# Experiment 06 - Clustering using Python

| | |
|---|---|
| Roll No. | 19 |
| Name | Manav Jawrani |
| Class | D15A |
| Subject | Business Intelligence Lab |
| LO Mapped | LO2: Organize and prepare the data needed for data mining algorithms in terms of attributes and class inputs, training, validating, and testing files.<br>LO4: Implement various data mining algorithms from scratch using languages like Python/ Java etc. |
| Grade | |

DATE:

Aim- To implement the ~~...~~ clustering algorithm using python.

Theory-

- K-means clustering-
K-means is a partition-based clustering algorithm that aims to partition a given dataset into K clusters, where K is a user-specified number of clusters. The algorithm works by iteratively assigning data points to the nearest cluster center and updating clusters centers based on the mean of the points assigned to each cluster.

- Hierarchical clustering-
Hierarchical clustering is a method of clustering that builds a hierarchy of clusters by recursively merging or dividing them. There are two main types: agglomerative and divisive. Agglomerative clustering starts with each data point in its own cluster and recursively merges the most similar clusters until a single cluster is formed. Divisive on other hand, starts with a single cluster containing all data points and recursively splits into smaller clusters until each point is in it own cluster.

DATE:

- DBSCAN Clustering -
DBSCAN is a density-based clustering algorithm that groups together points that are close to each other based on a specified distance metric. This algorithm works by defining a radius around each point and useful for identifying clusters of arbitrary shape.

- Python Library function used -

1. matplotlib.pyplot - A plotting library used to create visualizations such as line charts.
2. numpy - A library for numerical computing.
3. pandas - A library for data manipulation and analysis, to create and manipulate data frames.
4. seaborn - A library for data visualization, used to create statistical graphics.
5. scipy.cluster.hierarchy - A library for hierarchical clustering.
6. sklearn.cluster.DBSCAN - A clustering algorithm that groups together points that are close to each other.
7. sklearn.cluster.Kmeans - A clustering algorithm that partitions a dataset into K clusters.
8. sklearn.metrics.silhouette_score - A metric for measuring the quality of clustering solution based on how well-defined the clusters are.
9. sklearn.model_selection.train_test_split - A function for splitting a dataset.

DATE:

10. Sklearn.preprocessing.Standardscaler - A class for Standardizing data by removing the mean and scaling to unit variance.

11. Sklearn.preprocessing.preprocessing - A module containing functions for scaling and encoding.

• Observation+conclusion -

1. K-means clustering resulted in a distance of 0.554. This suggests that the data points in the data set may be relatively close to each other and therefore, can be grouped into clusters based on their proximity.

2. Agglomerative clustering resulted in dendrogram. A dendrogram is a visual representation of clustering process, which shows how the clusters are formed by merging individual data points.

3. DBSCAN resulted in 1 cluster, this suggests DBSCAN may have identified a single dense region in the data set and classified all data points within that region as belonging to same cluster.

# Implementation:

```
[1] from google.colab import files
    uploaded = files.upload()
```

Choose Files healthcare-...ke-data.csv

- **healthcare-dataset-stroke-data.csv**(text/csv) - 316971 bytes, last modified: 3/29/2023 - 100% done

```
Saving healthcare-dataset-stroke-data.csv to healthcare-dataset-stroke-data.csv
```

## KMeans

```
[2] import pandas as pd
    dataset = pd.read_csv("healthcare-dataset-stroke-data.csv")
    dataset.describe()
```

| | id | age | hypertension | heart_disease | avg_glucose_level | bmi | stroke |
|---|---|---|---|---|---|---|---|
| **count** | 5110.000000 | 5110.000000 | 5110.000000 | 5110.000000 | 5110.000000 | 4909.000000 | 5110.000000 |
| **mean** | 36517.829354 | 43.226614 | 0.097456 | 0.054012 | 106.147677 | 28.893237 | 0.048728 |
| **std** | 21161.721625 | 22.612647 | 0.296607 | 0.226063 | 45.283560 | 7.854067 | 0.215320 |
| **min** | 67.000000 | 0.080000 | 0.000000 | 0.000000 | 55.120000 | 10.300000 | 0.000000 |
| **25%** | 17741.250000 | 25.000000 | 0.000000 | 0.000000 | 77.245000 | 23.500000 | 0.000000 |
| **50%** | 36932.000000 | 45.000000 | 0.000000 | 0.000000 | 91.885000 | 28.100000 | 0.000000 |
| **75%** | 54682.000000 | 61.000000 | 0.000000 | 0.000000 | 114.090000 | 33.100000 | 0.000000 |
| **max** | 72940.000000 | 82.000000 | 1.000000 | 1.000000 | 271.740000 | 97.600000 | 1.000000 |

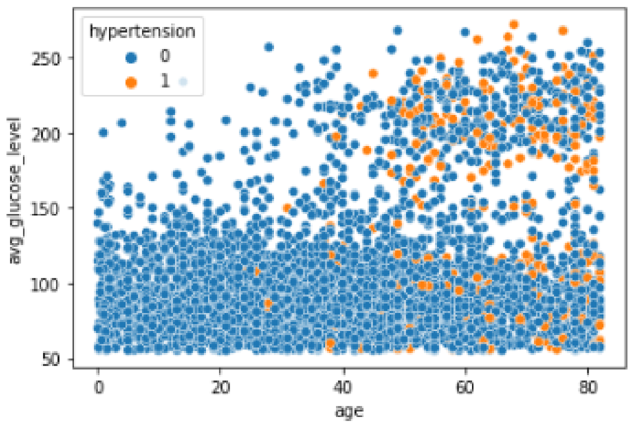
```
[3] dataset = dataset.dropna(subset=['bmi'])
```

```
[4] dataset.shape
```

```
(4909, 12)
```

```
[5] import seaborn as sns

sns.scatterplot(data = dataset, x = 'age', y = 'avg_glucose_level', hue = 'hypertension')
```

```
<Axes: xlabel='age', ylabel='avg_glucose_level'>
```




```
[6] # Split dataset into training set and test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dataset[['age', 'avg_glucose_level']], dataset[['bmi']], test_size=0.33, random_state=0)
```

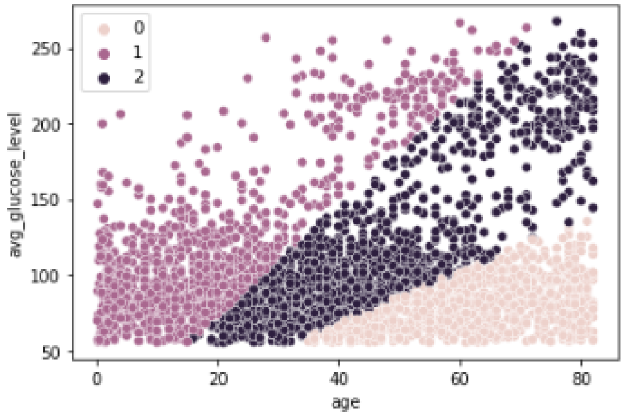
```
[7] from sklearn import preprocessing

X_train_norm = preprocessing.normalize(X_train)
X_test_norm = preprocessing.normalize(X_test)
```

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters = 3, random_state = 0, n_init='auto')
kmeans.fit(X_train_norm)
```

```
KMeans
KMeans(n_clusters=3, n_init='auto', random_state=0)
```

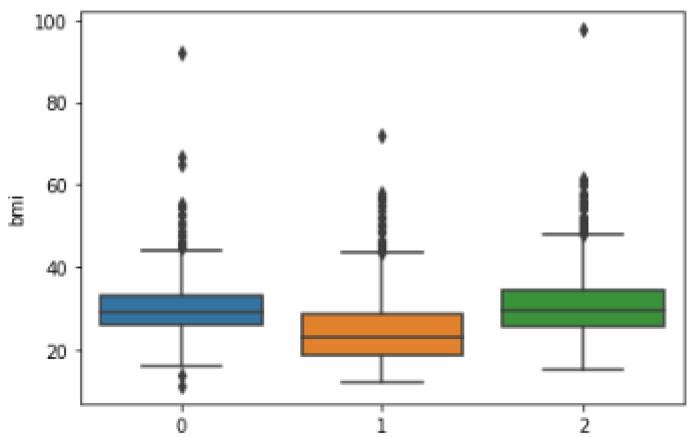
```
sns.scatterplot(data = X_train, x = 'age', y = 'avg_glucose_level', hue = kmeans.labels_)
```

<Axes: xlabel='age', ylabel='avg_glucose_level'>




```
[13] sns.boxplot(x = kmeans.labels_, y = y_train['bmi'])
```

<Axes: ylabel='bmi'>



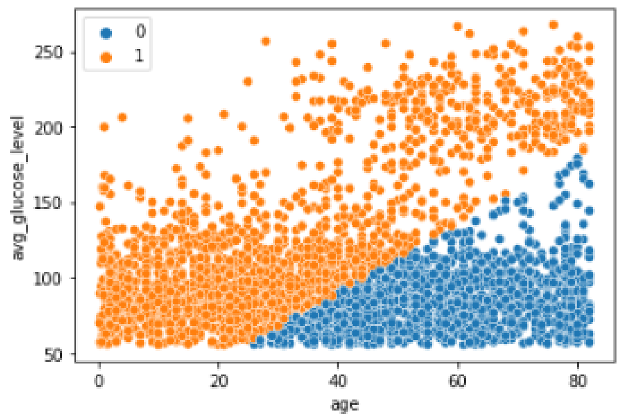

```
[11] from sklearn.metrics import silhouette_score

     silhouette_score(X_train_norm, kmeans.labels_, metric='euclidean')
```

0.553153558375945

```
K = range(2, 8)
fits = []
score = []


for k in K:
    # train the model for current value of k on training data
    model = KMeans(n_clusters = k, random_state = 0, n_init='auto').fit(X_train_norm)

    # append the model to fits
    fits.append(model)

    # Append the silhouette score to scores
    score.append(silhouette_score(X_train_norm, model.labels_, metric='euclidean'))
```

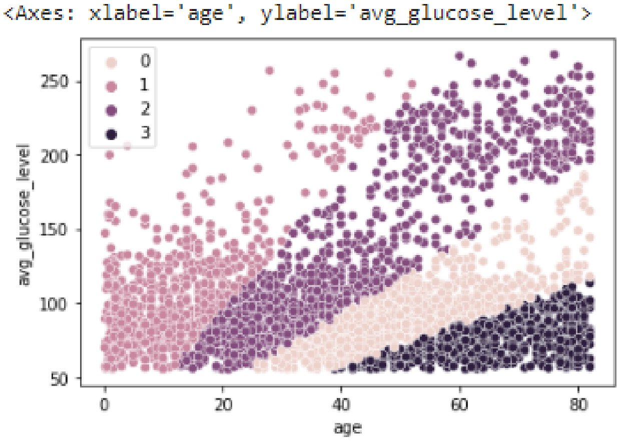
```
sns.scatterplot(data = X_train, x = 'age', y = 'avg_glucose_level', hue = fits[0].labels_)
```

```
<Axes: xlabel='age', ylabel='avg_glucose_level'>
```



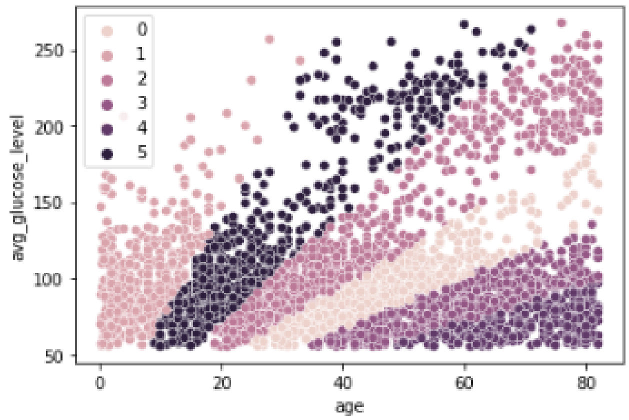

```
sns.scatterplot(data = X_train, x = 'age', y = 'avg_glucose_level', hue = fits[2].labels_)
```

```
<Axes: xlabel='age', ylabel='avg_glucose_level'>
```



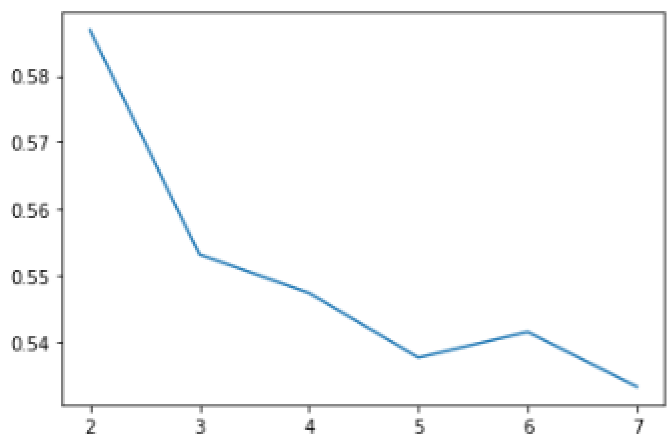
```
sns.scatterplot(data = X_train, x = 'age', y = 'avg_glucose_level', hue = fits[4].labels_)
```

```
<Axes: xlabel='age', ylabel='avg_glucose_level'>
```



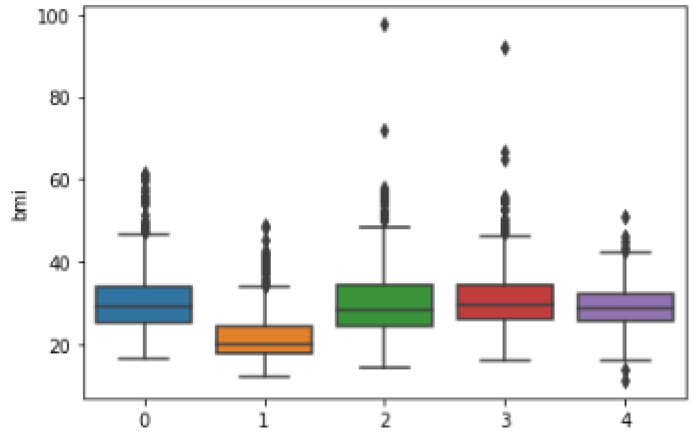

```
sns.lineplot(x = K, y = score)
```

```
<Axes: >
```



```
[17] sns.boxplot(x = fits[3].labels_, y = y_train['bmi'])
```

```
<Axes: ylabel='bmi'>
```

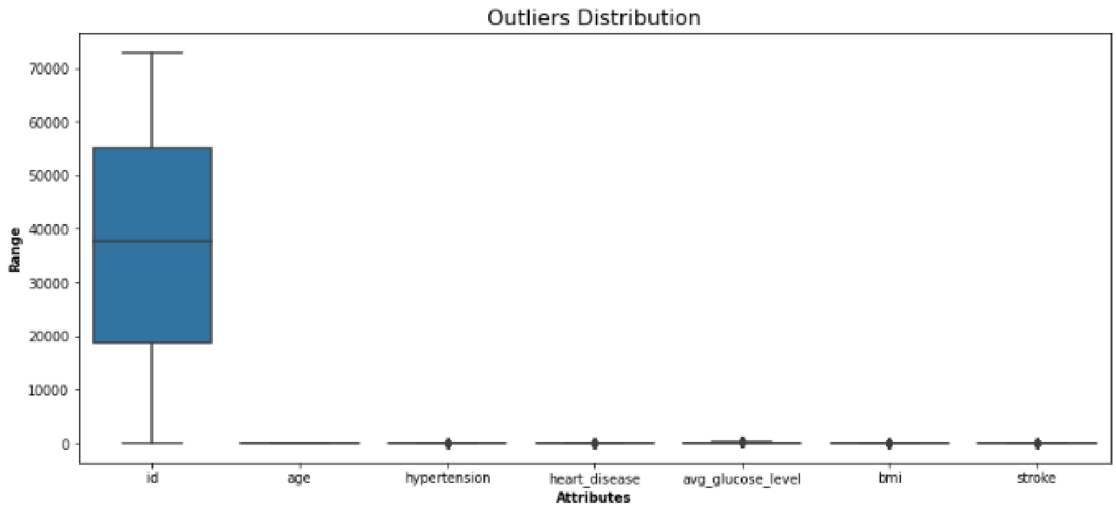



## Hierarchical

```
percent_missing =round(100*(dataset.isnull().sum())/len(dataset),2)
percent_missing
```

```
id                   0.0
gender               0.0
age                  0.0
hypertension         0.0
heart_disease        0.0
ever_married         0.0
work_type            0.0
Residence_type       0.0
avg_glucose_level    0.0
bmi                  0.0
smoking_status       0.0
stroke               0.0
dtype: float64
```

```python
import matplotlib.pyplot as plt
def show_boxplot(df):
  plt.rcParams['figure.figsize'] = [14,6]
  sns.boxplot(data = df, orient="v")
  plt.title("Outliers Distribution", fontsize = 16)
  plt.ylabel("Range", fontweight = 'bold')
  plt.xlabel("Attributes", fontweight = 'bold')

show_boxplot(dataset)
```



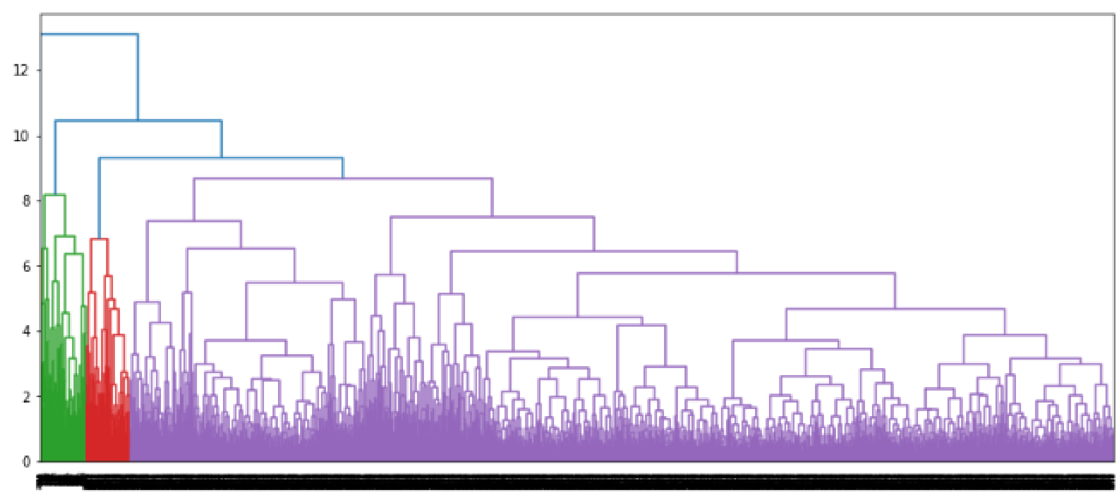
```
[20] dataset=dataset.drop(['gender','ever_married','work_type','Residence_type','smoking_status'], axis=1)
```

```
from sklearn.preprocessing import StandardScaler

data_scaler = StandardScaler()

scaled_data= data_scaler.fit_transform(dataset)
scaled_data.shape
```

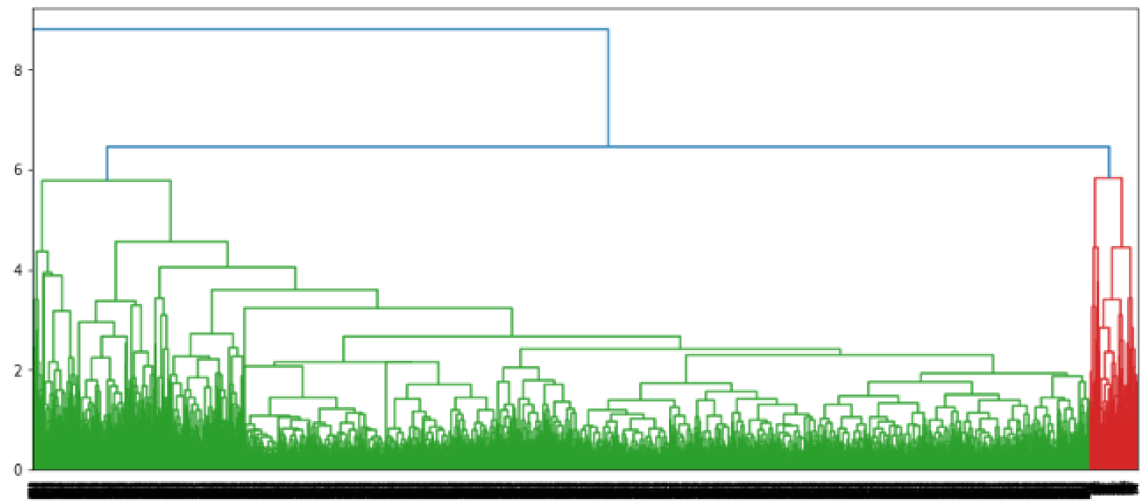
```
(4909, 7)
```

```
[22] from scipy.cluster.hierarchy import linkage, dendrogram

complete_clustering = linkage(scaled_data, method="complete", metric="euclidean")
average_clustering = linkage(scaled_data, method="average", metric="euclidean")
single_clustering = linkage(scaled_data, method="single", metric="euclidean")
```

```
[23] dendrogram(complete_clustering)
plt.show()
```



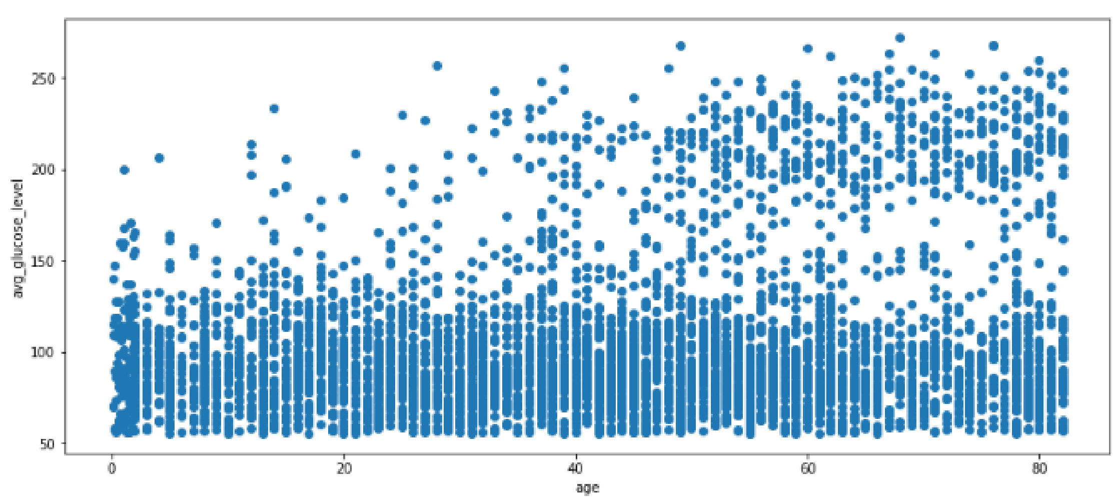
```
[24] dendrogram(average_clustering)
     plt.show()
```




## DBScan

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
x = dataset['age']
y = dataset['avg_glucose_level']

plt.scatter(x,y)
plt.xlabel("age")
plt.ylabel("avg_glucose_level")
plt.show()
```



```
[27] dataset = dataset[["age", "avg_glucose_level"]]
     dataset = dataset.to_numpy().astype("float32", copy = False)
```

```
stscaler = StandardScaler().fit(dataset)
dataset = stscaler.transform(dataset)
```

```
[29] from sklearn.cluster import DBSCAN
     dbsc = DBSCAN(eps = .5, min_samples = 15).fit(dataset)
```

```
[30] import numpy as np
     labels = dbsc.labels_
     core_samples = np.zeros_like(labels, dtype = bool)
     core_samples[dbsc.core_sample_indices_] = True
```

```
dbsc = DBSCAN(eps=0.3, min_samples=10).fit(dataset)
labels = dbsc.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

print("Estimated number of clusters: %d" % n_clusters_)
```

```
Estimated number of clusters: 1
```