Importing the Libraries

In [27]:
```
import pandas as pd
# The Pandas library is used for importing and managing the datasets. It is an op
```

Importing DataSets for training our model

In [19]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
# read_csv is used for Reading csv file and pd is a object of pandas which is use
X = dataset.iloc[:,[2,3]].values  # X is the dependent variable which is storing
Y = dataset.iloc[:,4].values # Y is the target variable here y represent the purc
```

Splitting Training and Test Set in which 25 % dataset is for testing and remaining you will use for training the model

Here sklearn is the library in which all algorithms of machine learning is implemented so we can directly use the required algorithm , and can use predefine function of resppective algorithm

x_train: features for the training data

x_test: features for testing data

y_train: Dependent variables for training data

y_test: Independent variable for testing data

In train_test_split() function, we have passed four parameters in which first two are for arrays of data, and test_size is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.

If there is no randomstate provided the system will use a randomstate that is generated internally. So, when you run the program multiple times you might see different train/test data points and the behavior will be unpredictable. In case, you have an issue with your model you will not be able to recreate it as you do not know the random number that was generated when you ran the program

In [28]:
```
from sklearn.model_selection import train_test_split #
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.25,random_state=
# If random_state =None then every time the training and testing dataset will be
# If random_state =any no(example 0 or 1 ) then every time the training and test
```

Feature Scaling

Most of the times, your dataset will contain features highly varying in magnitudes, units, and range. But since most of the machine learning algorithms use Euclidian distance between two data points in their computations, this is a problem. If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units, 5kg, and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes. To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling.

fit: when you want to train your model without any pre-processing on the data

transform: when you want to do pre-processing on the data using one of the functions from sklearn.preprocessing

fit_transform(): It's same as calling transform() and then fit()

While trining our model we need to transform and fit data to the model so we used the fit_transform() functionn but while testing we only need to transform the the data so we used the transform() function

In [21]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
C:\Users\HP\Anaconda3\envs\opencv\lib\site-packages\sklearn\utils\validation.p
y:595: DataConversionWarning: Data with input dtype int64 was converted to floa
t64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\HP\Anaconda3\envs\opencv\lib\site-packages\sklearn\utils\validation.p
y:595: DataConversionWarning: Data with input dtype int64 was converted to floa
t64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
C:\Users\HP\Anaconda3\envs\opencv\lib\site-packages\sklearn\utils\validation.p
y:595: DataConversionWarning: Data with input dtype int64 was converted to floa
t64 by StandardScaler.
  warnings.warn(msg, DataConversionWarning)
```

Fitting Decision Tree Classification Model

In [22]:
```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy',random_state=0)
classifier.fit(X_train,Y_train)
```

Out[22]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

Predicting the test set results

In [23]:
```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or “classifier”) on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. It allows easy identification of confusion between classes e.g. one class is commonly mislabeled as the other. Most performance measures are computed from the confusion matrix.

In [24]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test,y_pred)
```

Out[24]:
```
array([[48, 10],
       [ 7, 35]], dtype=int64)
```

Printing accuracy score

In [25]:
```
from sklearn.metrics import accuracy_score
accuracy_score(Y_test,y_pred)
```

Out[25]:
```
0.83
```

In [ ]:
```

```