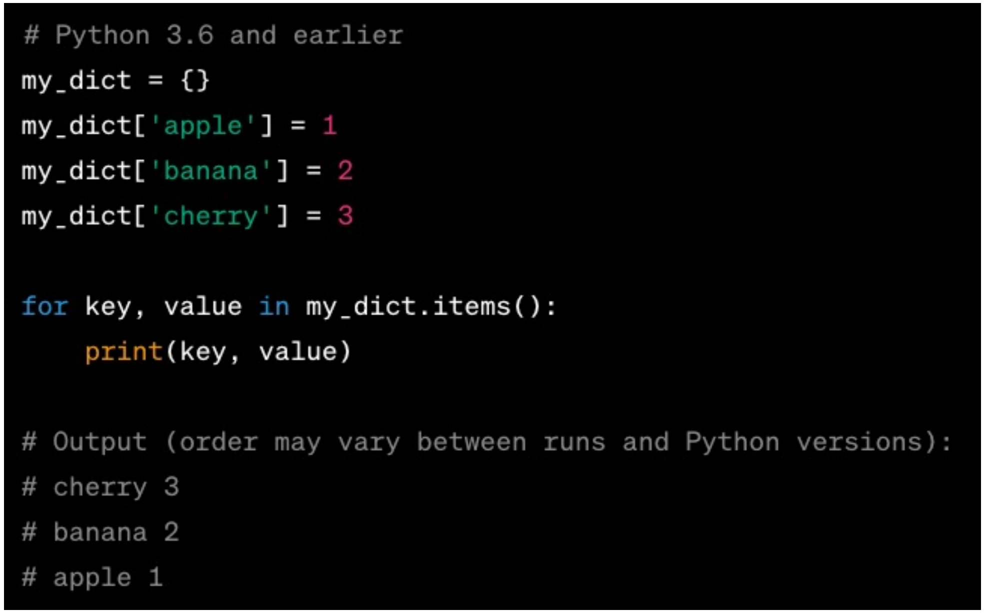
Lecture 9 Pandas

IDS 400

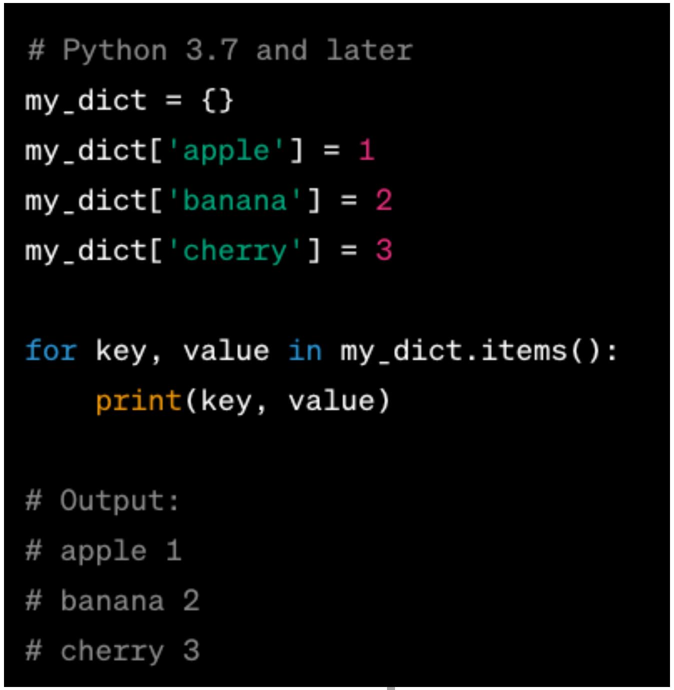
Programming for Data Science in Business

Announcement (Quiz2)

- Quiz 2- Question 10: Starting from Python 3.7 and later versions, dictionaries in Python maintain the order of insertion of key-value pairs.
- If you answer this question incorrectly, you will receive 4 points back.



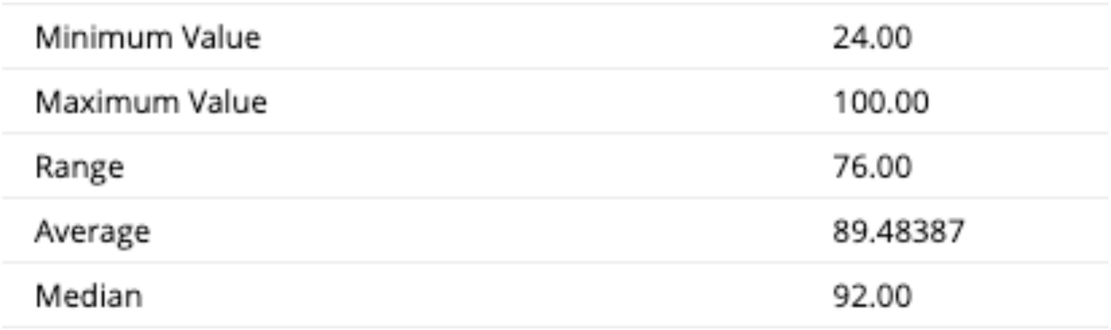
```
# Python 3.6 and earlier
my_dict = {}
my_dict['apple'] = 1
my_dict['banana'] = 2
my_dict['cherry'] = 3

for key, value in my_dict.items():
    print(key, value)

# Output (order may vary between runs and Python versions):
# cherry 3
# banana 2
# apple 1
```



```
# Python 3.7 and later
my_dict = {}
my_dict['apple'] = 1
my_dict['banana'] = 2
my_dict['cherry'] = 3

for key, value in my_dict.items():
    print(key, value)

# Output:
# apple 1
# banana 2
# cherry 3
```

Announcement (Quiz2)

Statistics for Quiz 2



Minimum Value	24.00
Maximum Value	100.00
Range	76.00
Average	89.48387
Median	92.00

Assignment

Assignment 4 due date has been **extended** to **Monday Oct 23rd at 6PM**.

Assignment 5 is available on Blackboard.

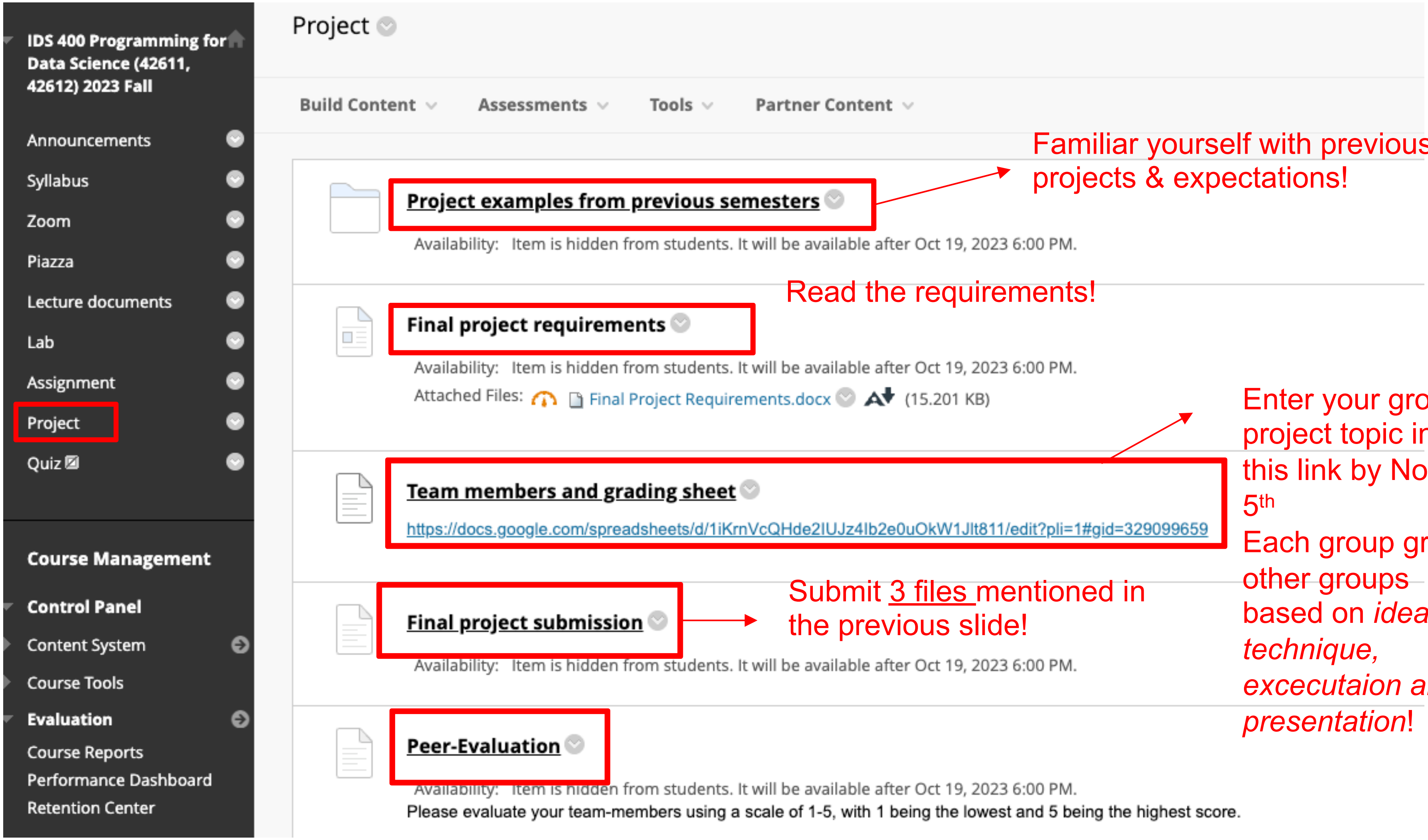
- 1 week to work on it.

Tentative schedule

Date	Lecture Number	Topics
08/24	Lecture 1	Introduction
08/31	Lecture 2	Basic
09/07	Lecture 3	Condition
09/14	Lecture 4	Loop
09/21	Lecture 5	String + **Quiz 1**
09/28	Lecture 6	Type
10/05	Lecture 7	Function
10/12	Lecture 8	File + **Quiz 2**
10/19	Lecture 9	Pandas
10/26	Lecture 10	Numpy
11/02	Lecture 11	Machine Learning
11/09	Lecture 12	Visualization
11/16	Lecture 13	Web Scraping & Deep Learning
11/23	***Thanksgiving***	***No lecture***
11/30	**Final presentation**	In class presentation
12/05	**Project submission due**	

Final Project

- See your teammates on the link below (please sign in with your **UIC email**) and enter your final project topic by Nov. 5th (**Blackboard → Project →** "**Team members and grading sheet**")
  - https://docs.google.com/spreadsheets/d/1iKrnVcQHde2IUJz4Ib2e0uOkW1Jlt811/edit#gid=329099659
- Please refer to the "**Final Project Requirement.docs"** under the project section on **blackboard**
- Each group should present within **8 minutes** (We will dedicate 2 minutes for Q&A after each presentation)
- Files to submit for the **Final Project** include:
  - Your code (**ipynb** format)
  - Report (**word document** format)
  - Presentation (**ppt** format)



IDS 400 Programming for Data Science (42611, 42612) 2023 Fall

- Announcements
- Syllabus
- Zoom
- Piazza
- Lecture documents
- Lab
- Assignment
- Project
- Quiz

Course Management

- Control Panel
- Content System
- Course Tools
- Evaluation
- Course Reports
- Performance Dashboard
- Retention Center

Project

Project examples from previous semesters

Availability: Item is hidden from students. It will be available after Oct 19, 2023 6:00 PM.

Final project requirements

Availability: Item is hidden from students. It will be available after Oct 19, 2023 6:00 PM.

Attached Files: Final Project Requirements.docx (15.201 KB)

Team members and grading sheet

https://docs.google.com/spreadsheets/d/1iKrnVcQHde2IUJz4lb2e0uOkW1Jlt811/edit?pli=1#gid=329099659

Final project submission

Availability: Item is hidden from students. It will be available after Oct 19, 2023 6:00 PM.

Peer-Evaluation

Availability: Item is hidden from students. It will be available after Oct 19, 2023 6:00 PM.

Please evaluate your team-members using a scale of 1-5, with 1 being the lowest and 5 being the highest score.

Familiar yourself with previous projects & expectations!

Read the requirements!

Enter your group project topic in this link by Nov. 5th

Each group grade other groups based on *idea, technique, excecutaion and presentation*!

Submit 3 files mentioned in the previous slide!

Datasets for Final Project

- Kaggle
  - https://www.kaggle.com/datasets/
- UC Irvine
  - https://archive.ics.uci.edu/ml/datasets
- EYARC
  - https://www.ey.com/en_us/about-us/ey-foundation-and-university-relations/academic-resource-center
- UI / Deloitte Center
  - https://www.centerforanalytics.giesbusiness.illinois.edu
- NE HUBAE
  - https://www.hubae.org
- ICPSR
  - https://www.icpsr.umich.edu/web/pages/
- UCI ML repository
  - https://archive.ics.uci.edu/ml/index.php
- Financial data (e.g., WRDS,CRSP, Audit Analytics)
  - https://www.sec.gov/dera/data/financial-statement-data-sets
- Cases published in journals
- Harvard Business Cases
- Datacamp
- Other Public data
  - Data.gov; https://datausa.io/; State governments; Just Google something (like cost of living by state)

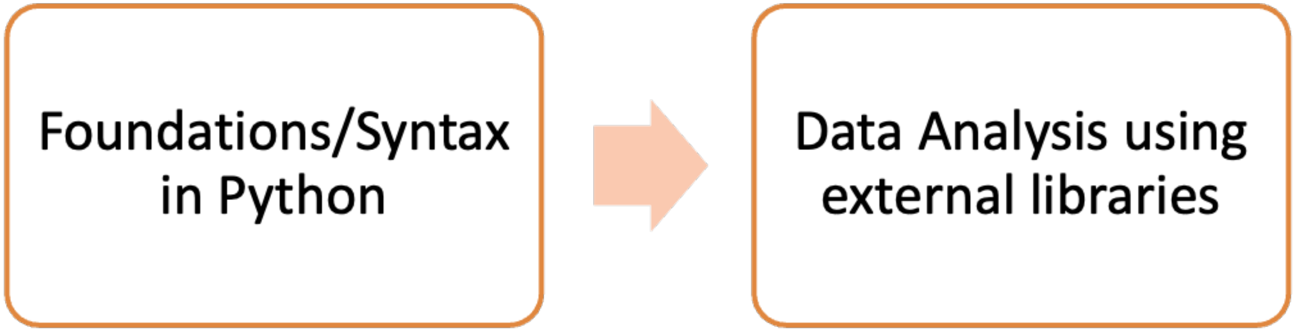
Final Project

- Useful links (about Numpy, Pandas and visualization techniques):
  - https://www.hackerearth.com/practice/machine-learning/data-manipulation-visualisation-r-python/tutorial-data-manipulation-numpy-pandas-python/tutorial/
  - https://cloudxlab.com/blog/numpy-pandas-introduction/
  - https://realpython.com/numpy-tutorial/#practical-example-1-implementing-a-maclaurin-series
  - https://towardsdatascience.com/the-simplest-data-science-project-using-pandas-matplotlib-9d7042e7ce6f
  - https://ourcodingclub.github.io/tutorials/pandas-python-intro/

Final Project Requirements

- Motivation/Research Questions.
  - Why your project topic is important.
- The Analytics Component.
  - Explaining how you choose different methods to analyze data, what analytical models you use, which model performs the best (and why).
- Findings & Insight.
  - What can be learned from your project, and how your project help the businesses improve their decisions.

Course topics



Foundations/Syntax in Python

Data Analysis using external libraries

Use External Libraries

- Use ***help()*** function.
- API documentation
  - A reference manual that has all of the information you need to work with the API. It tells the developer/partner/consumer everything that is possible with the API, and how to get started.
  - Pandas API reference: https://pandas.pydata.org/docs/reference/index.html
  - Numpy API reference: https://numpy.org/doc/stable/reference/
  - Scipy API reference: https://docs.scipy.org/doc/scipy/
  - Scikit-learn API reference: https://scikit-learn.org/stable/modules/classes.html
  - Matplotlib API reference: https://matplotlib.org/3.5.1/api/index.html
  - Seaborn API reference: https://seaborn.pydata.org/api.html

For This Class

- Pandas

Reference of this slide: https://www.w3schools.com/python/pandas/default.asp

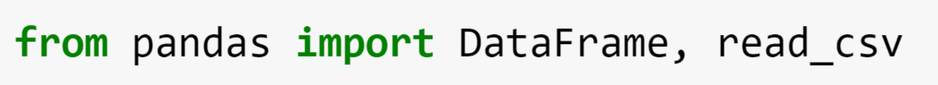
File processing

- Panel Data System
- Pandas is a Python library used for working with data sets.
- Pandas is an open-source library which has functions for analyzing, cleaning, exploring, and manipulating data.
- High-performance, easy-to-use data structures and data analysis tools
- Built for the Python programming language
- Key components
    - Series (columns)
    - DataFrame

Import module

- General syntax to import specific functions in a library:

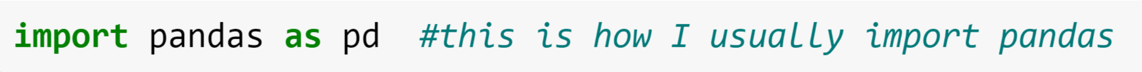
```
from <library> import <specific library function>
```



```
from pandas import DataFrame, read_csv
```

- General syntax to import a library but no functions:

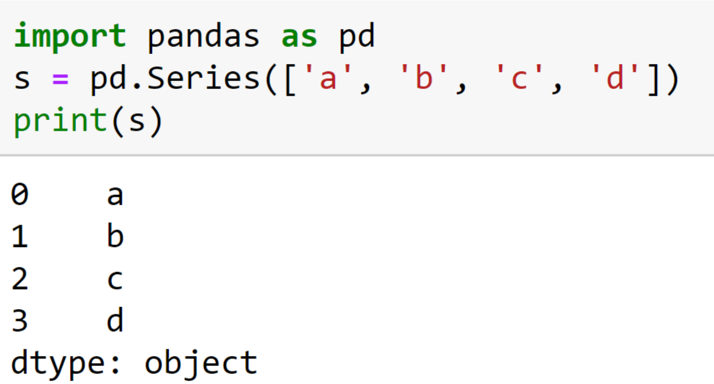
```
import <library> as <give the library a nickname/alias>
```



```
import pandas as pd  #this is how I usually import pandas
```

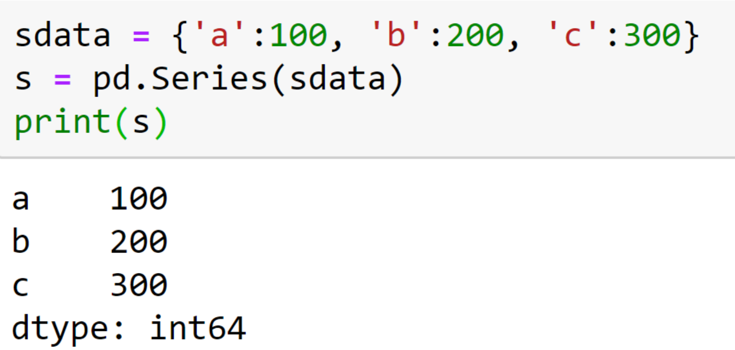
Data structure: Series

- A Pandas Series is like a column in a table.
- It is a one-dimensional array holding data of any type.
- One-dimensional array like object containing data and labels (or index).
- Lots of ways to build a Series



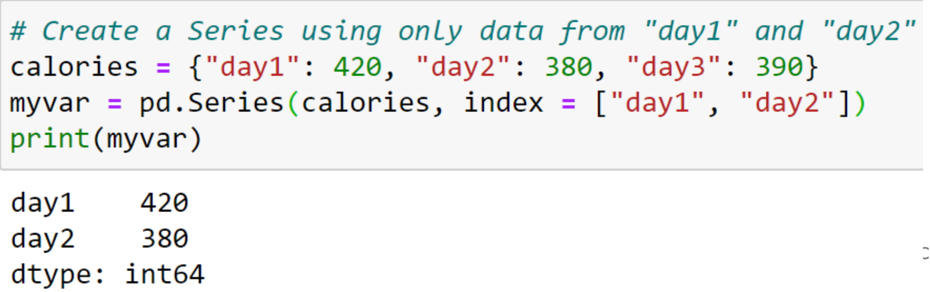
```python
import pandas as pd
s = pd.Series(['a', 'b', 'c', 'd'])
print(s)
```

```
0    a
1    b
2    c
3    d
dtype: object
```



```python
sdata = {'a':100, 'b':200, 'c':300}
s = pd.Series(sdata)
print(s)
```

```
a    100
b    200
c    300
dtype: int64
```

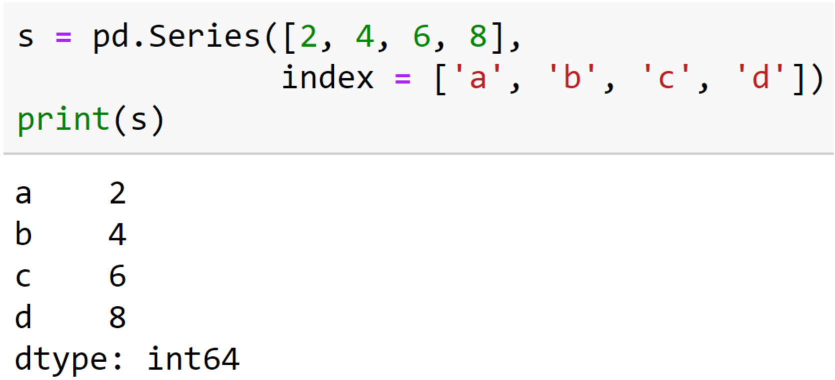


```python
# Create a Series using only data from "day1" and "day2"
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories, index = ["day1", "day2"])
print(myvar)
```

```
day1    420
day2    380
dtype: int64
```

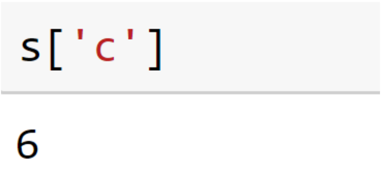
Series – working with index

- A Series index can be specified.
- Single values can be selected by index.
- Multiple values can be selected with multiple indexes.



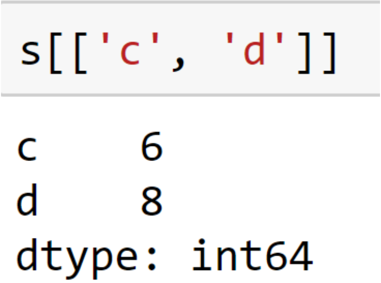
```
s = pd.Series([2, 4, 6, 8],
              index = ['a', 'b', 'c', 'd'])
print(s)
```

```
a    2
b    4
c    6
d    8
dtype: int64
```



```
s['c']
```

```
6
```

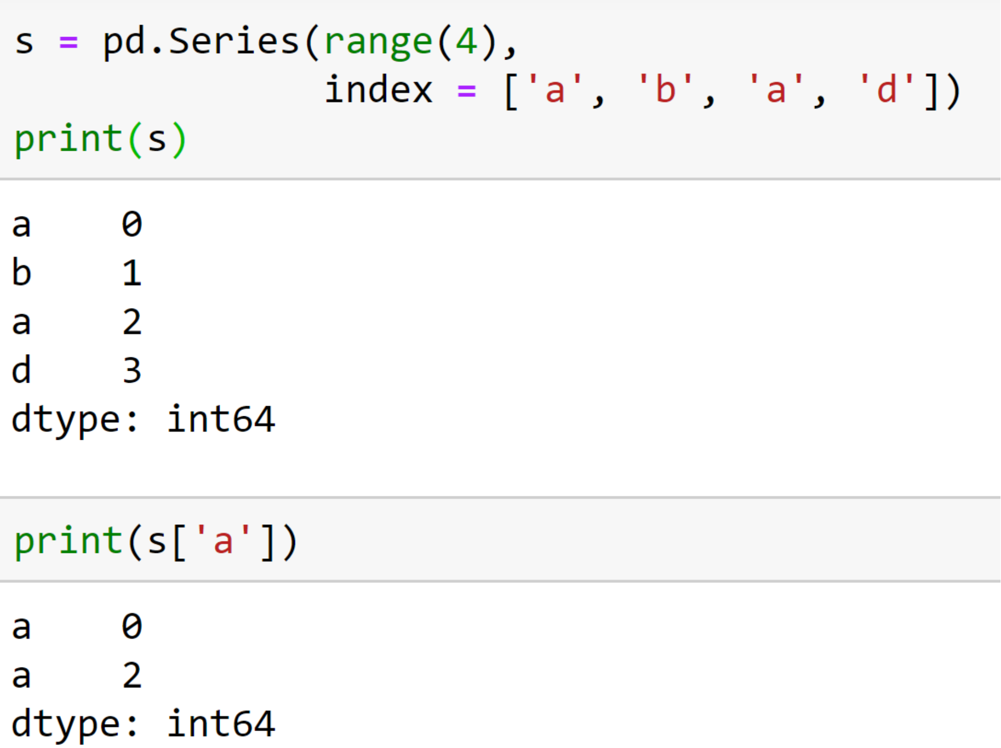


```
s[['c', 'd']]
```

```
c    6
d    8
dtype: int64
```

Series – working with index

- Think of a Series as a fixed-length order
- However, unlike dictionary, index items don’t have to be unique.



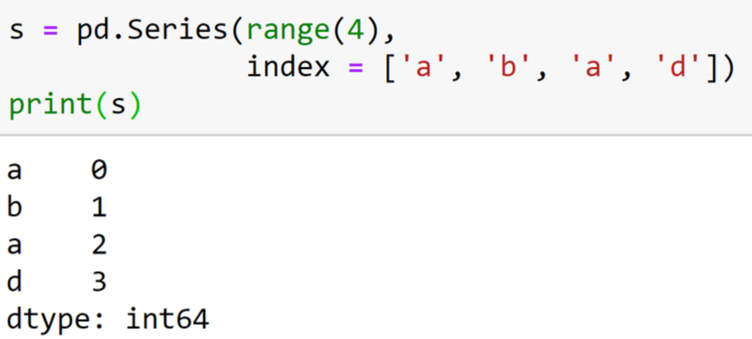
```
s = pd.Series(range(4),
              index = ['a', 'b', 'a', 'd'])
print(s)
```

```
a    0
b    1
a    2
d    3
dtype: int64
```

```
print(s['a'])
```

```
a    0
a    2
dtype: int64
```

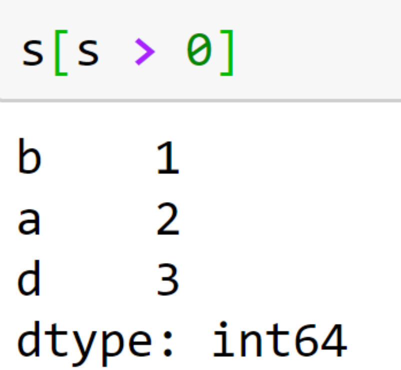
Series operations



```
s = pd.Series(range(4),
              index = ['a', 'b', 'a', 'd'])
print(s)
```

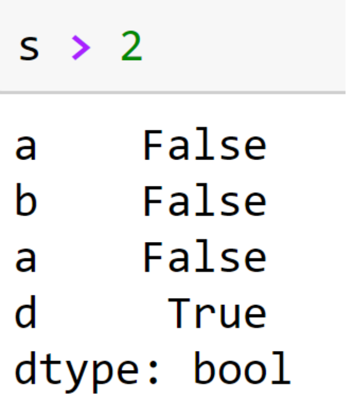
```
a    0
b    1
a    2
d    3
dtype: int64
```

- Filtering



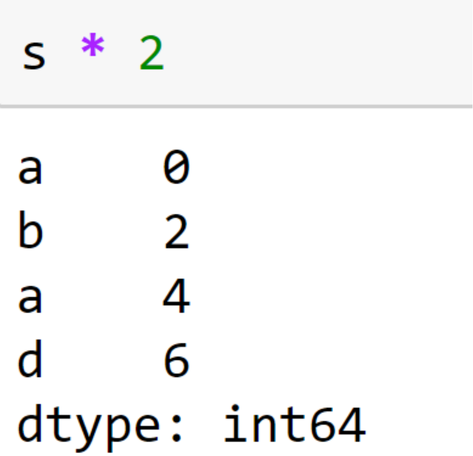
```
s[s > 0]
```

```
b    1
a    2
d    3
dtype: int64
```



```
s > 2
```

```
a    False
b    False
a    False
d     True
dtype: bool
```



```
s * 2
```

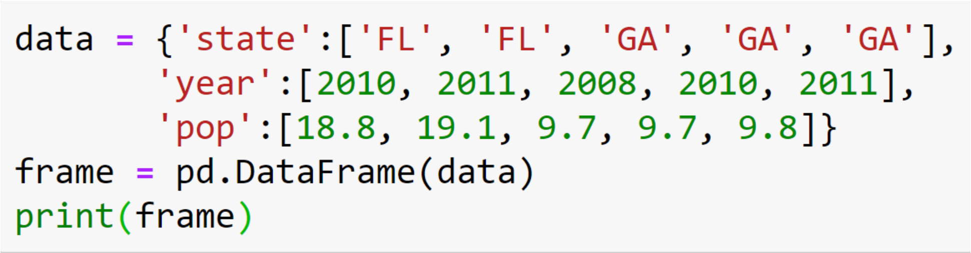
```
a    0
b    2
a    4
d    6
dtype: int64
```

Data structure: DataFrame

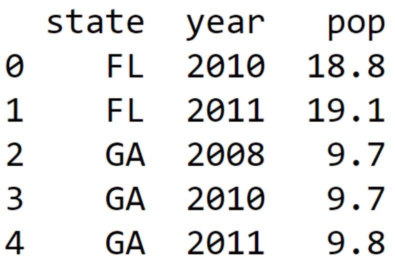
- Data sets in Pandas are usually multi-dimensional tables, called ***DataFrames***.
- Spread sheet-like data structure containing an order collection of columns.
- Has both a row and column index.
- Series is like a column, a DataFrame is the whole table.

Create data

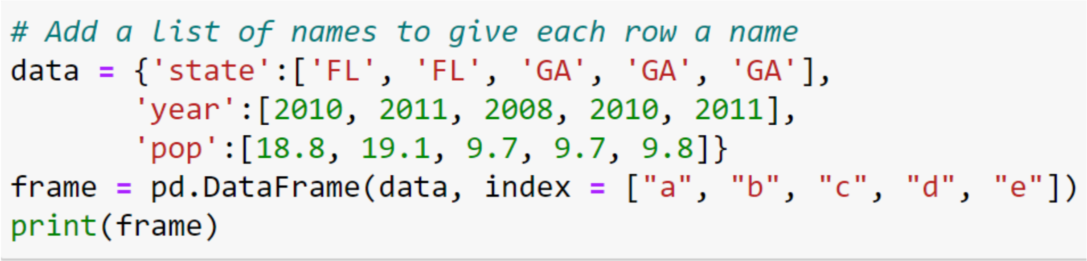
- Creation with dictionary of equal-length lists.



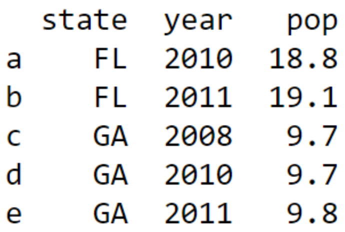
```
data = {'state':['FL', 'FL', 'GA', 'GA', 'GA'],
        'year':[2010, 2011, 2008, 2010, 2011],
        'pop':[18.8, 19.1, 9.7, 9.7, 9.8]}
frame = pd.DataFrame(data)
print(frame)
```



```
  state  year   pop
0    FL  2010  18.8
1    FL  2011  19.1
2    GA  2008   9.7
3    GA  2010   9.7
4    GA  2011   9.8
```



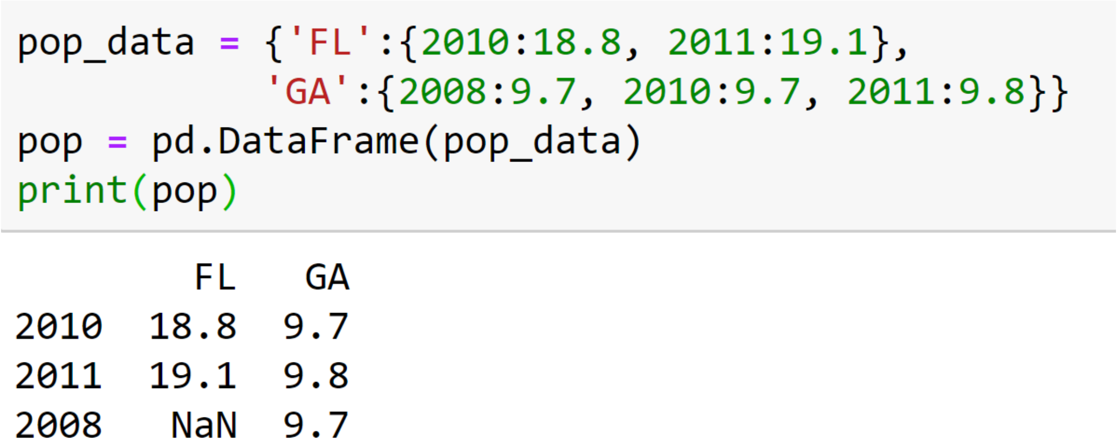
```
# Add a list of names to give each row a name
data = {'state':['FL', 'FL', 'GA', 'GA', 'GA'],
        'year':[2010, 2011, 2008, 2010, 2011],
        'pop':[18.8, 19.1, 9.7, 9.7, 9.8]}
frame = pd.DataFrame(data, index = ["a", "b", "c", "d", "e"])
print(frame)
```



```
  state  year   pop
a    FL  2010  18.8
b    FL  2011  19.1
c    GA  2008   9.7
d    GA  2010   9.7
e    GA  2011   9.8
```

Create data

- Creation with dict of dicts.

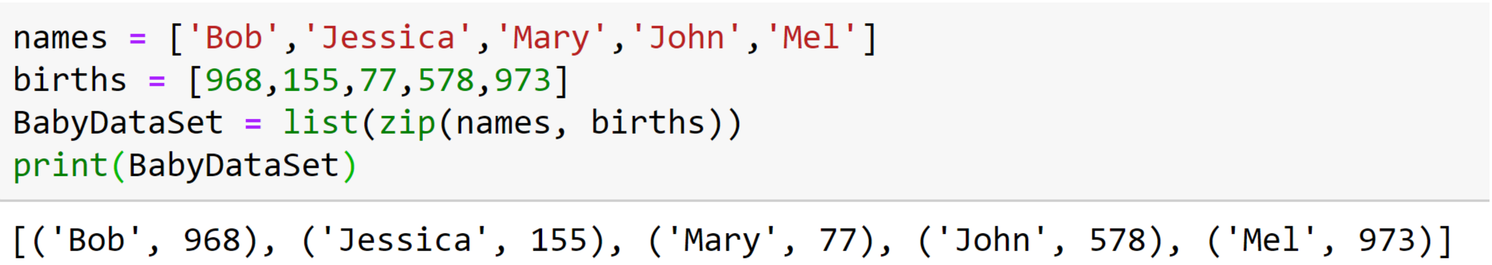


```
pop_data = {'FL':{2010:18.8, 2011:19.1},
            'GA':{2008:9.7, 2010:9.7, 2011:9.8}}
pop = pd.DataFrame(pop_data)
print(pop)
```

```
        FL   GA
2010  18.8  9.7
2011  19.1  9.8
2008   NaN  9.7
```

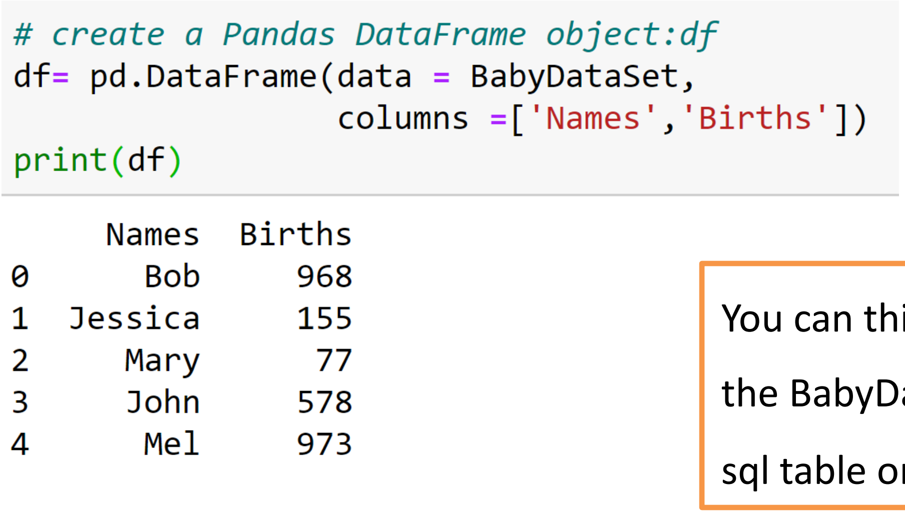
Create data

- Creation with lists.



```
names = ['Bob','Jessica','Mary','John','Mel']
births = [968,155,77,578,973]
BabyDataSet = list(zip(names, births))
print(BabyDataSet)
```

```
[('Bob', 968), ('Jessica', 155), ('Mary', 77), ('John', 578), ('Mel', 973)]
```



```
# create a Pandas DataFrame object:df
df= pd.DataFrame(data = BabyDataSet,
                 columns =['Names','Births'])
print(df)
```

```
     Names  Births
0      Bob     968
1  Jessica     155
2     Mary      77
3     John     578
4      Mel     973
```

You can think of df holding the contents of the BabyDataSet in a format similar to a sql table or an excel spread sheet.

Save data

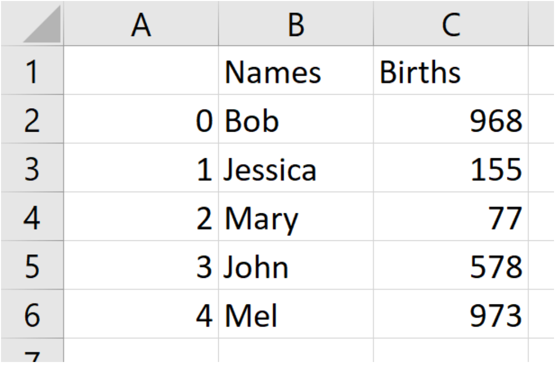
For Google Colab Only:

```
from google.colab import drive
drive.mount("/content/drive")

%cd /content/drive/MyDrive/Ids400
```

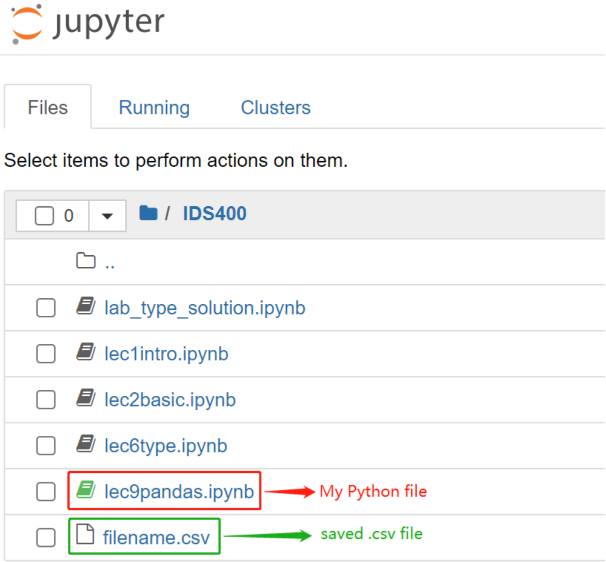
- You can export the dataframe to a csv file.

```
#df.to_csv('filename')
df.to_csv('filename.csv',index = True, header = True)
```

- The file will be saved in the same location as your python file unless specified otherwise.



	A	B	C
1		Names	Births
2	0	Bob	968
3	1	Jessica	155
4	2	Mary	77
5	3	John	578
6	4	Mel	973
7			



Select items to perform actions on them.

□ 0 ▾ / IDS400

..

□ lab_type_solution.ipynb

□ lec1intro.ipynb

□ lec2basic.ipynb

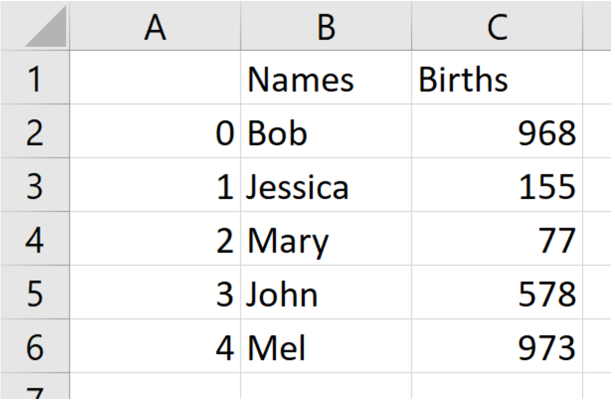
□ lec6type.ipynb

□ lec9pandas.ipynb → My Python file

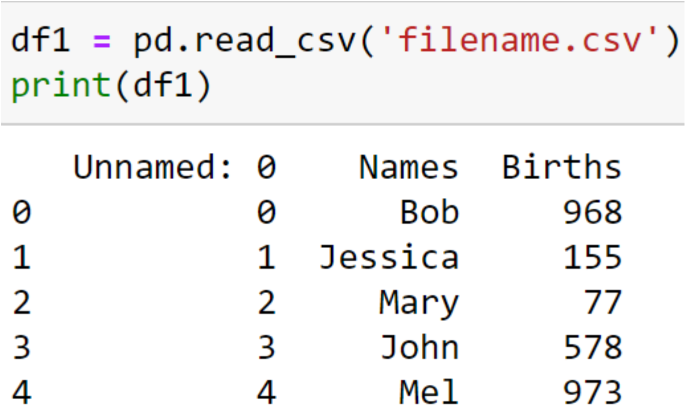
□ filename.csv → saved .csv file

Reading/Loading data – Read csv

- To pull in the csv file, we can use the pandas function: ***read_csv***

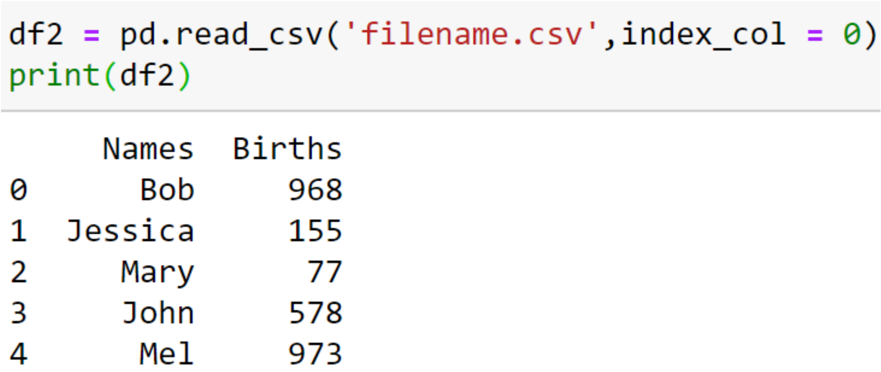


	A	B	C
1		Names	Births
2	0	Bob	968
3	1	Jessica	155
4	2	Mary	77
5	3	John	578
6	4	Mel	973
7			



```
df1 = pd.read_csv('filename.csv')
print(df1)
```

```
   Unnamed: 0    Names  Births
0           0      Bob     968
1           1  Jessica     155
2           2     Mary      77
3           3     John     578
4           4      Mel     973
```

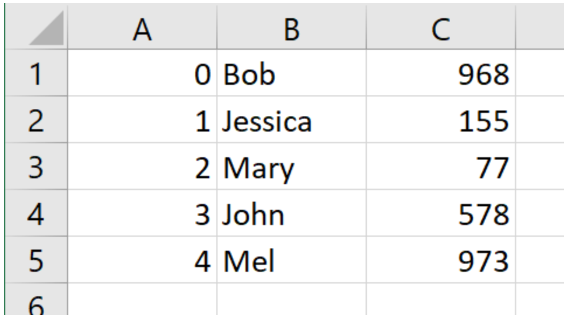


```
df2 = pd.read_csv('filename.csv',index_col = 0)
print(df2)
```

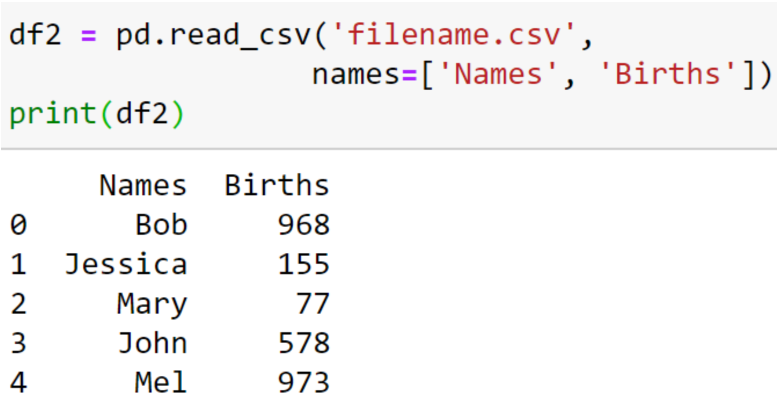
```
     Names  Births
0      Bob     968
1  Jessica     155
2     Mary      77
3     John     578
4      Mel     973
```

Loading data – Read csv

- If we wanted to give the columns specific names, we would have to pass another parameter called “names”.



	A	B	C
1	0	Bob	968
2	1	Jessica	155
3	2	Mary	77
4	3	John	578
5	4	Mel	973
6			

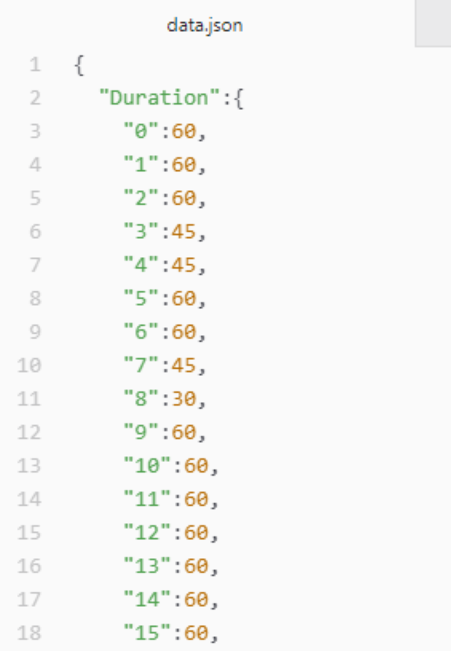


```
df2 = pd.read_csv('filename.csv',
                  names=['Names', 'Births'])
print(df2)
```

```
     Names  Births
0      Bob     968
1  Jessica     155
2     Mary      77
3     John     578
4      Mel     973
```

Loading data – Read json

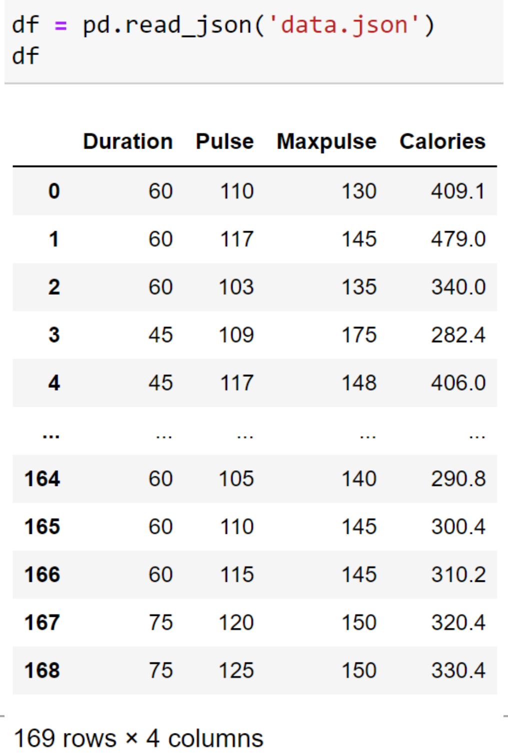
- Big data sets are often stored or extracted as ***JSON***.
- JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.
- JSON objects **have the same format as Python dictionaries**.



data.json

```
{
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60,
    "6":60,
    "7":45,
    "8":30,
    "9":60,
    "10":60,
    "11":60,
    "12":60,
    "13":60,
    "14":60,
    "15":60,
```

......



```python
df = pd.read_json('data.json')
df
```

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.1
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
...	...	...	...	...
164	60	105	140	290.8
165	60	110	145	300.4
166	60	115	145	310.2
167	75	120	150	320.4
168	75	125	150	330.4

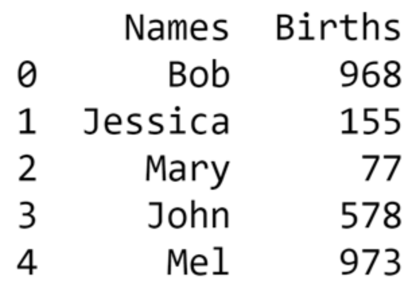
169 rows × 4 columns

Prepare data

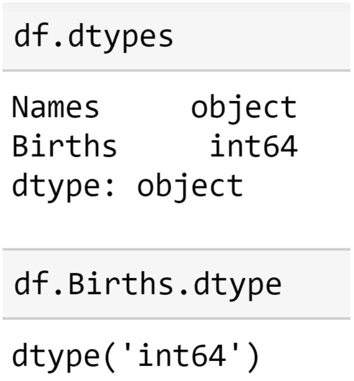
- **df.head()** returns the first 5 records.
- **df.head(n)** returns the first n records.
- **df.tail()** returns the last 5 records.
- **df.tail(n)** returns the last n records.

Prepare data

- Check data type of the columns.
- Find all the unique records of the "Names" column.
- Obtain all descriptive statistics.



```
     Names  Births
0      Bob     968
1  Jessica     155
2     Mary      77
3     John     578
4      Mel     973
```

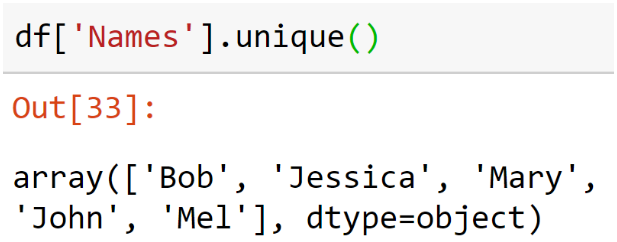


```
df.dtypes
```

```
Names       object
Births       int64
dtype: object
```

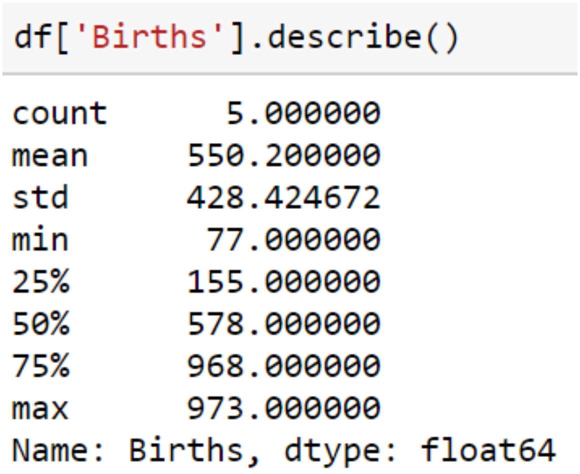
```
df.Births.dtype
```

```
dtype('int64')
```



```
df['Names'].unique()
```

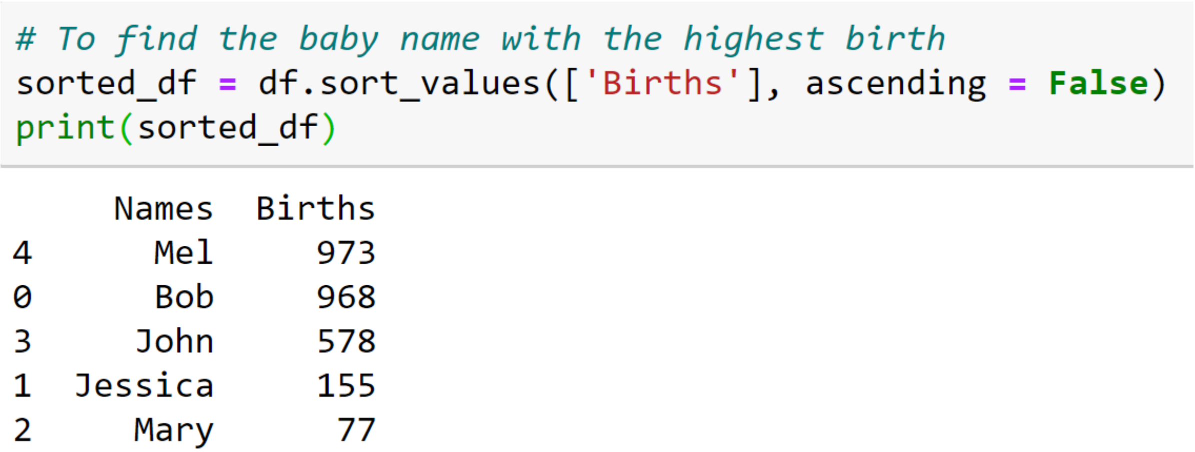
```
Out[33]:

array(['Bob', 'Jessica', 'Mary',
'John', 'Mel'], dtype=object)
```



```
df['Births'].describe()
```

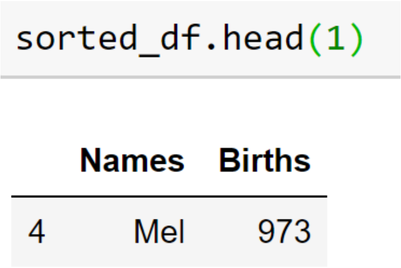
```
count      5.000000
mean     550.200000
std      428.424672
min       77.000000
25%      155.000000
50%      578.000000
75%      968.000000
max      973.000000
Name: Births, dtype: float64
```

Analyze data



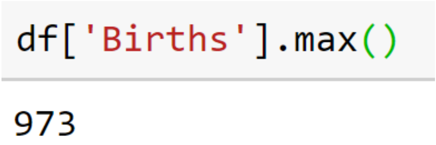
```
# To find the baby name with the highest birth
sorted_df = df.sort_values(['Births'], ascending = False)
print(sorted_df)
```

```
     Names  Births
4      Mel     973
0      Bob     968
3     John     578
1  Jessica     155
2     Mary      77
```



```
sorted_df.head(1)
```

	Names	Births
4	Mel	973



```
df['Births'].max()
```

```
973
```

Analyze data

- **df['Names’]** - This is the entire list of baby names, the entire Names column.
- **df['Births’]** - This is the entire list of Births in the year 1980, the entire Births column.
- **df['Births'].max()** -This is the maximum value found in the Births column.
- **df['Births'] == df['Births'].max()** *IS EQUAL TO* [Find all of the records in the Births column where it is equal to 973].
- **df['Names'][df['Births'] == df['Births'].max()]** *IS EQUAL TO* Select all of the records in the Names column WHERE [The Births column is equal to 973].

Exercise

- Start with creating a dataframe with the following data related to COVID 19 data for Apr 1st:

California, confirmed 9937, recovered 111, deaths 216

Michigan, confirmed 9334, recovered 15, deaths 337

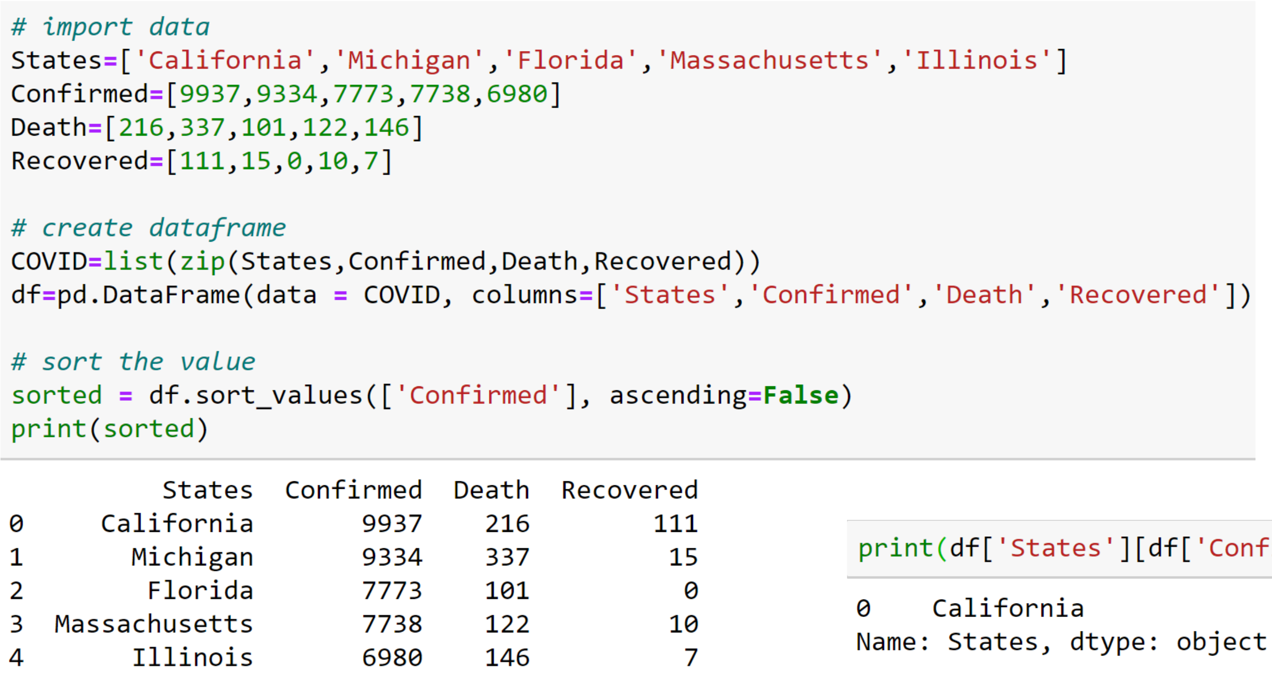
Florida, confirmed 7773, recovered 0, deaths 101

Massachusetts, confirmed 7738, recovered 10, deaths 122

Illinois, confirmed 6980, recovered 7, deaths 146

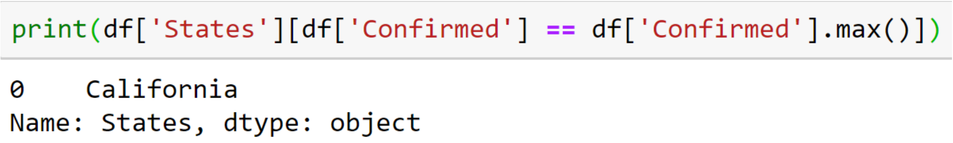
Task: Using Python to find out which state has the highest confirmed cases, highest recovered number and highest deaths number.

Exercise



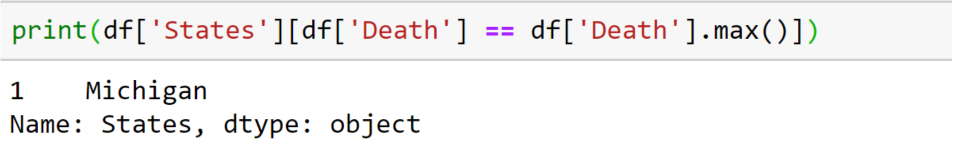
```
# import data
States=['California','Michigan','Florida','Massachusetts','Illinois']
Confirmed=[9937,9334,7773,7738,6980]
Death=[216,337,101,122,146]
Recovered=[111,15,0,10,7]

# create dataframe
COVID=list(zip(States,Confirmed,Death,Recovered))
df=pd.DataFrame(data = COVID, columns=['States','Confirmed','Death','Recovered'])

# sort the value
sorted = df.sort_values(['Confirmed'], ascending=False)
print(sorted)
```

```
          States  Confirmed  Death  Recovered
0     California       9937    216        111
1       Michigan       9334    337         15
2        Florida       7773    101          0
3  Massachusetts       7738    122         10
4       Illinois       6980    146          7
```



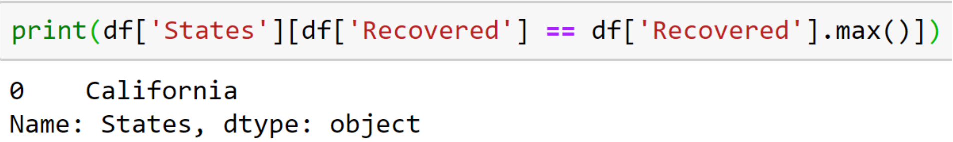
```
print(df['States'][df['Confirmed'] == df['Confirmed'].max()])
```

```
0    California
Name: States, dtype: object
```



```
print(df['States'][df['Death'] == df['Death'].max()])
```

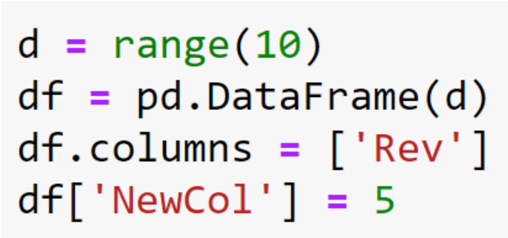
```
1    Michigan
Name: States, dtype: object
```



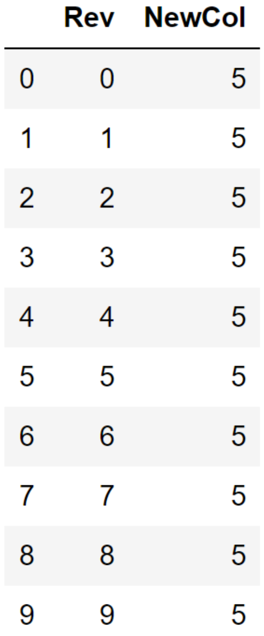
```
print(df['States'][df['Recovered'] == df['Recovered'].max()])
```

```
0    California
Name: States, dtype: object
```

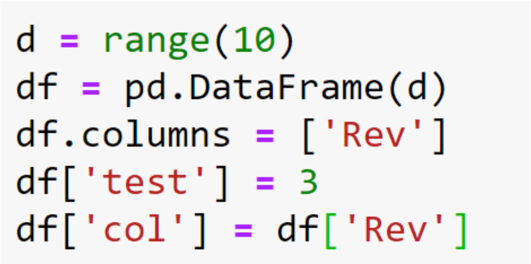
Column operations



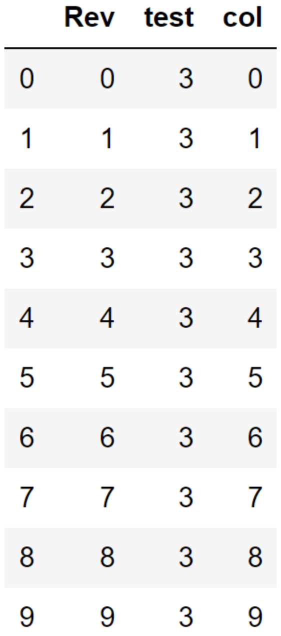
```
d = range(10)
df = pd.DataFrame(d)
df.columns = ['Rev']
df['NewCol'] = 5
```



	Rev	NewCol
0	0	5
1	1	5
2	2	5
3	3	5
4	4	5
5	5	5
6	6	5
7	7	5
8	8	5
9	9	5

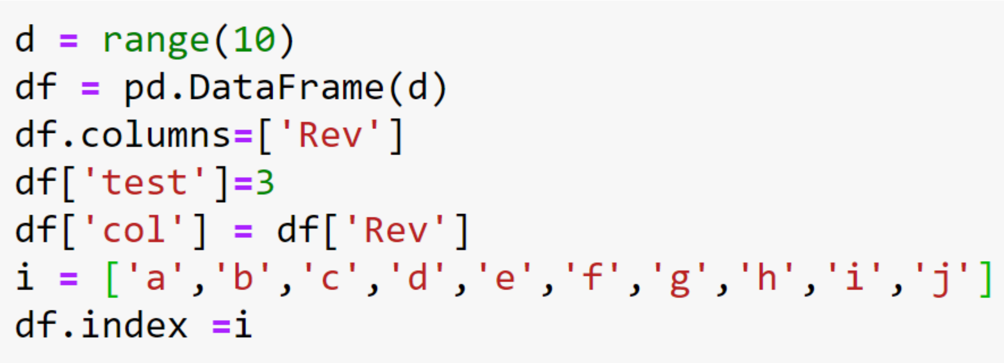


```
d = range(10)
df = pd.DataFrame(d)
df.columns = ['Rev']
df['test'] = 3
df['col'] = df['Rev']
```

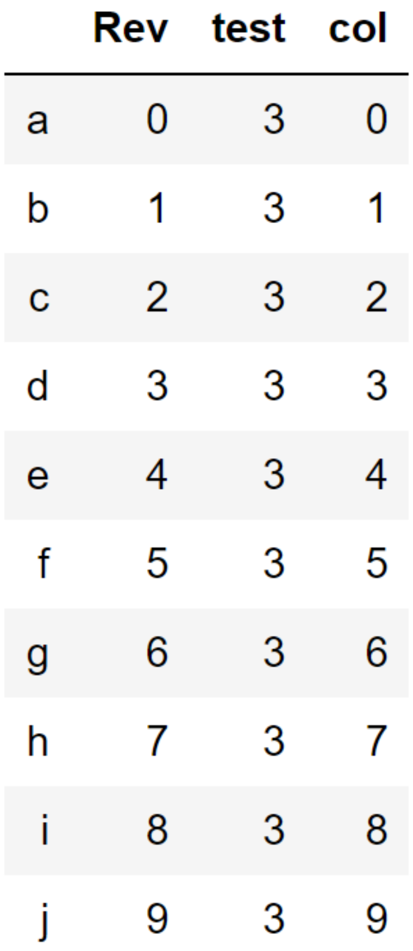


	Rev	test	col
0	0	3	0
1	1	3	1
2	2	3	2
3	3	3	3
4	4	3	4
5	5	3	5
6	6	3	6
7	7	3	7
8	8	3	8
9	9	3	9

Column operations



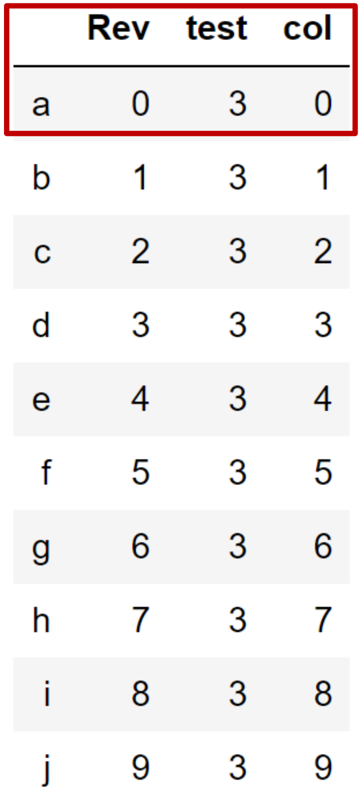
```
d = range(10)
df = pd.DataFrame(d)
df.columns=['Rev']
df['test']=3
df['col'] = df['Rev']
i = ['a','b','c','d','e','f','g','h','i','j']
df.index =i
```



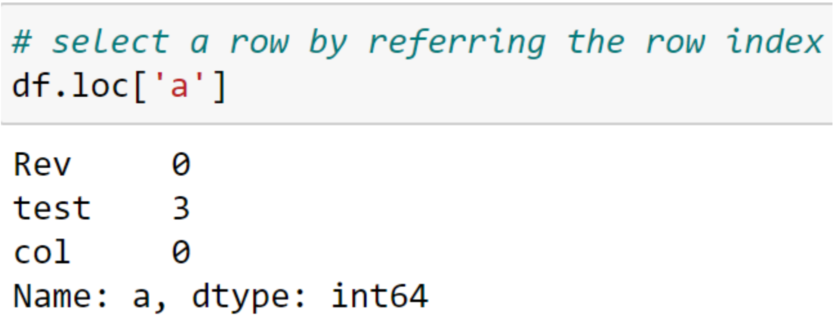
	Rev	test	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3
e	4	3	4
f	5	3	5
g	6	3	6
h	7	3	7
i	8	3	8
j	9	3	9

Row operations

- Pandas use the *loc* attribute to return one or more specified row(s)

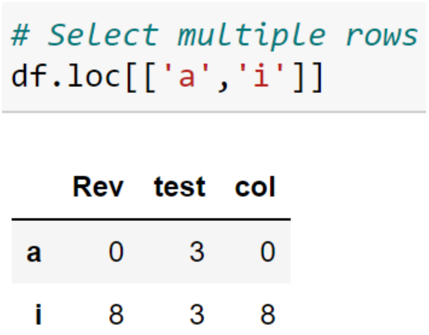


	Rev	test	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3
e	4	3	4
f	5	3	5
g	6	3	6
h	7	3	7
i	8	3	8
j	9	3	9



```
# select a row by referring the row index
df.loc['a']
```

```
Rev     0
test    3
col     0
Name: a, dtype: int64
```

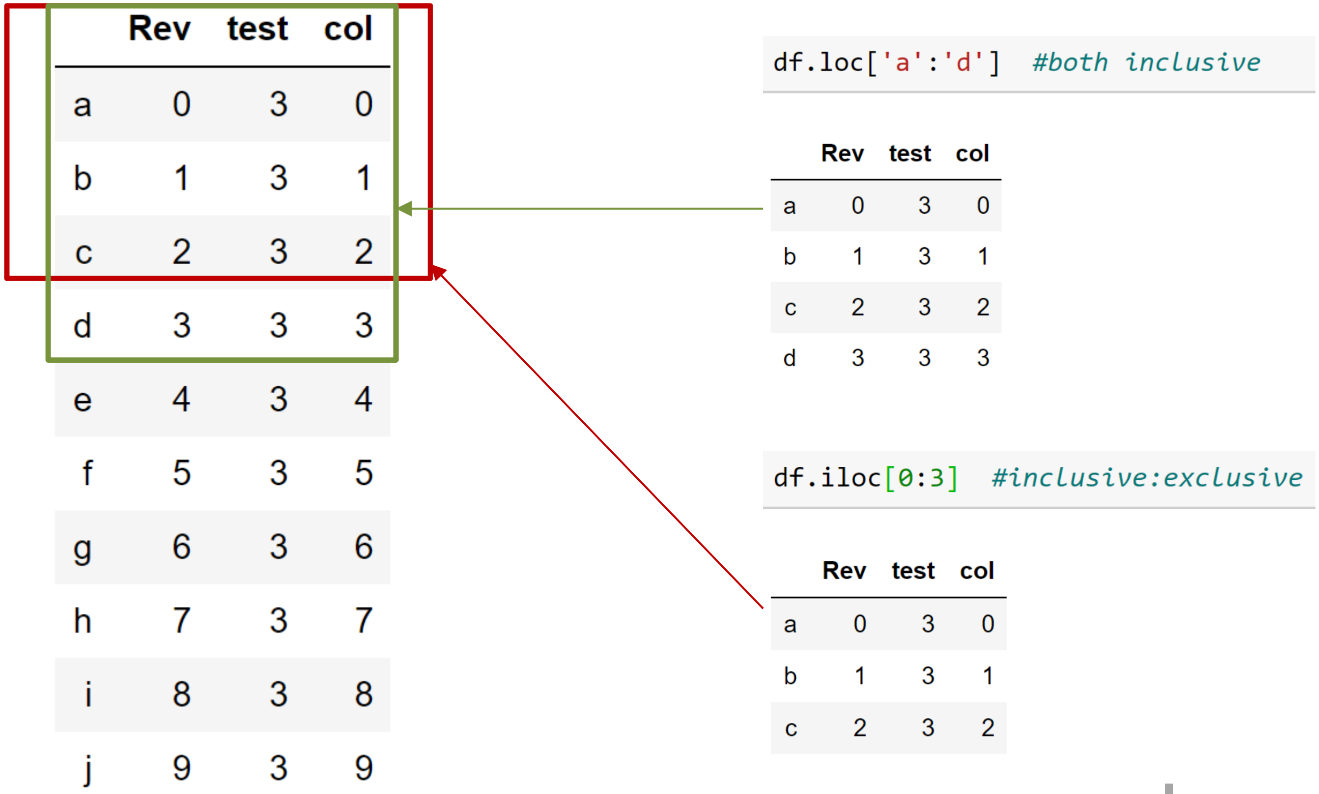


```
# Select multiple rows
df.loc[['a','i']]
```

	Rev	test	col
a	0	3	0
i	8	3	8

Row operations

- Difference between loc and iloc:
  - Loc returns rows/columns based on label from the index
  - iloc returns rows/columns based on position of the index (i.e. integer values)



	Rev	test	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3
e	4	3	4
f	5	3	5
g	6	3	6
h	7	3	7
i	8	3	8
j	9	3	9

```
df.loc['a':'d']  #both inclusive
```

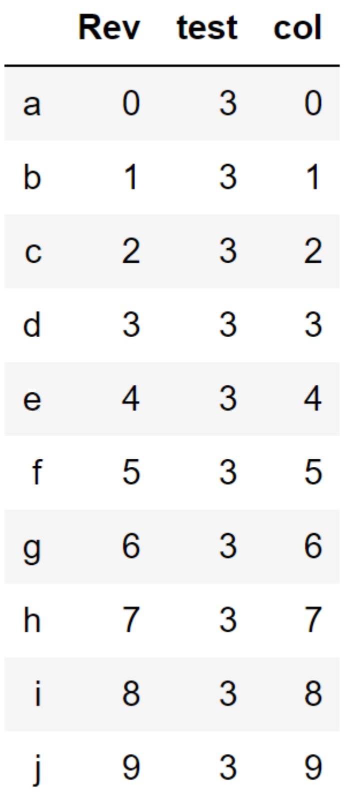
	Rev	test	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3

```
df.iloc[0:3]  #inclusive:exclusive
```

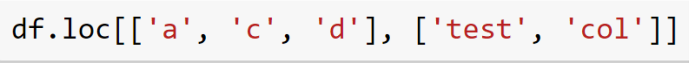
	Rev	test	col
a	0	3	0
b	1	3	1
c	2	3	2

Row and column operations

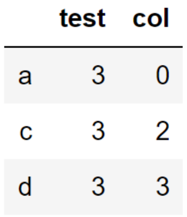
- Select rows and columns simultaneously.



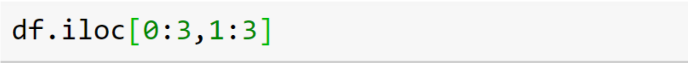
	Rev	test	col
a	0	3	0
b	1	3	1
c	2	3	2
d	3	3	3
e	4	3	4
f	5	3	5
g	6	3	6
h	7	3	7
i	8	3	8
j	9	3	9



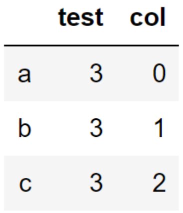
```
df.loc[['a', 'c', 'd'], ['test', 'col']]
```



	test	col
a	3	0
c	3	2
d	3	3

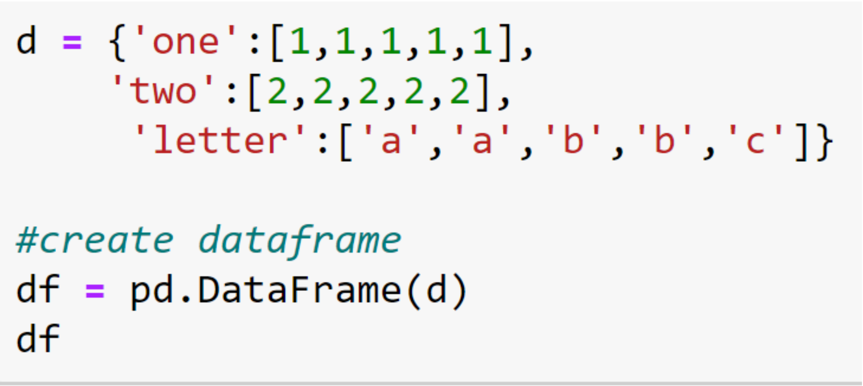


```
df.iloc[0:3,1:3]
```

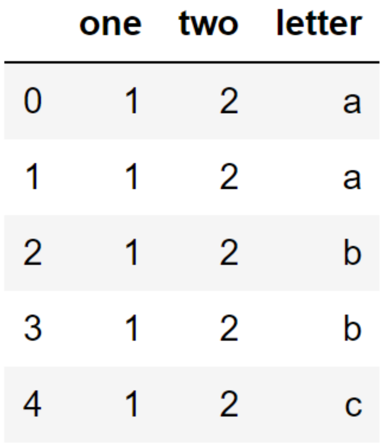


	test	col
a	3	0
b	3	1
c	3	2

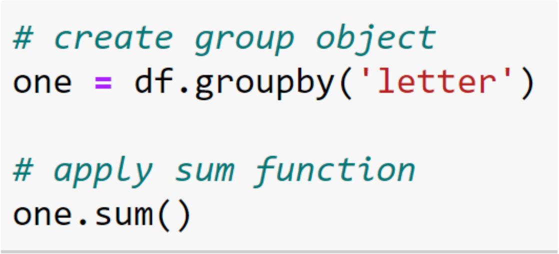
Groupby function



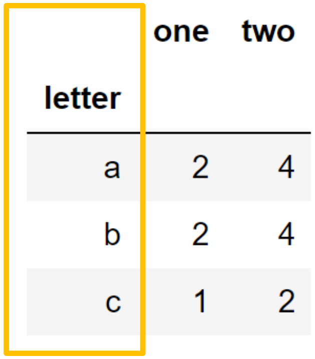
```
d = {'one':[1,1,1,1,1],
    'two':[2,2,2,2,2],
     'letter':['a','a','b','b','c']}

#create dataframe
df = pd.DataFrame(d)
df
```



	one	two	letter
0	1	2	a
1	1	2	a
2	1	2	b
3	1	2	b
4	1	2	c



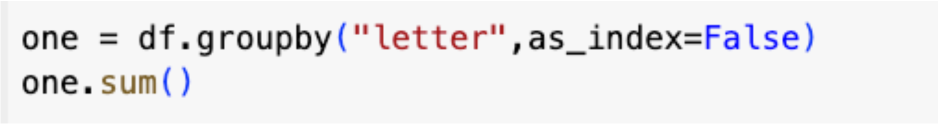
```
# create group object
one = df.groupby('letter')

# apply sum function
one.sum()
```



	one	two
letter		
a	2	4
b	2	4
c	1	2

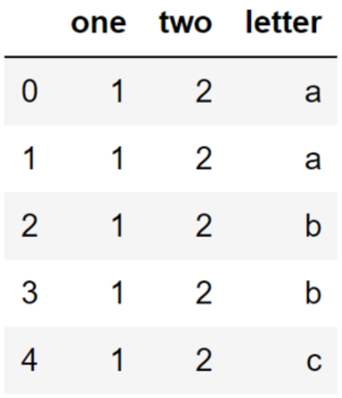
Notice “letter” is not a column of this dataframe.

This is actually the index of the dataframe.

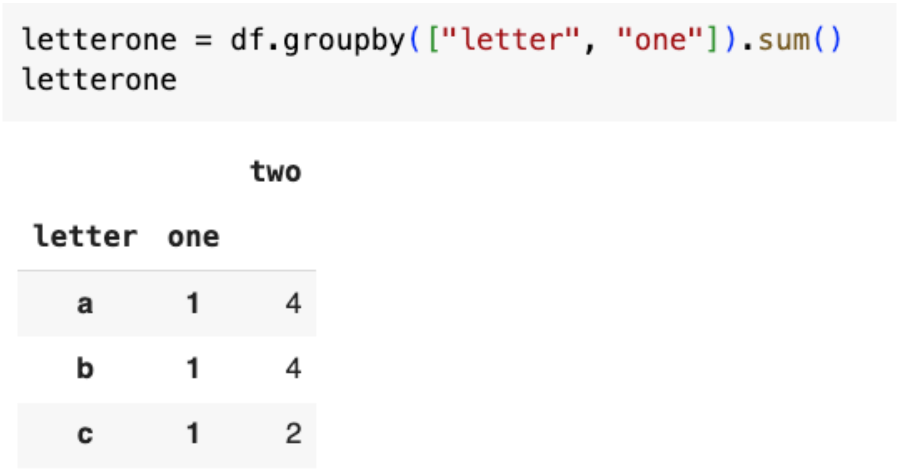


```
one = df.groupby("letter",as_index=False)
one.sum()
```

Groupby function

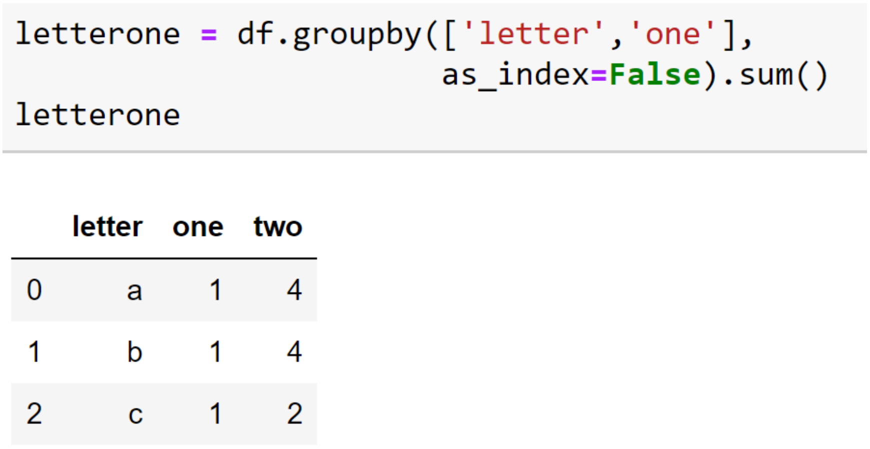


	one	two	letter
0	1	2	a
1	1	2	a
2	1	2	b
3	1	2	b
4	1	2	c



```
letterone = df.groupby(["letter", "one"]).sum()
letterone
```

		two
letter	one	
a	1	4
b	1	4
c	1	2

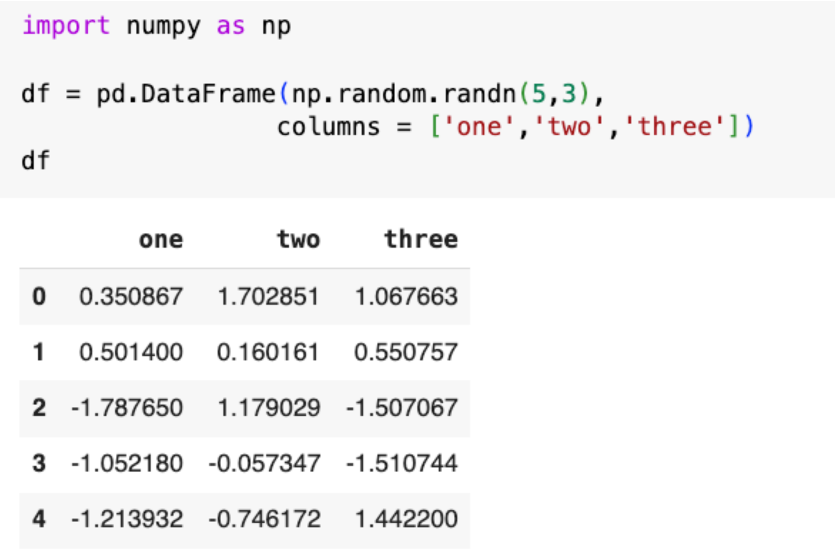


```
letterone = df.groupby(['letter','one'],
                       as_index=False).sum()
letterone
```

	letter	one	two
0	a	1	4
1	b	1	4
2	c	1	2

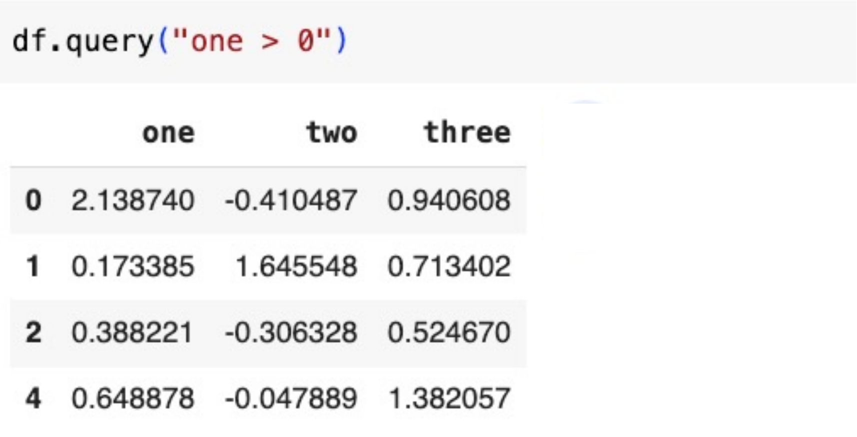
Pandas query data

- Use the query method where you can embed boolean expressions on columns with quotes.



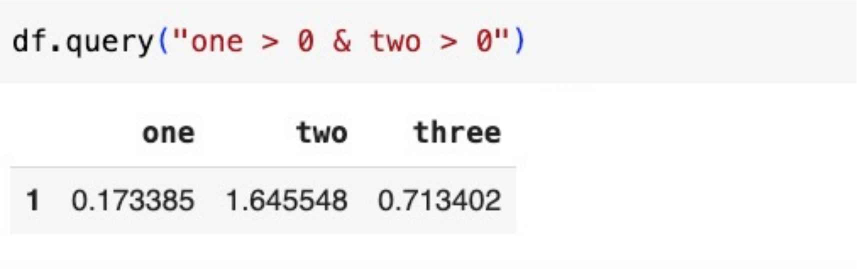
```python
import numpy as np

df = pd.DataFrame(np.random.randn(5,3),
                  columns = ['one','two','three'])
df
```

	one	two	three
0	0.350867	1.702851	1.067663
1	0.501400	0.160161	0.550757
2	-1.787650	1.179029	-1.507067
3	-1.052180	-0.057347	-1.510744
4	-1.213932	-0.746172	1.442200



```python
df.query("one > 0")
```

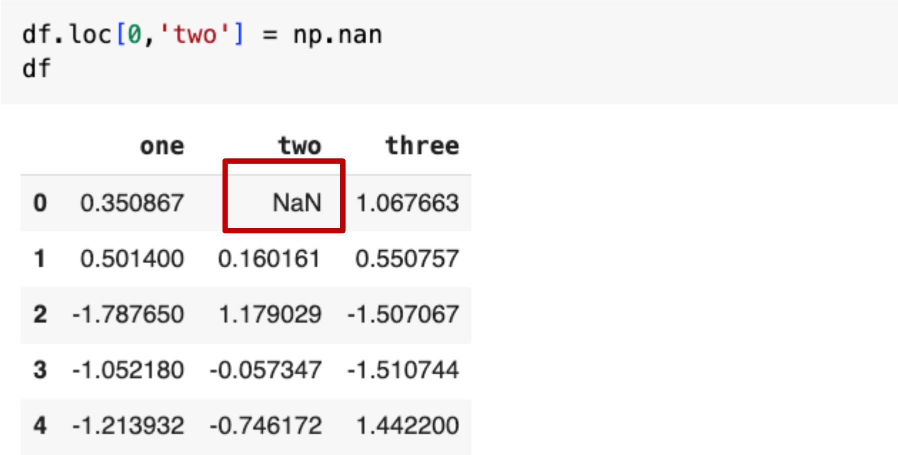
	one	two	three
0	2.138740	-0.410487	0.940608
1	0.173385	1.645548	0.713402
2	0.388221	-0.306328	0.524670
4	0.648878	-0.047889	1.382057



```python
df.query("one > 0 & two > 0")
```

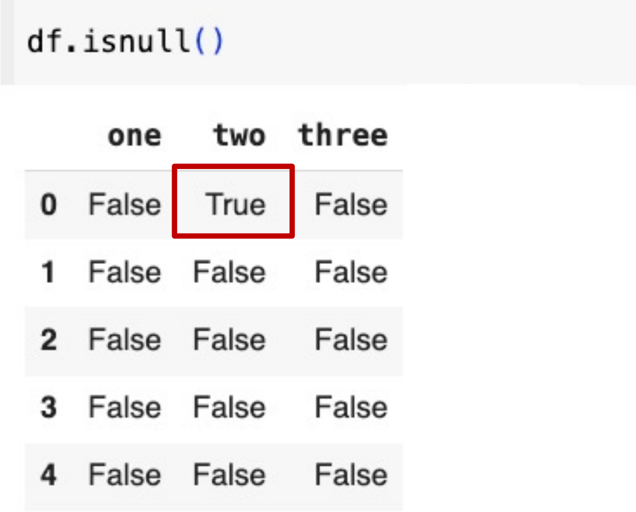
	one	two	three
1	0.173385	1.645548	0.713402

Pandas missing data



```python
df.loc[0,'two'] = np.nan
df
```

	one	two	three
0	0.350867	NaN	1.067663
1	0.501400	0.160161	0.550757
2	-1.787650	1.179029	-1.507067
3	-1.052180	-0.057347	-1.510744
4	-1.213932	-0.746172	1.442200

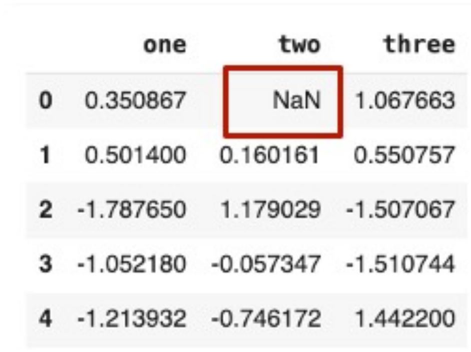


```python
df.isnull()
```

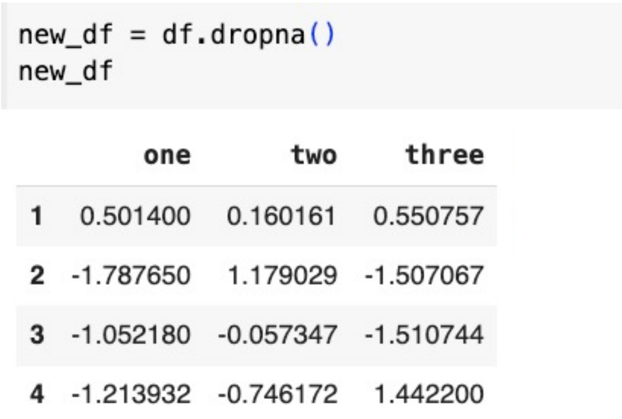
	one	two	three
0	False	True	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	False

Pandas missing data - remove

- One way to deal with empty cells is to remove rows that contain empty cells.
- This is usually OK if data sets can be very big, and removing a few rows will not have a big impact on the result.
- By default, the ***dropna()*** method **returns a new DataFrame, and will not change the original**. If you want to change the original DataFrame, use the *inplace = True* argument.

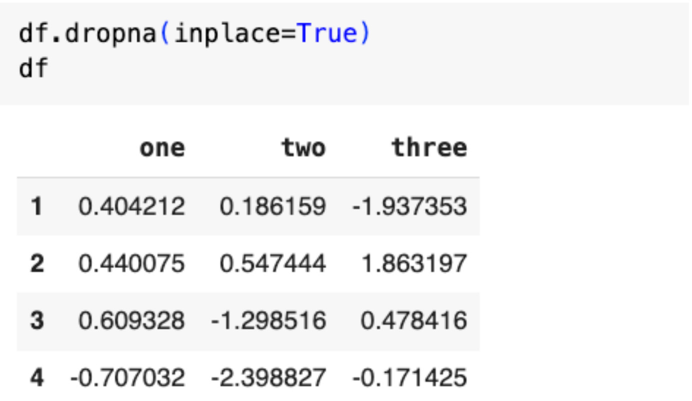


	one	two	three
0	0.350867	NaN	1.067663
1	0.501400	0.160161	0.550757
2	-1.787650	1.179029	-1.507067
3	-1.052180	-0.057347	-1.510744
4	-1.213932	-0.746172	1.442200



```
new_df = df.dropna()
new_df
```

	one	two	three
1	0.501400	0.160161	0.550757
2	-1.787650	1.179029	-1.507067
3	-1.052180	-0.057347	-1.510744
4	-1.213932	-0.746172	1.442200

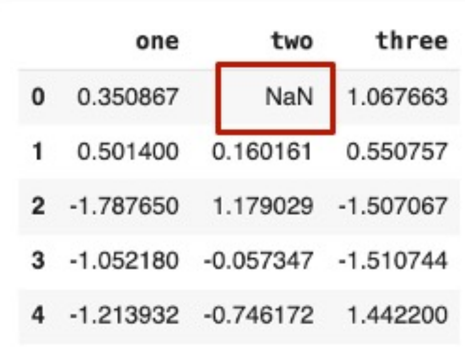


```
df.dropna(inplace=True)
df
```

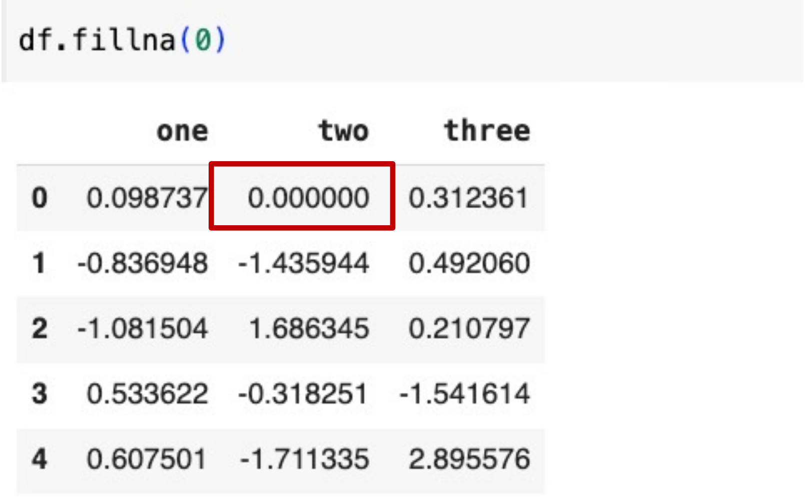
	one	two	three
1	0.404212	0.186159	-1.937353
2	0.440075	0.547444	1.863197
3	0.609328	-1.298516	0.478416
4	-0.707032	-2.398827	-0.171425

Pandas missing data - Replace

- Another way of dealing with empty cells is to insert a new value instead.
- This way you do not have to delete entire rows just because of some empty cells.
- The *fillna()* method allows us to replace empty cells with a value.
  - Replace with zero or another number.



	one	two	three
0	0.350867	NaN	1.067663
1	0.501400	0.160161	0.550757
2	-1.787650	1.179029	-1.507067
3	-1.052180	-0.057347	-1.510744
4	-1.213932	-0.746172	1.442200

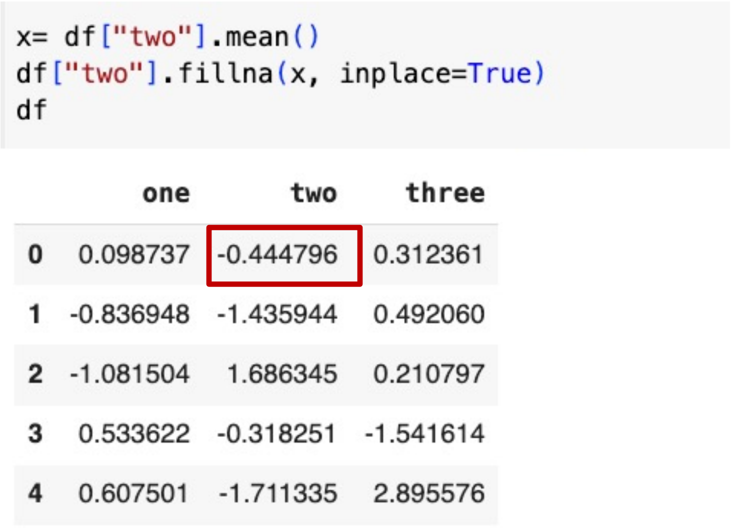


```
df.fillna(0)
```

	one	two	three
0	0.098737	0.000000	0.312361
1	-0.836948	-1.435944	0.492060
2	-1.081504	1.686345	0.210797
3	0.533622	-0.318251	-1.541614
4	0.607501	-1.711335	2.895576

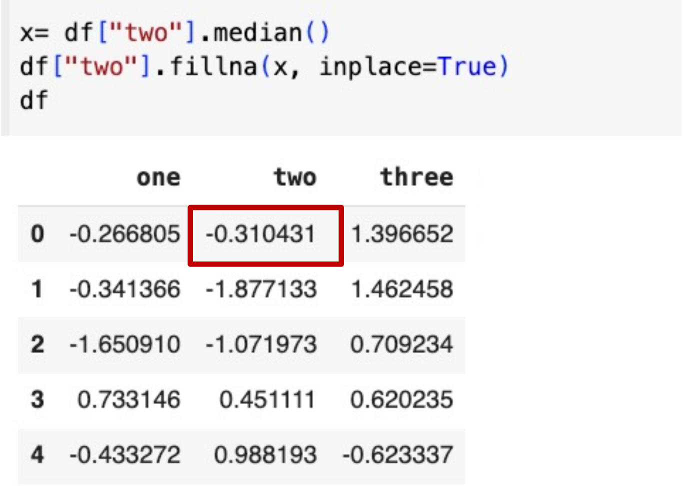
Pandas missing data - Replace

- Another way of dealing with empty cells is to insert a new value instead.
- This way you do not have to delete entire rows just because of some empty cells.
- The *fillna()* method allows us to replace empty cells with a value.
  - Replace with zero or another number.
  - Replace with the mean, median or mode value of the column using the mean() median() and mode() methods.



```
x= df["two"].mean()
df["two"].fillna(x, inplace=True)
df
```

	one	two	three
0	0.098737	-0.444796	0.312361
1	-0.836948	-1.435944	0.492060
2	-1.081504	1.686345	0.210797
3	0.533622	-0.318251	-1.541614
4	0.607501	-1.711335	2.895576

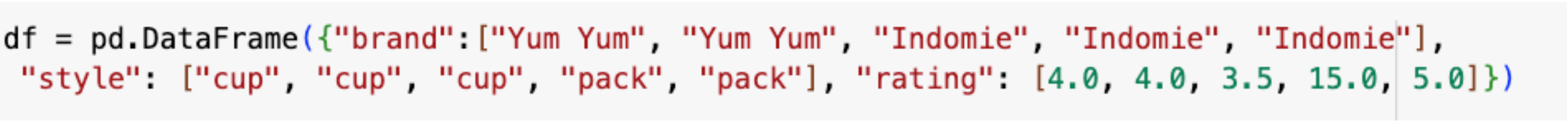


```
x= df["two"].median()
df["two"].fillna(x, inplace=True)
df
```

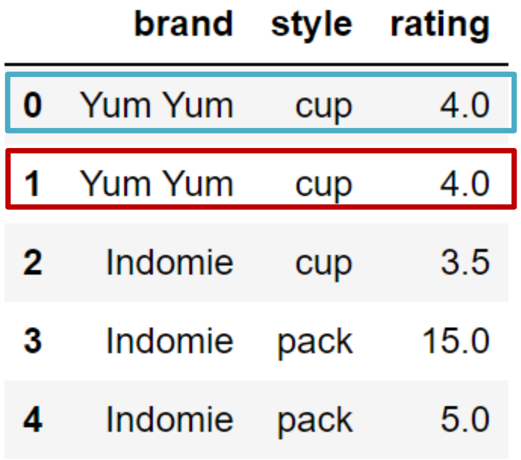
	one	two	three
0	-0.266805	-0.310431	1.396652
1	-0.341366	-1.877133	1.462458
2	-1.650910	-1.071973	0.709234
3	0.733146	0.451111	0.620235
4	-0.433272	0.988193	-0.623337

Pandas duplicate data

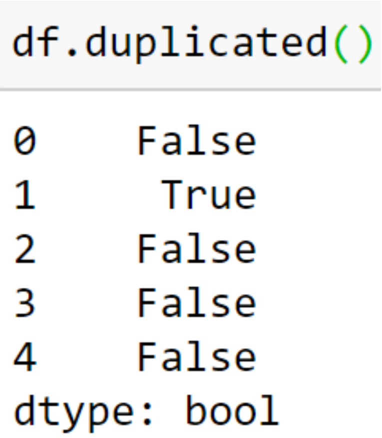
- Duplicate rows are rows that have been registered more than one time.
- To discover duplicates, we can use the duplicated() method, which returns a Boolean values for each row.



```
df = pd.DataFrame({"brand":["Yum Yum", "Yum Yum", "Indomie", "Indomie", "Indomie"],
 "style": ["cup", "cup", "cup", "pack", "pack"], "rating": [4.0, 4.0, 3.5, 15.0, 5.0]})
```



	brand	style	rating
0	Yum Yum	cup	4.0
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	15.0
4	Indomie	pack	5.0

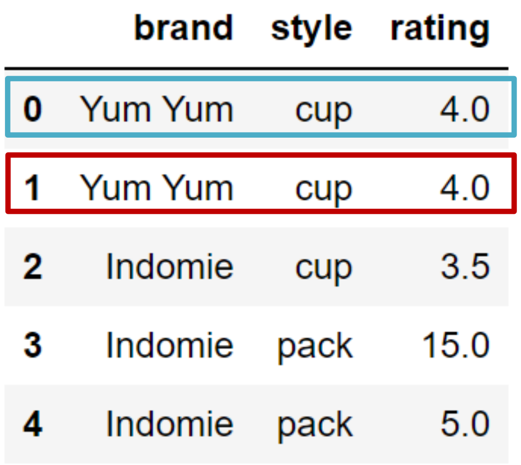


```
df.duplicated()
```

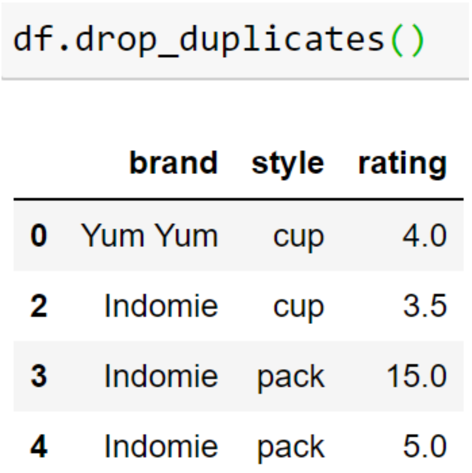
```
0     False
1      True
2     False
3     False
4     False
dtype: bool
```

Pandas duplicate data

- Duplicate rows are rows that have been registered more than one time.
- To discover duplicates, we can use the duplicated() method, which returns a Boolean values for each row.
- To remove duplicates, use the ***drop_duplicates()*** method.
  - Remove all duplicate rows.



	brand	style	rating
0	Yum Yum	cup	4.0
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	15.0
4	Indomie	pack	5.0

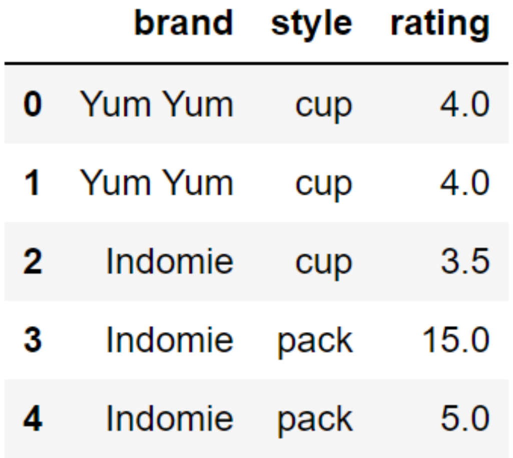


```
df.drop_duplicates()
```

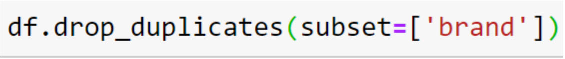
	brand	style	rating
0	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	15.0
4	Indomie	pack	5.0

Pandas duplicate data

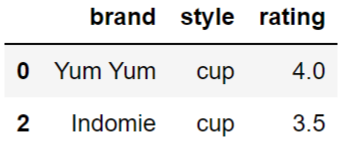
- Duplicate rows are rows that have been registered more than one time.
- To discover duplicates, we can use the duplicated() method, which returns a Boolean values for each row.
- To remove duplicates, use the ***drop_duplicates()*** method.
  - Remove all duplicate rows based on all columns.
  - Remove duplicates on specific column(s), use subset; use keep for keep strategy.



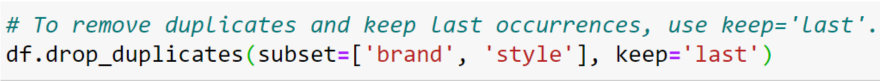
	brand	style	rating
0	Yum Yum	cup	4.0
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
3	Indomie	pack	15.0
4	Indomie	pack	5.0



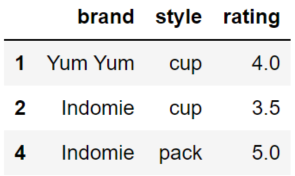
```
df.drop_duplicates(subset=['brand'])
```



	brand	style	rating
0	Yum Yum	cup	4.0
2	Indomie	cup	3.5



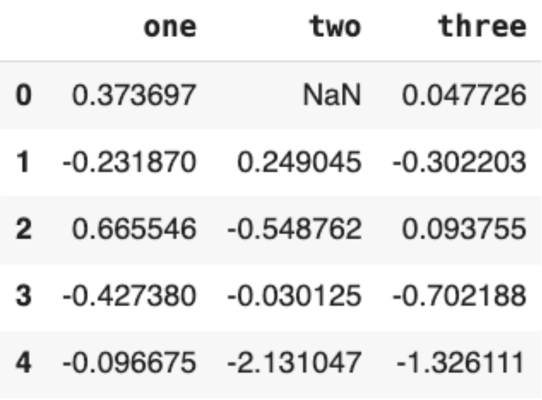
```
# To remove duplicates and keep last occurrences, use keep='last'.
df.drop_duplicates(subset=['brand', 'style'], keep='last')
```



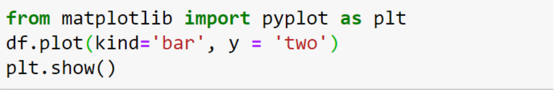
	brand	style	rating
1	Yum Yum	cup	4.0
2	Indomie	cup	3.5
4	Indomie	pack	5.0

Visualization

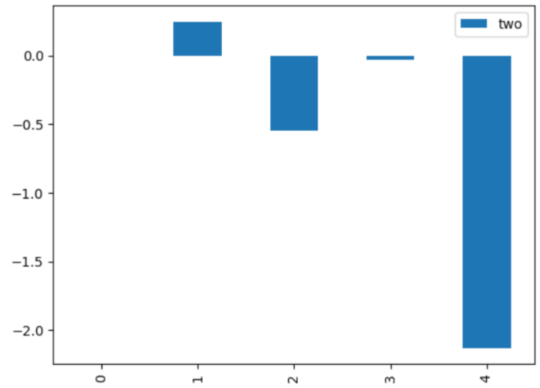
- Using df.plot function:
  - Line plot
  - Bar chart
  - Histogram
  - Scatter plot
  - Pie chart
  - .....

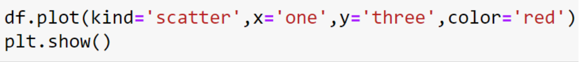


	one	two	three
0	0.373697	NaN	0.047726
1	-0.231870	0.249045	-0.302203
2	0.665546	-0.548762	0.093755
3	-0.427380	-0.030125	-0.702188
4	-0.096675	-2.131047	-1.326111

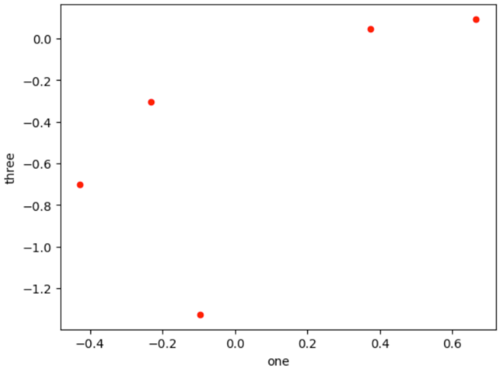


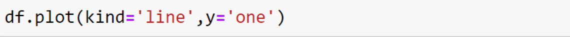
```
from matplotlib import pyplot as plt
df.plot(kind='bar', y = 'two')
plt.show()
```



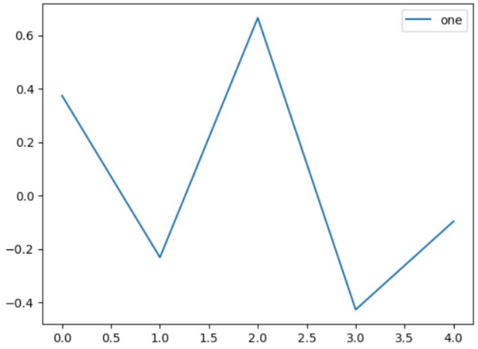


```
df.plot(kind='scatter',x='one',y='three',color='red')
plt.show()
```





```
df.plot(kind='line',y='one')
```

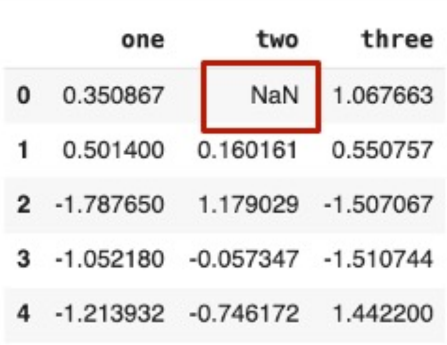


Pandas apply function

- ***Pandas.apply()*** allow the users to pass a function and apply it on every single value of the Pandas series.
- It comes as a huge improvement for the Pandas library as this function helps to segregate data according to the conditions.

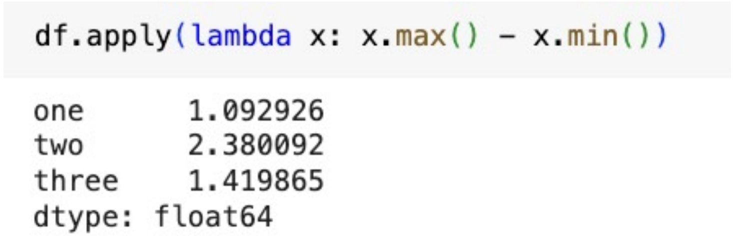
Reference of this slide: https://www.w3schools.com/python/pandas/default.asp

Pandas query data



	one	two	three
0	0.350867	NaN	1.067663
1	0.501400	0.160161	0.550757
2	-1.787650	1.179029	-1.507067
3	-1.052180	-0.057347	-1.510744
4	-1.213932	-0.746172	1.442200

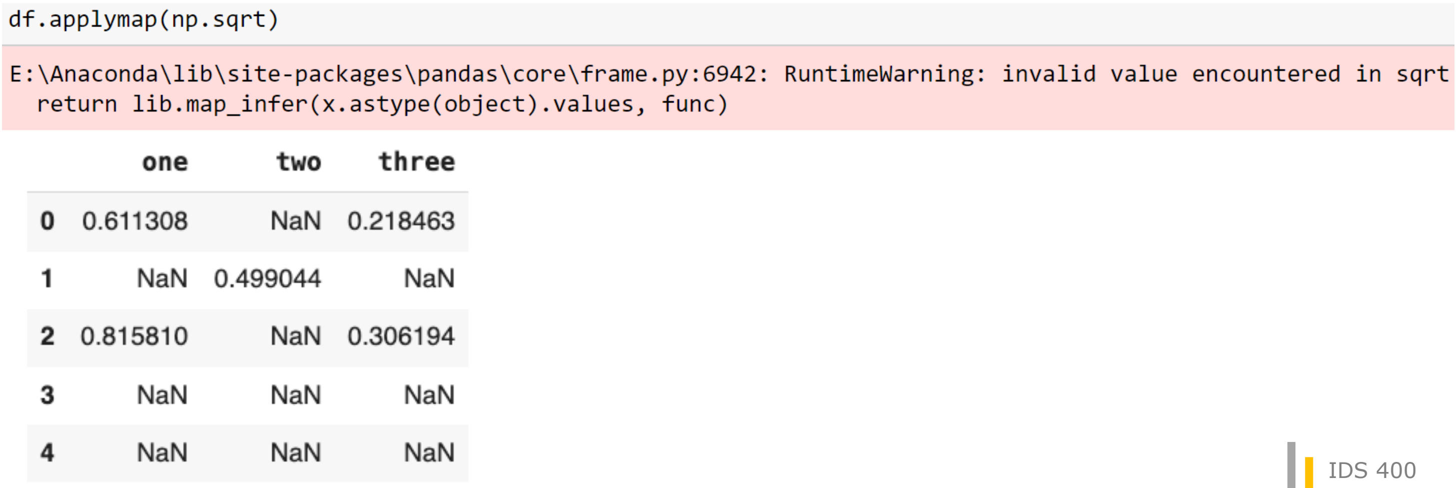
- You can apply any function to the columns in a dataframe.



```
df.apply(lambda x: x.max() - x.min())
```

```
one      1.092926
two      2.380092
three    1.419865
dtype: float64
```

- You can apply any function to the element wise data in a dataframe.



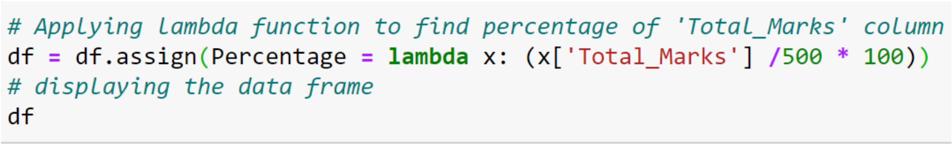
```
df.applymap(np.sqrt)
```

```
E:\Anaconda\lib\site-packages\pandas\core\frame.py:6942: RuntimeWarning: invalid value encountered in sqrt
  return lib.map_infer(x.astype(object).values, func)
```

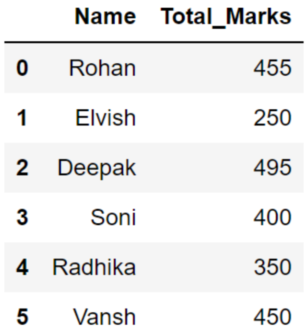
	one	two	three
0	0.611308	NaN	0.218463
1	NaN	0.499044	NaN
2	0.815810	NaN	0.306194
3	NaN	NaN	NaN
4	NaN	NaN	NaN

Pandas lambda function

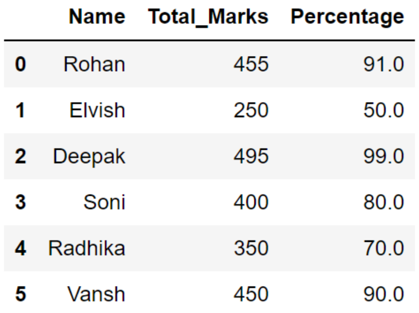
- In Pandas, we can apply a lambda function to both the columns and rows of the Pandas data frame.
    - Applying lambda function to **single column** using **Dataframe.assign()**



```
# Applying lambda function to find percentage of 'Total_Marks' column
df = df.assign(Percentage = lambda x: (x['Total_Marks'] /500 * 100))
# displaying the data frame
df
```



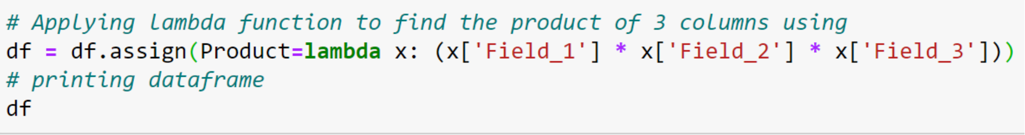
	Name	Total_Marks
0	Rohan	455
1	Elvish	250
2	Deepak	495
3	Soni	400
4	Radhika	350
5	Vansh	450



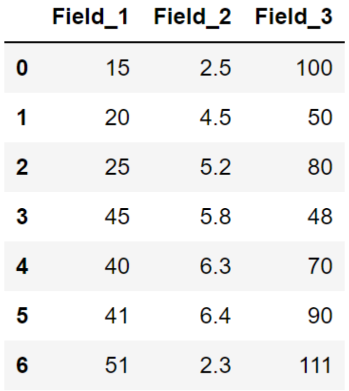
	Name	Total_Marks	Percentage
0	Rohan	455	91.0
1	Elvish	250	50.0
2	Deepak	495	99.0
3	Soni	400	80.0
4	Radhika	350	70.0
5	Vansh	450	90.0

Pandas lambda function

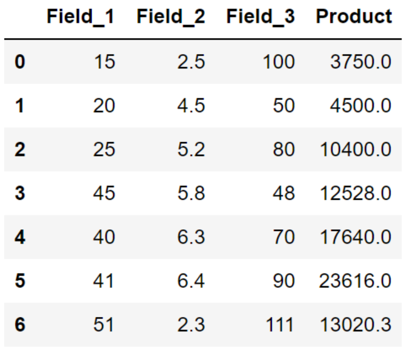
- In Pandas, we can apply a lambda function to both the columns and rows of the Pandas data frame.
  - Applying lambda function to single column using Dataframe.assign()
  - Applying lambda function to **multiple columns** using **Dataframe.assign()**



```
# Applying lambda function to find the product of 3 columns using
df = df.assign(Product=lambda x: (x['Field_1'] * x['Field_2'] * x['Field_3']))
# printing dataframe
df
```



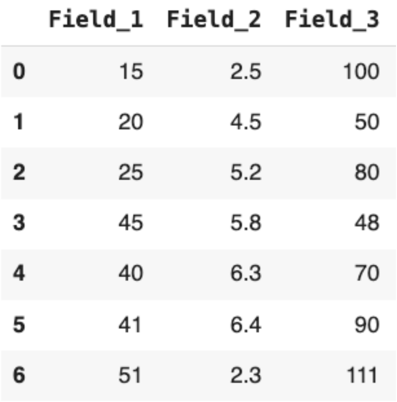
	Field_1	Field_2	Field_3
0	15	2.5	100
1	20	4.5	50
2	25	5.2	80
3	45	5.8	48
4	40	6.3	70
5	41	6.4	90
6	51	2.3	111



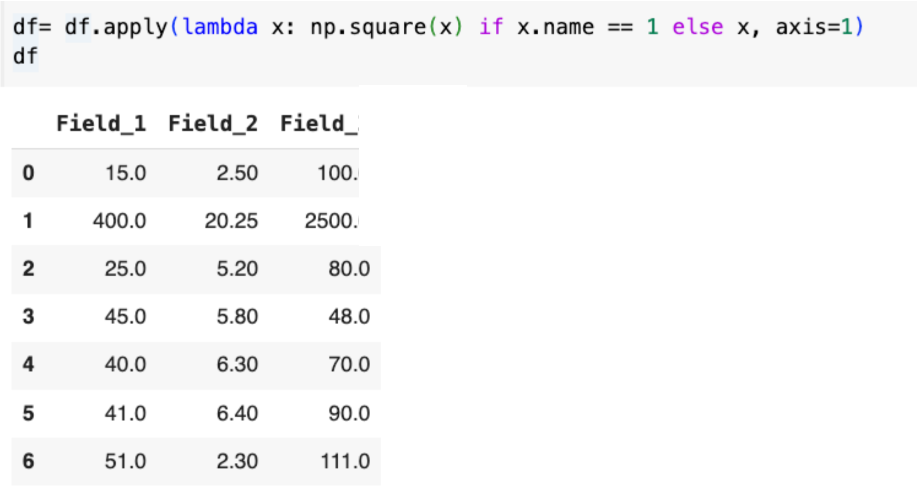
	Field_1	Field_2	Field_3	Product
0	15	2.5	100	3750.0
1	20	4.5	50	4500.0
2	25	5.2	80	10400.0
3	45	5.8	48	12528.0
4	40	6.3	70	17640.0
5	41	6.4	90	23616.0
6	51	2.3	111	13020.3

Pandas lambda function

- In Pandas, we can apply a lambda function to both the columns and rows of the Pandas data frame.
  - Applying lambda function to **single column** using Dataframe.assign()
  - Applying lambda function to **multiple columns** using Dataframe.assign()
  - Applying lambda function to **single row** using **Dataframe.apply()**



	Field_1	Field_2	Field_3
0	15	2.5	100
1	20	4.5	50
2	25	5.2	80
3	45	5.8	48
4	40	6.3	70
5	41	6.4	90
6	51	2.3	111

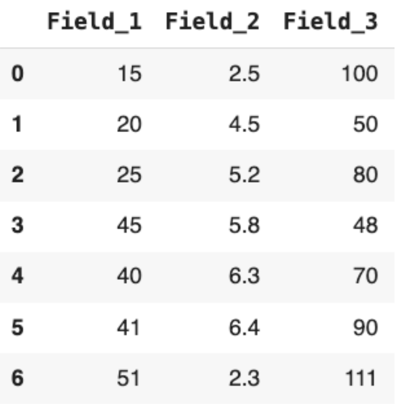


```
df= df.apply(lambda x: np.square(x) if x.name == 1 else x, axis=1)
df
```

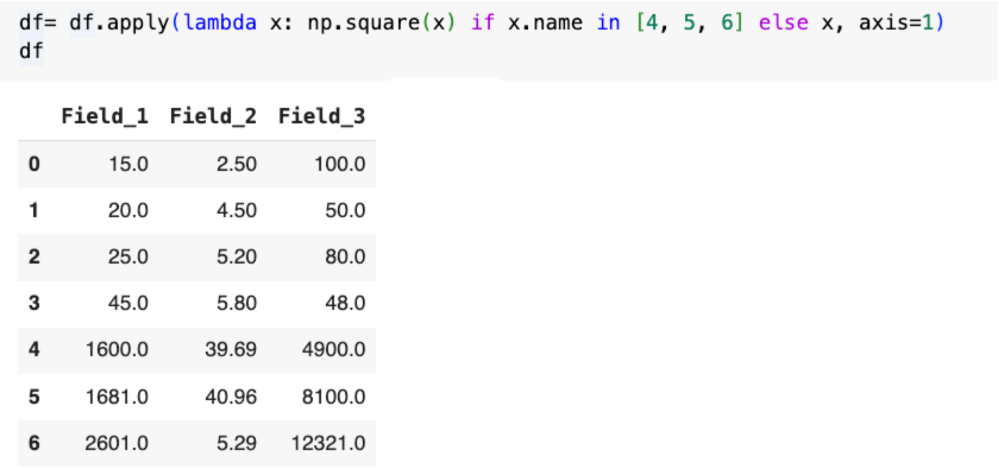
	Field_1	Field_2	Field_
0	15.0	2.50	100.
1	400.0	20.25	2500.
2	25.0	5.20	80.0
3	45.0	5.80	48.0
4	40.0	6.30	70.0
5	41.0	6.40	90.0
6	51.0	2.30	111.0

Pandas lambda function

- In Pandas, we can apply a lambda function to both the columns and rows of the Pandas data frame.
    - Applying lambda function to **single column** using Dataframe.assign()
    - Applying lambda function to **multiple columns** using Dataframe.assign()
    - Applying lambda function to **single row** using Dataframe.apply()
    - Applying lambda function to **multiple rows** using **Dataframe.apply()**



	Field_1	Field_2	Field_3
0	15	2.5	100
1	20	4.5	50
2	25	5.2	80
3	45	5.8	48
4	40	6.3	70
5	41	6.4	90
6	51	2.3	111

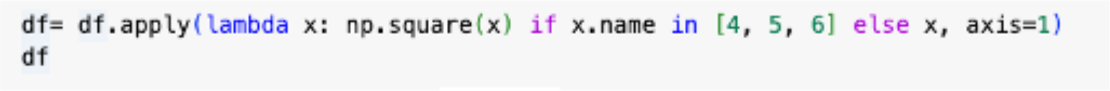


```
df= df.apply(lambda x: np.square(x) if x.name in [4, 5, 6] else x, axis=1)
df
```

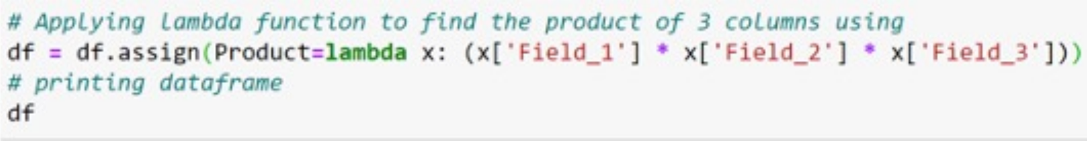
	Field_1	Field_2	Field_3
0	15.0	2.50	100.0
1	20.0	4.50	50.0
2	25.0	5.20	80.0
3	45.0	5.80	48.0
4	1600.0	39.69	4900.0
5	1681.0	40.96	8100.0
6	2601.0	5.29	12321.0

Pandas lambda function

- In Pandas, we can apply a lambda function to both the columns and rows of the Pandas data frame.
  - Applying lambda function to **single column** using Dataframe.assign()
  - Applying lambda function to **multiple columns** using Dataframe.assign()
  - Applying lambda function to **single row** using Dataframe.apply()
  - Applying lambda function to **multiple rows** using Dataframe.apply()
  - Applying the lambda function to **multiple columns** and **rows**



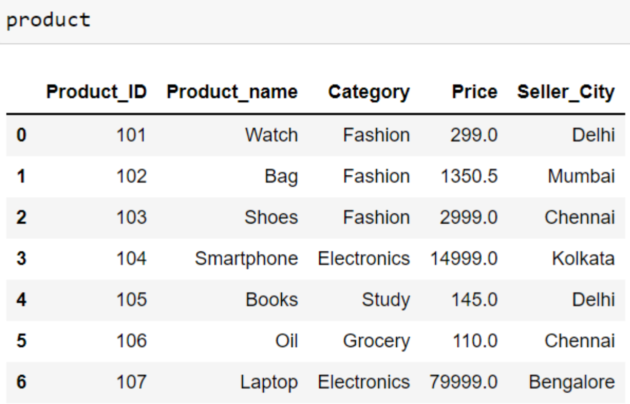
```
df= df.apply(lambda x: np.square(x) if x.name in [4, 5, 6] else x, axis=1)
df
```



```
# Applying lambda function to find the product of 3 columns using
df = df.assign(Product=lambda x: (x['Field_1'] * x['Field_2'] * x['Field_3']))
# printing dataframe
df
```

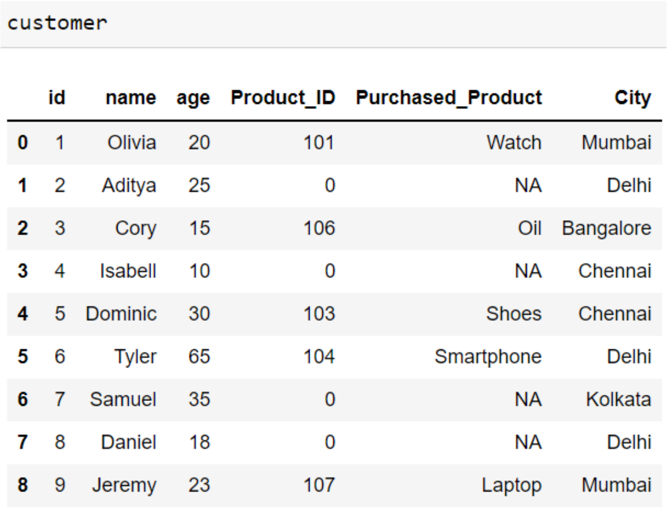
Pandas Dataframe Join

- In Pandas, you can join two dataframes using different methods.
- For example, we are given two tables – one which contains data about **products** and the other that has **customer-level** information.



```
product
```

	Product_ID	Product_name	Category	Price	Seller_City
0	101	Watch	Fashion	299.0	Delhi
1	102	Bag	Fashion	1350.5	Mumbai
2	103	Shoes	Fashion	2999.0	Chennai
3	104	Smartphone	Electronics	14999.0	Kolkata
4	105	Books	Study	145.0	Delhi
5	106	Oil	Grocery	110.0	Chennai
6	107	Laptop	Electronics	79999.0	Bengalore

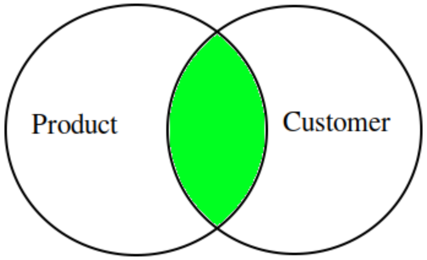


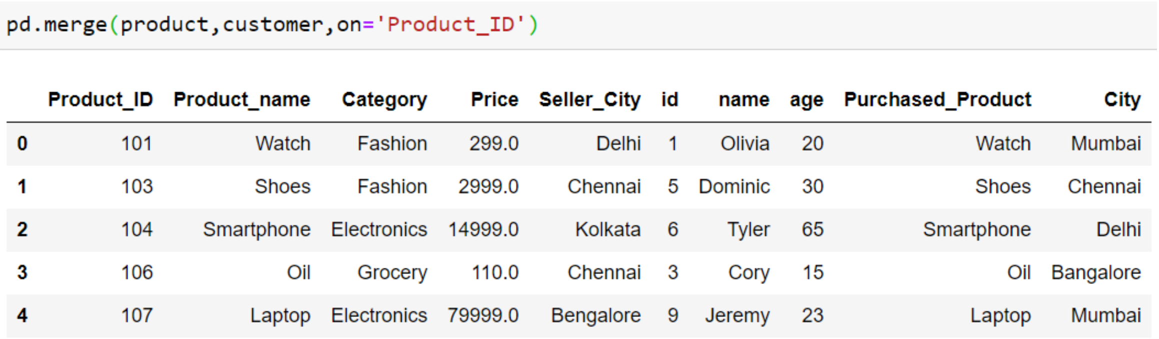
```
customer
```

	id	name	age	Product_ID	Purchased_Product	City
0	1	Olivia	20	101	Watch	Mumbai
1	2	Aditya	25	0	NA	Delhi
2	3	Cory	15	106	Oil	Bangalore
3	4	Isabell	10	0	NA	Chennai
4	5	Dominic	30	103	Shoes	Chennai
5	6	Tyler	65	104	Smartphone	Delhi
6	7	Samuel	35	0	NA	Kolkata
7	8	Daniel	18	0	NA	Delhi
8	9	Jeremy	23	107	Laptop	Mumbai

Pandas Dataframe Join

Inner Join

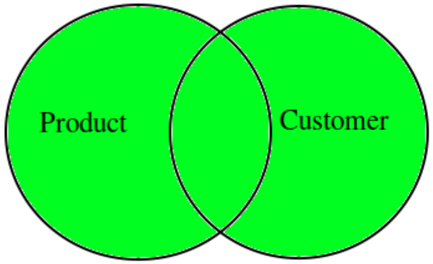


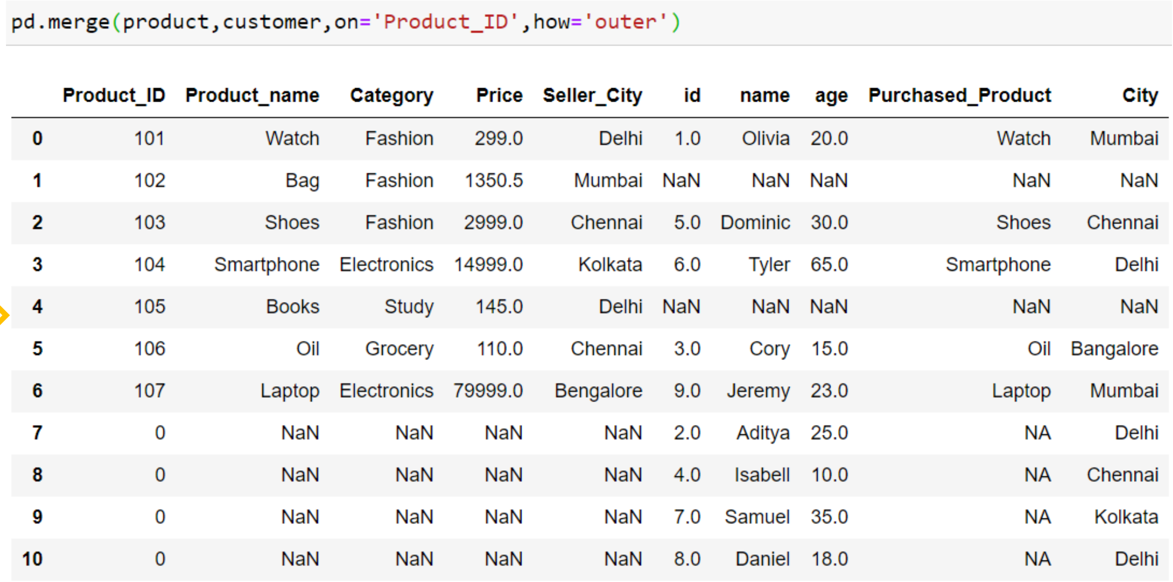


```
pd.merge(product,customer,on='Product_ID')
```

	Product_ID	Product_name	Category	Price	Seller_City	id	name	age	Purchased_Product	City
0	101	Watch	Fashion	299.0	Delhi	1	Olivia	20	Watch	Mumbai
1	103	Shoes	Fashion	2999.0	Chennai	5	Dominic	30	Shoes	Chennai
2	104	Smartphone	Electronics	14999.0	Kolkata	6	Tyler	65	Smartphone	Delhi
3	106	Oil	Grocery	110.0	Chennai	3	Cory	15	Oil	Bangalore
4	107	Laptop	Electronics	79999.0	Bengalore	9	Jeremy	23	Laptop	Mumbai

Full Outer Join

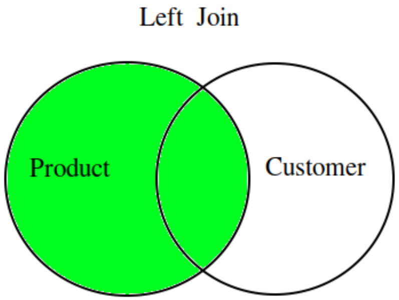


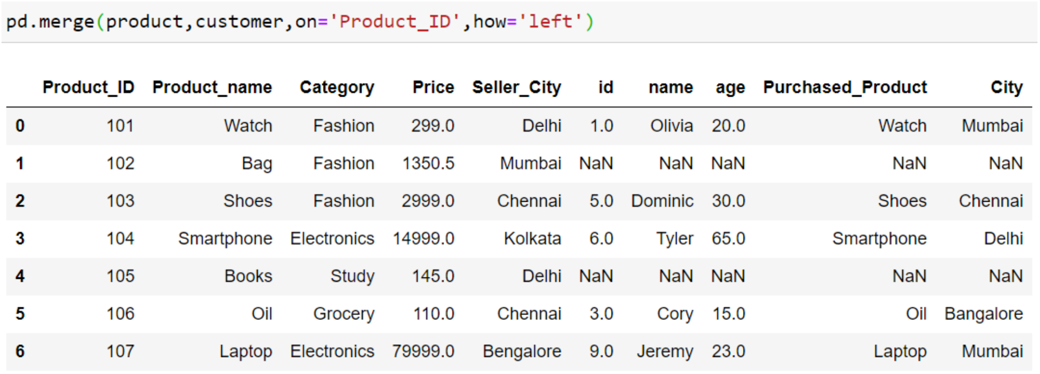


```
pd.merge(product,customer,on='Product_ID',how='outer')
```

	Product_ID	Product_name	Category	Price	Seller_City	id	name	age	Purchased_Product	City
0	101	Watch	Fashion	299.0	Delhi	1.0	Olivia	20.0	Watch	Mumbai
1	102	Bag	Fashion	1350.5	Mumbai	NaN	NaN	NaN	NaN	NaN
2	103	Shoes	Fashion	2999.0	Chennai	5.0	Dominic	30.0	Shoes	Chennai
3	104	Smartphone	Electronics	14999.0	Kolkata	6.0	Tyler	65.0	Smartphone	Delhi
4	105	Books	Study	145.0	Delhi	NaN	NaN	NaN	NaN	NaN
5	106	Oil	Grocery	110.0	Chennai	3.0	Cory	15.0	Oil	Bangalore
6	107	Laptop	Electronics	79999.0	Bengalore	9.0	Jeremy	23.0	Laptop	Mumbai
7	0	NaN	NaN	NaN	NaN	2.0	Aditya	25.0	NA	Delhi
8	0	NaN	NaN	NaN	NaN	4.0	Isabell	10.0	NA	Chennai
9	0	NaN	NaN	NaN	NaN	7.0	Samuel	35.0	NA	Kolkata
10	0	NaN	NaN	NaN	NaN	8.0	Daniel	18.0	NA	Delhi

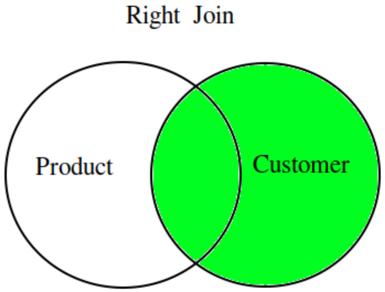
Pandas Dataframe Join

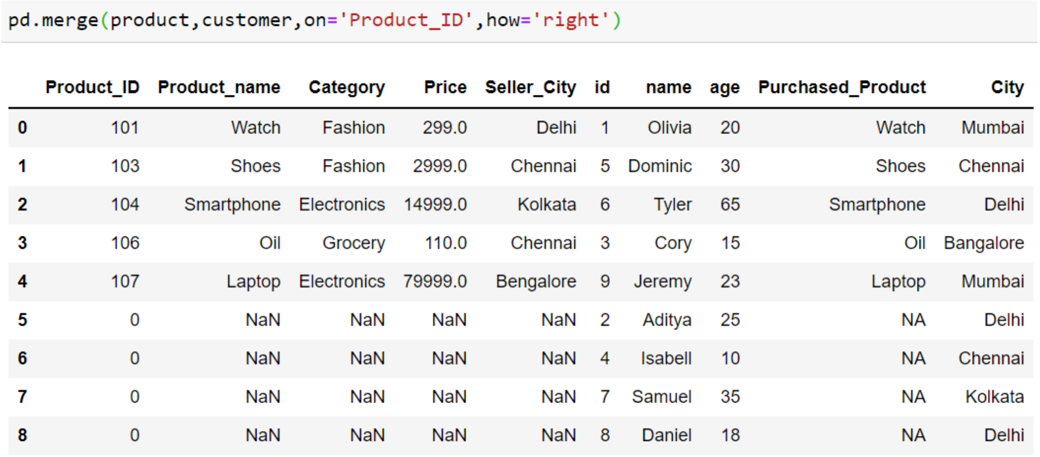




```
pd.merge(product,customer,on='Product_ID',how='left')
```

	Product_ID	Product_name	Category	Price	Seller_City	id	name	age	Purchased_Product	City
0	101	Watch	Fashion	299.0	Delhi	1.0	Olivia	20.0	Watch	Mumbai
1	102	Bag	Fashion	1350.5	Mumbai	NaN	NaN	NaN	NaN	NaN
2	103	Shoes	Fashion	2999.0	Chennai	5.0	Dominic	30.0	Shoes	Chennai
3	104	Smartphone	Electronics	14999.0	Kolkata	6.0	Tyler	65.0	Smartphone	Delhi
4	105	Books	Study	145.0	Delhi	NaN	NaN	NaN	NaN	NaN
5	106	Oil	Grocery	110.0	Chennai	3.0	Cory	15.0	Oil	Bangalore
6	107	Laptop	Electronics	79999.0	Bengalore	9.0	Jeremy	23.0	Laptop	Mumbai





```
pd.merge(product,customer,on='Product_ID',how='right')
```

	Product_ID	Product_name	Category	Price	Seller_City	id	name	age	Purchased_Product	City
0	101	Watch	Fashion	299.0	Delhi	1	Olivia	20	Watch	Mumbai
1	103	Shoes	Fashion	2999.0	Chennai	5	Dominic	30	Shoes	Chennai
2	104	Smartphone	Electronics	14999.0	Kolkata	6	Tyler	65	Smartphone	Delhi
3	106	Oil	Grocery	110.0	Chennai	3	Cory	15	Oil	Bangalore
4	107	Laptop	Electronics	79999.0	Bengalore	9	Jeremy	23	Laptop	Mumbai
5	0	NaN	NaN	NaN	NaN	2	Aditya	25	NA	Delhi
6	0	NaN	NaN	NaN	NaN	4	Isabell	10	NA	Chennai
7	0	NaN	NaN	NaN	NaN	7	Samuel	35	NA	Kolkata
8	0	NaN	NaN	NaN	NaN	8	Daniel	18	NA	Delhi

Lab *Pandas*