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

In [2]:
```
df = pd.read_csv("D:\Summer Training Video\ML\covid_toy.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	age	gender	fever	cough	city	has_covid
0	60	Male	103.0	Mild	Kolkata	No
1	27	Male	100.0	Mild	Delhi	Yes
2	42	Male	101.0	Mild	Delhi	No
3	31	Female	98.0	Mild	Kolkata	No
4	65	Female	101.0	Mild	Mumbai	No

In [4]:
```
df = df.dropna()
```

In [5]:
```
df = df.drop(columns = ['age' , 'fever'])
```

In [6]:
```
df.head()
```

Out[6]:

	gender	cough	city	has_covid
0	Male	Mild	Kolkata	No
1	Male	Mild	Delhi	Yes
2	Male	Mild	Delhi	No
3	Female	Mild	Kolkata	No
4	Female	Mild	Mumbai	No

In [7]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [9]:
```
df.shape
```

Out[9]:
```
(90, 4)
```

get_dummies method

In [10]:
```
p = pd.get_dummies(df , columns = ['gender' , 'cough' , 'city' , 'has_covid'])
```

In [11]:
```
p.shape
```

Out[11]:
```
(90, 10)
```

In [12]:
```
p
```

Out[12]:

	gender_Female	gender_Male	cough_Mild	cough_Strong	city_Bangalore	city_Delhi	city_Kolkata	city_Mumbai	has_covid_No	has_covid_Yes
0	False	True	True	False	False	False	True	False	True	False
1	False	True	True	False	False	True	False	False	False	True
2	False	True	True	False	False	True	False	False	True	False
3	True	False	True	False	False	False	True	False	True	False
4	True	False	True	False	False	False	False	True	True	False
...	...	...	...	...	...	...	...	...	...	...
95	True	False	True	False	True	False	False	False	True	False
96	True	False	False	True	False	False	True	False	False	True
97	True	False	True	False	True	False	False	False	True	False
98	True	False	False	True	False	False	False	True	True	False
99	True	False	False	True	False	False	True	False	False	True

90 rows × 10 columns

column Transformer

In [13]:
```
import numpy as np
import pandas as pd

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

from sklearn.preprocessing import OrdinalEncoder
```

In [14]:
```
df = pd.read_csv("D:\Summer Training Video\ML\covid_toy.csv")
```

In [15]:
```
df
```

Out[15]:

	age	gender	fever	cough	city	has_covid
0	60	Male	103.0	Mild	Kolkata	No
1	27	Male	100.0	Mild	Delhi	Yes
2	42	Male	101.0	Mild	Delhi	No
3	31	Female	98.0	Mild	Kolkata	No
4	65	Female	101.0	Mild	Mumbai	No
...	...	...	...	...	...	...
95	12	Female	104.0	Mild	Bangalore	No
96	51	Female	101.0	Strong	Kolkata	Yes
97	20	Female	101.0	Mild	Bangalore	No
98	5	Female	98.0	Strong	Mumbai	No
99	10	Female	98.0	Strong	Kolkata	Yes

100 rows × 6 columns

In [18]:
```
df.isnull().sum()
```

Out[18]:
```
age           0
gender        0
fever        10
cough         0
city          0
has_covid     0
dtype: int64
```

In [19]:
```
from sklearn.model_selection import train_test_split
```

In [20]:
```
x_train , x_test , y_train , y_test = train_test_split(df.drop(columns = ['has_covid']) , df['has_covid'] , test_size =0.
```

In [21]:
```
x_train
```

Out[21]:

	age	gender	fever	cough	city
75	5	Male	102.0	Mild	Kolkata
55	81	Female	101.0	Mild	Mumbai
18	64	Female	98.0	Mild	Bangalore
16	69	Female	103.0	Mild	Kolkata
57	49	Female	99.0	Strong	Bangalore
...	...	...	...	...	...
45	72	Male	99.0	Mild	Bangalore
20	12	Male	98.0	Strong	Bangalore
49	44	Male	104.0	Mild	Mumbai
37	55	Male	100.0	Mild	Kolkata
10	75	Female	NaN	Mild	Delhi

80 rows × 5 columns

Manually type output

In [22]:
```
# adding simple imputer to fever columns
si = SimpleImputer()
x_train_fever = si.fit_transform(x_train[['fever']])

# also the test data
x_test_fever = si.fit_transform(x_test[['fever']])

x_train_fever.shape
```

Out[22]:
```
(80, 1)
```

In [26]:
```
# Ordinal Encoding   ---> Cough

oe = OrdinalEncoder(categories = [['Mild' , 'Strong']])
x_train_cough = oe.fit_transform(x_train[['cough']])

# also the test data
x_test_cough = oe.fit_transform(x_test[['cough']])

x_train_cough.shape
```

Out[26]:
```
(80, 1)
```

In [27]:
```
# OneHotEncoding ---> Gender , city

ohe = OneHotEncoder(drop = 'first' , sparse = False)
x_train_gender_city = ohe.fit_transform(x_train[['gender' , 'city']])

# also the test data
x_test_gender_city = ohe.fit_transform(x_test[['gender' , 'city']])

x_test_gender_city.shape
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed t
o `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed t
o `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
```

Out[27]:
```
(20, 4)
```

In [28]:
```
# Extracting age

x_train_age = x_train.drop(columns = ['gender' , 'fever' , 'cough' , 'city']).values

# also the test data

x_test_age = x_test.drop(columns = ['gender' , 'fever' , 'cough' , 'city']).values
```

In [29]:
```
x_train_age.shape
```

Out[29]: (80, 1)

In [30]:
```
x_train_transformed = np.concatenate((x_train_age , x_train_fever , x_train_gender_city ,
                                      x_train_cough) , axis = 1)
```

In [31]:
```
# x_test_transformed = np.concatenate(x_train_age , x_train_fever , x_train_gender_city ,
#                                     x_train_cough) , axis = 1)
```

In [32]:
```
x_train_transformed.shape
```

Out[32]: (80, 7)

by the help of column Transformer

In [34]:
```
from sklearn.compose import ColumnTransformer     # this is  how to import ColumnTransformer

transformer = ColumnTransformer(transformers=[
    ('tnf1',SimpleImputer(),['fever']),      # in a 'fever' column by the SI we fill missing value
    ('tnf2',OrdinalEncoder(categories=[['Mild','Strong']]),['cough']),   # by this process we encode our
    ('tnf3',OneHotEncoder(sparse=False,drop='first'),['gender','city'])
],remainder = 'passthrough') # remainder = passthrough --> It means rest all the column same.
```

In [35]:
```
transformer.fit_transform(x_train).shape
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed t
o `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its
default value.
  warnings.warn(
```

Out[35]: (80, 7)

In [36]:
```
transformer.transform(x_test).shape
```

Out[36]: (20, 7)

what is function transformer in machine learning?

The Function Transformer is a tool in scikit-learn, a popular Python library for machine learning, that allows you to apply a specified function to the input data. The Function Transformer can be useful for performing custom transformations of input data in machine learning pipeline.

The Function Transformer takes as input a single function that will be applied to each sample in the data. This function can be any Python function that takes a single argument, such as a lambda function or a user-defined function. This function should return the transformed sample.

In [37]:
```
from sklearn.preprocessing import FunctionTransformer
import numpy as np

# create a dataset
X = np.array(([1,2] , [3,4]))

# define the trasformation function
log_transform = FunctionTransformer(np.log1p)

# apply the transformation to the dataset
X_transformed = log_transform.transform(X)

#view the transformed data
print(X_transformed)
```

```
[[0.69314718 1.09861229]
 [1.38629436 1.60943791]]
```

types of function transformer in machine learning ?

There are two types of FunctionTransformer available in scikit learn:

FunctionTransformer - This transformer allows you to specify a single function that will be applied to the entire input data matrix. This transformer can be useful for feature scaling or feature extraction.

Column Transformer - This transformer allows you to specify a different function for each column or subset of columns in the input data matrix. This transformer can be useful for appling different transformations to different features in a dataset.

Both of these transformers are part of the scikit-learn library in Python and can be used in machine learning pipeline to perprocess data before training a model

for which condition I have to use function transformer in machine learning ?

we might consider using a Function Transformer in a machine learning pipeline in the following situations:

Custom feature engineering: If you want to engineer new features using a custom function, you can use a Function Transformer to apply the function to the input data matrix and create new features based on the output.

Scaling and normalization: If you want to scale or normalize the input data matrix in a custom way, you can use a Function Transformer to apply a custom scaling or normalization function.

Data cleaning: If you want to clean the input data matrix by removing outliers, imputing missing values, or replacing certain values, you can use a Function Transformer to apply a custom cleaning function.

Dimensionality reduction: If you want to reduce the dimensionally of the input data matrix by selecting a subset of features or by applying & dimensionally reduction technique such as PCA, you can use a Function Transformer to apply the custom function.

In general, a Function Tranformer can be useful for any situation in which you want to apply a custom function to the input data matrix before training a machine learning model.

In [38]:
```
# 1. Custom Feature Engineering

from sklearn.preprocessing import FunctionTransformer
import numpy as np

# create a dataset
X = np.array(([1,2] , [3,4]))

# define a custom feature engineering function
def my_feature_engineering(X):
    return np.hstack((X,X**2))

# create a FunctionTransformer to apply the custom function
custom_transformer = FunctionTransformer(my_feature_engineering)

# apply the transformer to the input data
X_transformed = custom_transformer.transform(X)

# view the tranformed data
print(X_transformed)
```

```
[[ 1  2  1  4]
 [ 3  4  9 16]]
```

In [40]:
```
# 2. Scaling and Normalization

from sklearn.preprocessing import FunctionTransformer
import numpy as np

# creata a dataset
X = np.array(([1,2] , [3,4]))

# define a custom scaling function
def my_scaling(X):
    return X / np.max(X)

# create a FunctionTransformer to apply the custom function
custom_transformer = FunctionTransformer(my_scaling)

# apply the transformer to the input data
X_transformed = custom_transformer.transform(X)

# view the tranformed data
print(X_transformed)
```

```
[[0.25 0.5 ]
 [0.75 1.  ]]
```

In [41]:
```
# 3.  Data cleaning

from sklearn.preprocessing import FunctionTransformer
import numpy as np

# creata a dataset with missing values
X = np.array(([1,2] , [3,np.nan]))

# define a custom cleaning function
def my_cleaning(X):
    X[np.isnan(X)] = 0
    return X

# create a FunctionTransformer to apply the custom function
custom_transformer = FunctionTransformer(my_cleaning)

# apply the transformer to the input data
X_transformed = custom_transformer.transform(X)

# view the tranformed data
print(X_transformed)
```

```
[[1. 2.]
 [3. 0.]]
```

In [ ]: