Department of Computer Engineering

Academic Year: 2024-25 **Semester: VIII**
Class / Branch: BE Computer **Subject: Applied Data Science Lab**

Experiment No. 2

1. **Aim:** To apply data cleaning techniques.

Dataset: In this experiment, a fictitious data containing 10 observations and 4 variables is used. The dataset contains Country, Age, Salary, and purchased columns. The dataset has categorical variables and missing values in these columns.

2. Software used: Google Colaboratory / Jupyter Notebook

3. Theory :-

Data cleaning is just the collective name to a series of actions we perform on our data in the process of getting it ready for analysis.

Some of the steps in data cleaning are:

- Handling missing values
- Encoding categorical features
- Outliers detection
- Transformations etc.

Handling missing values is a key part of data preprocessing and hence, it is of utmost importance for data scientists/machine learning engineers to learn different techniques in relation imputing / replacing numerical or categorical missing values with appropriate value based on appropriate strategies. That's primarily the reason we need to convert categorical columns to numerical columns so that a machine learning algorithm understands it. This process is called categorical encoding.

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.

Typically, any structured dataset includes multiple columns – a combination of numerical as well as categorical variables. A machine can only understand the numbers. It cannot understand the text. That's essentially the case with Machine Learning algorithms too. There are multiple ways of handling Categorical variables.
The two most widely used techniques:

- Label Encoding
- One-Hot Encoding

Label Encoding is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical ordering.

One-Hot Encoding is the process of creating dummy variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature.

4. Program

Simple Imputer

SimpleImputer Explained With Python Code Example

In [1]:
```
import pandas as pd
import numpy as np
students = [[86, 'M', 'verygood'],[95, 'F', 'excellent'], [75, None,'good'],[np.NaN, 'M', 'average'],[7
1, 'M', 'good'],
[np.NaN, None, 'verygood'],[92, 'F', 'verygood'],[99, 'M', 'excellent']]
```

In [2]:
```
dfstud = pd.DataFrame(students)
```

In [3]:
```
dfstud.columns = ['marks', 'gender', 'result']
```

In [4]:
```
dfstud
```

Out[4]:

	marks	gender	result
0	86.0	M	verygood
1	95.0	F	excellent
2	75.0	None	good
3	NaN	M	average
4	71.0	M	good
5	NaN	None	verygood
6	92.0	F	verygood
7	99.0	M	excellent

In [5]:
```
dfstud.isnull().values.sum()
```

Out[5]: 4

In [6]:
```
X=dfstud.iloc[:,0:2].values
y=dfstud.iloc[:,2].values
type(X)
```

Out[6]: numpy.ndarray

SimpleImputer for Imputing Numerical Missing Data

In [7]:
```
from sklearn.impute import SimpleImputer
# Missing values is represented using NaN and hence specified. If it
# is empty field, missing values will be specified as ''
imputer = SimpleImputer(missing_values=np.NaN, strategy='mean')
X[:,0:1]= imputer.fit_transform(X[:,0:1])
print(X)
#OR
```

```
[[86.0 'M']
 [95.0 'F']
 [75.0 None]
 [86.33333333333333 'M']
 [71.0 'M']
 [86.33333333333333 None]
 [92.0 'F']
 [99.0 'M']]
```

In [8]:
```
imputer = SimpleImputer(missing_values=np.NaN, strategy='mean')
dfstud.marks = imputer.fit_transform(dfstud['marks'].values.reshape(-1,1))[:,0]
dfstud
```

Out[8]:

	marks	gender	result
0	86.000000	M	verygood
1	95.000000	F	excellent
2	75.000000	None	good
3	86.333333	M	average
4	71.000000	M	good
5	86.333333	None	verygood
6	92.000000	F	verygood
7	99.000000	M	excellent

Example: imputer = SimpleImputer(missing_values='NaN', strategy='mean') imputer.fit(x['Age'].values.reshape(-1, 1)) x['Age'] = imputer.transform(x['Age'].values.reshape(-1, 1)) imp = SimpleImputer(missing_values ="NaN",strategy = "mean",axis = 0) x['Age'] = imp.fit_transform(x['Age'].reshape(-1,1))

In [9]:
```
# Imputing with mean value
imputer = SimpleImputer(missing_values=np.NaN, strategy='mean')

# Imputing with median value
imputer = SimpleImputer(missing_values=np.NaN, strategy='median')

# Imputing with most frequent / mode value
imputer = SimpleImputer(missing_values=np.NaN, strategy='most_frequent')

# Imputing with constant value; The command below replaces the missing value with constant value such a
s 80
imputer = SimpleImputer(missing_values=np.NaN, strategy='constant', fill_value=80)
```

SimpleImputer for Imputing Categorical Missing Data

For handling categorical missing values, you could use one of the following strategies. However, it is the "most_frequent" strategy which is preferably used.

```
Most frequent (strategy='most_frequent')
Constant (strategy='constant', fill_value='someValue')
```

In [10]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=None, strategy='most_frequent')
dfstud.gender = imputer.fit_transform(dfstud['gender'].values.reshape(-1,1))[:,0]
dfstud
```

Out[10]:

	marks	gender	result
0	88.000000	M	verygood
1	95.000000	F	excellent
2	75.000000	M	good
3	86.333333	M	average
4	71.000000	M	good
5	86.333333	M	verygood
6	92.000000	F	verygood
7	99.000000	M	excellent

In [11]:
```
dfstud1 = pd.DataFrame(students)
```

In [12]:
```
dfstud1.columns = ['marks', 'gender', 'result']
```

In [13]:
```
dfstud1
```

Out[13]:

	marks	gender	result
0	88.0	M	verygood
1	95.0	F	excellent
2	75.0	None	good
3	NaN	M	average
4	71.0	M	good
5	NaN	None	verygood
6	92.0	F	verygood
7	99.0	M	excellent

In [14]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=None, strategy='constant', fill_value='F')
dfstud1.gender = imputer.fit_transform(dfstud1['gender'].values.reshape(-1,1))[:,0]
dfstud1
```

Out[14]:

	marks	gender	result
0	88.0	M	verygood
1	95.0	F	excellent
2	75.0	F	good
3	NaN	M	average
4	71.0	M	good
5	NaN	F	verygood
6	92.0	F	verygood
7	99.0	M	excellent

In [1]:
```python
import numpy as np
import pandas as pd
```

In [2]:
```python
ds=pd.read_csv(r'C:\Users\Ramya\Desktop\ML\datasets\CountryAgeSalary.csv')
ds
```

Out[2]:

	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 [3]:
```python
X=ds.iloc[:,:-1].values
Y=ds.iloc[:,3].values
```

In [4]:
```python
# Taking care of missing data
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy='mean')
imputer = imputer.fit(X[:,1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

In [5]:
```python
print(X)
```

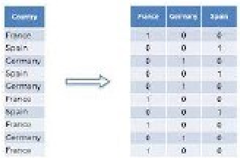
```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

In [6]:
```python
#from sklearn.preprocessing import LabelEncoder
#labelencoder = LabelEncoder()
#X[:,0] = labelencoder.fit_transform(X[:,0])
```

In [7]:
```python
print(X)
```

```
[['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]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

In [8]:
```python
#from sklearn.preprocessing import OneHotEncoder
#onehotencoder = OneHotEncoder(categorical_features = [0])
#X = onehotencoder.fit_transform(X).toarray()
```



In [9]:
```python
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

In [10]:
```python
columnTransformer = ColumnTransformer([('encoder', OneHotEncoder(), [0])], remainder='passthrough')
```

In [11]:
```python
X = np.array(columnTransformer.fit_transform(X), dtype = np.str)
```

In [12]:
```python
print(X)
```

```
[['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']
 ['0.0' '1.0' '0.0' '50.0' '83000.0']
 ['1.0' '0.0' '0.0' '37.0' '67000.0']]
```

5. Conclusion :-

Sklearn.impute class SimpleImputer can be used to impute/replace missing values for both numerical and categorical features. For numerical missing values, a strategy such as mean, median, most frequent, and constant can be used. For categorical features, a strategy such as the most frequent and constant can be used. Categorical variables can be converted into numerical using label encoding or one-hot encoding.