- Dataset can be found at Pima Indians Diabetes Database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)
- More about K-Means clustering at Decision Tree (https://scikit-learn.org/stable/modules/tree.html)

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

from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

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

A decision tree can be used for both classification and regression problems.
A decision tree is a flowchart-like tree structure where an internal node represents feature(or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning. This flowchart-like structure helps you in decision making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.

How does it work

The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures(ASM) to split the records.
2. Make that attribute a decision node and breaks the dataset into smaller subsets.
3. Starts tree building by repeating this process recursively for each child until one of the condition will match:

```
    * All the tuples belong to the same attribute value.
    * There are no more remaining attributes.
    * There are no more instances.
```

Advantages and Disadvantges of the algorithm

```
Advantages:
* Decision trees are easy to interpret and visualize.
* It can easily capture Non-linear patterns.
* It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.
* It can be used for feature engineering such as predicting missing values, suitable for variable selection.
* The decision tree has no assumptions about distribution because of the non-parametric nature of the
algorithm.

Disadvantages:
* Sensitive to noisy data. It can overfit noisy data.
* The small variation(or variance) in data can result in the different decision tree. This can be reduced by
bagging and boosting algorithms.
* Decision trees are biased with imbalance dataset, so it is recommended that balance out the dataset before
creating the decision tree.
```

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]:
```
classifier = tree.DecisionTreeClassifier()
classifier.fit(X_train_std, y_train)
```

Out[8]:
```
DecisionTreeClassifier()
```

In [9]:
```
classifier.score(X_test_std, y_test) ## Accuracy score on test dataset
```

Out[9]:
```
0.7056277056277056
```

In [10]:
```
X_full = ss.transform(X)
predictions = classifier.predict(X_full)
df['Prediction'] = predictions
df.head()
```

Out[10]:

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

In [11]:
```
print("The accuracy score of KNN on the dataset is: {}".format(classifier.score(X_full, y)))
```

```
The accuracy score of KNN on the dataset is: 0.9114583333333334
```

Summary

- The score on training dataset is 0.91
- The score on test ddataset is 0.70
- The model seems to be hurting from overfitting
- This could be overcome by Parameter Hypertuning and/or using bagging techniques like Random Forest

In [ ]: