Credit Card Approval

Aathithya-CB.EN.U4ELC20001

Adarsh-CB.EN.U4ELC20005

Arun-CB.EN.U4ELC20010

Summary of CA-1

- To classify people described by a set of attributes as good or bad credit risks for credit card applications.
- K-NN algorithm was used to achieve it.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- It can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

References

- Convergence Criteria:

https://www.ibm.com/docs/en/spss-statistics/beta?topic=analysis-k-means-cluster-convergence-criteria

- Cosine Similarity:

https://analyticsindiamag.com/cosine-similarity-in-machine-learning/

- Euclidean Distance:

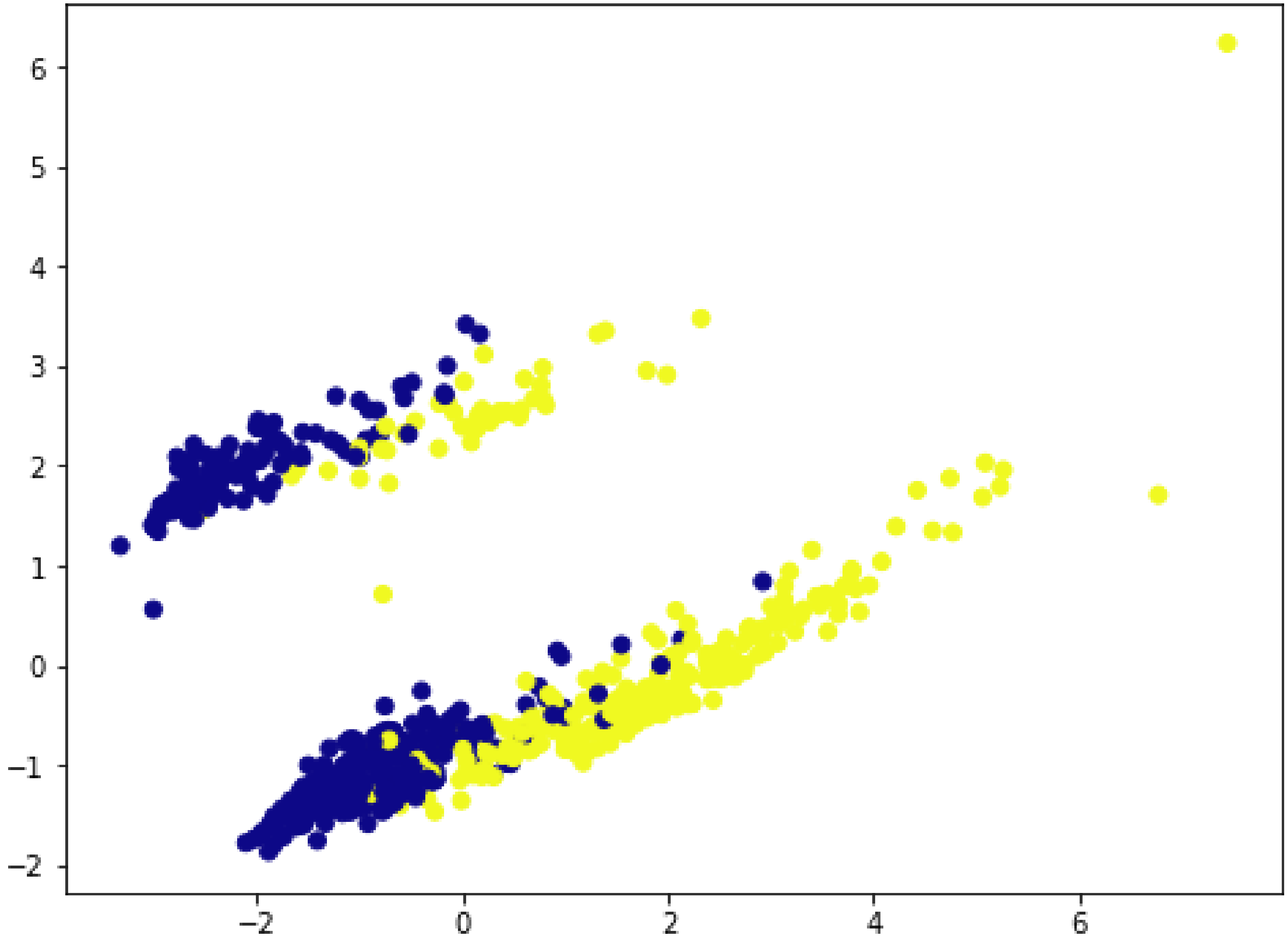
https://www.datanovia.com/en/lessons/clustering-distance-measures/

PCA application (10 marks)

Visualizing the data by applying PCA:

```
#Performing PCA
dataset = pd.read_csv('/content/clean_dataset.csv')
dataset=dataset.drop(['Industry'], axis= 1)
dataset.head()
for i in list(dataset.columns.values):
  dataset[i] = pd.to_numeric(dataset[i],errors = 'coerce')
dataset = dataset.fillna(dataset.mean())
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(dataset)
scaled_data=scaler.transform(dataset)
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
pca.fit(scaled_data)
x_pca=pca.transform(scaled_data)
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,-1],c=dataset['Approved'],cmap='plasma')
plt.show
```

<function matplotlib.pyplot.show(*args, **kw)>



Covariance Matrix for Standard Data (Without Applying PCA):

```
#Computing Covariance Matrix for Standard Data
dataset.drop(['Approved'],axis = 1,inplace = True)
x_std = StandardScaler().fit_transform(dataset)
mean_vec = np.mean(x_std , axis = 0)
cov_mat = (x_std - mean_vec).T.dot((x_std - mean_vec))/(x_std.shape[0]-1)
print('Covariance Matrix', cov_mat)
```

```
Covariance Matrix [[ 1.00145138  0.03509524 -0.04180682 -0.06816087 -0.07135329 -0.1095089
   0.08666977 -0.02608512 -0.07789644 -0.02466603  0.05174869 -0.07302713
   0.08613161 -0.00206599]
 [ 0.03509524  1.00145138  0.20247002  0.10708428  0.09962114  0.23753454
   0.39203177  0.20473073  0.08616193  0.18759925  0.0536764   0.02218441
  -0.07880455  0.01874588]
 [-0.04180682  0.20247002  1.00145138  0.07475713  0.08390301  0.12642039
   0.29933538  0.24467132  0.17509971  0.27160036 -0.01304233  0.09388774
  -0.21821919  0.12329985]
 [-0.06816087  0.10708428  0.07475713  1.00145138  0.99347294 -0.06240678
   0.07004608  0.14528372  0.17568222  0.11413317 -0.0097986   0.00322039
  -0.01709877 -0.0069089 ]
 [-0.07135329  0.09962114  0.08390301  0.99347294  1.00145138 -0.04652539
   0.07601472  0.13873628  0.1705153   0.11123802 -0.00240529  0.0048592
  -0.00952681  0.05735618]
```

- As the dataset contains higher dimensions covariance matrix is calculated to describe the relation between different dimensions.
- The covariance matrix has negative values so the features vary at opposite direction

Covariance Matrix for Data After Applying PCA:

```
#Covariance Matrix of Data by applying PCA
x_pca = StandardScaler().fit_transform(x_pca)
mean_vec = np.mean(x_pca, axis = 0)
cov_mat = (x_pca - mean_vec).T.dot((x_pca - mean_vec))/(x_pca.shape[0]-1)
print('Covariance Matrix', cov_mat)
```

```
Covariance Matrix [[1.00145138e+00 8.57353208e-08]
 [8.57353208e-08 1.00145138e+00]]
```

- To adjust the size of the state PCA is used on covariance matrix
- As you can see there was a large covariance matrix before applying PCA and now it is converted to 2x2 matrix
- The covariance matrix has only positive values so the features vary in same direction now.

Eigen Values and Eigen Vectors for Standard Data (Without Applying PCA):

```
#Computing Eigen Vectors and Eigen Values
cov_mat = np.cov(x_std.T)
eig_vals , eig_vecs = np.linalg.eig(cov_mat)
print('Eigen Vectors',eig_vecs)
print('Eigen Values',eig_vals)
```

```
Eigen Vectors [[-5.15833596e-02 -6.16299643e-02  2.64121264e-03 -4.11910435e-01
  -1.18269671e-01  1.93408193e-02  6.84399573e-02 -7.30066754e-01
   1.77185652e-01 -2.61521517e-01  1.83405026e-02  1.86671068e-01
  -3.69931978e-01 -2.40107384e-04]
 [ 2.70059739e-01 -1.63250562e-01  9.76318837e-03 -5.02881434e-02
  -5.17476372e-01  1.24302875e-01 -3.46517160e-02  3.10899051e-02
   5.27733710e-02 -3.25301169e-01  5.69152559e-01 -8.71846065e-02
   3.89153027e-01  1.44259148e-01]
 [ 3.03765833e-01 -2.01398464e-01 -3.03293584e-03  1.78318949e-01
  -1.53260458e-01  1.05572348e-01  9.91206847e-03 -2.10316270e-01
  -2.19908120e-01  4.48644692e-01  1.91111829e-01  1.08582113e-01
  -9.37601651e-02 -6.70556668e-01]
 [ 3.17635339e-01  6.13814824e-01 -7.06941136e-01  3.59224253e-02
  -1.10481227e-01 -3.10435814e-02 -1.12884697e-02 -4.29221690e-02
  -1.94162140e-02 -4.14157937e-02 -3.25459859e-02  1.14092878e-02
  -5.11471503e-02 -1.56752613e-02]
```

```
Eigen Values [2.78105821 1.84122623 0.00560917 1.37857406 1.28935962 0.38628149
 1.10915691 0.98181197 0.88339059 0.85364602 0.52754061 0.58342312
 0.72964381 0.66959749]
```

Eigen Values and Eigen Vectors for Data After Applying PCA:

```
#Computing Eigen Vectors and Eigen Values for the 2nd Covariance Matrix
cov_mat = np.cov(x_pca.T)
eig_vals , eig_vecs = np.linalg.eig(cov_mat)
print('Eigen Vectors',eig_vecs)
print('Eigen Values',eig_vals)
```

```
Eigen Vectors [[ 1.00000000e+00  1.18435813e-07]
 [-1.18435813e-07  1.00000000e+00]]
Eigen Values [3.247393   1.85487611]
```

- Always highest Eigen value is selected in this case(3.247393). Because the greater the Eigenvalue, the longer the Eigen vector.

Sum of Upper Triangular Values in Covariance Matrix for Standard Data (Without Applying PCA):

```
#Computing Sum of Upper Triangle of 1st Covariance Matrix
sup = 0
for i in range(0,13):
  for j in range(0,13):
    if i>j:
      sup = sup + cov_mat[i][j]
print('Sum of Upper Triangle of Co-Variance Matrix :',sup)
```

```
Sum of Upper Triangle of Co-Variance Matrix : 6.767849990600766
```

- As you can see the sum of upper triangle values is positive before applying PCA

Sum of Upper Triangular Values in Covariance Matrix for Data After Applying PCA:

```
#Computing Sum of Upper Triangular of 2nd Covariance Matrix
sup = 0
for i in range(0,2):
  for j in range(0,2):
    if i>j:
      sup = sup + cov_mat[i][j]
print('Sum of Upper Triangle of Co-Variance Matrix :',sup)
```

```
Sum of Upper Triangle of Co-Variance Matrix : -1.6492386927833614e-07
```

- As you can see the sum of upper triangle values negative after applying PCA.
- That is because when PCA is applied dimensionality of the matrix decrease which result in decrease in value of upper triangle of covariance matrix.

Link for the Code:

https://colab.research.google.com/drive/1123Kr7DNfSNKZrbotho6gO4ZcqkalRpa?usp=sharing

Link for the dataset:

https://www.kaggle.com/datasets/samuelcortinhas/credit-card-approval-clean-data

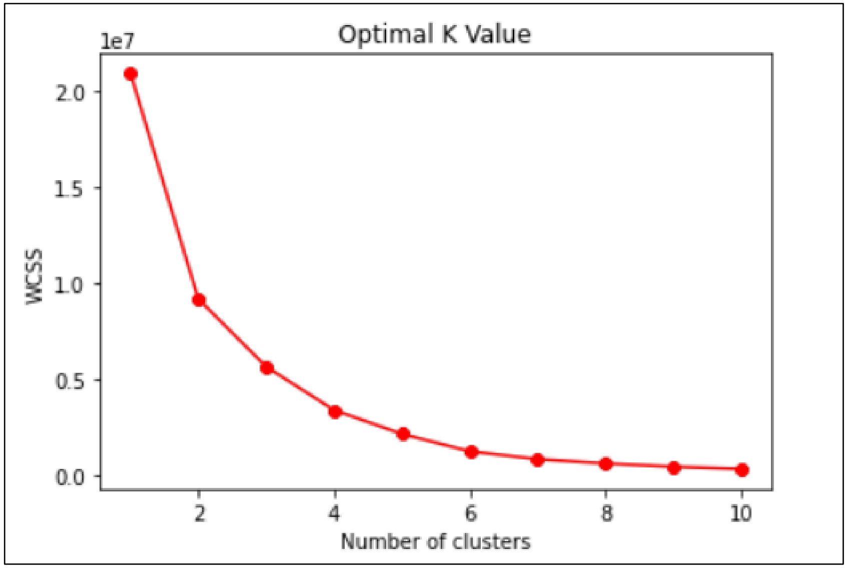
Clustering (10 marks)

```
# Segregating & Zipping Dataset
Debt = dataset['Debt'].values
YearsEmployed = dataset['YearsEmployed'].values
CreditScore = dataset['CreditScore'].values
ZipCode = dataset['ZipCode'].values
X = np.array(list(zip(Debt,YearsEmployed,CreditScore,ZipCode)))
X
```

```
   Gender    Age   Debt  Married  BankCustomer     Industry  Ethnicity  \
0       1  30.83  0.000        1             1  Industrials          0
1       0  58.67  4.460        1             1    Materials          1
2       0  24.50  0.500        1             1    Materials          1
3       1  27.83  1.540        1             1  Industrials          0
4       1  20.17  5.625        1             1  Industrials          0

   YearsEmployed  PriorDefault  Employed  CreditScore  DriversLicense  \
0           1.25             1         1            1               0
1           3.04             1         1            6               0
2           1.50             1         0            0               0
3           3.75             1         1            5               1
4           1.71             1         0            0               0

   Citizen  ZipCode  Income  Approved
0        1      202       0         1
1        1       43     560         1
2        1      280     824         1
3        1      100       3         1
4        0      120       0         1
```

Finding Optimal Value of K

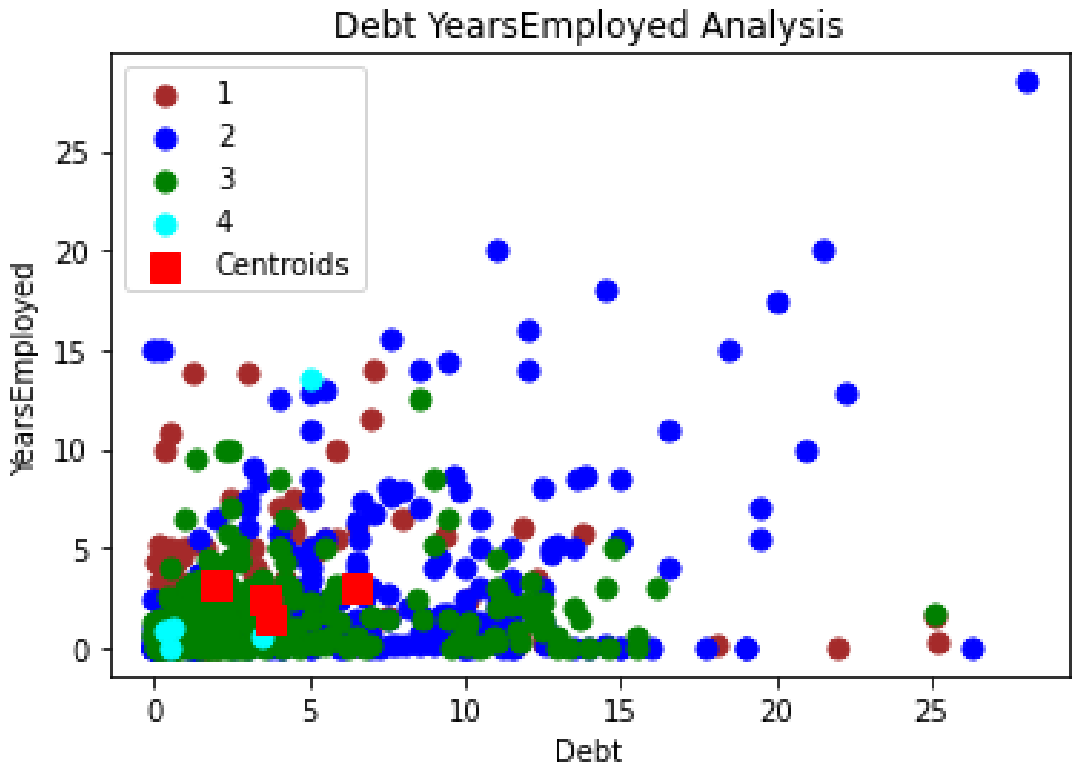
```
# Finding the Optimized K Value
from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
    km=KMeans(n_clusters=i, random_state=0)
    km.fit(X)
    wcss.append(km.inertia_)
plt.plot(range(1,11),wcss,color="red", marker ="8")
plt.title('Optimal K Value')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



- No of cluster take is 10 where 10 features from dataset out of 16 is consider to calculate k value
- From the graph it is identified k value = **4**

Visualizing the Clusters for Optimal Value of K

```
"""### Visualizing the clusters for k=4

Cluster 1: Customers with medium debt and low yearsEmployed

Cluster 2: Customers with high debt and medium to high yearsEmployed

Cluster 3: Customers with low debt

Cluster 4: Customers with medium debt but high yearsEmployed
"""


plt.scatter(X[y_means==0,0],X[y_means==0,1],s=50, c='brown',label='1')
plt.scatter(X[y_means==1,0],X[y_means==1,1],s=50, c='blue',label='2')
plt.scatter(X[y_means==2,0],X[y_means==2,1],s=50, c='green',label='3')
plt.scatter(X[y_means==3,0],X[y_means==3,1],s=50, c='cyan',label='4')
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1],s=100,marker='s', c='red', label='Centroids')
plt.title('Debt YearsEmployed Analysis')
plt.xlabel('Debt')
plt.ylabel('YearsEmployed')
plt.legend()
plt.show()
```



- The graph shows the visualization of cluster of k value **4.**
- As you can see 4 different clusters are described in 4 different colours.

Computing K-Means and fitting the clusters into the dataset

```
# Computing the K-Means for K=4 and fitting the clusters into the dataset
dataset.drop(dataset.iloc[:,14:53],axis = 1,inplace = True)
from sklearn.cluster import KMeans
clusters = KMeans(4)
dataset=dataset.drop(['Industry'], axis= 1) # dropping of columns as mentioned
clusters.fit(dataset)
dataset["clusterid"] = clusters.labels_
dataset[0:6]
```

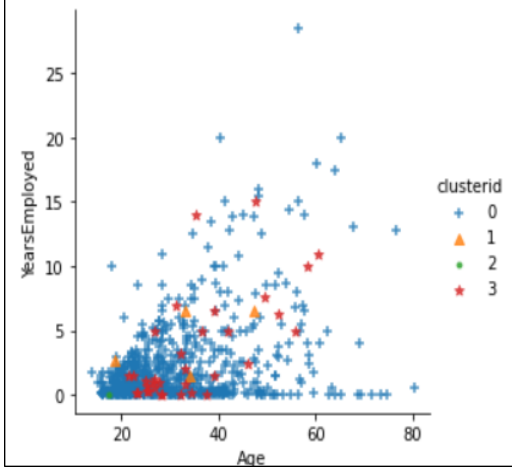
	Gender	Age	Debt	Married	BankCustomer	Ethnicity	YearsEmployed	PriorDefault	Employed	CreditScore
0	1	30.83	0.000	1	1	0	1.25	1	1	1
1	0	58.67	4.460	1	1	1	3.04	1	1	6
2	0	24.50	0.500	1	1	1	1.50	1	0	0
3	1	27.83	1.540	1	1	0	3.75	1	1	5
4	1	20.17	5.625	1	1	0	1.71	1	0	0
5	1	32.08	4.000	1	1	0	2.50	1	0	0

- K-Means is computed to minimize the sum of distances between the points and their respective cluster centroid.
- Cluster are created and fitted inside dataset in the name of clusterid

Visualizing the clusters for Age and Years Employed Columns

```
#Visualizing the clusters for Age and YearsEmployed columns before Normalization
markers = ['+','^','.','*']
sn.lmplot('Age','YearsEmployed',data=dataset, hue = 'clusterid', fit_reg=False, markers = markers,height = 4);
```

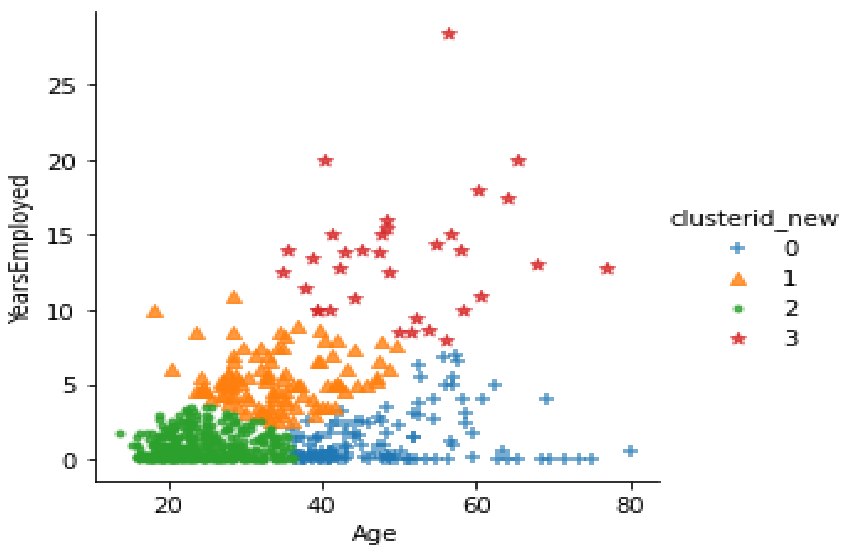
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```



- As you can see the clusters are scattered and are not observed in sorted manner as Normalization of data is not done.

Visualizing the clusters for Age and Years Employed Columns After Normalization

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler ()
scaled_dataset = scaler.fit_transform(dataset[['Age','YearsEmployed']])
scaled_dataset [0:5]

from sklearn.cluster import KMeans
clusters_new = KMeans ( 4, random_state=42)
clusters_new.fit( scaled_dataset)
dataset["clusterid_new"] = clusters_new. labels_
markers = ['+','^','.','*']
sn.lmplot('Age','YearsEmployed', data=dataset, hue = 'clusterid_new',fit_reg=False,markers=markers,height=4);
```

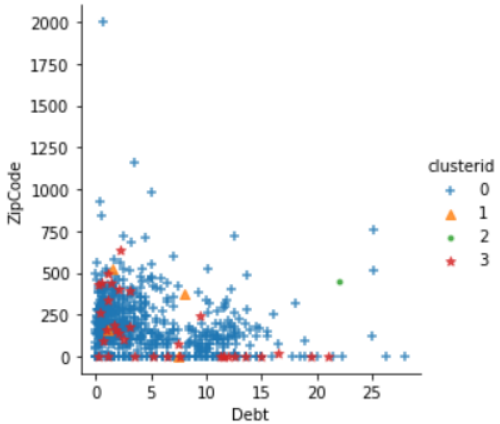


- Now you can observe the clusters in sorted way due to normalization of data.

Visualizing the clusters for Debt and Zip Code Columns before Normalization

```
#Visualizing the clusters for Debt and Zipcode columns before Normalization
markers = ['+','^','.','*']
sn.lmplot('Debt','ZipCode',data=dataset, hue = 'clusterid', fit_reg=False, markers = markers,height = 4);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```

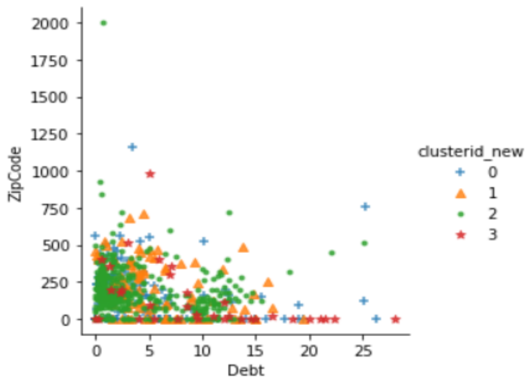


- As you can see the clusters are scattered and are not observed in sorted manner as Normalization of data is not done.

Visualizing the clusters for Debt and Zip Code Columns after Normalization

```
#Visualizing the clusters for Debt and Zipcode columns after Normalization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler ()
scaled_dataset = scaler.fit_transform(dataset[['Age','YearsEmployed']])
scaled_dataset [0:5]

from sklearn.cluster import KMeans
clusters_new = KMeans ( 4, random_state=42)
clusters_new.fit( scaled_dataset)
dataset["clusterid_new"] = clusters_new. labels_
markers = ['+','^','.','*']
sn.lmplot('Debt','ZipCode', data=dataset, hue = 'clusterid_new',fit_reg=False,markers=markers,height=4);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y.
  FutureWarning
```



- Now you can observe the clusters in sorted way due to normalization of data.

- Two classes are involved in our dataset
- The k value selected is related to number of classes
- As from this dataset cluster 4(medium debt and high year experience) will be approved for credit card
- As the k value is 4 which helps us to organize the data into 4 different cluster to determine which of the 2 classes it belongs

Link for the Code

- Link to full code:

https://colab.research.google.com/drive/1la-oFYqqXSbe__TpF3j6qGkgy8keVYJx?usp=sharing

- Link of Datasheet:

https://www.kaggle.com/datasets/samuelcortinhas/credit-card-approval-clean-data

Miscellaneous

- Euclidean Distance or Minimum Distance is the method used to find the distance between observations.
- Given a set of points in the two-dimensional plane, the minimum Euclidean distance between two distinct points is:

$$d(x,\ y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

- Cosine Similarity: This method is used to compare two different vectors on the basis of how similar their directions are regardless of magnitude.
- This method can be used to replace Euclidean Distance in Clustering Method

- <u>Convergence Criteria </u>is used in clustering algorithms to check if the data points are completely grouped into correct clusters.
- Each **Eigenvector** has a corresponding **eigenvalue**. It is the factor by which the eigenvector gets scaled, when it gets transformed by the matrix.
- Using eigenvalues and eigenvectors, we can find the main axes of our data.
- Principal component analysis uses the power of eigenvectors and eigenvalues to reduce the number of features in our data, while keeping most of the variance
- In PCA we specify the number of components we want to keep beforehand.