**Introduction**

There are many models for **clustering** out there. In this notebook, we will be presenting the model that is considered one of the simplest models amongst them. Despite its simplicity, the **K-means** is vastly used for clustering in many data science applications, especially useful if you need to quickly discover insights from **unlabeled data**. In this notebook, you will learn how to use k-Means for customer segmentation.

Some real-world applications of k-means:

* Customer segmentation
* Understand what the visitors of a website are trying to accomplish
* Pattern recognition
* Machine learning
* Data compression

In this notebook we practice k-means clustering with 2 examples:

* k-means on a random generated dataset
* Using k-means for customer segmentation

Importing the libraries:

import random

import numpy as np

import malplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.datasets.samples\_generator import make\_blobs

%matplotlib inline

# k-Means on a randomly generated dataset

Lets create our own dataset for this lab!

First we need to set up a random seed. Use **numpy's random.seed()** function, where the seed will be set to **0**

np.random.seed(0)

Next we will be making *random clusters*of points by using the **make\_blobs**class. The **make\_blobs**class can take in many inputs, but we will be using these specific ones.  
  
**Input**

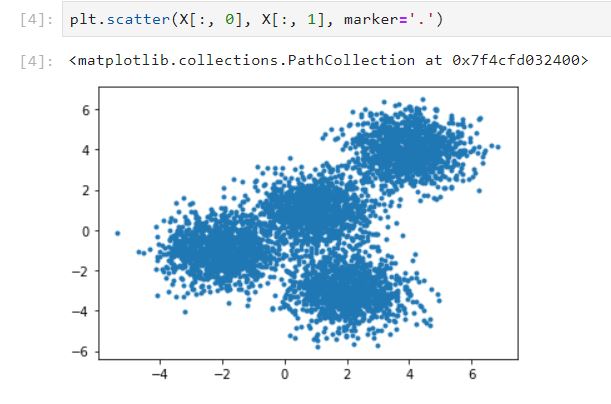
* **n\_samples**: The total number of points equally divided among clusters.
  + Value will be: 5000
* **centers**: The number of centers to generate, or the fixed center locations.
  + Value will be: [[4, 4], [-2, -1], [2, -3],[1,1]]
* **cluster\_std**: The standard deviation of the clusters.
  + Value will be: 0.9

**Output**

* **X**: Array of shape [n\_samples, n\_features]. (Feature Matrix)
  + The generated samples.
* **y**: Array of shape [n\_samples]. (Response Vector)
  + The integer labels for cluster membership of each sample.

X,y=make\_blobs(n\_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster\_std=0.9)

Now we will plot the data we randomly generated.

plt.scatter(X[:0],X[:1],marker=’.’)

**Setting up K-Means**

Now that we have our random data, let's set up our K-Means Clustering.

The KMeans class has many parameters that can be used, but we will be using these three:

* **init**: Initialization method of the centroids.
  + Value will be: "k-means++"
  + k-means++: Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence.
* **n\_clusters**: The number of clusters to form as well as the number of centroids to generate.
  + Value will be: 4 (since we have 4 centers)
* **n\_init**: Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia.
  + Value will be: 12

Initialize KMeans with these parameters, where the output parameter is called **k\_means**.

k\_means=KMeans(init=’k-means++’,n\_clusters=4,n\_init=12)

Now let's fit the KMeans model with the feature matrix we created above, **X**

k\_means.fit(X)

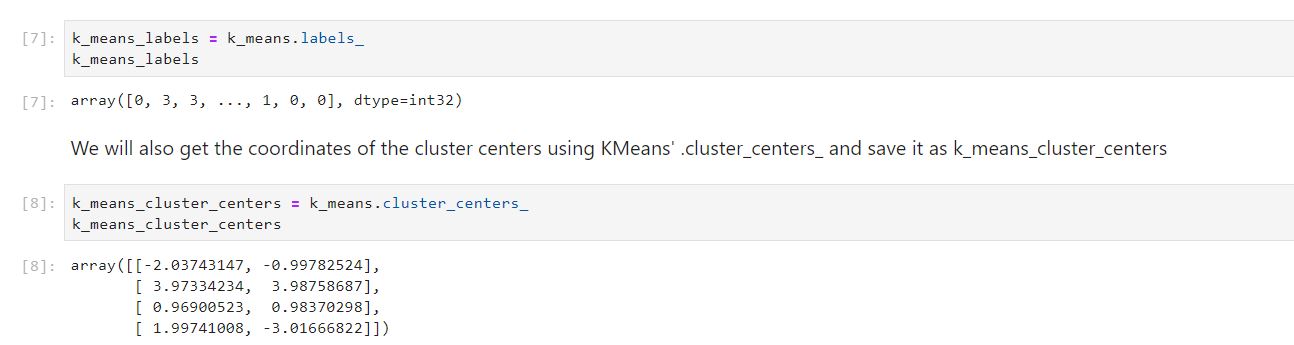
Now let's grab the labels for each point in the model using KMeans' **.labels\_**attribute and save it as **k\_means\_labels**

k\_means\_labels=k\_means.labels\_

k\_means\_labels

We will also get the coordinates of the cluster centers using KMeans' **.cluster\_centers\_**and save it as **k\_means\_cluster\_centers**

k\_means\_cluster\_clusters=k\_means. cluster\_clusters \_

k\_means\_ cluster\_clusters

To plot the k-means

# Initialize the plot with the specified dimensions.

fig = plt.figure(figsize=(6, 4))

# Colors uses a color map, which will produce an array of colors based on

# the number of labels there are. We use set(k\_means\_labels) to get the

# unique labels.

colors = plt.cm.Spectral(np.linspace(0, 1, len(set(k\_means\_labels))))

# Create a plot

ax = fig.add\_subplot(1, 1, 1)

# For loop that plots the data points and centroids.

# k will range from 0-3, which will match the possible clusters that each

# data point is in.

for k, col in zip(range(len([[4,4], [-2, -1], [2, -3], [1, 1]])), colors):

# Create a list of all data points, where the data poitns that are

# in the cluster (ex. cluster 0) are labeled as true, else they are

# labeled as false.

my\_members = (k\_means\_labels == k)

# Define the centroid, or cluster center.

cluster\_center = k\_means\_cluster\_centers[k]

# Plots the datapoints with color col.

ax.plot(X[my\_members, 0], X[my\_members, 1], 'w', markerfacecolor=col, marker='.')

# Plots the centroids with specified color, but with a darker outline

ax.plot(cluster\_center[0], cluster\_center[1], 'o', markerfacecolor=col, markeredgecolor='k', markersize=6)

# Title of the plot

ax.set\_title('KMeans')

# Remove x-axis ticks

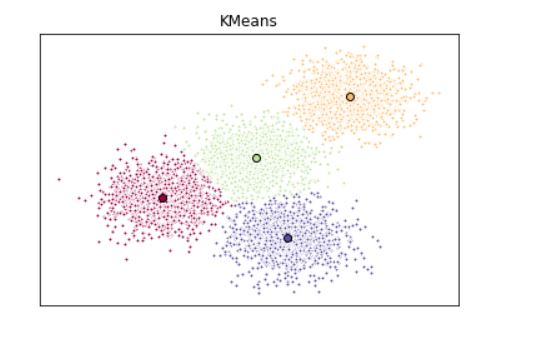
ax.set\_xticks(())

# Remove y-axis ticks

ax.set\_yticks(())

# Show the plot

plt.show()



# Customer Segmentation with K-Means

Imagine that you have a customer dataset, and you need to apply customer segmentation on this historical data. Customer segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics. It is a significant strategy as a business can target these specific groups of customers and effectively allocate marketing resources. For example, one group might contain customers who are high-profit and low-risk, that is, more likely to purchase products, or subscribe for a service. A business task is to retaining those customers. Another group might include customers from non-profit organizations. And so on.

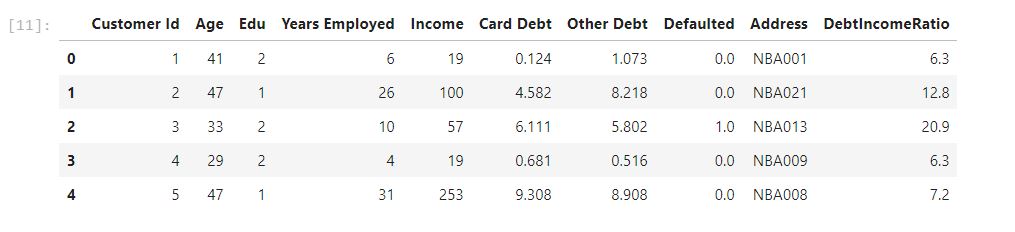
Lets download the dataset. To download the data, we will use **!wget** to download it from IBM Object Storage.

!wget -O Cust\_Segmentation.csv <https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/ML0101ENv3/labs/Cust_Segmentation.csv>

Now load the data file

import pandas as pd

cust\_df=pd.read\_csv(‘Cust\_segmentation.csv’)

cust\_df.head()

## Pre-processing

As you can see, **Address** in this dataset is a categorical variable. k-means algorithm isn't directly applicable to categorical variables because Euclidean distance function isn't really meaningful for discrete variables. So, lets drop this feature and run clustering.

df=cust\_df.drop(‘Address’,axis=1)

df.head()

#### Normalizing over the standard deviation

Now let's normalize the dataset. But why do we need normalization in the first place? Normalization is a statistical method that helps mathematical-based algorithms to interpret features with different magnitudes and distributions equally. We use **StandardScaler()** to normalize our dataset.

from sklearn.preprocessing import StandardScaler

X=df.values[:,1:]

X=np.nan\_to\_num(X)

Clus\_dataset=StardardScaler().fit\_transform(X)

Clus\_dataset

Now modeling using k-means

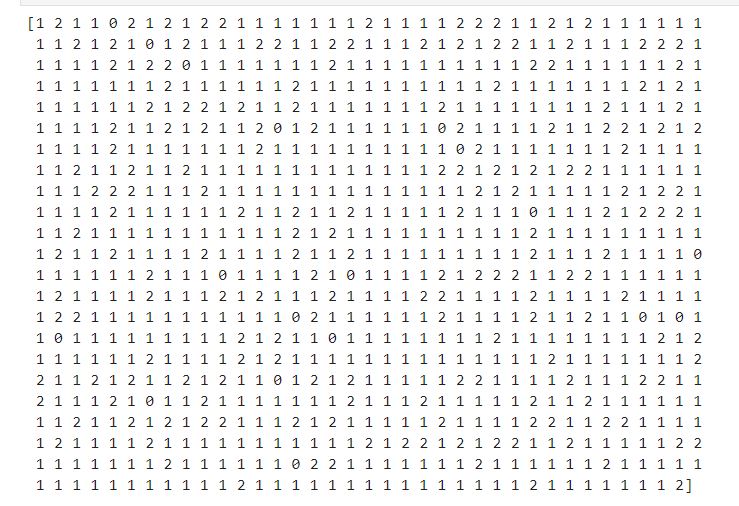
clusterNum=3

k\_means=KMeans(init=’k-means++’,n\_clusters=clusterNum,n\_init=12)

k\_means.fit(X)

labels=k\_means.labels\_

print(labels)



Adding a column for labels

df[“Clus\_km”]=labels

Now we will group by with respect to the labels

df.group\_by(‘Clus\_km’).mean()

Plot age vs income graph

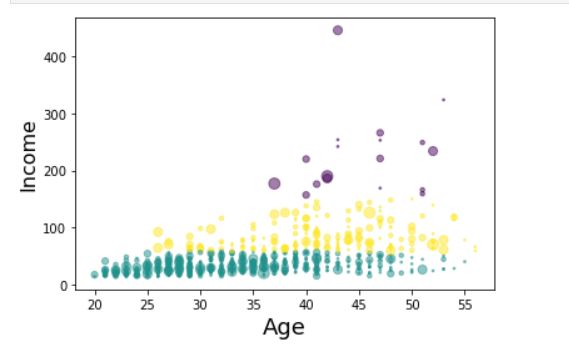
area = np.pi \* ( X[:, 1])\*\*2

plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)

plt.xlabel('Age', fontsize=18)

plt.ylabel('Income', fontsize=16)

plt.show()



from mpl\_toolkits.mplot3d import Axes3D

fig = plt.figure(1, figsize=(8, 6))

plt.clf()

ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()

# plt.ylabel('Age', fontsize=18)

# plt.xlabel('Income', fontsize=16)

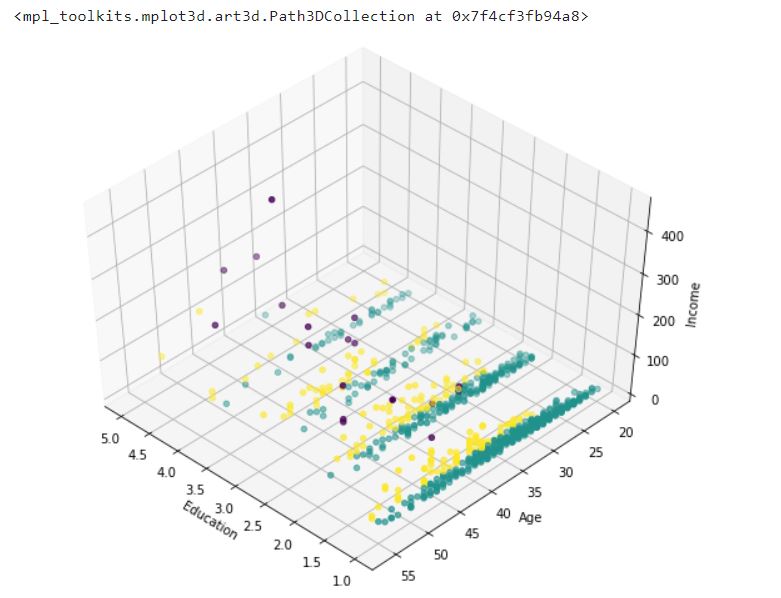
# plt.zlabel('Education', fontsize=16)

ax.set\_xlabel('Education')

ax.set\_ylabel('Age')

ax.set\_zlabel('Income')

ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))



k-means will partition your customers into mutually exclusive groups, for example, into 3 clusters. The customers in each cluster are similar to each other demographically.

Now we can create a profile for each group, considering the common characteristics of each cluster.

For example, the 3 clusters can be:

- AFFLUENT, EDUCATED AND OLD AGED

- MIDDLE AGED AND MIDDLE INCOME

- YOUNG AND LOW INCOME