## K-Means Clustering

## Input Data - Customer Dataset

In [1]:
```
# Import the standard modules to be used in this lab
import pandas as pd
import numpy as np
%matplotlib inline
```

In [2]:
```
# Load the dataset into IPython and rename it into a dataframe
cust_pd = pd.read_csv("input/customers.csv")
```

In [3]:
```
# First five data available in the dataset
cust_pd.head()
```

Out[3]:

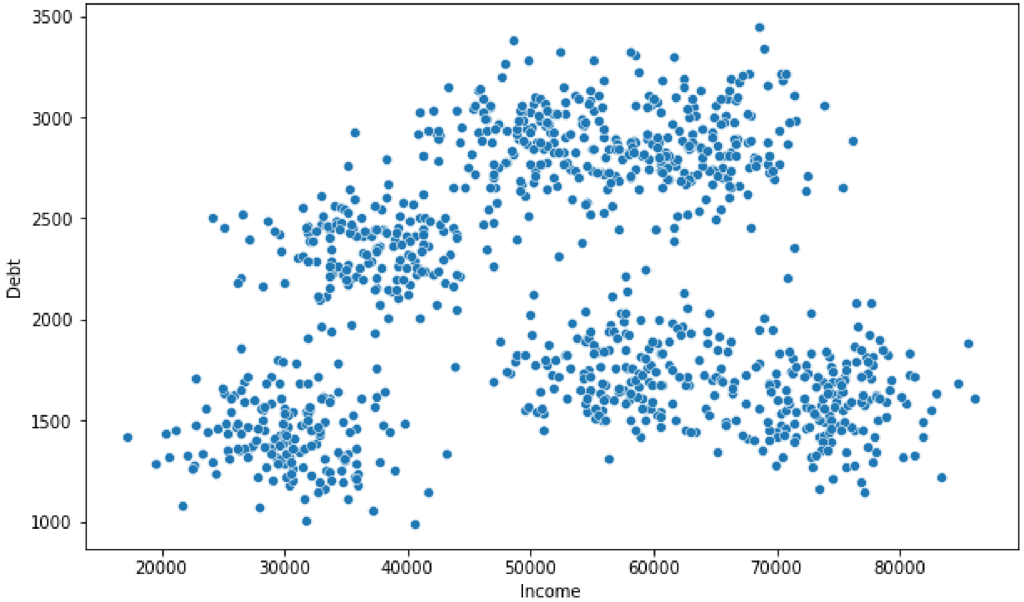
| | Income | Debt |
|---|---|---|
| **0** | 30309 | 1405 |
| **1** | 38969 | 1256 |
| **2** | 27268 | 1370 |
| **3** | 27970 | 1074 |
| **4** | 56432 | 1976 |

In [4]:
```
# The size of the dataset
cust_pd.shape
```

Out[4]:
```
(1000, 2)
```

In [5]:
```
# Let's plot the data to see if we can see any cluster using a scatter plot of Seaborn library
from matplotlib import pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,6))
sns.scatterplot(x=cust_pd["Income"], y=cust_pd["Debt"])
```

Out[5]:
```
<AxesSubplot:xlabel='Income', ylabel='Debt'>
```




As we can see, attributes Income and Debt are in very different ranges whereby Income is about 10-20 times larger than Debt. Therefore, before we perform the clustering, we normalize the data to [0-1]. Here, we normalize manually by using min() and max() functions. We can also use MinMaxScaler to normalize data.

### Normalization of Data

In [6]:
```
# Normalize the data [0-1] using min() and max() function
cust_pd["Income"] = (cust_pd["Income"] - cust_pd["Income"].min()) /(cust_pd["Income"].max() - cust_pd["Income"].min())
cust_pd["Debt"] = (cust_pd["Debt"] - cust_pd["Debt"].min()) / (cust_pd["Debt"].max() - cust_pd["Debt"].min())
cust_pd.head()
```

Out[6]:

| | Income | Debt |
|---|---|---|
| **0** | 0.189918 | 0.169857 |
| **1** | 0.315582 | 0.109165 |
| **2** | 0.145790 | 0.155601 |
| **3** | 0.155977 | 0.035031 |
| **4** | 0.568985 | 0.402444 |

In [7]:
```
# Normalize the data [0-1] using MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
cust_pd = scaler.fit_transform(cust_pd)
cust_pd = pd.DataFrame(cust_pd, columns=["Income", "Debt"])
cust_pd.head()
```

Out[7]:

| | Income | Debt |
|---|---|---|
| **0** | 0.189918 | 0.169857 |
| **1** | 0.315582 | 0.109165 |
| **2** | 0.145790 | 0.155601 |
| **3** | 0.155977 | 0.035031 |
| **4** | 0.568985 | 0.402444 |

### Clustering using K-Means

Now lets import K-means from Scikit-Learn to create an object of K-means. We will set the number of clusters, K=4. Assuming there are 4 clusters.

In [8]:
```
# To do a k-means clustering, import K-means from Scikit-Learn
from sklearn.cluster import KMeans
```

First, we initialize the centroids. Then, we create a KMeans object and set the following arguments. n_clusters is used to specify the number of clusters. init is used to specify the initial centroid. n_init is set to 1 indicating that the K-means algorithm will be run one time.

In [9]:
```
# Set the number of clusters, K = 4. Assuming there are 4 clusters.
# Initialize the centroids
init_centroids = np.array([[0.01,0.02],[0.01,0.015],[0.01,0.022],[0.01,0.023]])
# Define K-means
kmeans = KMeans(n_clusters=4, init=init_centroids, n_init=1)
kmeans.fit(cust_pd)
```

Out[9]:
```
KMeans(init=array([[0.01 , 0.02 ],
       [0.01 , 0.015],
       [0.01 , 0.022],
       [0.01 , 0.023]]),
       n_clusters=4, n_init=1)
```

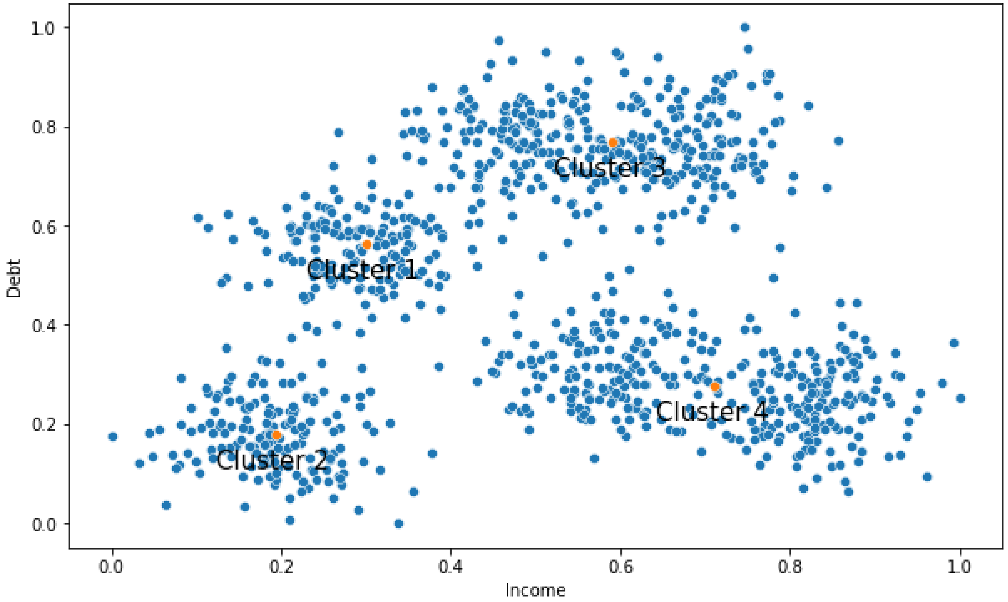
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [10]:
```
# Generate the centroid values for the final clusters
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[0.29983346 0.56387364]
 [0.19310559 0.17907332]
 [0.59123375 0.76930336]
 [0.71086573 0.27637849]]
```

In [11]:
```
# Display the clusters using the scatter plot
offset = 0.07
fig, ax = plt.subplots(figsize = (10,6))
sns.scatterplot(x=cust_pd["Income"], y=cust_pd["Debt"])
sns.scatterplot(x=centroids[:,0], y=centroids[:,1])
ax.annotate('Cluster 1', xy=(centroids[0,0]-offset,centroids[0,1]-offset),size=15)
ax.annotate('Cluster 2', xy=(centroids[1,0]-offset,centroids[1,1]-offset),size=15)
ax.annotate('Cluster 3', xy=(centroids[2,0]-offset,centroids[2,1]-offset),size=15)
ax.annotate('Cluster 4', xy=(centroids[3,0]-offset,centroids[3,1]-offset),size=15)
```

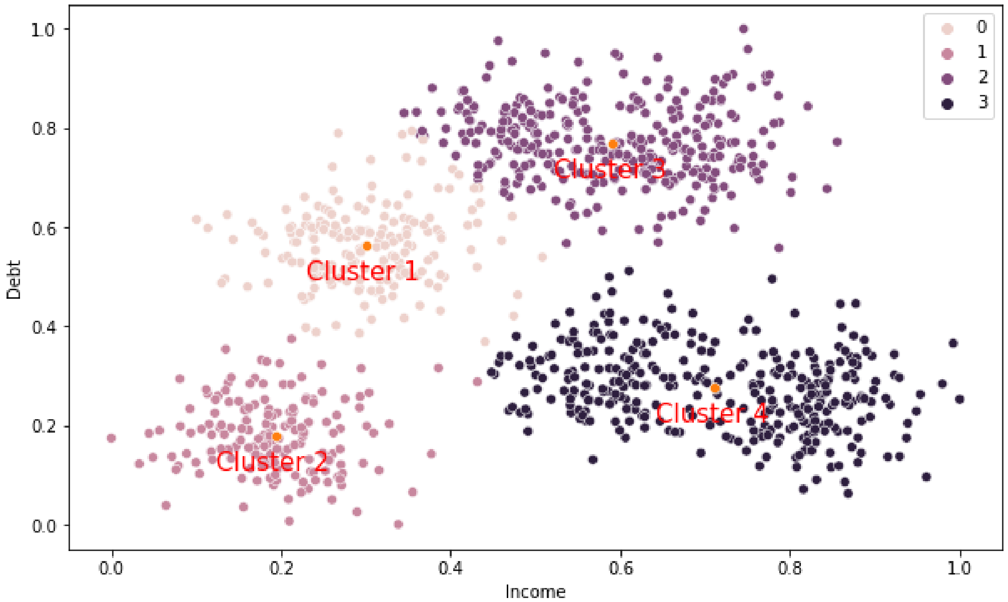
Out[11]:
```
Text(0.6408657344950788, 0.2063784938712781, 'Cluster 4')
```




The plot shows 4 clusters available for this dataset. The centroids are marked with yellow ‘o’ which represent 4 centroids for each cluster. We put some annotation to indicate the cluster numbers.

In [12]:
```
# Perform a prediction to classify the data points using the K-means
offset = 0.07
pred = kmeans.predict(cust_pd)
fig, ax = plt.subplots(figsize = (10,6))
sns.scatterplot(x=cust_pd["Income"], y=cust_pd["Debt"], hue=pred)
sns.scatterplot(x=centroids[:,0], y=centroids[:,1])
ax.annotate('Cluster 1', xy=(centroids[0,0]-offset,centroids[0,1]-offset),size=15, color = 'red')
ax.annotate('Cluster 2', xy=(centroids[1,0]-offset,centroids[1,1]-offset),size=15, color = 'red')
ax.annotate('Cluster 3', xy=(centroids[2,0]-offset,centroids[2,1]-offset),size=15, color = 'red')
ax.annotate('Cluster 4', xy=(centroids[3,0]-offset,centroids[3,1]-offset),size=15, color = 'red')
```

Out[12]:
```
Text(0.6408657344950788, 0.2063784938712781, 'Cluster 4')
```




In [13]:
```
# Specify the initial centroid using init='k-means++'
kmeans = KMeans(n_clusters=4, init='k-means++')
kmeans.fit(cust_pd)
```

Out[13]:
```
KMeans(n_clusters=4)
```

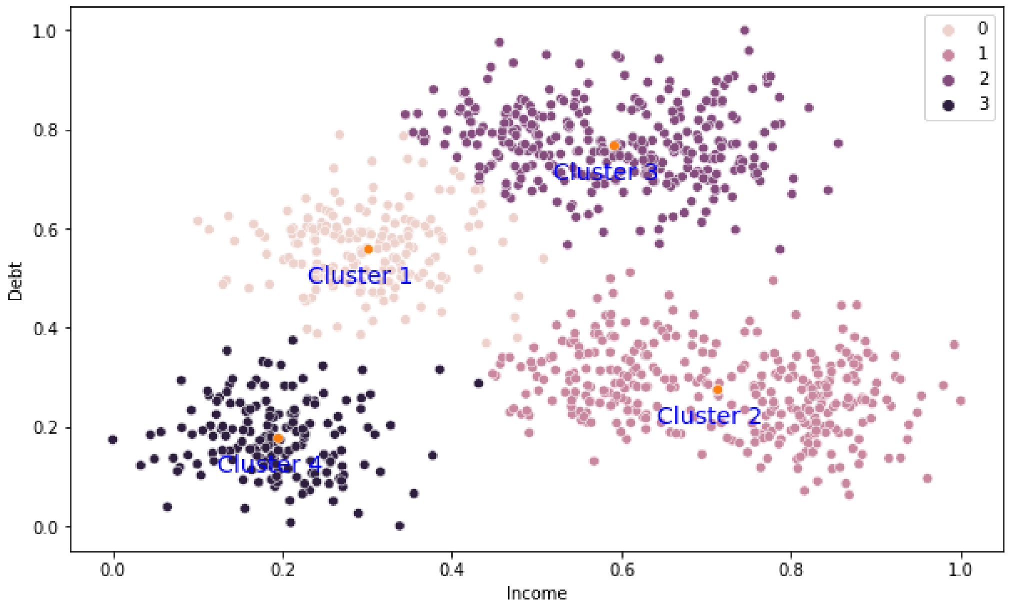
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [14]:
```
# Generate centroid values for final clusters
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[0.30013745 0.56044873]
 [0.71158411 0.27606079]
 [0.5898113  0.76940107]
 [0.19310559 0.17907332]]
```

In [15]:
```
# Display the clusters based on k-means++ initialization method
offset = 0.07
pred = kmeans.predict(cust_pd)
fig, ax = plt.subplots(figsize = (10,6))
sns.scatterplot(x=cust_pd["Income"], y=cust_pd["Debt"], hue=pred)
sns.scatterplot(x=centroids[:,0], y=centroids[:,1])
ax.annotate('Cluster 1', xy=(centroids[0,0]-offset,centroids[0,1]-offset),color='blue', size=14)
ax.annotate('Cluster 2', xy=(centroids[1,0]-offset,centroids[1,1]-offset),color='blue', size=14)
ax.annotate('Cluster 3', xy=(centroids[2,0]-offset,centroids[2,1]-offset),color='blue', size=14)
ax.annotate('Cluster 4', xy=(centroids[3,0]-offset,centroids[3,1]-offset),color='blue', size=14)
```

Out[15]:
```
Text(0.12310558916081743, 0.10907331975560075, 'Cluster 4')
```



In [16]:
```
# Let's experiment with K=6
kmeans = KMeans(n_clusters=6) # by default: init='k-means++' and n_init=10
kmeans.fit(cust_pd)
```

Out[16]:
```
KMeans(n_clusters=6)
```

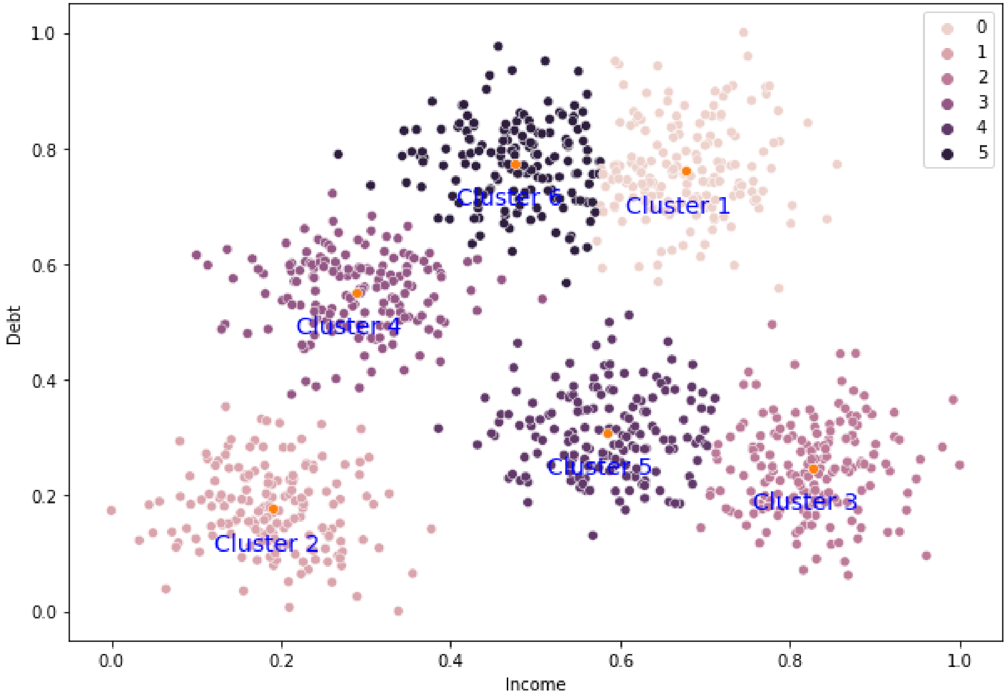
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [17]:
```
# Centroids values for the final cluster
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[0.67658314 0.76036913]
 [0.1903772  0.17640437]
 [0.82645433 0.24756813]
 [0.28823029 0.55107354]
 [0.58348958 0.30898005]
 [0.47679365 0.77345764]]
```

In [18]:
```
# Display the clusters for customer dataset
offset = 0.07
pred = kmeans.predict(cust_pd)
fig, ax = plt.subplots(figsize = (10,7))
sns.scatterplot(x=cust_pd["Income"], y=cust_pd["Debt"], hue=pred)
sns.scatterplot(x=centroids[:,0], y=centroids[:,1])
ax.annotate('Cluster 1', xy=(centroids[0,0]-offset,centroids[0,1]-offset),color='blue', size=14)
ax.annotate('Cluster 2', xy=(centroids[1,0]-offset,centroids[1,1]-offset),color='blue', size=14)
ax.annotate('Cluster 3', xy=(centroids[2,0]-offset,centroids[2,1]-offset),color='blue', size=14)
ax.annotate('Cluster 4', xy=(centroids[3,0]-offset,centroids[3,1]-offset),color='blue', size=14)
ax.annotate('Cluster 5', xy=(centroids[4,0]-offset,centroids[4,1]-offset),color='blue', size=14)
ax.annotate('Cluster 6', xy=(centroids[5,0]-offset,centroids[5,1]-offset),color='blue', size=14)
```

Out[18]:
```
Text(0.40679365243600946, 0.7034576412695601, 'Cluster 6')
```



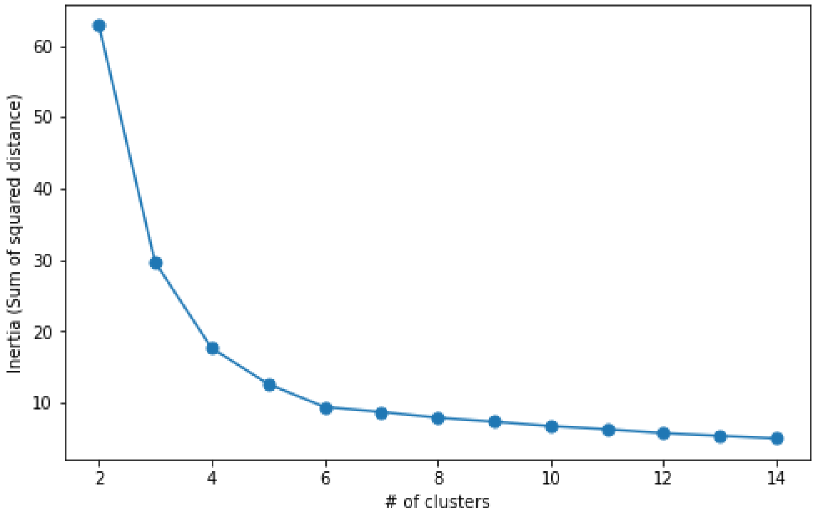

We can categorize the customer dataset into six cluster based on the Income and Debt. The yellow dot represent the centroids for each clusters. As an example, Cluster 2 belongs to the customer with high income and low debt.

### Optimal Number of Clusters (Elbow Method)

In [19]:
```
# Determine optimal value for number of clusters, use summation of squared,distance (inertia) of different number of clusters
inrt = []
m = 15
for c in range(2,m):
    km = KMeans(n_clusters=c)
    km.fit(cust_pd.iloc[:,0:2])
    inrt.append(km.inertia_)

plt.figure(figsize=(8,5))
plt.plot([c for c in range(2,m)], inrt, marker='.', markersize=14)
plt.xlabel("# of clusters")
plt.ylabel("Inertia (Sum of squared distance)")
```

Out[19]:
```
Text(0, 0.5, 'Inertia (Sum of squared distance)')
```




The value of inertia decreases as the number of clusters increases. We can see that after 6 clusters the inertia is not reducing significantly. We can conclude that 6 is the optimal number of clusters.

In [ ]:
```

```