- Dataset can be found at Pima Indians Diabetes Database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)
- More about K-Means clustering at Neural Network (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)

In [1]:
```
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn.neural_network import MLPClassifier
```

In [2]:
```
df = pd.read_csv("diabetes.csv")
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

What does the dataset contain ?

The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

What is the algorithm

```
An Artificial Neural Network represents interconnected input and output units in which each connection has an
associated weight. During the learning phase, the network learns by adjusting these weights in order to be
able to predict the correct class for input data.
* ANN can be used for both classification and regressions.
```

How does it work

```
* Take inputs
* Add bias (if required)
* Assign random weights to input features
* Run the code for training.
* Find the error in prediction.
* Update the weight by gradient descent algorithm.
* Repeat the training phase with updated weights.
* Make predictions.
```

Advantages and Disadvantges of the algorithm

```
Advantages:
* Gradual corruption:  A network slows over time and undergoes relative degradation. The network problem does
not immediately corrode immediately.
* Ability to work with incomplete knowledge :  After ANN training, the data may produce output even with
incomplete information. The loss of performance here depends on the importance of the missing information.
* Having fault tolerance:  Corruption of one or more cells of ANN does not prevent it from generating output.
This feature makes the networks fault tolerant
* Having a distributed memory: The network's success is directly proportional to the selected instances, and
if the event can not be shown to the network in all its aspects, the network can produce false output

Disadvantages:
* Unexplained behavior of the network: This is the most important problem of ANN. When ANN produces a probing
solution, it does not give a clue as to why and how. This reduces trust in the network.
* Determination of proper network structure:  There is no specific rule for determining the structure of
artificial neural networks. Appropriate network structure is achieved through experience and trial and error.
* Difficulty of showing the problem to the network:  ANNs can work with numerical information. Problems have
to be translated into numerical values before being introduced to ANN. The display mechanism to be determined
here  will directly influence the performance of the network . This depends on the user's ability.
```

How is it performed on the dataset

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

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [5]:
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)
```

In [7]:
```
ss = StandardScaler()

X_train_std = ss.fit_transform(X_train)
X_test_std = ss.transform(X_test)
```

In [8]:
```
X_train.shape
```

Out[8]:
```
(537, 8)
```

In [9]:
```
classifier = MLPClassifier(max_iter=5000)
classifier.fit(X_train_std, y_train)
```

Out[9]:
```
MLPClassifier(max_iter=5000)
```

In [10]:
```
classifier.score(X_train_std, y_train)
```

Out[10]:
```
0.9962756052141527
```

In [11]:
```
classifier.score(X_test_std, y_test)
```

Out[11]:
```
0.6926406926406926
```

Summary

- The score of train dataset is larger than test dataset
- Clearly model is suffering from overfitting
- Overfitting can be avoided by tuning hyperparameters

In [ ]:
```

```