Objective:

1- Demonstration of K-Means Alghorithm.

2- Building an unsuperivsed K means clustering model to segment customers in different groups.

In [2]:
```
# Libraries
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
%matplotlib inline
```

1- Demonstration of K-Means Alghorithm.

Creating dataset

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

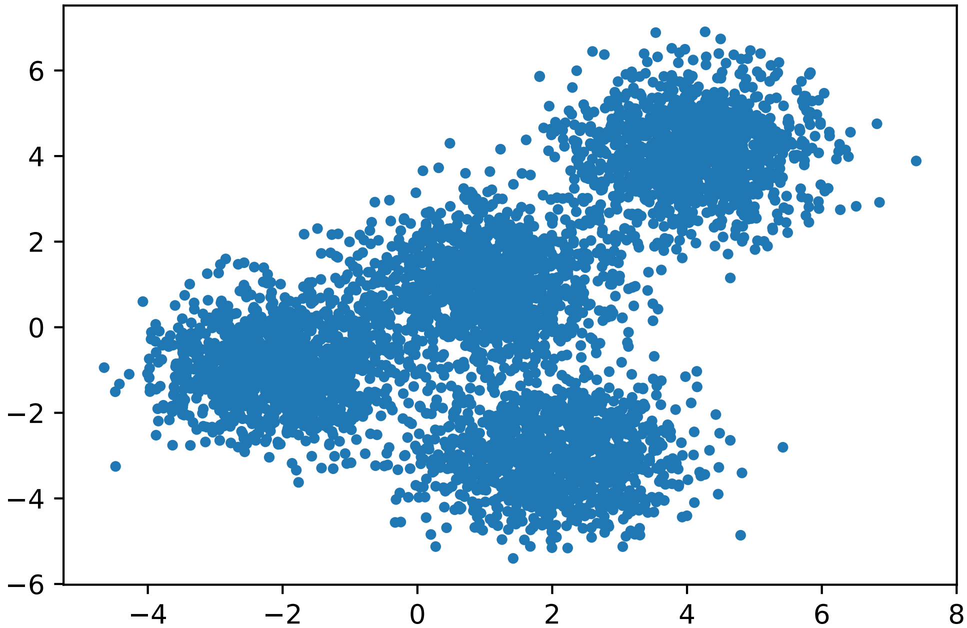
Output

- **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_s
plt.scatter(X[:, 0], X[:, 1], marker='.')
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x19c0df54e20>
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Model Setup with 4 Clusters

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**.

In [5]:
```
k_means = KMeans(init = "k-means++", n_clusters = 4, n_init = 12)
k_means.fit(X)
```

Out[5]:
```
KMeans(n_clusters=4, n_init=12)
```

In [6]:
```
# Saving labels
k_means_labels = k_means.labels_
k_means_labels
```

Out[6]:
```
array([0, 1, 2, ..., 0, 0, 0])
```

In [7]:
```
# Saving Centers
k_means_cluster_centers = k_means.cluster_centers_
k_means_cluster_centers
```

Out[7]:
```
array([[-2.02241528, -1.00115458],
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
       [ 2.01523949, -2.99062651],
       [ 1.00673121,  1.00773874]])
```

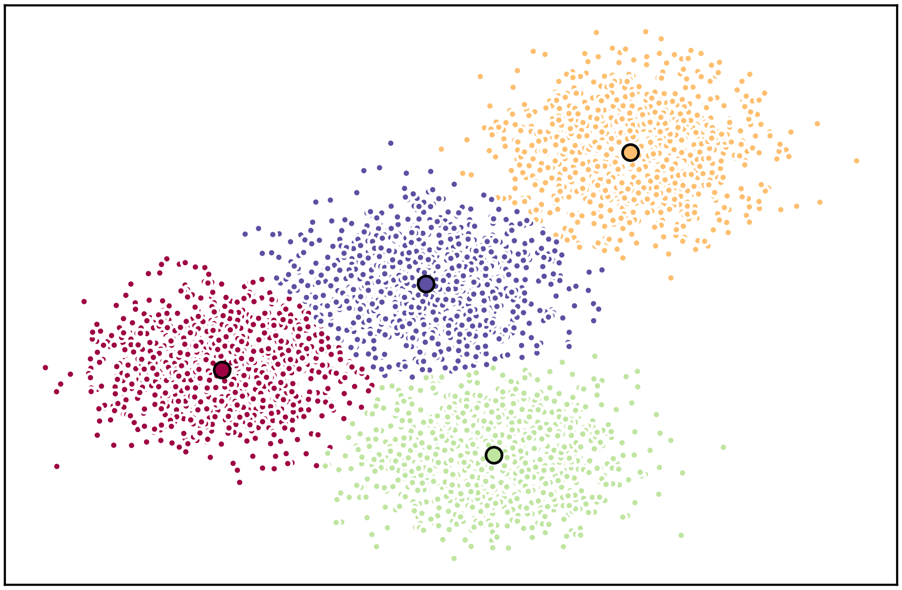
Creating the Visual Plot

In [16]:

```
# 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,  markeredge

# 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()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

KMeans



2- Building an unsuperivsed K means clustering model to segment customers.

Data Loading and Pre-processing

In [1]:
```
!wget -O Cust_Segmentation.csv https://cf-courses-data.s3.us.cloud-object-storage.appdo
```

In [67]:
```
import pandas as pd
cust_df = pd.read_csv("Cust_Segmentation.csv")
cust_df.head()
```

Out[67]:

	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 [68]:
```
# Dropping non-numerical value address
df = cust_df.drop("Address", axis=1)
df.head()
```

Out[68]:

	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

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio
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

In [79]:
```python
# Getting the X dataset
X = df.values[:,1:] # Slicing the np array to remove id
X = np.nan_to_num(X)
X
```

Out[79]:
```
array([[41. ,  2. ,  6. , ...,  0. ,  6.3,  1. ],
       [47. ,  1. , 26. , ...,  0. , 12.8,  2. ],
       [33. ,  2. , 10. , ...,  1. , 20.9,  1. ],
       ...,
       [25. ,  4. ,  0. , ...,  1. , 33.4,  1. ],
       [32. ,  1. , 12. , ...,  0. ,  2.9,  1. ],
       [52. ,  1. , 16. , ...,  0. ,  8.6,  2. ]])
```

Modeling

In [80]:
```python
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

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

Insights

In [81]:
```python
df["Clus_km"] = labels
df.head(5)
```

Out[81]:

	Customer	Age	Edu	Years	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km

	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	0
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	0
3	4	29	2	4	19	0.681	0.516	0.0	6.3	0
4	5	47	1	31	253	9.308	8.908	0.0	7.2	1

In [82]:

```
# Checking the centroid values by averaging the features in each cluster.
df.groupby('Clus_km').mean()
```

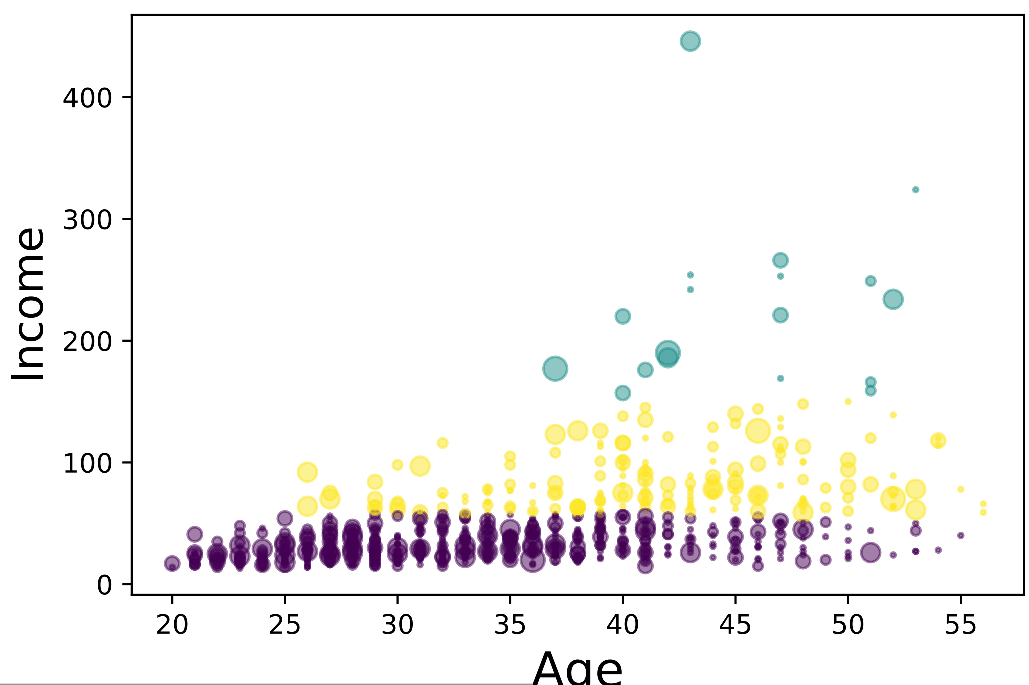
Out[82]:

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

Visualization using age and income features

In [83]:

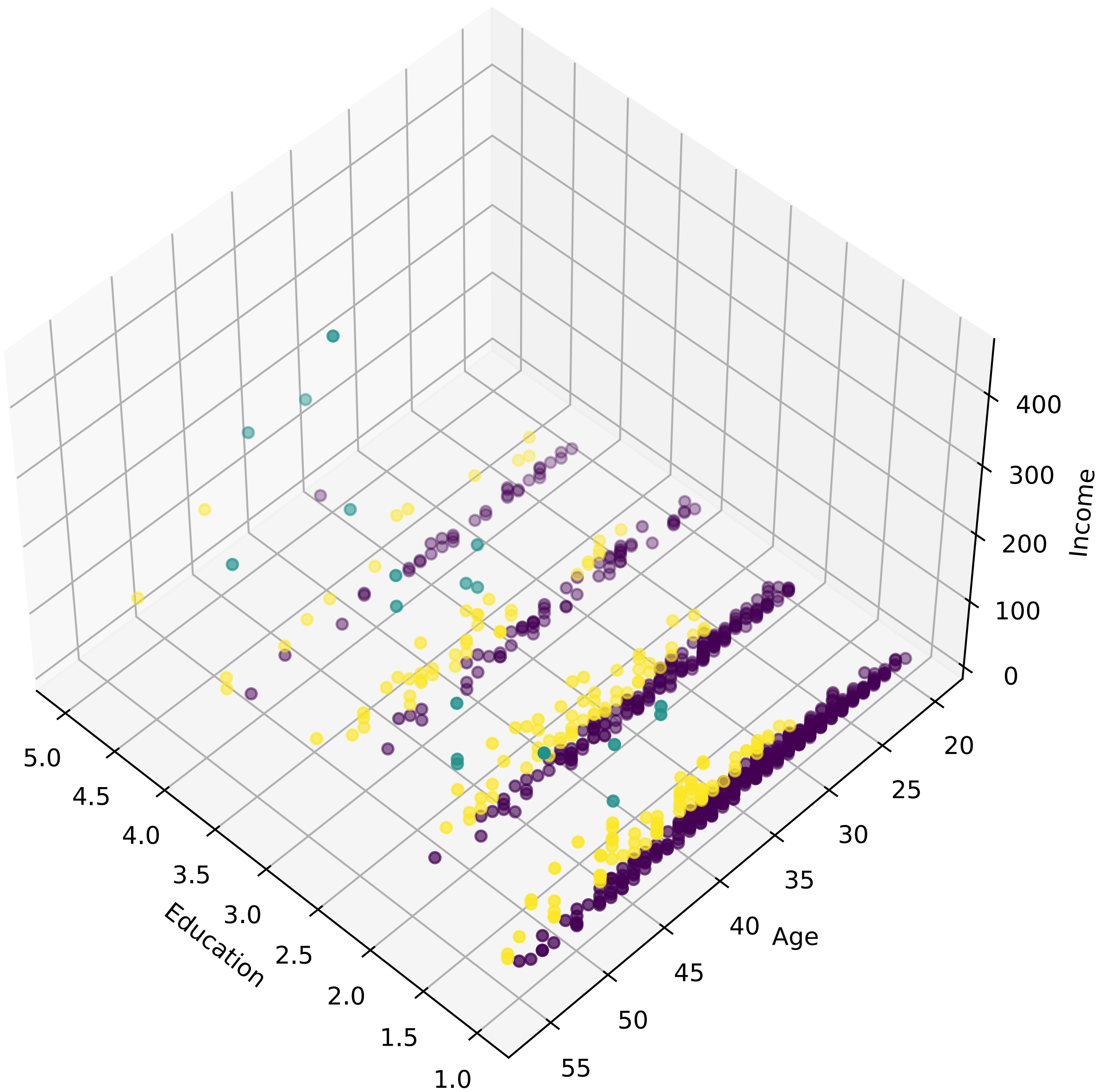
```
area = np.pi * ( X[:, 1])**2
plt.scatter(X[:, 0], X[:, 3], s=area, c=labels.astype(np.float), alpha=0.5)
plt.xlabel('Age', fontsize=18)
plt.ylabel('Income', fontsize=16)

plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [84]:
```
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))
```

Out[84]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x19c1278b4f0>



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js