Data Preprocessing

Importing the libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Importing the dataset

In [2]:
```
dataset = pd.read_csv('Data.csv')
dataset
```

Out[2]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [3]:
```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 3].values
```

In [4]:
```
X
```

Out[4]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [5]:
```
y
```

Out[5]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Taking Care of Missing Data

SimpleImputer can replace the missing data of X by the mean (median or most_frequent) of the column

In [6]:
```
from sklearn.impute import SimpleImputer
# strategy: string, optional(default = 'mean')
imp = SimpleImputer(missing_values = np.nan,strategy = 'mean')
imp = imp.fit(X[:,1:3])
X[:,1:3] = imp.transform(X[:,1:3])
```

In [7]:
```
X
```

Out[7]:
```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 63777.77777777778],
       ['France', 35.0, 58000.0],
       ['Spain', 38.77777777777778, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

In [8]:
```
y
```

Out[8]:
```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

Encoding Categorical Data

In dataset, it contains "Country"(France,Spain...) and "Purchased"(Yes,No) as Categorical variables.
We only need numbers for machine learning
machine learning model接受数字，不接受文字信息

In [9]:
```
from sklearn.preprocessing import LabelEncoder
```

In [10]:
```
labelencoder_X = LabelEncoder()
X[:,0] = labelencoder_X.fit_transform(X[:,0])    # for column "Country"
X
```

Out[10]:
```
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, 63777.77777777778],
       [0, 35.0, 58000.0],
       [2, 38.77777777777778, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

New problem : Machine Learning models are based on equations, it will consider Spain(2) has higher value than Germany(1)...

So we need use "Dummy Encoding" to prevent this issue. Dummy Encoding will have a number of columns equal to the number of categories

machine learning model会误认为**Spain**大于**Germany,**但如果刻意强调大小（如衣服**S,M,L)**则可以不必用**OneHotEncoder**转换

In [11]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [12]:
```
# 'categorical_features' needs the index of the column we need encode, it's column[0] here
onehotencoder = OneHotEncoder(categorical_features=[0])
# We don't have to put the first column of X info into 'fit_transform' because we have already specified it above
X = onehotencoder.fit_transform(X).toarray()
X
```

```
C:\Users\XZV838\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\preprocessing\_encoders.py:415: FutureWarning: The handling of integer dat
a will change in version 0.22. Currently, the categories are determined bas
ed on the range [0, max(values)], while in the future they will be determin
ed based on the unique values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the ca
tegories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
C:\Users\XZV838\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn
\preprocessing\_encoders.py:451: DeprecationWarning: The 'categorical_featu
res' keyword is deprecated in version 0.20 and will be removed in 0.22. You
can use the ColumnTransformer instead.
  "use the ColumnTransformer instead.", DeprecationWarning)
```

Out[12]:
```
array([[1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.40000000e+01,
        7.20000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 2.70000000e+01,
        4.80000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.00000000e+01,
        5.40000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.10000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.37777778e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.50000000e+01,
        5.80000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.87777778e+01,
        5.20000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 5.00000000e+01,
        8.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.70000000e+01,
        6.70000000e+04]])
```

For dependent variable (y), the machine learning will know that it's a category and there's no order between the two ('Yes','No')
In the case, we will only use label encoder rather than one hot encoder

In [13]:
```
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
y
```

Out[13]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

Feature Scaling

Since each column has different scale (ie, 'Age' and 'Salary'), it will casue some issues in machine learning models
Because a lot of machine learning models are based on Euclidean distance

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

the Euclidean distance will be dominated by 'Salary' (***'Salary'的值远大于'Age',所以导致了'Age'权重影响极小***)
所以，我们需要**put the variables in the same scale** (we will transform these column variables in the same range and scale)

两种方法：

Normalization rescales the values into a range of [0,1]. This might be useful in some cases where all parameters need to have the same positive scale. However, the outliers from the data set are lost.

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

> Standardization rescales data to have a mean (μ) of 0 and standard deviation (σ) of 1 (unit variance).

$$X_{changed} = \frac{X - \mu}{\sigma}$$

For most applications standardization is recommended.

注：Scale后，也会大大加快machine learning处理的速度

In [14]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X = sc_X.fit_transform(X)
X
```

Out[14]:
```
array([[ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
        -1.27555478e+00, -8.91265492e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.63077256e+00,  1.75214693e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01]])
```

We don't need to apply feature scale on the categorical dependent variable(y) for classification
But we need apply feature scale on the huge range values in dependent variable for regression

Splitting the Dataset into the Training & Test set

Use 'random_state' to set it to the same number, so that we will have the same results

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
```

In [16]:
```
X_train
```

Out[16]:
```
array([[-8.16496581e-01,  1.52752523e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [-8.16496581e-01, -6.54653671e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [ 1.22474487e+00, -6.54653671e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01]])
```

In [17]:
```
X_test
```

Out[17]:
```
array([[-0.81649658,  1.52752523, -0.65465367, -1.27555478, -0.89126549],
       [-0.81649658,  1.52752523, -0.65465367,  1.63077256,  1.75214693]])
```

In [18]:
```
y_train
```

Out[18]:
```
array([1, 1, 1, 0, 1, 0, 0, 1])
```

In [19]:
```
y_test
```

Out[19]:
```
array([0, 0])
```