Session5-6-Pandas (Groupby, Useful Operations, Pivot, Stack)

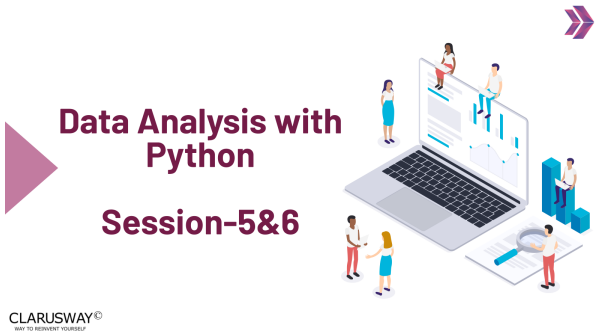
DS 12/22_EU_DAwP_S5&6_Groupby_UsefullOperations_6thAgu22
Training Clarusway
Pear Deck - August 6, 2022 at 0:51AM

Part 1 - Summary

Use this space to summarize your thoughts on the lesson

Part 2 - Responses

Slide 1



Use this space to take notes:

Slide 2

pandas

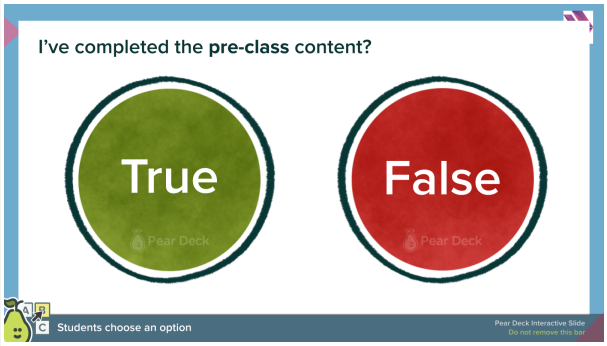
CLARUSWAY©

Use this space to take notes:

Slide 3

Table of Contents

Use this space to take notes:

Slide 4	Your Response
	You Chose • **False** Other Choices • True

Use this space to take notes:

Slide 5	Your Response
	

Use this space to take notes:

Slide 6

Basic Aggregation Methods

- count()
- mean()
- median()
- idxmin()
- idxmax()
- corr()
- min()
- max()
- std()
- var()
- sum()
- describe()

6

Use this space to take notes:

Slide 7

Grouping Data (DataFrame.groupby)

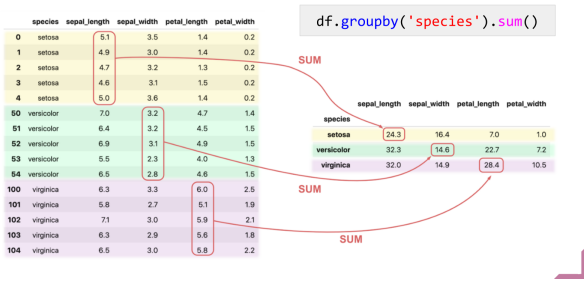
- DataFrame.groupby operation involves some combination of **splitting the object**, **applying a function**, and **combining the results**.
- This can be used to group large amounts of data and compute operations on these groups.

7

Use this space to take notes:

Slide 8

Grouping Data (DataFrame.groupby())



	species	sepal_length	sepal_width	petal_length	petal_width
0	setosa	5.1	3.5	1.4	0.2
1	setosa	4.9	3.0	1.4	0.2
2	setosa	4.7	3.2	1.3	0.2
3	setosa	4.6	3.1	1.5	0.2
4	setosa	5.0	3.6	1.4	0.2
50	versicolor	7.0	3.2	4.7	1.4
51	versicolor	6.4	3.2	4.5	1.5
52	versicolor	6.9	3.1	4.9	1.5
53	versicolor	5.5	2.3	4.0	1.3
54	versicolor	6.5	2.8	4.6	1.5
100	virginica	6.3	3.3	6.0	2.5
101	virginica	5.8	2.7	5.1	1.9
102	virginica	7.1	3.0	5.9	2.1
103	virginica	6.3	2.9	5.6	1.8
104	virginica	6.5	3.0	5.8	2.2

species	sepal_length	sepal_width	petal_length	petal_width
setosa	24.3	16.4	7.0	1.0
versicolor	32.3	14.6	22.7	7.2
virginica	32.0	14.9	28.4	10.5

Use this space to take notes:

Slide 9

Useful Operations

DataFrame.aggregate/agg

```
>>> df = pd.DataFrame([[1, 2, 3],
...                    [4, 5, 6],
...                    [7, 8, 9],
...                    [np.nan, np.nan, np.nan]],
...                   columns=['A', 'B', 'C'])
```

Aggregate these functions over the rows.

```
>>> df.agg(['sum', 'min'])
        A     B     C
sum  12.0  15.0  18.0
min   1.0   2.0   3.0
```

Different aggregations per column.

```
>>> df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
        A    B
sum  12.0  NaN
min   1.0  2.0
max   NaN  8.0
```

Aggregate different functions over the columns and rename the index

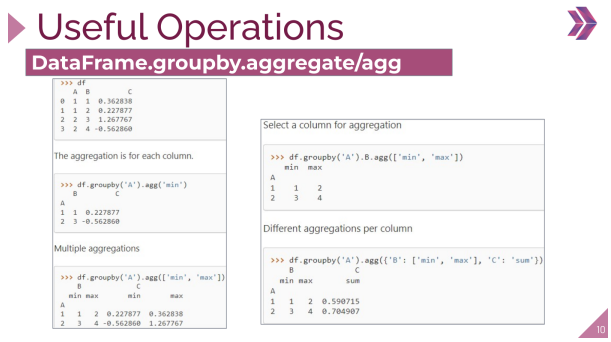
```
>>> df.agg(x=('A', max), y=('B', 'min'), z=('C', np.mean))
     A    B    C
x  7.0  NaN  NaN
y  NaN  2.0  NaN
z  NaN  NaN  6.0
```

Aggregate over the columns.

```
>>> df.agg("mean", axis="columns")
0    2.0
1    5.0
2    8.0
3    NaN
```

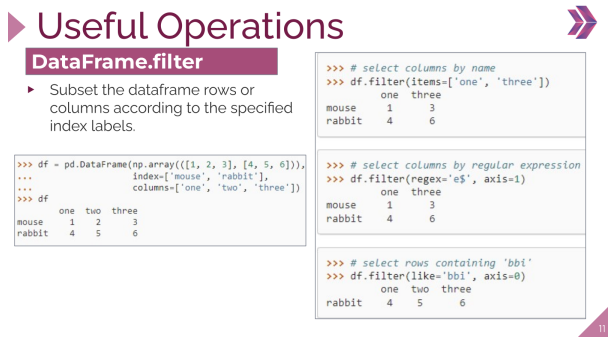
Use this space to take notes:

Slide 10



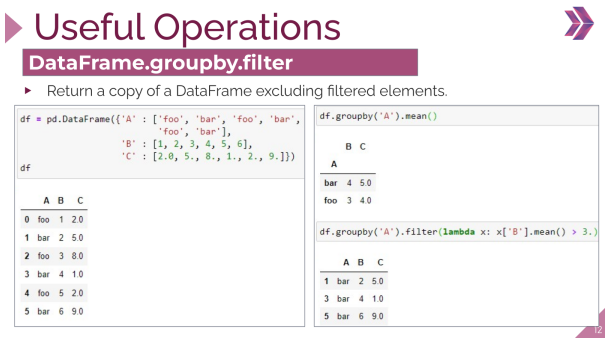
Use this space to take notes:

Slide 11



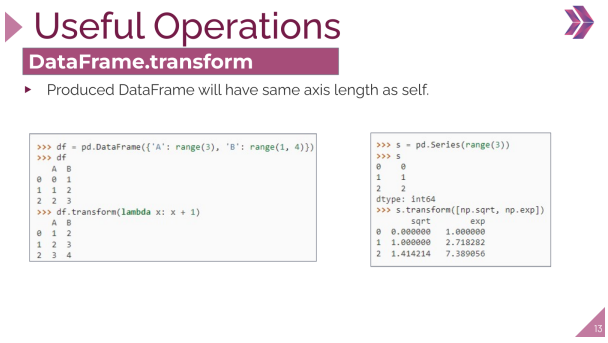
Use this space to take notes:

Slide 12



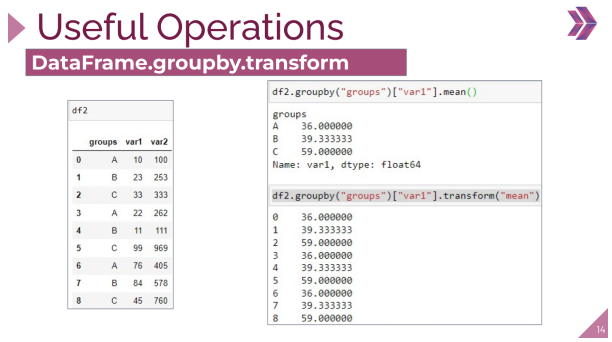
Use this space to take notes:

Slide 13



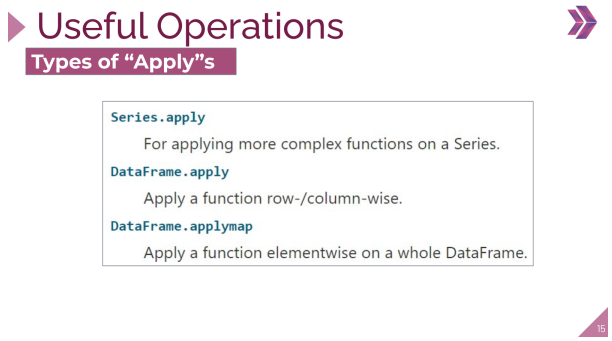
Use this space to take notes:

Slide 14



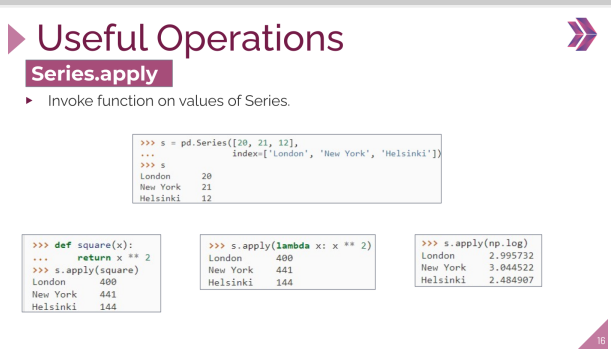
Use this space to take notes:

Slide 15



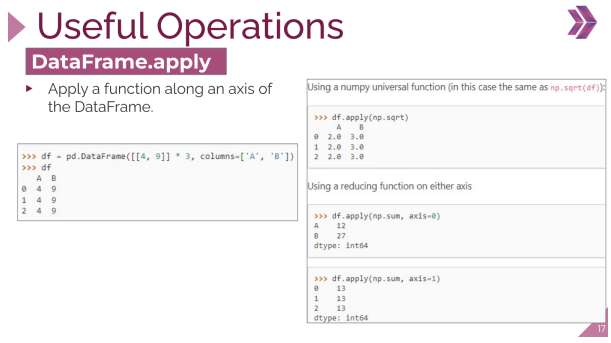
Use this space to take notes:

Slide 16



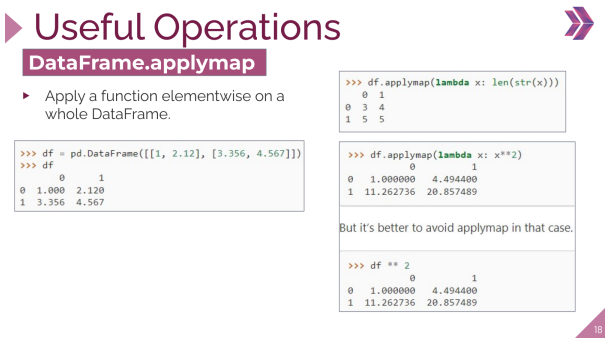
Use this space to take notes:

Slide 17



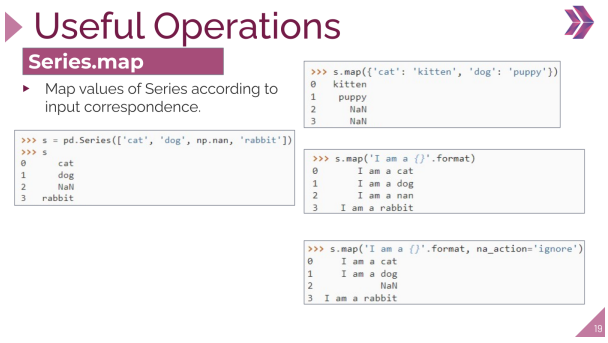
Use this space to take notes:

Slide 18



Use this space to take notes:

Slide 19



Use this space to take notes:

Slide 20

Useful 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**.

20

Use this space to take notes:

Slide 21

Useful Operations

apply() vs transform()

```
df5.apply(lambda x: x+10)
```

	A	B
0	11	20
1	12	30
2	13	40

	A	B
0	1	10
1	2	20
2	3	30

```
df5.transform(lambda x: x+10)
```

	A	B
0	11	20
1	12	30
2	13	40

21

Use this space to take notes:

Slide 22

Useful Operations

apply() vs transform()

- **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

Use this space to take notes:

Slide 23

DataFrame/Series Operations

apply() vs transform()

- **transform()** cannot produce aggregated results.

	A	B
0	1	10
1	2	20
2	3	30

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

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


Use this space to take notes:

Slide 24

Useful 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	1	10
1	2	20
2	3	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
```


Use this space to take notes:

Slide 25

Useful 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
1	b	1	2
2	c	2	3
3	a	3	1
4	b	4	2
5	c	5	3
6	a	6	1
7	b	7	2
8	c	8	3

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

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


Use this space to take notes:

Slide 26

Useful Operations

apply() vs transform()

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

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

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

key
a     9
b    12
c    15
```

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

key
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
```

26

Use this space to take notes:

Slide 27

Useful 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
0	Item0	Gold	1$	1€
1	Item0	Bronze	2$	2€
2	Item1	Gold	3$	3€
3	Item1	Silver	4$	4€

ix=Item	Bronze	Gold	Silver
Item0	2$	1$	NaN
Item1	NaN	3$	4$

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

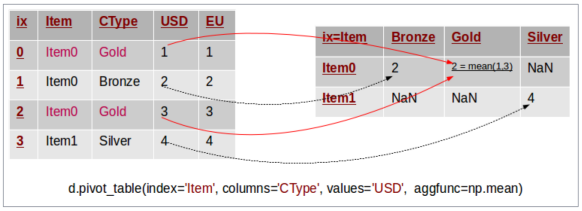
27

Use this space to take notes:

Slide 28

Useful Operations

DataFrame.pivot_table



28

Use this space to take notes:

Slide 29

Useful Operations

.pivot() vs .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=".

29

Use this space to take notes:

Slide 30

Useful Operations

.pivot() vs .pivot_table()

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

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')
```

	age				height				weight			
sport	archery	baseball	basketball	golf	archery	baseball	basketball	golf	archery	baseball	basketball	golf
gender												
female	22.0	NaN	21.0	22.0	68.0	NaN	72.0	73.0	170.0	NaN	130.0	150.0
male	NaN	21.0	20.0	NaN	NaN	72.0	75.0	NaN	NaN	200.0	175.0	NaN

```
df.pivot(index='gender', columns='sport',
         values=['age','height','weight'])
```

	age				height				weight			
sport	archery	baseball	basketball	golf	archery	baseball	basketball	golf	archery	baseball	basketball	golf
gender												
female	22.0	NaN	21.0	22.0	68.0	NaN	72.0	73.0	170.0	NaN	130.0	150.0
male	NaN	21.0	20.0	NaN	NaN	72.0	75.0	NaN	NaN	200.0	175.0	NaN

30

Use this space to take notes:

Slide 31

Useful Operations

.pivot() vs .pivot_table()

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

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

```
df.pivot_table(index='gender', columns='sport',
               values=['age','height','weight'],
               aggfunc='mean')
```

	age			height			weight		
sport	baseball	basketball	golf	baseball	basketball	golf	baseball	basketball	golf
gender									
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

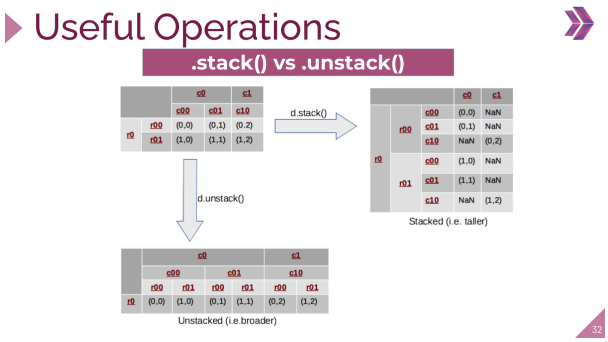
```
df.pivot(columns='sport', values=['age','height','weight'])
```

	age			height			weight		
sport	baseball	basketball	golf	baseball	basketball	golf	baseball	basketball	golf
0	21.0	NaN	NaN	72.0	NaN	NaN	200.0	NaN	NaN
1	NaN	21.0	NaN	NaN	72.0	NaN	NaN	130.0	NaN
2	NaN	NaN	22.0	NaN	NaN	73.0	NaN	NaN	150.0
3	NaN	20.0	NaN	NaN	75.0	NaN	NaN	175.0	NaN
4	NaN	22.0	NaN	NaN	68.0	NaN	NaN	170.0	NaN

31

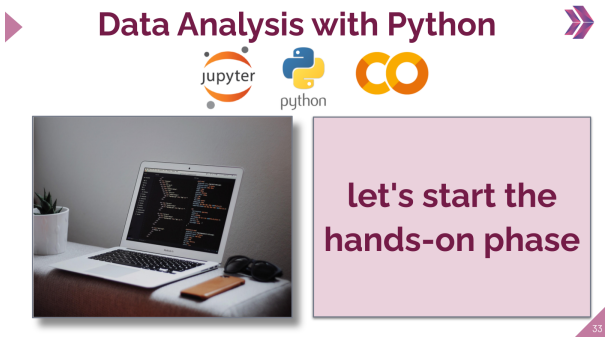
Use this space to take notes:

Slide 32



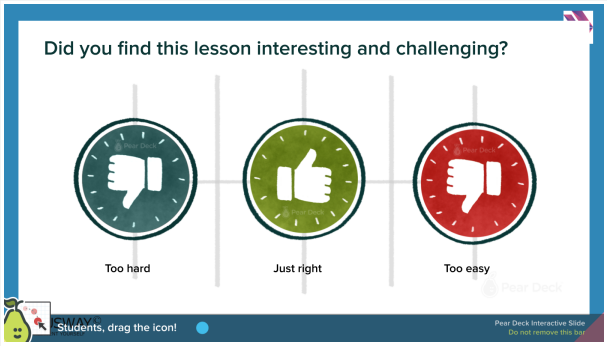
Use this space to take notes:

Slide 33



Use this space to take notes:

Slide 34	Your Response

Slide 34	Your Response
	

Use this space to take notes:

Slide 35

Use this space to take notes: