K-means

is vastly used for clustering in many data science applications, it is 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.

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

1- k-Means on a randomly generated dataset

In [1]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
%matplotlib inline
```

In [2]:
```
np.random.seed(0)
```

Next we will be making random clusters of points by using the make_blobs class.

Input

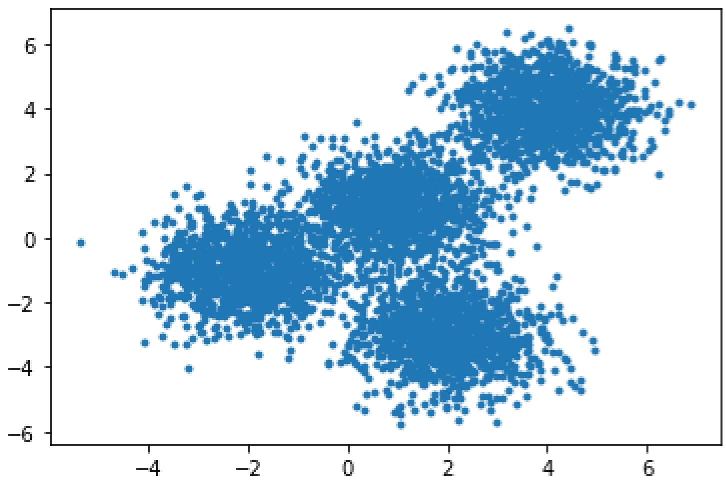
n_samples : The total number of points equally divided among clusters. Value will be: 5000 centers : The number of centers to generate, or the fixed center locations. Value will be: [[4, 4], [-2, -1], [2, -3],[1,1]] cluster_std : The standard deviation of the clusters. Value will be: 0.9

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.

In [3]:
```
X, y = make_blobs(n_samples=5000, centers=[[4,4], [-2, -1], [2, -3], [1, 1]], cluster_st
```

In [4]:
```
plt.scatter(X[:, 0], X[:, 1], marker='.')
```
Out[4]:
```
<matplotlib.collections.PathCollection at 0x2a55d91f280>
```


Setting up K-Means

KMeans Class Parameters :

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.

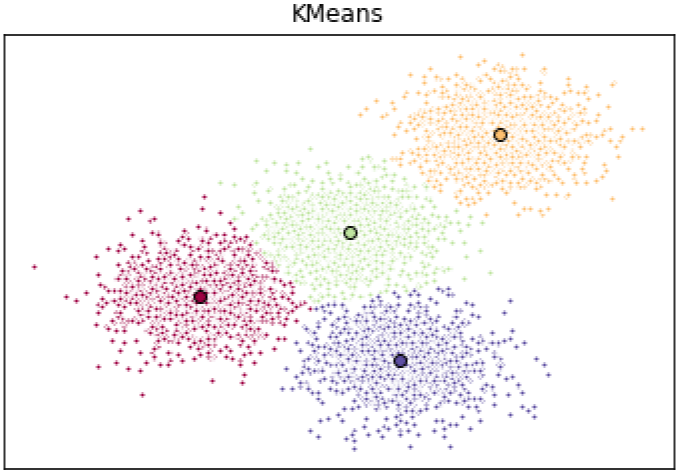
- Value will be: 12

In [5]:
```
k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
```
In [6]:
```
k_means.fit(X)
```
Out[6]:
```
KMeans(n_clusters=4, n_init=12)
```
In [7]:
```
# grab the labels for each point in the model.
k_means_labels = k_means.labels_
k_means_labels[0:10]
```
Out[7]:
```
array([0, 3, 3, 0, 1, 3, 3, 2, 1, 2])
```
In [8]:
```
# get the coordinates of the cluster centers
k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```
Out[8]:
```
array([[-2.03743147, -0.99782524],
       [ 3.97334234,  3.98758687],
       [ 0.96900523,  0.98370298],
       [ 1.99741008, -3.01666822]])
```

Creating the Visual Plot

In [9]:

```
# 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 points 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,  markeredgec

# 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()
```



Part2 Using Kmeans with Customer Segmentation

DataSet

In [10]:
```
import os
import pandas as pd
os.chdir(r"C:\Users\HP\Downloads\Cust_Segmentation")
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

Out[10]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

In [11]:
```
# Drop adress colbeacuse its  categorical variable
df = cust_df.drop('Address', axis=1)
df.head()
```

Out[11]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	6.3
1	2	47	1	26	100	4.582	8.218	0.0	12.8
2	3	33	2	10	57	6.111	5.802	1.0	20.9
3	4	29	2	4	19	0.681	0.516	0.0	6.3
4	5	47	1	31	253	9.308	8.908	0.0	7.2

Normalizing over the standard deviation

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.

In [12]:
```
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

Out[12]:
```
array([[ 0.74291541,  0.31212243, -0.37878978, ..., -0.59048916,
        -0.52379654, -0.57652509],
       [ 1.48949049, -0.76634938,  2.5737211 , ...,  1.51296181,
        -0.52379654,  0.39138677],
       [-0.25251804,  0.31212243,  0.2117124 , ...,  0.80170393,
         1.90913822,  1.59755385],
       ...,
       [-1.24795149,  2.46906604, -1.26454304, ...,  0.03863257,
         1.90913822,  3.45892281],
       [-0.37694723, -0.76634938,  0.50696349, ..., -0.70147601,
        -0.52379654, -1.08281745],
       [ 2.1116364 , -0.76634938,  1.09746566, ...,  0.16463355,
        -0.52379654, -0.2340332 ]])
```

In [13]:
```python
# Modeling
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

```
[1 2 1 1 0 2 1 2 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 2 1 2 1 1 1 1 1 1
 1 1 2 1 2 1 0 1 2 1 1 1 2 2 1 1 2 2 1 1 1 2 1 2 1 2 2 1 1 2 1 1 1 2 2 2 1
 1 1 1 1 2 1 2 2 0 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1
 1 1 1 1 1 1 2 1 2 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1
 1 1 1 1 2 1 1 2 1 2 1 1 2 0 1 2 1 1 1 1 1 1 0 2 1 1 1 1 2 1 1 2 2 1 2 1 2
 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 0 2 1 1 1 1 1 1 1 2 1 1 1 1
 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 1 2 2 1 1 1 1 1 1
 1 1 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1
 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 0 1 1 1 2 1 2 2 2 1
 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
 1 2 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 0
 1 1 1 1 1 1 2 1 1 1 0 1 1 1 1 2 1 0 1 1 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1
 1 2 1 1 1 1 2 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1
 1 2 2 1 1 1 1 1 1 1 1 1 1 1 0 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 2 1 1 0 1 0 1
 1 0 1 1 1 1 1 1 1 1 1 2 1 2 1 1 0 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 2
 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2
 2 1 1 2 1 2 1 1 2 1 2 1 1 0 1 2 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 2 2 1 1
 2 1 1 1 2 1 0 1 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1
 1 1 2 1 1 2 1 2 1 2 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1
 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 1 1 2 1 1 1 1 1 2 2
 1 1 1 1 1 1 1 2 1 1 1 1 1 1 0 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2]
```

In [14]:
```python
# We assign the labels to each row in the dataframe.
df["Clus_km"] = labels
df.head(5)
```

Out[14]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	1
1	2	47	1	26	100	4.582	8.218	0.0	12.8	2
2	3	33	2	10	57	6.111	5.802	1.0	20.9	1
3	4	29	2	4	19	0.681	0.516	0.0	6.3	1
4	5	47	1	31	253	9.308	8.908	0.0	7.2	0

In [15]:
```python
# We can easily check the centroid values by averaging the features in each cluster.
df.groupby('Clus_km').mean()
```

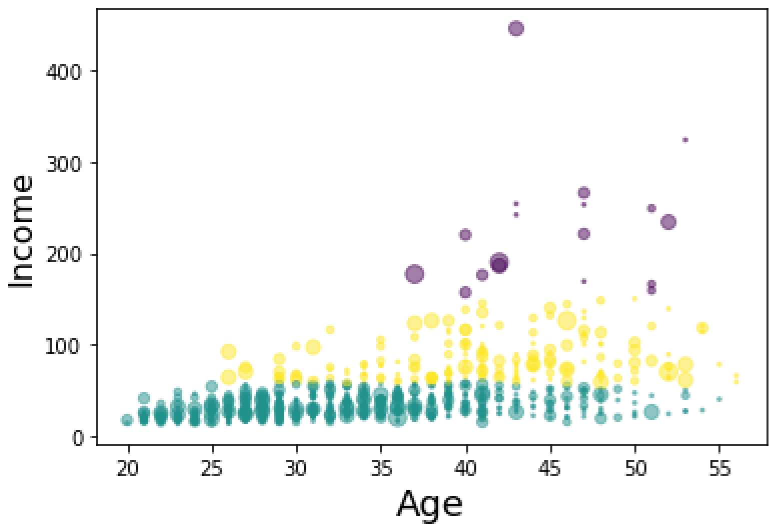
Out[15]:

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRati
Clus_km									
0	410.166667	45.388889	2.666667	19.555556	227.166667	5.678444	10.907167	0.285714	7.32222
1	432.006154	32.967692	1.613846	6.389231	31.204615	1.032711	2.108345	0.284658	10.09538
2	403.780220	41.368132	1.961538	15.252747	84.076923	3.114412	5.770352	0.172414	10.72582

In [16]:
```python
# Distribution of customers based on their age and income:
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()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_16908\1944001637.py:3: DeprecationWarning: `np.
float` is a deprecated alias for the builtin `float`. To silence this warning, use `floa
t` by itself. Doing this will not modify any behavior and is safe. If you specifically w
anted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)
```



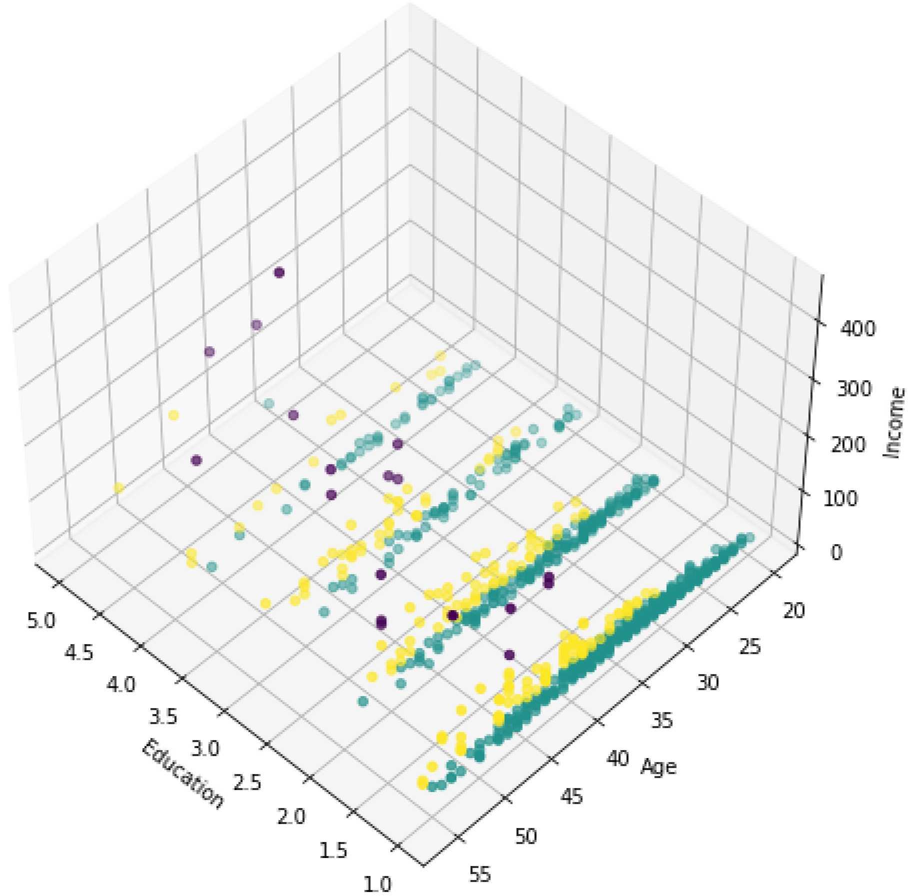
In [17]:

```
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))
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_16908\546968922.py:4: MatplotlibDeprecationWarn
ing: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass the keyword a
rgument auto_add_to_figure=False and use fig.add_axes(ax) to suppress this warning. The
default value of auto_add_to_figure will change to False in mpl3.5 and True values will
no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
C:\Users\HP\AppData\Local\Temp\ipykernel_16908\546968922.py:14: DeprecationWarning: `np.
float` is a deprecated alias for the builtin `float`. To silence this warning, use `floa
t` by itself. Doing this will not modify any behavior and is safe. If you specifically w
anted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/relea
se/1.20.0-notes.html#deprecations
  ax.scatter(X[:, 1], X[:, 0], X[:, 3], c= labels.astype(np.float))
```

Out[17]:

```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x2a55f35d610>
```



Education
Age
Income
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
20
25
30
35
40
45
50
55
0
100
200
300
400

In [ ]: