Lecture 02a:

Data Sampling and Pandas

Markus Hohle

University California, Berkeley

Data Science for Scientific Computing

MSSE 277A, 3 Units

Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING

Outline

Part I

Part I

Outline

Reading/Writing Files

- Pandas and Data Frames
- Faster Alternatives
- More about `.txt`

Visualization

- Matplotlib
- Seaborn
- `plt, ax, fig`

Outline

Part II

Data Sampling: Larger Than Memory Files

Outline

Standard Pandas

Line By Line

Chunks With Dask

PySpark

Outline

Reading/Writing Files

- **Pandas and Data Frames**
- Faster Alternatives
- More about `.txt`

Visualization

- Matplotlib
- Seaborn
- `plt, ax, fig`

Data Sampling and Pandas:

Pandas & Data Frames

most common file formats

plain text:	`.dat` `.txt` `.fa`
tables:	`.csv` `.xls` `.xlsx`
python:	`.py` `.npy` `.pkl`

```
import pandas as pd
```

```
pd.re
```

- read_clipboard
- read_csv
- read_excel
- read_feather
- read_fwf
- read_gbq
- read_hdf

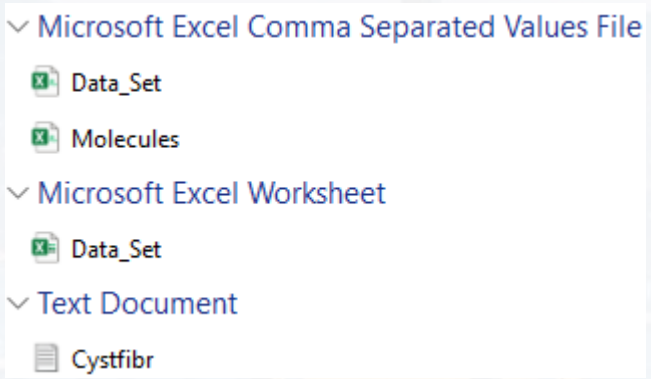
Berkeley UNIVERSITY OF CALIFORNIA Data Sampling and Pandas:

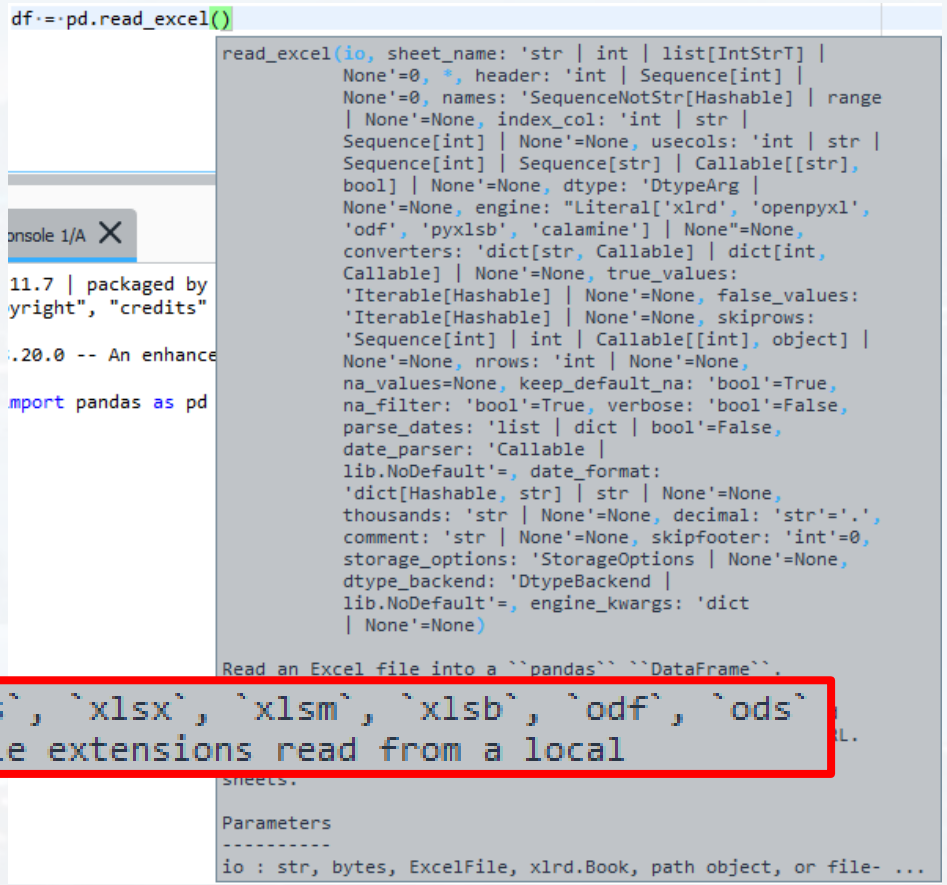
Pandas & Data Frames

most common file formats

plain text:	`.dat .txt .fa`
tables:	`.csv .xls .xlsx`
python:	`.py .npy .pkl`

```
import pandas as pd
```





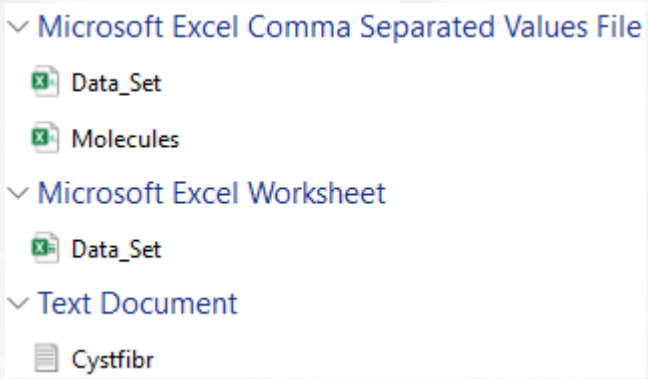
Data Sampling and Pandas:

Pandas & Data Frames

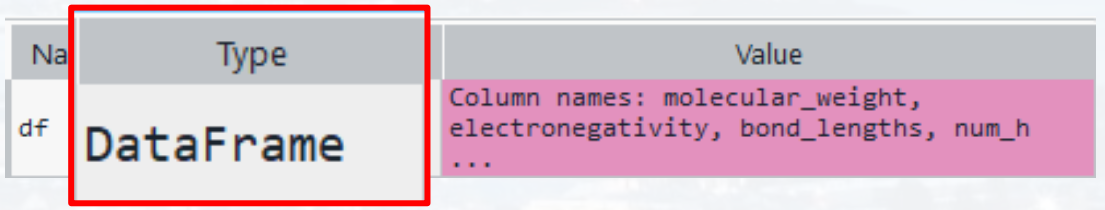
most common file formats

plain text:	`.dat` `.txt` `.fa`
tables:	`.csv` `.xls` `.xlsx`
python:	`.py` `.npy` `.pkl`

```
import pandas as pd
```




```
df = pd.read_csv('Molecules.csv')
```



Data Sampling and Pandas:

Pandas & Data Frames

```
import pandas as pd

df = pd.read_csv('Molecules.csv')
```

Nam	Type	Size	Value
df	DataFrame	(200, 6)	Column names: molecular_weight, electronegativity, bond_lengths, num_h ...

df - DataFrame

Index	molecular_weight	electronegativity	bond_lengths	num_hydrogen_bonds	logP	label
0	341.704	2.65585	3.09407	2	9.11147	Toxic
1	335.951	3.22262	2.89039	7	8.92848	Toxic
2	235.203	2.44115	2.48203	1	6.49731	Toxic
3	246.505	2.76656	2.71547	7	7.45089	Toxic
4	437.939	3.4801	3.59569	3	10.9156	Toxic
5	336.453	2.81474	3.11	9	8.55696	Toxic
6	372.542	3.17969	3.3866	8	9.48685	Toxic
7	349.19	3.1814	3.19359	7	9.10357	Toxic
8	399.353	3.02359	3.50278	4	9.8421	Toxic

Data Sampling and Pandas:

Pandas & Data Frames

```
import pandas as pd

df = pd.read_csv('Molecules.csv')

df.head()
```

Nam	Type	Size	Value
df	DataFrame	(200, 6)	Column names: molecular_weight, electronegativity, bond_lengths, num_h ...

```
In [3]: df.head()
Out[3]:
   molecular_weight  electronegativity  ...       logP  label
0        341.704142           2.655846  ...   9.111473  Toxic
1        335.950798           3.222621  ...   8.928483  Toxic
2        235.203185           2.441153  ...   6.497307  Toxic
3        246.504930           2.766560  ...   7.450888  Toxic
4        437.938926           3.480105  ...  10.915629  Toxic
```

fully equivalent for `excel` files:

```
df = pd.read_excel('Molecules.xlsx')
```

Pandas & Data Frames

a data frame has numerous functions

`df.head()` shows header

```
   molecular_weight  electronegativity  ...      logP  label
0        341.704142           2.655846  ...  9.111473  Toxic
1        335.950798           3.222621  ...  8.928483  Toxic
2        235.203185           2.441153  ...  6.497307  Toxic
```

`df.index` returns rows

```
df.index
RangeIndex(start=0, stop=200, step=1)
```

`df.columns` returns columns

```
Index(['molecular_weight', 'electronegativity', 'bond_lengths',
       'num_hydrogen_bonds', 'logP', 'label'],
      dtype='object')
```

Data Sampling and Pandas:

Pandas & Data Frames

a data frame has numerous functions

`df.columns` returns columns

```
Index(['molecular_weight', 'electronegativity', 'bond_lengths',
       'num_hydrogen_bonds', 'logP', 'label'],
      dtype='object')
```

`df.corr()` returns Pearsons' correlation coefficient

columns `0` to `4` contain `float`/`int`

```
Corr = df[df.columns[:-1]].corr()
```

Index	molecular_weight	electronegativity	bond_lengths	um_hydrogen_bond	logP
molecular_weight	1	0.0280505	0.953066	0.0157675	0.969772
electronegativity	0.0280505	1	0.0343733	-0.0526109	0.00634745
bond_lengths	0.953066	0.0343733	1	0.0258849	0.926063
num_hydrogen_bonds	0.0157675	-0.0526109	0.0258849	1	0.0104456
logP	0.969772	0.00634745	0.926063	0.0104456	1

Pandas & Data Frames

a data frame has numerous functions

```
df[['logP', 'label']]
```

returns **data frame** of **selected columns**

	logP	label
0	9.111473	Toxic
1	8.928483	Toxic
2	6.497307	Toxic
3	7.450888	Toxic
4	10.915629	Toxic
..	...	...
195	8.794466	Non-Toxic
196	9.651463	Toxic
197	7.651613	Non-Toxic
198	9.060061	Toxic

```
df.loc[[1, 5]]
```

returns **data frame** of **selected rows**

	molecular_weight	electronegativity	...	logP	label
1	335.950798	3.222621	...	8.928483	Toxic
5	336.453422	2.814735	...	8.556958	Toxic

Pandas & Data Frames

a data frame has numerous functions

Try out the following commands!

`df.iloc[4:6, 5:9]`	slicing **data frame** using `iloc`
`df.iloc[4,5] = 999`	manipulating individual entries
`df.insert(2, 'New', df.iloc[:,1])`	inserting another column called *New*
`df.rename(index = {1: 'bbb'})`	changing name of row 1
`df.rename(columns = {'label': 'Toxic or Not'})`	changing column name

Pandas & Data Frames

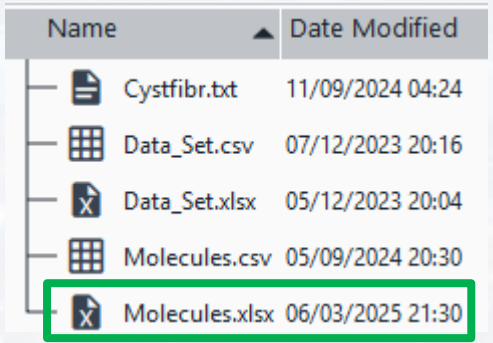
```
import pandas as pd

df = pd.read_csv('Molecules.csv')
```

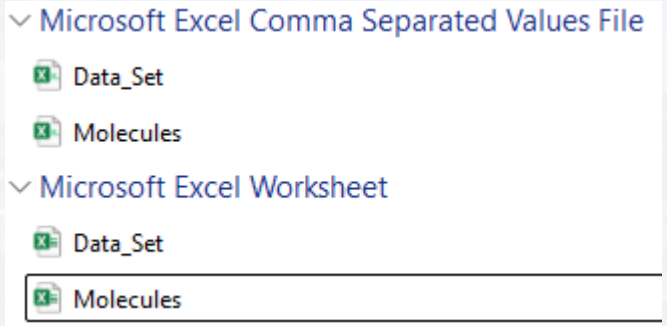
finally, saving the data frame to an excel file

```
df.t
```

```
tail
take
to_clipboard
to_csv
to_dict
to_excel
to_feather
```

```
df.to_excel('Molecules.xlsx')
```





Pandas & Data Frames

pandas can also read **text files**:


```
cf = pd.read_csv('cystfibr.txt')
```

cf - DataFrame

Index	age sex height weight bmp fev1 rv frc tlc pemax
0	7 0 109 13.1 68 32 258 183 137 95
1	7 1 112 12.9 65 19 449 245 134 85
2	8 0 124 14.1 64 22 441 268 147 100

```
cf = pd.read_csv('cystfibr.txt', sep = '\s+')
```

cf - DataFrame

Index	age	sex	height	weight	bmp	fev1	rv	frc	tlc	pemax
0	7	0	109	13.1	68	32	258	183	137	95
1	7	1	112	12.9	65	19	449	245	134	85
2	8	0	124	14.1	64	22	441	268	147	100
3	8	1	125	16.2	67	41	234	146	124	85

Data Sampling and Pandas:

Pandas & Data Frames

note: data frames are

- very **user friendly** and
- have many **useful functions**

but:

- they are **slow** when working with bigger data sets
- and for any computational operation

therefore:

- **don't use data frames** for the **data analysis** part!
- use **numpy** arrays instead!
- pandas data frames are more like a **nice summary for the user**

Outline

Reading/Writing Files

- Pandas and Data Frames
- **Faster Alternatives**
- More about `.txt`

Visualization

- Matplotlib
- Seaborn
- `plt, ax, fig`

Data Sampling and Pandas:

Faster Alternatives

pandas is the standard library, but it is slow

reading a 130MB **excel file**:

```
import pandas as pd
import time


t1 = time.monotonic()

df = pd.read_excel('Data_Set.xlsx')

t2 = time.monotonic()
dt = t2 - t1


print("Total runtime: " + str(dt) + ' seconds')
```

83 seconds

Data Sampling and Pandas:

Faster Alternatives

pandas is the standard library, but it is slow

reading a 180MB **csv file** with the **same content**:

```
import pandas as pd
import time


t1 = time.monotonic()

df = pd.read_csv('Data_Set.csv')

t2 = time.monotonic()
dt = t2 - t1


print("Total runtime: " + str(dt) + ' seconds')
```

1.2 seconds!

Faster Alternatives

pandas:

excel file:	83.0 sec
csv file:	1.2 sec

reading a 180MB **txt file** with the **same content**:

```
import pandas as pd
import time


t1 = time.monotonic()

df = pd.read_csv('Data_Set.txt')

t2 = time.monotonic()
dt = t2 - t1


print("Total runtime: " + str(dt) + ' seconds')
```

1.5 seconds!

Data Sampling and Pandas:

Faster Alternatives

pandas:

excel file:	83.0 sec
csv file:	1.2 sec
txt file:	1.5 sec

faster than pandas, but fewer functions

- `dask`

- `polars`

- `fireducks`

Data Sampling and Pandas:

Faster Alternatives

pandas:

excel file:	83.0 sec
csv file:	1.2 sec
txt file:	1.5 sec

```
pip install dask
pip install polars
pip install xlsx2csv  #for excel API
pip install fastexcel
```


```
import dask.dataframe as dd
```

Data Sampling and Pandas:

Faster Alternatives

pandas:	
excel file:	83.0 sec
csv file:	1.2 sec
txt file:	1.5 sec

```
import dask.dataframe as dd


t1 = time.monotonic()

df = dd.read_csv('Data_Set.csv')

t2 = time.monotonic()
dt = t2 - t1


print("Total runtime: " + str(dt) + ' seconds')


df = pd.DataFrame(df)
```

0.016 seconds!

However, we might need to transform the output

Data Sampling and Pandas:

Faster Alternatives

pandas:	
excel file:	83.0 sec
csv file:	1.2 sec
txt file:	1.5 sec

```
import dask.dataframe as dd


t1 = time.monotonic()

df = pd.DataFrame(dd.read_csv('Data_Set.csv'))

t2 = time.monotonic()
dt = t2 - t1


print("Total runtime: " + str(dt) + ' seconds')
```

1.7 seconds!

As of Dec 2025, dask **doesn't have an excel API**

Faster Alternatives

pandas:			
	excel file:	**83.0**	**sec**
	csv file:	**1.2**	**sec**
	txt file:	**1.5**	**sec**
dask	**csv file:**	**0.016**	**sec**
	to df	**1.7**	**sec**

```
import polars as pl

t1 = time.monotonic()

df = pl.read_excel('Data_Set.xlsx')

t2 = time.monotonic()
dt = t2 - t1

print("Total runtime: " + str(dt) + ' seconds')

df = pd.DataFrame(df)
```

8.2 seconds!

However, we might need to transform the output

Faster Alternatives

pandas:			
	excel file:	**83.0**	**sec**
	csv file:	**1.2**	**sec**
	txt file:	**1.5**	**sec**
dask	**csv file:**	**0.016**	**sec**
	to df	**1.7**	**sec**

```
import polars as pl


t1 = time.monotonic()

df = pd.DataFrame(pl.read_excel('Data_Set.xlsx'))

t2 = time.monotonic()
dt = t2 - t1


print("Total runtime: " + str(dt) + ' seconds')
```

10.2 seconds!

Data Sampling and Pandas:

Faster Alternatives

pandas:			
	excel file:	**83.0**	**sec**
	csv file:	**1.2**	**sec**
	txt file:	**1.5**	**sec**
dask	**csv file:**	**0.016**	**sec**
	to df	**1.7**	**sec**
polars	**excel file**	**8.2**	**sec**
	to df	**10.2**	**sec**

```
import polars as pl

t1 = time.monotonic()

df = pl.read_csv('Data_Set.csv')

t2 = time.monotonic()
dt = t2 - t1

print("Total runtime: " + str(dt) + ' seconds')
```

0.27 seconds!

Faster Alternatives

pandas:			
	excel file:	**83.0**	**sec**
	csv file:	**1.2**	**sec**
	txt file:	**1.5**	**sec**
dask	**csv file:**	**0.016**	**sec**
	to df	**1.7**	**sec**
polars	**excel file**	**8.2**	**sec**
	to df	**10.2**	**sec**

```
import polars as pl

t1 = time.monotonic()

df = pd.DataFrame(pl.read_csv('Data_Set.csv'))

t2 = time.monotonic()
dt = t2 - t1

print("Total runtime: " + str(dt) + ' seconds')
```

0.20 seconds!

Data Sampling and Pandas:

Faster Alternatives

	pandas	dask	polars
`.xlsx`	83 sec	na	8.2 sec 10.2 sec
`.csv`	1.2 sec	0.016 sec 1.6 sec	0.27 sec 0.20 sec
`.txt`	1.5 sec	0.016 sec 0.92 sec	0.25 sec 0.23 sec

Data Sampling and Pandas:

Faster Alternatives

	pandas	dask	polars
`.xlsx`	**83 sec**	na	8.2 sec **10.2 sec**
`.csv`	1.2 sec	0.016 sec **1.6 sec**	0.27 sec **0.20 sec**
`.txt`	**1.5 sec**	0.016 sec 0.92 sec	0.25 sec **0.23 sec**

check out `Benchmark_Pandas_Dask_Polars.py`

Outline

Reading/Writing Files

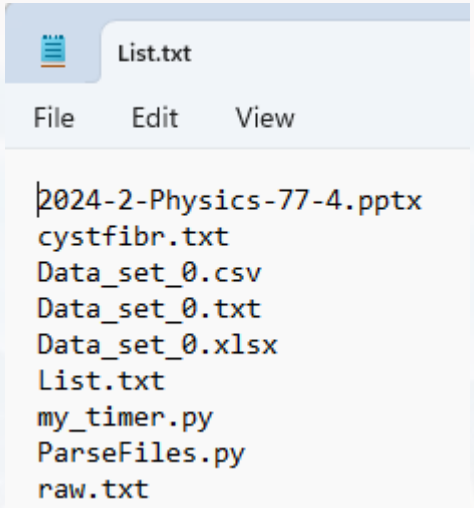
- Pandas and Data Frames
- Faster Alternatives
- **More about `.txt`**

Visualization

- Matplotlib
- Seaborn
- `plt, ax, fig`

More about `.txt`

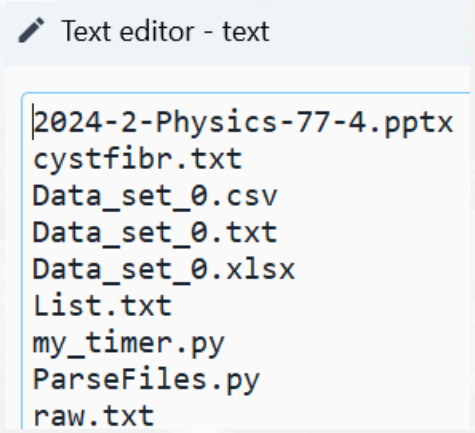
sometimes txt files don't come in a nice format



→ using open

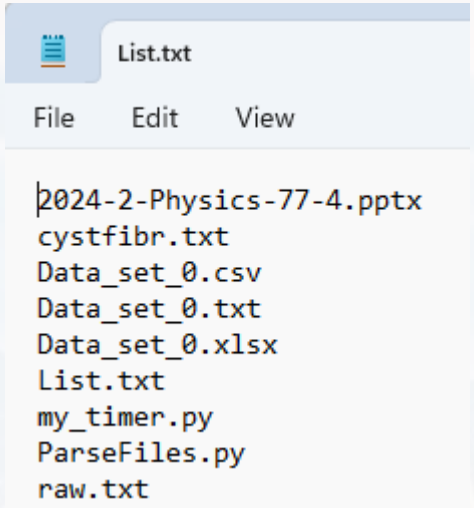
```
with open('List.txt', errors = "ignore") as f:
        text = f.read()
```

reads file line by line and stops automatically when has reached the end

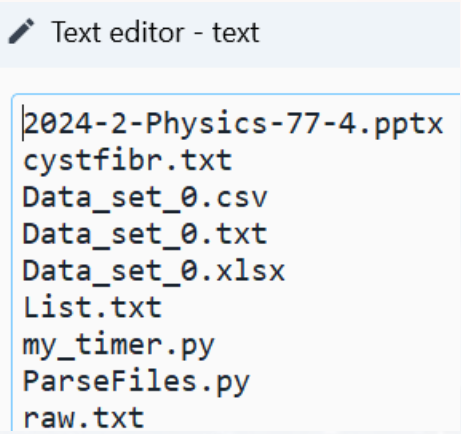


More about `.txt`

sometimes txt files don't come in a nice format




```
with open('List.txt', errors = "ignore") as f:
        text = f.read()
```




```
T = list(text.split())
```

Index	Type	Size	
0	str	24	2024-2-Physics-77-4.pptx
1	str	12	cystfibr.txt
2	str	14	Data_set_0.csv
3	str	14	Data_set_0.txt
4	str	15	Data_set_0.xlsx
5	str	8	List.txt
6	str	11	my_timer.py

More about `.txt`

syntax of open

```
with open('my_file.txt', flags, errors = "ignore") as f:
    text = f.read()
```

finishes once it has reached the end and closes *f*

flags/modes:

`'r'`	opens file for reading only.
`'w'`	opens file for writing. If the file exists, it overwrites it, otherwise, it creates a new file.
`'a'`	opens file for appending only. If the file doesn't exist, it creates the file.
`'x'`	creates new file. If the file exists, it fails.
`'+'`	opens file for updating
`'t'`	opens file in text mode (default)
`'b'`	opens file in binary mode

flags can be combined like, eg `'wb'`

`with` opens a loop for reading the file line by line

More about `.txt`

syntax of open

```
with open('List.txt', 'r') as read_f:
    with open('List_copy.txt', 'w') as write_f:
        for r in read_f:
            write_f.write(r)
```

opens file line by line

opens **new** file line by line for **writing**

writes each line to **new** file

```
In [5]: print(r)
raw.txt
```

```
In [6]: print(type(r))
<class 'str'>
```

Outline

Reading/Writing Files

- Pandas and Data Frames
- Faster Alternatives
- More about `.txt`

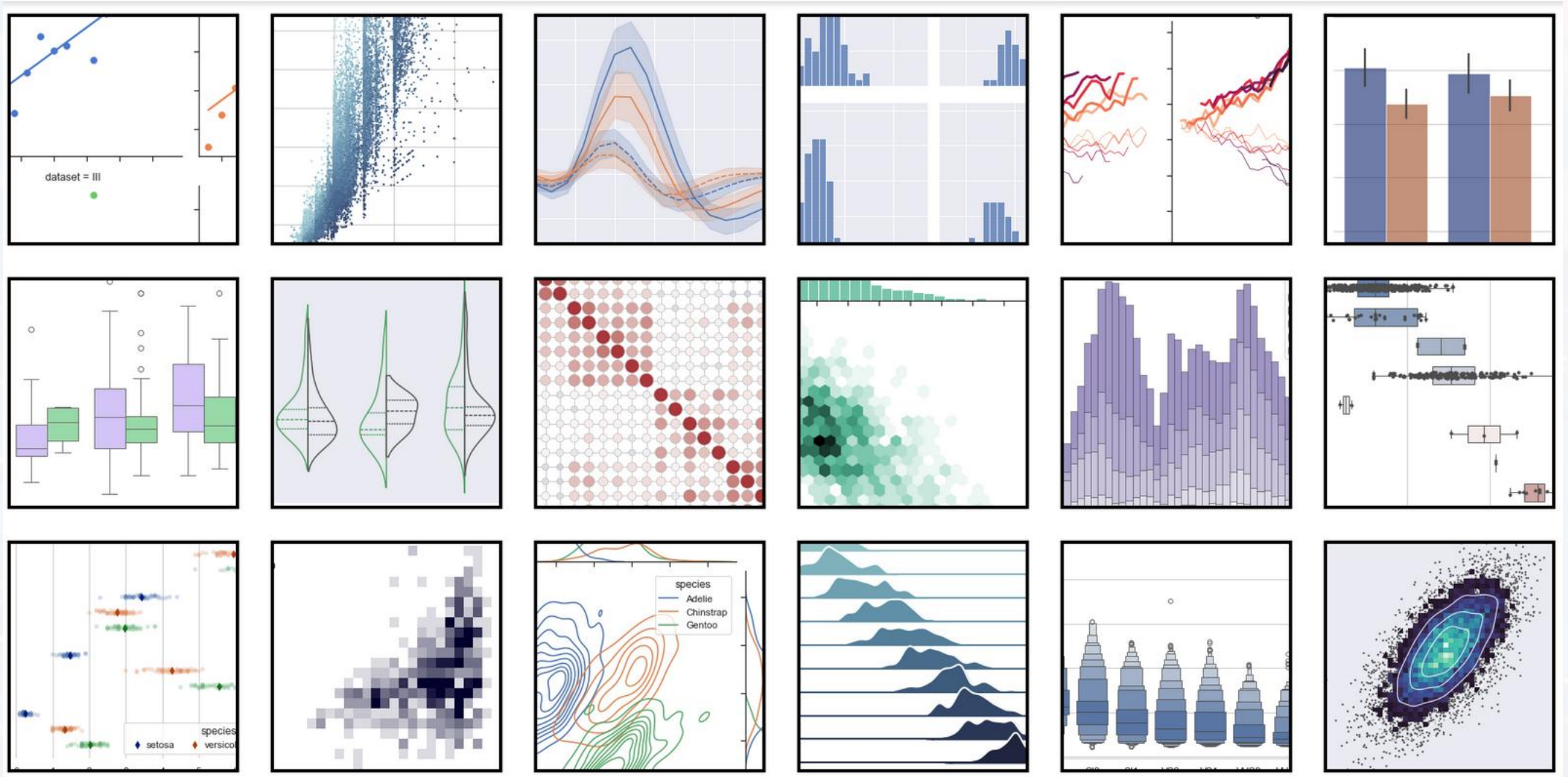
Visualization

- **Matplotlib**
- Seaborn
- `plt, ax, fig`

matplotlib

Python Graph Gallery

https://seaborn.pydata.org/examples/index.html

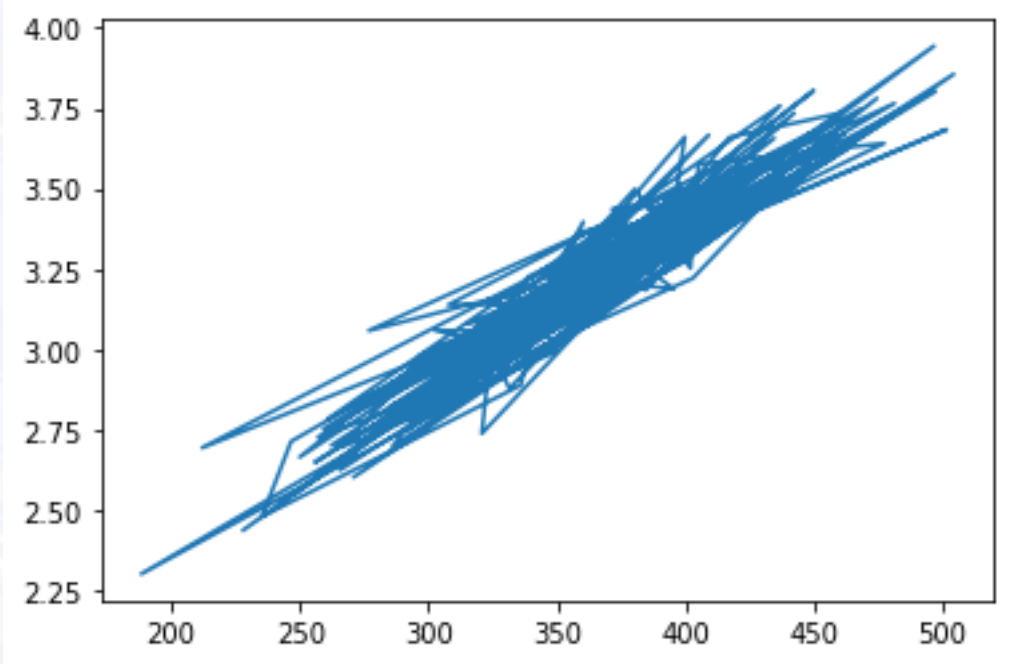


Data Sampling and Pandas:

matplotlib

basic plots
arguments
settings

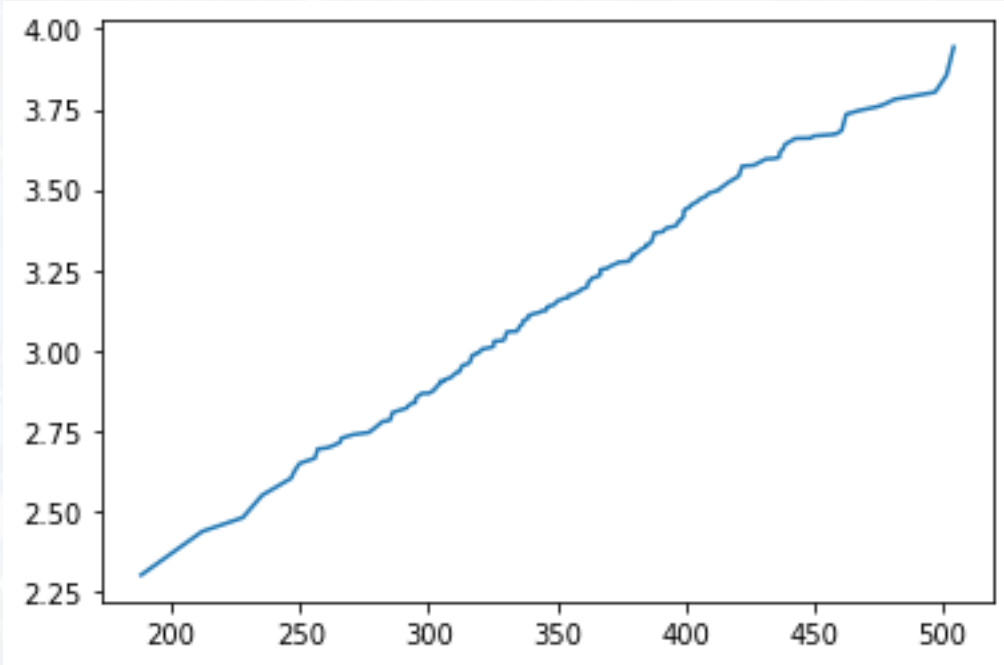
```
import pandas as pd
import matplotlib.pyplot as plt

Data = pd.read_csv('Molecules.csv')

x    = Data.molecular_weight
y    = Data.bond_lengths


plt.plot(x,y)
```



matplotlib

basic plots
arguments
settings

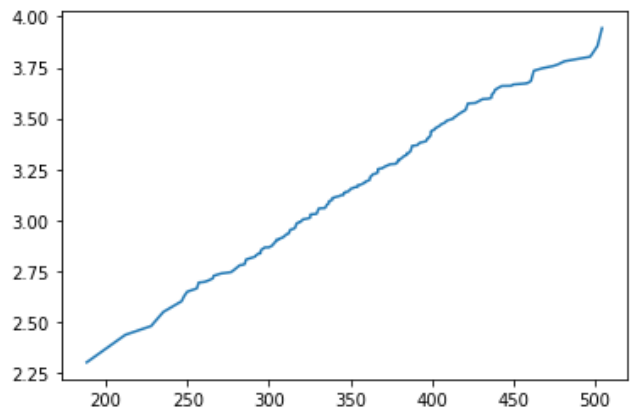
```
import pandas as pd
import matplotlib.pyplot as plt

Data = pd.read_csv('Molecules.csv')

x    = Data.molecular_weight
y    = Data.bond_lengths


plt.plot(sorted(x), sorted(y))
```



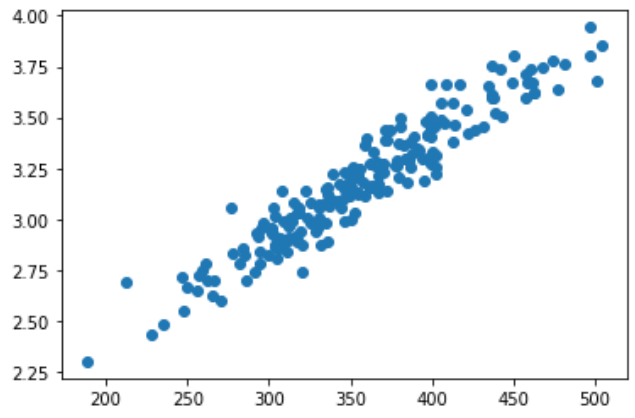
matplotlib

basic plots
arguments
settings

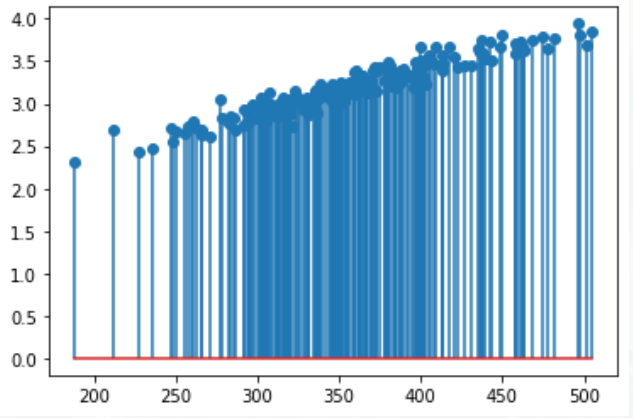
```
plt.plot(sorted(x),
         sorted(y))
```



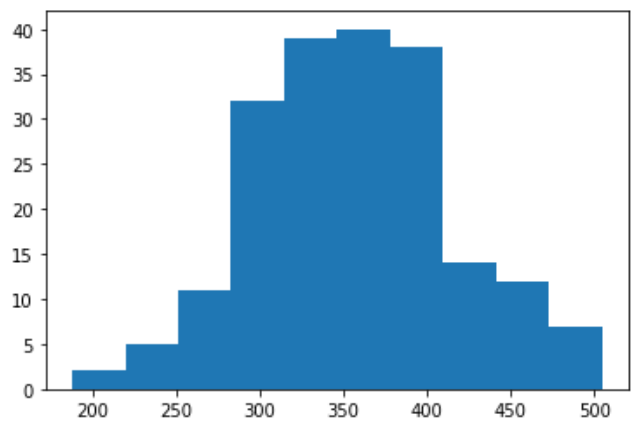

```
plt.scatter(x, y)
```



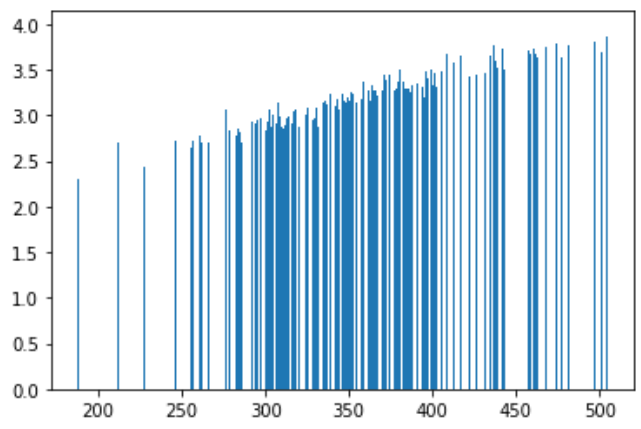

```
plt.stem(x, y)
```



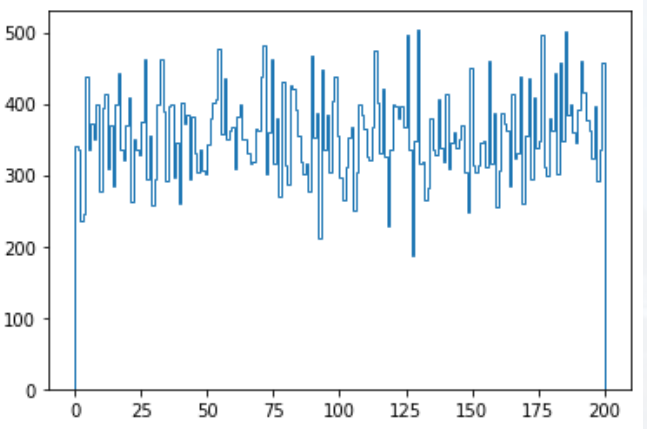

```
plt.hist(x)
```




```
plt.bar(x, y)
```




```
plt.stairs(x)
```



matplotlib


```
plt.scatter(
```

```
scatter(x: 'float | ArrayLike', y: 'float | ArrayLike', s:
        'float | ArrayLike | None' = None, c: 'ArrayLike |
        Sequence[ColorType] | ColorType | None' = None,
        marker: 'MarkerType | None' = None, cmap: 'str |
        Colormap | None' = None, norm: 'str | Normalize |
        None' = None, vmin: 'float | None' = None, vmax:
        'float | None' = None, alpha: 'float | None' = None,
        linewidths: 'float | Sequence[float] | None' = None,
        *, edgecolors: "Literal['face', 'none'] | ColorType
        | Sequence[ColorType] | None" = None, plotnonfinite:
        'bool' = False, data=None, **kwargs,)

A scatter plot of *y* vs. *x* with varying marker size
and/or color.

Parameters
----------
x, y : float or array-like, shape (n, )
The data positions. ...
```

matplotlib

basic plots
arguments
settings

```
plt.scatter(x, y,
            marker = 'p',
            s = 135,
            color = [0.5, 0.1, 0.1],
            edgecolor = 'black',
            alpha = 0.3)
```

marker style: `str`

marker	symbol	description
"."	•	point
","	·	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	Y	tri_down
"2"	⅄	tri_up
"3"	≺	tri_left
"4"	≻	tri_right
"8"	●	octagon
"s"	■	square
"p"	⬟	pentagon
"P"	✚	plus (filled)
"*"	★	star
"h"	⬢	hexagon1
"H"	⬢	hexagon2

"+"	+	plus
"x"	×	x
"X"	✖	x (filled)
"D"	◆	diamond
"d"	♦	thin_diamond
"|"	|	vline
"_"	–	hline
0 (TICKLEFT)	–	tickleft
1 (TICKRIGHT)	–	tickright
2 (TICKUP)	|	tickup
3 (TICKDOWN)	|	tickdown
4 (CARETLEFT)	◀	caretleft
5 (CARETRIGHT)	▶	caretright
6 (CARETUP)	▲	caretup
7 (CARETDOWN)	▼	caretdown
8 (CARETLEFTBASE)	◀	caretleft (centered at base)
9 (CARETRIGHTBASE)	▶	caretright (centered at base)
10 (CARETUPBASE)	▲	caretup (centered at base)
11	▼	caretdown (centered at base)

matplotlib


```
plt.scatter(x, y,
            marker = 'p',
            s = 135,
            color = [0.5, 0.1, 0.1],
            edgecolor = 'black',
            alpha = 0.3)
```

marker size in pixel: `int`

matplotlib


```
plt.scatter(x, y,
            marker = 'p',
            s = 135,
            color = [0.5, 0.1, 0.1],
            edgecolor = 'black',
            alpha = 0.3)
```

color:

`array` RGB code if **three** values, RGB code plus alpha, if **four** values

`str`	full string:	`'green' 'yellow'`
	abbreviation:	`'g' 'y'`
	HEX code:	`'#4b8333' '#fff8de'`

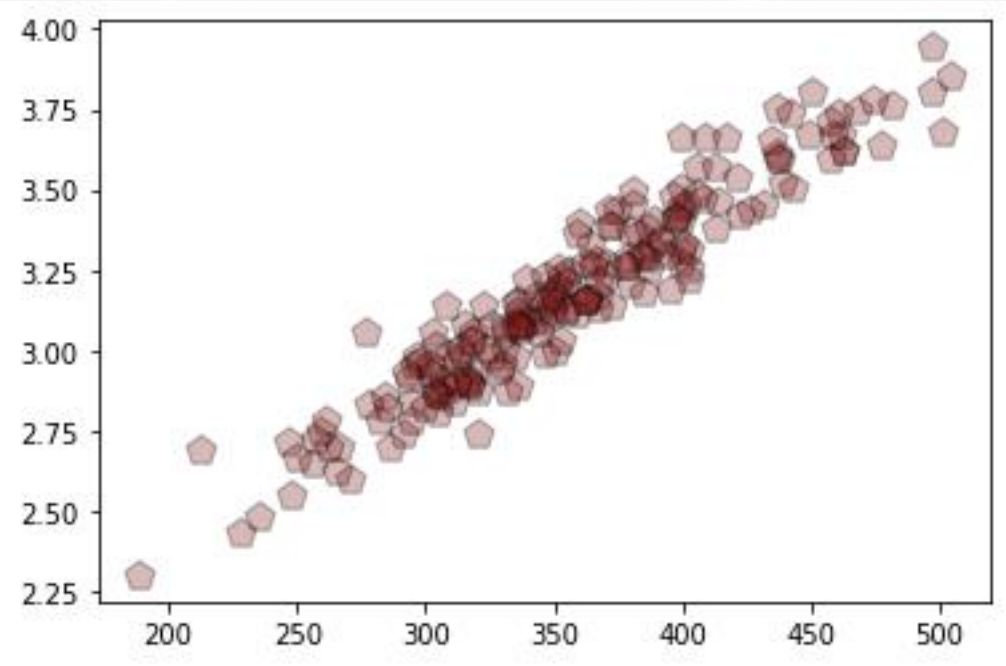
matplotlib


```
plt.scatter(x, y,
            marker = 'p',
            s = 135,
            color = [0.5, 0.1, 0.1],
            edgecolor = 'black',
            alpha = 0.3)
```

alpha (opaqueness/opacity): `float`

matplotlib


```
plt.scatter(x, y,
            marker = 'p',
            s = 135,
            color = [0.5, 0.1, 0.1],
            edgecolor = 'black',
            alpha = 0.3)
```



matplotlib

```
plt.scatter(x, y, marker = 'p', s = 135, color = [0.5, 0.1, 0.1],
                  edgecolor = 'black', alpha = 0.3)

plt.xlabel(r'x values $\tau^{ij}_{def}$')
plt.ylabel('y values')
plt.title('first plot')
plt.legend(['data'])
plt.xscale('log')
plt.savefig('new_plot.pdf')
plt.show()
```

basic plots
arguments
settings

Python speaks LaTeX, but needs raw string

matplotlib

```
plt.scatter(x, y, marker = 'p', s = 135, color = [0.5, 0.1, 0.1],
                edgecolor = 'black', alpha = 0.3)

plt.xlabel(r'x values $\tau^{ij}_{def}$')
plt.ylabel('y values')
plt.title('first plot')
plt.legend(['data'])
plt.xscale('log')
plt.savefig('new_plot.pdf')
plt.show()
```

basic plots
arguments
settings

legend needs to be a `list`

matplotlib

```
plt.scatter(x, y, marker = 'p', s = 135, color = [0.5, 0.1, 0.1],
                  edgecolor = 'black', alpha = 0.3)

plt.xlabel(r'x values $\tau^{ij}_{def}$')
plt.ylabel('y values')
plt.title('first plot')
plt.legend(['data'])
plt.xscale('log')
plt.savefig('new_plot.pdf')
plt.show()
```

basic plots
arguments
settings

plots can be saved to any common format

matplotlib

basic plots
arguments
settings

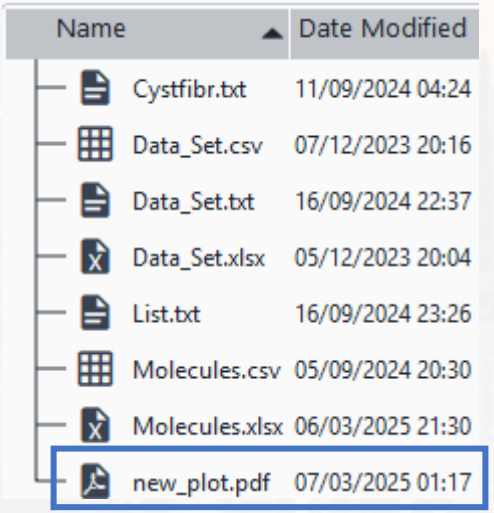
```
plt.scatter(x, y, marker = 'p', s = 135, color = [0.5, 0.1, 0.1],
                edgecolor = 'black', alpha = 0.3)

plt.xlabel(r'x values $\tau^{ij}_{def}$')
plt.ylabel('y values')
plt.title('first plot')
plt.legend(['data'])
plt.xscale('log')
plt.savefig('new_plot.pdf')
plt.show()
```

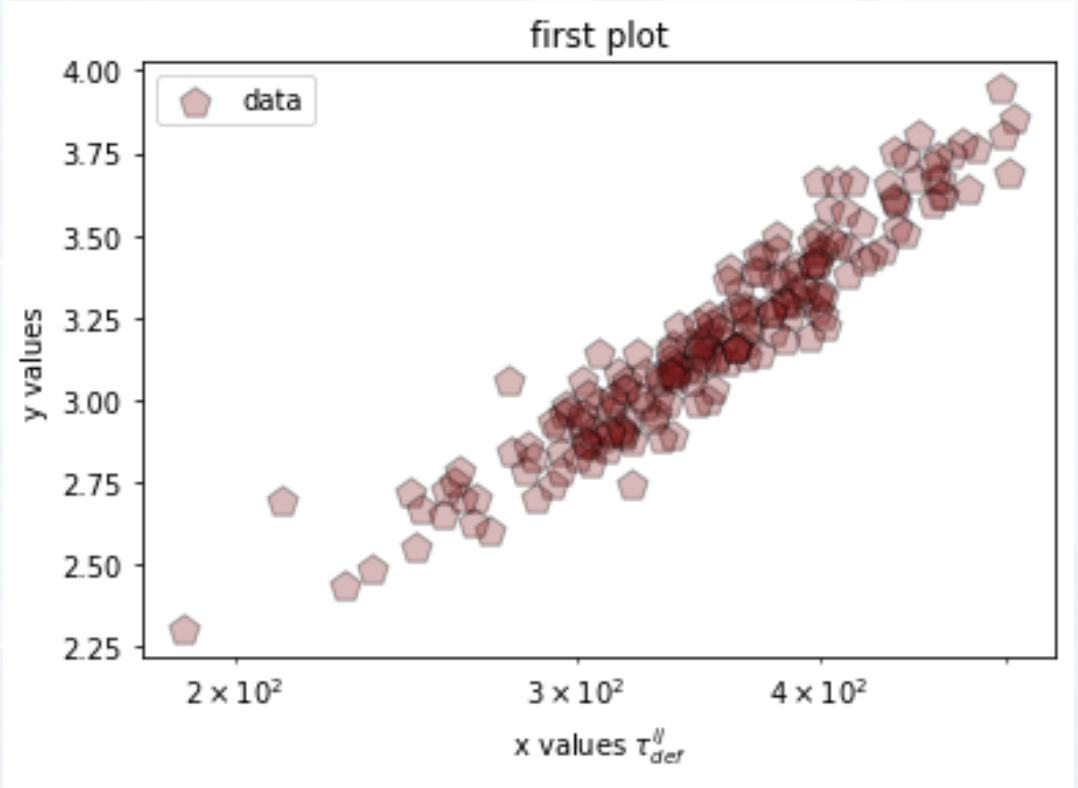
sometims plots don‘t show up
(depending on settings)
type `plt.show()`
at the **very end**

matplotlib

basic plots
arguments
settings

```
plt.scatter(x, y, marker = 'p', s = 135, color = [0.5, 0.1, 0.1],
                edgecolor = 'black', alpha = 0.3)

plt.xlabel(r'x values $\tau^{ij}_{def}$')
plt.ylabel('y values')
plt.title('first plot')
plt.legend(['data'])
plt.xscale('log')
plt.savefig('new_plot.pdf')
plt.show()
```





matplotlib

basic plots
arguments
settings

more than one data set:

```
for a in range(3):
    a += 1
    plt.scatter(x, y**a,
                marker = 'p',
                s      = 30,
                alpha  = 0.3,
                label  = 'Data Set ' + str(a))


plt.xlabel(r'x values')
plt.ylabel('y values')
plt.title('first plot')
plt.legend()
plt.show()
```

assigns a label to each data set → stored for legend

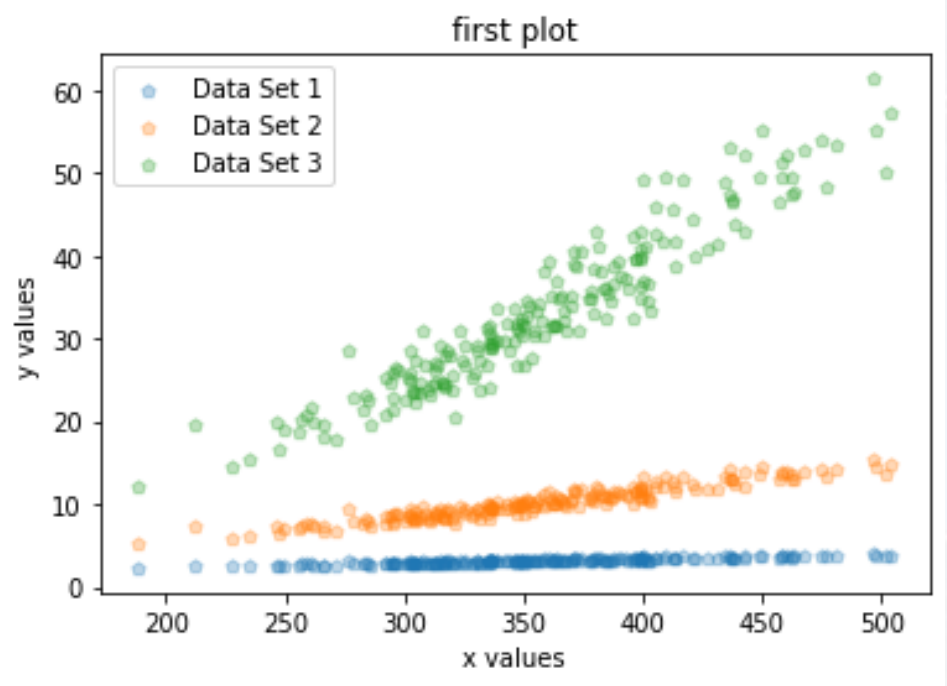
calling the legend (no input argument)

matplotlib

basic plots
arguments
settings

more than one data set:

```
for a in range(3):
    a += 1
    plt.scatter(x, y**a,
                marker = 'p',
                s      = 30,
                alpha  = 0.3,
                label  = 'Data Set ' + str(a))


plt.xlabel(r'x values')
plt.ylabel('y values')
plt.title('first plot')
plt.legend()
plt.show()
```



Outline

Reading/Writing Files

- Pandas and Data Frames
- Faster Alternatives
- More about `.txt`

Visualization

- Matplotlib
- **Seaborn**
- `plt, ax, fig`

Data Sampling and Pandas:

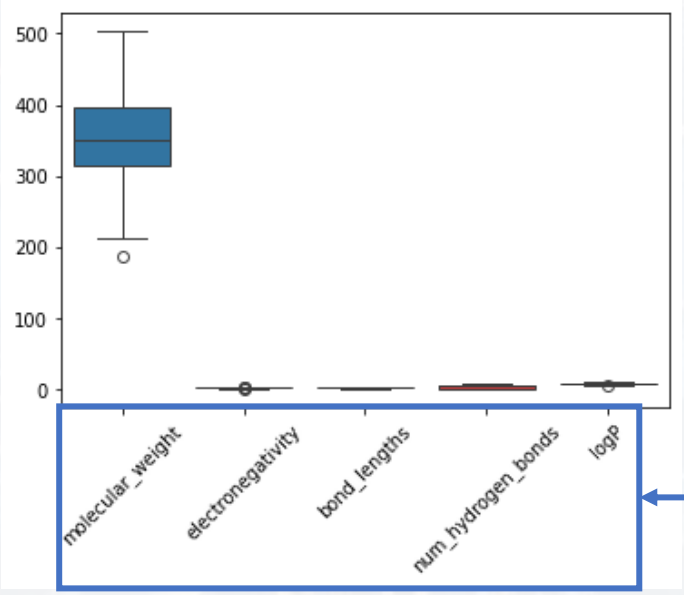
seaborn

sophisticated plots:

```
import seaborn as sns
```

starting from **data frames** right away

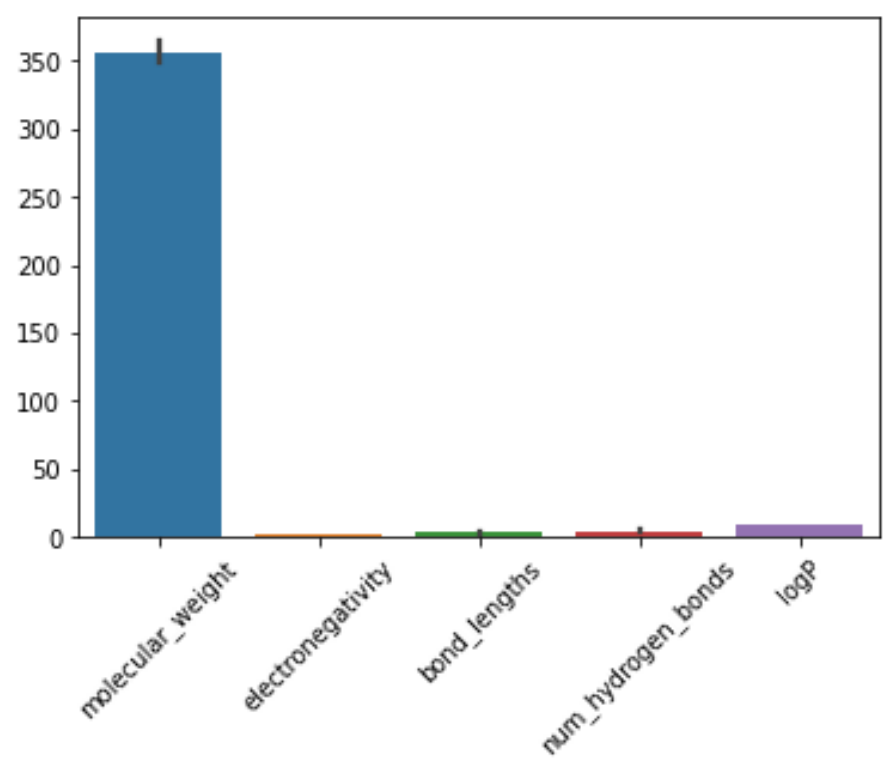
```
sns.boxplot(Data)
plt.xticks(rotation = 45)
plt.show()
```



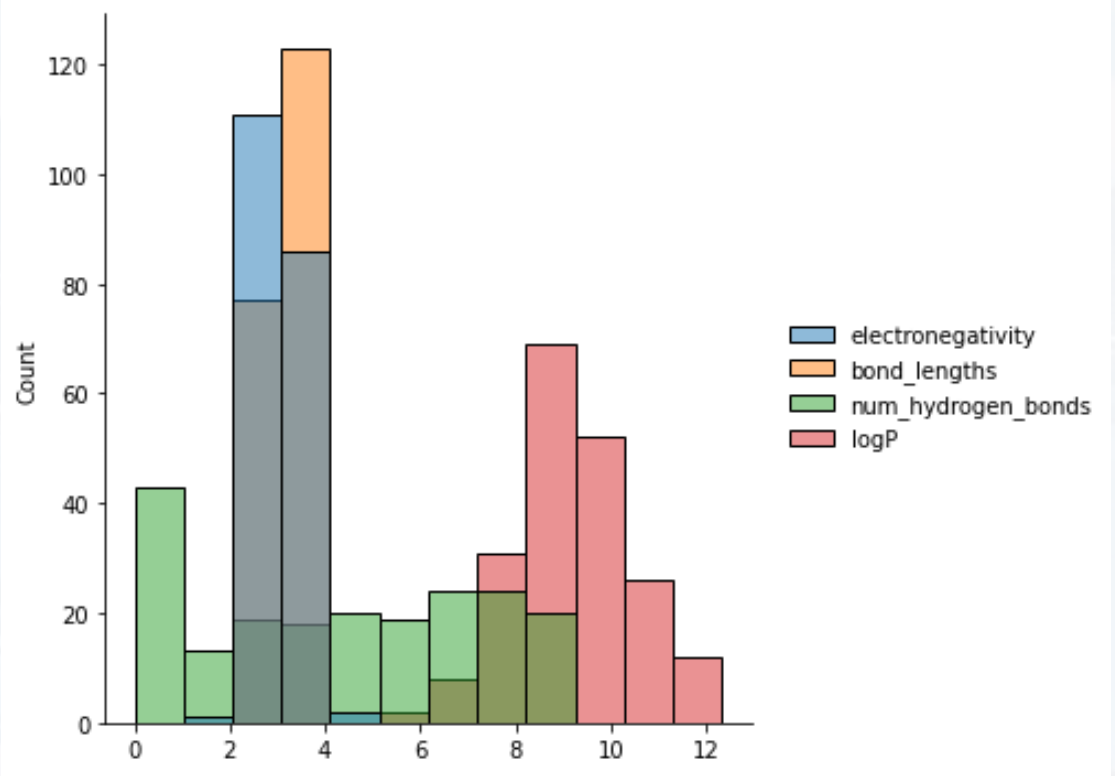
reads labels from data frame

seaborn

```
sns.barplot(Data)
plt.xticks(rotation = 45)
plt.show()
```

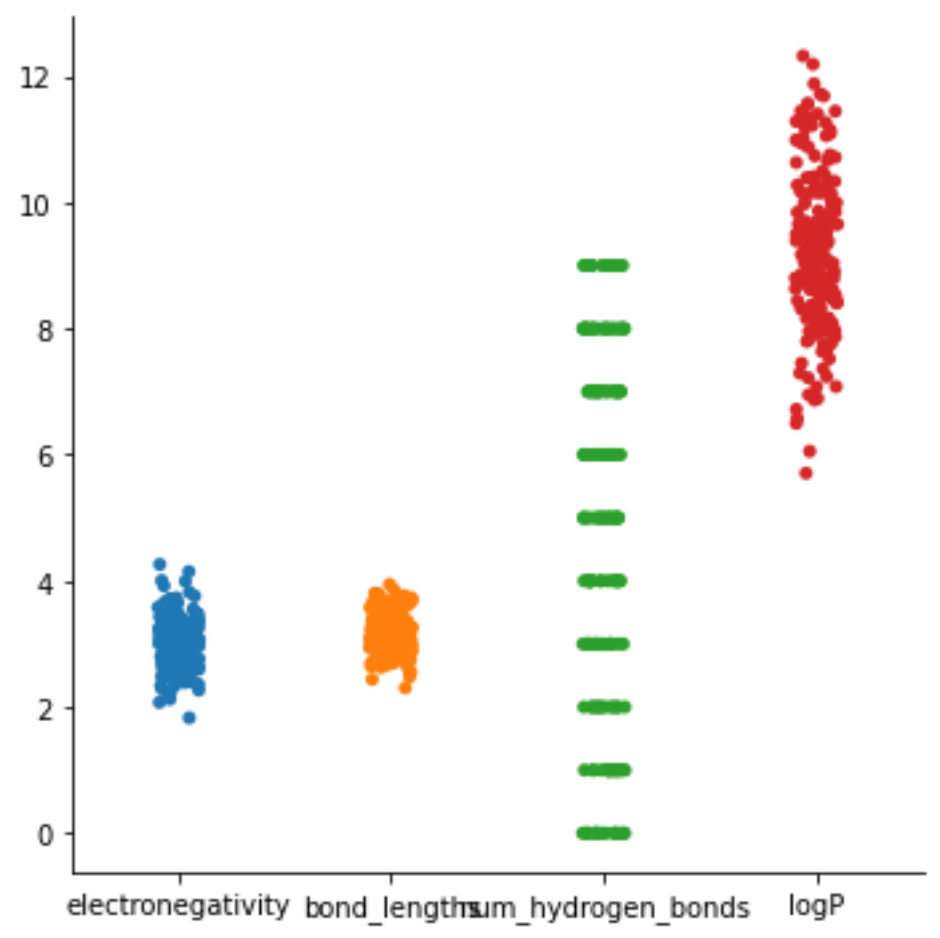



```
sns.displot(Data[Data.columns[1:-1]])
plt.show()
```



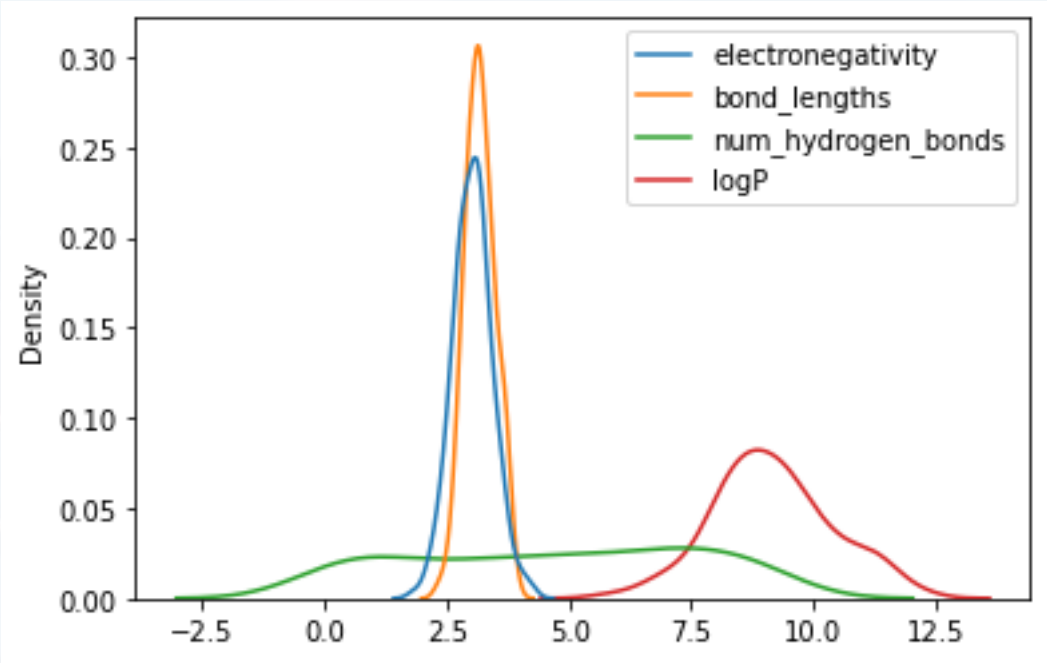
seaborn

```
sns.catplot(Data[Data.columns[1:-1]])
plt.show()
```



seaborn

```
sns.kdeplot(Data[Data.columns[1:-1]])
plt.show()
```



Graph Gallery with examples

seaborn

```
sns.dogplot()
```

```
sns.dogplot(1)
```

Outline

Reading/Writing Files

- Pandas and Data Frames
- Faster Alternatives
- More about `.txt`

Visualization

- Matplotlib
- Seaborn
- `plt, ax, fig`

Data Sampling and Pandas:


```
.plt. ax
fig
```

Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure (which can include numerous subplots)

mosaic subplots
classical subplots

Data Sampling and Pandas:

.plt. ax
fig

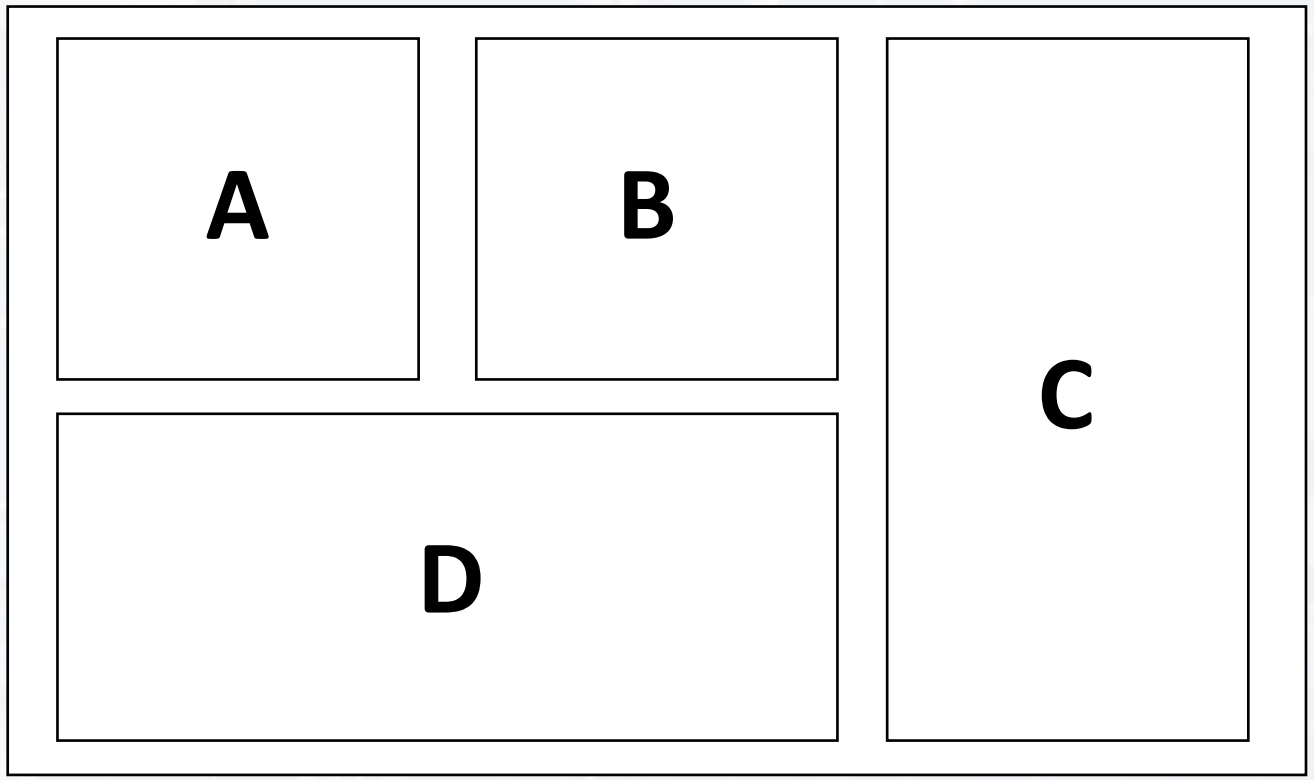
Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure (which can include numerous subplots)

mosaic subplots

classical subplots

```
figMo, axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                  ['D', 'D', 'C']], layout = "constrained")
```



Data Sampling and Pandas:

.plt. ax
fig

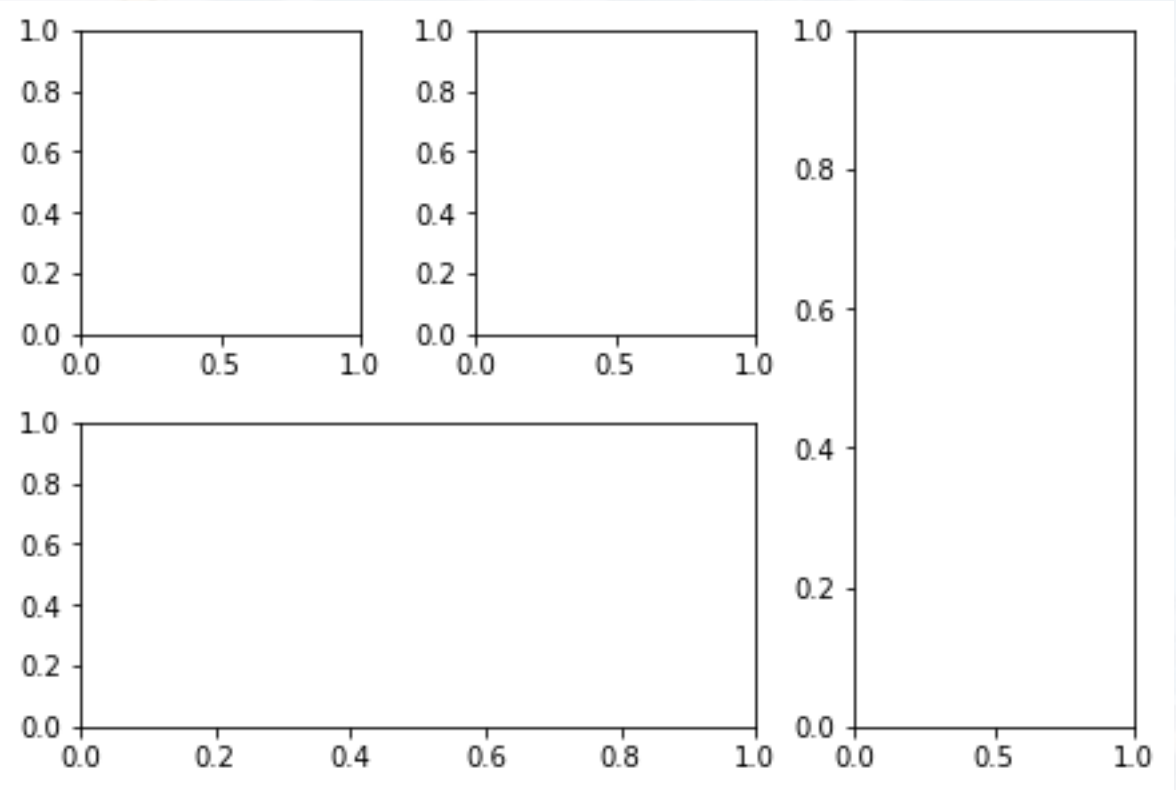
Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure (which can include numerous subplots)

mosaic subplots

classical subplots

```
figMo, axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                  ['D', 'D', 'C']], layout = "constrained")
```



`.plt. ax`
`fig`

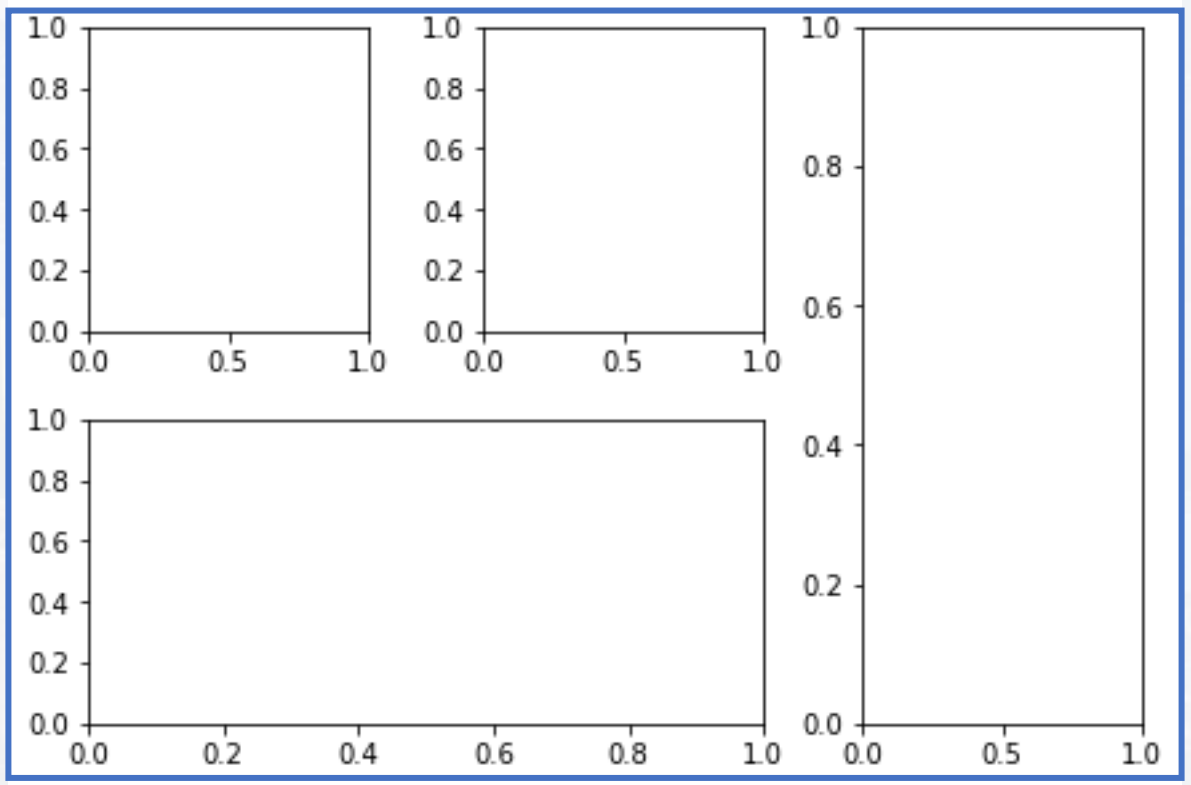
Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure (which can include numerous subplots)

mosaic subplots

classical subplots

```
figMo, axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                  ['D', 'D', 'C']], layout = "constrained")
```



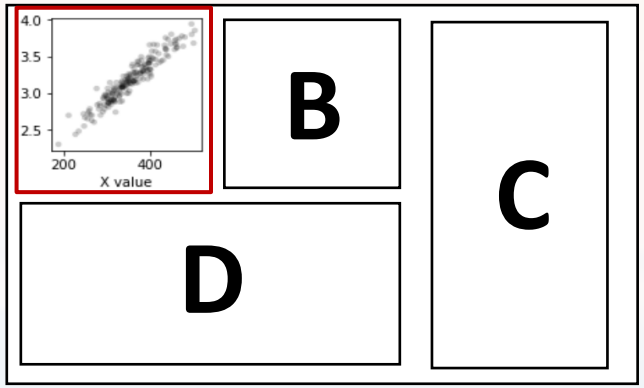
points to the current figure

Data Sampling and Pandas:

`.plt. ax`
`fig`

Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure



mosaic subplots

classical subplots

```
figMo, axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                  ['D', 'D', 'C']], layout = "constrained")
```

populating the first axis object with a **scatter plot**

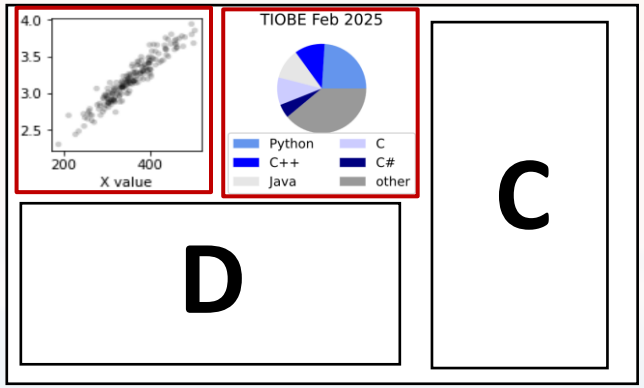
```
axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')
```

Data Sampling and Pandas:

`.plt. ax`
`fig`

Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure



mosaic subplots

classical subplots

```
figMo,  axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                   ['D', 'D', 'C']], layout = "constrained")

axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')

axes['B'].pie([24, 11, 11, 10, 5, 39],\
              colors = ['#6495ED', 'blue', [0.9, 0.9, 0.9], '#CCCCFF',\
              '#000080', '#999999'])
axes['B'].set(title = 'TIOBE Feb 2025')
axes['B'].legend(['Python', 'C++', 'Java', 'C', 'C#', 'other'],\
                 bbox_to_anchor = (0.5,-0.5), loc = 'lower center',\
                 ncol = 2)
```

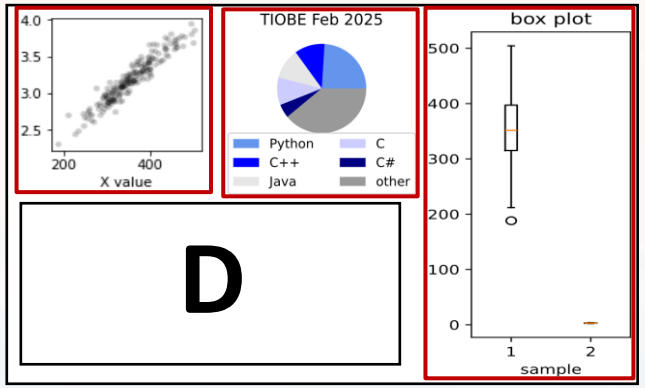
populating the second axis object with a **pie chart**

Data Sampling and Pandas:

`.plt. ax`
`fig`

Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure



mosaic subplots

classical subplots

```
figMo,  axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                   ['D', 'D', 'C']], layout = "constrained")

axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')

axes['B'].pie([24, 11, 11, 10, 5, 39], colors = ['#6495ED', 'blue', [0.9, 0.9, 0.9], '#CCCCFF',\
                                                 '#000080', '#999999'])
axes['B'].set(title = 'TIOBE Feb 2025')
axes['B'].legend(['Python', 'C++', 'Java', 'C', 'C#', 'other'],\
                 bbox_to_anchor = (0.5,-0.5), loc = 'lower center', ncol = 2)
```

and so on…

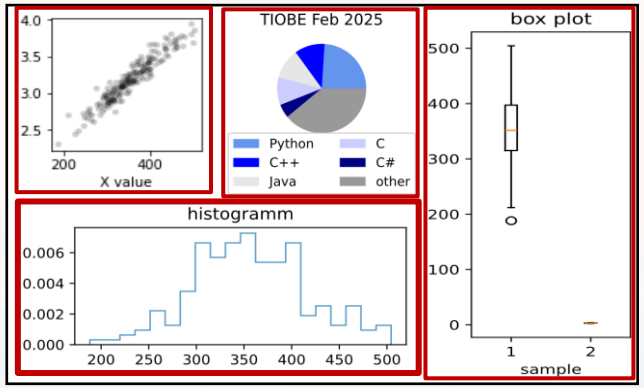
```
axes['C'].boxplot([x,y])
axes['C'].set(xlabel = 'sample')
axes['C'].set(ylabel = 'values')
axes['C'].set(title  = 'box plot')
```

Data Sampling and Pandas:


```
.plt. ax
fig
```

Three levels in Python:

plt	the plot itself
ax	referring to axis of specific plot
fig	referring to a figure



mosaic subplots

classical subplots

```
figMo,  axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                   ['D', 'D', 'C']], layout = "constrained")

axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')

axes['B'].pie([24, 11, 11, 10, 5, 39], colors = ['#6495ED', 'blue', [0.9, 0.9, 0.9], '#CCCCFF',\
                                                 '#000080', '#999999'])
axes['B'].set(title = 'TIOBE Feb 2025')
axes['B'].legend(['Python', 'C++', 'Java', 'C', 'C#', 'other'],\
                 bbox_to_anchor = (0.5,-0.5), loc = 'lower center', ncol = 2)
axes['C'].boxplot([x,y])
axes['C'].set(xlabel = 'sample')
axes['C'].set(ylabel = 'values')
axes['C'].set(title  = 'box plot')
axes['D'].hist(x, 20, density = True, histtype = 'step', facecolor = 'g',\
               alpha = 0.75)
axes['D'].set(title = 'histogram')
```

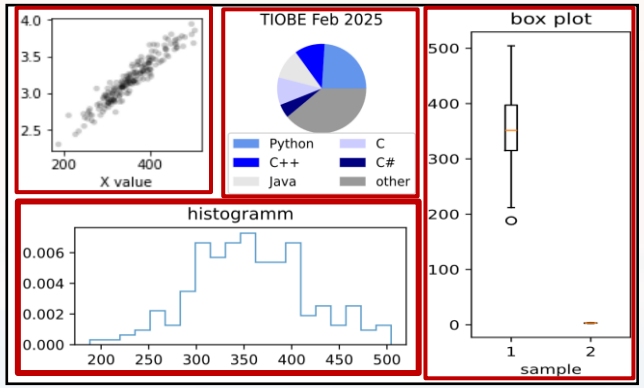
and so on…

Data Sampling and Pandas:


```
.plt. ax
fig
```

Three levels in Python:

plt	the plot itself
ax	referring to axis of specific plot
fig	referring to a figure



mosaic subplots

classical subplots

reference to the individual axes

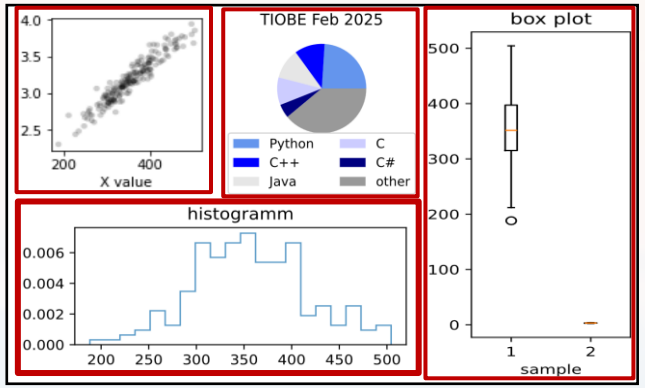
```
figMo,  axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                   ['D', 'D', 'C']], layout = "constrained")

axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')

axes['B'].pie([24, 11, 11, 10, 5, 39], colors = ['#6495ED', 'blue', [0.9, 0.9, 0.9], '#CCCCFF',\
                                                 '#000080', '#999999'])
axes['B'].set(title = 'TIOBE Feb 2025')
axes['B'].legend(['Python', 'C++', 'Java', 'C', 'C#', 'other'],\
                 bbox_to_anchor = (0.5,-0.5), loc = 'lower center', ncol = 2)
axes['C'].boxplot([x,y])
axes['C'].set(xlabel = 'sample')
axes['C'].set(ylabel = 'values')
axes['C'].set(title  = 'box plot')

axes['D'].hist(x, 20, density = True, histtype = 'step', facecolor = 'g',\
         alpha = 0.75)
axes['D'].set(title = 'histogram')
```

Data Sampling and Pandas:


```
.plt. ax
fig
```

Three levels in Python:

plt	the plot itself
ax	referring to axis of specific plot
fig	referring to a figure



mosaic subplots

classical subplots

```
figMo,  axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                   ['D', 'D', 'C']], layout = "constrained")

axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')

axes['B'].pie([24, 11, 11, 10, 5, 39], colors = ['#6495ED', 'blue', [0.9, 0.9, 0.9], '#CCCCFF',\
                                                 '#000080', '#999999'])
axes['B'].set(title = 'TIOBE Feb 2025')
axes['B'].legend(['Python', 'C++', 'Java', 'C', 'C#', 'other'],\
                 bbox_to_anchor = (0.5,-0.5), loc = 'lower center', ncol = 2)
axes['C'].boxplot([x,y])
axes['C'].set(xlabel = 'sample')
axes['C'].set(ylabel = 'values')
axes['C'].set(title  = 'box plot')

axes['D'].hist(x, 20, density = True, histtype = 'step', facecolor = 'g',\
         alpha = 0.75)
axes['D'].set(title = 'histogram')
```

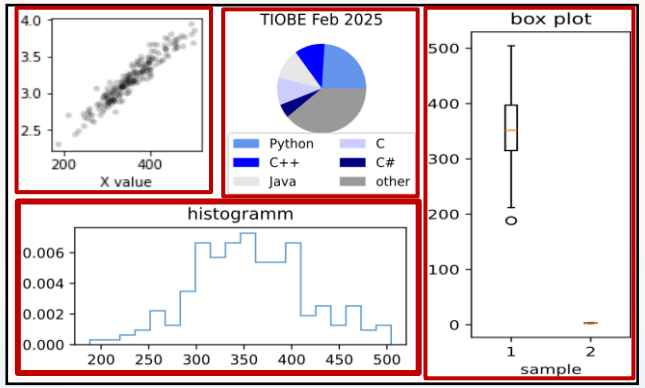
notice the different color codes

Data Sampling and Pandas:

`.plt. ax`
`fig`

Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure



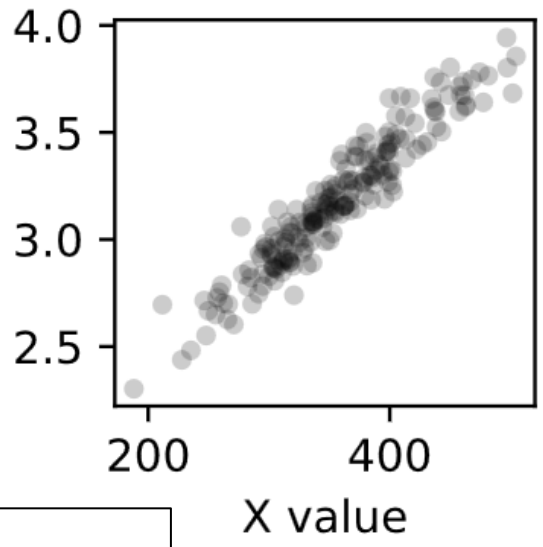
mosaic subplots

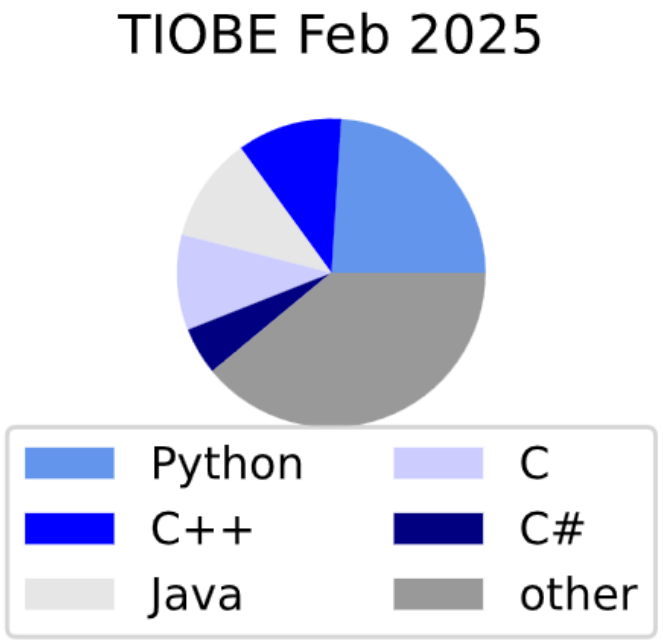
classical subplots

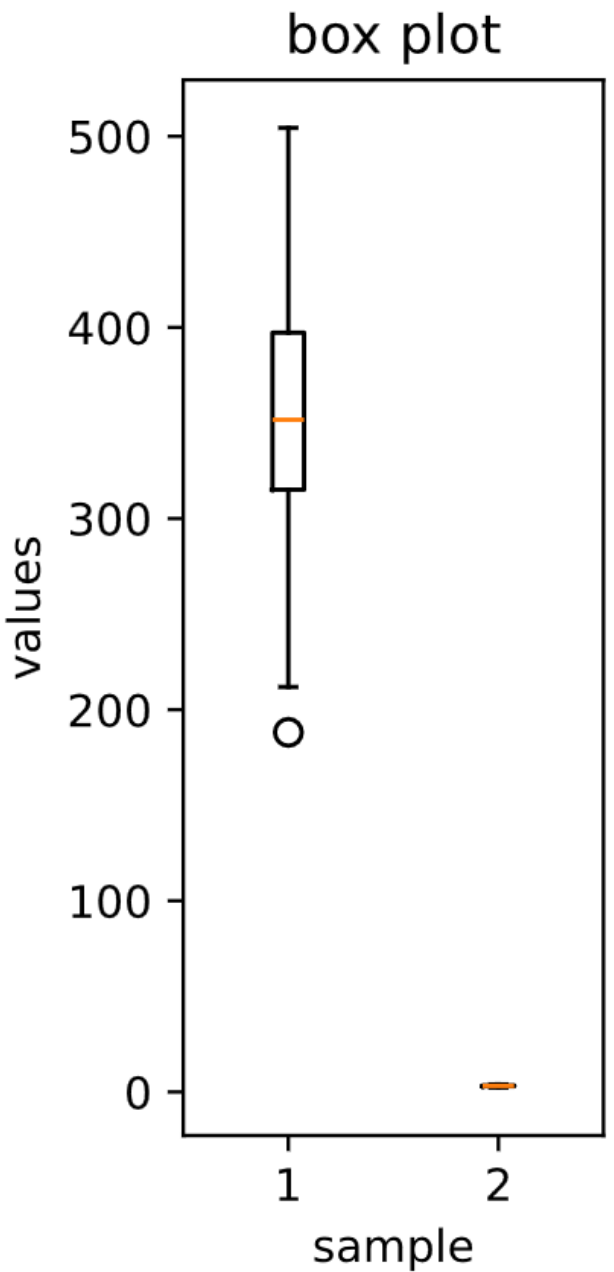
```
figMo, axes = plt.subplot_mosaic([['A', 'B', 'C'],\
                                  ['D', 'D', 'C']], layout = "constrained")

axes['A'].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes['A'].set(xlabel = 'X value')

...

axes['D'].hist(x, 20, density = True, histtype = 'step', facecolor = 'g',\
         alpha = 0.75)
axes['D'].set(title = 'histogram')


figMo.savefig('test.pdf', dpi = 1600)
```

referring to the specific figure

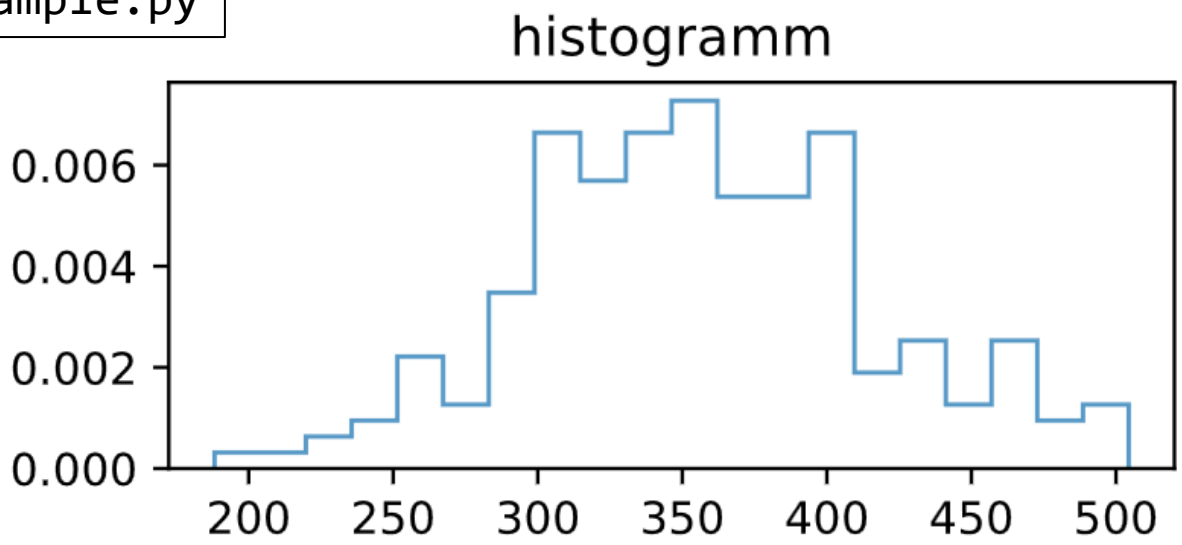
```
.plt. ax
fig
```







check out
`PlotMosaicExample.py`



Data Sampling and Pandas:

.plt. ax
fig

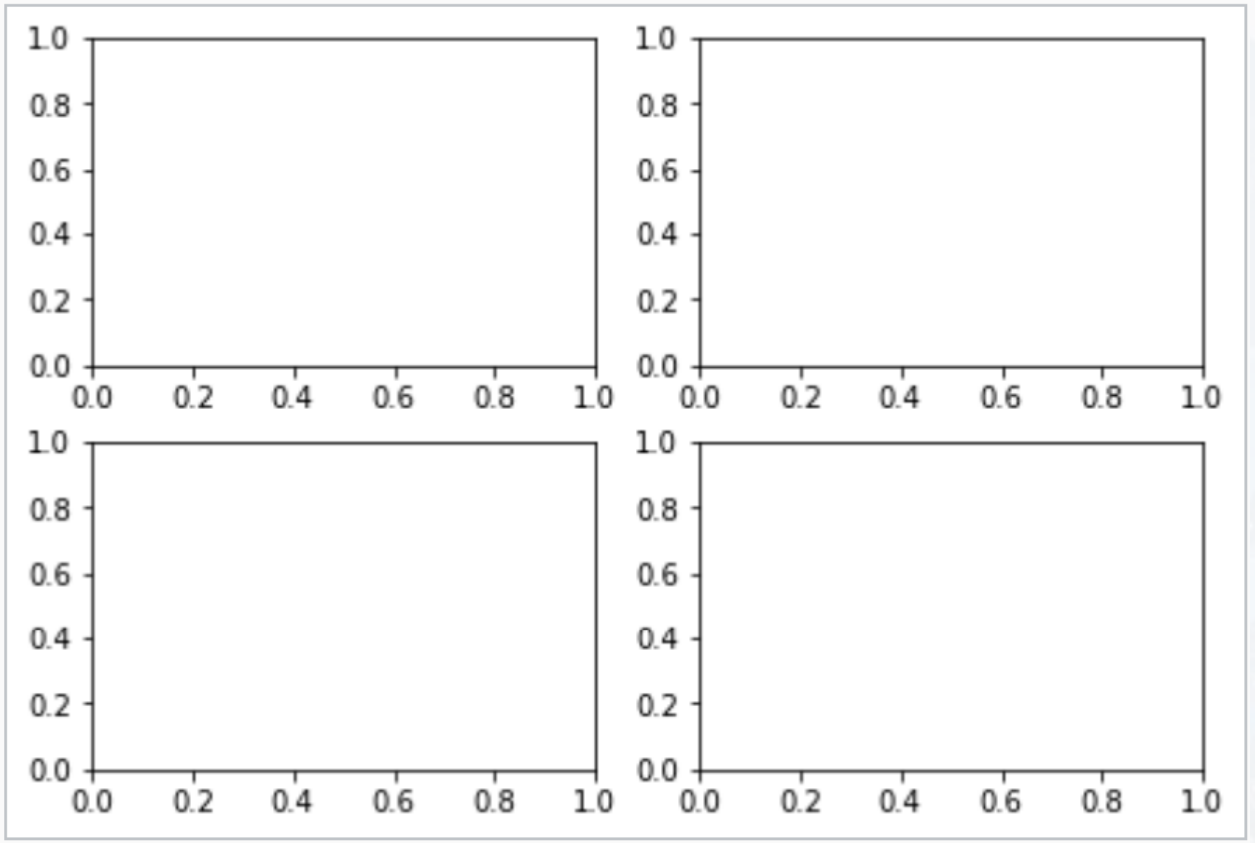
Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure (which can include numerous subplots)

mosaic subplots

classical subplots **Same idea!**

```
figSt, axes = plt.subplots(nrows = 2, ncols = 2, layout = "constrained")
```



Data Sampling and Pandas:

`.plt. ax`
`fig`

Three levels in Python:

`plt`	the plot itself
`ax`	referring to axis of specific plot
`fig`	referring to a figure (which can include numerous subplots)

mosaic subplots

classical subplots **Same idea!**

in mosaic mode: label can be numeric or `str`. here: has to be `int` (referres to locaton in figure)

```
figSt, axes = plt.subplots(nrows = 2, ncols = 2, layout = "constrained")

axes[0, 0].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes[0, 0].set(xlabel = 'X value')

axes[1, 0].pie([24, 11, 11, 10, 5, 39], colors = ['#6495ED', 'blue', [0.9, 0.9, 0.9],\
                                                  '#CCCCFF', '#000080', '#999999'])
axes[1, 0].set(title = 'TIOBE Feb 2025')
axes[1, 0].legend(['Python', 'C++', 'Java', 'C', 'C#', 'other'],\
                  bbox_to_anchor = (0.5, -0.5), loc = 'lower center', ncol = 2)


axes[0, 1].boxplot([x, y])
axes[0, 1].set(xlabel = 'sample')
axes[0, 1].set(ylabel = 'values')
axes[0, 1].set(title  = 'box plot')

axes[1, 1].hist(x, 20, density = True, histtype = 'step', facecolor = 'g', alpha = 0.75)
axes[1, 1].set(title = 'histogram')
```

Data Sampling and Pandas:

.plt. ax
fig

Three levels in Python:

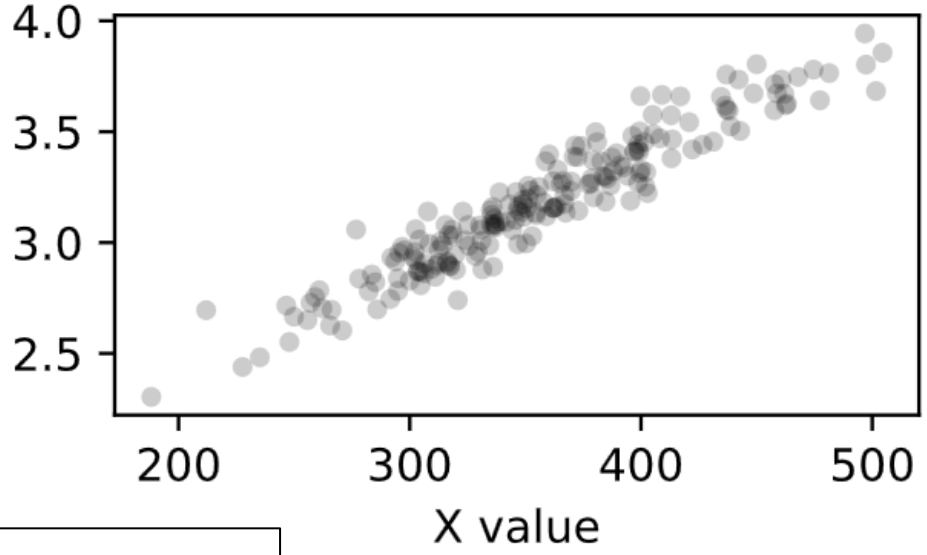
plt	the plot itself
ax	referring to axis of specific plot
fig	referring to a figure (which can include numerous subplots)

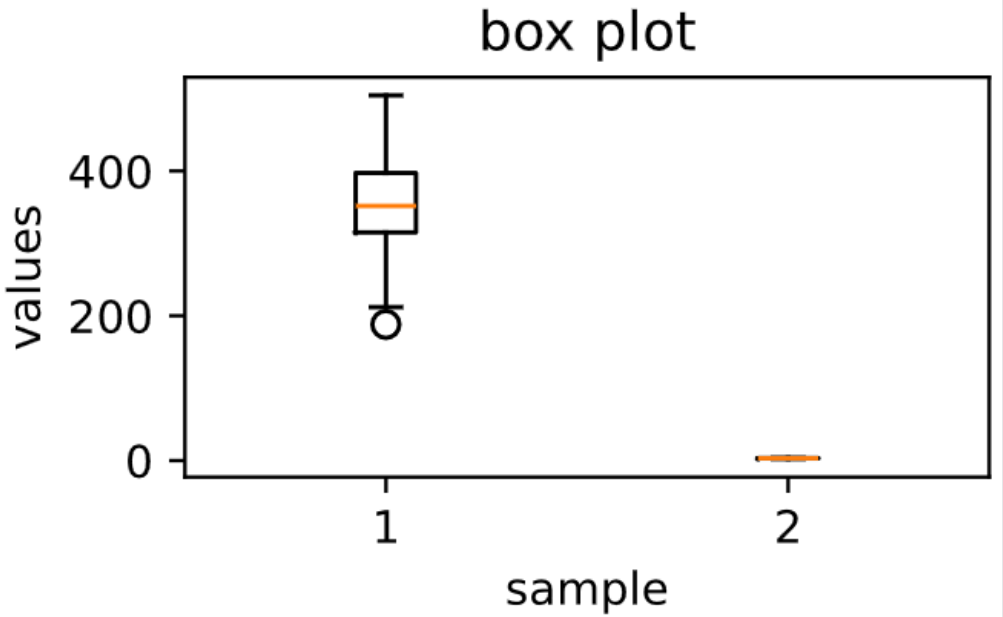
mosaic subplots

classical subplots **Same idea!**

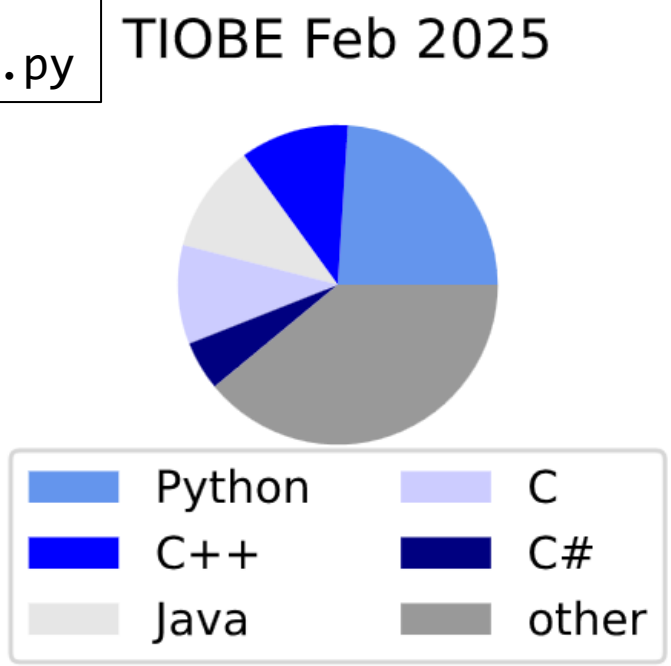
```
figSt, axes = plt.subplots(nrows = 2, ncols = 2, layout = "constrained")

axes[0, 0].scatter(x, y, s = 20, c = 'k', alpha = 0.2, edgecolors = 'none')
axes[0, 0].set(xlabel = 'X value')

...

axes[1, 1].hist(x, 20, density=True, histtype = 'step', facecolor = 'g', alpha = 0.75)
axes[1, 1].set(title = 'histogramm‘)

figSt.savefig('test.pdf', dpi = 1600)
```

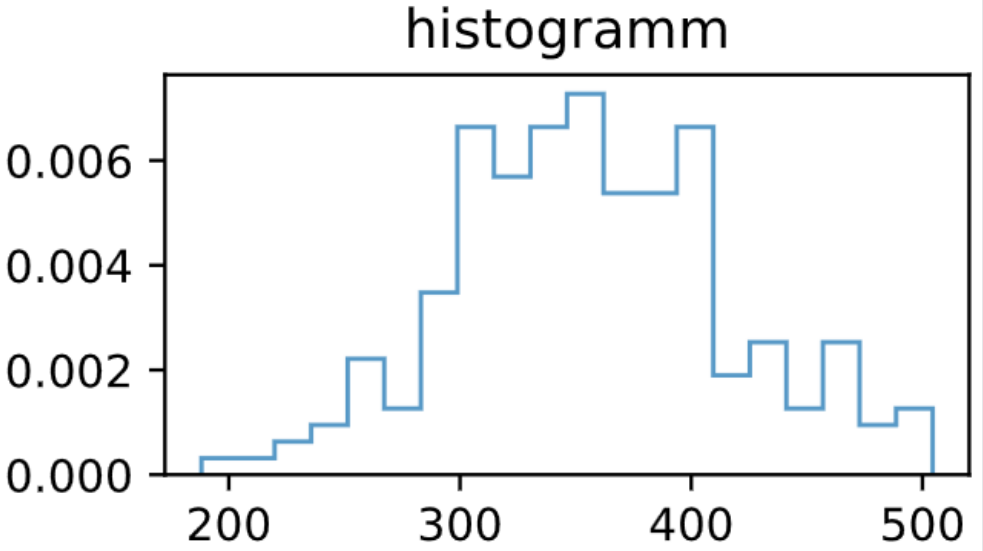
```
.plt. ax
fig
```





check out
`PlotStandardExample.py`






```
75)
```

Thank you very much for your attention!