AIM: IMPLEMENTATION AND ANALYSIS OF CLUSTERING ALGORITHMS LIKE K-MEANS

THEORY:

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is 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 is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.
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.
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.

1) **IMPORTING LIBRARIES:**

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

2) **DATA PREPROCESSING:**

```
data = pd.read_csv('Countrries.csv')
data = data.replace((np.inf, -np.inf, np.nan), 0).reset_index(drop=True)
data
```

	country	latitude	longitude	name
0	AD	42.546245	1.601554	Andorra
1	AE	23.424076	53.847818	United Arab Emirates
2	AF	33.939110	67.709953	Afghanistan
3	AG	17.060816	-61.796428	Antigua and Barbuda
4	AI	18.220554	-63.068615	Anguilla
...	...	...	...	...
240	YE	15.552727	48.516388	Yemen
241	YT	-12.827500	45.166244	Mayotte
242	ZA	-30.559482	22.937506	South Africa
243	ZM	-13.133897	27.849332	Zambia
244	ZW	-19.015438	29.154857	Zimbabwe

245 rows × 4 columns

3) PLOTTING GRAPH:

```
plt.scatter(data['longitude'],data['latitude'])
plt.xlim(-180,180)
plt.ylim(-90,90)
plt.show()
```

4) DATA SPLITTING INTO TRAINING DATASET & TESTING DATASET & CREATING CLUSTERS:

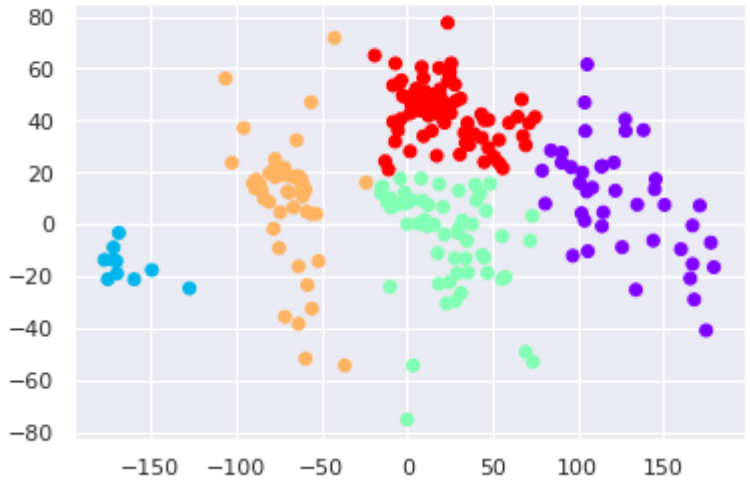
```
x = data.iloc[:,1:3]
kmeans = KMeans(5)
kmeans.fit(x)
identified_clusters = kmeans.fit_predict(x)
identified_clusters
```

```
array([2, 2, 2, 1, 1, 2, 2, 1, 4, 4, 1, 3, 2, 0, 1, 2, 2, 1, 0, 2, 4, 2,
       2, 4, 4, 1, 0, 1, 1, 1, 0, 4, 4, 2, 1, 1, 0, 4, 4, 4, 2, 4, 3, 1,
       4, 0, 1, 1, 1, 1, 0, 2, 2, 2, 4, 2, 1, 1, 2, 1, 2, 2, 2, 4, 2, 4,
       2, 0, 1, 0, 2, 2, 4, 2, 1, 2, 1, 2, 4, 2, 1, 4, 4, 1, 4, 2, 1, 1,
       0, 4, 1, 2, 0, 4, 1, 2, 1, 2, 0, 2, 2, 2, 0, 4, 2, 2, 2, 2, 2, 1,
       2, 0, 4, 2, 0, 3, 4, 1, 0, 0, 2, 1, 2, 0, 2, 1, 2, 0, 4, 4, 2, 2,
       2, 2, 2, 2, 2, 2, 4, 0, 2, 4, 0, 0, 0, 0, 1, 2, 1, 2, 4, 0, 4, 1,
       0, 4, 4, 0, 4, 0, 4, 1, 2, 2, 0, 0, 3, 0, 2, 1, 1, 3, 0, 0, 2, 2,
       1, 3, 1, 2, 2, 0, 1, 2, 4, 2, 2, 0, 4, 2, 0, 4, 4, 2, 0, 4, 2, 2,
       2, 4, 2, 4, 4, 1, 4, 1, 2, 4, 1, 4, 4, 4, 0, 2, 3, 0, 2, 2, 3, 2,
       1, 0, 0, 4, 2, 4, 4, 1, 1, 2, 2, 1, 1, 1, 1, 0, 0, 3, 3, 2, 4, 4,
       4, 4, 4], dtype=int32)
```

5) PLOTTING CLUSTERS:

```
data_with_clusters = data.copy()
data_with_clusters['clusters'] = identified_clusters
plt.scatter(data_with_clusters['longitude'],data_with_clusters['latitude'],c=data_with_cluste
rs['clusters'],cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x7fa01d1aa110>
```



CONCLUSION:

From this practical, I have learned the implementation of k-means in python.

AIM: IMPLEMENTATION AND ANALYSIS OF CLUSTERING ALGORITHMS LIKE K-MEANS

THEORY:

K-medoids clustering is a variant of K-means that is more robust to noises and outliers. Instead of using the mean point as the center of a cluster, K medoids uses an actual point in the cluster to represent it. Medoid is the most centrally located object of the cluster, with minimum sum of distances to other points.

The group of points in the right form a cluster, while the rightmost point is an outlier. Mean is greatly influenced by the outlier and thus cannot represent the correct cluster center, while medoid is robust to the outlier and correctly represents the cluster center. It is a clustering algorithm resembling the K-Means clustering technique. It falls under the category of unsupervised machine learning. It majorly differs from the K-Means algorithm in terms of the way it selects the clusters' centres. The former selects the average of a cluster's points as its centre (which may or may not be one of the data points) while the latter always picks the actual data points from the clusters as their centres (also known as 'exemplars' or 'medoids'). K-Medoids also differ in this respect from the K-Medians algorithm which is the same as K-means, except that it chooses the medians (instead of means) of the clusters as centres

SOURCE CODE:

1) **IMPORTING LIBRARIES & READING DATASET:**

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

    df = pd.read_csv('breast-cancer-wisconsin.data')
    df.head()
```

	1000025	5	1	1.1	1.2	2	1.3	3	1.4	1.5	2.1
0	1002945	5	4	4	5	7	10	3	2	1	2
1	1015425	3	1	1	1	2	2	3	1	1	2
2	1016277	6	8	8	1	3	4	3	7	1	2
3	1017023	4	1	1	3	2	1	3	1	1	2
4	1017122	8	10	10	8	7	10	9	7	1	4

2) DATA PREPROCESSING:

```
dummy = pd.DataFrame(df.columns).T
for col in dummy.columns:
    dummy[col]=dummy[col].astype(float)
    dummy[col]=dummy[col].astype(int)

df.columns=range(0,df.shape[1])
df[6] = df[6].str.extract('(\d+)').astype(float)
for col in df.columns:
    df[col]=df[col].astype(float)
df = pd.concat([dummy, df],axis=0)
df.reset_index(inplace=True, drop=True)

column_names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromati',
                'Normal Nucleoli', 'Mitoses', 'Class']

df.columns=column_names
df['Class'] = df['Class'].replace(to_replace={2:1, 4:0})
df.tail()
```

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromati	Normal Nucleoli	Mitoses	Class
694	776715.0	3.0	1.0	1.0	1.0	3.0	2.0	1.0	1.0	1.0	1.0
695	841769.0	2.0	1.0	1.0	1.0	2.0	1.0	1.0	1.0	1.0	1.0
696	888820.0	5.0	10.0	10.0	3.0	7.0	3.0	8.0	10.0	2.0	0.0
697	897471.0	4.0	8.0	6.0	4.0	3.0	4.0	10.0	6.0	1.0	0.0
698	897471.0	4.0	8.0	8.0	5.0	4.0	5.0	10.0	4.0	1.0	0.0

3) COUNT OF NULL VALUES:

```
df.isnull().sum()
```

```
Sample code number            0
Clump Thickness               0
Uniformity of Cell Size       0
Uniformity of Cell Shape      0
Marginal Adhesion             0
Single Epithelial Cell Size   0
Bare Nuclei                  16
Bland Chromati                0
Normal Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

4) DROPPING NULL VALUES:

```
df = df.dropna(axis=0)
df.isnull().sum()
```

```
Sample code number            0
Clump Thickness               0
Uniformity of Cell Size       0
Uniformity of Cell Shape      0
Marginal Adhesion             0
Single Epithelial Cell Size   0
Bare Nuclei                   0
Bland Chromati                0
Normal Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

5) NORMALIZATION:

```
[8] from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import MinMaxScaler

    X = df.iloc[:,:-1].values
    y = df.iloc[:,-1].values

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

    scaler = MinMaxScaler()

    X_train = scaler.fit_transform(X_train)
    X_test  = scaler.transform(X_test)
```

6) INSTALLING & IMPORTING LIBRARIES FOR KMEDIOD:

```
!pip install scikit-learn-extra
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.2.0-cp37-cp37m-manylinux2010_x86_64.whl (1.7 MB)
     |████████████████████████████████| 1.7 MB 3.8 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.21.6)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.0.2)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.7/dist-packages (from scikit-learn-extra) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.1.0)
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.2.0
```

7) BUILDING MODEL OF K-MEDIOD & PREDICTION OF BREAST CANCER:

```
0s
from sklearn_extra.cluster import KMedoids

kmedoids = KMedoids(n_clusters=2, random_state=0).fit(X_train)
y_pred = kmedoids.predict(X_test)
y_pred
```

```
array([1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 1])
```

8) CONFUSION MATRIX:

```
0s
[20] from sklearn.metrics import classification_report,confusion_matrix
     print(confusion_matrix(y_test, y_pred))
```

```
[[43  7]
 [ 1 86]]
```

CONCLUSION:

From this practical, I have learned the implementation of k-medoid in python.