UnSupervised Learning Algorithm - Customer Segmentation with K-Means

K-means is one of the most basic clustering algorithms. It relies on finding cluster centers to group data points based on minimizing the sum of squared errors between each datapoint and its cluster center.Partioning Customers with similar characteristics into differnt groups.

In [35]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

In [36]:
```
data = pd.read_csv("data/Cust_Segmentation.csv")
```

In [37]:
```
print(data.shape)
```

```
(850, 10)
```

In [38]:
```
#Print no of integers, floats and strings
data.dtypes.value_counts()
```

Out[38]:
```
int64      5
float64    4
object     1
dtype: int64
```

In [39]:
```
data.head()
```

Out[39]:

	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

Preprocessing Steps

1. Select Features .

In [40]:
```
#Address is a categorical Variable and We can drop it
df = data.drop('Address', axis=1)
df.head()
```

Out[40]:

	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

2. Normalize Features .

In [41]:
```
#Normalization is used to interpret features with differnt magnitude and di
from sklearn.preprocessing import StandardScaler
X = df.values[:,1:]
X = np.nan_to_num(X)
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

Out[41]:
```
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 ]])
```

Modeling with K-means

In [42]:
```python
from sklearn.cluster import KMeans
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

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

In [43]:
```
#Assigning labels generated by K-means to our original dataset
df["Clus_km"] = labels
df.head(5)
```

Out[43]:

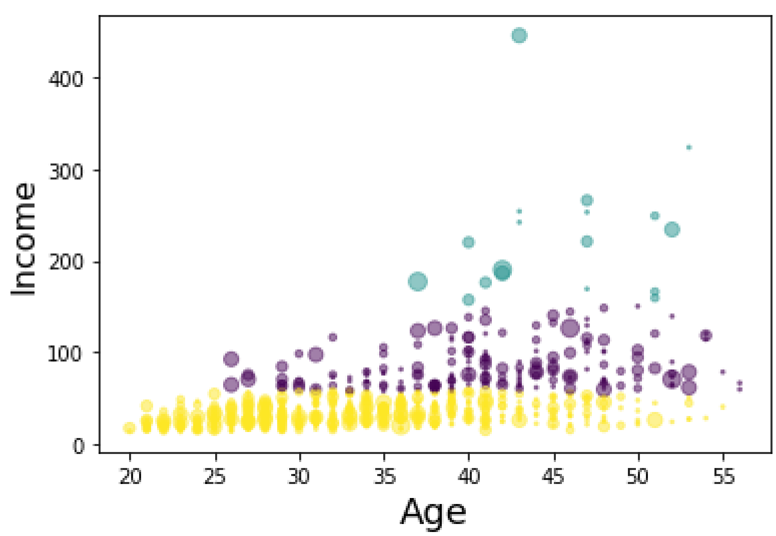
	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	DebtIncomeRatio	Clus_km
0	1	41	2	6	19	0.124	1.073	0.0	6.3	2
1	2	47	1	26	100	4.582	8.218	0.0	12.8	0
2	3	33	2	10	57	6.111	5.802	1.0	20.9	2
3	4	29	2	4	19	0.681	0.516	0.0	6.3	2
4	5	47	1	31	253	9.308	8.908	0.0	7.2	1

In [44]:
```
df.groupby('Clus_km').mean()
```

Out[44]:

Clus_km	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted
0	402.295082	41.333333	1.956284	15.256831	83.928962	3.103639	5.765279	0.171233
1	410.166667	45.388889	2.666667	19.555556	227.166667	5.678444	10.907167	0.285714
2	432.468413	32.964561	1.614792	6.374422	31.164869	1.032541	2.104133	0.285185

In [45]:
```
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()
```



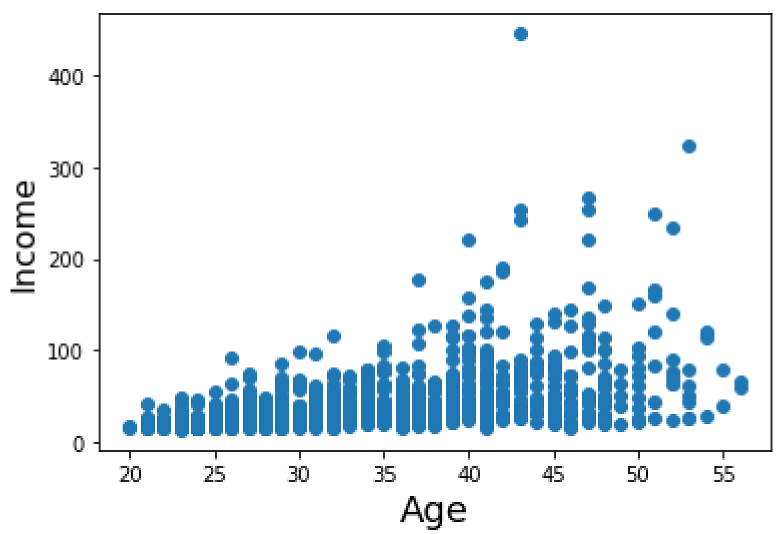
In [46]:
```
#Just to evaluate elbow method Extracting only two features Age and Income.
X_new = df[['Age','Income']]
X_new[1:10]
```

Out[46]:

	Age	Income
1	47	100
2	33	57
3	29	19
4	47	253
5	40	81
6	38	56
7	42	64
8	26	18
9	47	115

In [47]:
```
plt.scatter(X_new['Age'],X_new['Income'])
plt.xlabel('Age', fontsize=18)
plt.ylabel('Income', fontsize=16)
```

Out[47]:
```
Text(0, 0.5, 'Income')
```



In [48]:
```
Clus_dataSet = StandardScaler().fit_transform(X_new)
Clus_dataSet
```

Out[48]:
```
array([[ 0.74291541, -0.71845859],
       [ 1.48949049,  1.38432469],
       [-0.25251804,  0.26803233],
       ...,
       [-1.24795149, -0.74441888],
       [-0.37694723, -0.484816  ],
       [ 2.1116364 ,  0.44975434]])
```

In [49]:
```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(X_new)
labels = k_means.labels_
print(labels)
```

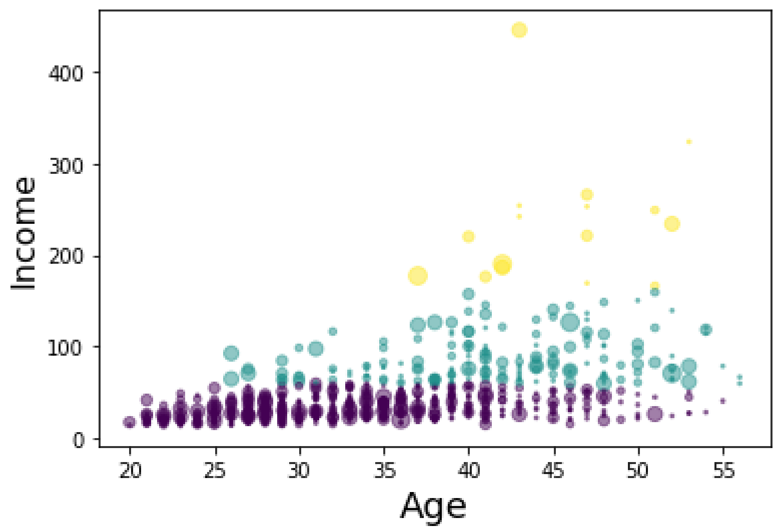
```
[0 1 0 0 2 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 1 0 0 0 0 0
0
 0 0 1 0 1 0 2 0 1 0 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 1 1 0 0 1 0 0 0 1 1 1
0
 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1
0
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0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

In [50]:
```
#Assigning labels generated by K-means to our original dataset
X_new["Clus_km"] = labels
X_new.head(5)
```

Out[50]:

	Age	Income	Clus_km
0	41	19	0
1	47	100	1
2	33	57	0
3	29	19	0
4	47	253	2

In [51]:
```
#area = np.pi * ( X_new[:, 1])**2
plt.scatter(X_new['Age'], X_new['Income'], s=area, c=labels.astype(np.float
plt.xlabel('Age', fontsize=18)
plt.ylabel('Income', fontsize=16)

plt.show()
```



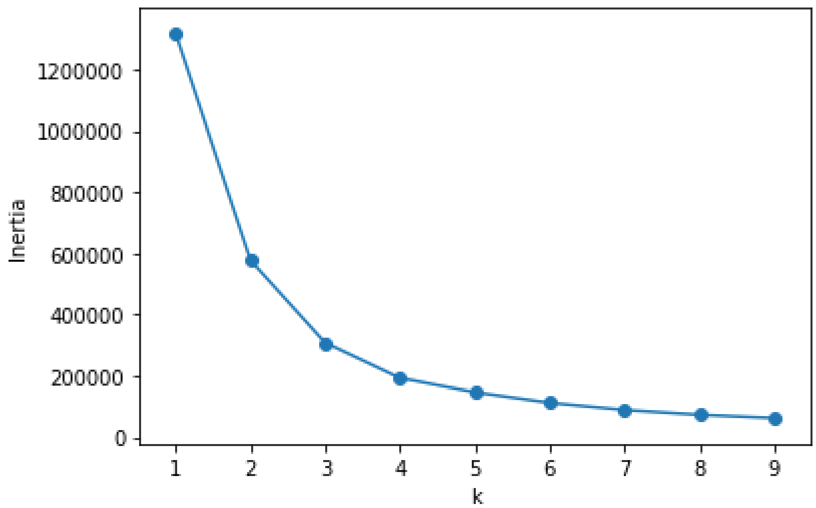
How to use a inertia Curve to determine optimal number of Clusters?

In [52]:
```

k_range = range(1,10)
inertia = []
for k in k_range:
    k_means = KMeans(init = "k-means++", n_clusters = k, n_init = 12)
    k_means.fit(X_new)
    inertia.append(k_means.inertia_)

```

In [53]:
```
print(inertia)
```

```
[1316336.7635294115, 576898.8796737788, 307362.0293573582, 192707.0114372
141, 145956.98191263332, 111875.90413485553, 88757.602416176, 73150.40656
242585, 62339.92751946109]
```

In [54]:
```
plt.plot(k_range,inertia)
plt.scatter(k_range,inertia)
plt.xlabel('k')
plt.ylabel('Inertia');
```



Summary

K-means can classify customers to mutually exclusive groups .

From distribution of customers based on income and age we can group them into three categories. Elbow Curve helps us to determine the number of Clusters required . Inertia continues to go down as number of clusters increases but after sometime number it flattens down.

In [ ]: