Assignment 11 | Data Preprocessing Lab

Akash Duttachowdhury

21052386

For file: data.csv

Importing the dataset data.csv

In [ ]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the datasets
dataset = pd.read_csv('Data.csv')
dataset
```

Out[ ]:

	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 [ ]:

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

Replace the missing value with column mean

In [ ]:

```
# Taking care of missing data
from sklearn.impute import SimpleImputer
```

```
# Initialize the SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='mean')

# Fit and transform the imputer on the data
X[:, 1:3] = imputer.fit_transform(X[:, 1:3])

# Print the first few rows of X after imputation
print("X after imputation:")
print(X[:5])  # Print the first 5 rows for demonstration
```

```
X after imputation:
[['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]]
```

Replace the missing value with constant values

In [ ]:

```
# Initialize the SimpleImputer with mean strategy
imputer = SimpleImputer(strategy='constant', fill_value=69000)

# Fit and transform the imputer on the data
X[:, 1:3] = imputer.fit_transform(X[:, 1:3])

# Print the first few rows of X after imputation
print("X after imputation:")
print(X[:5])  # Print the first 5 rows for demonstration
```

```
X after imputation:
[['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]]
```

Encoding the Independent Variable with OneHotEncoder

In [ ]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.pipeline import Pipeline

# Creating a pipeline for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('encoder', OneHotEncoder(), [0])  # Apply OneHotEncoder to colum
    ],
    remainder='passthrough'  # Keep the remaining columns as they are
)

# Fit and transform the data using the pipeline
X_encoded = preprocessor.fit_transform(X)

# Print the shape and a sample of the encoded X
```

```
print("Shape of X_encoded:", X_encoded.shape)
print("Sample of X_encoded:")
print(X_encoded[:5])  # Print the first 5 rows for demonstration
```

```
Shape of X_encoded: (10, 5)
Sample of X_encoded:
[[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]]
```

Encoding the Dependent Variable with LabelEncoder

In [ ]:

```
# Creating the object of LabelEncoder class
labelencoder_y = LabelEncoder()

# fit labelencoder_y object to last coulmn Purchased, we will get encoded
y = labelencoder_y.fit_transform(y)
y
```

Out[ ]:

```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

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

In [ ]:

```
# Splitting the dataset into training set and test set
from sklearn.model_selection import train_test_split

# Choosing 20% data as test data, so we will have 80% data in training se
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
```

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

In [ ]:

```
from sklearn.preprocessing import MinMaxScaler
sc_X = MinMaxScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

For file: iris.csv

load iris.csv dataset and locate rows of duplicate data

In [ ]:

```
iris_data = pd.read_csv('iris.csv')
iris_data
```

Out[ ]:

	5.1	3.5	1.4	0.2	Iris-setosa
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	5.0	3.6	1.4	0.2	Iris-setosa
4	5.4	3.9	1.7	0.4	Iris-setosa
...	...	...	...	...	...
144	6.7	3.0	5.2	2.3	Iris-virginica
145	6.3	2.5	5.0	1.9	Iris-virginica
146	6.5	3.0	5.2	2.0	Iris-virginica
147	6.2	3.4	5.4	2.3	Iris-virginica
148	5.9	3.0	5.1	1.8	Iris-virginica

149 rows × 5 columns

In [ ]:

```
duplicates = iris_data.duplicated()
print(duplicates)
```

```
0      False
1      False
2      False
3      False
4      False
       ...
144    False
145    False
146    False
147    False
148    False
Length: 149, dtype: bool
```

Delete duplicate rows in iris dataset

In [ ]:

```
print(iris_data.shape)
iris = iris_data.drop_duplicates()
print(iris.shape)
```

```
(149, 5)
(146, 5)
```

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

In [ ]:

```
pid = pd.read_csv('pima-indians-diabetes.csv')
pid.shape
pid.info()
```

```
<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
```

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

```
In [ ]: # Replace 0 with NaN in the entire DataFrame
        pid.replace(0, np.NaN, inplace=True)
```

```
In [ ]: # Count the number of missing values (NaN) in each column
        missing_values_count = pid.isnull().sum()
        missing_values_count
```

```
Out[ ]: 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 [ ]: print(pid.shape)
        pid.dropna(inplace=True)
        print(pid.shape)
```

```
(767, 9)
(111, 9)
```