GANPAT UNIVERSITY

U. V. Patel College of Engineering

Practical:2

2CEIT6PE4: Data Science / Machine Learning

B.Tech Semester: VI

Computer Engineering/ Information Technology

Enrolment No: 19012012009 **Name: Priyank Bhavsar**

Aim:

1. Understanding of Data Pre-processing for given dataset 1 using Spyder (Python)

Solution:

- import pandas as pd
  import numpy as np
  df=pd.read_csv("dataset1.csv")

df - DataFrame

Index	Country	Age	Salary	Purchased
0	France	44	72000	No
1	Spain	27	48000	Yes
2	Germany	30	54000	No
3	Spain	38	61000	No
4	Germany	40	nan	Yes
5	France	35	58000	Yes
6	Spain	nan	52000	No
7	France	48	79000	Yes
8	Germany	50	83000	No
9	France	37	67000	Yes

- df.shape

```
In [5]: df.shape
Out[5]: (10, 4)
```

- df.info()

```
In [6]: 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
```

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

```
In [8]: x
Out[8]:
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)

In [9]: y
Out[9]:
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

#simpleImputer to fill nan

#Mean

- 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 [11]: x
Out[11]:
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],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

#Most Frequent

- from sklearn.impute import SimpleImputer
  imputer = SimpleImputer(missing_values = np.nan,strategy='most_frequent')
  imputer = imputer.fit(x[:,1:3])
  x[:,1:3] = imputer.transform(x[:,1:3])

```
In [15]: x
Out[15]:
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 48000.0],
       ['France', 35.0, 58000.0],
       ['Spain', 27.0, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

#Median

- from sklearn.impute import SimpleImputer
  imputer = SimpleImputer(missing_values = np.nan,strategy=’median’)
  imputer = imputer.fit(x[:,1:3])
  x[:,1:3] = imputer.transform(x[:,1:3])

```
In [18]: x
Out[18]:
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 61000.0],
       ['France', 35.0, 58000.0],
       ['Spain', 38.0, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

#Constant

- from sklearn.impute import SimpleImputer
  imputer = SimpleImputer(missing_values = np.nan,strategy=constant)
  imputer = imputer.fit(x[:,1:3])
  x[:,1:3] = imputer.transform(x[:,1:3])

```
In [21]: x
Out[21]:
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, 'missing_value'],
       ['France', 35.0, 58000.0],
       ['Spain', 'missing_value', 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

3. Understanding of categorical data.

Solution:

There are many ways to convert categorical values into numerical values. Each approach has its own trade-offs and impact on the feature set.

1)LabelEncoder

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

2)OnehotEncoder

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.

3)CountVectorizer

CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

4. Replace Country Attribute for given dataset 1 by fit_transform method

Solution:

#labelEncoder

- from sklearn.preprocessing import LabelEncoder

  label_x=LabelEncoder()

  x[:,0]=label_x.fit_transform(x[:,0])

```
Out[34]:
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, 63777.77777777778],
       [0, 35.0, 58000.0],
       [2, 38.77777777777778, 52000.0],
       [0, 48.0, 79000.0],
       [1, 50.0, 83000.0],
       [0, 37.0, 67000.0]], dtype=object)
```

5. Replace categorical and numerical Attributes for given dataset 2 by OneHotEncoder Class.

Solution:

#OneHotEncoder

- df1=pd.read_csv('dataset2.csv')
  X=df1.iloc[:,:].values
  from sklearn.preprocessing import LabelEncoder , OneHotEncoder
  from sklearn.compose import ColumnTransformer

  transform = ColumnTransformer([("Col 0",OneHotEncoder(),[0])], remainder = 'passthrough')
  X = transform.fit_transform(X)

  transform = ColumnTransformer([("Column 4 converted",OneHotEncoder(),[4])],remainder = 'passthrough')
  X = transform.fit_transform(X)

  transform = ColumnTransformer([("Outlook_OL0_OL1",OneHotEncoder(),[6])],remainder = 'passthrough')
  X = transform.fit_transform(X)
  print(X.astype(int))

```
In [24]: X
Out[24]:
array([[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0],
       [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0],
       [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0],
       [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0],
       [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0],
       [0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0],
       [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0],
       [0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0]],
      dtype=object)
```

EXERCISE

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

```
array([[0.5       , 0.61904762, 0.50896057],
       [0.        , 0.47619048, 0.61290323],
       [1.        , 0.        , 0.        ],
       [1.        , 0.56084656, 0.12903226],
       [0.        , 1.        , 1.        ],
       [1.        , 0.52380952, 0.41935484],
       [0.        , 0.80952381, 0.77419355],
       [0.        , 0.38095238, 0.32258065]])
```

  X_test

```
In [8]: X_test
Out[8]:
array([[0.5       , 0.14285714, 0.19354839],
       [0.5       , 1.0952381 , 1.12903226]])
```

- Calculate a feature scaling by using Standard Scaler and MinMax Scaler both Mathematically.

  1> Min Max Scaler

Country	**x-min(x)/max(x)-min(x)**
0	0
2	1
1	½
2	1
1	½
0	0
2	1
0	0
1	½
0	0
Age	**x-min(x)/max(x)-min(x)**
44	0.739130435

27	0
30	0.130434783
38	0.47826087
40	0.565217391
35	0.347826087
38.7778	0.512078261
48	0.913043478
50	1
37	0.434782609

Salary	**x-min(x)/max(x)-min(x)**
72000	-0.314285714
48000	-1
54000	-0.828571429
61000	-0.628571429
63777.7778	-0.549206349
58000	-0.714285714
52000	-0.885714286
79000	-0.114285714
83000	0
67000	-0.457142857

2>Standard Scaler

Country	**xi-x'**	**(xi-x')2**	**Value**
0	-0.90000	0.81	-1.027872433
2	1.10000	1.21	1.256288529
1	0.10000	0.01	0.114208048
2	1.10000	1.21	1.256288529
1	0.10000	0.01	0.114208048
0	-0.90000	0.81	-1.027872433

2	1.10000	1.21	1.256288529
0	-0.90000	0.81	-1.027872433
1	0.10000	0.01	0.114208048
0	-0.90000	0.81	-1.027872433
Mean=	0.90		
		6.9	0.76666667
standard daviation	0.875595036		

Age	xi-x'	(xi-x')2		Value
44	5.22222	27.27158173		0.719625078
27	-11.77778	138.7161017		-1.623981499
30	-8.77778	77.04942173		-1.210403867
38	-0.77778	0.604941728		-0.107530184
40	1.22222	1.493821728		0.168188237
35	-3.77778	14.27162173		-0.521107815
38.7778	0.00002	4E-10		-0.00030329
48	9.22222	85.04934173		1.27106192
50	11.22222	125.9382217		1.546780341
37	-1.77778	3.160501728		-0.245389395
		473.5555556	52.61728395	
Mean=	38.78			
standard daviation	7.25377722			

Salary__1	xi-x'	(xi-x')2		Value
72000	-9777.77778	67604938.24		0.711012566
48000	-2777.77778	248938271.7		-1.364376026
54000	0.00002	95604938.32		-0.845528878
61000	-5777.77778	7716049.395		-0.240207205
63777.7778	0.77778	4E-10		-1.90244E-07
58000	-11777.77778	33382716.08		-0.499630779
52000	15222.22222	138716049.4		-1.018477927
79000	19222.22222	231716049.3		1.316334239
83000	3222.22222	369493827.1		1.662232338
67000	0.00002	10382716.04		0.278639943
		1203555556	133728395.1	133728395.1
Mean=	63777.78			
standard daviation	11564.09941			