

# Session6-Pandas (Operations, Pivot, Stack)\_temp

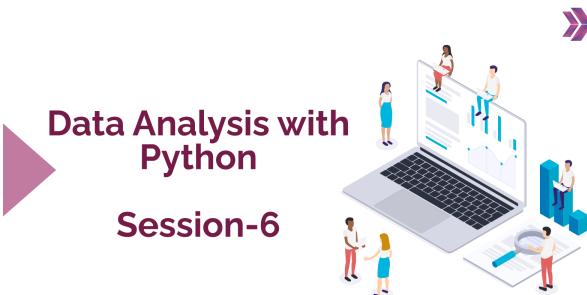
DAwithPython S-6  
Training Clarusway  
Pear Deck - January 26, 2022 at 7:39PM

## Part 1 - Summary

Use this space to summarize your thoughts on the lesson

## Part 2 - Responses

Slide 1



The slide features a large purple play button icon on the left. To its right, the text "Data Analysis with Python" is written in a large, bold, maroon font, with "Session-6" in a slightly smaller font below it. To the right of the text is a 3D illustration of a laptop screen displaying various charts and graphs. Several small human figures are interacting with the laptop and the data. A purple double-headed arrow icon is positioned above the laptop. At the bottom left, there is a small CLARUSWAY logo with the tagline "data to knowledge revealed".

Use this space to take notes:

## Slide 2



CLARUSWAY®  
WAY TO REINVENT YOURSELF

Use this space to take notes:

## Slide 3

### ▶ Table of Contents

- ▶ DataFrame/Series Operations
  - Transform
  - Apply
  - Applymap
  - Map
  - Pivot & Pivot Table
  - Stack & Unstack



3

Use this space to take notes:

## Slide 4

## Your Response

I've completed the pre-class content?

True

False

You Chose

- **True**

Other Choices

- False

A student icon says: Students choose an option

Pear Deck Interactive Slide  
Do not remove this slide

Use this space to take notes:

## Slide 5

### ► DataFrame/Series Operations ➞

#### DataFrame.transform

- ▶ Produced DataFrame will have same axis length as self.

```
>>> df = pd.DataFrame({'A': range(3), 'B': range(1, 4)})
>>> df
   A  B
0  0  1
1  1  2
2  2  3
>>> df.transform(lambda x: x + 1)
   A  B
0  1  2
1  2  3
2  3  4
```

```
>>> s = pd.Series(range(3))
>>> s
0    0
1    1
2    2
dtype: int64
>>> s.transform([np.sqrt, np.exp])
      sqrt      exp
0  0.000000  1.000000
1  1.000000  2.718282
2  1.414214  7.389056
```

5

Use this space to take notes:

## Slide 6

## ▶ DataFrame/Series Operations ➤

### DataFrame.groupby.transform

df2		
groups	var1	var2
0	A	10
1	B	23
2	C	33
3	A	22
4	B	11
5	C	99
6	A	76
7	B	84
8	C	45

df2.groupby("groups")["var1"].mean()		
groups		
A	36.000000	
B	39.333333	
C	59.000000	
Name: var1, dtype: float64		
df2.groupby("groups")["var1"].transform("mean")		
0	36.000000	
1	39.333333	
2	59.000000	
3	36.000000	
4	39.333333	
5	59.000000	
6	36.000000	
7	39.333333	
8	59.000000	

8

Use this space to take notes:

## Slide 7

## ▶ DataFrame/Series Operations ➤

### Types of "Apply"s

<b>Series.apply</b>
For applying more complex functions on a Series.
<b>DataFrame.apply</b>
Apply a function row-/column-wise.
<b>DataFrame.applymap</b>
Apply a function elementwise on a whole DataFrame.

9

Use this space to take notes:

## Slide 8

## ► DataFrame/Series Operations ➞

### Series.apply

- Invoke function on values of Series.

```
>>> s = pd.Series([20, 21, 12],  
...                 index=['London', 'New York', 'Helsinki'])  
>>> s  
London    20  
New York  21  
Helsinki  12
```

```
>>> def square(x):  
...     return x ** 2  
>>> s.apply(square)  
London    400  
New York  441  
Helsinki 144
```

```
>>> s.apply(lambda x: x ** 2)  
London    400  
New York  441  
Helsinki 144
```

```
>>> s.apply(np.log)  
London    2.995732  
New York  3.044522  
Helsinki 2.484907
```

8

Use this space to take notes:

## Slide 9

## ► DataFrame/Series Operations ➞

### DataFrame.apply

- Apply a function along an axis of the DataFrame.

```
>>> df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])  
>>> df  
   A  B  
0  4  9  
1  4  9  
2  4  9
```

Using a numpy universal function (in this case the same as `np.sqrt(df)`):

```
>>> df.apply(np.sqrt)  
   A    B  
0  2.0  3.0  
1  2.0  3.0  
2  2.0  3.0
```

Using a reducing function on either axis

```
>>> df.apply(np.sum, axis=0)  
   A    B  
0  12   27  
dtype: int64
```

```
>>> df.apply(np.sum, axis=1)  
0    13  
1    13  
2    13  
dtype: int64
```

9

Use this space to take notes:

## Slide 10

## ► DataFrame/Series Operations ➞

### DataFrame.applymap

- ▶ Apply a function elementwise on a whole DataFrame.

```
>>> df = pd.DataFrame([[1, 2.12], [3.356, 4.567]])  
>>> df  
          0      1  
0  1.000  2.120  
1  3.356  4.567
```

```
>>> df.applymap(lambda x: len(str(x)))  
          0      1  
0  3  4  
1  5  5
```

```
>>> df.applymap(lambda x: x**2)  
          0      1  
0  1.000000  4.494400  
1  11.262736 20.857489
```

But it's better to avoid applymap in that case.

```
>>> df ** 2  
          0      1  
0  1.000000  4.494400  
1  11.262736 20.857489
```

10

Use this space to take notes:

## Slide 11

## ► DataFrame/Series Operations ➞

### Series.map

- ▶ Map values of Series according to input correspondence.

```
>>> s = pd.Series(['cat', 'dog', np.nan, 'rabbit'])  
>>> s  
0    cat  
1    dog  
2    NaN  
3   rabbit
```

```
>>> s.map({'cat': 'kitten', 'dog': 'puppy'})  
0    kitten  
1     puppy  
2      NaN  
3      NaN
```

```
>>> s.map("I am a {}".format)  
0    I am a cat  
1    I am a dog  
2      NaN  
3  I am a rabbit
```

```
>>> s.map("I am a {}".format, na_action='ignore')  
0    I am a cat  
1    I am a dog  
2      NaN  
3  I am a rabbit
```

11

Use this space to take notes:

## Slide 12

## ► DataFrame/Series Operations ➞

### apply() vs applymap() vs map()

	DataFrame	Series
apply	✓	✓
map		✓
applymap	✓	

- ▶ **apply()** is used to apply a function **along an axis of the DataFrame or on values of Series**.
- ▶ **applymap()** is used to apply a function to a **DataFrame elementwise**.
- ▶ **map()** is used to **substitute each value in a Series with another value**.

12

Use this space to take notes:

## Slide 13

## ► DataFrame/Series Operations ➞

### apply() vs transform()

df5.apply(lambda x: x+10)	df5.transform(lambda x: x+10)																
<table border="1"><thead><tr><th>A</th><th>B</th></tr></thead><tbody><tr><td>0</td><td>1 10</td></tr><tr><td>1</td><td>2 20</td></tr><tr><td>2</td><td>3 30</td></tr></tbody></table>	A	B	0	1 10	1	2 20	2	3 30	<table border="1"><thead><tr><th>A</th><th>B</th></tr></thead><tbody><tr><td>0</td><td>11 20</td></tr><tr><td>1</td><td>12 30</td></tr><tr><td>2</td><td>13 40</td></tr></tbody></table>	A	B	0	11 20	1	12 30	2	13 40
A	B																
0	1 10																
1	2 20																
2	3 30																
A	B																
0	11 20																
1	12 30																
2	13 40																

13

Use this space to take notes:

## Slide 14

## ► DataFrame/Series Operations ➤

### apply() vs transform()

- ▶ **transform()** works with function, a string function, a list of functions, and a dict. However, **apply()** is only allowed with function.
- ▶ **transform()** cannot produce aggregated results.
- ▶ **apply()** works with multiple Series at a time. But, **transform()** is only allowed to work with a single Series at a time.
- ▶ **transform()** returns a DataFrame that has the same length as the input

14

Use this space to take notes:

## Slide 15

## ► DataFrame/Series Operations ➤

### apply() vs transform()

- ▶ **transform()** works with function, a string function, a list of functions, and a dict. However, **apply()** is only allowed with function.

`df5.transform("np.sqrt")  
df5.apply('np.sqrt')` gives an error

A	B
0	1 10
1	2 20
2	3 30

`df5.transform(['np.sqrt', 'np.exp'])  
df5.apply(['np.sqrt', 'np.exp'])`

	A	B	sqrt	exp
0	1.000000	3.162278	2.718282	3.162278
1	1.414214	4.472136	4.472136	4.851652e+08
2	1.732051	5.477226	1.732051	20.088657

`df5.transform(np.sqrt)  
df5.apply(np.sqrt)`

A	B
0	1.000000 3.162278
1	1.414214 4.472136
2	1.732051 5.477226

`df5.transform(['A': np.sqrt, 'B': np.exp])  
df5.apply([('A': np.sqrt, 'B': np.exp)])`

	A	B
0	1.000000 2.205674e+04	2.205674e+04
1	1.414214 4.851652e+08	4.851652e+08
2	1.732051 1.068947e+13	1.068947e+13

15

Use this space to take notes:

## Slide 16

## ► DataFrame/Series Operations ➞

### apply() vs transform()

- ▶ `transform()` cannot produce aggregated results.

A	B
0	10
1	20
2	30

```
df5.apply(lambda x: x.sum())
# df5.transform(Lambda x:x.sum()) gives an error
```

```
A      6
B     60
dtype: int64
```

18

Use this space to take notes:

## Slide 17

## ► DataFrame/Series Operations ➞

### apply() vs transform()

- ▶ `apply()` works with multiple Series at a time. But, `transform()` is only allowed to work with a single Series at a time.

A	B
0	10
1	20
2	30

```
df5.apply(lambda x: x["B"]-x["A"], axis=1)
# df5.transform(Lambda x: x["B"]-x["A"], axis=1) gives an error
```

```
0    9
1   18
2   27
dtype: int64
```

17

Use this space to take notes:

## Slide 18

## ► DataFrame/Series Operations ➤

### apply() vs transform()

- ▶ `apply()` works with multiple Series at a time. But, `transform()` is only allowed to work with a single Series at a time.

key	A	B
0	a	0
1	b	1
2	c	2
3	a	3
4	b	4
5	c	5
6	a	6
7	b	7
8	c	8

```
df6.groupby('key').apply(lambda x: x["B"]-x["A"])
# df6.groupby('key').transform(Lambda x: x["B"]-x["A"]) gives an error
```

key	0	1
a	0	1
	3	-2
	6	-5
b	1	1
	4	-2
	7	-5
c	2	1
	5	-2
	8	-5

18

Use this space to take notes:

Slide 19

## ► DataFrame/Series Operations ➤

### apply() vs transform()

- ▶ `transform()` returns a DataFrame that has the same length as the input

key	A	B
0	a	0
1	b	1
2	c	2
3	a	3
4	b	4
5	c	5
6	a	6
7	b	7
8	c	8

```
df6.groupby('key')[['A']].sum()
```

key	9
a	9
b	12
c	15

```
df6.groupby('key')[['A']].transform(lambda x: x.sum())
```

0	9
1	12
2	15
3	9
4	12
5	15
6	9
7	12
8	15

19

Use this space to take notes:

Slide 20

## ► DataFrame/Series Operations ➞

### DataFrame.pivot

Row index in the new table	Columns in the new table	Cell values in the new table						
ix	Item	CType	USD	EU	ix=Item	Bronze	Gold	Silver
0	Item0	Gold	1\$	1€	Item0	2\$	1\$	NaN
1	Item0	Bronze	2\$	2€	Item1	NaN	3\$	4\$
2	Item1	Gold	3\$	3€				
3	Item1	Silver	4\$	4€				

d.pivot(index='Item', columns='CType', values='USD')

20

Use this space to take notes:

Slide 21

## ► DataFrame/Series Operations ➞

### DataFrame.pivot\_table

ix	Item	CType	USD	EU	ix=Item	Bronze	Gold	Silver
0	Item0	Gold	1	1	Item0	2	2 = mean(1,3)	NaN
1	Item0	Bronze	2	2	Item1	NaN	NaN	4
2	Item0	Gold	3	3				
3	Item1	Silver	4	4				

d.pivot\_table(index='Item', columns='CType', values='USD', aggfunc=np.mean)

21

Use this space to take notes:

Slide 22

## ► DataFrame/Series Operations ➤

### DataFrame.pivot vs DataFrame.pivot\_table

- ▶ **Pivot\_table** is a generalization of **pivot** that can handle duplicate values for **one pivoted index/column pair**.
- ▶ **Pivot\_table** will only allow **numeric types** as "values=", whereas **pivot** will take **string types** as "values=".

22

Use this space to take notes:

## Slide 23

## ► DataFrame/Series Operations ➤

### DataFrame.pivot vs DataFrame.pivot\_table

gender	sport	age	height	weight	
0	male	baseball	21	72	200
1	female	basketball	21	72	130
2	female	golf	22	73	150
3	male	basketball	20	75	175
4	female	archery	22	68	170

Pivot and pivot\_table may only exhibit the same functionality if the data allows.

df.pivot_table(index='gender', columns='sport', values=['age','height','weight'], aggfunc='mean')				
gender	age	height	weight	
sport	archery	baseball	basketball	golf
female	22.0	NaN	21.0	72.0
male	NaN	21.0	20.0	75.0
				NaN
				200.0
				175.0
				NaN

df.pivot(index='gender', columns='sport', values=['age','height','weight'])				
gender	age	height	weight	
sport	archery	baseball	basketball	golf
female	22.0	NaN	21.0	72.0
male	NaN	21.0	20.0	75.0
				NaN
				200.0
				175.0
				NaN

23

Use this space to take notes:

## Slide 24

## ► DataFrame/Series Operations ➤

### DataFrame.pivot vs DataFrame.pivot\_table

gender	sport	age	height	weight	
0	male	baseball	21	72	200
1	female	baseball	21	72	130
2	female	golf	22	73	150
3	male	baseball	20	75	175
4	female	baseball	22	68	170

gender	baseball	basketball	golf	baseball	basketball	golf	baseball	basketball	golf
female	NaN	21.5	22.0	NaN	70.0	73.0	NaN	150.0	150.0
male	21.0	20.0	NaN	72.0	75.0	NaN	200.0	175.0	NaN

age	height	weight
0	21.0	NaN
1	NaN	21.0
2	NaN	22.0
3	NaN	20.0
4	NaN	22.0

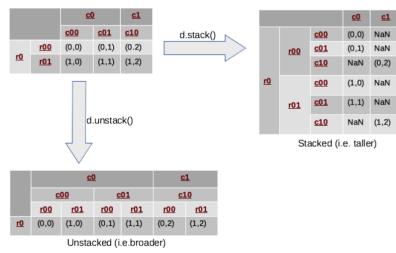
If there are duplicate entries possible from the index(es) of interest you will need to aggregate the data in pivot\_table, not pivot.

Use this space to take notes:

## Slide 25

## ► DataFrame/Series Operations ➤

### DataFrame.stack vs DataFrame.unstack



Use this space to take notes:

## Slide 26

## Data Analysis with Python



let's start the  
hands-on phase

28

Use this space to take notes:

### Slide 27

### Your Response

Did you find this lesson interesting and challenging?

Too hard      Just right      Too easy

Students, drag the icon!      Pear Deck Interactive Slide  
Do not remove this bar

Did you find this lesson interesting and challenging?

Too hard      Just right      Too easy

Students, drag the icon!      Pear Deck Interactive Slide  
Do not remove this bar

Use this space to take notes:

### Slide 28

# THANKS!

**Any questions?**

You can find us at:

steve\_w@clarusway.com  
michael\_g@clarusway.com

CLARUSWAY®  
WAY TO REINVENT YOURSELF



28

Use this space to take notes: