Data Preprocessing in Machine learning

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

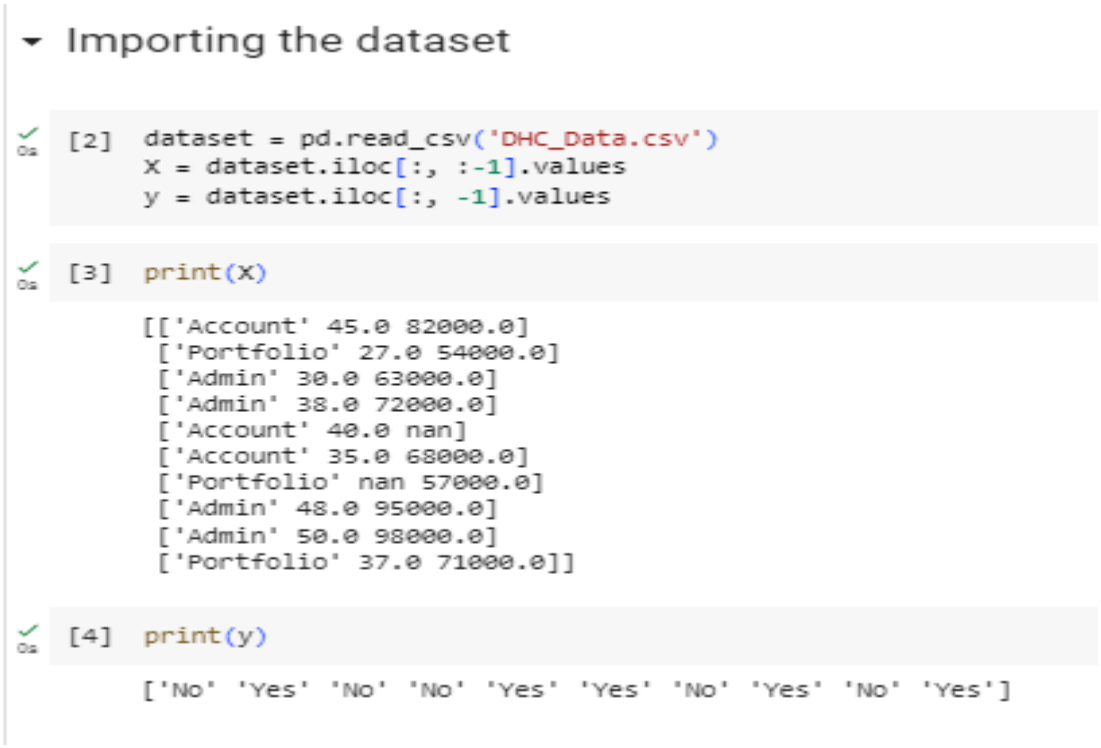
Data Pre-Processing
- Import the data
- Clean the data
- Split into training & test sets
- Feature Scaling

Importing Data & Libraries

Data: DHC_Data.csv

	A	B	C	D
1	Department	Age	Salary	Purchased
2	Account	45	82000	No
3	Portfolio	27	54000	Yes
4	Admin	30	63000	No
5	Admin	38	72000	No
6	Account	40		Yes
7	Account	35	68000	Yes
8	Portfolio		57000	No
9	Admin	48	95000	Yes
10	Admin	50	98000	No
11	Portfolio	37	71000	Yes

Load Data with Python Standard Library



```
▾ Importing the dataset

[2]  dataset = pd.read_csv('DHC_Data.csv')
     X = dataset.iloc[:, :-1].values
     y = dataset.iloc[:, -1].values

[3]  print(X)

     [['Account' 45.0 82000.0]
      ['Portfolio' 27.0 54000.0]
      ['Admin' 30.0 63000.0]
      ['Admin' 38.0 72000.0]
      ['Account' 40.0 nan]
      ['Account' 35.0 68000.0]
      ['Portfolio' nan 57000.0]
      ['Admin' 48.0 95000.0]
      ['Admin' 50.0 98000.0]
      ['Portfolio' 37.0 71000.0]]

[4]  print(y)

     ['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

With Python Standard Library, you will be using the module CSV (**Comma-Separated Values**) and the function reader() to load your CSV files. Upon loading, the CSV data will be automatically converted to NumPy array which can be used for machine learning.

Importing Libraries:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices.

Matplotlib: The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot.**

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets.

Importing the libraries

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

Handling Missing data

There are mainly two ways to handle missing data, which are:

By deleting the particular row:

The first way is used to commonly deal with null values. If it is less than 1% data is null values then you can simply delete it.

By calculating the mean:

In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library.

Taking care of missing data

```
[5]  from sklearn.impute import SimpleImputer
     imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
     imputer.fit(X[:, 1:3])
     X[:, 1:3] = imputer.transform(X[:, 1:3])
```

```
[6]  print(X)
```

```
[['Account' 45.0 82000.0]
 ['Portfolio' 27.0 54000.0]
 ['Admin' 30.0 63000.0]
 ['Admin' 38.0 72000.0]
 ['Account' 40.0 73333.33333333333]
 ['Account' 35.0 68000.0]
 ['Portfolio' 38.888888888888886 57000.0]
 ['Admin' 48.0 95000.0]
 ['Admin' 50.0 98000.0]
 ['Portfolio' 37.0 71000.0]]
```

Encoding Categorical Data

Encoding categorical data is a process of converting categorical data into integer format so that the data can be provided to different models. Categorical data will be in the form of strings or object data types. But, machine learning or deep learning algorithms can work only on numbers.

Categorical Data: Department & Purchased

Encoding categorical data

Encoding the Independent Variable

```
[7] from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import OneHotEncoder
    ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')
    X = np.array(ct.fit_transform(X))
```

```
print(X)
```

```
[[1.0 0.0 0.0 45.0 82000.0]
 [0.0 0.0 1.0 27.0 54000.0]
 [0.0 1.0 0.0 30.0 63000.0]
 [0.0 1.0 0.0 38.0 72000.0]
 [1.0 0.0 0.0 40.0 73333.33333333333]
 [1.0 0.0 0.0 35.0 68000.0]
 [0.0 0.0 1.0 38.888888888888886 57000.0]
 [0.0 1.0 0.0 48.0 95000.0]
 [0.0 1.0 0.0 50.0 98000.0]
 [0.0 0.0 1.0 37.0 71000.0]]
```

Encoding the Dependent Variable

```
[9] from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    y = le.fit_transform(y)
```

```
[10] print(y)
```

```
[0 1 0 0 1 1 0 1 0 1]
```

Split into training & test sets

To accurately assess your ML model's performance without overfitting or underfitting issues, it's necessary to split your dataset into two separate sets:

- **Training set:** Helps train the algorithm on real-world examples
- **Testing set:** Used later for evaluating its generalization capabilities on unseen instances. It is common to use a train/test split of 70/30 or 80/20.

Splitting data into training and testing sets is an essential step in the development of machine learning models . It involves dividing the available dataset into separate subsets for training, validation, and testing the model. The most common approach is to split the dataset into a **training set** and a **testing set**. The training set is used to train the model, while the testing set is used to evaluate the model's performance . The regular split is 70-80% for training and 20-30% for testing, but this may vary depending on the size of the dataset and the specific use case .
The primary reason for splitting data into training and testing sets is to **prevent overfitting** . Overfitting occurs when a model is trained too well on the training data, resulting in poor performance on new, unseen data. By evaluating the model's performance on a separate testing set, we can estimate how well it will perform on new data .
It's important to note that splitting data into training and testing sets is not enough to prevent overfitting. Other techniques such as **cross-validation** and **regularization** are also used to prevent overfitting .

Why we have to apply feature scaling after the splitting data training set and test set?

Test set suppose to be brand new set which is going to be evaluated your machine learning model. Your training machine learning model your training set after that you are going to deploy on new observation. So, test set not supposed to work with your training. Feature scaling is a technique that you will get the mean and standard deviation of features. If we apply feature scaling before the split that mean it will get the mean and standard deviation all the values including once of test set. Applying feature scaling on original data before split which cause some information leakage on the test set. So we grab some information from the test set that which not suppose to get because it is supposed to be new data new observation.

Point should be noted: feature scaling after the splitting data int test set and training set to prevent the information leakage of the test set.

Splitting the dataset into the Training set and Test set

```
[11] 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 = 1)
```

```
[12] print(X_train)
```

```
[[0.0 0.0 1.0 38.888888888888886 57000.0]
 [1.0 0.0 0.0 40.0 73333.33333333333]
 [1.0 0.0 0.0 45.0 82000.0]
 [0.0 1.0 0.0 38.0 72000.0]
 [0.0 0.0 1.0 27.0 54000.0]
 [0.0 1.0 0.0 48.0 95000.0]
 [0.0 1.0 0.0 50.0 98000.0]
 [1.0 0.0 0.0 35.0 68000.0]]
```

```
[13] print(X_test)
```

```
[[0.0 1.0 0.0 30.0 63000.0]
 [0.0 0.0 1.0 37.0 71000.0]]
```

```
[14] print(y_train)
```

```
[0 1 0 0 1 1 0 1]
```

```
[15] print(y_test)
```

```
[0 1]
```

Feature scaling

Feature scaling is a technique used in **machine learning** to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing stage. The purpose of feature scaling is to bring all the features to the same level of magnitude, which helps in improving the performance of machine learning algorithms that use optimization algorithms or metrics that depend on some kind of distance metric .
There are different methods for feature scaling, including **standardization**, **min-max scaling**, and **unit vector scaling** . Standardization scales the data to have a mean of zero and a standard deviation of one. Min-max scaling scales the data to a fixed range, usually between 0 and 1. Unit vector scaling scales the data to have a length of 1. Feature scaling is important because it helps in avoiding bias towards features with higher magnitudes, which can lead to poor performance of machine learning models. It also helps in reducing the time required for training machine learning models.

Normalization

Xn = (X - Xmin) / (Xmax - Xmin)

Xn = Value of Normalization

Xmax= Maximum value of a feature

Xmin = Minimum value of a feature

Standardization

Normalization

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

[0 ; 1]

Standardization

$$X' = \frac{X - \mu}{\sigma}$$

[-3 ; +3]

Standardization = (Current_value – Mean) / Standard Deviation.

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
[17] print(X_train)
```

```
[[-0.77459667 -0.77459667  1.73205081 -0.19350256 -1.19321136]
 [ 1.29099445 -0.77459667 -0.57735027 -0.03391282 -0.10544659]
 [ 1.29099445 -0.77459667 -0.57735027  0.68424102  0.47173472]
 [-0.77459667  1.29099445 -0.57735027 -0.32117436 -0.19424371]
 [-0.77459667 -0.77459667  1.73205081 -1.90111282 -1.39300489]
 [-0.77459667  1.29099445 -0.57735027  1.11513333  1.33750669]
 [-0.77459667  1.29099445 -0.57735027  1.40239487  1.53730022]
 [ 1.29099445 -0.77459667 -0.57735027 -0.75206667 -0.46063508]]
```

```
print(X_test)
```

```
[[-0.77459667  1.29099445 -0.57735027 -1.47022051 -0.7936243 ]
 [-0.77459667 -0.77459667  1.73205081 -0.46480513 -0.26084155]]
```