Importing the dataset data.csv

In [18]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

In [9]:

```
df = pd.read_csv('data.csv')
df
```

Out[9]:

	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 [10]:

```
X = df.iloc[:,:-1].values
y = df.iloc[:,-1].values
```

Replace the missing value with column mean

In [11]:

```
numeric_columns = df.select_dtypes(include=['number']).columns
df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].me
df
```

Out[11]:

	Country	Age	Salary	Purchased
0	France	44.000000	72000.000000	No
1	Spain	27.000000	48000.000000	Yes
2	Germany	30.000000	54000.000000	No
3	Spain	38.000000	61000.000000	No
4	Germany	40.000000	63777.777778	Yes
5	France	35.000000	58000.000000	Yes
6	Spain	38.777778	52000.000000	No
7	France	48.000000	79000.000000	Yes
8	Germany	50.000000	83000.000000	No
9	France	37.000000	67000.000000	Yes

Replace the missing value with constant values

In [12]:

```
df.fillna(-1, inplace=True)
df
```

Out[12]:

	Country	Age	Salary	Purchased
0	France	44.000000	72000.000000	No
1	Spain	27.000000	48000.000000	Yes
2	Germany	30.000000	54000.000000	No
3	Spain	38.000000	61000.000000	No
4	Germany	40.000000	63777.777778	Yes
5	France	35.000000	58000.000000	Yes
6	Spain	38.777778	52000.000000	No
7	France	48.000000	79000.000000	Yes
8	Germany	50.000000	83000.000000	No
9	France	37.000000	67000.000000	Yes

Encoding the Independent Variable with OneHotEncoder

In [13]:

```
X
```

Out[13]:

```
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)
```

Encoding the Dependent Variable with LabelEncoder

In [14]:

```
y
```

Out[14]:

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Ye
s'],
      dtype=object)
```

Splitting the dataset into the 80: 20 Training set and Test set

In [16]:

```
X = df.drop(columns=['Purchased'])  # Features
y = df['Purchased']  # Target

# Split the dataset into 80% training and 20% test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Display the shapes of the resulting datasets
print("Training set - Features:", X_train.shape, "Target:", y_train.sha
print("Test set - Features:", X_test.shape, "Target:", y_test.shape)
```

```
Training set - Features: (8, 3) Target: (8,)
Test set - Features: (2, 3) Target: (2,)
```

Perform Feature Scaling using Column-normalization (Hints: use MinMaxScaler)

In [20]:

```
numeric_columns = df.select_dtypes(include=['number']).columns
categorical_columns = df.select_dtypes(exclude=['number']).columns

# Apply MinMaxScaler only to numeric features
scaler = MinMaxScaler()
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Display the first few rows of the DataFrame
print(df.head())
```

```
   Country       Age    Salary Purchased
0   France  0.739130  0.685714        No
1    Spain  0.000000  0.000000       Yes
2  Germany  0.130435  0.171429        No
3    Spain  0.478261  0.371429        No
4  Germany  0.565217  0.450794       Yes
```

load iris.csv dataset and locate rows of duplicate data

In [30]:

```python
df = pd.read_csv('iris.csv')
# Find duplicate rows
duplicate_rows = df[df.duplicated()]

# Display duplicate rows
print("Duplicate rows:")
duplicate_rows
```

```
Duplicate rows:
```

Out[30]:

	5.1	3.5	1.4	0.2	Iris-setosa
33	4.9	3.1	1.5	0.1	Iris-setosa
36	4.9	3.1	1.5	0.1	Iris-setosa
141	5.8	2.7	5.1	1.9	Iris-virginica

Delete duplicate rows in iris dataset

In [31]:

```python
# Display the shape of the dataframe before removing duplicates
print("Shape before removing duplicates:", df.shape)

# Remove duplicate rows
df.drop_duplicates(inplace=True)

# Display the shape of the dataframe after removing duplicates
print("Shape after removing duplicates:", df.shape)
```

```
Shape before removing duplicates: (149, 5)
Shape after removing duplicates: (146, 5)
```

load and summarize the pima-indians-diabetes.csv dataset

In [47]:

```
# Load the dataset
df = pd.read_csv('pima-indians-diabetes.csv')

# Display the first few rows of the DataFrame
print("First few rows of the dataset:")
print(df.head())

# Display summary statistics of the dataset
print("\nSummary statistics of the dataset:")
print(df.describe())

# Display information about the dataset
print("\nInformation about the dataset:")
print(df.info())
```

```
First few rows of the dataset:
   6  148  72  35    0  33.6  0.627  50  1
0  1   85  66  29    0  26.6  0.351  31  0
1  8  183  64   0    0  23.3  0.672  32  1
2  1   89  66  23   94  28.1  0.167  21  0
3  0  137  40  35  168  43.1  2.288  33  1
4  5  116  74   0    0  25.6  0.201  30  0

Summary statistics of the dataset:
                6         148          72          35           0  
33.6  \
count  767.000000  767.000000  767.000000  767.000000  767.000000  76
7.000000
mean     3.842243  120.859192   69.101695   20.517601   79.903520   3
1.990482
std      3.370877   31.978468   19.368155   15.954059  115.283105   
7.889091
min      0.000000    0.000000    0.000000    0.000000    0.000000   
0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   2
7.300000
50%      3.000000  117.000000   72.000000   23.000000   32.000000   3
2.000000
75%      6.000000  140.000000   80.000000   32.000000  127.500000   3
6.600000
max     17.000000  199.000000  122.000000   99.000000  846.000000   6
7.100000

            0.627          50           1
count  767.000000  767.000000  767.000000
mean     0.471674   33.219035    0.348110
std      0.331497   11.752296    0.476682
min      0.078000   21.000000    0.000000
25%      0.243500   24.000000    0.000000
50%      0.371000   29.000000    0.000000
75%      0.625000   41.000000    1.000000
max      2.420000   81.000000    1.000000

Information about the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 767 entries, 0 to 766
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   6       767 non-null    int64  
 1   148     767 non-null    int64  
 2   72      767 non-null    int64  
 3   35      767 non-null    int64  
 4   0       767 non-null    int64  
 5   33.6    767 non-null    float64
 6   0.627   767 non-null    float64
 7   50      767 non-null    int64  
 8   1       767 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

Count the number of missing values for each column (In this dataset 0 is treated as missing value)

In [52]:

```
missing_values = (df == 0).sum()

# Display the count of missing values for each column
print("Number of missing values (including zeros) for each column:")
missing_values
```

```
Number of missing values (including zeros) for each column:
```

Out[52]:

```
6        111
148        5
72        35
35       227
0        373
33.6      11
0.627      0
50         0
1        500
dtype: int64
```

drop rows with missing values

In [53]:

```
df = df.replace(0, pd.NA)

# Drop rows with missing values
df_cleaned = df.dropna()

# Display the shape of the cleaned dataset
print("Shape of the cleaned dataset after dropping rows with missing va
```

```
Shape of the cleaned dataset after dropping rows with missing values:
(111, 9)
```