Roll No: 205229133

Lab6. Pandas Data Cleaning Part-II

LabelEncoder in Scikit Learn

- Encodes string values as integer values

In [1]:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

In [2]:

```
le = LabelEncoder()
#New object
df = pd.DataFrame(
    data={
        'col1': ['foo', 'bar', 'foo', 'bar'],
        'col2': ['x', 'y', 'x', 'z'],
        'col3': [1, 2, 3, 4]
    })
```

In [3]:

```
#Now convert string values of each column into integer values
df.apply(le.fit_transform)
```

Out[3]:

	col1	col2	col3
0	1	0	0
1	0	1	1
2	1	0	2
3	0	2	3

One Hot Encoder

- Consider the following dataframe. You will have to represent string values of column A and B with integers

In [4]:

```
import pandas as pd

df = pd.DataFrame({'A': ['a', 'b', 'a'], 'B': ['b', 'a', 'c'], 'C': [1, 2, 3]})
df
```

Out[4]:

	A	B	C
0	a	b	1
1	b	a	2
2	a	c	3

In [5]:

```
# Call get_dummies method. It will create a new column for each string value in DF columns
# here prefix tells which columns should be encoded
pd.get_dummies(df, prefix=['col1', 'col2'])
```

Out[5]:

	C	col1_a	col1_b	col2_a	col2_b	col2_c
0	1	1	0	0	1	0
1	2	0	1	1	0	0
2	3	1	0	0	0	1

MinMaxScaler

- It will transform values into a range of 0 to 1

In [6]:

```
from sklearn.preprocessing import MinMaxScaler

mm_scaler = MinMaxScaler(feature_range=(0, 1))  # (0,1) is default range
df2 = pd.DataFrame({
    "col1": [5, -41, -67],
    "col2": [23, -53, -36],
    "col3": [-25, 10, 17]
})
mm_scaler.fit_transform(df2)
```

Out[6]:

```
array([[1.        , 1.        , 0.        ],
       [0.36111111, 0.        , 0.83333333],
       [0.        , 0.22368421, 1.        ]])
```

Binarizer

- It will encode values into 0 or 1, depending on the threshold

In [7]:

```
from sklearn.preprocessing import Binarizer

dfb = pd.DataFrame({
    "col1": [110, 200],
    "col2": [120, 800],
    "col3": [310, 400]
})
bin = Binarizer(threshold=300)
bin.fit_transform(dfb)
```

Out[7]:

```
array([[0, 0, 1],
       [0, 1, 1]], dtype=int64)
```

Imputer

- You can also use Imputer from sklearn to handle NaN objects in each columns. Here, we replace NaN with column mean value. This is good alternative to fillna() method.

In [8]:

```
import numpy as np
from sklearn.impute import SimpleImputer
import pandas as pd

imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
df = pd.DataFrame({
    "col1": [7, 2, 3],
    "col2": [4, np.nan, 6],
    "col3": [np.nan, np.nan, 3],
    "col4": [10, np.nan, 9]
})
print(df)
imp_mean.fit_transform(df)
```

```
   col1  col2  col3  col4
0     7   4.0   NaN  10.0
1     2   NaN   NaN   NaN
2     3   6.0   3.0   9.0
```

Out[8]:

```
array([[ 7. ,  4. ,  3. , 10. ],
       [ 2. ,  5. ,  3. ,  9.5],
       [ 3. ,  6. ,  3. ,  9. ]])
```

De-duplication or Entity Resolution and String Matching

- You can use dedupe and fuzzywuzzy packages. Install them using pip3 and import inside your Python code

fuzzywuzzy

In [9]:

```
import warnings
warnings.filterwarnings('ignore')

from fuzzywuzzy import fuzz
from fuzzywuzzy import process

a = 'Welcome to Bishop Heber College '
b = 'I am Maheshvaran Pursuing Masters in DataScience at Bishop Heber College'
ratio = fuzz.ratio(a, b)
weighted_ratio = fuzz.WRatio(a, b)
unicode_ratio = fuzz.UQRatio(a, b)
print('Ratio is:', ratio)
print('Weighted ratio:', weighted_ratio)
print('Unicode ratio:', unicode_ratio)
```

```
Ratio is: 50
Weighted ratio: 86
Unicode ratio: 50
```

In [10]:

```
c=a+b
```

In [11]:

```
ex_tract = process.extract('I', c)
ex_tract
```

Out[11]:

```
[('i', 100), ('I', 100), ('i', 100), ('i', 100), ('i', 100)]
```

In [12]:

```
extract1 = process.extractOne('I', c)
extract1
```

Out[12]:

```
('i', 100)
```

dedupe

In [13]:

```
import dedupe
# List of duplicate character names
dupes = [
    'Sathish Kumar', 'Dinesh Kumar', 'Praveen Kumar', 'Kumaresan', 'Kumaran',
    'Pradeep Kumar', 'Ashok Kumar', 'Raj Kumar', 'Kumar','Paveeen Kumar','Sathish'
]
# Print the duplicate values
# As the threshold decreases the number of duplications found will increase
process.dedupe(dupes, threshold=10)
```

Out[13]:

```
dict_keys(['Paveeen Kumar', 'Sathish Kumar'])
```

In [14]:

```
extract = process.extract('Pra', dupes, limit=3)
extract
```

Out[14]:

```
[('Praveen Kumar', 90), ('Pradeep Kumar', 90), ('Kumaran', 60)]
```

In [15]:

```
extractone = process.extractOne('Pra', dupes)
extractone
```

Out[15]:

```
('Praveen Kumar', 90)
```