Data Transformation

Before analyzing the dataset, you need to transform this dataset. This process is called data transformation.

Data Transformation is required for:

1. Finding the specific values
2. Changing all the values in a particular format.
3. Applying functions to all the values in a row/column.
4. Mapping
5. Replacing
6. Selecting

i) df.transform()

> **The transform() method allows you to execute a function for each value of the DataFrame.**
>
> **Syntax:** DataFrame.transform(func, axis, *args, **kwargs)
>
> Parameter :
>
> > **func :** Function to use for transforming the data
> >
> > **axis :** {0 or ‘index’, 1 or ‘columns’}, default 0
> >
> > **args :** Positional arguments to pass to func.
> >
> > ** kwargs : Keyword arguments to pass to func.

Example 1: Use DataFrame.transform() function to add 10 to each element in the dataframe.

```
# importing pandas as pd
import pandas as pd

# Creating the DataFrame
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                   "B":[7, 2, 54, 3, None],
                   "C":[20, 16, 11, 3, 8],
                   "D":[14, 3, None, 2, 6]})

# Print the DataFrame
print(df)
```

```
      A     B   C     D
0  12.0   7.0  20  14.0
1   4.0   2.0  16   3.0
2   5.0  54.0  11   NaN
3   NaN   3.0   3   2.0
4   1.0   NaN   8   6.0
```

```
# add 10 to each element of the dataframe
a = df.transform(func = lambda x : x + 10)

print(a)
```

```
      A     B   C     D
0  22.0  17.0  30  24.0
1  14.0  12.0  26  13.0
```

```
2  15.0  64.0  21   NaN
3   NaN  13.0  13  12.0
4  11.0   NaN  18  16.0
```

Example 2 : Use DataFrame.transform() function to find the square root and the result of euler's number raised to each element of the dataframe.

Now we will use DataFrame.transform() function to find the square root and the result of euler's number raised to each element of the dataframe.

```
# pass a list of functions
b = df.transform(func = ['sqrt', 'exp','ceil','floor'])

# Print the result
print(b)
```

```
          A                                  B                             \
       sqrt            exp  ceil floor      sqrt           exp  ceil floor
0  3.464102  162754.791419  12.0  12.0  2.645751  1.096633e+03   7.0   7.0
1  2.000000      54.598150   4.0   4.0  1.414214  7.389056e+00   2.0   2.0
2  2.236068     148.413159   5.0   5.0  7.348469  2.830753e+23  54.0  54.0
3       NaN            NaN   NaN   NaN  1.732051  2.008554e+01   3.0   3.0
4  1.000000       2.718282   1.0   1.0       NaN           NaN   NaN   NaN

          C                                 D                          
       sqrt           exp  ceil floor      sqrt           exp  ceil floor
0  4.472136  4.851652e+08  20.0  20.0  3.741657  1.202604e+06  14.0  14.0
1  4.000000  8.886111e+06  16.0  16.0  1.732051  2.008554e+01   3.0   3.0
2  3.316625  5.987414e+04  11.0  11.0       NaN           NaN   NaN   NaN
3  1.732051  2.008554e+01   3.0   3.0  1.414214  7.389056e+00   2.0   2.0
4  2.828427  2.980958e+03   8.0   8.0  2.449490  4.034288e+02   6.0   6.0
```

Transforming Data using Function Application and Mapping

i) apply()

- The apply() function is used to execute a function to a single row/column, all or list of multiple rows/columns.
- **apply() is used to apply a function along an axis of the DataFrame or on values of Series.**

Syntax:

> DataFrame.apply(func)

Let's create a sample DataFrame

For example:

Let's say we have three columns and would like to apply a function on a single column without touching other two columns and return a DataFrame with three columns.

```
import pandas as pd
import numpy as np
data = [(3,5,7), (2,4,6),(5,8,9)]
df = pd.DataFrame(data, columns = ['A','B','C'])
print(df)
```

```
   A  B  C
0  3  5  7
1  2  4  6
2  5  8  9
```

A) Pandas apply() in Single Column

We will create a function add_A() which adds value 4 to a column and use this on apply() function.

To apply it to a single column, use the column name using df["col_name"].

The below example applies a function to a column B.

```
# Using apply function single column
def add_A(x):
   return x+4


df["B"] = df["B"].apply(add_A)
print(df)
```

```
   A   B  C
0  3   9  7
1  2   8  6
2  5  12  9
```

B) Pandas apply() in All Columns

In some cases we would want to apply a function on all pandas columns, you can do this using apply() function. Here the add_3() function will be applied to all DataFrame columns.

```
# Using Dataframe.apply() to apply function add column
def add_3(x):
   return x+3

df2 = df.apply(add_3)
print(df2)
```

```
   A   B   C
0  6  12  10
1  5  11   9
2  8  15  12
```

C) Pandas apply() in Multiple List of Columns

Using apply() method, you can apply a function on a selected multiple list of columns. In this case, the function will apply to only selected two columns without touching the rest of the columns.

```
# apply() function on selected list of multiple columns
df = pd.DataFrame(data, columns = ['A','B','C'])

df[['A','B']] = df[['A','B']].apply(add_3)

print(df)
```

```
   A   B  C
```

```
0  6   8  7
1  5   7  6
2  8  11  9
```

Lambda Function

> A lambda function in python is a small function that can take any number of arguments and execute an expression.

Apply Lambda Function to Each Column

pandas.DataFrame.apply() can be used with python lambda to execute expression.

```
# apply a lambda function to each column
df2 = df.apply(lambda x : x + 10)
print(df2)
```

```
    A   B   C
0  16  18  17
1  15  17  16
2  18  21  19
```

Apply Lambda Function to Single Column

```
# Using Dataframe.apply() and lambda function
df["A"] = df["A"].apply(lambda x: x-2)
print(df)
```

```
   A   B  C
0  4   8  7
1  3   7  6
2  6  11  9
```

```
# Using DataFrame.map() to Single Column
df['A'] = df['A'].map(lambda A: A/2.)
print(df)
```

```
     A   B  C
0  2.0   8  7
1  1.5   7  6
2  3.0  11  9
```

```
# Using DataFrame.map() to Single Column
df['A'] = df['A'].map(lambda A: A*3)
print(df)
```

```
     A   B  C
0  6.0   8  7
1  4.5   7  6
2  9.0  11  9
```

```
#row1+row2+row3
df.apply(np.sum, axis=0)
```

```
A    19.5
B    26.0
```

```
C    22.0
dtype: float64
```

```
#col1col2+col3
df.apply(np.sum, axis=1)
```

```
0    21.0
1    17.5
2    29.0
dtype: float64
```

```
import numpy as np

df.apply(np.sqrt)
```

	A	B	C
0	2.44949	2.828427	2.645751
1	2.12132	2.645751	2.449490
2	3.00000	3.316625	3.000000

```
df.apply(lambda x: [1, 2], axis=1)
```

```
0    [1, 2]
1    [1, 2]
2    [1, 2]
dtype: object
```

ii) applymap() function

The applymap() function is used to apply a function to a Dataframe elementwise.

This method applies a function that accepts and returns a scalar to every element of a DataFrame.

Syntax:

> **DataFrame.applymap(self, func)**

```
import pandas as pd

df = pd.DataFrame({
    'name':['john','mary','peter','jeff','bill','lisa','jose'],
    'age':[23,78,22,19,45,33,20],
    'gender':['M','F','M','M','M','F','M'],
    'state':['california','dc','california','dc','california','texas','texas'],
    'num_children':[2,1,3,3,2,1,4],
    'num_pets':[5,1,0,5,2,2,3]
})

df
```

	name	age	gender	state	num_children	num_pets
0	john	23	M	california	2	5
1	mary	78	F	dc	1	1

```
df.applymap(lambda x: len(str(x)))
```

	name	age	gender	state	num_children	num_pets
0	4	2	1	10	1	1
1	4	2	1	2	1	1
2	5	2	1	10	1	1
3	4	2	1	2	1	1
4	4	2	1	10	1	1
5	4	2	1	5	1	1
6	4	2	1	5	1	1

```
df.applymap(lambda x: str(x) + '_X')
```

	name	age	gender	state	num_children	num_pets
0	john_X	23_X	M_X	california_X	2_X	5_X
1	mary_X	78_X	F_X	dc_X	1_X	1_X
2	peter_X	22_X	M_X	california_X	3_X	0_X
3	jeff_X	19_X	M_X	dc_X	3_X	5_X
4	bill_X	45_X	M_X	california_X	2_X	2_X
5	lisa_X	33_X	F_X	texas_X	1_X	2_X
6	jose_X	20_X	M_X	texas_X	4_X	3_X

Note:

- apply() works on a row / column basis of a DataFrame
- applymap() works element-wise on a DataFrame and
- map() works element-wise on a Series/ Single Column