In [1]:

```
import numpy as np
import pandas as pd
```

In [2]:

```
dataset= pd.read_csv("data_for_preprocessing.csv")
```

In [3]:

```
dataset
```

Out[3]:

| | Branch | CGPA | Salary | Placed | Age |
|---|---|---|---|---|---|
| **0** | CSE | 9.5 | 30.0 | Yes | 21.0 |
| **1** | ECE | 9.0 | 25.0 | Yes | NaN |
| **2** | Mech | 8.7 | 20.0 | Yes | NaN |
| **3** | Civil | 8.5 | 15.0 | Yes | NaN |
| **4** | CSE | 9.0 | 28.0 | Yes | 19.0 |
| **5** | ECE | 8.5 | 10.0 | Yes | NaN |
| **6** | Mech | NaN | 7.0 | Yes | NaN |
| **7** | Civil | 7.5 | NaN | No | NaN |
| **8** | CSE | 8.5 | 15.0 | NaN | NaN |
| **9** | ECE | 8.0 | NaN | Yes | 16.0 |
| **10** | Mech | NaN | NaN | Yes | NaN |
| **11** | Civil | 6.8 | 1.0 | No | NaN |
| **12** | Aero | NaN | NaN | NaN | NaN |

In [4]:

```
dataset.isnull().sum()
```

Out[4]:

```
Branch      0
CGPA        3
Salary      4
Placed      2
Age        10
dtype: int64
```

# Delete the rows

In [ ]:

```
dataset.dropna(axis=0)
```

In [ ]:

```
dataset.dropna(thresh=3, axis=0)
```

In [ ]:

```
dataset
```

In [10]:

```
dataset.dropna(thresh=3, axis=0, inplace=True)
```

In [11]:

```
dataset
```

Out[11]:

|  | Branch | CGPA | Salary | Placed | Age |
|---|---|---|---|---|---|
| **0** | CSE | 9.5 | 30.0 | Yes | 21.0 |
| **1** | ECE | 9.0 | 25.0 | Yes | NaN |
| **2** | Mech | 8.7 | 20.0 | Yes | NaN |
| **3** | Civil | 8.5 | 15.0 | Yes | NaN |
| **4** | CSE | 9.0 | 28.0 | Yes | 19.0 |
| **5** | ECE | 8.5 | 10.0 | Yes | NaN |
| **6** | Mech | NaN | 7.0 | Yes | NaN |
| **7** | Civil | 7.5 | NaN | No | NaN |
| **8** | CSE | 8.5 | 15.0 | NaN | NaN |
| **9** | ECE | 8.0 | NaN | Yes | 16.0 |
| **11** | Civil | 6.8 | 1.0 | No | NaN |

# Delete the columns

dataset.dropna(axis=1)

In [126]:

```
dataset.dropna(thresh=9,axis=1,inplace=True)
```

In [127]:

```
dataset
```

Out[127]:

|  | Branch | CGPA | Salary | Placed |
|---|---|---|---|---|
| **0** | CSE | 9.5 | 30.0 | Yes |
| **1** | ECE | 9.0 | 25.0 | Yes |
| **2** | Mech | 8.7 | 20.0 | Yes |
| **3** | Civil | 8.5 | 15.0 | Yes |
| **4** | CSE | 9.0 | 28.0 | Yes |
| **5** | ECE | 8.5 | 10.0 | Yes |
| **6** | Mech | NaN | 7.0 | Yes |
| **7** | Civil | 7.5 | NaN | No |
| **8** | CSE | 8.5 | 15.0 | NaN |
| **9** | ECE | 8.0 | NaN | Yes |
| **11** | Civil | 6.8 | 1.0 | No |

# Filling the Missing Value by Imputation

In [128]:

```
from sklearn.impute import SimpleImputer
```

SimpleImputer is a class found in package sklearn. impute. It is used to impute / replace the numerical or categorical missing data related to one or more features with appropriate values such as following: Each of the above type represents strategy when creating an instance of SimpleImputer.

In [129]:

```
imputer = SimpleImputer(missing_values=np.nan,strategy='mean')
```

In [130]:

```
imputer
```

Out[130]:

```
SimpleImputer(copy=True, fill_value=None, missing_values=nan, stra
tegy='mean',
       verbose=0)
```

In [131]:

```
X=dataset[['Branch','CGPA','Salary']]
```

In [132]:

```
X
```

Out[132]:

| | Branch | CGPA | Salary |
|---|---|---|---|
| **0** | CSE | 9.5 | 30.0 |
| **1** | ECE | 9.0 | 25.0 |
| **2** | Mech | 8.7 | 20.0 |
| **3** | Civil | 8.5 | 15.0 |
| **4** | CSE | 9.0 | 28.0 |
| **5** | ECE | 8.5 | 10.0 |
| **6** | Mech | NaN | 7.0 |
| **7** | Civil | 7.5 | NaN |
| **8** | CSE | 8.5 | 15.0 |
| **9** | ECE | 8.0 | NaN |
| **11** | Civil | 6.8 | 1.0 |

In [133]:

```
#X -convert as an array
```

In [134]:

```
X=dataset[['Branch','CGPA','Salary']].values
```

In [135]:

```
X[:,:]
```

Out[135]:

```
array([['CSE', 9.5, 30.0],
       ['ECE', 9.0, 25.0],
       ['Mech', 8.7, 20.0],
       ['Civil', 8.5, 15.0],
       ['CSE', 9.0, 28.0],
       ['ECE', 8.5, 10.0],
       ['Mech', nan, 7.0],
       ['Civil', 7.5, nan],
       ['CSE', 8.5, 15.0],
       ['ECE', 8.0, nan],
       ['Civil', 6.8, 1.0]], dtype=object)
```

In [136]:

```
X[:,1:3]
```

Out[136]:

```
array([[9.5, 30.0],
       [9.0, 25.0],
       [8.7, 20.0],
       [8.5, 15.0],
       [9.0, 28.0],
       [8.5, 10.0],
       [nan, 7.0],
       [7.5, nan],
       [8.5, 15.0],
       [8.0, nan],
       [6.8, 1.0]], dtype=object)
```

In [137]:

```
imputer = imputer.fit(X[:,1:3])
X
```

Out[137]:

```
array([['CSE', 9.5, 30.0],
       ['ECE', 9.0, 25.0],
       ['Mech', 8.7, 20.0],
       ['Civil', 8.5, 15.0],
       ['CSE', 9.0, 28.0],
       ['ECE', 8.5, 10.0],
       ['Mech', nan, 7.0],
       ['Civil', 7.5, nan],
       ['CSE', 8.5, 15.0],
       ['ECE', 8.0, nan],
       ['Civil', 6.8, 1.0]], dtype=object)
```

In [138]:

```
X[:,1:3]=imputer.transform(X[:,1:3])
X
```

Out[138]:

```
array([['CSE', 9.5, 30.0],
       ['ECE', 9.0, 25.0],
       ['Mech', 8.7, 20.0],
       ['Civil', 8.5, 15.0],
       ['CSE', 9.0, 28.0],
       ['ECE', 8.5, 10.0],
       ['Mech', 8.4, 7.0],
       ['Civil', 7.5, 16.777777777777778],
       ['CSE', 8.5, 15.0],
       ['ECE', 8.0, 16.777777777777778],
       ['Civil', 6.8, 1.0]], dtype=object)
```

## Filling the Categorical Missing Values - for null also it assigns a category

In [12]:

```
y=dataset['Placed'].tolist()
print(type(y))
print(y)

from sklearn.preprocessing import LabelEncoder
label_encode=LabelEncoder()
le = LabelEncoder()
y=le.fit_transform(y)

print(y)
```

```
<class 'list'>
['Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', nan, 'Ye
s', 'No']
[1 1 1 1 1 1 1 0 2 1 0]
```

# Filling the Categorical Missing Values with Mode

In [13]:

```
y=dataset['Placed']
y.mode()
```

Out[13]:

```
0    Yes
dtype: object
```

In [14]:

```
y=y.fillna(y.mode().iloc[0])
```

In [15]:

```
y
```

Out[15]:

```
0      Yes
1      Yes
2      Yes
3      Yes
4      Yes
5      Yes
6      Yes
7       No
8      Yes
9      Yes
11      No
Name: Placed, dtype: object
```

# Encoding categorical data

In [16]:

```
print(y)
from sklearn.preprocessing import LabelEncoder
label_encode=LabelEncoder()
le = LabelEncoder()
y=le.fit_transform(y)

print(y)
```

```
0     Yes
1     Yes
2     Yes
3     Yes
4     Yes
5     Yes
6     Yes
7      No
8     Yes
9     Yes
11     No
Name: Placed, dtype: object
[1 1 1 1 1 1 1 0 1 1 0]
```

# Label Encoding with OneHotEncoder

If you feel this may not useful when branches are given as input, you may go for other encoding methods

In [144]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [145]:

```
onehotencoder = OneHotEncoder()
```

In [146]:

```
#X=dataset[['Branch','CGPA','Salary']].values
```

In [147]:

```
type(X[0])
```

Out[147]:

```
numpy.ndarray
```

In [148]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [149]:

```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
# MODEL GENERATED FOR INDEX 0
```

In [150]:

```
X = np.array(ct.fit_transform(X))
```

In [151]:

```
X
```

Out[151]:

```
array([[1.0, 0.0, 0.0, 0.0, 9.5, 30.0],
       [0.0, 0.0, 1.0, 0.0, 9.0, 25.0],
       [0.0, 0.0, 0.0, 1.0, 8.7, 20.0],
       [0.0, 1.0, 0.0, 0.0, 8.5, 15.0],
       [1.0, 0.0, 0.0, 0.0, 9.0, 28.0],
       [0.0, 0.0, 1.0, 0.0, 8.5, 10.0],
       [0.0, 0.0, 0.0, 1.0, 8.4, 7.0],
       [0.0, 1.0, 0.0, 0.0, 7.5, 16.77777777777778],
       [1.0, 0.0, 0.0, 0.0, 8.5, 15.0],
       [0.0, 0.0, 1.0, 0.0, 8.0, 16.77777777777778],
       [0.0, 1.0, 0.0, 0.0, 6.8, 1.0]], dtype=object)
```

# Congratulations, Completed Data Cleaning

In [ ]:

```

```