ASSIGNMENT 6

- Name : Achal Rajesh Mate
- Roll No : 2203541
- Enroll No : MITU20BTCSD001
- Branch : CSE
- Class : TY CSE Is - 3
- Guided By : Prof Nagesh Jadhav Sir

Title:- Implement K means algorithm on dataset

Objectives:-
1. To learn unsupervised learning
2. To implement K means algorithm

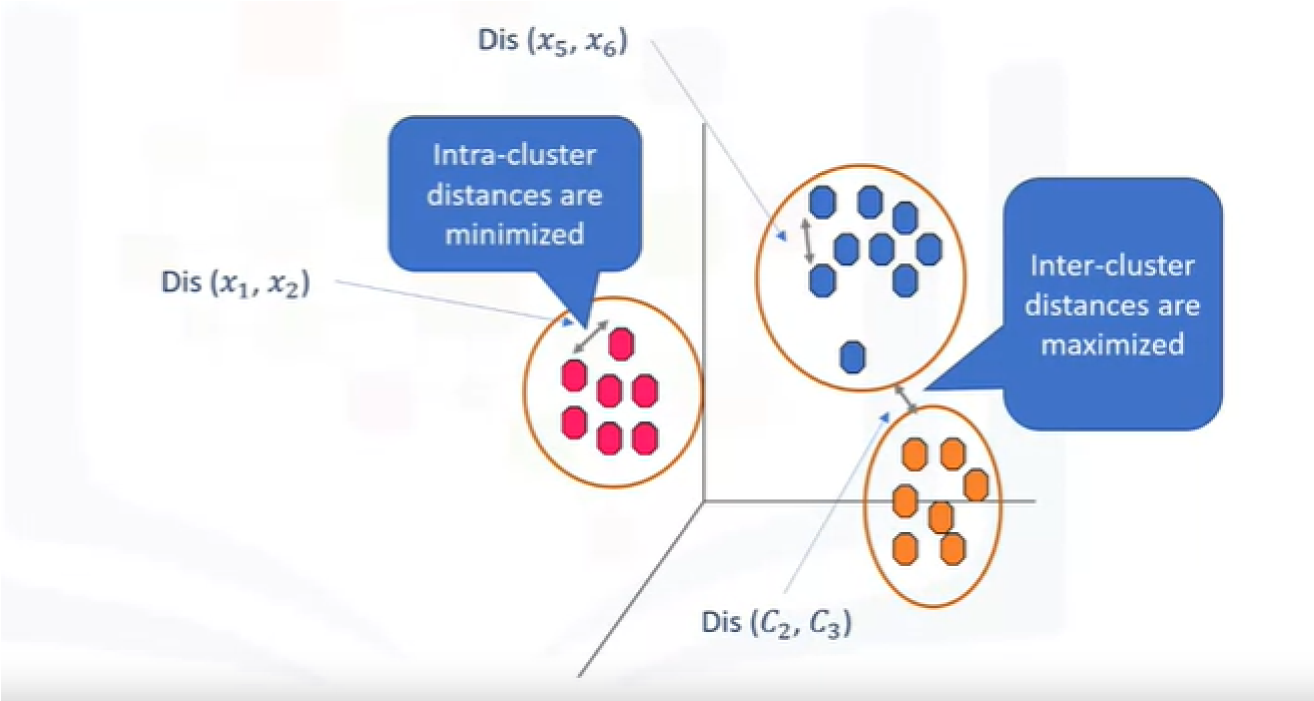
Theory:

K-Means Algorithm?

- K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
- It allows us to cluster the data into different groups
- It is a centroid-based algorithm, where each cluster is associated with a centroid.
- The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

$$\text{Dis}\,(x_1, x_2) = \sqrt{\sum_{i=0}^{n}(x_{1i} - x_{2i})^2}$$

- Distance of samples from each other is used to shape the cluster

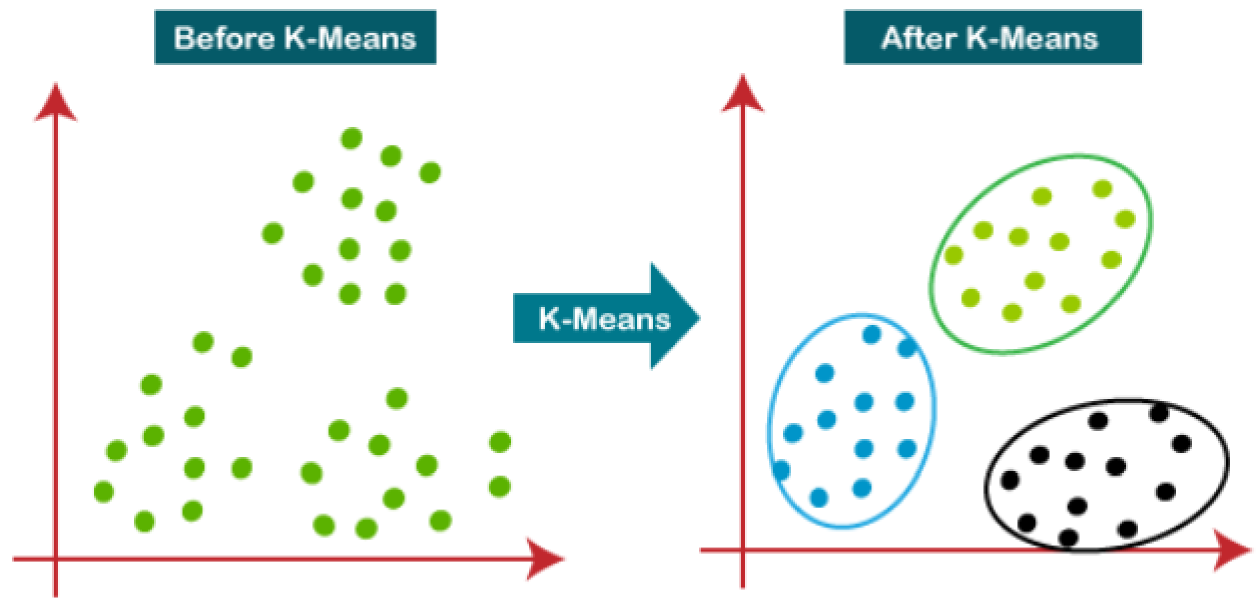


- The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.
- Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



Steps to implement K-Means Algorithm

- Step-1: Select the number K to decide the number of clusters.
- Step-2: Select random K points or centroids. (It can be other from the input dataset).
- Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.
- Step-4: Calculate the variance and place a new centroid of each cluster.
- Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
- Step-7: The model is ready.

RULE FOR CHOOSING VALUE OF K

- Never go with k=2 as the option because it means you divide the complete data into two halfs and it's not useful for any business case.
- If you have option to choose between two values of K, always go with a lesser value.
- Since we will be taking business decisions based on the cluster result, it's awlays a good idea to go with a lower value of K so that it's easy to take and implement business decisions.
- Silhouette: That value of k for which the score is maximum

- Elbow, you look at the elbow of the curve

Problem Statement : Customer Segmentation

Prolem Context :

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 retain those customers. Another group might include customers from non-profit organizations and so on.

Dataset Link :

https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-ML0101EN-SkillsNetwork/labs/Module%204/data/Cust_Segmentation.csv

Dataset Information

This Dataset Contains 10 features with 850 instances Attributes are

1. Customer Id,
2. Age
3. Edu
4. Years Employed
5. Income
6. Card Debt
7. Other Debt
8. Defaulted
9. Address 10.DebtIncomeRatio

Import All Necessary Files

In [1]:

```
import pandas as pd
import numpy as np
```

```python
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

In [31]:
```python
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder,StandardScaler
from scipy import stats
from sklearn.metrics import silhouette_score,accuracy_score,confusion_matrix,classification_report
from sklearn import metrics
```

Read the Dataset

In [3]:
```python
df = pd.read_csv("Cust_Segmentation.csv")
df.head()
```

Out[3]:

	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

View Dimension of dataste

In [4]:
```python
df.shape
```

Out[4]:
```
(850, 10)
```

This dataset contains 850 instance or rows and 10 columns

Columns in dataset

In [5]:
```
df.columns
```

Out[5]:
```
Index(['Customer Id', 'Age', 'Edu', 'Years Employed', 'Income', 'Card Debt',
       'Other Debt', 'Defaulted', 'Address', 'DebtIncomeRatio'],
      dtype='object')
```

Concise Summary

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 850 entries, 0 to 849
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Customer Id      850 non-null    int64  
 1   Age              850 non-null    int64  
 2   Edu              850 non-null    int64  
 3   Years Employed   850 non-null    int64  
 4   Income           850 non-null    int64  
 5   Card Debt        850 non-null    float64
 6   Other Debt       850 non-null    float64
 7   Defaulted        700 non-null    float64
 8   Address          850 non-null    object 
 9   DebtIncomeRatio  850 non-null    float64
dtypes: float64(4), int64(5), object(1)
memory usage: 66.5+ KB
```

form above information we can say that Address feature is in object datatype and defaulted feature contains null values ,Lets check them

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Customer Id         0
Age                 0
Edu                 0
Years Employed      0
Income              0
Card Debt           0
Other Debt          0
```

```
Defaulted          150
Address              0
DebtIncomeRatio      0
dtype: int64
```

defaulted feature contains 150 NUll values

Drop the Column That Contains Null Values and Catergorical Values

The k-means algorithm isn't directly applicable to categorical variables because the Euclidean distance function isn't really meaningful for discrete variables.

In [8]:

```
df.Address.value_counts()
```

Out[8]:

```
NBA001    71
NBA002    71
NBA000    60
NBA004    58
NBA003    55
NBA006    50
NBA008    49
NBA009    45
NBA005    43
NBA007    41
NBA010    37
NBA011    36
NBA012    28
NBA014    24
NBA016    22
NBA013    22
NBA017    20
NBA015    18
NBA019    16
NBA018    14
NBA023    11
NBA021    10
NBA026    10
NBA022     9
NBA025     9
NBA020     8
NBA024     4
NBA027     4
NBA031     2
NBA034     1
NBA029     1
```

```
NBA030     1
Name: Address, dtype: int64
```

In [9]:

```python
df.drop(['Customer Id','Defaulted','Address'],axis=1,inplace=True)
```

In [10]:

```python
df.columns
```

Out[10]:

```
Index(['Age', 'Edu', 'Years Employed', 'Income', 'Card Debt', 'Other Debt',
       'DebtIncomeRatio'],
      dtype='object')
```

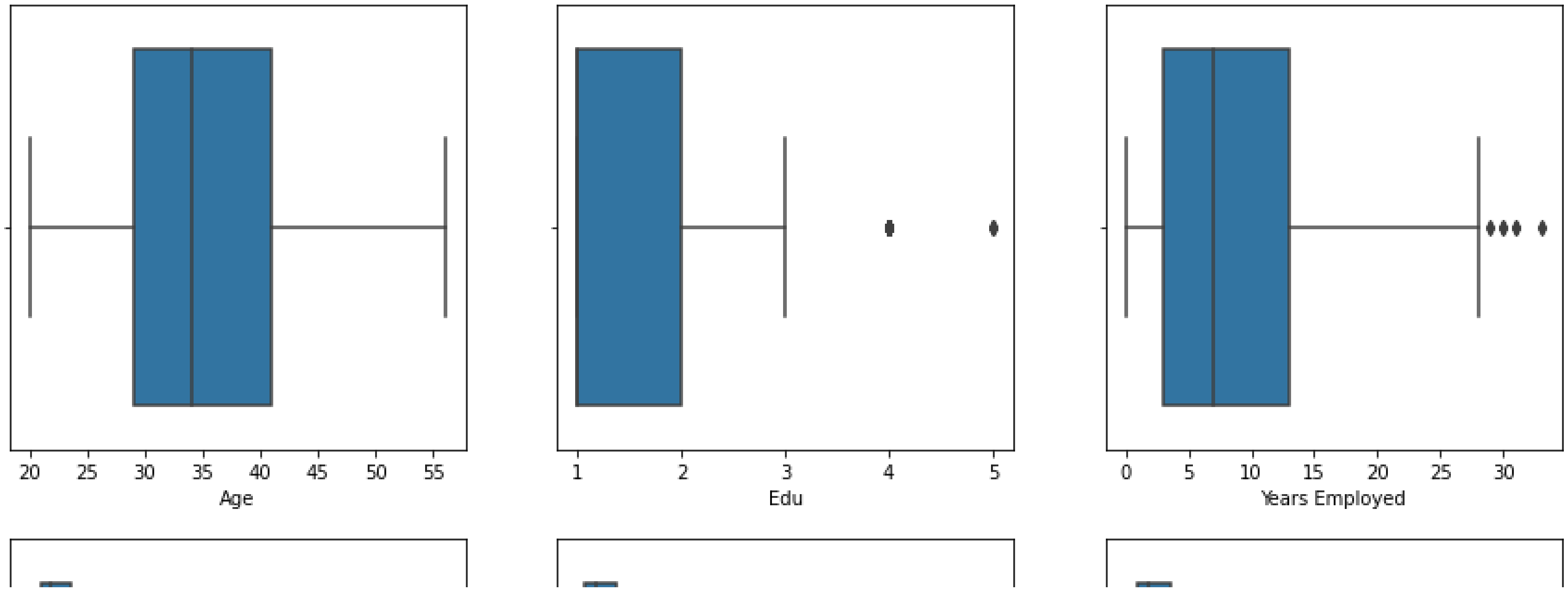
Check the outlier by ploting box plot

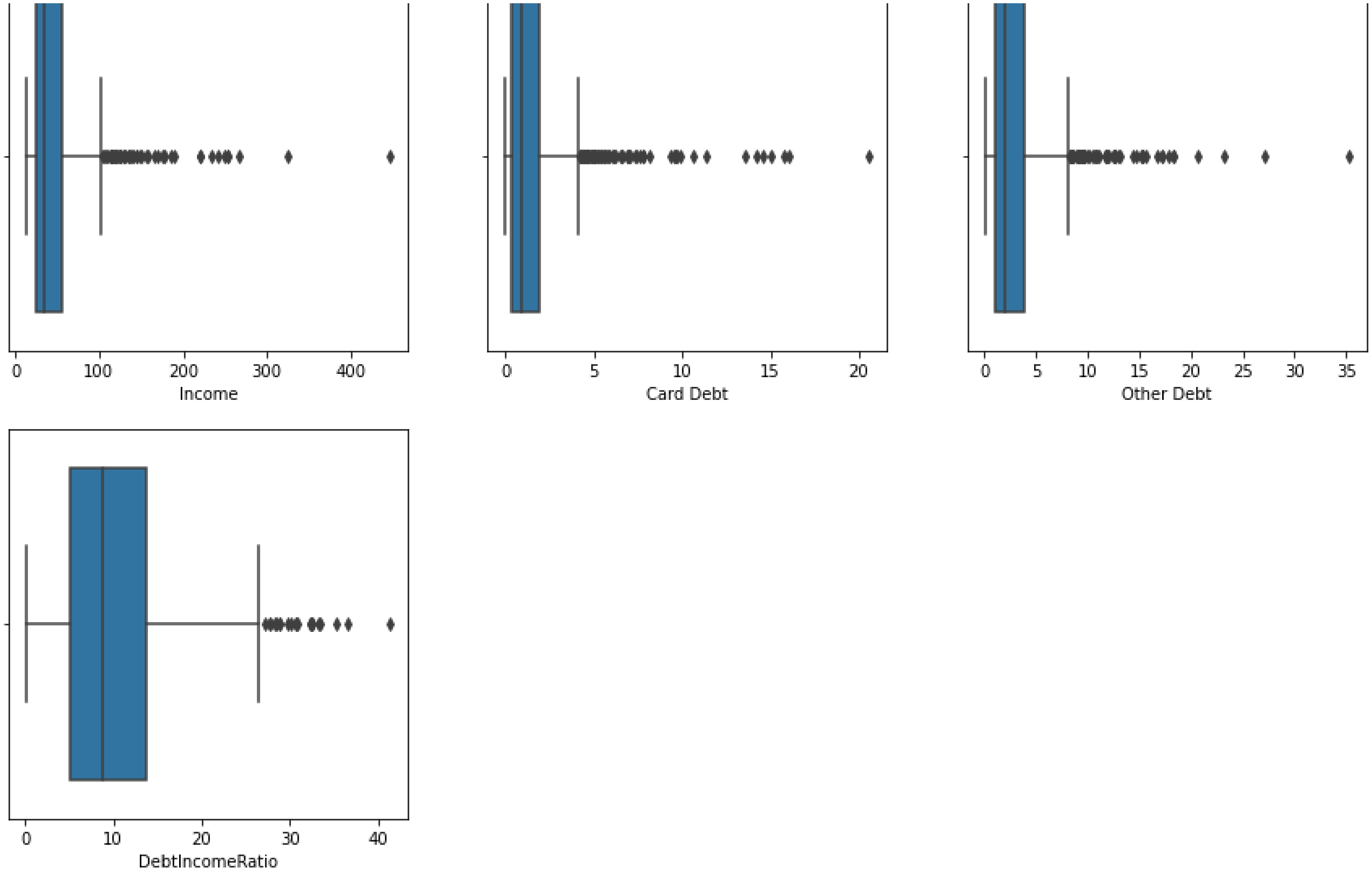
In [11]:

```python
f = df.columns[:]
```

In [12]:

```python
plt.figure(figsize=(15,15))
for col in enumerate(f):
    plt.subplot(3,3,col[0] + 1)
    sns.boxplot(data=df, x=col[1])
```





Drop The Outlier Using Z score

In [13]:

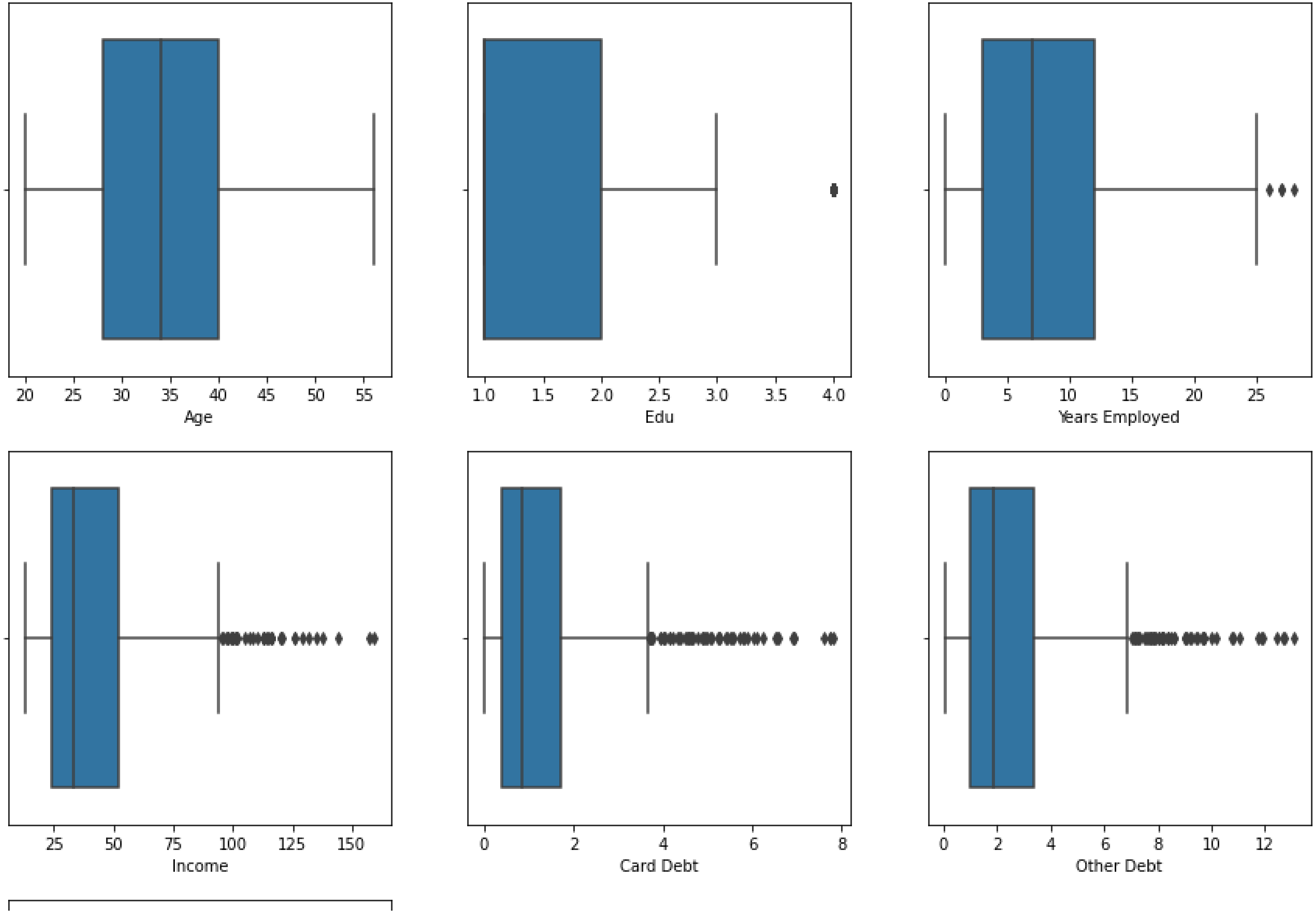
```
z = np.abs(stats.zscore(df))
df = df[(z<3).all(axis=1)]
df.shape
```

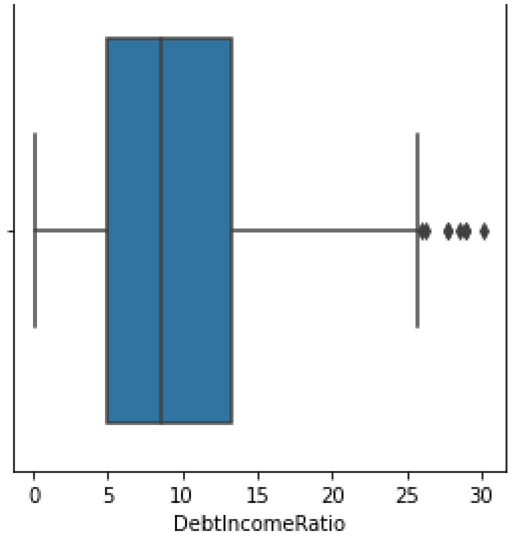
Out[13]: (802, 7)

48 rows are drop when outliers are removed using Z-score

In [14]:

```
plt.figure(figsize=(15,15))
for col in enumerate(f):
    plt.subplot(3,3,col[0] + 1)
    sns.boxplot(data=df, x=col[1])
```





Data Normalizing over the standard deviation

In [16]:
```
df_scalar = StandardScaler().fit_transform(df)
```

In [17]:
```
df_scalar = pd.DataFrame(df_scalar ,columns=df.columns)
```

In [18]:
```
df_scalar
```

Out[18]:

	Age	Edu	Years Employed	Income	Card Debt	Other Debt	DebtIncomeRatio
0	0.816545	0.373949	-0.333197	-0.906959	-0.850017	-0.659345	-0.568286
1	1.579796	-0.766382	2.872850	2.367348	2.375862	2.425850	0.520179
2	-0.201123	0.373949	0.308012	0.629135	3.482271	1.382627	1.876575
3	-0.709957	0.373949	-0.653802	-0.906959	-0.446963	-0.899856	-0.568286
4	0.689337	-0.766382	2.391943	1.599300	-0.217576	2.258744	0.202012
...	...	...	...	...	...	...	...
797	0.816545	-0.766382	-0.172895	0.063206	-0.437556	-0.605370	-0.886453

	Age	Edu	Years Employed	Income	Card Debt	Other Debt	DebtIncomeRatio
798	-0.964374	-0.766382	-0.493499	-0.623994	-0.543204	-0.595871	-0.484558
799	-0.837165	0.373949	-0.172895	-0.300606	-0.679967	-0.250001	-0.451067
800	-0.328331	-0.766382	0.628617	-0.543147	-0.855806	-0.822133	-1.137638
801	2.215839	-0.766382	1.269827	0.912100	0.410522	0.448216	-0.183137

802 rows × 7 columns

Model Building

In [19]:

```
clusterNum = 3
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 12)
k_means.fit(df_scalar)
labels = k_means.labels_
print(labels)
```

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

Let's run K-Means with different value of K to check SILHOUETTE SCORE

In [20]:

```
sil = []
for k in range(2,11):
    kmean = KMeans(n_clusters=k, random_state=0,init = "k-means++", n_init = 12).fit(df_scalar)
    sil.append([k,silhouette_score(df_scalar,kmean.labels_)])
```

In [21]:

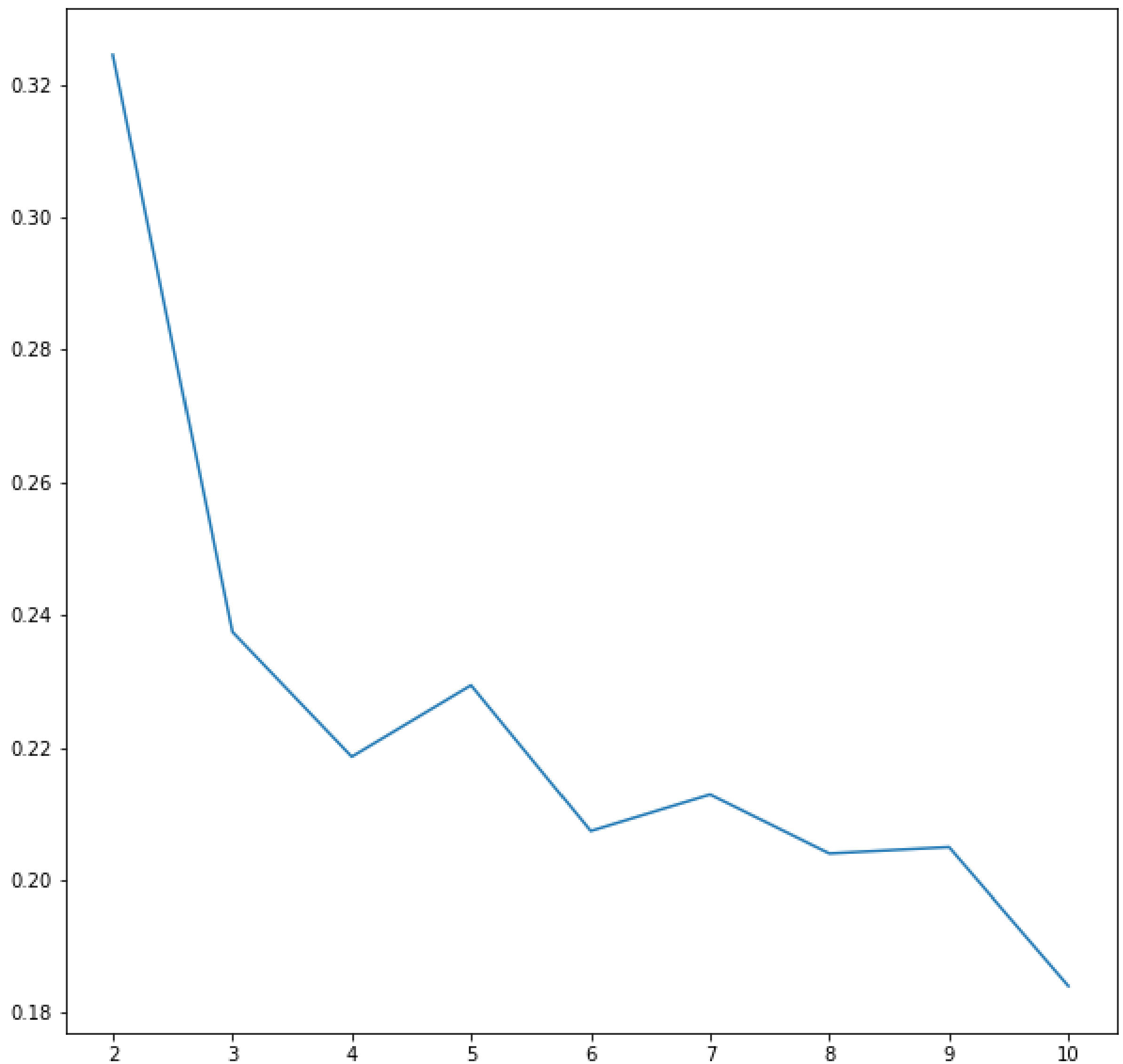
```
sil
```

Out[21]:

```
[[2, 0.32447376073926576],
 [3, 0.23747510415997428],
 [4, 0.2186339487233202],
 [5, 0.22942591581160304],
 [6, 0.20742263869379915],
 [7, 0.21293537439736465],
 [8, 0.20401959301325623],
 [9, 0.20499215165510967],
 [10, 0.18406829643929346]]
```

In [22]:

```
sil = pd.DataFrame(sil)
plt.figure(figsize=(10,10))
plt.plot(sil[0], sil[1])
plt.show()
```



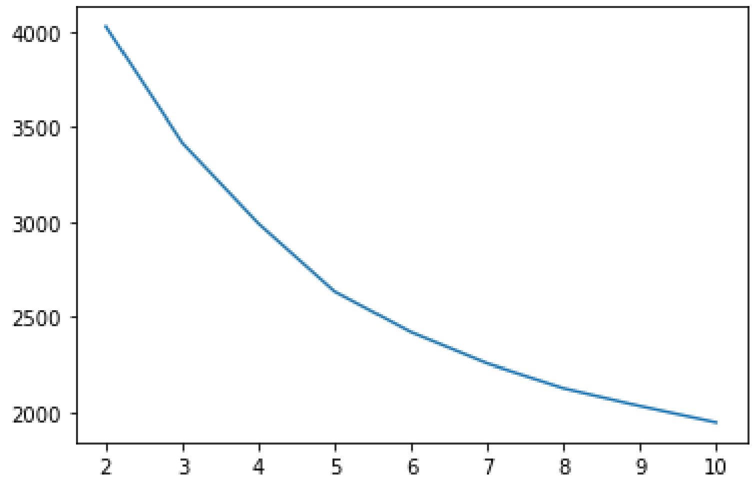
Choosing Value of K

K value with 2 has maximum Silhouette Score is 0.32 but choosing k=2 is not good choice because it means you divide the complete data into two halfs and it's not useful for any business case.
Second Largest Silhouette Score is 0.23 with k= 3,so we can go with k=3

Let's run K-Means with different value of K using Elbow Method

In [23]:

```
ssd = []
for k in range(2,11):
    kmean_elbow  = KMeans(n_clusters=k).fit(df_scalar)
    ssd.append([k, kmean_elbow.inertia_])
ssd = pd.DataFrame(ssd)
plt.plot(ssd[0], ssd[1])
plt.show()
```



In Elbow method we you look at the elbow of the curve here K= 3 form elbow of curve so we select k = 3

In [50]:

```
kmean = KMeans(n_clusters=3, random_state=0,init = "k-means++", n_init = 12)
kmean.fit(df_scalar)
```

Out[50]:

```
KMeans(n_clusters=3, n_init=12, random_state=0)
```

In [51]:

```
labels = k_means.labels_
print(labels)
```

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

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

Assign the labels to each row in the dataframe.

In [52]:
```python
df_scalar["Label"] = labels
df_scalar.head(5)
```

Out[52]:

	Age	Edu	Years Employed	Income	Card Debt	Other Debt	DebtIncomeRatio	Label
0	0.816545	0.373949	-0.333197	-0.906959	-0.850017	-0.659345	-0.568286	1
1	1.579796	-0.766382	2.872850	2.367348	2.375862	2.425850	0.520179	2
2	-0.201123	0.373949	0.308012	0.629135	3.482271	1.382627	1.876575	2
3	-0.709957	0.373949	-0.653802	-0.906959	-0.446963	-0.899856	-0.568286	1
4	0.689337	-0.766382	2.391943	1.599300	-0.217576	2.258744	0.202012	2

In [53]:
```python
df_scalar.Label.value_counts()
```

Out[53]:
```
1    453
0    212
2    137
Name: Label, dtype: int64
```

In [54]:
```
df_scalar.groupby('Label').mean()
```

Out[54]:

Label	Age	Edu	Years Employed	Income	Card Debt	Other Debt	DebtIncomeRatio
0	0.821946	-0.083259	0.849411	0.635237	-0.192339	-0.180070	-0.593642
1	-0.554105	-0.006161	-0.613107	-0.582946	-0.378133	-0.411896	-0.071906
2	0.560271	0.149212	0.712864	0.944557	1.547956	1.640612	1.156391

from above result We can easily check the centroid values by averaging the features in each cluste

conclusion

I have successfully Studied Unsupervised Machine Learning and pratice and implement k- Means Algorithm

In [55]:
```
pred1  = kmean.fit_predict(df_scalar)
score = accuracy_score(df_scalar.Label, pred1)
score
```

Out[55]:
```
1.0
```

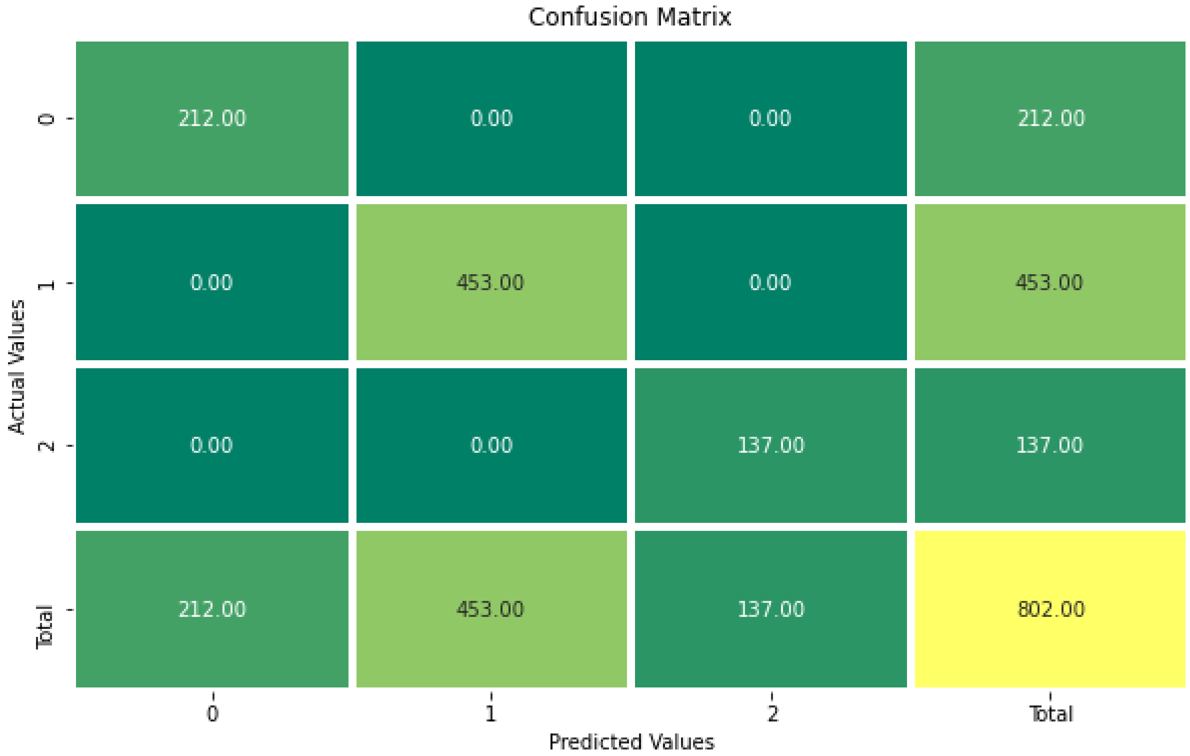
In [56]:
```
print(classification_report(df_scalar.Label, pred1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       212
           1       1.00      1.00      1.00       453
           2       1.00      1.00      1.00       137

    accuracy                           1.00       802
   macro avg       1.00      1.00      1.00       802
weighted avg       1.00      1.00      1.00       802
```

In [57]:
```
label = [0,1,2]
```

In [58]:

```python
# confusion matrix
cm = confusion_matrix(df_scalar.Label, pred1)
row_sum = cm.sum(axis=0)
cm = np.append(cm,row_sum.reshape(1,-1),axis=0)
col_sum = cm.sum(axis=1)
cm = np.append(cm,col_sum.reshape(-1,1),axis=1)
labels = label+['Total']
plt.figure(figsize=(10,6))
sns.heatmap(cm,annot=True,cmap='summer',fmt='0.2f',xticklabels=labels,
yticklabels=labels,linewidths=3,cbar=None,)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
plt.title('Confusion Matrix')
plt.show()
```



Conclusion

Thus I have studied unsupervised learning and Successfully Implemented K means algorithm

In [ ]: