 | 7 | 8.259748e+09 |
| 7 | 8 | 6.751767e+09 |
| 8 | 9 | 5.421144e+09 |
| 9 | 10 | 4.475335e+09 |
| 10 | 11 | 3.699100e+09 |

***k*-means++**[1][2] is an algorithm for choosing the initial values (or "seeds") for the *k*-means clustering algorithm

This creates the arrays
Inertia (WSS) vs. K for the elbow method

```
# Using elbow method to select the correct number of clusters
```

```python
# Using elbow method to select the correct number of clusters

_ = plt.figure(figsize = (10,7))
_ = plt.plot(range(1,12), wss, linewidth = 2, color = 'blue', marker='+', markersize = 8)
_ = plt.title('Elbow Method', fontsize = 12)
_ = plt.xlabel('Number of Clusters',fontsize = 10)
_ = plt.ylabel('Inertia',fontsize = 10)

n_clusters = 4

_ = plt.axvline(x = n_clusters, linewidth = 2, color = 'red', linestyle = '--')
_ = plt.show()
```



# sklearn.metrics.silhouette_score ¶

```
sklearn.metrics.silhouette_score(X, labels, *, metric='euclidean', sample_size=None, random_state=None, **kwds)
```

[source]

Compute the mean Silhouette Coefficient of all samples.

The Silhouette Coefficient is calculated using the mean intra-cluster distance (`a`) and the mean nearest-cluster distance (`b`) for each sample. The Silhouette Coefficient for a sample is `(b - a) / max(a, b)`. To clarify, `b` is the distance between a sample and the nearest cluster that the sample is not a part of. Note that Silhouette Coefficient is only defined if number of labels is `2 <= n_labels <= n_samples - 1`.

This function returns the mean Silhouette Coefficient over all samples. To obtain the values for each sample, use **`silhouette_samples`**.

The best value is 1 and the worst value is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar.

Read more in the User Guide.

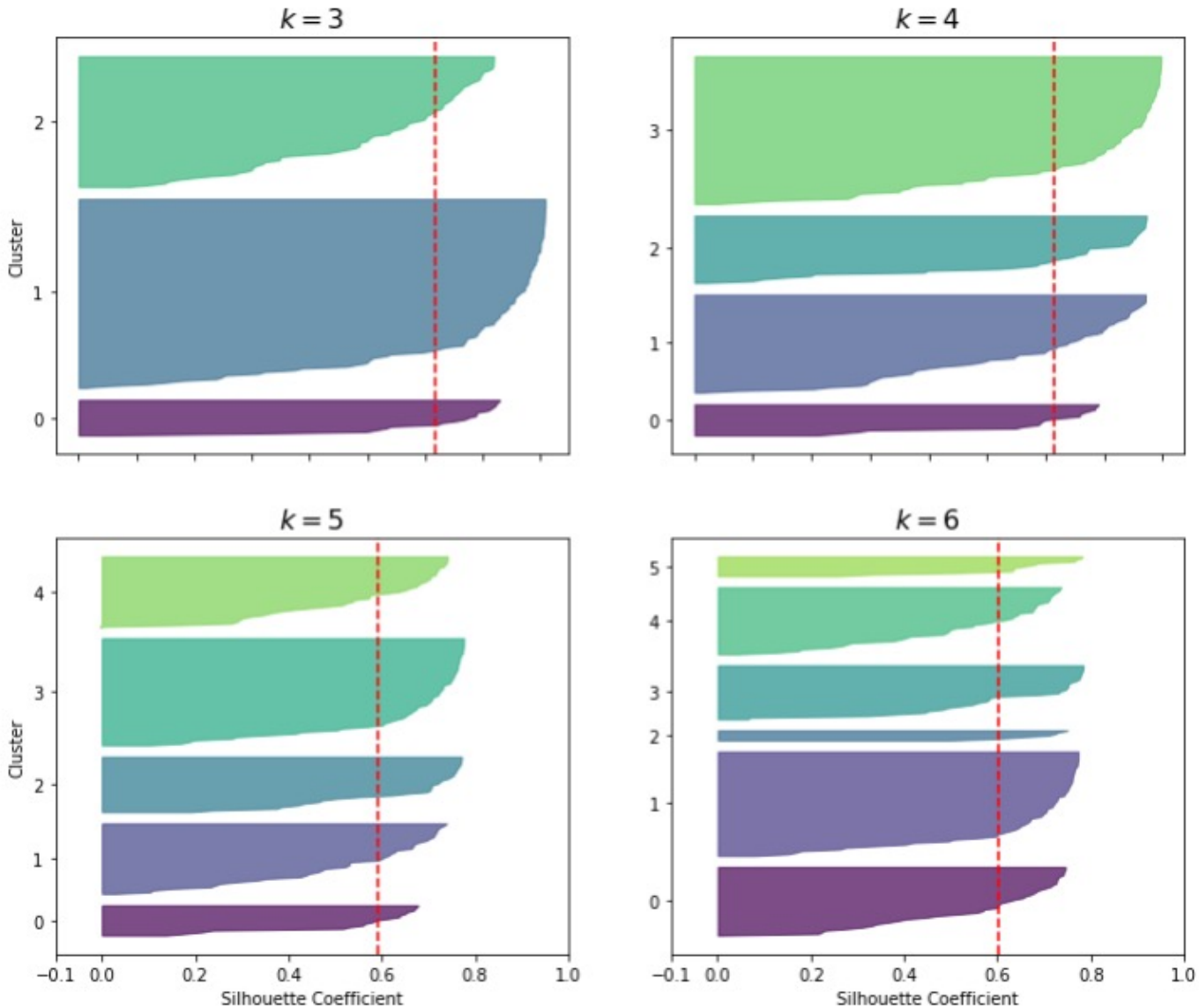
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

Data

Labeled Data in Clusters

```
kmeans_per_k = [KMeans(n_clusters=k, random_state=200).fit(df_Short) for k in range(2,13)]

silhouette_scores = [silhouette_score(df_Short, model.labels_)
                     for model in kmeans_per_k[0:]]
silhouette_scores
```

```
[0.6318406031651713,
 0.6169908455360216,
 0.6137312790738855,
 0.5917947190551022,
 0.600401661891466,
 0.5751446924939103,
 0.5570354583782806,
 0.5333345136151681,
 0.5509497316744281,
 0.5593658264732064,
 0.5603203087516221]
```




```
Silhouette score for k(clusters) = 2 is 0.6318406031651713
Silhouette score for k(clusters) = 3 is 0.6169908455360215
Silhouette score for k(clusters) = 4 is 0.613731279073885
Silhouette score for k(clusters) = 5 is 0.5917947190551015
Silhouette score for k(clusters) = 6 is 0.6004016618914658
Silhouette score for k(clusters) = 7 is 0.5751446924939094
Silhouette score for k(clusters) = 8 is 0.5570354583782787
Silhouette score for k(clusters) = 9 is 0.5333345136151703
Silhouette score for k(clusters) = 10 is 0.5509497316744304
Silhouette score for k(clusters) = 11 is 0.5593658264732276
Silhouette score for k(clusters) = 12 is 0.560320308751648
```

Note: The visualization code comes from the python library (not trivial)

# Clustering and Segmentation using PySAL

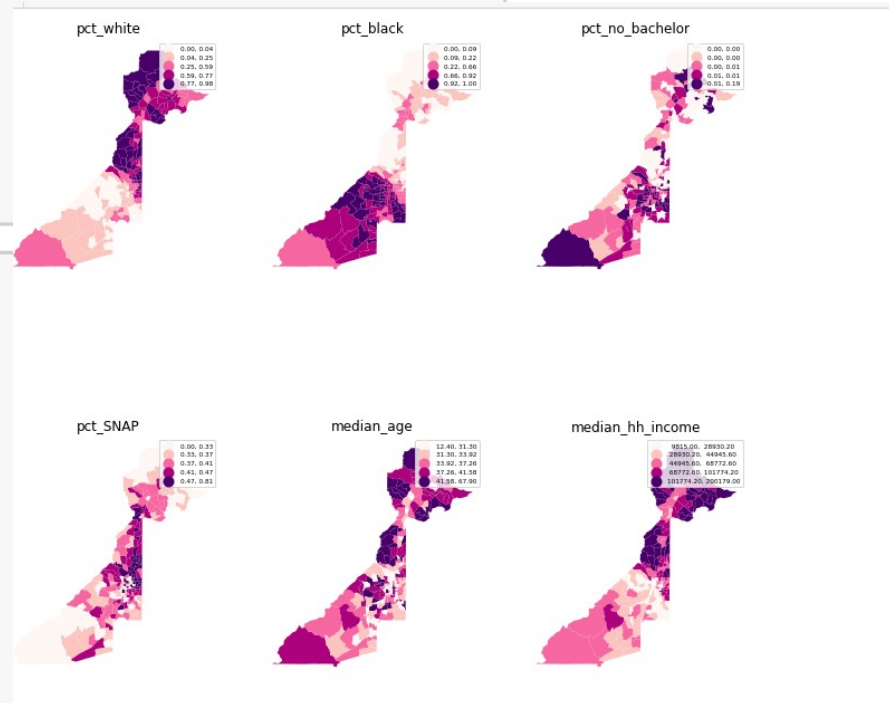
```
db = geopandas.read_file('atlanta.gpkg')
db.columns
```

```
/Users/marta/opt/anaconda3/lib/python3.7/site-packages/geopandas/geodataframe.py:422: RuntimeWarning: Sequential read
of iterator was interrupted. Resetting iterator. This can negatively impact the performance.
  for feature in features_lst:
```

```
Index(['GEOID', 'median_age', 'total_pop', 'total_pop_white',
       'total_pop_black', 'access_to_vehicle', 'hh_total', 'total_bachelor',
       'median_hh_income', 'SNAP_hh', 'NAME', 'state', 'county', 'tract',
       'area_sqm', 'pct_bachelor', 'pct_black', 'pct_white', 'pct_SNAP',
       'geometry'],
      dtype='object')
```

```
cluster_variables =  [
    'pct_white',           # Percent of tract population that is white
    'pct_black',           # Percent of tract population that is black
    'pct_bachelor',        # Percent of tract population with a Bachelors degree
    'pct_SNAP',
    'median_age',          # Median age of tract population
    'median_hh_income'     # Median household income
]
```

```
f, axs = plt.subplots(nrows=2, ncols=3, figsize=(12, 12))
# Make the axes accessible with single indexing
axs = axs.flatten()
# Start a loop over all the variables of interest
for i, col in enumerate(cluster_variables):
    # select the axis where the map will go
    ax = axs[i]
    # Plot the map
    db.plot(column=col, ax=ax, scheme='Quantiles',
            linewidth=0, cmap='RdPu')
    # Remove axis clutter
    ax.set_axis_off()
    # Set the axis title to the name of variable being plotted
    ax.set_title(col)
# Display the figure
plt.show()
```



## Measuring spatial autocorrelations

```
w = Queen.from_dataframe(db)
```

```
w.islands
```

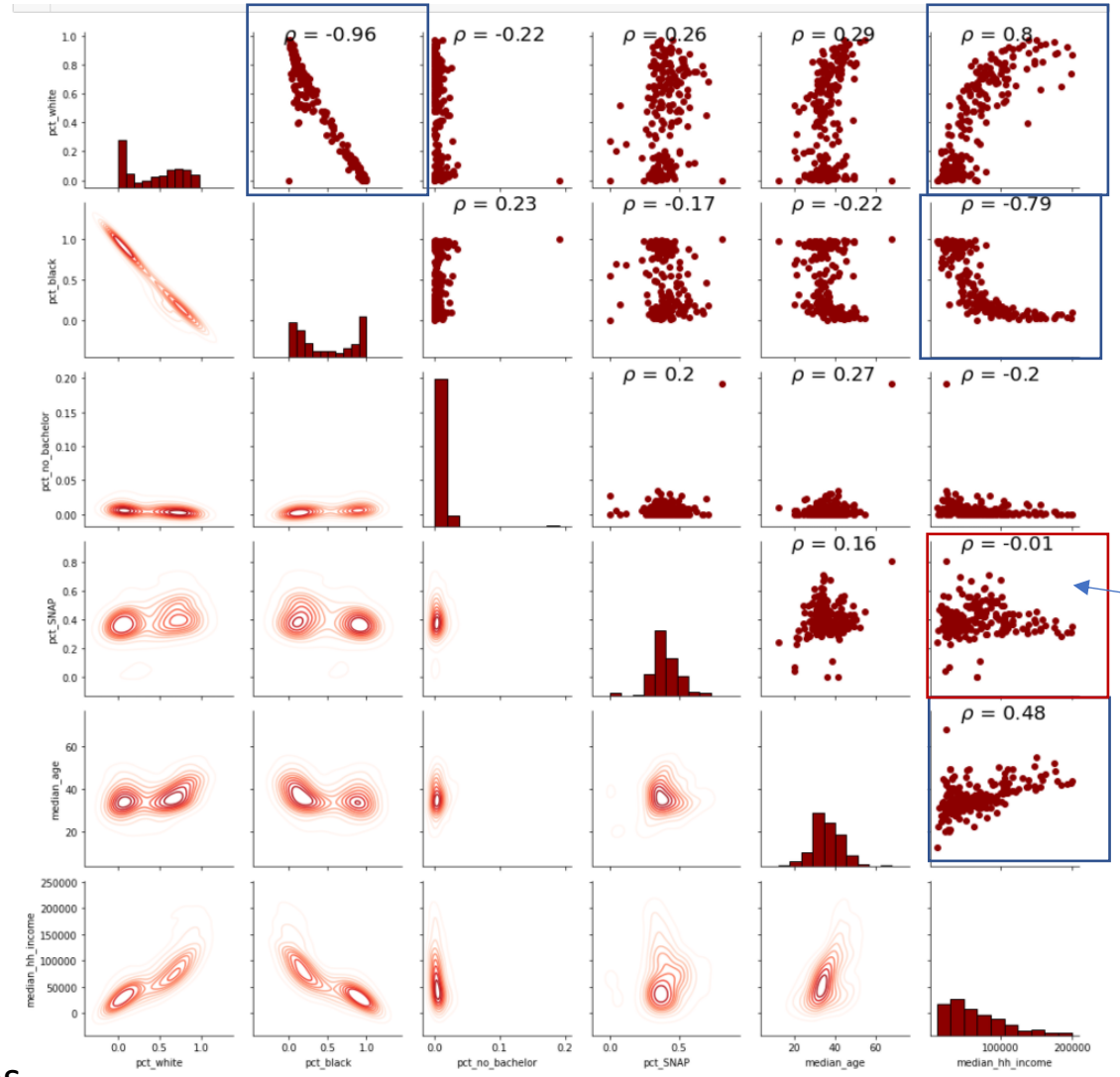
```
[]
```

```
# Set seed for reproducibility
numpy.random.seed(123456)
# Calculate Moran's I for each variable
mi_results = [Moran(db[variable], w) for variable in cluster_variables]
table = pd.DataFrame([(variable, res.I, res.p_sim) \
                          for variable,res \
                          in zip(cluster_variables, mi_results)
                         ], columns=['Variable', "Moran's I", 'P-value']
                        )\
              .set_index('Variable')
table
```

| Variable | Moran's I | P-value |
|---|---|---|
| pct_white | 0.868826 | 0.001 |
| pct_black | 0.890161 | 0.001 |
| pct_no_bachelor | 0.039838 | 0.061 |
| pct_SNAP | 0.238407 | 0.001 |
| median_age | 0.166408 | 0.001 |
| median_hh_income | 0.650458 | 0.001 |

Note: Interestingly the SNAP households do not show spatial autocorrelation, it should follow similar spatial autocorrelati
as income, if the adoption would follow income-based needs

$$\rho = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

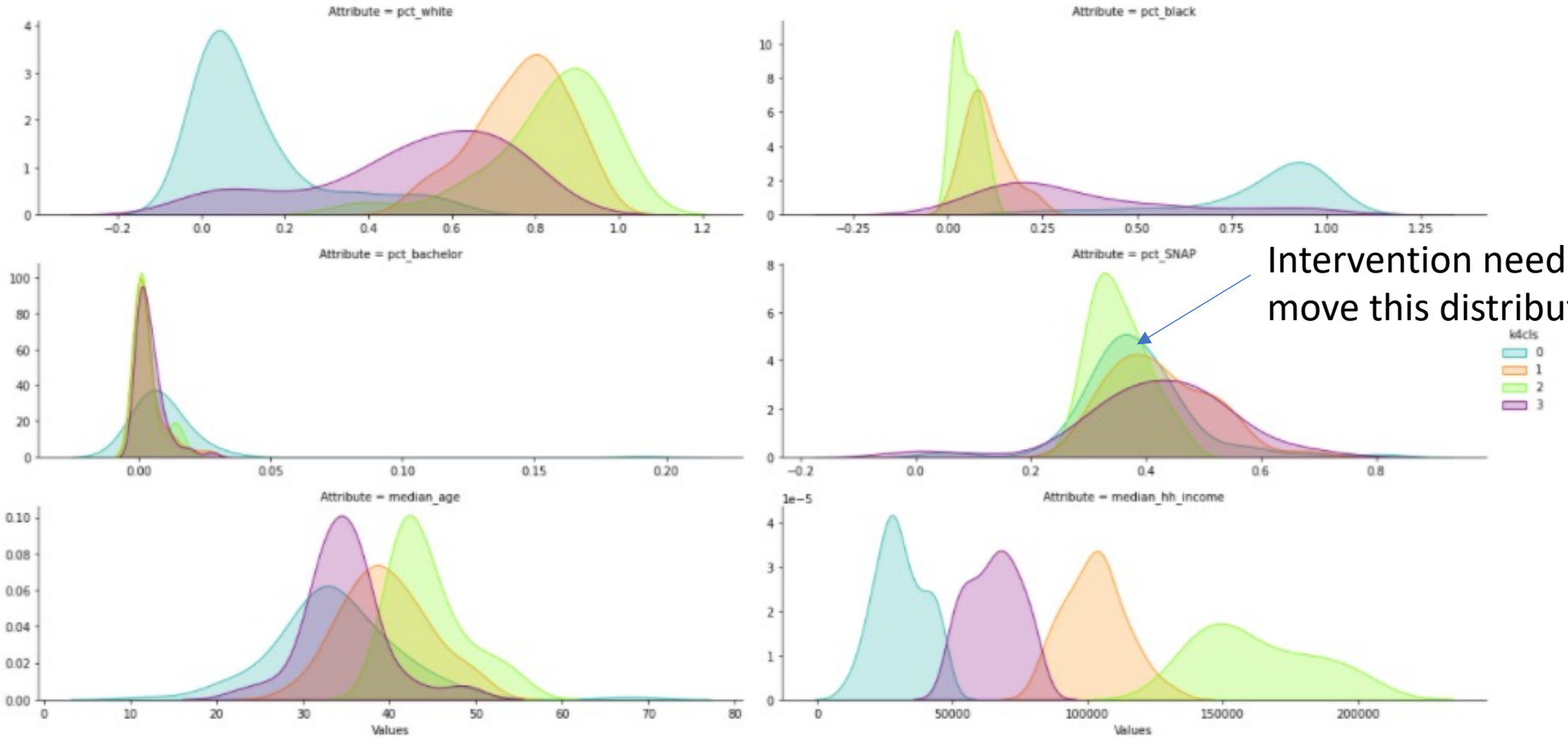



Surprise:
No strong correlation
With income
And SNAP

Note:
High correlation coefficient are expected based on common knowledge

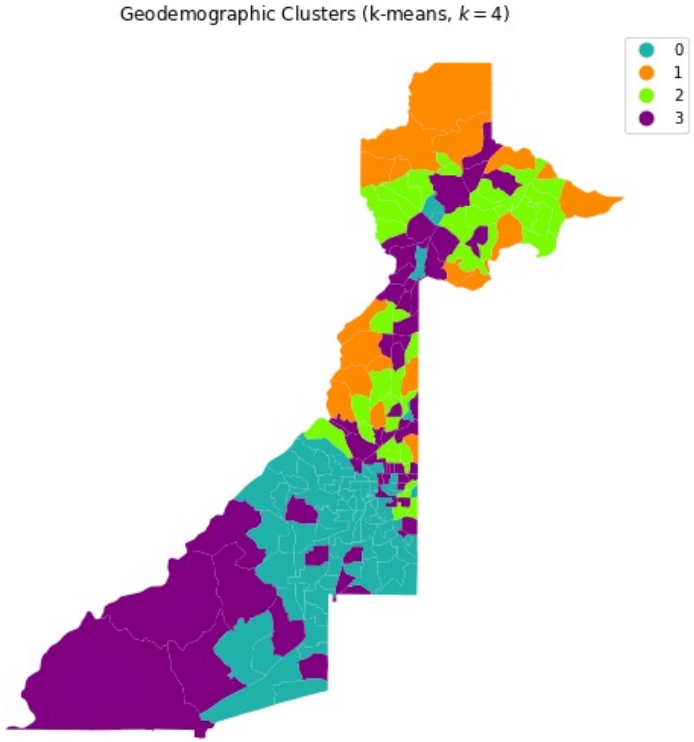
Surprising result: Low correlation of Income and pct of SNAP.
Policy action: We need to identify the places where we should promote or increase the adoption

```
# Setup the facets

color = {0:"lightseagreen", 1:"darkorange", 2:"lawngreen", 3:"purple"}
facets = seaborn.FacetGrid(data=tidy_db, col='Attribute', palette=color, hue='k4cls', \
                  sharey=False, sharex=False, aspect=3, col_wrap=2)

_ = facets.map(seaborn.kdeplot, 'Values', shade=True).add_legend()
```




Intervention needs to
move this distribution to the right

Surprise: Cluster 0 with the lowest income is the second with less % of SNAP

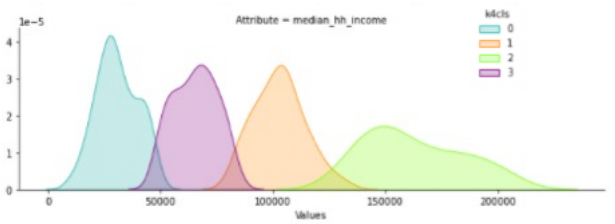



| k4cls | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| pct_white | 0.128 | 0.831 | 0.765 | 0.504 |
| pct_black | 0.822 | 0.048 | 0.101 | 0.356 |
| pct_no_bachelor | 0.010 | 0.003 | 0.003 | 0.004 |
| pct_SNAP | 0.382 | 0.353 | 0.425 | 0.417 |
| median_age | 33.834 | 44.532 | 39.615 | 34.811 |
| median_hh_income | 30579.414 | 162552.000 | 102675.975 | 65177.475 |

Observation: The method allowed us to locate cluster 0 as the tracts that need intervention, they have largest black population with lowest income ($30k per year) and the % of households with food stamps is only 38%

```
]: 1 # Grouping data table by cluster label and count observations
   2 k4sizes = db.groupby('k4cls').size()
   3 k4sizes
   4
```

```
]: k4cls
   0    87
   1    19
   2    40
   3    58
   dtype: int64
```



# Recommendations

- Check various values of K, see the distributions and make a conclusive Story to interpret the clusters and draw your conclusions

# For next class

- Do Assignment 2, part 1