

# Index

## DATA SCIENCE

NAME: Ravi Tega.

STD: \_\_\_\_\_ SEC: \_\_\_\_\_ ROLL: \_\_\_\_\_

S.No.	Date	<u>Pandas</u> — Particulars — <u>Pandas</u> —	Page No.	Remarks
1)		To read a data file	1-2	
2)		Pandas Series	3-4	
3)		To rename columns	5-6	
4)		To remove column/row	6	
5)		To sort a series / of	7	
6)		To filter rows by column multi-filters (isIn)	8-9	
7)		Reading limited no. columns	10	
8)		About axis, inplace	11	
9)		using string methods	12-13	
10)		To change datatype (astype)	14	
11)		using groupby	15	
12)		To Explore a series	16-17	
13)		To Handle missing values	18-19	
14)		To remove NaN (dropna)	20-21	
15)		About Pandas Index	22-23	
16)		using (.loc, .iloc)	24-28	
17)		To reduce DF size (category)	29-30	
18)		To set a (categorical order)	31-34	
19)		dealing (dummy variable)	35-36	
20)		using (to_datetime)	36-39	
21)		To find (duplicates) in col.	40-43	
22)		+ copy warning --	44-46	
23)		display options	47-49	
24)		Dummy Dataframe	50	
25)		using (apply, map, applymap)	51	
26)		About multi-index Series	52-54	
27)		About (mult-index) DF	55-57	
28)		About (merge)	58-60	
29)		About (join)	61-64	
30)		changes in Pandas	64-66	
31)		— projects —	66-67	

— projects — (100)

17/7/2020

## Pandas

[At any method, Place cursor and press Shift + Tab (to see help) / more details]

[In Jupyter working on pandas with python]

To read a tabular data file into pandas

In[1]

In[1]: Import pandas as pd

(ctrl+enter) → to run that cell

(shift+ctrl+enter) → to run & save to new level

In[2] pd.read\_table('http://bit.ly/chiporders')

read\_table() → can directly read from an url

And by using pd.read\_table → it directly

Prints the data. If we don't want to

Print: we can save it as a data frame object

order = pd.read\_table('http://bit.ly/chiporders')

out[2] order.head()

head() → prints the first 5 rows

In[3]

In[3]: pd.read\_table('http://bit.ly/movieusers')

out[3] 

1	Paul/mechanical	8534
0	2153 Arthur	194043
1	3123 Hunter	132067

 → this looks unsorted.  
" | " line that divides the column.

→ so, we have mention that " | " is a separator

pd.read\_table("http://--", sep='|') → By default  
sep='|t'.

now this gives a proper table

But yet there is no header for table.

1/24/mechanical is a rough fit of taken as header

→ source have to mention there's no header

int pd.read\_table('http://... ', sep='|', header=None)

[now we want a header list for this data]

int user\_cols = ['user\_id', 'age', 'gender', 'occupation', 'zip\_code']

int pd.read\_table('http://... ', sep='|', header=None, names=user\_cols)

[now get the perfect output]

### Note B

→ read\_table(file path, sep='\t', delimiter=None,  
names=None, header=0, index\_col=False,

more  
useful

{ skiprows=None,  
skipfooter=0 etc... }

→ if file contains a header row, then have  
to explicitly pass header=0 to override columns (name)

→ index\_col=False { index\_col → column to use as row label,  
when to force pandas not to use first column as index }

\* instead of read\_table

we can also use read\_csv

CSV - comma separated values.

CSV file has sep='|' by default

→ so, for CSV files better do use

read\_csv

Panda Series [To create `pseries = pd.Series([1, 2, 3, 4, 5])`]

Each column or a data frame is called panda series.

We can also create separate panda series.

We might need to manipulate the series or do analysis on it.

### Ex1

Qn1 Import Pandas as pd

Qn1 ufo = pd.read\_csv('http://bit.ly/uforeports')

Qn1 type(ufo) → gives type

Qn1 ufo.head()

### Output

	City	Colors Reported	Shape Reported	State	Type
0	Staten Island	NA	TRIANGLE	NY	2210
1	Holyoke	NA	OTHER	NJ	2010
2					
3					
4					

(\*) Now, basic way to select series is [ ] → bracket

(Ex) ':' we use `ufo['City'] / ufo.City`

Qn1 ufo['City'] → could be safe

(1)

Qn1 ufo.City

But, to select Color Reported by ':'  
we can't do that as it have space  
only [ ] for space having names

→ How do create new pandas series in a DataFrame

int ufo.city + ufo.state

out 0 Maryland  
1 New York  
2 New Mexico

[b, to provide 'y']

int ufo.city + ' ' + ufo.state

[now it gives '']

But to add it to the DataFrame

data-frame.name['columnname'] =

int ufo['Location'] = ufo.city + ' ' + ufo.state

→ [now, it adds new series]

Notes

→ Imagine 'movies' file

now,

int movies.describe()

[gives an statistical analysis about a song]

int movies.head()

int movies.shape

out (67, 6) → rows, columns.

int movies.dtypes

out object object

→ Here describe, head need () Parentheses

but shape, dtypes doesn't. Because, describe, head are actions while as shape is property

## → How to rename columns in a DataFrame

m1 Syntax

df-name.rename\_axis(' - ')

(to set index name)

df-name.columns = {  
old-name : new-name,  
old-name : new-name  
df-name.rename(inplace = True)}

- Have to remember 'inplace = True', that's what changes the name.
- can change multiple names at a time.

uto.rename(columns = {'color\_Reported' : 'colors\_Reported'})  
inplace = True)

to check if

uto.columns

→ to see the column names.

index[ 'city', ... ]

m2

\* But if you want to change all the column names above method will be difficult to type all old & new.

→ we create a list & assign to uto.columns.

uto-coll = ['city', 'color\_reported', 'category', 'state', 'time']

\* uto.columns = uto-coll → modified all names

### method 7-3

we can change them while we are reading itself. must be done

ex: `ufo = pd.read_csv('http://...', names=ufo.columns, header=0)`

to override the org. column name we have to use, `header=0`.

### Note 8

\* there is a direct method to remove all the spaces in columns

`ufo.columns = ufo.columns.str.replace(' ', '_')`

Space with the

→ to remove a column / row

[we have to know the name of column index to remove.]

ex: `ufo.drop(['Colors Reported'], axis=1, inplace=True)`

only for strings.

→ this is for rows

→ this is for columns

ex: `ufo.drop(0, axis=0, inplace=True)`

### Notes

we can remove multiple rows/columns at

⑥ `ufo.drop([0, 1], axis=0, inplace=True)`

## How to sort a pandas DataFrame / series.

Eg:-

int movie['title'].sort\_values()  
↓  
series name

function-  
df.name. series.name.  
sort\_values()

out our (too) days or summer  
1

(or)

int movie['title'].sort\_values() → on other way.  
with [ ]  
series name.

default

→ In the both case sort by descending order.

→ we can change it to descending.

int movie['title'].sort\_values(ascending=False)

out 864 cccj  
526 July

↓  
by default  
ascending=True.  
But we can change it.

↳ now gives  
in descending order.

\* In the both case, it won't change the org. data, but gives a sorted output of series (not remaining data).

→ But if you want to see the complete data sorted by corresponding to a particular column, then

movie.sort\_values('title')

column name

→ I can see the complete data sorted (org. same)

movie.sort\_values('duration', ascending=False)

Simp 8

→ sorting by multiple columns.

`movies.sort_values([['content_rating'], 'duration'])`

→ we can give  
multiple column to  
sort.

↓  
first sorts  
by `title` & then sort by  
`length` ..

④ How do filter rows or a pandas df by column

Eg1 (m1) [to select all movies according to duration]

import pandas as pd

`movies = pd.read_csv('https://bit.ly/2mdBtng')`

`longdur = [True if length >= 200 else False for length`

`movies.duration`

`long = pd.Series(longdur)` → creating series from  
a list of boolean

on `movies[long]`

out	duration
	200
	201
	202

Here Pandas takes the  
True, False. And remove  
all the False → rows  
[or accepts booleans].

Eg (m2) → instead of doing all the above

we name,

`long = movies.duration >= 200`

`movies[long]`

only works  
in pandas

→ this  
directly  
creates  
series of  
True, False

(8)



881

so, the Boat method to filter rows by column value.

infmovie [movie.duration  $\geq 200$ ] → single line is enough. !!

df-name [df-name.col-name(operation)]

Same way,

inf movie [movie.genre == 'Crime']

out

some
action
crime

→ If you want to see the genre of all the movie that are  $\geq 200$  duration.

inf movies[movies.duration  $\geq 200$ ].genre

two operations

But sometimes this could cause trouble. Single operation

so, we prefer

movie.loc[movie.duration  $\geq 200$ , 'genre']

→ loc is very powerful. It separates row, column we want. (we can see.. later)

another diff

what

what

- so what is the problem with this

## ④ multi-filter

using '&'

- If we want movie & 200 & Drama genre.  
then
- movie[ (movie.duration >= 200) & (movie.genre == 'drama') ]
- Parentheses are important.
- movie.duration >= 200 → is a boolean query  
& False are eliminated

⑤ If you want some categories in a column,

→ like people from waz, chhala, etc...

→ In such case we can't give or(||)  
for many times like

Eg)

movies[(movie.genre == 'crime') || (movie.genre == 'sci')]

we have simpler syntax

|| 'or' operation  
connects even entire one  
if True.

movie[movie.genre.isin(['crime', 'drama', 'action'])]

Simple-code



easy  
but and tway.

(In a column)

only for multi-filter by or-  
operator

⑩

## ④ Reading limited no. of columns

```
anoviel = pd.read_csv('http://...', usecols=[('time'), ('rating')])
```

out: anoviel.head()

out:

	time	rating

→ instead of  
col-name  
we can use  
index no.

usecols=[0, 4]

## ⑤ To take a quick look at a big data

```
uto = pd.read_csv('http://...', nrows=3)
```

uto

→ preferable

out: this only gives the first 3 rows (last).

## ⑥ Iteration (to access index)

```
for index, row in uto.iterrows():
```

```
    print(index, row.city, row.state)
```

co, staca, (ny)

(1, holyoke, co)

→ Selecting only numeric types from a data

import numpy as np

in drinks.select\_dtypes(include=[np.number]).dtypes

out beer\_fermentation int  
spirit int

↓  
to check

18/2/2020

[('spirit')]

→ How to use [drop] parameter in print

changes original data

+ First about [print(inplace=True)]

By default everything

has inplace=False

→ By default 'drop' → False inplace=False

that means it won't affect the original data and just prints copied data by removing the specified.

→ print(inplace=True) / not using inplace all same

(just shows data, won't affect original)

→ while inplace=True, changes original data

↑ column.

drinks.drop(['continent'], axis=1).head()

inplace  
False

shows without continent → temporary

drop - statement

$axis=0 \Rightarrow axis = \text{index}$   
 $axis=1 \Rightarrow axis = \text{column}$

For column.

`df-name.drop('col-name', axis=1, inplace=True)`

For row

`df-name.drop('row-name', axis=0, inplace=True)`

so

$axis=1 \rightarrow \text{column axis}$ ,

$axis=0 \rightarrow \text{row axis}$ ,

Eg:

If we want 'mean'.

ans → `drinks.mean()`

Both are same.

$axis$  is direction  
 $axis=0$  says go down.  
row by row.  
 $axis=1$  says go left-

out beer-servings 06.16  
wine-servings 00.99

→ what's happening is,  
by default mean gives  
the mean of first columns,  
not rows. (so, it's going down)

ans → `drinks.mean(axis=0)`

→ But, we can tell pandas  
to give mean of rows  
by saying  $axis=1$ ,  
by default it has  $axis=0$

out → `drinks.mean(axis=1)`

By saying  $axis=1$ , it

goes in left direction

0 0.00  
1 0.99  
2 0.93

$axis$  is down  
movement,

`drinks.mean(axis=1).shape`

ans

`drinks.mean(axis=0).shape`

4

Ans 1

Instead of 0, 1.

we can use (axis= 'index') → axis=0

(axis= 'columns') → axis=1,



dt-name, col-name str.slice(-5,-4) ↗ can be auth



How do we use string method [str. containing  
str. upper]

(there will be many string methods)

here are some useful methods:

→ to change a complete series or DF to upper case.

→ orders.item-name.str.upper()

→ column / rowname

→ But it won't affect original data.

→ to override this with org. data,

→ orders.item-name = orders.item-name.str.upper()



now it changes the data.

→ If you want to find 'CHICKEN' from all data

then,

This returns boolean  
of

orders[orders.item-name.str.contains('CHICKEN')]

→ These upper, contains are methods of str

to replace []

→ order. choice-description. str.replace('(', ')').str.replace(')', ')')

→ consecutively two times do remove/replace  
two things.

(②) → order. choice-description. str.replace('[(1)]', '')  
notell → with this we can remove  
multiple 'or' between them.

① df-name [can take boolean & save output]

→ how to change the datatype of Pandas DataFrame

(.astype)

→ order. datatype

all

item-price object

itemPrice \$9.0

so, do modifications to price we have  
to convert it to float

→ order. item-price. datatype (float)

But, as we also need to move '\$'. we can do  
it in one line

→ order. item-price. str.replace('\$', ''). datatype (float)

↳ math-operations.

Simpl useful for ML.

→ also type allo needed to convert booleans,

orders. Item\_name.str.contains('chicken'), astype(int)

→ false } → are converted to { ;  
true }

applies  
for all  
numerical

④ Using `groupby`:  $\text{df\_name.groupby('col\_name').op}$

(agg - useful to give multiple)

ESI chunked -data { not compulsory do & specify  
columns or not, if so only  
num\_cols }

env drinks, beer-servings, mean( $\gamma$ )

106-1606

\* Groupby - (mainly)  
when you want  
to categorize a  
column,,

→ But, what if we want the mean of only Europe country or like only India.

F) drinks [drinks + condiment = salsa]. beer-Sorvymen

In this way we can find

→ But, what if we want a data for separate countries.

then → 'groupby' comes into role.

→ First, it 'filters' the data by each name and then does the analysis.

also  
we have  
min, max  
etc

so,

on1 `drinks.groupby('continent').beer_serving.mean()`

out1

continent	mean
Africa	61.67
America	32.04
Europe	193.11

This filters the columns and applies the operation for each category.

categoryed  
the  
column,

Now,

we have 'agg' method which lets you to give multiple arguments.

on1 `drinks.groupby('continent').beer_serving.agg(['min', 'max', 'count'])`

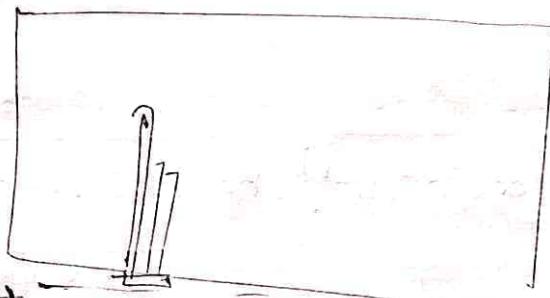
continent	min	max	count
Africa	44	0	:
America	21	0	:
Europe	53	1	:

gave multiple

To see the above data in ball in a graph

out `%matplotlib inline`

out `drinks.groupby('continent').mean().plot(kind='bar')`



④ To explore a panda series — —

[ useful methods  
describe, value\_count etc ]

→ take a movie data

out `movie.genre.describe()` → describe is so

out `count 979  
unique 16  
top Drama  
freq 828`

useful in call

or objects

→ here of gives  
no. of unique  
object.

→ And the most  
common one  
by frequency.

→ If you want to see  
the all unique type

unique() to know  
number

out `movie.genre.unique()`

show all

out `array(['Comme', 'Drama', ...])` unique obte  
in genre

→ If you want to explore about the unique object. like movie counts then,

in `moviel.genre.value_counts()`

out

Drama	238
comedy	156
crime	124
:	
:	

→ It is highly useful for object type

in `moviel.genre.value_counts(normalize = True)`

out

Drama	0.2839
comedy	0.1503
:	
:	

→ gives  
Percentage  
data

→ If you want a crosstab b/w two columns.

in `pd.crosstab(moviel.genre, moviel.content_rating)`

out

content_rating	Approved	AP+
sci-fi	3	1
drama	9	2
comedy	1	1

→ delombe,  
value\_counts  
With int type  
not much useful

in `moviel.duration.describe()`

`moviel.duration.value_counts()`

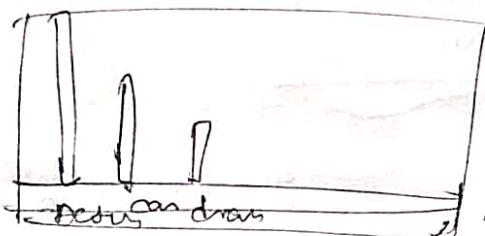
out

count	939,00
mean	120.39
min	1
max	1

↓  
With int type,  
give mean,  
min,  
max

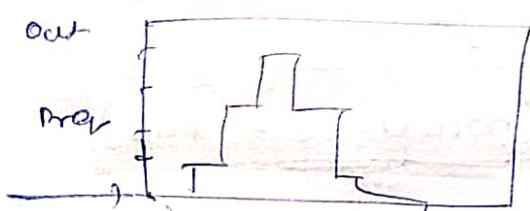
42 93

- we can also see graph/plot for value count
- `%matplotlib inline`
- `movielens.genre.value_counts().plot(kind='bar')`



histogram

- `movielens.duration.plot(kind='hist')`



### To Handle missing values in pandas

→ take 'info' rule

→ info, describe



isnull()  
notnull()  
sum()

city	colorReported	shape
Rock	NaN	NaN
Lake	NaN	tray
ME	NaN	wave
Borne	RD	light

→ If we see above table the 'NaN' character represent null.

→ No element is present, only values.



→ we can find the missing value by using isnull.

→ `uto.isnull().any()` → see last ⚡

city	color	shape
False	True	True
False	True	False
False	True	False
1	False	

`isnull()` converts all the characters into booleans and all the NaN (missing values) becomes True, others become False.

→ so, all the one elements here are missing elements.

→ `uto.notnull().any()`

`notnull` is reverse, null element - False

→ to find all the missing value, we have `sum()`,

out `pd.Series([True, False, True]).sum()`

out 2.

→ so, `sum()` has 'axis=0' default and counts all the 'True' values, so,

→ `uto.isnull().sum()`

out city at

color 4859

shape 2844

state 0

time 0

Given all the missing values details in a data frame

QMPF

④ To wst out all the missing values

→ `uto[uto['City'].isnull()]`

col name

→ gives the details  
of missing values in  
City

out

	<u>city</u>	<u>colors</u>	<u>shape</u>	<u>state</u>
0	NaN	NaN	NaN	LA
1	NaN	NaN	light	LA

(with index number)

⑤ To remove all the data, even if there is a single missing value in row,

`uto.dropna(how='any')`

this directly

Print data. But  
won't modify it.

`uto.dropna(how='all')`

'or'  
(and)

'any' tells that  
In a row, if  
any element is  
missing dit row

we can check the  
shape or pure data,

`uto.dropna(how='any').shape`

dit tells  
that dit  
a row  
only if  
all the element are NaN

means

→ this gives a shape of  
clean data without  
any missing a

⑥ mainly,

if we have to modify the missing  
value by something, then,

`uto['colors reported'].fillna(value='notseen', inplace=True)`

(inplace = True) → It's temporary change

④ If we want drop a row, if either  
city / shape is missing.

i.e; comparing columns, to drop. Then,

`ufo.dropna(subset=['City', 'Shape Reported'], how='any')`

now, this drops a row if there is a missing  
value in city / shape.

value counts

↳ by default `dropna=True`,

→ `ufo['Shape Reported'].value_counts()`

This will drop a  
row, if there  
is a missing value  
in both columns

`how='any'`

up

light  
dark  
|

→ But this won't  
give the count of missing value

→ By default value counts has  
(dropna=True). So, (dropna=False) → change.

`ufo['Shape Reported'].value_counts(dropna=False)`

all  
near 99%  
dark ~  
light ~

now gives  
missing  
value count

Note: Keywords

`birth=None`

↓  
it requires no non  
-NaN

`(how='any'))`

or

`(how='all'))`

`isnull()`

`notnull()`

`isnull().sum()`

`dropna()` / `dropna(subset=[''])`

`fillna() → (value=())`

\* at thresh=2

it deletes all  
rows which do not  
min 2 non-NaN

23

## \* About Pandas Index

(label)

- indexification
- selection
- alignment

### ① Indexification

→ Even if we filter the data, it keeps index / row number as int.

Ex1

in1 - `drinks[drinks['continent' == 'Asia']]`

out1

	country
6	Angola
20	
101	
112	

→ we can identify at what row no.

so np.

### ② selection [loc]

→ when we want a piece of df.  
we need index.

→ If you want to access the beer\_serving  
of India.

then, we need its index no.

in2

`drinks.loc[6, 'beer_serving']`

out2



→ But, this kind hard to remember the index no. all the time. So, we can change the index.

inp drinks.set\_index(['country', inplace=True])  
drinks.head(1)

out

country	beer-penny	-	-	-
Asia	- - -	-	-	-
Europe	- -	-	-	-
?	-	-	-	-

If you want to remove it, we can

inp drinks.index.name = None → it removed index  
drinks.head(1)

out

	beer-penny	-	-	-
Asia	- - -	-	-	-
Europe	- -	-	-	-

Now we can access 'Asia beer-penny' without index number.

inp drinks.loc[['India', 'beer-penny']]

It you want to reset the index then  
first give name

inp drinks.set\_index('country')

drinks.reset\_index(inplace=True)

loc → we can use loc for any database

Eg.

```
drinks.describe().loc['Asia', 'beer-consumption']
```

→ we can use loc, even with describe method.

### (3) Assignment

→ first we have to know every series has a index (same as its DataFrame - mostly).

Eg 1

```
drinks.continent.value_counts()
```

out:

Africa	53
Asia	44
Europe	48
!	!

→ If we see here Africa, Asia etc... are the index of that series.

→ As we know, we can access somethings by index. Same here,

out:

```
drinks.continent.value_counts()['Asia']
```

out:

44

out:

```
drinks.continent.value_counts().sort_values()
```

Asia	44
Europe	48
!	!

But, we can also sort by index.

Index for this series

out:

```
drinks.continent.value_counts().sort_index()
```

26

→ Eg 1

```
people = pd.Series([3000000, 85000], index=['Albania',  
                                         'Andorra'],  
                                         name='population')
```

↳ This is creation of data frame.

↓  
This good  
as Index  
name

out1 `people`

```
out1 Albania    3000000  
          Andorra     85000
```

Name=population, dtype=int64.

→ Now, If you want multiply the population of Albania, Andorra with Pd's beer\_servings in drinks. Easy,

out2 `drinks.beer_servings * people`

```
out2 Afghanistan      NaN  
          Albania      26700000  
          Oregon       NaN  
          Andorra      20825000  
          Asia         NaN
```

→ with the full  
Index of drink,  
it searched for  
people index &  
aligned those index  
with those & does  
the math.

→ They are at the  
same position.

Every they are  
became none

This is caused  
assignment

→

→ Assignment, also useful, if you want to add a new column or row to the given data.

→ For column → ans=1, & index-matched / align

→ If you want to add people to chunks.

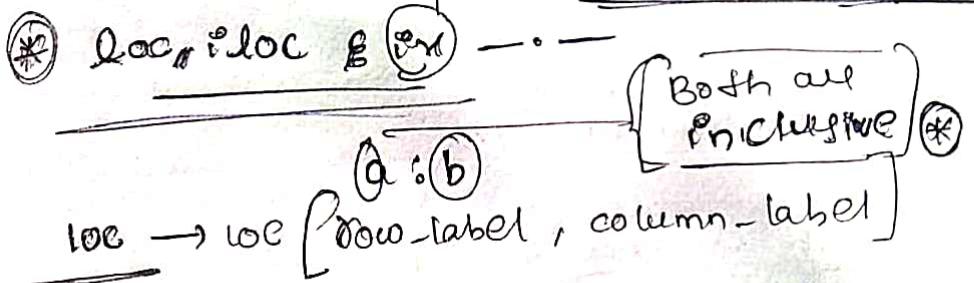
Ex. pd.concat([drinks, people], axis=1).head()

	beers			population
Afghanistan				NaN
Albania				8000000
Algeria				NaN
Andorra				0000

→ as name is population, it takes that as a column name, and align the index & gives a column, everything else is NaN.

only if axis=0

Created a row, with a new column as there is no such name as population



It is to select specified rows & columns.

in: uto.loc()

out:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca				
1					
2					
3					

in: uto.loc[0,:] → this specifies take

out:

City	Ithaca
Colors Report	None
Shape Rep	triangle
State	NY
Time	6/1/1980 21:00

in: uto.loc[0:3,:]

out:

	City	Color	Sh	St	T
0					
1					
2					
3					

inclusive.

thus specified that, we '0' to '3' rows (both all inclusive) and columns go to end.

in: uto.loc[:3, 'city':('state')]

	City	Col	Sh	St
0				
1				
2				
3				

0 to 3 rows & city to state column

one `uto[uto.City == 'Oakland']`

answ

c	col	sh	st	su
oak				
oak				
"				
"				
u				

Thus can be  
achieved by  
loc also

same

one

`uto.loc[uto.City == 'Oakland']`

iLoc `[0:1:6]` → exclusive dr by

→iloc won't work

wke, 29/1

`uto.iloc[0:3, [city]]`

Error

sb can't be single

→ `uto.iloc[0:2, :]`

out

	city	cows	The	sta	out
0					
1					

as & is one

→ to read multiple columns

→ `uto[['city', 'states']]` `[0:4]`, explicitly

Mondays 8:00 AM  
20/7/20

④ To make the dataframe smaller and faster

[using 'category']

→ when we import a file, it just creates a reference or objects when we check the info or df, it won't show the full info or objects. Because, as we know strings consume more memory than ints so what we can do to reduce size is convert the string to int. And this can be done by Category.

Ex: `drinks.info()`

In: `drinks.info()` → give complete info about column & everyth

country	object	about column & everyth
beers	int	
:		
nonalcoholic_beverages	object	
memory_usage: 9.2+ kb		Here, it says it has some more data, that is present is not surfing into object column deep.

Or we wanna free the memory.

In: `drinks.info(memory_usage='deep')`

country	object	about column & everyth
beers	int	
:		
nonalcoholic_beverages	object	
memory_usage: 30.5 kb		This is the total memory used.

→ If you want to see memory consumed by separate column.

out

drinks.memory\_usage()

out

index	128
country	1544
beer_serving	1524
"	"
continent	12332
city	12332

still there

hides the object usage

out

drinks.memory\_usage(deep=True)

out

index	128
country	12588
beer_serving	1524
"	"
continent	12332
city	12332

originally  
consumed  
data

↳ we can use

sum()

to get total

→ Now, to change these objects to category.

First, we have to see the no. or unique types in above. It only works when there are only a few unique types.

→ It is waste to use when there are many unique types.

on

sorted(drinks['Continent'].unique())

out [ 'Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South America' ]

(there are only 6 unique values in continent column)

So, we can use category

Basically, what category does is, In the sorted order for unique values, it gives

0 → (category)

1 →

Africa - 0

Asia - 1

Europe - 2

NA - 3

Oce - 4

{ → there are  
just the  
code for it,

→ col-name

drinks['continent'] = drinks['continent'].astype('category')

on

drinks.dtypes

get

?  
continent category

on

drinks.memory\_usage(deep=True)

out

country	brand
continent	744

[size has been reduced]

→ reduced from (12332) to (204)

on

drinks['continent'].cat.codes.head()

out

0	1
1	2
2	0
3	2
4	0
5	1

→ to see them  
in code form

For country

grt `drinks.country = drinks.country.astype('category')`

obj `drinks.dtype`

out

gender	128
country	18094
beer	way
!	
category	244

For country it has been increased too much after converting to int. That is because, it has too many unique values.

obj `drinks.country.unique()`

out

12193

there are 193 values. It takes more memory than int.

so,

Notes

# category can only be used when there are few no. of unique values.

- ④ As we know category converts them as int, we can use it for another advantage. When sorting a column by a logical order, we can specify the order.

89

ans df = pd.DataFrame([{'ID': [60, 100, 20, 80],  
 'Quality': ['good', 'excellent', 'good', 'very good']}])

ans df

out

	ID	Quality
0	60	good
1	100	excellent
2	20	good
3	80	very good

ans df.sort\_values('Quality')

out

	ID	Quality
0	100	Excellent
1	80	good
2	20	good
3	60	very good

ans

As basically, sort values  
sort by alphabetical  
order

But we know this is  
not what we want

But, if we want pt do sort, on the logical  
order as we want then we sort categories  
as good, 20, very good = 1, Excellent = 2

ans

from pandas.api.types import CategoricalDtype

quality\_cat = CategoricalDtype(['good', 'vgood', 'excellent'], ordered=True)

df.Quality = df.Quality.astype(quality\_cat)

df.sort\_values('Quality')

out

	ID	Quality
0	60	good
1	20	good
2	80	very good
3	100	Excellent

Now, pt sorted  
as we wanted

→ But, this can also be taken advantage in another way.

→ If we want all the data & good. Actually, we can't apply comparison operator on strings. But, as we changed it category in CategoricalOrder. Now, we can apply.

on `df10g [dequality & good]` → in this way, category is so easily

out

	20	Quality
1	100	excellent
2	20	very good

 →

### ④ Dummy Variables

It's creating 0, 1 for every unique value in a column. For every unique value, It creating 1 and all 0's.

Eg:-

2) Import Pandas as pd.

on `train = pd.read_csv('https://bit.ly/Kaggletrain')`

on `train.head()`

on

sex
male
female
male
male

 5

→ how to identify easily, male & female.  
we created a new column, mapping

in `train['sex-male'] = train.sex.map({'female': 0, 'male': 1})`

out `train.head()`

sex	sex-male
male	1
female	0
male	1
male	1

map method creates  
a series with the  
way aligns values.

→ aligns all  
we mentioned,  
→ now here all are  
are female,  
→ then → male.

→ this mapping is also  
called dummie,

in `Pd.get_dummies(train.sex) → one hot-encoding`

	female	male
0	1	0
1	0	1
2	0	0
3	0	1
4	1	0

→ get\_dummies, gives all the  
possible levels in a column  
→ there are k unique elements  
it gives k columns for  
get\_dummies. And each one  
has one 1 and zero 0.

→ But, to identify  
we only need  
male here or female, like we only need one  
level.

→ if we take male all others as male & vice versa

→ so we always need k-1

ans pd.get\_dummies([train['Sex'], Prefixes('Sex')]).iloc[:, 1:]

out

Sex-male
0
1

→ kernel

ans

male\_dummy = pd.get\_dummies([train['Sex'], Prefixes('Sex')], dropfirst=True)

[C, NC]

ans

pd.concat([train, male\_dummy], axis=1)

out

This gives same output as we got before

Sex	Sex-male
male	0
female	1
male	0

Same way,

→ kernel

ans

embarked\_dummy = pd.get\_dummies([train['Embarked'], Prefixes('Embarked')], dropfirst=True)

Prefixes('Embarked')

.iloc[:, 1:]

out

pd.concat([train, embarked\_dummy], axis=1)

out

Sex	Embarked	Sex-male	Embarked C	Embarked Q
male	S	0	0	1
female	C	1	0	0
male	S	0	0	0
male	S	1	0	1
female	Q	0	1	0

↓  
newly added  
columns

→ At we see,

that's why  
we said,  
we only need  
 $K-1$ .

	embarked-Q	embarked-S	
0	1	0	S
0	0	1	C
0	1	0	C
0	1	0	S
1	0	0	Q

→ Here, two are enough find all u.

notes

↑ Now we are passing a series (a column) to get\_dummies. But, we can also pass a data frame to get\_dummies.

eg

```
Pd.get_dummies (train, columns = ['sex', 'Embarked'])  
drop_First = True)
```

out

	sex	Embarked	sex_male	embarked-Q	embarked-S
male	0	S	1	0	1
female	1	S	0	0	0
			1	0	1
			1	0	1
			0	0	1
			0	1	0

→ Here, we accomplished what we need on a single line without using 'loc' or 'concat'.

→ But, a little drawbacks is we lost the org. sex, embark column by this.

→ And we exactly gets  $K-1$  column, by saying drop\_First=True.

## Date and Time in pandas

(to work with date & time)

887

in `uto.head(3)`

Time
8/1/1930 22:00
8/20/1980 20:00

Suppose to extract hour from time. we can do it by slice. But first gotta check the type.

in `uto.dtypes`

out

time object

so have to convert it

in

`uto['Time'].str.slice(-5,-3).astype(int).head()`

out

0	22
1	20
2	1

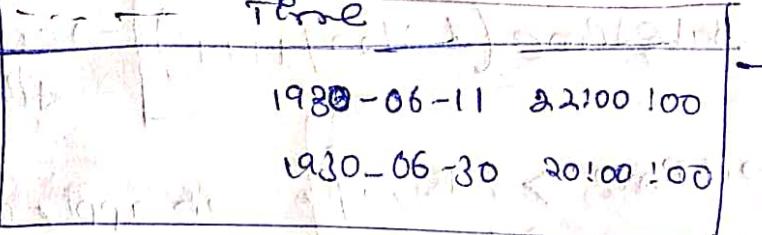
we can do it this way. But what if we want year again the same.

But, instead all this pandas provided `to_datetime` which have a lot of excellent operation to use like `weekday` (`year`, `time` etc.)

overriding to datetime type

int `uto[('Time')] = uto pd.to_datetime(uto.Time)`

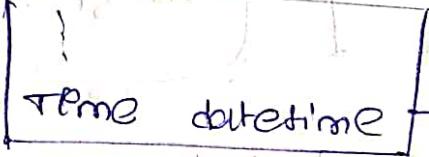
int `uto.head()`

out 

→ data is same  
But the format has changed.

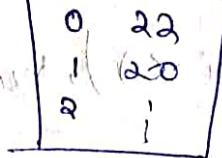
Actually, what really changed of type?

int `uto.dtypes`

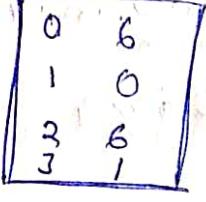
out 

now, it's a 'dt' object.  
we can perform some operations.

int `uto.Time.dt.hour`

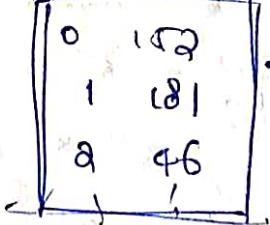
out 

int `uto.Time.dt.weekday`

out 

0-monday  
1-tuesday  
2-wednesday  
3-thursday  
4-friday  
5-saturday  
6-sunday

int `uto.Time.dt.dayofyear`

out 

→ gives the day of year(:)

This gives the day by calculating the date.

and we can also perform all the  
math operations on datetimes

eg1

any  $ts = pd.to_datetime('7/27/1999')$  → creating a time stamp

out  $uto.loc[uto.Time >= ts, :]$  can apply math operation

out

	Date
1	1999-7-27 20:00
2	1999-7-29
3	

→ now this gives the data at all dates which are greater than time stamp

any  $uto.Time.max()$  → gives the latest time.

out  $Timestamp('2000-12-31 23:59:00')$

any  $(uto.Time.max() - uto.Time.min()).days$

out 25781 days



notes

→ we can see this complete usage on API reference.

→ ctrl+f1 open search bar

→ we can find everything

make a date-time graph

(matplotlib)

ans `%matplotlib inline`

gm `uto['Year'] = uto.TPme.dt.Year` Add a new column,

gyr `uto['Year'].value_counts().sort_index()`

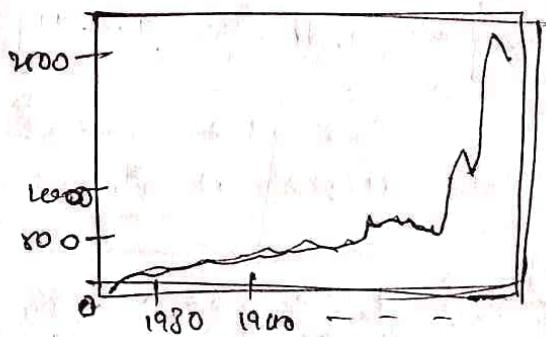
out

1980	2
1981	2
⋮	⋮

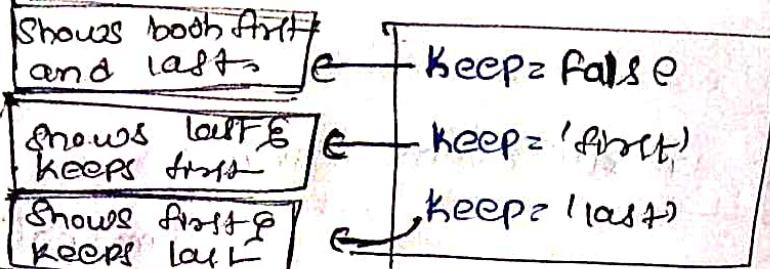
gm

`uto['Year'].value_counts().sort_index().plot()`

out



11/2/20



\* To remove and add duplicates in a column.

import pandas as pd.

in user\_cols = ['user\_id', 'age', 'gender', 'occupation', 'zip\_code']  
users = pd.read\_table('http://bit.ly/movielens',

sep = '|', header = None, names = user\_cols  
index\_col = 'user\_id')

out user.head()

user_id	age	zip_code
1	33	48134
2	32	48134
3	32	48134
4	32	48134

\* To check the shape.

in users.shape

out (943, 4)

→ rows.

But, In 943 rows there could some duplicate rows, like same data in rows, or like zip\_code - duplicate, repeated zip-codes.  
→ If there are duplicates we have to find & remove them.

out `users.zip_code.duplicated()`

col-name → to check no. of duplicates  
use `sum()` → gives count of true  
this gives a series or boolean

out

0	False
1	False
2	False
3	True
4	False

→ In here, 'True' is something that has already there above it.  
That means, actually duplicated by default has keep = 'first' which keeps the first one & shows the second one as `duplicated(i.e., True)`.

As we know to see the True's

out `users[users.zip_code.duplicated()]`

out

496
532
621
684

→ gives all the True's

`users[users.zip_code.duplicated(), :]`

we can also get same output by `loc`

out `users.loc[users.duplicated(), :]`

out `users.loc[users.duplicated(keep='last'), :]`

out

496	21	F	student	55419
532	51	M	student	20003
621	51	M	student	55419
684	51	M	student	20003

→ This is due to True, only if all the columns or dat have repeated.

→ Here when `duplicated()` → on `keep=first`. It keeps 198 → first occurred & shows up - 198.

→ In `keep='last'`, it keeps 496 & shows up 198 (completes same rows)

df-name.duplicated

This is due to True, only if all the columns or dat have repeated, and by default `keep = 'first'`

+ and now we have `keep=none`, which shows both of them.

gpt `users.loc[users.duplicated(keep=False), :]`

out

age
age
...

112 rows

→ Both of them.

to drop-duplicates

gpt

`users.drop_duplicates().shape`

out

986, 4

\* And this won't affect our data unless you use, `inplace=True`

we first have 943, 1000, 936, so all of the 7 duplicate rows has been lost. (lost ones are the last duplicates.)

gpt

`users.duplicated(subset=['age', 'zip-code']).sum()`

out

32

→ to check the duplicated by two columns.

→ it gives True, if a row has same age, same zip-code.

→ so, there are 32 age, zip-code same rows,

## \* Copy-warning

case-1

in  
moviel.content.rating.value\_count()

out

```
[{'notrated': 65}]
```

in  
moviel.content.rating.value\_count()

out  
3

# we have change all notrated to NaN

so, for that first we have to take the 'Not rated' series and then we have to assign it to np.nan to make them null elements.

in  
import numpy as np

in  
moviel[moviel.content.rating == 'Notrated'].content.rating  
= np.nan

out

```
[{'notrated': 65}]
```

in  
moviel.content.rating.value\_count()

out

3 → this will be 3. It hasn't changed

→ Actually what happens with.

`movie[movie. -- > 'not rated'].content_rated` copy

such line or code R. It is a two operation code.

→ This part is `get-item`.

→ Remains is `set-item`, which performs an operation on the produced result of `get-item`.

→ Generally, here is the problem.

Pandas can't guarantee whether the `set-item`, returned a view or a copy of the data.

→ So, it gives a warning.

But, this is not the case with `.loc`

`.loc` is a single operation.

`movie.loc[movie.content_rated == 'not rated', 'content.not']`

in

2D print

~~movie[content\_rated == 'not rated', sum]~~

out

68

now, pt

changed.

48

note ①

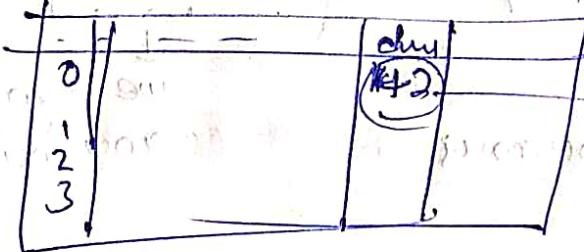
→ so, it's always better to prefer .loc

acc - 1

any  $\text{top-movie} = \text{movie}.loc[\text{movie.duration} > 29, :]$

any  $\text{top-movie}$

out



→ if we want to change this we can do by .loc

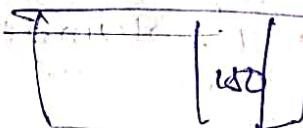
any  $\text{top-movie}.loc[0, ('duration')] = \text{B}$

out

→ warning

out  $\text{top-movie}$

out



→ even though, it's changed

But the warning is, it don't know dependency  
is a copy / view

note ②

→ so, it's always better to specify by copy() when copying data

any  $\text{top-movie} = \text{movie}.loc[\text{movie.duration} > 29, :].copy()$

any  $\text{top-movie}.loc[0, ('duration')] = \text{B}$

no error →

## \* Display option

to see the mark - only  
Pd.set-option → thus to check.

to display all the rows

none shows all the rows

int

Pd.set-option('display.mark-rows', None)

we can give any number here

int

Pd.set-option('display.mark-rows')

to get mark-column

int

Pd.set-option('display.mark-column', any number)

to see the full text in a column

int

Pd.set-option('display.mark-column-width', none)

to print 5 in numbers (like 4,600.0) can give any number

int

Pd.set-option('display.float-format', {:, ., fo.})

this only works for float columns, not for ints //

## ④ Dummy Dataframe

method①

→ creating a df by dictionary.  
But, it won't follow a order.  
we can specify.

↑  
column  
names

keeps  
values

↓  
column  
values

inpt Import pandas as pd

out

```
Pd.DataFrame ({'id': [100, 101, 102],  
              'color': ['red', 'blue', 'red']}  
            )
```

out

	color	id
1	red	100
2	blue	101
3	red	102

→ we can specify

the columns

order, if we want.

→ we can also change the index

inpt

```
Pd.DataFrame ({'id': [100, 101, 102], 'color': ['red', 'blue', 'red']},  
              columns=['id', 'color'], index=[a, b, c])
```

out

	id	color
a	100	red
b	101	blue
c	102	red

→ nowe specified the order

we also gave index value.

note

out)

1) we can create a normal dataframe by dictionary.

2) other way by list of dict.

3) another way by numpy array

## ④ Apply, map and applymap

- map is a vectorized method, creates a dummy or a series.
- apply is both series & data frame method.
- applymap is only for data frame method. And applies to each & every element in data frame.

map

apply operation on  
series

Ex `train['sex-num'] = train.sex.map({'female': 0, 'male': 1})`

Ex `train.loc[:, ('sex', 'sex-num')]`

cell

sex	sexnum
male	1
female	0
male	1

apply (both for series, data frame).

on series

Ex `train['name-len'] =`  
`train.Name.apply(len)`

\* But, not do the complete data, only for an axis,

only specific operations, count, sum, argmax

in: import numpy as np

only  
operation

out: train['Pare ceil'] = train.Pare.apply(np.ceil)

out:

2.2	3
1	1

in:

train.Name.str.split(',').head()

to take the  
first letter  
or name.

str-split-cut  
,,

in:

def get\_element(my\_list, position):  
 return my\_list[position]

out:

train.Name.str.split(',').apply(get\_element, position=0)

out:

0	Brais
1	camry

Instead of function, we know lambda,

in:

train.Name.str.split(',').apply(lambda x: x[0])

out:

0	Brais
1	camry

apply on dataframe

in:

drinks.groupby(['beer\_servings', 'wine\_servings']).apply(many  
and few)

out:

beer_servings	326
wine_servings	320

→ see the more to a row

out

```
drinks.loc[:, 'beer-servings': 'wine-servings'].apply(lambda x: x)
```

out

0	0
1	132
2	25

out

```
drinks.loc[:, 'beer-servings': 'wine-servings'].apply(np.argmax, axis=1)
```

out

0	0
1	1
2	0
3	2

np.argmax

axis=1

This gives the column-number of the entry having maximum value in a row.

applymap

→ This applies to full data frame or on columns.

out

```
drinks.loc[:, 'beer_servings': 'wine_servings'].applymap(float)
```

out

	beers	birns	carbs
0	0.0	0.0	0.0
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	20.0	132.0	0.0

→ applymap

applies full data frame

22/2/2020

## ④ multi-index

↑ series

[Selecting a multi-index is almost  
[or works like grouping]

similar to a normal

in import pandas as pd

in stocks = pd.read\_csv('http://bit.ly/smallstocks')

out stocks

out

	date	close	volume	symbol
0				CSCO
1				APPL
2				MSPR
3				APPL
4				MSPR

in

stocks.groupby(['symbol']).close.mean()

out

Symbol	close
APPL	12.05
CSCO	31.48
MSPR	5.45

→ This is a general  
groupby output.

→ But, we can also do groupby on  
multiple columns.

in

stocks.groupby(['symbol', 'date']).close.mean()

out

symbol	date	close
APPL	2016-10-3	12.52
	2016-10-4	11.80
	2016-10-5	11.80
CSCO	2016-10-3	31.45
	2016-10-4	;
	2016-10-5	;

→ Now, this is a  
multi-indexed series,  
with both symbol  
and date.

\* this multi-index series can easily be transformed onto a dataframe by unstack

one `ser.unstack()` → <sup>(outer)</sup> first index become index  
→ second index become column  
inner

out

<u>date</u>	<u>2016-10-3</u>	<u>2016-10-4</u>	<u>2016-10-5</u>
<u>Symbol</u>	-	-	-
APPL	-	-	-
ESCO	-	-	-
MIRT	-	-	-

\* But this output can also be obtained by a method 'pivot-table'.

\* For the index is column of take the unique values of the columns of dataframe.

\* And the values are the mean values by default. we can change that through of some output.

one `df = stocks.pivot_table(values='close', index='symbol', columns='date')`

## Selecting data from multi-index Series

[Almost same as normal df]

inpt ser

out

symbol	date	price
APPL	2016-10-3	125
	2016-10-4	125.5
	2016-10-5	130
CSCO	2016-10-3	85
	2016-10-4	85.5
MSFT	2016-10-3	21
	2016-10-4	21.5

If we want the stock data of a company.  
(for all dates).

inpt

ser.loc['APPL']

out

date	
2016-10-3	125
2016-10-4	125.5
2016-10-5	130.5

Selecting the outer index  
gives complete per data  
& values of it.

If we want value at that date

inpt

ser.loc['APPL', '2016-10-03']

out

ser.loc[:, '2016-10-03']

This gives out the  
value of the data  
selecting outer &  
a panel index.

out

symbol	
APPL	125.5
CSCO	85.5
MSFT	21.5

Selecting just a  
panel index, gives  
the values &  
outer index.

## ④ Dataframe w/ Pdt : multi-index

inpt

```
stocks.set_index(['Symbol', 'Date'], inplace=True)
```

inpt

stocks

setting multi-index

out

Symbol	Date	close	volume
ESCO			
OPPC			
MSPR			
OPPL			

→ To set in order.

inpt

```
stocks.set_index(inplace=True)
```

inpt

stocks

out

Symbol	Date	close	volume
OPPL	2016-10-3		
	2016-10-4		
	2016-10-5		
ESCO	2016-10-3		
	2016-10-4		
	2016-10-5		
MSPR			

→ multi-index  
data frame

inpt

```
stocks.loc['OPPL']
```

out

Date	close	volume
2016-10-3		
2016-10-4		
2016-10-5		

## Selecting from multi-indexed DataFrame

Inp = `stocks.loc[['APPL', '2016-10-03']]` → This will give output, but not a good practice.  
 Out = 112.52  
 Inp = `stocks.loc[['APPL', '2016-10-03'], :]` ← As we know `loc[index, column]`  
 Out = 112.52  
good practice  
 Inp = `stocks.loc[('APPL', '2016-10-03'), 'close']` → As both are index, we should give them in tuple.  
 Out = 112.52

⚡ If you want multiple outer index / inner index, we have give them in a list.

Inp = `stocks.loc[[['APPL', 'MSFT'], '2016-10-03'], :]` → multiple outer-  
 Out = Symbol Date close volume  
 APPL 2016-10-03 112.52 2102  
 MSFT 2016-10-03 304.1 6000 → all columns  
 Inp = `stocks.loc[[['APPL', 'MSFT'], '2016-10-03'], 'close']` → single column  
 Out = Symbol Date close  
 APPL 2016-10-03  
 MSFT 2016-10-03 → multiple inner index.  
 Inp = `stocks.loc[['APPL'], [2016-10-03, 2016-10-04], 'close']`  
 Out = `stocks.loc[[slice(None)], [2016-10-03, 2016-10-04], 'close']`

X  $(:, \dots)$  thus says all outer index for the selected multi-index.  
 Cannot be used.

ans Stock. loc [(slice(None), slice(2016-10-03, 2016-10-04)), 'close']

out

Symbol	Date	Close
APPL	2016-10-03	100.00
	2016-10-04	101.00
CSCO	2016-10-03	100.00
	2016-10-04	101.00
MSFT	2016-10-03	100.00
	2016-10-04	101.00

All out of  
index

For the  
selected  
index

\* Imagine two same multi-indexed  
dataframes, → close, volume.

\* To combine them, we use merge.

ans

both = pd.merge(close, volume, left\_index=True,  
right\_index=True)

notes

→ to reset multi-index

dt.name.reset\_index()

## ④ merge and join()

↳ by column keys → By row name

df1.append(df2)

pd.concat([df1, df2]) → Best way.

df1.append(df2)

pd.merge([df1, df2]) → Best way,

Ex:-

→ movie1.head()

→ ratings.head()

movie_id	title	user_id	movie_id	rating	threshold
0	star wars	196	900	3	—
1	Inception	123	302	3	—
2	Dark Knight	140	226	1	—
3	Donnie Darko	23	386	20	—
4	Interstellar	45	432	1	—

↓  
1 shape]

↳ 1

These both  
are same movie\_id

[1 shape]

100000

But 1st diff rating  
table without title.

→ So, we can merge both or then,  
to get title.

For 1682  
movies - 100000P  
has given  
rating.

⑤ movie\_ratings = pd.merge(movie, ratings)

id	movie_id	title	user_id	rating	threshold
0	1	Star Wars	196	3	—
1	1	One	123	3	—
2	1	One	140	2	—
3	1	One	23	5	—
4	1	One	45	4	—

④ Here, left df checks the index or right df  
of that merged on that for first  
movie\_id, title & last word, return it

→ movie\_ratings.shape

(100000, 5)

even movie id or 162, it  
checks every row for rating  
matched & gives pts column value

⑤ what if columns you want to form on don't  
have same name

→ movie.columns = {'m\_id', 'title'}

→ movie1.columns

as index ([m\_id, title])

→ many columns (why they diff. name)

as index [refer\_id, movie\_id]

→ pd.merge(movie, ratings, left\_on='m\_id',  
right\_on='movie\_id')

left\_on → there says which column do  
we have to merge with  
right

④ what if you want to join on one-index

→ movie & movie[per index] (movie\_id)

→ movie[head]

movie_id	title
1	One
2	Two
3	Three
4	Four
5	Five

→ pd.merge(movie, ratings, left\_index=True, right\_index=True)  
 (movie\_id)

↳ now at the only we want to match on  
 left ej index, we gives 'left\_index=True'

↳ when these both merged,

→ merge happens like -

in first data frame, it takes the first element of the columns we selected

↳ checks it with every single

element in the second df columns (that we selected). And if it finds a match,  
 it prints it.

so  
out

movie_id	title	user_id	movie_id	rating
1	One	308	1	7
2	Two	282	1	8
3	Three	108	1	9

- If the both have common index,  
at given that common index
- But if common we want as an index,  
then the resultant index changes  
randomly.
- In the above case, it takes "1" & checks  
with every & prints.
- But, as the left one is index, it  
won't put on resultant table.

### ④ Join on two index

[considers rows]

→ rating = rating.set\_index('movie\_id')

→ ready, here

on	movie_id	user_id	rating	name
	movie_id			
	202			
	202			
	322			
	:			

→ pd.merge(movie, ratings, left\_index=True,  
right\_index=True)

movie_id	user_id	rating	name
1	1	1.0	Toy Story
1	2	1.0	Toy Story
1	3	1.0	Toy Story
1	4	1.0	Toy Story
1	5	1.0	Toy Story
1	6	1.0	Toy Story
1	7	1.0	Toy Story
1	8	1.0	Toy Story
1	9	1.0	Toy Story
1	10	1.0	Toy Story
1	11	1.0	Toy Story
1	12	1.0	Toy Story
1	13	1.0	Toy Story
1	14	1.0	Toy Story
1	15	1.0	Toy Story
1	16	1.0	Toy Story
1	17	1.0	Toy Story
1	18	1.0	Toy Story
1	19	1.0	Toy Story
1	20	1.0	Toy Story
1	21	1.0	Toy Story
1	22	1.0	Toy Story
1	23	1.0	Toy Story
1	24	1.0	Toy Story
1	25	1.0	Toy Story
1	26	1.0	Toy Story
1	27	1.0	Toy Story
1	28	1.0	Toy Story
1	29	1.0	Toy Story
1	30	1.0	Toy Story
1	31	1.0	Toy Story
1	32	1.0	Toy Story
1	33	1.0	Toy Story
1	34	1.0	Toy Story
1	35	1.0	Toy Story
1	36	1.0	Toy Story
1	37	1.0	Toy Story
1	38	1.0	Toy Story
1	39	1.0	Toy Story
1	40	1.0	Toy Story
1	41	1.0	Toy Story
1	42	1.0	Toy Story
1	43	1.0	Toy Story
1	44	1.0	Toy Story
1	45	1.0	Toy Story
1	46	1.0	Toy Story
1	47	1.0	Toy Story
1	48	1.0	Toy Story
1	49	1.0	Toy Story
1	50	1.0	Toy Story
1	51	1.0	Toy Story
1	52	1.0	Toy Story
1	53	1.0	Toy Story
1	54	1.0	Toy Story
1	55	1.0	Toy Story
1	56	1.0	Toy Story
1	57	1.0	Toy Story
1	58	1.0	Toy Story
1	59	1.0	Toy Story
1	60	1.0	Toy Story
1	61	1.0	Toy Story
1	62	1.0	Toy Story
1	63	1.0	Toy Story
1	64	1.0	Toy Story
1	65	1.0	Toy Story
1	66	1.0	Toy Story
1	67	1.0	Toy Story
1	68	1.0	Toy Story
1	69	1.0	Toy Story
1	70	1.0	Toy Story
1	71	1.0	Toy Story
1	72	1.0	Toy Story
1	73	1.0	Toy Story
1	74	1.0	Toy Story
1	75	1.0	Toy Story
1	76	1.0	Toy Story
1	77	1.0	Toy Story
1	78	1.0	Toy Story
1	79	1.0	Toy Story
1	80	1.0	Toy Story
1	81	1.0	Toy Story
1	82	1.0	Toy Story
1	83	1.0	Toy Story
1	84	1.0	Toy Story
1	85	1.0	Toy Story
1	86	1.0	Toy Story
1	87	1.0	Toy Story
1	88	1.0	Toy Story
1	89	1.0	Toy Story
1	90	1.0	Toy Story
1	91	1.0	Toy Story
1	92	1.0	Toy Story
1	93	1.0	Toy Story
1	94	1.0	Toy Story
1	95	1.0	Toy Story
1	96	1.0	Toy Story
1	97	1.0	Toy Story
1	98	1.0	Toy Story
1	99	1.0	Toy Story
1	100	1.0	Toy Story
1	101	1.0	Toy Story
1	102	1.0	Toy Story
1	103	1.0	Toy Story
1	104	1.0	Toy Story
1	105	1.0	Toy Story
1	106	1.0	Toy Story
1	107	1.0	Toy Story
1	108	1.0	Toy Story
1	109	1.0	Toy Story
1	110	1.0	Toy Story
1	111	1.0	Toy Story
1	112	1.0	Toy Story
1	113	1.0	Toy Story
1	114	1.0	Toy Story
1	115	1.0	Toy Story
1	116	1.0	Toy Story
1	117	1.0	Toy Story
1	118	1.0	Toy Story
1	119	1.0	Toy Story
1	120	1.0	Toy Story
1	121	1.0	Toy Story
1	122	1.0	Toy Story
1	123	1.0	Toy Story
1	124	1.0	Toy Story
1	125	1.0	Toy Story
1	126	1.0	Toy Story
1	127	1.0	Toy Story
1	128	1.0	Toy Story
1	129	1.0	Toy Story
1	130	1.0	Toy Story
1	131	1.0	Toy Story
1	132	1.0	Toy Story
1	133	1.0	Toy Story
1	134	1.0	Toy Story
1	135	1.0	Toy Story
1	136	1.0	Toy Story
1	137	1.0	Toy Story
1	138	1.0	Toy Story
1	139	1.0	Toy Story
1	140	1.0	Toy Story
1	141	1.0	Toy Story
1	142	1.0	Toy Story
1	143	1.0	Toy Story
1	144	1.0	Toy Story
1	145	1.0	Toy Story
1	146	1.0	Toy Story
1	147	1.0	Toy Story
1	148	1.0	Toy Story
1	149	1.0	Toy Story
1	150	1.0	Toy Story
1	151	1.0	Toy Story
1	152	1.0	Toy Story
1	153	1.0	Toy Story
1	154	1.0	Toy Story
1	155	1.0	Toy Story
1	156	1.0	Toy Story
1	157	1.0	Toy Story
1	158	1.0	Toy Story
1	159	1.0	Toy Story
1	160	1.0	Toy Story
1	161	1.0	Toy Story
1	162	1.0	Toy Story
1	163	1.0	Toy Story
1	164	1.0	Toy Story
1	165	1.0	Toy Story
1	166	1.0	Toy Story
1	167	1.0	Toy Story
1	168	1.0	Toy Story
1	169	1.0	Toy Story
1	170	1.0	Toy Story
1	171	1.0	Toy Story
1	172	1.0	Toy Story
1	173	1.0	Toy Story
1	174	1.0	Toy Story
1	175	1.0	Toy Story
1	176	1.0	Toy Story
1	177	1.0	Toy Story
1	178	1.0	Toy Story
1	179	1.0	Toy Story
1	180	1.0	Toy Story
1	181	1.0	Toy Story
1	182	1.0	Toy Story
1	183	1.0	Toy Story
1	184	1.0	Toy Story
1	185	1.0	Toy Story
1	186	1.0	Toy Story
1	187	1.0	Toy Story
1	188	1.0	Toy Story
1	189	1.0	Toy Story
1	190	1.0	Toy Story
1	191	1.0	Toy Story
1	192	1.0	Toy Story
1	193	1.0	Toy Story
1	194	1.0	Toy Story
1	195	1.0	Toy Story
1	196	1.0	Toy Story
1	197	1.0	Toy Story
1	198	1.0	Toy Story
1	199	1.0	Toy Story
1	200	1.0	Toy Story
1	201	1.0	Toy Story
1	202	1.0	Toy Story
1	203	1.0	Toy Story
1	204	1.0	Toy Story
1	205	1.0	Toy Story
1	206	1.0	Toy Story
1	207	1.0	Toy Story
1	208	1.0	Toy Story
1	209	1.0	Toy Story
1	210	1.0	Toy Story
1	211	1.0	Toy Story
1	212	1.0	Toy Story
1	213	1.0	Toy Story
1	214	1.0	Toy Story
1	215	1.0	Toy Story
1	216	1.0	Toy Story
1	217	1.0	Toy Story
1	218	1.0	Toy Story
1	219	1.0	Toy Story
1	220	1.0	Toy Story
1	221	1.0	Toy Story
1	222	1.0	Toy Story
1	223	1.0	Toy Story
1	224	1.0	Toy Story
1	225	1.0	Toy Story
1	226	1.0	Toy Story
1	227	1.0	Toy Story
1	228	1.0	Toy Story
1	229	1.0	Toy Story
1	230	1.0	Toy Story
1	231	1.0	Toy Story
1	232	1.0	Toy Story
1	233	1.0	Toy Story
1	234	1.0	Toy Story
1	235	1.0	Toy Story
1	236	1.0	Toy Story
1	237	1.0	Toy Story
1	238	1.0	Toy Story
1	239	1.0	Toy Story
1	240	1.0	Toy Story
1	241	1.0	Toy Story
1	242	1.0	Toy Story
1	243	1.0	Toy Story
1	244	1.0	Toy Story
1	245	1.0	Toy Story
1	246	1.0	Toy Story
1	247	1.0	Toy Story
1	248	1.0	Toy Story
1	249	1.0	Toy Story
1	250	1.0	Toy Story
1	251	1.0	Toy Story
1	252	1.0	Toy Story
1	253	1.0	Toy Story
1	254	1.0	Toy Story
1	255	1.0	Toy Story
1	256	1.0	Toy Story
1	257	1.0	Toy Story
1	258	1.0	Toy Story
1	259	1.0	Toy Story
1	260	1.0	Toy Story
1	261	1.0	Toy Story
1	262	1.0	Toy Story
1	263	1.0	Toy Story
1	264	1.0	Toy Story
1	265	1.0	Toy Story
1	266	1.0	Toy Story
1	267	1.0	Toy Story
1	268	1.0	Toy Story
1	269	1.0	Toy Story
1	270	1.0	Toy Story
1	271	1.0	Toy Story
1	272	1.0	Toy Story
1	273	1.0	Toy Story
1	274	1.0	Toy Story
1	275	1.0	Toy Story
1	276	1.0	Toy Story
1	277	1.0	Toy Story
1	278	1.0	Toy Story
1	279	1.0	Toy Story
1	280	1.0	Toy Story
1	281	1.0	Toy Story
1	282	1.0	Toy Story
1	283	1.0	Toy Story
1	284	1.0	Toy Story
1	285	1.0	Toy Story
1	286	1.0	Toy Story
1	287	1.0	Toy Story
1	288	1.0	Toy Story
1	289	1.0	Toy Story
1	290	1.0	Toy Story
1	291	1.0	Toy Story
1	292	1.0	Toy Story
1	293	1.0	Toy Story
1	294	1.0	Toy Story
1	295	1.0	Toy Story
1	296	1.0	Toy Story
1	297	1.0	Toy Story
1	298	1.0	Toy Story
1	299	1.0	Toy Story
1	300	1.0	Toy Story
1	301	1.0	Toy Story
1	302	1.0	Toy Story
1	303	1.0	Toy Story
1	304	1.0	Toy Story
1	305	1.0	Toy Story
1	306	1.0	Toy Story
1	307	1.0	Toy Story
1	308	1.0	Toy Story
1	309	1.0	Toy Story
1	310	1.0	

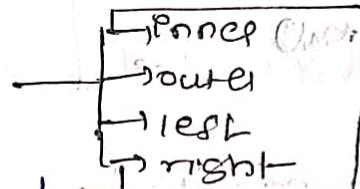
out

	title	year of release	rating	dimensions
1	Brown	1990	PG	100x100
2	Shaw	1990	PG	100x100
3	Shaw	1990	PG	100x100
4	Shaw	1990	PG	100x100

Gal left index searched for all the values or right. It comes this way (first comes first)

→ like 1st element in left searched & gets placed & after 2nd & 3rd etc.

### ④ types of join



on as

	color	num
0	green	1
1	yellow	2
2	red	3

	color	size
0	green	S
1	yellow	M
2	Pink	L

→ when we simply say merge, it merges based on common → column-name

→ pd.merge(A, B, how='inner')

→ pd.merge(A, B, how='outer')

out

	color	num	size
0	green	1	S
1	yellow	2	M

↓  
only common  
on both gets  
printed

out

	color	num	size
0	green	1.0	S
1	yellow	2.0	M
2	red	3.0	NAN
3	Pink	NAN	C

↓  
both of those get.  
all in A  
all in B

→ pd.merge(A, B, how='left')

Q1

	color	num	size
0	Green	1	S
1	Yellow	2	M
2	Red	3	Nan

common,

All in left

→ pd.merge(A, B, how='right')

Q2

	color	num	size
0	Green	1	S
1	Yellow	2	M
2	Pink	NaN	L

common

All in right

changed in pandas

④ To change to type category

(whole ready).

drinks = pd.read\_csv('https://bit.ly/drinks\_by\_country.csv')

dtype = {'continent': 'category'}

⑤ To convert multiple datatypes at once

drinks = drinks.astype({'beer\_servings': 'float'})

'spirit\_consumption': 'float'})

### ③ apply multiple aggregation

old way

`drinks.groupby(['continent']).beer_servings.agg`

`(['mean', 'min', 'max'])`

new way

`drinks.beer_servings.agg(['mean', 'min', 'max'])`

also

`drinks.agg(['mean', 'min', 'max'])`

more flexible than describe

### ④ exp-deprecated

[new]

⑤ isnull() → isna()

notnull() ← notna()

⑥ → in drop no need to use axis,

df.drop([0, 1], axis=0) × → noneed

df.drop(['Pindex=0, 1']) → (same for column)

⑦ → for rename.

df.rename(columns={('day'): ('CITY')}) → oldway

df.rename({'day': 'CITY'}, axis=1) → newway

⑧ from pandas.api.dtypes import CategoricalDtype

QualVarCat = CategoricalDtype(categories=['good', 'vs', 'new'], ordered=False)

dt('QualVar') = df['QualVar'].astype(QualVarCat)

→ to give a categorical order

67