Applied Data Science

Session 3: Data Manipulation

Dr. Soharab Hossain Shaikh

1

Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis.

In particular, it offers data structures and operations for manipulating numerical tables and time series.

It is free software released under the three-clause BSD license.

2

Pandas Data Structures - Series

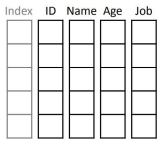
`pandas.Series`

homogeneous 1-dimensional array, each element has the same dtype

3

Pandas Data Structures - DataFrame

`pandas.DataFrame`



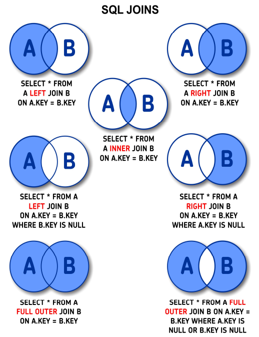
Collection of named Series with index.

4

Operations on Pandas DataFrame

5

SQL Join


SQL JOINS
A
B
SELECT * FROM A LEFT JOIN B ON A.KEY = B.KEY
SELECT * FROM A RIGHT JOIN B ON A.KEY = B.KEY
SELECT * FROM A INNER JOIN B ON A.KEY = B.KEY
SELECT * FROM A LEFT JOIN B ON A.KEY = B.KEY WHERE B.KEY IS NULL
SELECT * FROM A RIGHT JOIN B ON A.KEY = B.KEY WHERE A.KEY IS NULL
SELECT * FROM A FULL OUTER JOIN B ON A.KEY = B.KEY
SELECT * FROM A FULL OUTER JOIN B ON A.KEY = B.KEY WHERE A.KEY IS NULL OR B.KEY IS NULL

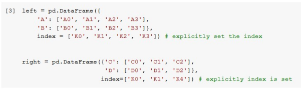
6

Join on Pandas DataFrame

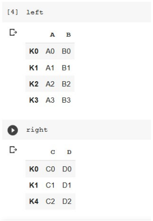
- We perform join operations on two dataframes to combine them.
- Join will always try to match the **index of the second dataframe with the index (or specified column) of the first dataframe**.
- **join** takes an optional on argument which may be a column or multiple column names, which specifies that the passed DataFrame is to be aligned on that column in the DataFrame.

7

Join



```
[3]  left = pd.DataFrame({
         'A': ['A0', 'A1', 'A2', 'A3'],
         'B': ['B0', 'B1', 'B2', 'B3']},
         index = ['K0', 'K1', 'K2', 'K3']) # explicitly set the index


     right = pd.DataFrame({'C': ['C0', 'C1', 'C2'],
                           'D': ['D0', 'D1', 'D2']},
                          index=['K0', 'K1', 'K4']) # explicitly index is set
```



```
[4]  left
```

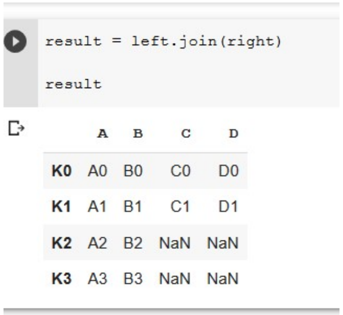
	A	B
K0	A0	B0
K1	A1	B1
K2	A2	B2
K3	A3	B3

```
right
```

	C	D
K0	C0	D0
K1	C1	D1
K4	C2	D2

8

Join

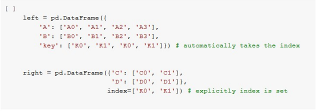


```
result = left.join(right)

result
```

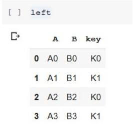
	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	C1	D1
K2	A2	B2	NaN	NaN
K3	A3	B3	NaN	NaN

9

Join

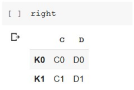


```
[ ]
    left = pd.DataFrame({
        'A': ['A0', 'A1', 'A2', 'A3'],
        'B': ['B0', 'B1', 'B2', 'B3'],
        'key': ['K0', 'K1', 'K0', 'K1']}) # automatically takes the index


    right = pd.DataFrame({'C': ['C0', 'C1'],
                           'D': ['D0', 'D1']},
                          index=['K0', 'K1']) # explicitly index is set
```



```
[ ] left
```

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K0
3	A3	B3	K1

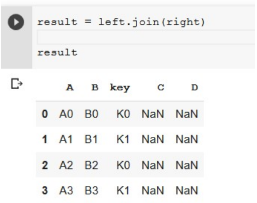


```
[ ] right
```

	C	D
K0	C0	D0
K1	C1	D1

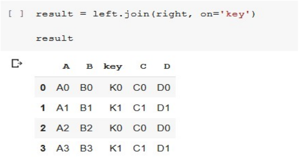
10

Join



```
result = left.join(right)

result
```

	A	B	key	C	D
0	A0	B0	K0	NaN	NaN
1	A1	B1	K1	NaN	NaN
2	A2	B2	K0	NaN	NaN
3	A3	B3	K1	NaN	NaN



```
[ ] result = left.join(right, on='key')

    result
```

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K0	C0	D0
3	A3	B3	K1	C1	D1

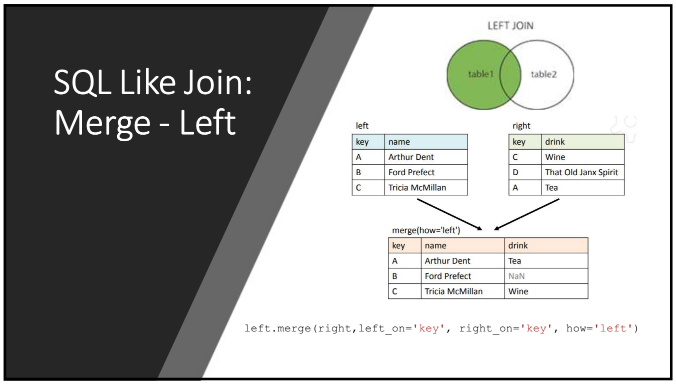
11

What is Merge in Pandas?

- Working with 2-D data having rows and columns in Pandas dataframe - often we want a better understanding of the data by merging the common values of two dataframe (somewhat identical to **join** operation).
- We use a function called **merge()** in pandas that takes the commonalities of two dataframes just like we do in SQL.
- The Syntax for merge in pandas is:

DataFrame.merge(self, right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)

12



SQL Like Join: Merge - Left

left

key	name
A	Arthur Dent
B	Ford Prefect
C	Tricia McMillan

right

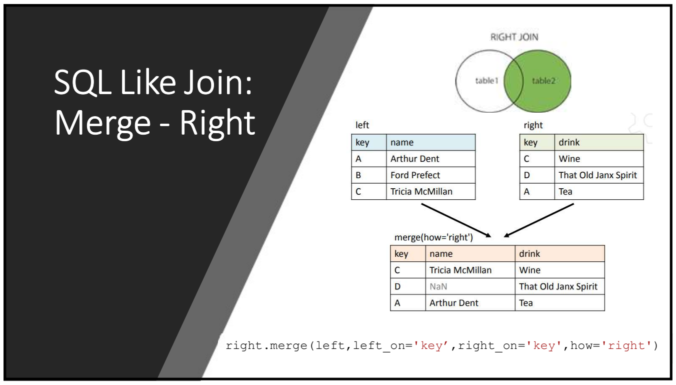
key	drink
C	Wine
D	That Old Janx Spirit
A	Tea

merge(how='left')

key	name	drink
A	Arthur Dent	Tea
B	Ford Prefect	NaN
C	Tricia McMillan	Wine

```
left.merge(right,left_on='key', right_on='key', how='left')
```

13



SQL Like Join: Merge - Right

left

key	name
A	Arthur Dent
B	Ford Prefect
C	Tricia McMillan

right

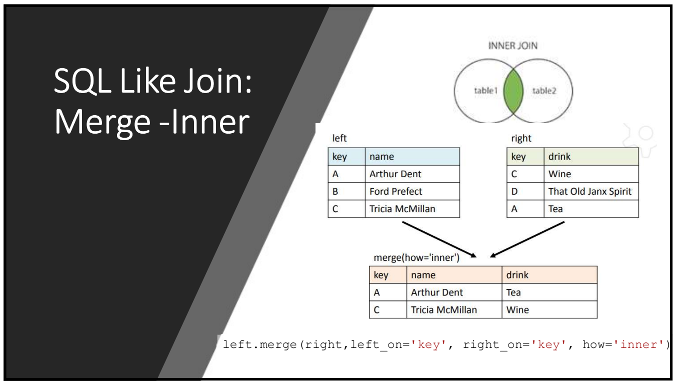
key	drink
C	Wine
D	That Old Janx Spirit
A	Tea

merge(how='right')

key	name	drink
C	Tricia McMillan	Wine
D	NaN	That Old Janx Spirit
A	Arthur Dent	Tea

```
right.merge(left,left_on='key’,right_on='key',how='right')
```

14



SQL Like Join: Merge -Inner

left

key	name
A	Arthur Dent
B	Ford Prefect
C	Tricia McMillan

right

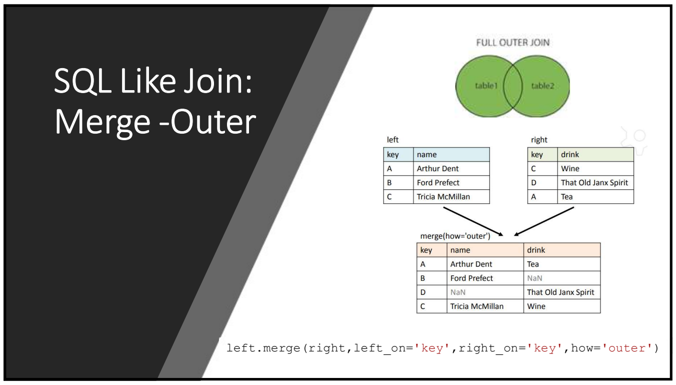
key	drink
C	Wine
D	That Old Janx Spirit
A	Tea

merge(how='inner')

key	name	drink
A	Arthur Dent	Tea
C	Tricia McMillan	Wine

```
left.merge(right,left_on='key', right_on='key', how='inner')
```

15



SQL Like Join: Merge -Outer

left

key	name
A	Arthur Dent
B	Ford Prefect
C	Tricia McMillan

right

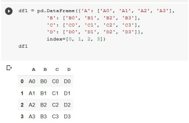
key	drink
C	Wine
D	That Old Janx Spirit
A	Tea

merge(how='outer')

key	name	drink
A	Arthur Dent	Tea
B	Ford Prefect	NaN
D	NaN	That Old Janx Spirit
C	Tricia McMillan	Wine

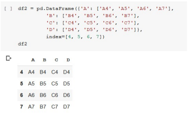
```
left.merge(right,left_on='key',right_on='key',how='outer')
```

16



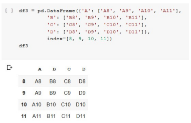
```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                    index=[0, 1, 2, 3])
df1
```

```
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
```



```
[ ] df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                        'B': ['B4', 'B5', 'B6', 'B7'],
                        'C': ['C4', 'C5', 'C6', 'C7'],
                        'D': ['D4', 'D5', 'D6', 'D7']},
                        index=[4, 5, 6, 7])
    df2
```

```
    A   B   C   D
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
```

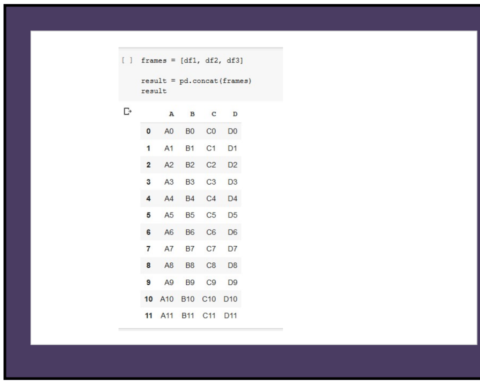


```
[ ] df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                        'B': ['B8', 'B9', 'B10', 'B11'],
                        'C': ['C8', 'C9', 'C10', 'C11'],
                        'D': ['D8', 'D9', 'D10', 'D11']},
                        index=[8, 9, 10, 11])
    df3
```

```
      A    B    C    D
8    A8   B8   C8   D8
9    A9   B9   C9   D9
10  A10  B10  C10  D10
11  A11  B11  C11  D11
```

Concatenation

17



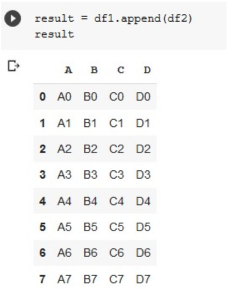
```
[ ] frames = [df1, df2, df3]

    result = pd.concat(frames)
    result
```

```
      A    B    C    D
0    A0   B0   C0   D0
1    A1   B1   C1   D1
2    A2   B2   C2   D2
3    A3   B3   C3   D3
4    A4   B4   C4   D4
5    A5   B5   C5   D5
6    A6   B6   C6   D6
7    A7   B7   C7   D7
8    A8   B8   C8   D8
9    A9   B9   C9   D9
10  A10  B10  C10  D10
11  A11  B11  C11  D11
```

Concat

18

Append



```
result = df1.append(df2)
result
```

```
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
2  A2  B2  C2  D2
3  A3  B3  C3  D3
4  A4  B4  C4  D4
5  A5  B5  C5  D5
6  A6  B6  C6  D6
7  A7  B7  C7  D7
```

19

Pandas Index and MultiIndex

Go to the Coding Demo…

MultiIndex.ipynb
Create_MultiIndex_Access_Data.ipynb

20

Hierarchical Representation with Pivot

21

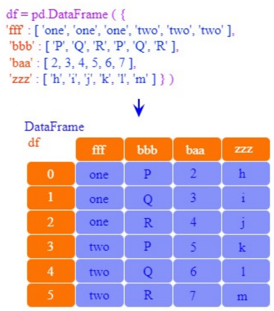
Pivot

- The pivot() function is used to reshape a given DataFrame organized by given index / column values.
- This function does not support data aggregation, multiple values will result in a MultiIndex in the columns.

- **Syntax:**
- **DataFrame.pivot(self, index=None, columns=None, values=None)**

22

Pivot



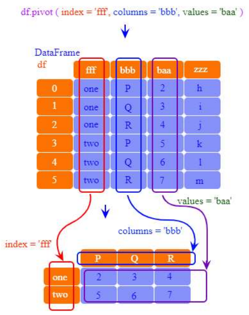
```
df = pd.DataFrame ( {
'fff' : [ 'one', 'one', 'one', 'two', 'two', 'two' ],
 'bbb' : [ 'P', 'Q', 'R', 'P', 'Q', 'R' ],
 'baa' : [ 2, 3, 4, 5, 6, 7 ],
 'zzz' : [ 'h', 'i', 'j', 'k', 'l', 'm' ] } )
```

DataFrame
df

	fff	bbb	baa	zzz
0	one	P	2	h
1	one	Q	3	i
2	one	R	4	j
3	two	P	5	k
4	two	Q	6	l
5	two	R	7	m

23

Pivot



```
df.pivot ( index = 'fff', columns = 'bbb', values = 'baa' )
```

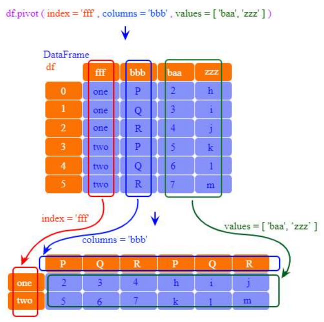
DataFrame
df

	fff	bbb	baa	zzz
0	one	P	2	h
1	one	Q	3	i
2	one	R	4	j
3	two	P	5	k
4	two	Q	6	l
5	two	R	7	m

	P	Q	R
one	2	3	4
two	5	6	7

24

Pivot



```
df.pivot ( index = 'fff' , columns = 'bbb' , values = [ 'baa', 'zzz' ] )
```

DataFrame
df

	fff	bbb	baa	zzz
0	one	P	2	h
1	one	Q	3	i
2	one	R	4	j
3	two	P	5	k
4	two	Q	6	l
5	two	R	7	m

	P	Q	R	P	Q	R
one	2	3	4	h	i	j
two	5	6	7	k	l	m

25

Hierarchical Aggregation with Pivot Table

26

What is a Pivot Table in Pandas?

- The pivot_table() function is used to create a spreadsheet-style pivot table as a DataFrame.
- The levels in the pivot table will be stored in MultiIndex objects (hierarchical indexes) on the index and columns of the result DataFrame.
- Those familiar using Microsoft excel are aware of pivot tables as it is the backbone for business analysis because it provides a fold of the data provided in new dimensions making data look more summarized and classified.

- **Syntax:**
- **DataFrame.pivot_table(self, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False)**

27

Pivot Table

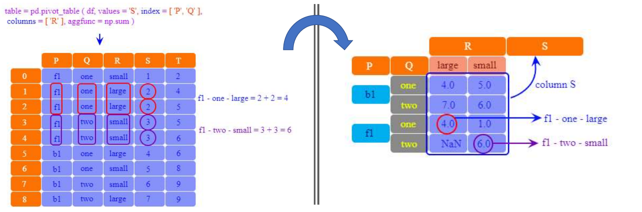
```
df = pd.DataFrame ( {
"P" : [ "f1", "f1", "f1", "f1", "f1", "b1", "b1", "b1", "b1" ],
 "Q" : [ "one", "one", "one", "two", "two", "one", "one", "two", "two" ],
 "R" : [ "small", "large", "large", "small", "small", "large", "small", "small", "large" ]
 "S" : [ 1, 2, 2, 3, 3, 4, 5, 6, 7 ],
 "T" : [ 2, 4, 5, 5, 6, 6, 8, 9, 9 ] } )
```

	P	Q	R	S	T
0	f1	one	small	1	2
1	f1	one	large	2	4
2	f1	one	large	2	5
3	f1	two	small	3	5
4	f1	two	small	3	6
5	b1	one	large	4	6
6	b1	one	small	5	8
7	b1	two	small	6	9
8	b1	two	large	7	9

28

Pivot Table

table = pd.pivot_table (df, values = 'S', index = ['P', 'Q'], columns = ['R'], aggfunc = np.sum)



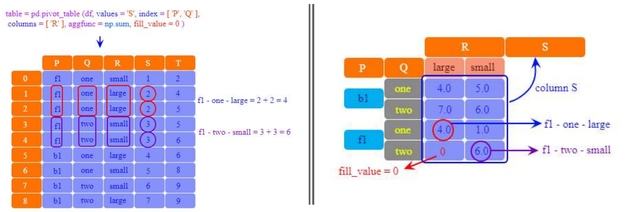
	P	Q	R	S	T
0	f1	one	small	1	2
1	f1	one	large	2	4
2	f1	one	large	2	5
3	f1	two	small	3	5
4	f1	two	small	3	6
5	b1	one	large	4	6
6	b1	one	small	5	8
7	b1	two	small	6	9
8	b1	two	large	7	9

		R		S
P	Q	large	small	
b1	one	4.0	5.0	
	two	7.0	6.0	
f1	one	4.0	1.0	
	two	NaN	6.0	

29

Pivot Table

table = pd.pivot_table (df, values = 'S', index = ['P', 'Q'], columns = ['R'], aggfunc = np.sum, fill_value = 0)



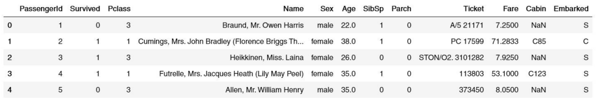
	P	Q	R	S	T
0	f1	one	small	1	2
1	f1	one	large	2	4
2	f1	one	large	2	5
3	f1	two	small	3	5
4	f1	two	small	3	6
5	b1	one	large	4	6
6	b1	one	small	5	8
7	b1	two	small	6	9
8	b1	two	large	7	9

		R		S
P	Q	large	small	
b1	one	4.0	5.0	
	two	7.0	6.0	
f1	one	4.0	1.0	
	two	0	6.0	

30

Pivot Table

```
import pandas as pd
df = pd.read_csv('Titanic_train.csv')
df.head()
```



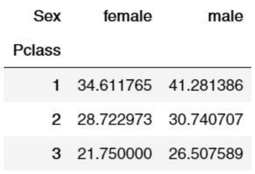
```
   PassengerId  Survived  Pclass                                               Name     Sex   Age  SibSp  Parch            Ticket     Fare  Cabin  Embarked
0            1         0       3                            Braund, Mr. Owen Harris    male  22.0      1      0         A/5 21171   7.2500    NaN         S
1            2         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1      0          PC 17599  71.2833    C85         C
2            3         1       3                             Heikkinen, Miss. Laina  female  26.0      0      0  STON/O2. 3101282   7.9250    NaN         S
3            4         1       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1      0            113803  53.1000   C123         S
4            5         0       3                           Allen, Mr. William Henry    male  35.0      0      0            373450   8.0500    NaN         S
```

31

Pivot Table

- Let's say that we want to find out the mean age of both male and female based on the Pclass they were travelling on, so how do we tell pandas to present us the desired dataframe?
- The best way to do that is by using the **pivot_table method**.

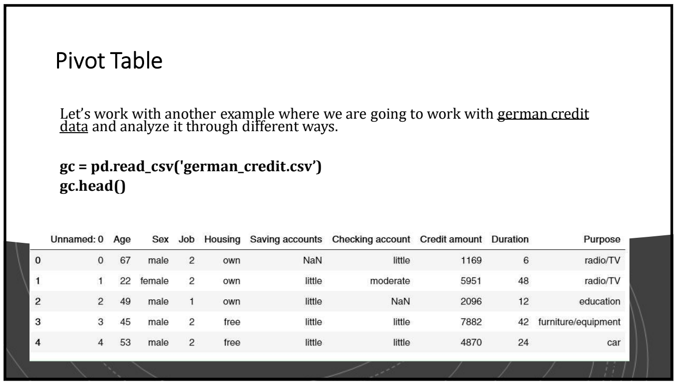
- **df.pivot_table**(index="Pclass", columns = "Sex" , values="Age", aggfunc='mean')



```
Sex        female       male
Pclass
1       34.611765  41.281386
2       28.722973  30.740707
3       21.750000  26.507589
```

Result folds the data based on what we want and displays a new dataframe.

32



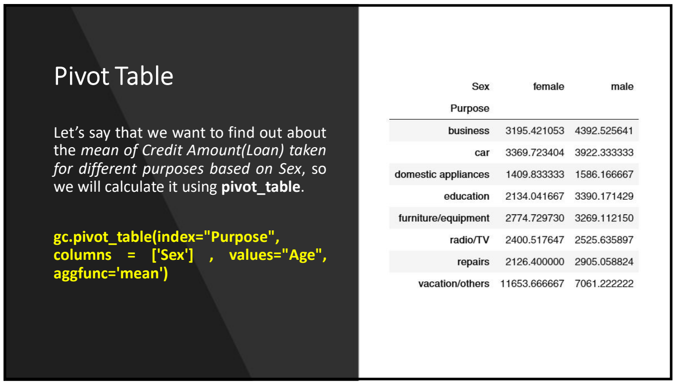
Pivot Table

Let's work with another example where we are going to work with german credit data and analyze it through different ways.

gc = pd.read_csv('german_credit.csv')
gc.head()

	Unnamed: 0	Age	Sex	Job	Housing	Saving accounts	Checking account	Credit amount	Duration	Purpose
0	0	67	male	2	own	NaN	little	1169	6	radio/TV
1	1	22	female	2	own	little	moderate	5951	48	radio/TV
2	2	49	male	1	own	little	NaN	2096	12	education
3	3	45	male	2	free	little	little	7882	42	furniture/equipment
4	4	53	male	2	free	little	little	4870	24	car

33



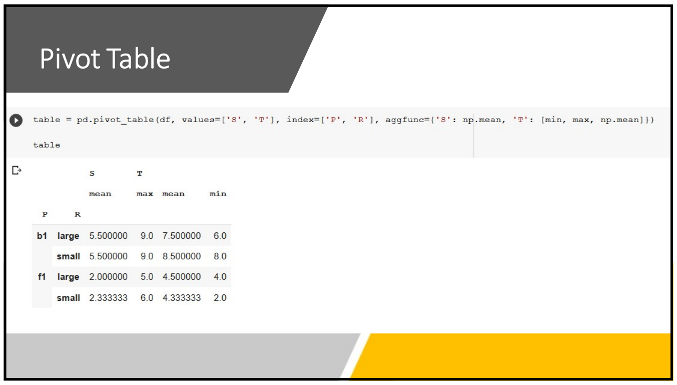
Pivot Table

Let's say that we want to find out about the *mean of Credit Amount(Loan) taken for different purposes based on Sex*, so we will calculate it using **pivot_table**.

gc.pivot_table(index="Purpose", columns = ['Sex'] , values="Age", aggfunc='mean')

Sex	female	male
Purpose		
business	3195.421053	4392.525641
car	3369.723404	3922.333333
domestic appliances	1409.833333	1586.166667
education	2134.041667	3390.171429
furniture/equipment	2774.729730	3269.112150
radio/TV	2400.517647	2525.635897
repairs	2126.400000	2905.058824
vacation/others	11653.666667	7061.222222

34

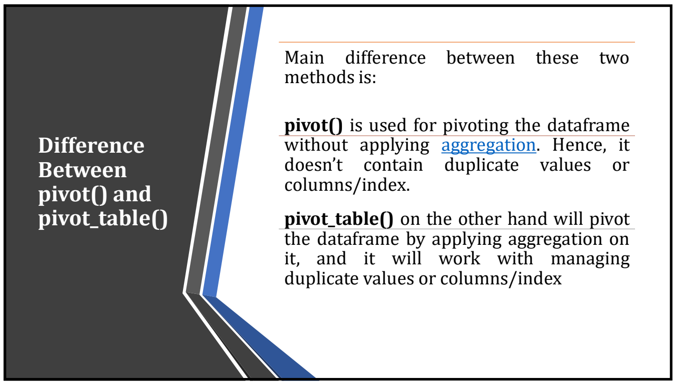


Pivot Table

```
table = pd.pivot_table(df, values=['S', 'T'], index=['P', 'R'], aggfunc={'S': np.mean, 'T': [min, max, np.mean]})
table
```

		S	T		
		mean	max	mean	min
P	R				
b1	large	5.500000	9.0	7.500000	6.0
	small	5.500000	9.0	8.500000	8.0
f1	large	2.000000	5.0	4.500000	4.0
	small	2.333333	6.0	4.333333	2.0

35



Difference Between pivot() and pivot_table()

Main difference between these two methods is:

pivot() is used for pivoting the dataframe without applying aggregation. Hence, it doesn't contain duplicate values or columns/index.

pivot_table() on the other hand will pivot the dataframe by applying aggregation on it, and it will work with managing duplicate values or columns/index

36

Stacking and Unstacking

37

Stacking

- The stack() function is used to stack the prescribed level(s) from columns to index.
- Return a **reshaped DataFrame** or Series having a **multi-level index** with one or more **new inner-most levels** compared to the current DataFrame.
- The **new inner-most levels** are created by pivoting the **columns** of the current dataframe:

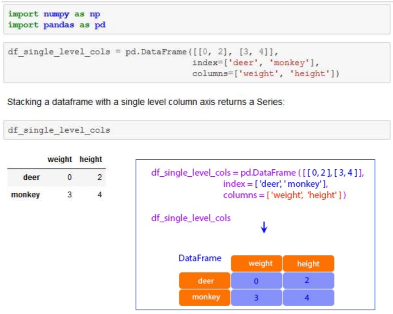
> if the columns have a single level, the output is a Series;

> if the columns have multiple levels, the new index level(s) is (are) taken from the prescribed level(s) and the output is a DataFrame.

- The new index levels are sorted.
- **Syntax: DataFrame.stack(self, level=-1, dropna=True)**

38

Stacking



```
import numpy as np
import pandas as pd
```

```
df_single_level_cols = pd.DataFrame([[0, 2], [3, 4]],
                                     index=['deer', 'monkey'],
                                     columns=['weight', 'height'])
```

Stacking a dataframe with a single level column axis returns a Series:

```
df_single_level_cols
```

	weight	height
deer	0	2
monkey	3	4

```
df_single_level_cols = pd.DataFrame ( [ [ 0, 2 ], [ 3, 4 ] ],
                                      index = [ 'deer', ' monkey' ],
                                      columns = [ 'weight',  'height' ] )
```

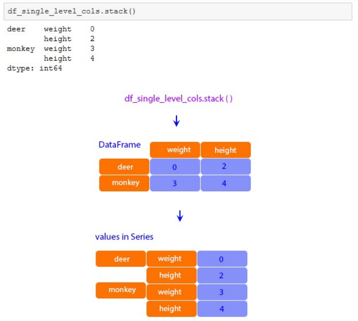
df_single_level_cols

↓

DataFrame	weight	height
deer	0	2
monkey	3	4

39

Stacking



```
df_single_level_cols.stack()
```

```
deer    weight    0
        height    2
monkey  weight    3
        height    4
dtype: int64
```

df_single_level_cols.stack ( )

↓

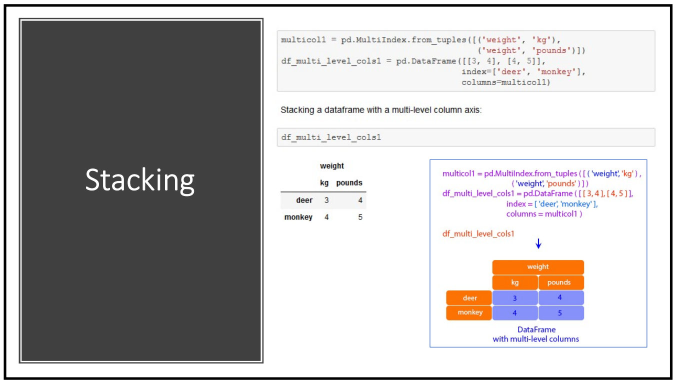
DataFrame	weight	height
deer	0	2
monkey	3	4

↓

values in Series

deer	weight	0
	height	2
monkey	weight	3
	height	4

40



Stacking

```
multicol1 = pd.MultiIndex.from_tuples([('weight', 'kg'),
                                       ('weight', 'pounds')])
df_multi_level_cols1 = pd.DataFrame([[3, 4], [4, 5]],
                                    index=['deer', 'monkey'],
                                    columns=multicol1)
```

Stacking a dataframe with a multi-level column axis:

```
df_multi_level_cols1
```

	weight	
	kg	pounds
deer	3	4
monkey	4	5

```
multicol1 = pd.MultiIndex.from_tuples ( [ ( 'weight', 'kg' ) ,
                                          ( 'weight', 'pounds' ) ] )
df_multi_level_cols1 = pd.DataFrame ( [ [ 3, 4 ], [ 4, 5 ] ],
                                      index = [ 'deer', 'monkey' ],
                                      columns = multicol1 )
```

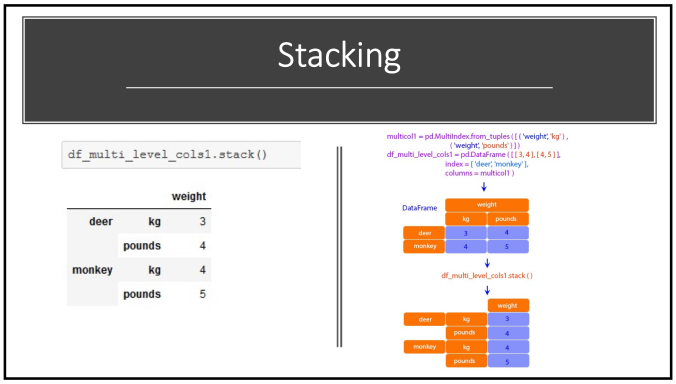
df_multi_level_cols1

↓

	weight	
	kg	pounds
deer	3	4
monkey	4	5

DataFrame
with multi-level columns

41



Stacking

```
df_multi_level_cols1.stack()
```

		weight
deer	kg	3
	pounds	4
monkey	kg	4
	pounds	5

```
multicol1 = pd.MultiIndex.from_tuples ( [ ( 'weight', 'kg' ) ,
                                          ( 'weight', 'pounds' ) ] )
df_multi_level_cols1 = pd.DataFrame ( [ [ 3, 4 ], [ 4, 5 ] ],
                                      index = [ 'deer', 'monkey' ],
                                      columns = multicol1 )
```

↓

DataFrame

	weight	
	kg	pounds
deer	3	4
monkey	4	5

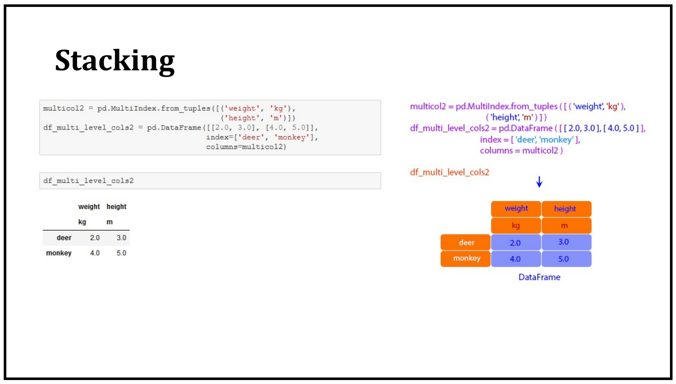
↓

df_multi_level_cols1.stack ( )

↓

		weight
deer	kg	3
	pounds	4
monkey	kg	4
	pounds	5

42



Stacking

```
multicol2 = pd.MultiIndex.from_tuples([('weight', 'kg'),
                                       ('height', 'm')])
df_multi_level_cols2 = pd.DataFrame([[2.0, 3.0], [4.0, 5.0]],
                                    index=['deer', 'monkey'],
                                    columns=multicol2)
```

```
df_multi_level_cols2
```

	weight	height
	kg	m
deer	2.0	3.0
monkey	4.0	5.0

```
multicol2 = pd.MultiIndex.from_tuples ( [ ( 'weight', 'kg' ),
                                          ( 'height', 'm' ) ] )
df_multi_level_cols2 = pd.DataFrame ( [ [ 2.0, 3.0 ], [ 4.0, 5.0 ] ],
                                      index = [ 'deer', 'monkey' ],
                                      columns = multicol2 )
```

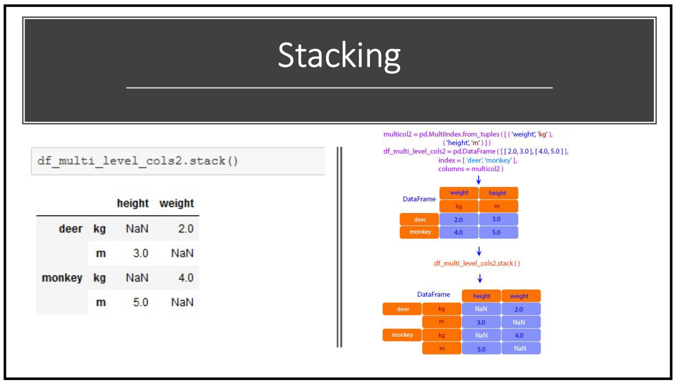
df_multi_level_cols2

↓

	weight	height
	kg	m
deer	2.0	3.0
monkey	4.0	5.0

DataFrame

43



Stacking

```
df_multi_level_cols2.stack()
```

		height	weight
deer	kg	NaN	2.0
	m	3.0	NaN
monkey	kg	NaN	4.0
	m	5.0	NaN

```
multicol2 = pd.MultiIndex.from_tuples ( [ ( 'weight', 'kg' ),
                                          ( 'height', 'm' ) ] )
df_multi_level_cols2 = pd.DataFrame ( [ [ 2.0, 3.0 ], [ 4.0, 5.0 ] ],
                                      index = [ 'deer', 'monkey' ],
                                      columns = multicol2 )
```

↓

DataFrame

	weight	height
	kg	m
deer	2.0	3.0
monkey	4.0	5.0

↓

df_multi_level_cols2.stack ( )

↓

DataFrame

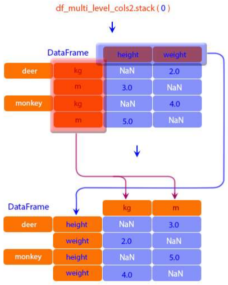
		height	weight
deer	kg	NaN	2.0
	m	3.0	NaN
monkey	kg	NaN	4.0
	m	5.0	NaN

44

Prescribing the level(s) to be stacked:

```
df_multi_level_cols2.stack(0)
```

		kg	m
deer	height	NaN	3.0
	weight	2.0	NaN
monkey	height	NaN	5.0
	weight	4.0	NaN


df_multi_level_cols2.stack (0)

DataFrame		height	weight
deer	kg	NaN	2.0
	m	3.0	NaN
monkey	kg	NaN	4.0
	m	5.0	NaN

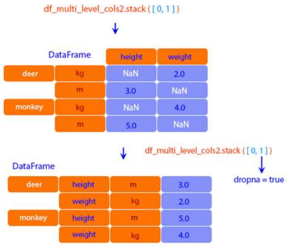
DataFrame		kg	m
deer	height	NaN	3.0
	weight	2.0	NaN
monkey	height	NaN	5.0
	weight	4.0	NaN

45

Stacking

```
df_multi_level_cols2.stack([0, 1])
```

```
deer    height  m     3.0
        weight  kg    2.0
monkey  height  m     5.0
        weight  kg    4.0
dtype: float64
```


df_multi_level_cols2.stack ([0, 1])

DataFrame		height	weight
deer	kg	NaN	2.0
	m	3.0	NaN
monkey	kg	NaN	4.0
	m	5.0	NaN

df_multi_level_cols2.stack ([0, 1])

dropna = true

DataFrame			
deer	height	m	3.0
	weight	kg	2.0
monkey	height	m	5.0
	weight	kg	4.0

46

Stacking – Missing Values

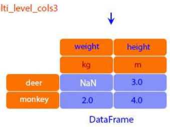
```
df_multi_level_cols3 = pd.DataFrame([[None, 2.0], [3.0, 4.0]],
                                    index=['deer', 'monkey'],
                                    columns=multicol2)
```

```
df_multi_level_cols3
```

	weight	height
	kg	m
deer	NaN	2.0
monkey	3.0	4.0

```
multicol2 = pd.MultiIndex.from_tuples ( [ ( 'weight', 'kg' ),
                                          ( 'height', 'm' ) ] )
df_multi_level_cols3 = pd.DataFrame ( [ [ None, 2.0 ], [ 3.0, 4.0 ] ],
                                      index = [ 'deer', 'monkey' ],
                                      columns = multicol2 )
```

df_multi_level_cols3



	weight	height
	kg	m
deer	NaN	3.0
monkey	2.0	4.0

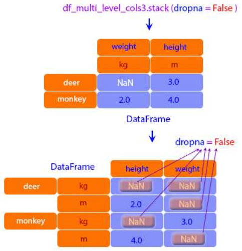
DataFrame

47

Stacking – Missing Values

```
df_multi_level_cols3.stack(dropna=False)
```

		height	weight
deer	kg	NaN	NaN
	m	2.0	NaN
monkey	kg	NaN	3.0
	m	4.0	NaN


df_multi_level_cols3.stack (dropna = False)

	weight	height
	kg	m
deer	NaN	3.0
monkey	2.0	4.0

DataFrame

dropna = False

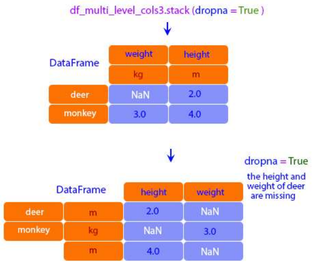
DataFrame		height	weight
deer	kg	NaN	NaN
	m	2.0	NaN
monkey	kg	NaN	3.0
	m	4.0	NaN

48

Stacking – Missing Values

```
df_multi_level_cols3.stack(dropna=True)
```

		height	weight
deer	m	2.0	NaN
monkey	kg	NaN	3.0
	m	4.0	NaN



49

Unstacking

- DataFrame - unstack() function
- Pivot a level of the (necessarily hierarchical) index labels, returning a **DataFrame** having a new level of column labels whose inner-most level consists of the pivoted index labels.
- If the index is not a MultiIndex, the output will be a Series (the analogue of stack when the columns are not a MultiIndex).
- **Syntax: DataFrame.unstack(self, level=-1, fill_value=None)**

50

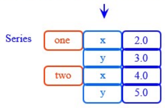
Unstacking

```
import numpy as np
import pandas as pd
```

```
index = pd.MultiIndex.from_tuples([('one', 'x'), ('one', 'y'),
                                   ('two', 'x'), ('two', 'y')])
s = pd.Series(np.arange(2.0, 6.0), index=index)
s
```

```
one  x    2.0
     y    3.0
two  x    4.0
     y    5.0
dtype: float64
```

index = pd.MultiIndex.from_tuples ([('one', 'x'), ('one', 'y'), ('two', 'x'), ('two', 'y')])
s = pd.Series (np.arange (2.0, 6.0), index = index)

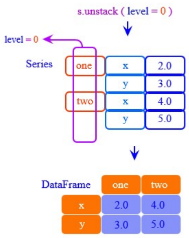


51

Unstacking

```
s.unstack(level=0)
```

	one	two
x	2.0	4.0
y	3.0	5.0

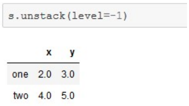


```
df = s.unstack(level=0)
df.unstack()
```

```
one  x    2.0
     y    3.0
two  x    4.0
     y    5.0
dtype: float64
```

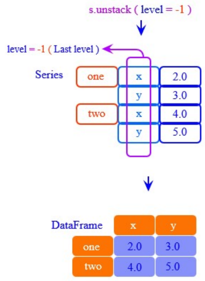
52

Unstacking



```
s.unstack(level=-1)
```

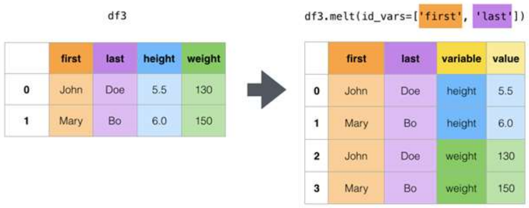
	x	y
one	2.0	3.0
two	4.0	5.0



53

Melt

Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.



df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

```
df3.melt(id_vars=['first', 'last'])
```

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150

54

What is Groupby in Pandas?

- Pandas is an awesome tool for classifying data into groups through the groupby() method.
- We can distribute the objects in pandas on any of their axis.
- In short, groupby means to analyze a pandas Series/DataFrame by some category.
- **If you have repeated categories in your dataset, then you can create groups in order to classify your data into sub-groups.**

Go to the coding Demo…….

GroupBy_Pandas.ipynb

Groupby.ipynb

55

datetime Module and TimeStamp object in Pandas

Go to the coding Demo…….

Working_with_Dates_and_Time.ipynb

DateTime_Example.ipynb

56

References

- https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html
- https://www.w3resource.com/python/python-ide.php
- https://pandas.pydata.org/pandas-docs/stable/reference/series.html

57

58

To be continued in the next session…..

59