SECTION A

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#Q1-Display the number of attributes available in the dataset

data = pd.read_csv('exam_dataset.csv')
```

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Age                      1470 non-null   int64 
 1   BusinessTravel           1470 non-null   object
 2   MonthlyIncome            1470 non-null   int64 
 3   JobSatisfaction          1470 non-null   int64 
 4   Bonus                    1470 non-null   int64 
 5   Department               1470 non-null   object
 6   DistanceFromHome         1470 non-null   int64 
 7   Education                1470 non-null   int64 
 8   EducationField           1470 non-null   object
 9   EmployeeCount            1470 non-null   int64 
 10  EmployeeNumber           1470 non-null   int64 
 11  EnvironmentSatisfaction  1470 non-null   int64 
 12  Gender                   1470 non-null   object
 13  JobLevel                 1470 non-null   int64 
 14  JobRole                  1470 non-null   object
 15  MaritalStatus            1470 non-null   object
 16  PerformanceRating        1470 non-null   int64 
 17  StockOptionLevel         1470 non-null   int64 
 18  TrainingTimesLastYear    1470 non-null   int64 
 19  WorkLifeBalance          1470 non-null   int64 
 20  YearsAtCompany           1470 non-null   int64 
 21  YearsSinceLastPromotion  1470 non-null   int64 
 22  OverTime                 1470 non-null   object
 23  Attrition                1470 non-null   object
dtypes: int64(16), object(8)
memory usage: 275.8+ KB
```

In [4]:
```
data.head()
```

Out[4]:

	Age	BusinessTravel	MonthlyIncome	JobSatisfaction	Bonus	Department	DistanceFromHome	Education	EducationField	EmployeeCount	...	JobRole	MaritalStatus	PerformanceRating	StockOptionLevel	Trainir
0	41	Travel_Rarely	5993	4	17979	Sales	1	2	Life Sciences	1	...	Sales Executive	Single	3	0	
1	49	Travel_Frequently	5130	2	20520	Research & Development	8	1	Life Sciences	1	...	Research Scientist	Married	4	1	
2	37	Travel_Rarely	2090	3	6270	Research & Development	2	2	Other	1	...	Laboratory Technician	Single	3	0	
3	33	Travel_Frequently	2909	3	8727	Research & Development	3	4	Life Sciences	1	...	Research Scientist	Married	3	0	
4	27	Travel_Rarely	3468	2	10404	Research & Development	2	1	Medical	1	...	Laboratory Technician	Married	3	1	

5 rows × 24 columns

In [5]:
```
#Q2-Find the dimension number of this dataset

data.shape
```

Out[5]: (1470, 24)

In [6]:
```
print("The dimension of dataset is: ",data.ndim)
```

```
The dimension of dataset is:  2
```

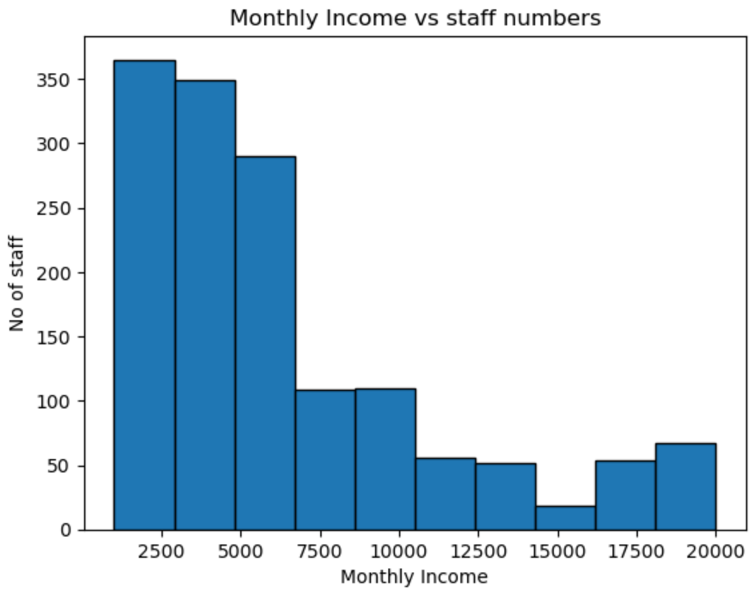
In [7]:
```
#Q3-Display the average of these attributes: 'Age', 'Monthly Income'
#and 'Years at Company'. Be sure to round your answer to 2 decimal places

age = np.average(data['Age'])
income = np.average(data['MonthlyIncome'])
years = np.average(data['YearsAtCompany'])

print("Average age is: ", round(age,2))
print("Average Monthly income is: ", round(income,2))
print("Average Years at company is: ", round(years,2))
```

```
Average age is:  36.92
Average Monthly income is:  6502.93
Average Years at company is:  7.01
```

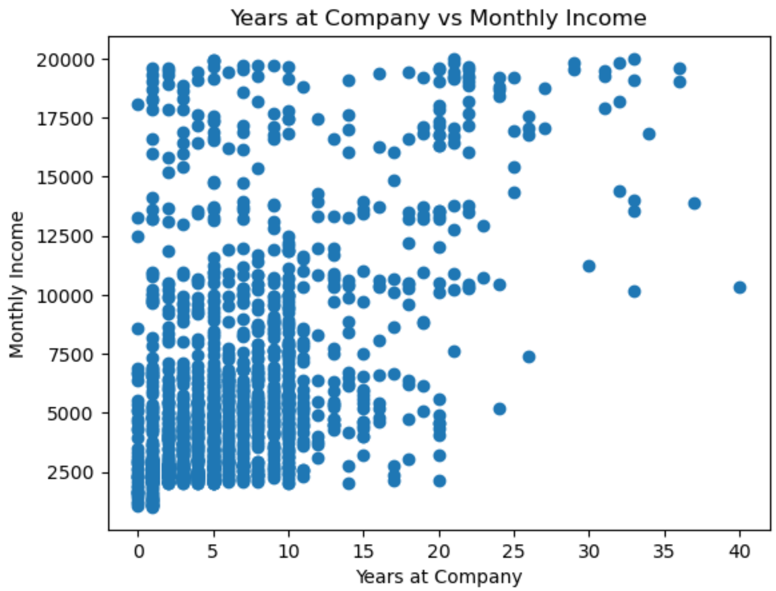
In [8]:
```
#Q4-Find the minimum and maximum 'Monthly Income'

minMI = min(data['MonthlyIncome'])
print("Minimum income is: ", minMI)

maxMI = max(data['MonthlyIncome'])
print("Maximum income is: ", maxMI)
```

```
Minimum income is:  1009
Maximum income is:  19999
```

In [9]:
```
#Q5-Histogram of ‘Monthly Income vs staff numbers’

plt.hist(data['MonthlyIncome'], edgecolor='black')
plt.title('Monthly Income vs staff numbers')
plt.xlabel('Monthly Income')
plt.ylabel('No of staff')
plt.show()
```



In [10]:
```
#Q6-Provide graphical Visualization of the distribution between
#'Year at Company' and 'Monthly Income' using the scatter plot

x = data.iloc[:, 20].values
y = data.iloc[:, 2].values

plt.scatter(x,y)
plt.title('Years at Company vs Monthly Income')
plt.xlabel('Years at Company')
plt.ylabel('Monthly Income')
plt.show()
```



In [11]:
```
#Q7-Find the correlation between 'Years at Company' and 'Monthly Income'

cor = cor = data[['YearsAtCompany','MonthlyIncome']].corr()
cor
```

Out[11]:

	YearsAtCompany	MonthlyIncome
YearsAtCompany	1.000000	0.514285
MonthlyIncome	0.514285	1.000000

#Q8-

a) Range of Monthly Income at Company A is from 1009 to 19999

In [12]:
```
data.MonthlyIncome.value_counts()
```

Out[12]:
```
2342     4
6142     3
2741     3
2559     3
2610     3
        ..
7104     1
2773     1
19513    1
3447     1
4404     1
Name: MonthlyIncome, Length: 1349, dtype: int64
```

b) From the histogram, monthly income values at Company A: -Most Frequent monthly income values at Company A around +-2500 ;more specifically 2342 with frequency of 4 -Least Frequent monthly income values at Company A around +-15000

c) Histogram of Monthly Income shows a distribution which peaks in the lower income(up until 5000) , with average income of 6502.93.The monthly income is quite concentrated in the first quartile, and scattered in forth quartile.These observations suggest that most frequent monthly income is first quartile.

d) There are positive linear relationship with correlation of 0.51 correlation between them. However, these correlation is not significant in determine the relationship between income and years of service at Company A

SECTION B

In [13]:
```
#Q1-Import necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [14]:
```
#Q2-Import dataset

data = pd.read_csv('exam_dataset.csv')
```

In [15]:
```
data.head()
```

Out[15]:

	Age	BusinessTravel	MonthlyIncome	JobSatisfaction	Bonus	Department	DistanceFromHome	Education	EducationField	EmployeeCount	...	JobRole	MaritalStatus	PerformanceRating	StockOptionLevel	Trainir
0	41	Travel_Rarely	5993	4	17979	Sales	1	2	Life Sciences	1	...	Sales Executive	Single	3	0	
1	49	Travel_Frequently	5130	2	20520	Research & Development	8	1	Life Sciences	1	...	Research Scientist	Married	4	1	
2	37	Travel_Rarely	2090	3	6270	Research & Development	2	2	Other	1	...	Laboratory Technician	Single	3	0	
3	33	Travel_Frequently	2909	3	8727	Research & Development	3	4	Life Sciences	1	...	Research Scientist	Married	3	0	
4	27	Travel_Rarely	3468	2	10404	Research & Development	2	1	Medical	1	...	Laboratory Technician	Married	3	1	

5 rows × 24 columns

In [16]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 24 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   Age                      1470 non-null   int64 
 1   BusinessTravel           1470 non-null   object
 2   MonthlyIncome            1470 non-null   int64 
 3   JobSatisfaction          1470 non-null   int64 
 4   Bonus                    1470 non-null   int64 
 5   Department               1470 non-null   object
 6   DistanceFromHome         1470 non-null   int64 
 7   Education                1470 non-null   int64 
 8   EducationField           1470 non-null   object
 9   EmployeeCount            1470 non-null   int64 
 10  EmployeeNumber           1470 non-null   int64 
 11  EnvironmentSatisfaction  1470 non-null   int64 
 12  Gender                   1470 non-null   object
 13  JobLevel                 1470 non-null   int64 
 14  JobRole                  1470 non-null   object
 15  MaritalStatus            1470 non-null   object
 16  PerformanceRating        1470 non-null   int64 
 17  StockOptionLevel         1470 non-null   int64 
 18  TrainingTimesLastYear    1470 non-null   int64 
 19  WorkLifeBalance          1470 non-null   int64 
 20  YearsAtCompany           1470 non-null   int64 
 21  YearsSinceLastPromotion  1470 non-null   int64 
 22  OverTime                 1470 non-null   object
 23  Attrition                1470 non-null   object
dtypes: int64(16), object(8)
memory usage: 275.8+ KB
```

In [17]:
```python
#Q3-Allocate the relevant attributes as input and output

x = data.iloc[:, [0,1,2,3]].values
y = data.iloc[:, 23].values

print(x)
print(y)
```

```
[[41 'Travel_Rarely' 5993 4]
 [49 'Travel_Frequently' 5130 2]
 [37 'Travel_Rarely' 2090 3]
 ...
 [27 'Travel_Rarely' 6142 2]
 [49 'Travel_Frequently' 5390 2]
 [34 'Travel_Rarely' 4404 3]]
['Yes' 'No' 'Yes' ... 'No' 'No' 'No']
```

In [18]:
```python
np.unique(x[:,1])
```

Out[18]:
```
array(['Non-Travel', 'Travel_Frequently', 'Travel_Rarely'], dtype=object)
```

In [ ]:
```python

```

In [19]:
```
#Q4-Use LabelEncoder to encode categorical data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
lc_x = LabelEncoder()
x[:,1] = lc_x.fit_transform(x[:,1])

lc_y = LabelEncoder()
y = lc_y.fit_transform(y)
```

In [20]:
```
print(x)
print(y)
```

```
[[41 2 5993 4]
 [49 1 5130 2]
 [37 2 2090 3]
 ...
 [27 2 6142 2]
 [49 1 5390 2]
 [34 2 4404 3]]
[1 0 1 ... 0 0 0]
```

In [21]:
```
from sklearn.compose import ColumnTransformer

ct = ColumnTransformer([('BusinessTravel', OneHotEncoder(), [1])], remainder='passthrough')
x = ct.fit_transform(x)

print(x)
```

```
[[0.0 0.0 1.0 41 5993 4]
 [0.0 1.0 0.0 49 5130 2]
 [0.0 0.0 1.0 37 2090 3]
 ...
 [0.0 0.0 1.0 27 6142 2]
 [0.0 1.0 0.0 49 5390 2]
 [0.0 0.0 1.0 34 4404 3]]
```

In [22]:
```
x = x[:, 2:]
print(x)
```

```
[[1.0 41 5993 4]
 [0.0 49 5130 2]
 [1.0 37 2090 3]
 ...
 [1.0 27 6142 2]
 [0.0 49 5390 2]
 [1.0 34 4404 3]]
```

In [23]:
```
#Q5-Split your data into training and test sets

from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
                                               random_state=0)
```

In [24]:
```
#Q6-Normalized your data using StandardScaler

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)

print(x_train)
```

```
[[ 0.64565275  2.3389367   2.41725694  1.14972558]
 [ 0.64565275  0.9043263  -0.91612115  1.14972558]
 [ 0.64565275  0.35255307  0.41020041 -1.57257768]
 ...
 [ 0.64565275  0.68361701  0.29395671  1.14972558]
 [ 0.64565275  0.13184377 -0.72026428  0.24229116]
 [ 0.64565275  0.35255307  0.68736435 -0.66514326]]
```

In [25]:
```
#Q7-Fit the and predict results using the Naïve Bayes Classifier

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(x_train,y_train)
```

Out[25]:
```
GaussianNB()
```

In [26]:
```
y_pred = classifier.predict(x_test)
```

In [27]:
```
print(y_test)
print(y_pred)
```

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

In [28]:
```
#Q8-Evaluate your results using confusion matrix and calculate
#the prediction accuracy

from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test,y_pred)
cm
```

Out[28]:
```
array([[242,   3],
       [ 47,   2]], dtype=int64)
```

In [29]:
```
score = accuracy_score(y_test,y_pred)
score
```

Out[29]:
```
0.8299319727891157
```

In [30]:
```
#Q9-Discuss your results and findings

from sklearn.metrics import classification_report
accuracy = round(accuracy_score(y_test, y_pred),4)*100
error = round(100 - accuracy,4)
print("Accuracy:",accuracy,'%')
print("Error rate:",error,'%')
print(classification_report(y_test, y_pred))
```

```
Accuracy: 82.99 %
Error rate: 17.01 %
              precision    recall  f1-score   support

           0       0.84      0.99      0.91       245
           1       0.40      0.04      0.07        49

    accuracy                           0.83       294
   macro avg       0.62      0.51      0.49       294
weighted avg       0.76      0.83      0.77       294
```

The model has an accuracy of 82.99%. This indicate that it is a good model.

Based on the result, we could say that "Age", "BusinessTravel", "MonthlyIncome", "JobSatisfaction" are important features to predict staff attrition.

SECTION C

In [31]:
```
#Q1-Perform K-Means clustering (use WCSS to help find best K value)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

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

In [32]:
```
data = pd.read_csv('clustering.csv')
```

In [33]:
```
data.head()
```

Out[33]:

	Unnamed: 0	A	B
0	0	0.329241	0.841783
1	1	1.697407	-0.236075
2	2	-0.831460	0.584743
3	3	1.825271	-0.297894
4	4	1.236577	0.121528

In [34]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300 entries, 0 to 299
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  300 non-null    int64  
 1   A           300 non-null    float64
 2   B           300 non-null    float64
dtypes: float64(2), int64(1)
memory usage: 7.2 KB
```

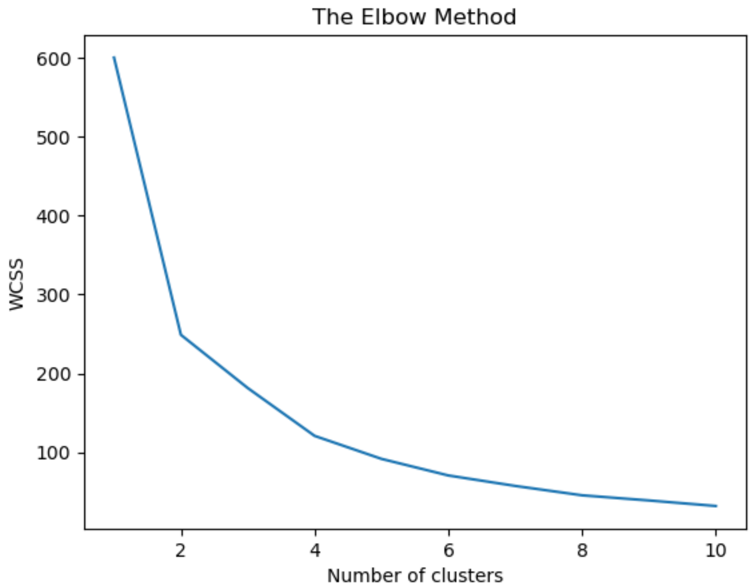
In [35]:
```
x = data.iloc[:, [1,2]].values

print(x)
```

```
 [-3.16613472e-01  9.12637022e-01]
 [-7.44288620e-01  5.33883584e-01]
 [ 8.82870511e-01 -9.39191030e-02]
 [ 4.70862998e-01 -4.48137064e-01]
 [ 3.30129133e-01  9.48687978e-01]
 [ 1.85021110e+00  5.27278518e-01]
 [ 5.10160050e-02  2.08282719e-01]
 [-3.04212080e-02  4.55428711e-01]
 [ 2.65729662e-01  9.58333858e-01]
 [ 1.14466723e+00 -4.72098220e-01]
 [ 2.11063283e+00  3.15386651e-01]
 [ 7.47395558e-01  7.27057584e-01]
 [-1.01618313e+00  9.48843610e-02]
 [ 9.60881972e-01 -3.70942319e-01]
 [ 2.11616125e+00  1.28657950e-02]
 [ 1.65823396e+00 -4.74901130e-02]
 [-9.64387544e-01  4.18223971e-01]
 [-5.88863100e-02  8.52369728e-01]
 [ 1.31991089e+00 -4.68789511e-01]
 [ 7.06101557e-01  5.84692524e-01]
```

In [36]:
```
#Apply StandardScaler

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_scaled = sc.fit_transform(x)
```

In [37]:

```
#Apply Elbow Method

from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state=42)
    kmeans.fit(x_scaled)
    wcss.append(kmeans.inertia_)

plt.plot(range(1,11),wcss)
plt.title("The Elbow Method")
plt.xlabel("Number of clusters")
plt.ylabel("WCSS")
plt.show()
```



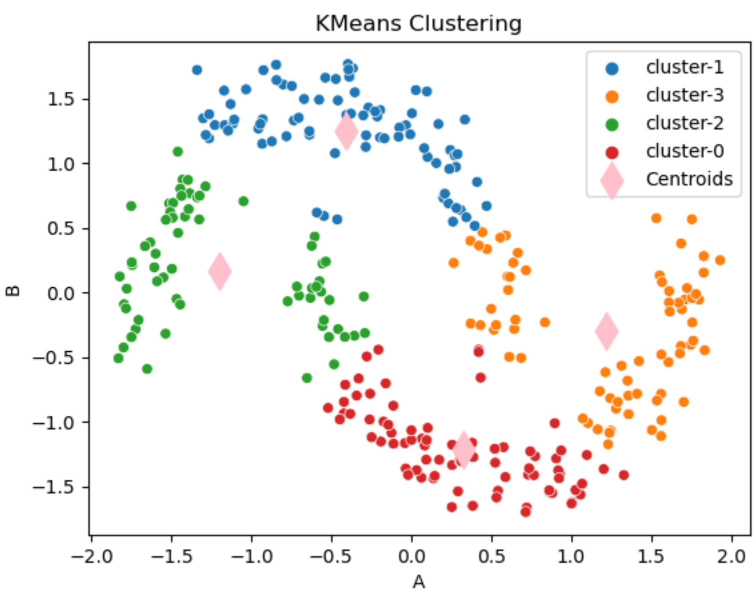
In [38]:

```
#Fitting training sets to KMeans

kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state=42)
kmeans.fit(x_scaled)

y_kmeans = kmeans.predict(x_scaled)
```

In [39]:
```
y_kmeans
```

Out[39]:
```
array([1, 3, 2, 3, 3, 3, 2, 2, 0, 0, 2, 3, 0, 2, 2, 2, 3, 0, 1, 3, 3, 0,
       1, 1, 1, 0, 1, 0, 2, 0, 2, 2, 0, 1, 3, 3, 2, 0, 1, 1, 0, 0, 1, 0,
       3, 1, 1, 2, 0, 1, 3, 2, 0, 2, 1, 0, 1, 3, 1, 3, 3, 1, 3, 0, 3, 3,
       3, 2, 3, 3, 3, 0, 2, 0, 1, 1, 2, 3, 1, 3, 0, 2, 1, 2, 0, 0, 1, 3,
       2, 2, 1, 0, 3, 1, 2, 0, 3, 3, 2, 1, 0, 1, 0, 0, 2, 1, 3, 0, 1, 3,
       0, 1, 1, 1, 0, 1, 2, 1, 0, 3, 2, 2, 1, 0, 2, 3, 0, 0, 1, 1, 0, 3,
       3, 0, 1, 0, 1, 0, 0, 0, 1, 3, 0, 1, 3, 0, 0, 3, 1, 3, 1, 0, 3, 1,
       1, 3, 1, 1, 1, 0, 1, 1, 0, 3, 1, 1, 3, 3, 2, 2, 0, 0, 3, 3, 1, 2,
       2, 0, 3, 1, 1, 0, 1, 3, 2, 0, 1, 1, 0, 1, 0, 1, 2, 2, 2, 0, 2, 1,
       2, 3, 1, 1, 0, 1, 3, 0, 0, 3, 2, 0, 3, 1, 2, 2, 3, 2, 1, 2, 2, 2,
       3, 0, 1, 2, 2, 2, 2, 0, 0, 0, 3, 1, 0, 3, 3, 0, 0, 1, 1, 1, 0, 1,
       3, 1, 2, 3, 0, 3, 0, 2, 3, 2, 2, 2, 3, 2, 3, 2, 1, 3, 2, 2, 3, 1,
       0, 1, 3, 3, 2, 0, 3, 0, 3, 3, 0, 2, 1, 1, 0, 1, 2, 1, 1, 0, 1, 2,
       2, 1, 2, 2, 0, 2, 1, 0, 0, 3, 0, 3, 1, 3])
```

In [40]:
```
#Plot the clusters

sns.scatterplot(x_scaled[:,0],x_scaled[:,1], hue=
                ['cluster-{}'.format(x) for x in y_kmeans])
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],
            marker ='d',s=200, c='pink', label="Centroids")

plt.title('KMeans Clustering')
plt.legend()
plt.xlabel('A')
plt.ylabel('B')
plt.show()
```

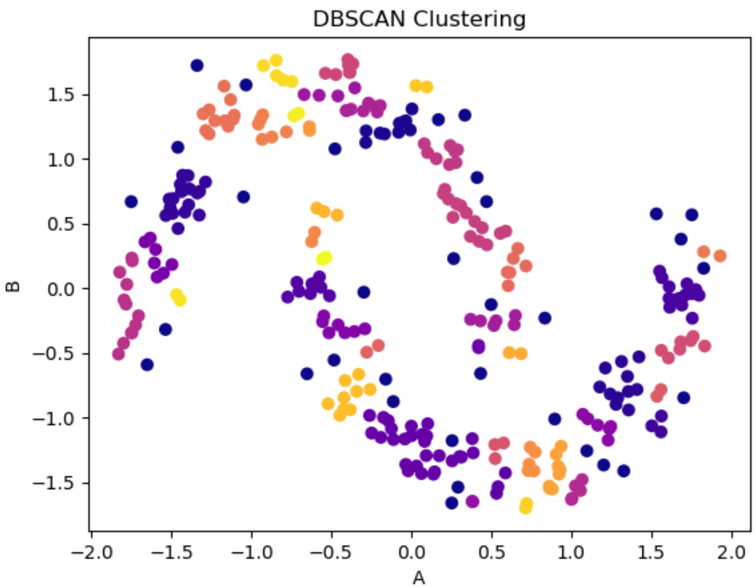


In [41]:
```
#Silhouette Score

from sklearn.metrics import silhouette_score
score = silhouette_score(x_scaled, kmeans.labels_, metric='euclidean')
score
```

Out[41]: 0.4329118241119466

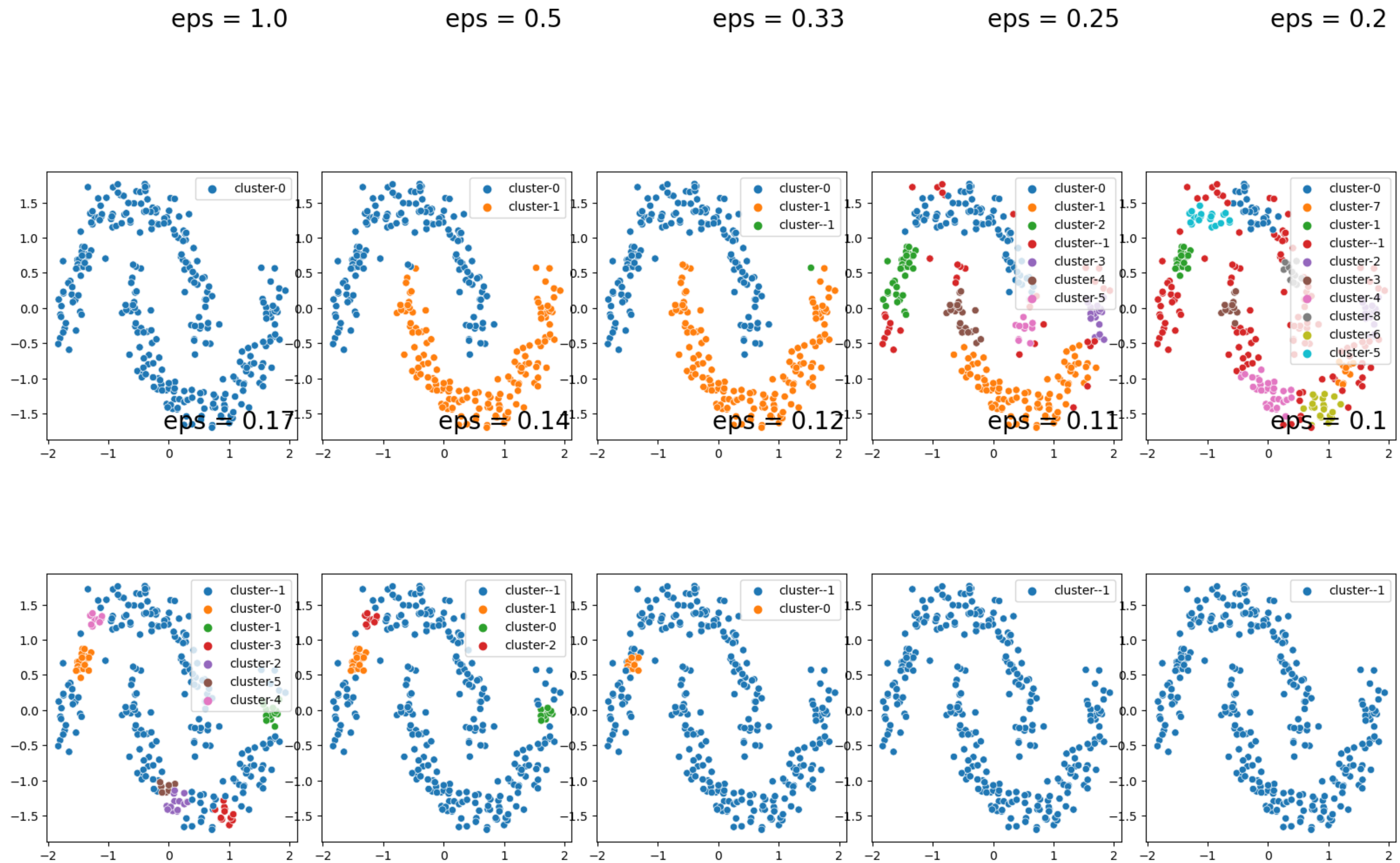
In [42]:
```
#Q2-Perform DBSCAN clustering (use knee locator to help find optimal
#parameter) on the given dataset

from sklearn.cluster import DBSCAN
dbs = DBSCAN(eps=0.123, min_samples=2)
clusters = dbs.fit_predict(x_scaled)
```

In [43]:
```
#Plot DBSCAN Cluster

plt.scatter(x_scaled[:,0],x_scaled[:,1], c=clusters, cmap='plasma')
plt.title("DBSCAN Clustering")
plt.xlabel('A')
plt.ylabel('B')
plt.show()
```



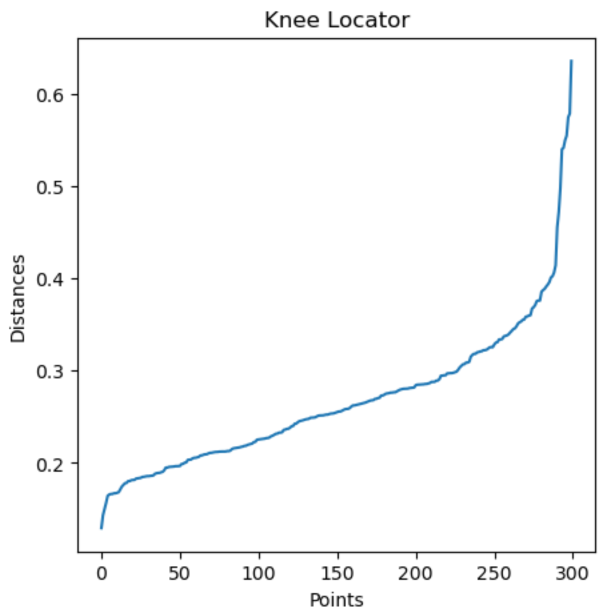
In [44]:

```
#DBSCAN Fine Tuning with Varied eps (ε) (Varied ε = 0.1 -> 1)

fig = plt.figure(figsize=(20,10))
fig.subplots_adjust(hspace=0.5, wspace=.1)

i=1
for x in range(10,0,-1):
    eps = 1/(11-x)
    db = DBSCAN(eps=eps, min_samples=10)
    db.fit(x_scaled)

    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    clusters = db.labels_

    ax = fig.add_subplot(2, 5, i)
    ax.text(1, 4, "eps = {}".format(round(eps, 2)), fontsize=20,ha="center")
    sns.scatterplot(x_scaled[:,0], x_scaled[:,1], hue=["cluster-{}".format(x) for x in clusters])
    i += 1
```


eps = 1.0
eps = 0.5
eps = 0.33
eps = 0.25
eps = 0.2
cluster-0
cluster-0
cluster-1
cluster-0
cluster-1
cluster--1
cluster-0
cluster-1
cluster-2
cluster--1
cluster-3
cluster-4
cluster-5
cluster-0
cluster-7
cluster-1
cluster--1
cluster-2
cluster-3
cluster-4
cluster-8
cluster-6
cluster-5
eps = 0.17
eps = 0.14
eps = 0.12
eps = 0.11
eps = 0.1
cluster--1
cluster-0
cluster-1
cluster-3
cluster-2
cluster-5
cluster-4
cluster--1
cluster-1
cluster-0
cluster-2
cluster--1
cluster-0
cluster--1
cluster--1

In [45]:

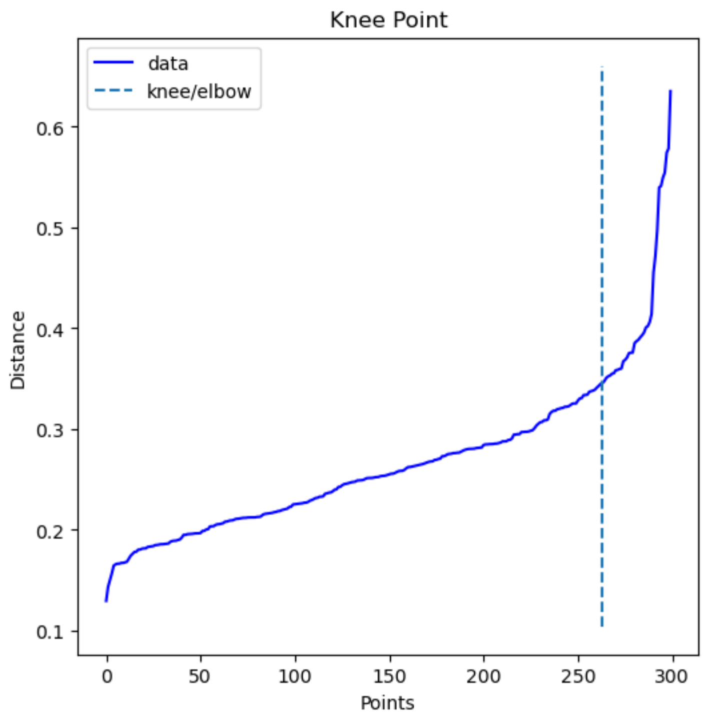
```python
#DBSCAN fine tuning with knee Locator

from sklearn.neighbors import NearestNeighbors
from kneed import KneeLocator

NN = NearestNeighbors(n_neighbors=11)
neighbors = NN.fit(x_scaled)
distances, indices = neighbors.kneighbors(x_scaled)

distances = np.sort(distances[:,10],axis=0)
i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex',
                   direction = 'increasing', interp_method = 'polynomial')

fig = plt.figure(figsize=(5,5))
plt.plot(distances)
plt.title('Knee Locator')
plt.xlabel('Points')
plt.ylabel('Distances')
plt.show()
```



In [46]:
```
#Find optimum epsilon

fig = plt.figure(figsize=(5, 5))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")
#plt.savefig("knee.png", dpi=300)
print(distances[knee.knee])
```

```
0.34567786876134754
```

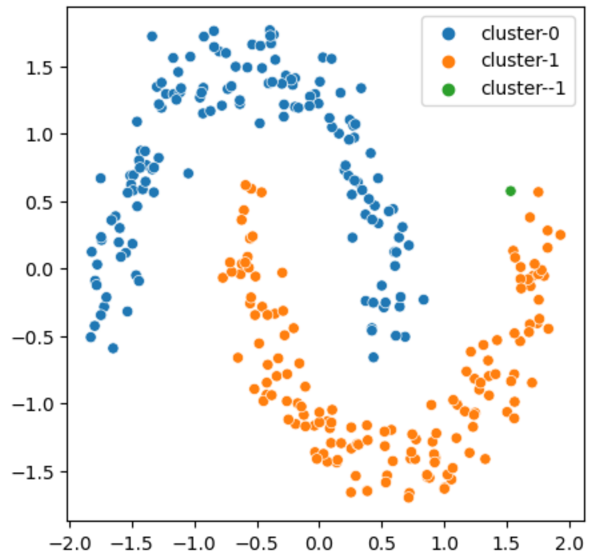
```
<Figure size 500x500 with 0 Axes>
```



In [47]:

```python
db = DBSCAN(eps=distances[knee.knee], min_samples=10).fit(x_scaled)
labels = db.labels_

fig = plt.figure(figsize=(5, 5))
sns.scatterplot(x_scaled[:,0], x_scaled[:,1],
                hue=["cluster-{}".format(x) for x in labels])
```

Out[47]: <AxesSubplot:>



In [49]:

```python
#Score for DBSCAN

from sklearn.metrics import silhouette_score
score = silhouette_score(x_scaled, db.labels_, metric = 'euclidean')
score
```

Out[49]: 0.2144392865052958

```
#Q3

Silhouette Score:
Kmeans = 0.43
DBSCAN = 0.21
K-Means model gives better silhouette score compared to DBSCAN,
so this means Kmeans is a better model than DBSCAN

Kmeans-4 clusters and no outlier
DBSCAN-2 clusters + outlier
```