Feature Scaling

```
In [ ]: import numpy as np
        import pandas as pd
```

```
In [2]: df = pd.read_csv("C:/Users/SW20407278/Desktop/Final AI/Hands-On/Data Preprocessin
```

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

Out[3]:

	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

```
In [4]: df
```

Out[4]:

	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	NaN	50.0	83000.0	No
9	France	37.0	67000.0	Yes

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    9 non-null      object 
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

In [6]:

```
#### Sk-learn doesnot support the imputation for non-numerical columns.
#### Pandas is used for the imputation for non-numerical columns.

df.Country.fillna(df.Country.mode()[0],inplace = True)
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Country    10 non-null     object 
 1   Age        9 non-null      float64
 2   Salary     9 non-null      float64
 3   Purchased  10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

In [8]:

```
### Features and Labels
features = df.iloc[:,:-1].values
label = df.iloc[:, -1].values
```

In [9]:

```
from sklearn.impute import SimpleImputer

### Creating and Instatiate the Object

age = SimpleImputer(strategy = "mean", missing_values = np.nan )
salary = SimpleImputer(strategy = "mean", missing_values = np.nan)
```

In [10]:

```
### Fitting of the object with the data

age.fit(features[:,[1]])
salary.fit(features[:,[2]])
```

Out[10]:

```
SimpleImputer()
```

```
In [11]: ### Transforming the data with fitted values

         features[:, [1]] = age.transform(features[:,[1]])
         features[:, [2]] = salary.transform(features[:,[2]])
```

```
In [12]: features
```

```
Out[12]: 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],
                ['France', 50.0, 83000.0],
                ['France', 37.0, 67000.0]], dtype=object)
```

```
In [13]: ### One Hot Encoding

         from sklearn.preprocessing import OneHotEncoder
         oh = OneHotEncoder(sparse = False)
```

```
In [14]: country = oh.fit_transform(features[:,[0]])
```

```
In [15]: country
```

```
Out[15]: array([[1., 0., 0.],
                [0., 0., 1.],
                [0., 1., 0.],
                [0., 0., 1.],
                [0., 1., 0.],
                [1., 0., 0.],
                [0., 0., 1.],
                [1., 0., 0.],
                [1., 0., 0.],
                [1., 0., 0.]])
```

```
In [16]: final_set = np.concatenate((country, features[:,[1,2]]), axis = 1)
```

```
In [17]: final_set
```

```
Out[17]: array([[1.0, 0.0, 0.0, 44.0, 72000.0],
                [0.0, 0.0, 1.0, 27.0, 48000.0],
                [0.0, 1.0, 0.0, 30.0, 54000.0],
                [0.0, 0.0, 1.0, 38.0, 61000.0],
                [0.0, 1.0, 0.0, 40.0, 63777.77777777778],
                [1.0, 0.0, 0.0, 35.0, 58000.0],
                [0.0, 0.0, 1.0, 38.77777777777778, 52000.0],
                [1.0, 0.0, 0.0, 48.0, 79000.0],
                [1.0, 0.0, 0.0, 50.0, 83000.0],
                [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

In [19]:

```
## Feature Scaling

## Standard Scaler

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(final_set)
feat_standard_scaler = sc.transform(final_set)
```

In [20]:

```
feat_standard_scaler
```

Out[20]:

```
array([[ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
         7.58874362e-01,  7.49473254e-01],
       [-1.00000000e+00, -5.00000000e-01,  1.52752523e+00,
        -1.71150388e+00, -1.43817841e+00],
       [-1.00000000e+00,  2.00000000e+00, -6.54653671e-01,
        -1.27555478e+00, -8.91265492e-01],
       [-1.00000000e+00, -5.00000000e-01,  1.52752523e+00,
        -1.13023841e-01, -2.53200424e-01],
       [-1.00000000e+00,  2.00000000e+00, -6.54653671e-01,
         1.77608893e-01,  6.63219199e-16],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
        -5.48972942e-01, -5.26656882e-01],
       [-1.00000000e+00, -5.00000000e-01,  1.52752523e+00,
         0.00000000e+00, -1.07356980e+00],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
         1.34013983e+00,  1.38753832e+00],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
         1.63077256e+00,  1.75214693e+00],
       [ 1.00000000e+00, -5.00000000e-01, -6.54653671e-01,
        -2.58340208e-01,  2.93712492e-01]])
```

In [ ]:

```
## MinMaxScaler
```

```
from sklearn.preprocessing import MinMaxScaler
mms = MinMaxScaler(feature_range=(0,1))
mms.fit(final_set)
feat_minmax_scaler = mms.transform(final_set)
```

In [24]:

```
feat_minmax_scaler
```

Out[24]:

```
array([[1.        , 0.        , 0.        , 0.73913043, 0.68571429],
       [0.        , 0.        , 1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        , 0.13043478, 0.17142857],
       [0.        , 0.        , 1.        , 0.47826087, 0.37142857],
       [0.        , 1.        , 0.        , 0.56521739, 0.45079365],
       [1.        , 0.        , 0.        , 0.34782609, 0.28571429],
       [0.        , 0.        , 1.        , 0.51207729, 0.11428571],
       [1.        , 0.        , 0.        , 0.91304348, 0.88571429],
       [1.        , 0.        , 0.        , 1.        , 1.        ],
       [1.        , 0.        , 0.        , 0.43478261, 0.54285714]])
```

In [ ]: