

Diabeties Data

this file contains 3 Classification & 3 Clustring methods based on the Diabeties Data, the steps are as follows

1. Importing (calling) libraries and methods
2. Initilizing Functions
3. Loading Diabities Dataset
4. Preprocessing the data
5. Spiliting Data
6. Classification
 - K Nearest Neighbors
 - SVM: Support Vector Machine
 - NN: Nureal Network
7. Clustring
 - Heretical Clustering
 - KMeans
 - DBSCAN
8. conclusion

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

from sklearn.impute import KNNImputer
from sklearn.preprocessing import RobustScaler, MinMaxScaler
from sklearn.model_selection import train_test_split

# Clasification Libraries
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from sklearn.metrics import classification_report, confusion_matrix
# Clustring Libraries
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
```

In Python, you can turn off user warnings by using the warnings module and setting the appropriate warning filter. To disable all user warnings, you can use the following code:

```
import warnings

warnings.filterwarnings('ignore', category=UserWarning)
```

This code imports the warnings module and sets a filter to ignore any warnings of the UserWarning category. Keep in mind that ignoring all user warnings might hide important information, so it's generally better to address the underlying issues that cause the warnings instead of suppressing them entirely.

```
'''
import warnings
warnings.filterwarnings('ignore', category=UserWarning)
'''
```

2.Initializing Functions

- **Show function:**

The function is to demonstrate the graphs using matplotlib, it first sets the x-axis and y-axis labels to "x1" and "x2". It then sets the title of the plot to the value of the title argument. Finally, it creates the scatter plot using the `plt.scatter()` function, passing in the first and second columns of X as the x and y coordinates, respectively. The `c` parameter is set to the label argument, which colors each data point according to its label. The `s` parameter sets the size of the data points.

- **accuracy function :**

The function calculates the accuracy of the classifier using the score method of the `classification_method` object. The score method returns the mean accuracy of the classifier on the test dataset. Next, the function calculates the confusion matrix and classification report using the `confusion_matrix` and `classification_report` functions from the sklearn metrics module. The confusion matrix is a table that shows the number of true positive, true negative, false positive, and false negative predictions made by the classifier. The classification report shows the precision, recall, and F1-score for each class in the dataset is to gives the user feedback on thier Classification methods by thier test set of data.

- **C_accuracy function :**

The function that calculates and prints the accuracy of a given set of predictions for Cluster methods. The function first calculates the number of correct predictions by counting the number of indices where the predicted label matches the true label. It does this using the `np.count_nonzero` function from the NumPy library. the function calculates the accuracy by dividing the number of correct predictions by the total number of predictions and multiplying by 100 to convert to a percentage. this is to gives the user feedback on thier Clustring methods by thier test set of data.

- **highlight_min function :**

This function uses a pandas DataFrame in order to highlight the minimum value in a given series `s` with the provided properties `props`.

```
def show(title,X,Y,label,xlabel,ylabel):
    if title != "Diabetes sample view":
        plt.figure(figsize=(4,4))
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)

        # displaying the title
        plt.title(title)

        # display the graph
        plt.scatter(X, Y, c=label, s=2)

def accuracy (classification_method, method_pred,X_test,Y_test):
    # Print the accuracy of the classifier
    print("Accuracy:", classification_method.score(X_test,Y_test))

    # Print the confusion matrix and classification report
    print("\nConfusion Matrix:")
    print(confusion_matrix(Y_test, method_pred)) #A confusion matrix
    is a table that is used to define the performance of a classification
    algorithm.

    print("\nClassification Report:")
    print(classification_report(Y_test, method_pred))

def C_accuracy(Pred,Y_test):
    no = np.count_nonzero(Pred == Y_test)
    l = len(Pred)
    print(f"Accuracy :{(no/l)*100}")

def highlight_min(s, props=''):
    return np.where(s == np.nanmin(s.values), props, '')
```

3.Loading Dataset

Ploting the Diabeties Dataset with and witout labels

```
pd.set_option('display.max_rows', 20)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

```
pd.options.mode.chained_assignment = None
plt.rcParams["figure.figsize"] = [8, 7]
plt.rcParams["figure.autolayout"] = True
```

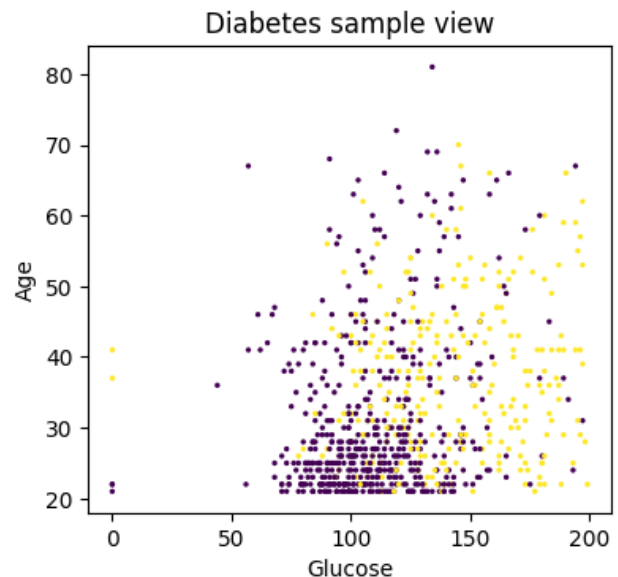
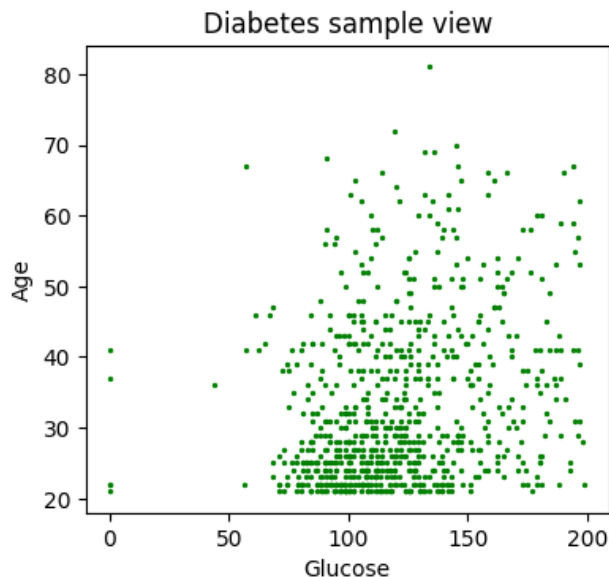
```
# Define the CSV file path
csv_file = "Diabetes.csv"
```

```
# Read the CSV file into a pandas DataFrame
dataset = pd.read_csv(csv_file)
```

```
# Print the DataFrame
dataset.head()
```

	pregnancies	glucose	bloodpressure	skinthickness	insulin	bmi
diabetespedigreefunction	age	outcome				
0	6	148	72	35	0	33.6
0.627	50	True				
1	1	85	66	29	0	26.6
0.351	31	False				
2	8	183	64	0	0	23.3
0.672	32	True				
3	1	89	66	23	94	28.1
0.167	21	False				
4	0	137	40	35	168	43.1
2.288	33	True				

```
plt.subplot(2,2,1)
show(title="Diabetes sample view", X= dataset.glucose, Y= dataset.age,
label="green" ,xlabel="Glucose",ylabel="Age")
plt.subplot(2,2,2)
show(title="Diabetes sample view", X= dataset.glucose, Y= dataset.age,
label=dataset.outcome ,xlabel="Glucose",ylabel="Age")
```



4.preprocessing Dataset

- *checking for redundant and irrelevant values*
- *clean these values and replace None type value*
- *replace the value with the value of the nearest neighbours (knn)*

plotting the minimum in every column to see if there are zero/NaN

```
dataset.T.style.apply(highlight_min, props='color:Black;background-color:Grey', axis=0)
```

```
<pandas.io.formats.style.Styler at 0x7f5467602350>
```

the cell below firstly replaces zero values with NaN in "glucose", "bloodpressure", "skinthickness", "insulin", and "bmi" columns then display the number of NaN values in each column before and after the replacement.

```
dataset[["glucose", "bloodpressure", "skinthickness", "insulin", "bmi"]] =  
dataset[["glucose", "bloodpressure", "skinthickness", "insulin", "bmi"]].r  
eplace(0, np.NaN)  
dataset.isnull().sum()
```

pregnancies	0
glucose	5
bloodpressure	35
skinthickness	227
insulin	374
bmi	11
diabetespedigreefunction	0
age	0
outcome	0

dtype: int64

The cell below applies the RobustScaler from `scikit-learn` to the dataset DataFrame and stores the resulting scaled data in a new DataFrame `C_dataset`. The RobustScaler scales the data by removing the median and scaling the data according to the Interquartile Range (IQR).

```
scaler = RobustScaler()  
C_dataset = pd.DataFrame(scaler.fit_transform(dataset),  
columns=dataset.columns)
```

plotting the minimum in every column to see if there are still zero values

```
imputer = KNNImputer(n_neighbors=10, weights='distance')  
C_dataset = pd.DataFrame(imputer.fit_transform(dataset),
```

```
columns=dataset.columns)
C_dataset.T.style.apply(highlight_min, props='color:Black;background-
color:Grey', axis=0)
```

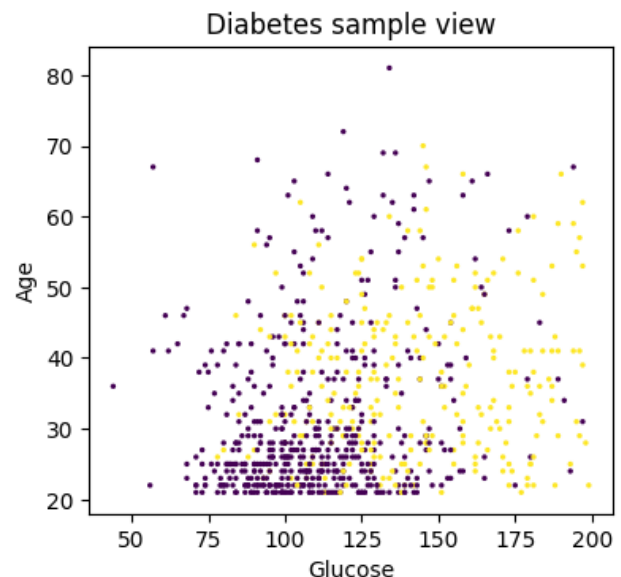
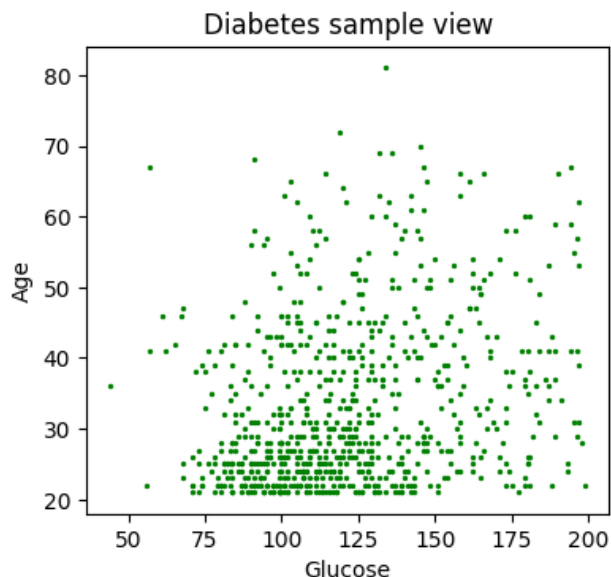
```
<pandas.io.formats.style.Styler at 0x7f54676b3510>
```

the cell below is making sure there is no more NaN values.

```
C_dataset.isnull().sum()
```

```
pregnancies      0
glucose          0
bloodpressure    0
skinthickness    0
insulin          0
bmi             0
diabetespedigreefunction  0
age             0
outcome         0
dtype: int64
```

```
plt.subplot(2,2,1)
show(title="Diabetes sample view", X= dataset.glucose, Y= dataset.age,
label="green" ,xlabel="Glucose",ylabel="Age")
plt.subplot(2,2,2)
show(title="Diabetes sample view", X= dataset.glucose, Y= dataset.age,
label=dataset.outcome ,xlabel="Glucose",ylabel="Age")
```



5. Split the data into train and test sets

the following code splits dataset into training and testing sets using the `train_test_split` function from `scikit-learn` library. it splits the data into 75% training dataset and 25% testing dataset.

```
X = C_dataset.iloc[:, :-1]
y = C_dataset.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)
```

6. Classification

6.1. KNN: K Nearest Neighbour

This cell trains a k-nearest neighbours (KNN) classifier using the `KNeighborsClass` class from `scikit-learn`, and then uses it to predict the labels the test set. The `KNeighborsClass` class takes a parameter `n_neighbors` which specifies the number of nearest neighbors to use for classification. In this case, `n_neighbors` is set to 16. Also `metric` that classifies which metric method to be used for calculation that is set to `'euclidean'`. In this function there is a possibility to set the weights to uniform (same) for all the neighbours or weighting them by distance. The `fit` method trains the classifier using the training set, and the `predict` method uses the trained classifier to predict the labels of the test set. Finally, The `accuracy` and `show` functions that in order they compute the accuracy of the classifier and visualize the predicted labels.

```
knn = KNeighborsClassifier(n_neighbors=16, weights='uniform')

# Train the classifier using the training set
knn.fit(X_train, y_train)

# Use the trained classifier to predict the labels of the test set
knn_pred = knn.predict(X_test)

accuracy(knn, knn_pred, X_test, y_test)
show(title="KNN", X=X_test.glucose, Y=X_test.age,
label=knn_pred, xlabel="Glucose", ylabel="Age")
```

Accuracy: 0.703125

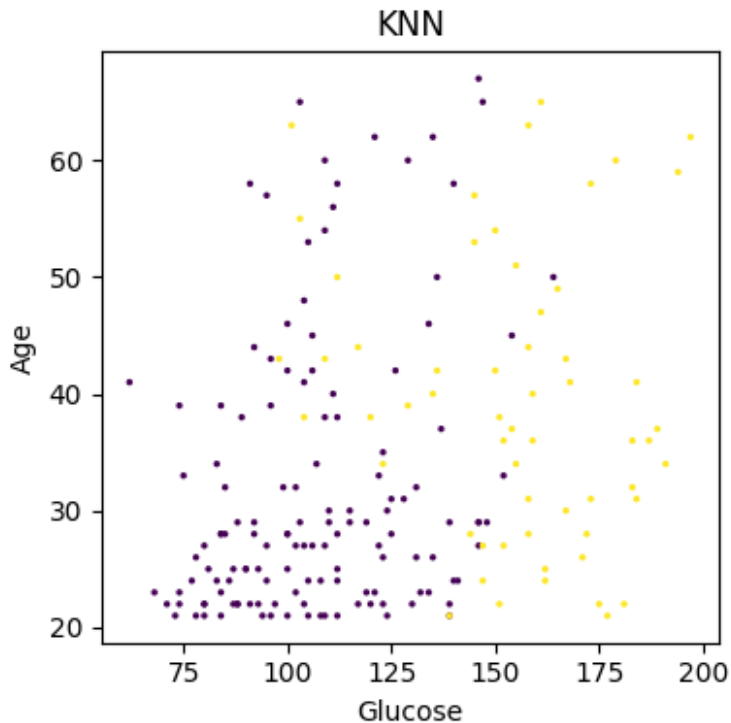
Confusion Matrix:

```
[[100  23]
 [ 34  35]]
```

Classification Report:

	precision	recall	f1-score	support
0.0	0.75	0.81	0.78	123
1.0	0.60	0.51	0.55	69

accuracy			0.70	192
macro avg	0.67	0.66	0.66	192
weighted avg	0.69	0.70	0.70	192



KNN Findings : by adjusting `n_neighbors` starting from 5 the accuracy is set to 68.2% and by raising this value to 14 we get more accuracy nearly 69.7% this value will be the same until reaching `n_neighbors` equal to 16 the accuracy will be 70.3% after 17 every value will decrease the accuracy of the KNN method. the accuracy will drop when the weighting of the points is set to 'distance'. and it's kept to 'uniform' where all the point have the same weight.

6.2.SVM: Support Vector Machine

This cell provides a Support Vector Classifier (SVM) trained with a training set and then used to predict the labels of a test set. The accuracy, confusion matrix, and classification report are then printed and plotted using `accuracy` and `show` functions.

```
svm = SVC(kernel='linear', C=1,probability=True, max_iter=200)

# Train the classifier using the training set
svm.fit(X_train, y_train)

# Use the trained classifier to predict the labels of the test set
svm_pred = svm.predict(X_test)
```



```
accuracy (svm, svm_pred,X_test,y_test)
show(title="NN", X=X_test.glucose, Y=X_test.age,
label=svm_pred,xlabel="Glucose",ylabel="Age")
```

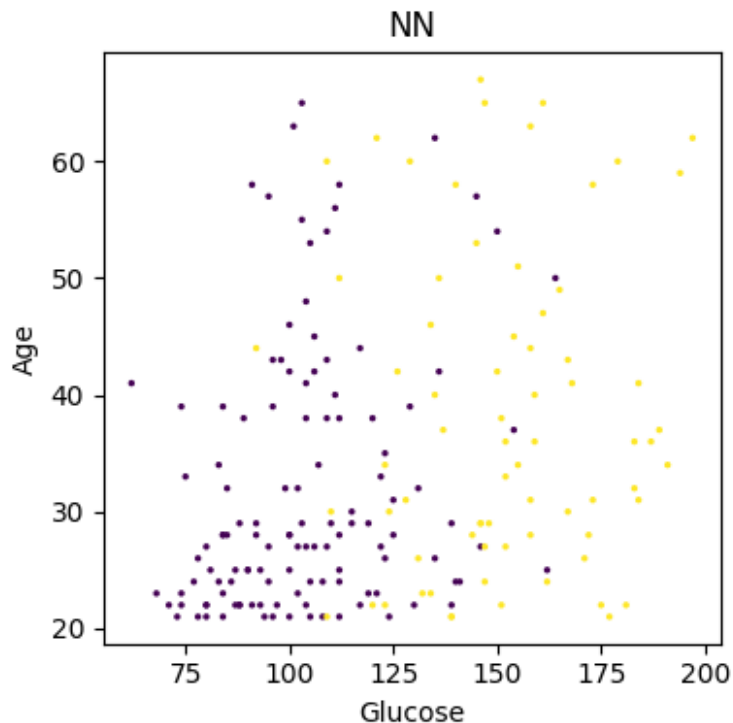
Accuracy: 0.7395833333333334

Confusion Matrix:

```
[[97 26]
 [24 45]]
```

Classification Report:

	precision	recall	f1-score	support
0.0	0.80	0.79	0.80	123
1.0	0.63	0.65	0.64	69
accuracy			0.74	192
macro avg	0.72	0.72	0.72	192
weighted avg	0.74	0.74	0.74	192



SVM findings: by using the kernel of SVM where are choosing how does the algorithm works and by choosing 'linear' as the kernel and setting the maximum iterations equal or above 200 and turning the probability function to True we get the accuracy of 73.9% from this method on the Diabetes data.

6.3.NN: Nureal Network (Multilayer)

This cell will create a neural network, it will train the network on the training data and at the end it will make predictions on the test data and calculate the accuracy of those predictions. Finally, it will display a scatter plot of the test data, with each point colored according to the network's predicted class.

```
NN = MLPClassifier(hidden_layer_sizes=(16,4), activation='identity',
alpha= 0.1 ,max_iter= 2000)
NN.fit(X_train, y_train)
NN_pred = NN.predict(X_test)
```

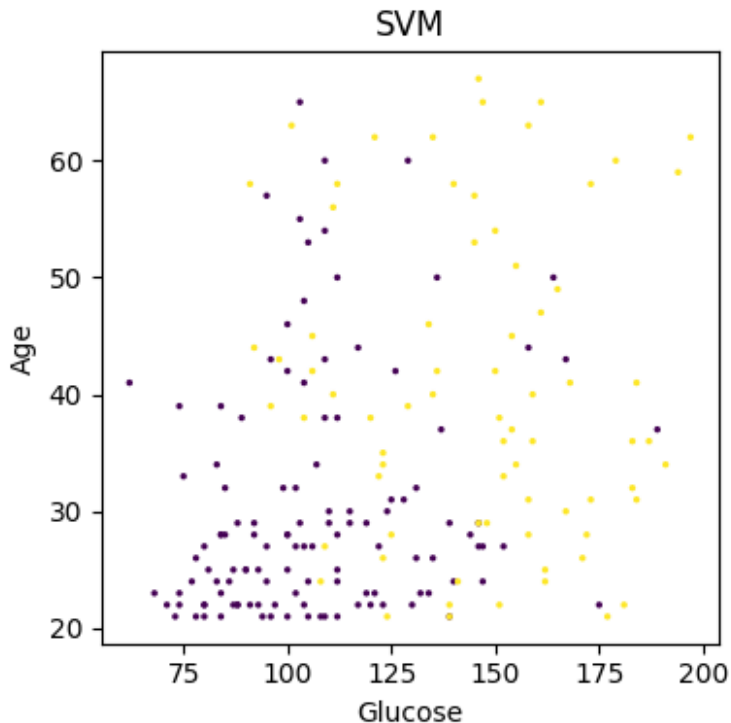
```
accuracy (NN, NN_pred,X_test,y_test)
show(title="SVM", X=X_test.glucose, Y=X_test.age,
label=NN_pred,xlabel="Glucose",ylabel="Age")
```

Accuracy: 0.7083333333333334

Confusion Matrix:
[[93 30]
[26 43]]

Classification Report:

	precision	recall	f1-score	support
0.0	0.78	0.76	0.77	123
1.0	0.59	0.62	0.61	69
accuracy			0.71	192
macro avg	0.69	0.69	0.69	192
weighted avg	0.71	0.71	0.71	192



NN findings: This neural network will adapt an accuracy of over 73.4% with 3 hidden layer of size 16 and 8 and 4 in order , using the rectified linear unit (identity) as activation function. and maximum iterations of 2000, with an learning rate of 0.01.

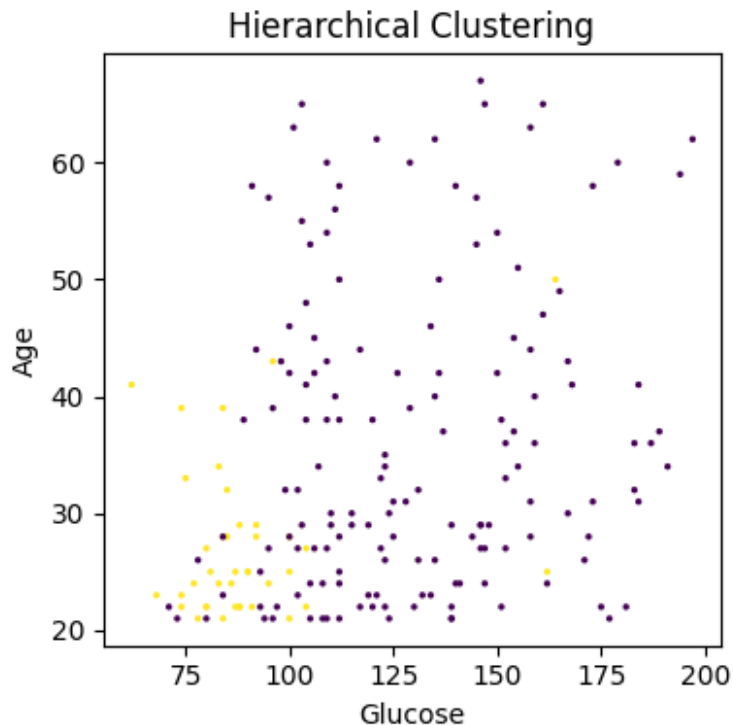
7.Clustering

7.1.Hierarchical Clustering

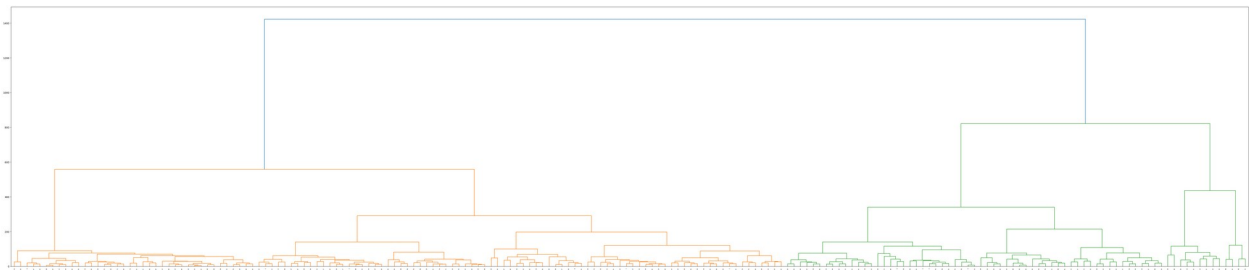
This cell implements a hierarchical clustering using the AgglomerativeClustering algorithm from the sklearn library, like the classifications method these methods also been plotted and their accuracy have been calculated. in this hierarchical clustering part we are having two methods of plotting firstly plotting the points with the labels and the other one the agglomerative tree of choosing the points structure.

```
n_clusters = 2
hierarchical_clustering =
AgglomerativeClustering(n_clusters=n_clusters, metric='cosine',
linkage='average')
hierarchical_clustering.fit(X_train, y_train)
HC_pred = hierarchical_clustering.fit_predict(X_test)
C_accuracy(HC_pred,y_test)
show(title="Hierarchical Clustering", X=X_test.glucose, Y=X_test.age,
label=HC_pred,xlabel="Glucose",ylabel="Age")
```

Accuracy :48.95833333333333



```
plt.figure(figsize=(70, 15))
linkage_data = linkage(X_test, method='ward', metric='euclidean')
dendrogram(linkage_data)
plt.show()
```



hierarchical clustering findings: the highest accuray is set to 68 by using **complete** for linkage and metric to **l1/l2** but by lokking to the visulized data we notice all the point are in the same cluster. this method seems not working as expected on high demintional data.

7.2.Kmeans Clustering

This cell is implementing K-means cluster, The algorithm is fit on the training data and then the fit_predict method is called on the test data to get the cluster predictions. The accuracy of the predictions is then calculated using the C_accuracy function. Finally, the clusters are visualized using the show function and the cluster centers are plotted using matplotlib's scatter function.

```

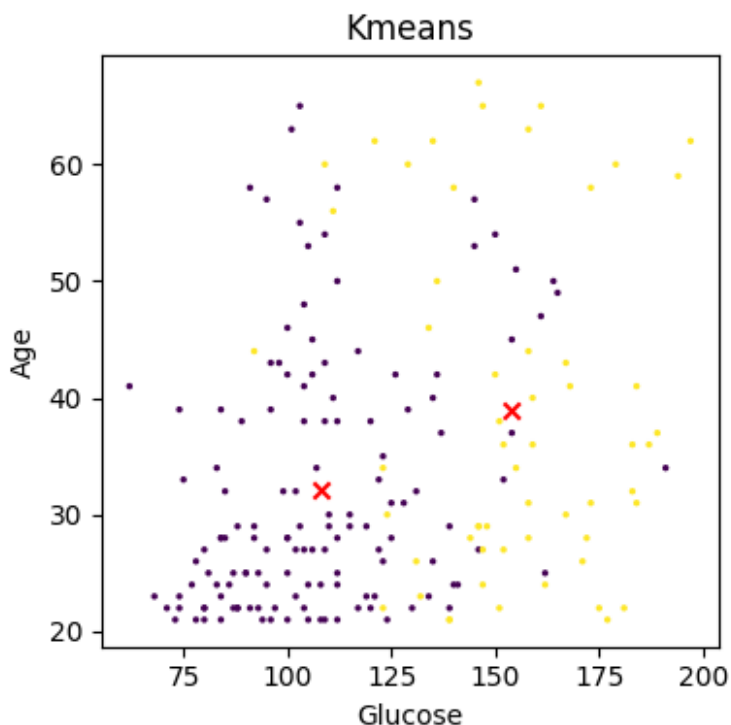
n_clusters = 2
kmeans_clustering = KMeans(n_clusters=n_clusters, n_init=2 , init= "k-
means++", max_iter= 300)
kmeans_clustering.fit(X_train, y_train)
KM_pred = kmeans_clustering.fit_predict(X_test)

C_accuracy(KM_pred,y_test)
show(title="Kmeans", X=X_test.glucose, Y=X_test.age,
label=KM_pred,xlabel="Glucose",ylabel="Age")
plt.scatter(kmeans_clustering.cluster_centers_[0,1],
kmeans_clustering.cluster_centers_[0,7] ,marker='x', color='red')

Accuracy :68.75

<matplotlib.collections.PathCollection at 0x7f5467505590>

```



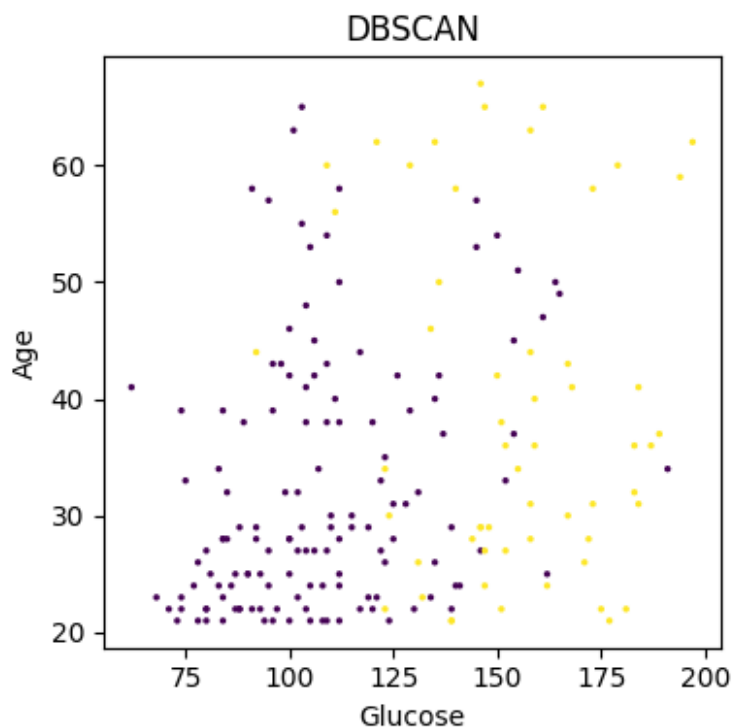
KMeans findings: KMeans clustering with `n_init` parameter to 2, which is the number time the k-means algorithm will be run with different centroid seeds. and by using "k-means++" algorithm that enhances the kMeans, in which it chooses the initial centers in a smart way to speed up convergence. The `max_iter` parameter is set to 300, which is the maximum number of iterations of the k-means algorithm for a single run and by setting these the maximum accuracy of this method on the synthesis data happens with 68.75% happens which is still lower than the classifiers but more accurate than the hierarchical clustering.

7.3.DBSCAN Clustering : Density Based Spatial Clustering

This cell is creating an instance of the DBSCAN clustering algorithm from the sklearn library. and like all others this algorithm is set to be fitted and trained and then uses the C_accuracy and plotting.

```
eps=1
min_samples=5
dbscan_clustering = DBSCAN(eps=eps, min_samples=min_samples)
dbscan_clustering.fit(X_train, y_train)
dbscan_pred = dbscan_clustering.predict(X_test)
C_accuracy(dbscan_pred,y_test)
show(title="DBSCAN", X=X_test.glucose, Y=X_test.age,
label=KM_pred,xlabel="Glucose",ylabel="Age")
```

Accuracy :31.25



DBSCAN findings: by setting the radius of DBSCAN to more than 0.5 in this example its been set to 1 and minimum samples to more than 5 in this example 20 we arrive to the maximum accuracy of 68.75%

8. Conclusion

in this Diabities data the highest accuracy is achived by classification methods (SVM and Nureal network around 73%), and it is beacuse these methods are supervised but the clustering are

unsupervised and do not use the labels to train the model, but on the other hand the worst are unsupervised with the highest of 68% but by usualizing we understand that the answers are totally wrong and there is a need for data Binarization to elevate the accuracy, the classification even have the same accuracy after running multiple times but on the other hand, it is not the same with Clustering method , every time of running there maybe slight changes in the accuracy.