uc3m | Universidad Carlos III de Madrid

OPENCOURSEWARE
ADVANCED PROGRAMMING
STATISTICS FOR DATA SCIENCE

Ricardo Aler

Visualization in Python

Visualization in Python

- Matplotlib:
  - https://matplotlib.org/contents.html
  - It plots numpy arrays (but not Pandas dataframes)
- Seaborn:
  - https://seaborn.pydata.org/
  - It works with Pandas dataframes
- Other options:
  - Bokeh (interactive)
  - ggplot2 (for python)

Matplotlib

- Contexts:
  - Script (i.e. your code is in a file, for instance mycode.py)
  - Terminal / shell (e.g. in Spyder): plots are done interactively in the Spyder console
  - Jupyter notebook

Matplotlib: in a script

- Figures are constructed step by step
- At the end, plt.show()

Matplotlib: in a script (Python code in a file)

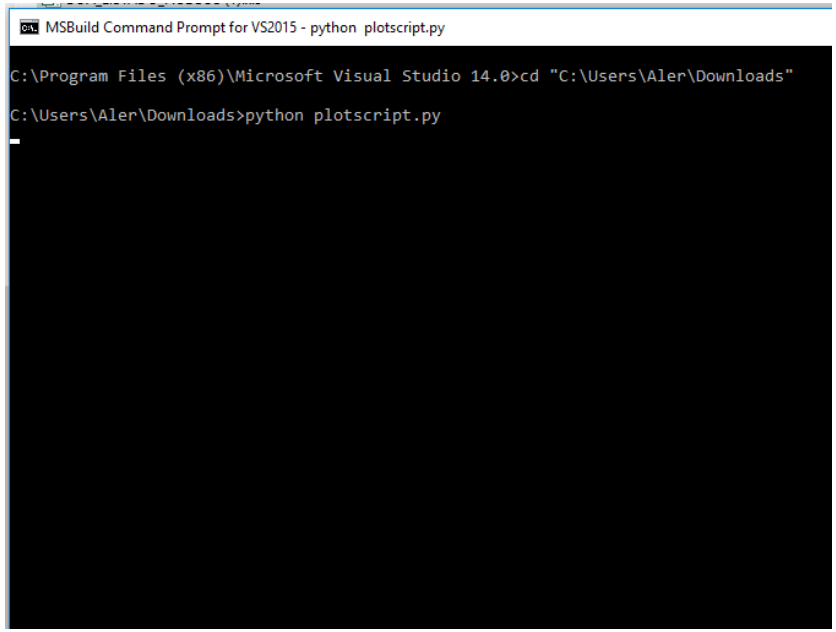
```python
# -*- coding: utf-8 -*-
"""
Created on Tue Jan  7 19:44:29 2020

@author: Aler
"""

import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, 0.01*x**2)
# show must be used in a script
plt.show()
```

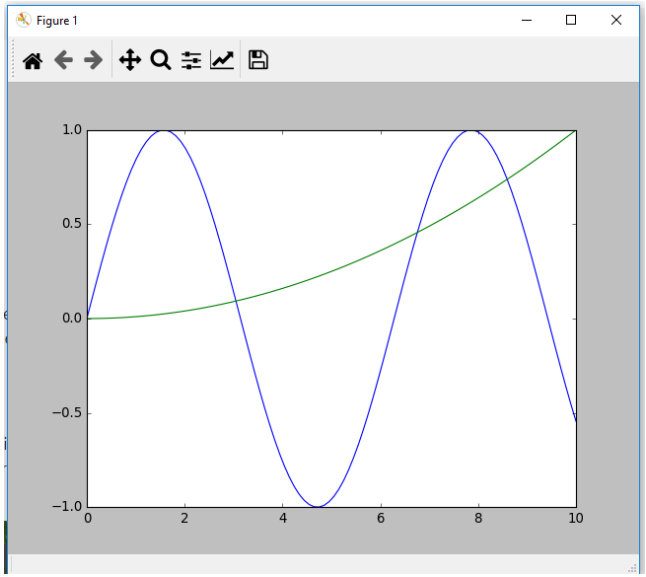
- Note: plt.show should be used just once, at the end of the file, when all the components of the figure have been created.

```python
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, 0.01*x**2)
# show must be used in a script
plt.show()
```

Matplotlib: in a script (Python code in a file)

- We now execute the script

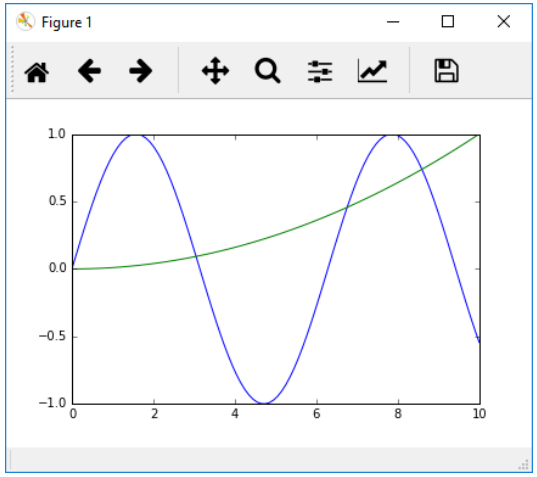




Matplotlib: in a shell (e.g.: Spyder)

- With %matplotlib, in spyder, a new window will open
- Every plot command updates the plot. *plt.show* is **not** required. *plt.draw()* might be needed in some cases to update the plot.

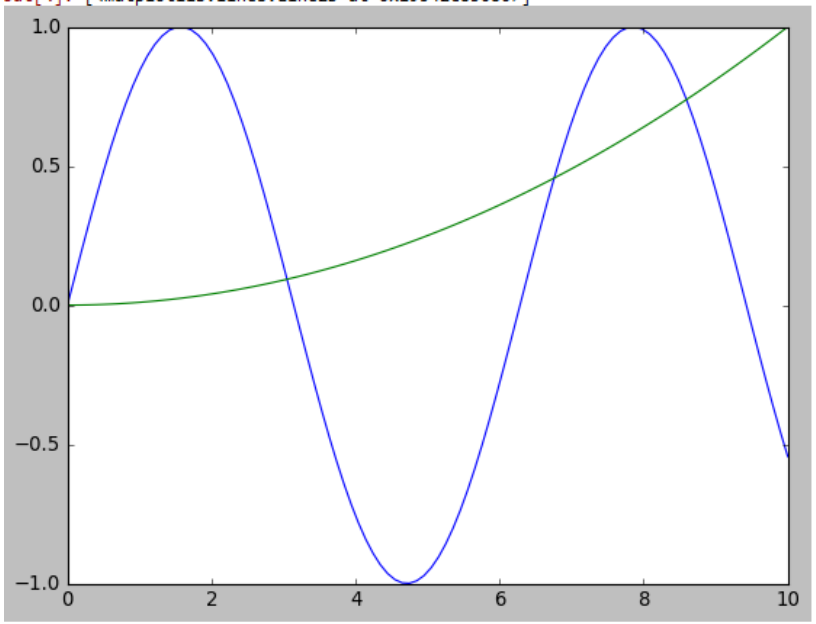
```
%matplotlib
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, 0.01*x**2)
```



Matplotlib: in a shell (e.g.: Spyder)

- If %matplotlib is not used, no window is opened in spyder, but the whole sequence of commands must be executed in a single go
  - (e.g. selecting the whole code block and using F9)

```
In [4]: import matplotlib.pyplot as plt
   ...: import numpy as np
   ...: plt.style.use('classic')
   ...:
   ...: x = np.linspace(0, 10, 100)
   ...: plt.plot(x, np.sin(x))
   ...: plt.plot(x, 0.01*x**2)
   ...:
Out[4]: [<matplotlib.lines.Line2D at 0x19542c83080>]
```



Matplotlib: in a jupyter notebook

- "Interactive" plots (zoom, pan, ...)

  %matplotlib notebook

- Static plots:

  %matplotlib inline

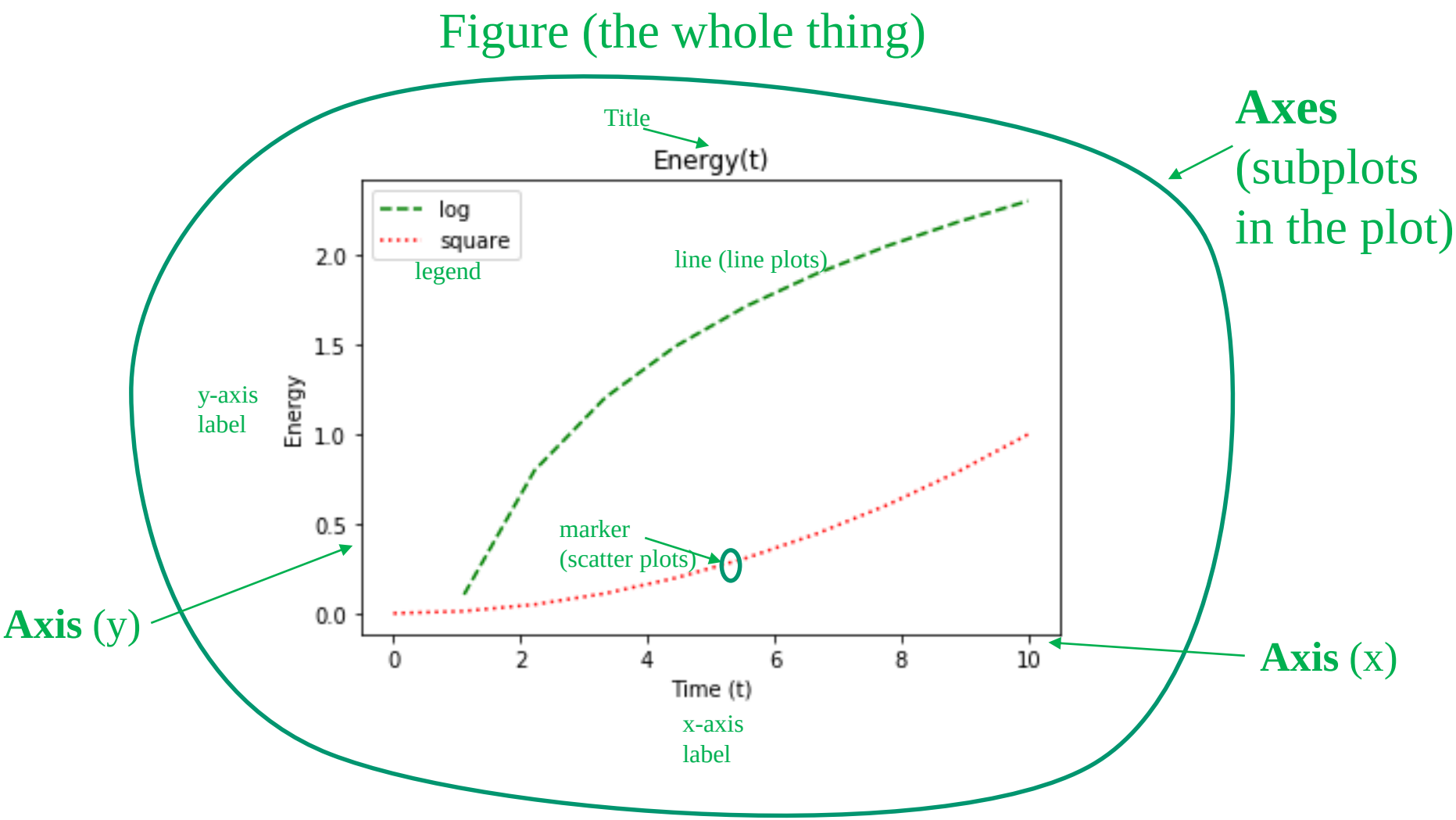
- %matplotlib, just once at the beginning of the notebook

Saving to a file

```
x = np.linspace(0, 10, 100)
fig = plt.figure()
plt.plot(x, 0.01*x**2)

# Save figure to file
fig.savefig('figure.png')

# Recover figure from file
from IPython.display import Image
Image('my_figure.png')
```

Other formats: fig.savefig("figure.pdf", format="pdf")

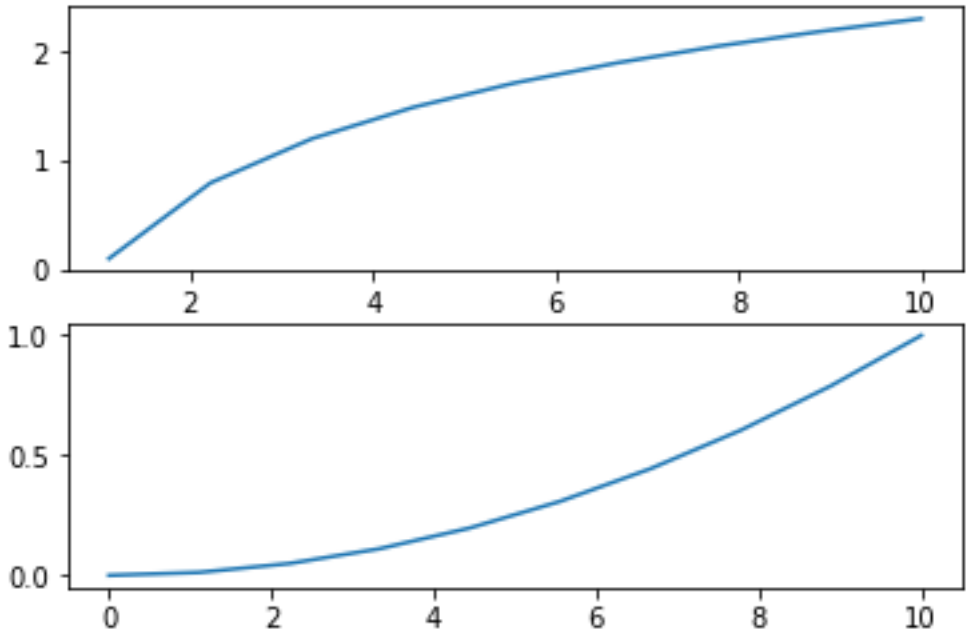
Matplotlib: general concepts



Matplotlib: general concepts

- A Figure can contain several subplots (Axes)

Figure (the whole thing)

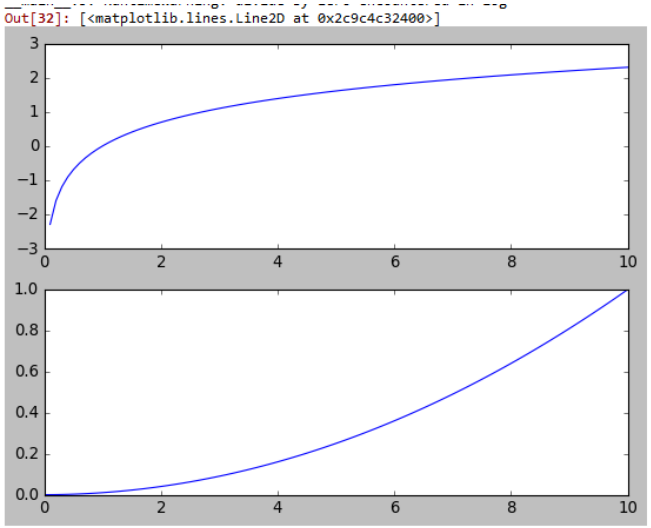


Axes 1 (subplot 1)

Axes 2 (subplot 2)

Two ways to use Matplotlib

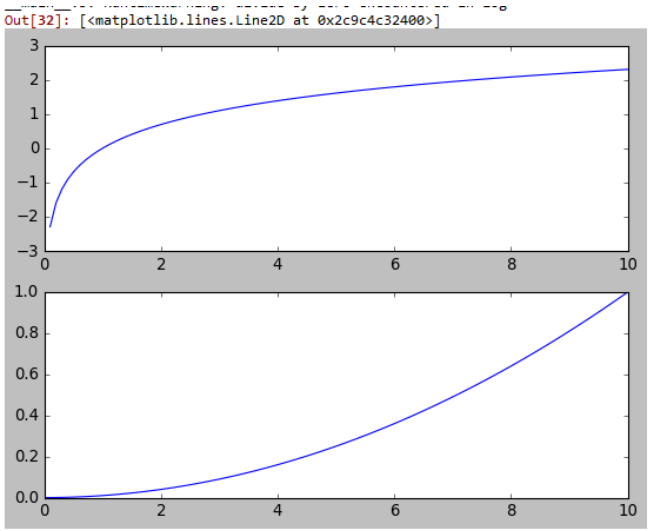
- Matlab-like:
  - Each subplot is defined with plt.subplot sentences.
- Object oriented:
  - Each subplot (axes) is a Python object.
  - Python methods are applied on each object



Two ways to use Matplotlib

- Matlab-like: plt.subplot(number of rows, number of columns, plot number)

```
# Create an (empty) figure. It is the active figure
import matplotlib.pyplot as plt
plt.figure()
# plt.subplot(number of rows, of columns, panel number)
# First axes for the active figure
plt.subplot(2, 1, 1)
# plot for the active axes
plt.plot(x, np.log(x))
plt.subplot(2, 1, 2)
# plot for the active axes
plt.plot(x, 0.01*x**2)
```



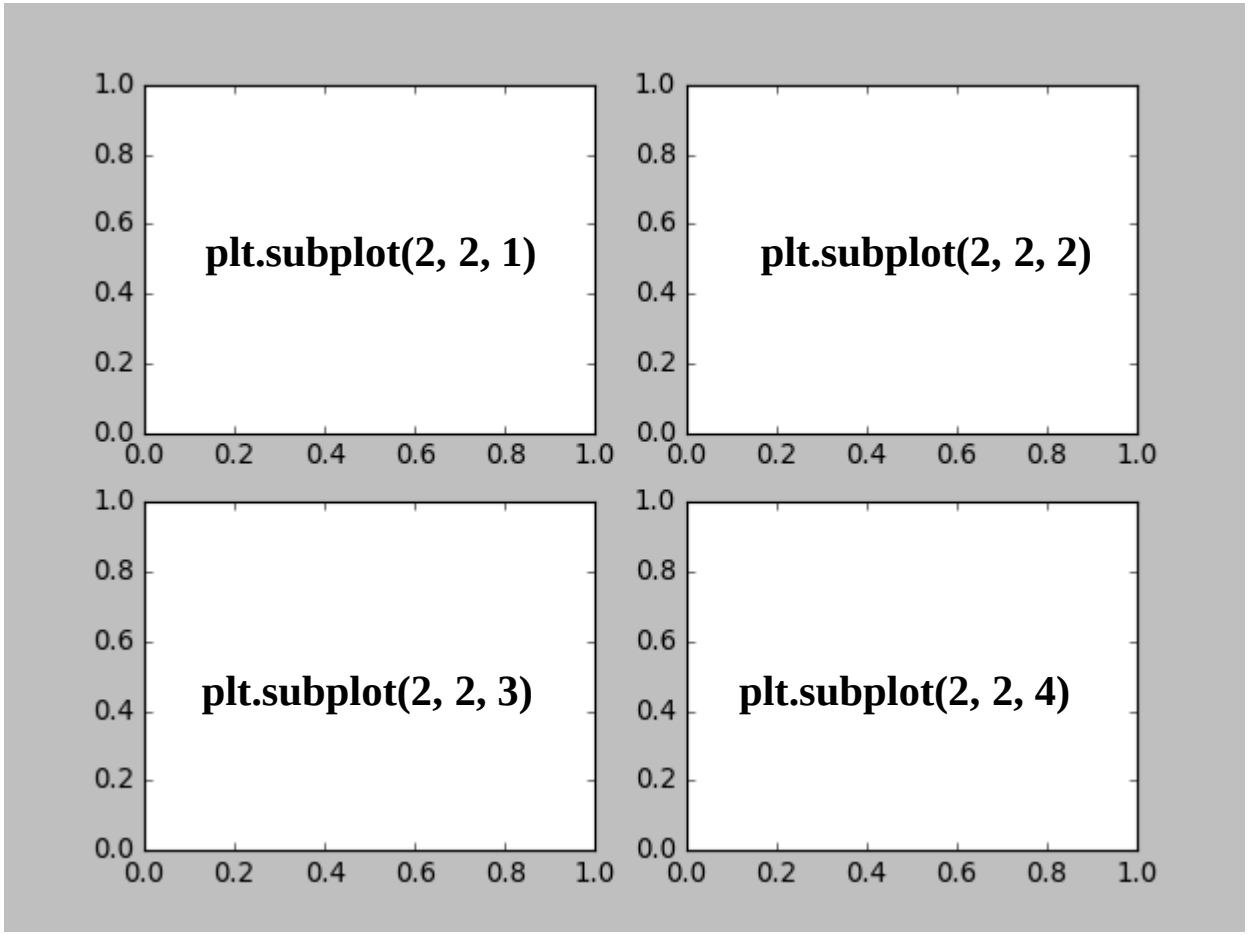
- Object oriented:

```
# Create an array of two panels (two objects)
fig, axes = plt.subplots(2)
# Call plot method on object axes[0]
axes[0].plot(x, np.log(x))
# Call plot on second axes
axes[1].plot(x, x**2)
```

- fig represents the whole figure
- axes is an array with two objects (subplots)
- axes can also be found in fig.axes

```
In [11]: axes
Out[11]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000022D8C4F6208>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x0000022D8C521B08>],
               dtype=object)
```

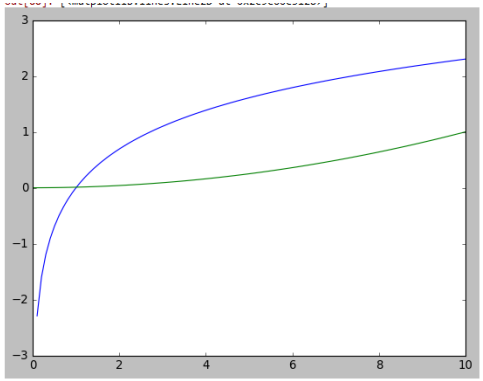
Matlab-like


plt.subplot(2, 2, 1)
plt.subplot(2, 2, 2)
plt.subplot(2, 2, 3)
plt.subplot(2, 2, 4)
0.0
0.2
0.4
0.6
0.8
1.0

Plotting multiple lines

- Matlab-like will be used for the remaining slides
- Multiple calls to plot

```
plt.figure()
plt.plot(x, np.log(x))
plt.plot(x, 0.01*x**2)
```



- Note: in this case the figure contains just one subplot, so subplot function not needed

Colors and line styles

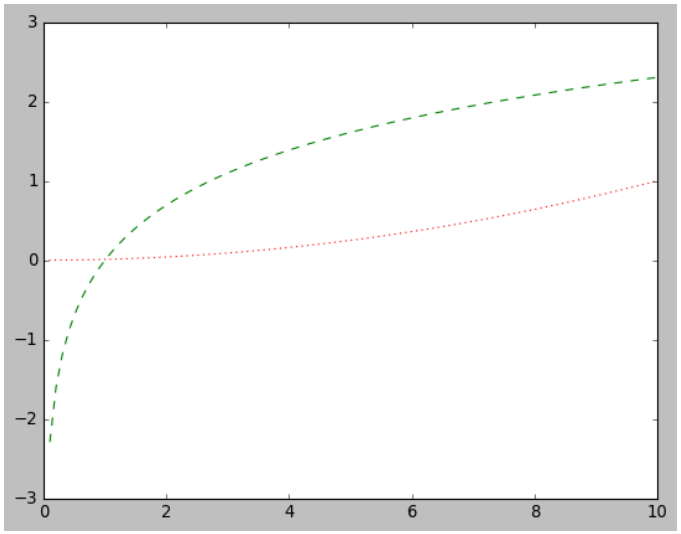
Styles

- - Solid
- -- dashed
- -. dashdot
- : dotted

Colors

- g, r, b: green, red, blue
- c, m, y, k: cyan, magenta, yellow, blac**K**

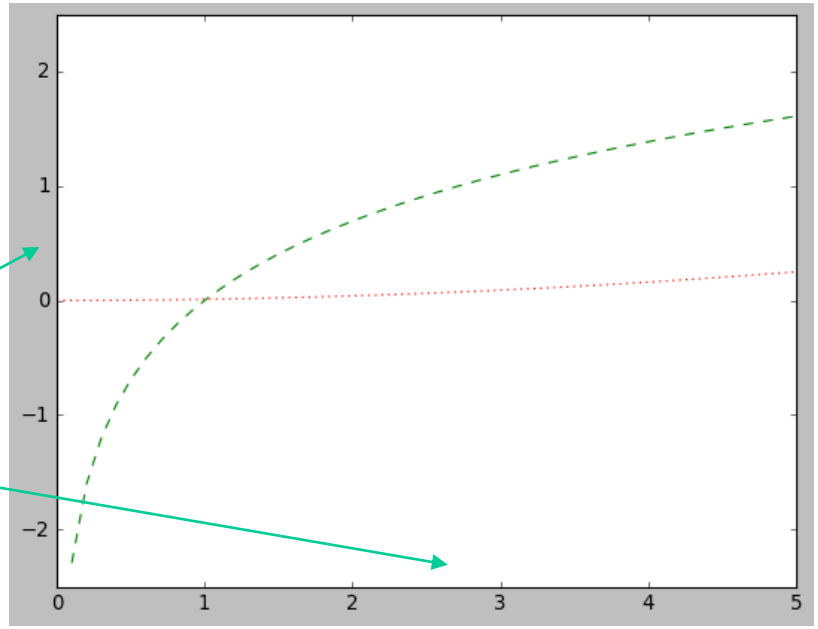
```
plt.figure()
plt.plot(x, np.log(x), "--g")
plt.plot(x, 0.01*x**2, ":r")
```



If color not specified, they change automatically every plot call

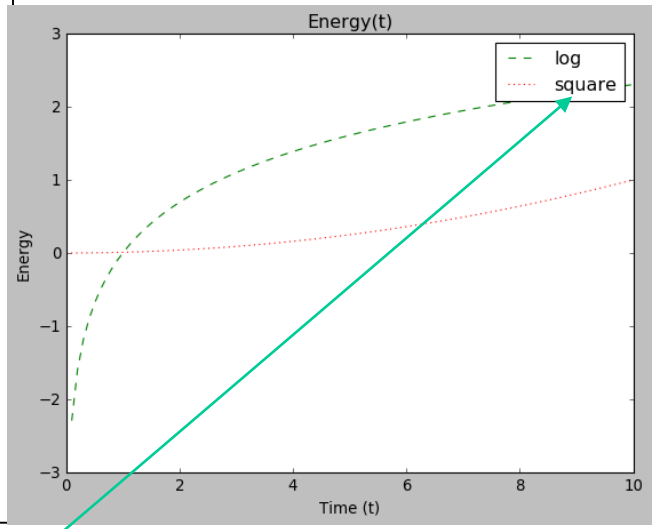
Axis limits

```
plt.figure()
plt.plot(x, np.log(x), "--g")
plt.plot(x, 0.01*x**2, ":r")
plt.xlim(0, 5)
plt.ylim(-2.5, 2.5)
```



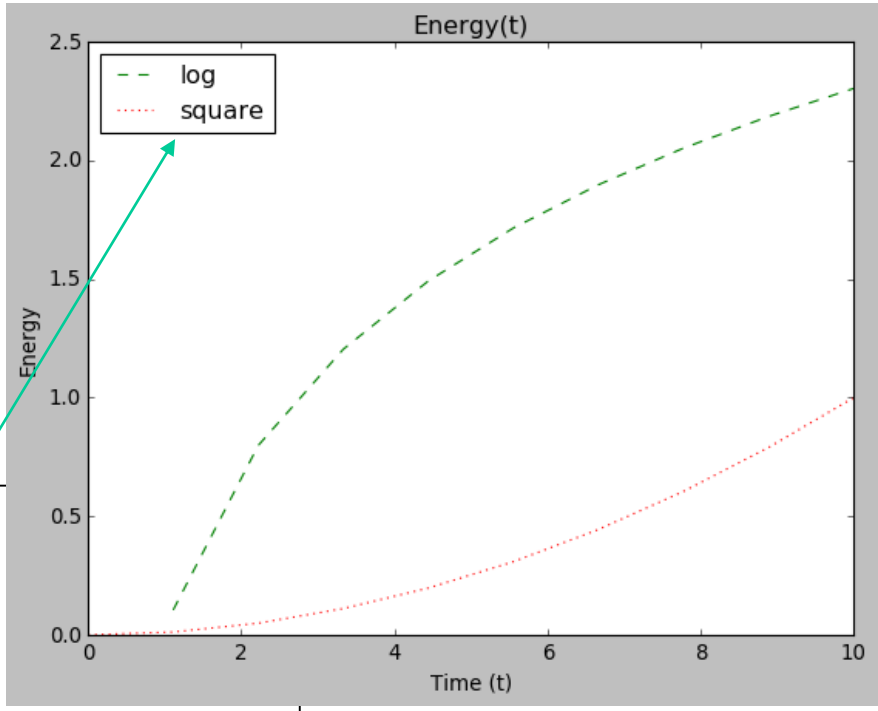
Title, axis labels, legend

```
plt.figure()
plt.title("Energy(t)")
plt.xlabel("Time (t)")
plt.ylabel("Energy")
plt.plot(x, np.log(x), "--g", label="log")
plt.plot(x, 0.01*x**2, ":r", label="square")
plt.legend()
```



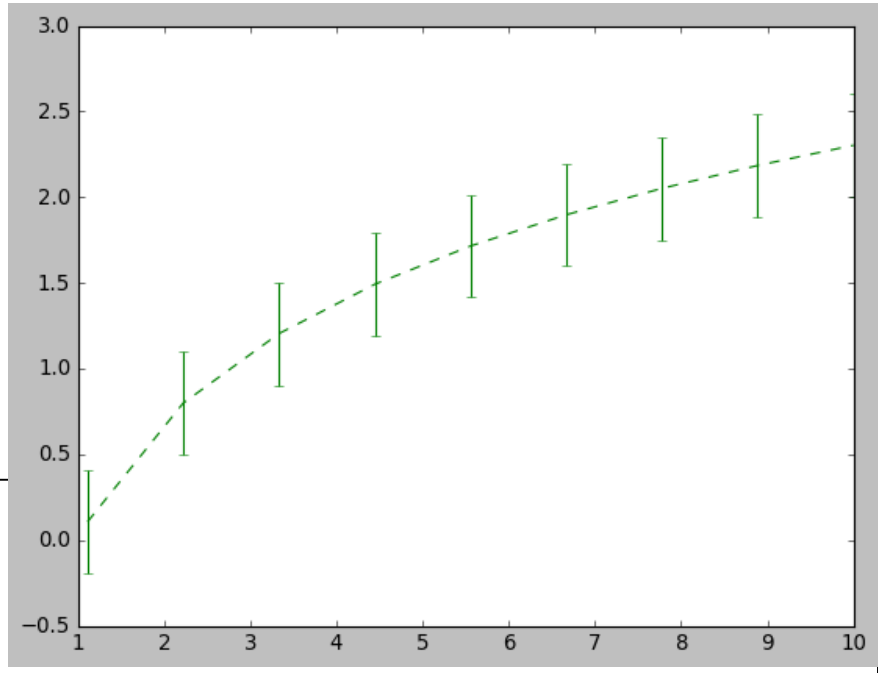
For the legend

Legend location




```
plt.figure()
plt.title("Energy(t)")
plt.xlabel("Time (t)")
plt.ylabel("Energy")
plt.plot(x, np.log(x), "--g", label="log")
plt.plot(x, 0.01*x**2, ":r", label="square")
plt.legend(loc="upper left")
```

Visualizing errors



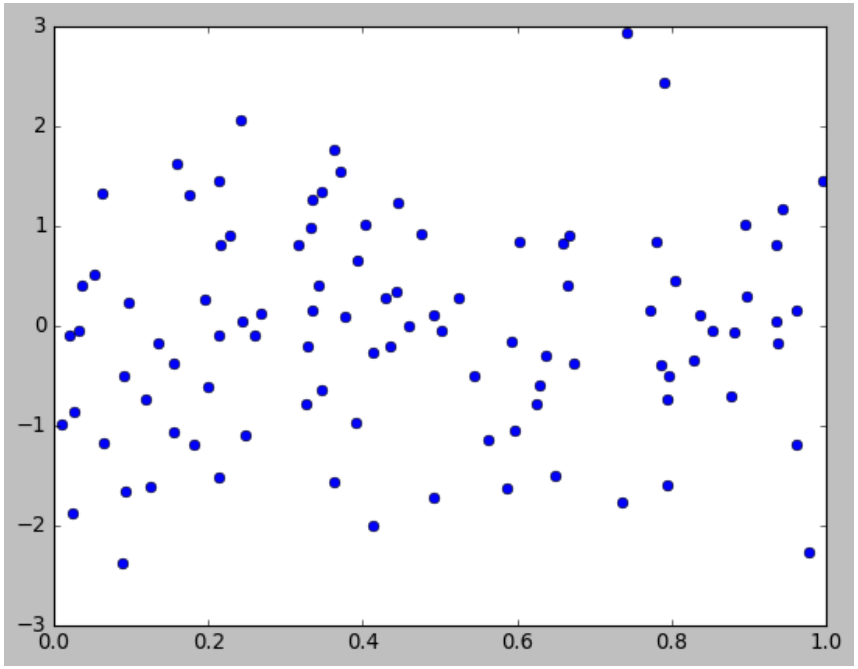

```
plt.figure()
# 10 x values
x = np.linspace(0, 10, 10)
# errors = (0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3, 0.3)
errors = np.full(10,  0.3)
plt.errorbar(x, np.log(x), yerr=errors, fmt="--g", label="log")
```

Main types of plots

- Line plots (we have already seen them)
- Scatter plots:
  - Plots points (where each point is (x,y))

Scatter plots

```
x=np.random.uniform(size=100)
y=np.random.normal(size=100)
plt.plot(x,y,'o')
```

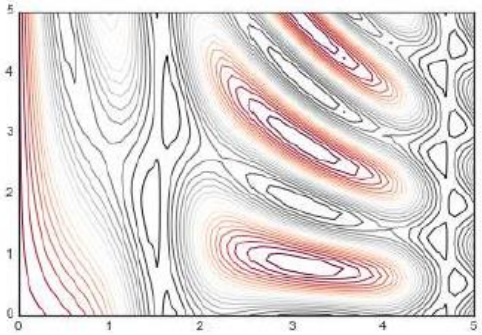


Different symbols: 'o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd'

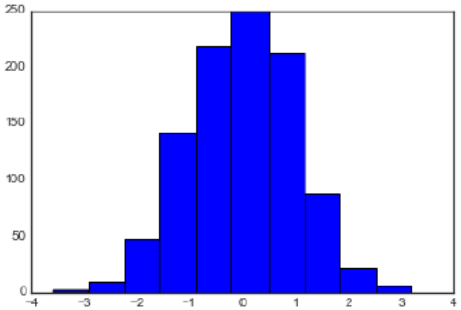
plt.scatter() allows more flexibility

Other plots

- Contour plots: *plt.contour*

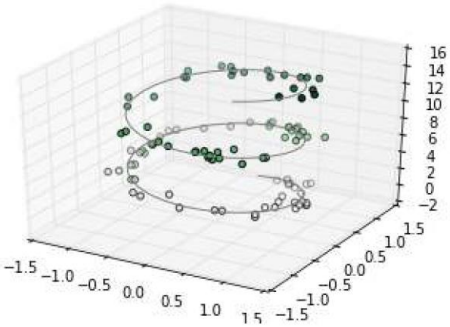


- Histograms: *plt.hist*



3D plots:

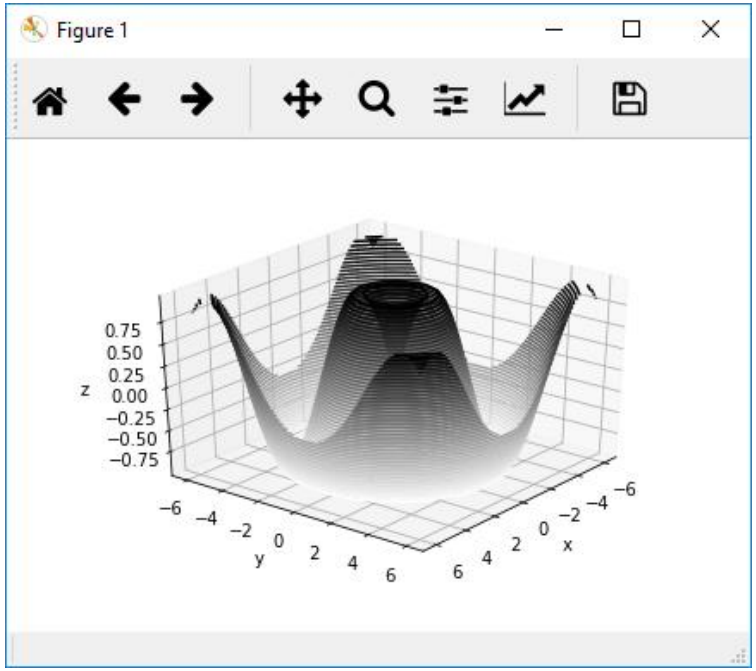
scatter3D



https://matplotlib.org/tutorials/introductory/sample_plots.html#sphx-glr-tutorials-introductory-sample-plots-py

Interactive 3D-plots

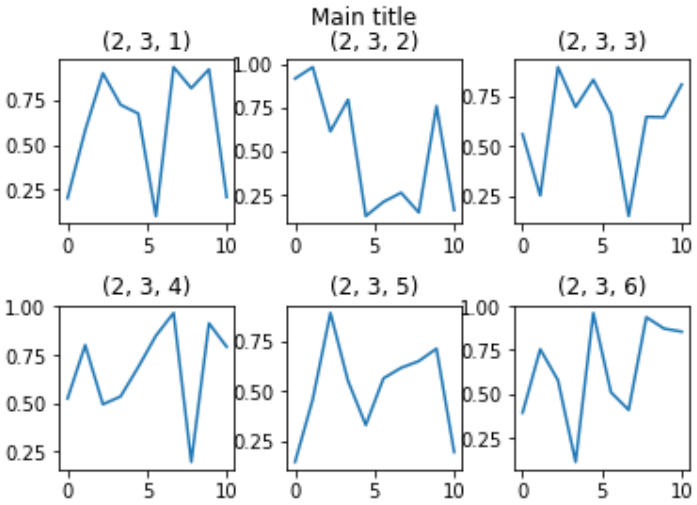
```
from mpl_toolkits.mplot3d import Axes3D
def f(x, y):
  return np.sin(np.sqrt(x ** 2 + y ** 2))
x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 50, cmap='binary')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```



Multiple subplots / axes

- Regular grid of plots
  - hspace: horizontal space
  - vspace: vertical space

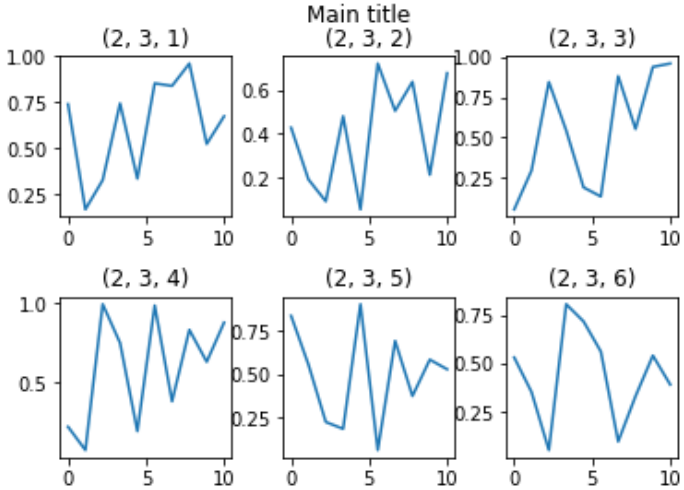
```
x = np.linspace(0, 10, 10)
fig = plt.figure()
# Here, we use fig to give global properties to
# the figure
fig.subplots_adjust(hspace=0.5, wspace=0.3)
fig.suptitle("Main title")
# Here, we create each subplot (axes)
for i in range(1, 7):
  plt.subplot(2, 3, i) # Activate subplot i
  plt.title(str((2, 3, i)))
  plt.plot(x,np.random.uniform(size=x.shape[0]))
```



Multiple subplots / axes: object oriented interface

- Regular grid of plots

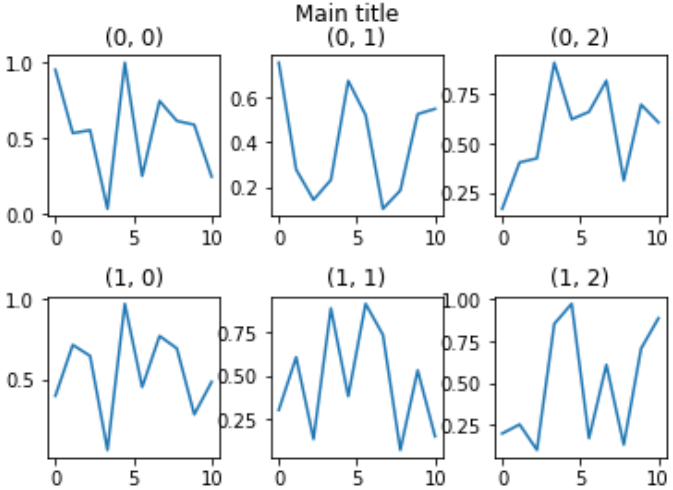
```
x = np.linspace(0, 10, 10)

fig = plt.figure()
fig.subplots_adjust(hspace=0.5, wspace=0.3)
fig.suptitle("Main title")

# Each axe (subplot) is created and then updated via methods
for i in range(1, 7):
  ax = fig.add_subplot(2,3,i)
  ax.set_title(str((2, 3, i)))
  ax.plot(x,np.random.uniform(size=x.shape[0]))
```



Multiple subplots / axes: object oriented interface

- Regular grid of plots

```
x = np.linspace(0, 10, 10)
# We can also create all subplots at the beginning
# and then update them
fig, ax = plt.subplots(2, 3)
fig.subplots_adjust(hspace=0.5, wspace=0.3)
fig.suptitle("Main title")

for i in range(0, 2):
    for j in range(0,3):
      ax[i,j].set_title(str((i, j)))
      ax[i,j].plot(x,np.random.uniform(size=x.shape[0]))
```

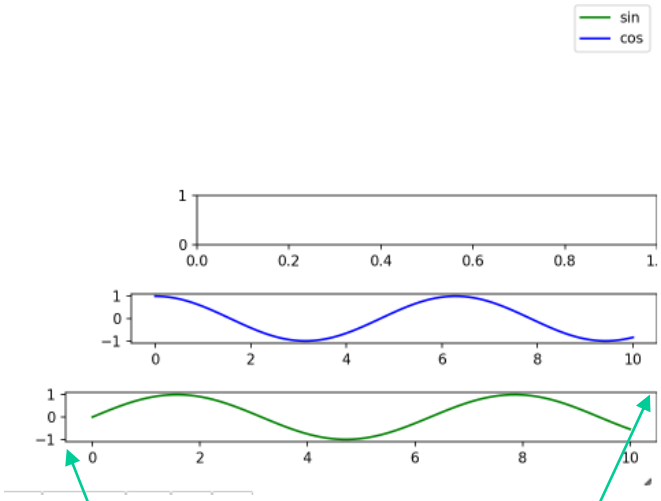


Multiple subplots / axes: object oriented interface

- Axes can be created at irregular locations

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

fig = plt.figure()

# A global coordinate system [0,1]x[0,1] is assumed
fig.add_axes([0.1, 0.1, 0.9, 0.1])
fig.add_axes([0.2, 0.3, 0.8, 0.1])
fig.add_axes([0.3, 0.5, 0.7, 0.1])

# fig.axes is an array with all axes in the figure
fig.axes[0].plot(x,y1, "g", label="sin")
fig.axes[1].plot(x,y2, "b", label="cos")

fig.legend()
```

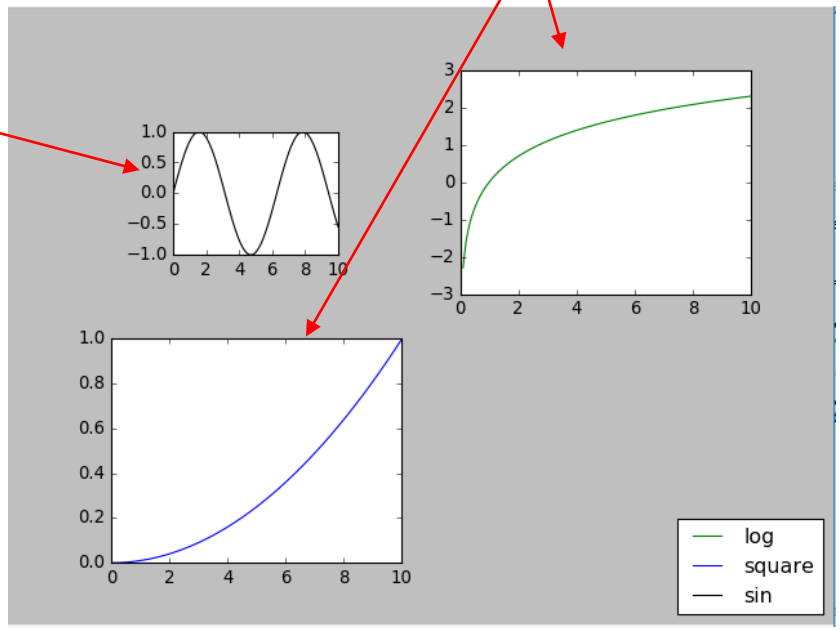
```
# We can check the axes inside a figure
fig.axes

Out[62]:
[<matplotlib.axes._axes.Axes at 0x1c9640c7048>,
 <matplotlib.axes._axes.Axes at 0x1c96349fa88>,
 <matplotlib.axes._axes.Axes at 0x1c9634d8a88>]
```



Exercise

- Try to do the following plot
- You need to use subplot(2,2,x) to create these two
- and fig.add_axes

```
x = np.linspace(0, 10, 100)
ylog = np.log(x)
ysq = 0.01*x**2
ysin = np.sin(x)

# Create an (empty) figure
fig = plt.figure()

YOUR CODE HERE

fig.legend(loc="lower right")
```



Solution: mixing Matlab and object-oriented

```
x = np.linspace(0, 10, 100)
ylog = np.log(x)
ysq = 0.01*x**2
ysin = np.sin(x)

# Create an (empty) figure
fig = plt.figure()

# Only the second and third subplots are used
plt.subplot(2, 2, 2)
plt.plot(x, ylog , "g", label="log")
plt.subplot(2, 2, 3)
plt.plot(x, ysq, "b", label="square")

# Check what axes are inside the figure
fig.axes
# Now créate a third axes (3'rd)
fig.add_axes([0.2, 0.6, 0.2, 0.2])
fig.axes[2].plot(x, ysin, "k", label="sin")

fig.legend(loc="lower right")
```

Second solution: object-oriented all the way

```
x = np.linspace(0, 10, 100)
ylog = np.log(x)
ysq = 0.01*x**2
ysin = np.sin(x)

# Create an (empty) figure
fig = plt.figure()

# Second
fig.add_subplot(2,2,2)
fig.axes
fig.axes[0].plot(x, ylog , "g", label="log")
# Third
fig.add_subplot(2,2,3)
fig.axes[1].plot(x, ysq, "b", label="square")

# Now create another subplot/axes
fig.add_axes([0.2, 0.6, 0.2, 0.2])
fig.axes
fig.axes[2].plot(x, ysin, "k", label="sin")

fig.legend(loc="lower right")
```

Visualization with Seaborn

- It works with Pandas dataframes
- Graphics tend to look better
- Complex graphics require less coding
- https://seaborn.pydata.org/
- Install last version (0.9)
  - Check version with:
    - import seaborn as sns
    - sns.__version__
  - In conda, remove current versión:
    - conda remove seaborn
    - conda install seaborn

Visualization with Seaborn

```
import seaborn as sns
tips = sns.load_dataset("tips")
tips.head()
```

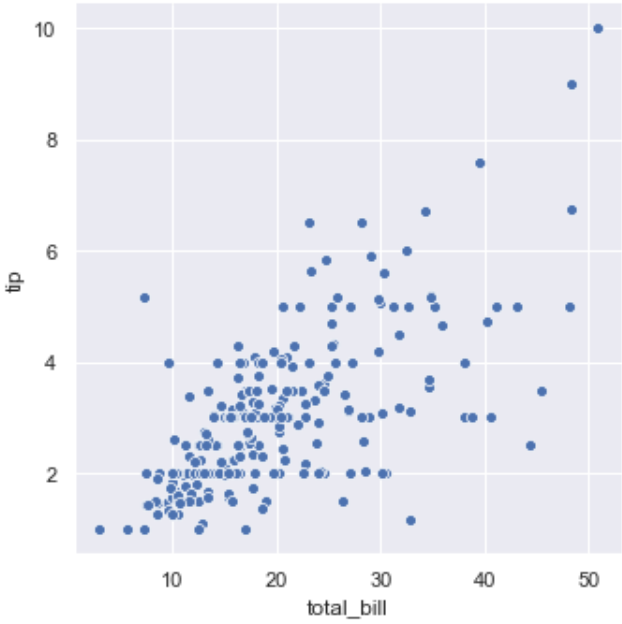
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

import seaborn as sns
tips = sns.load_dataset('tips')
tips.head()

relplot: relating two variables

- Similar to a scatterplot: tip vs. total_bill

```
# Initialization
sns.set()
sns.relplot(x="total_bill", y="tip", data=tips)
```



- Notes:
  - all data in a single dataframe
  - variables are specified by name
  - x-label, y-label: automatic

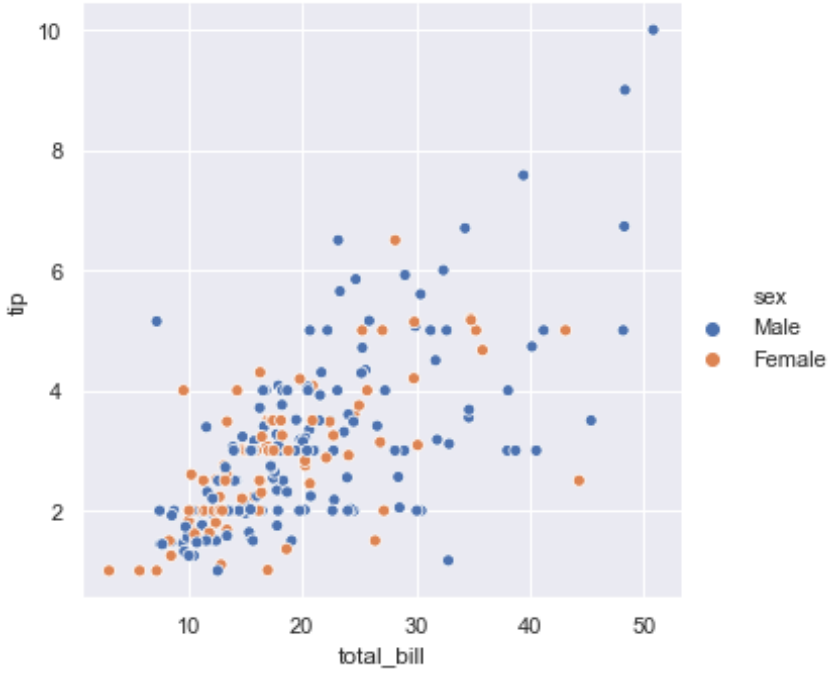
relplot: relating two variables

- Two data series (sex = male / female) in the same plot:

```
sns.relplot(x="total_bill", y="tip", hue="sex", data=tips)
```

- Notes:
  - Colors chosen automatically
  - The legend is automatic

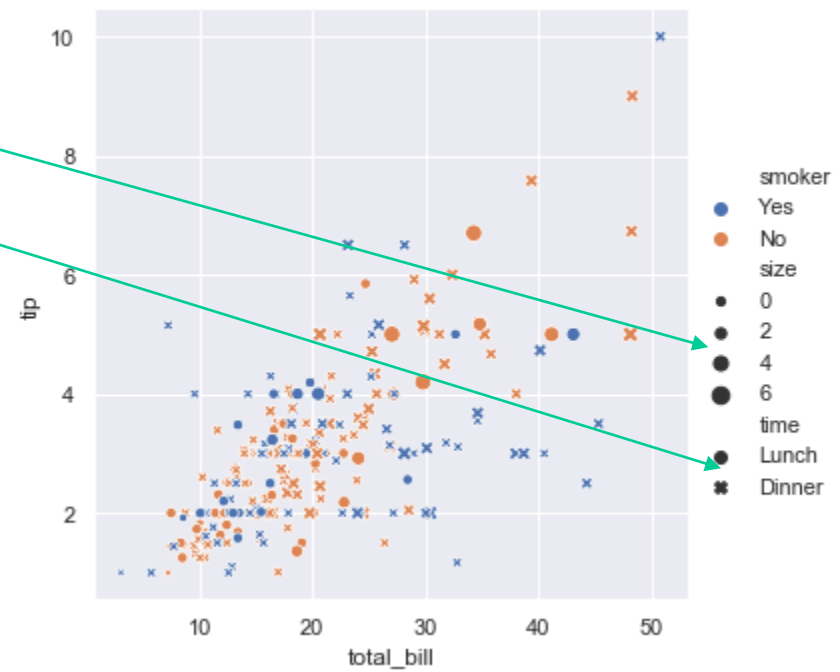
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4



relplot: relating two variables

- Aspect of points can be controlled by several properties (in addition to hue):

```
sns.relplot(x="total_bill", y="tip",
hue="smoker", style="time", size="size",
data=tips)
```

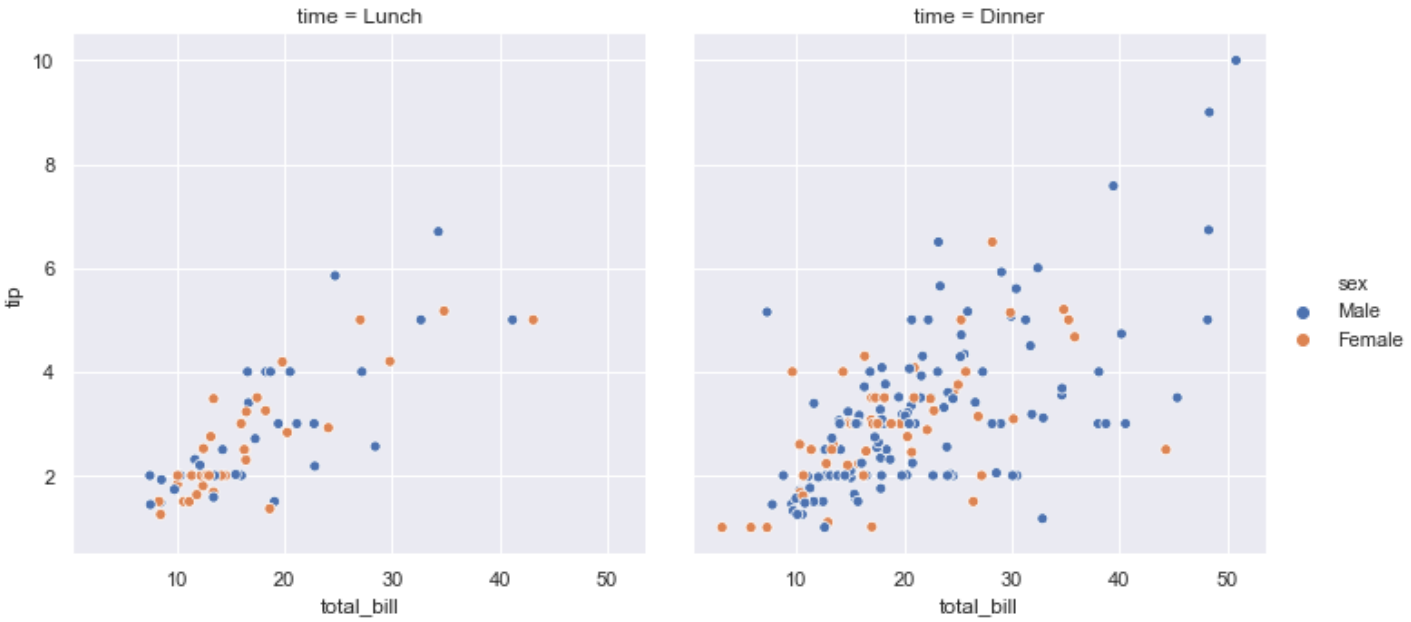


- Hue = color
- Style = shape
- Size = size
- Notes:
  - automatic legend

relplot: relating two variables

- Faceted plots (i.e. subplots) can be done easily

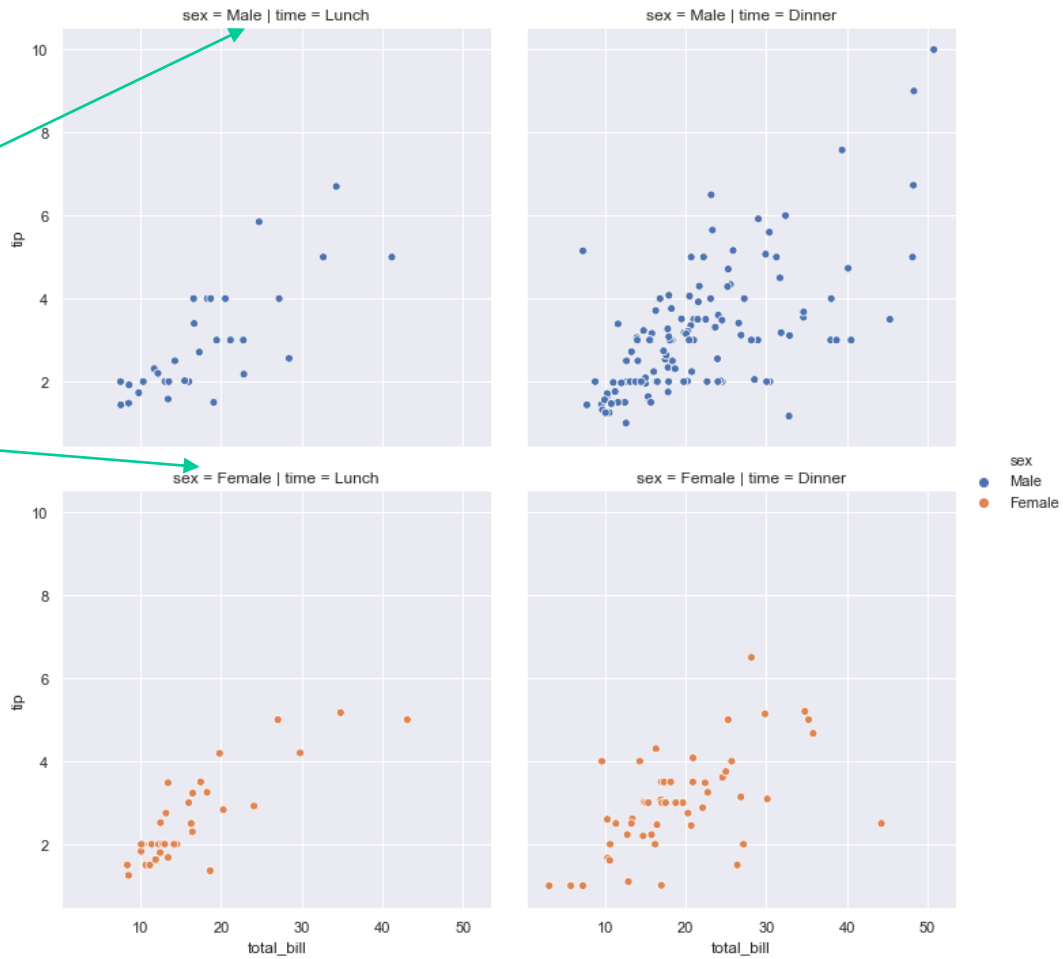
```
sns.relplot(x="total_bill", y="tip", hue="sex", col="time", data=tips)
```



relplot: relating two variables

- **Faceted** plots can be done easily

```
sns.relplot(x="total_bill", y="tip",
hue="sex",
col="time",
row="sex",
data=tips)
```



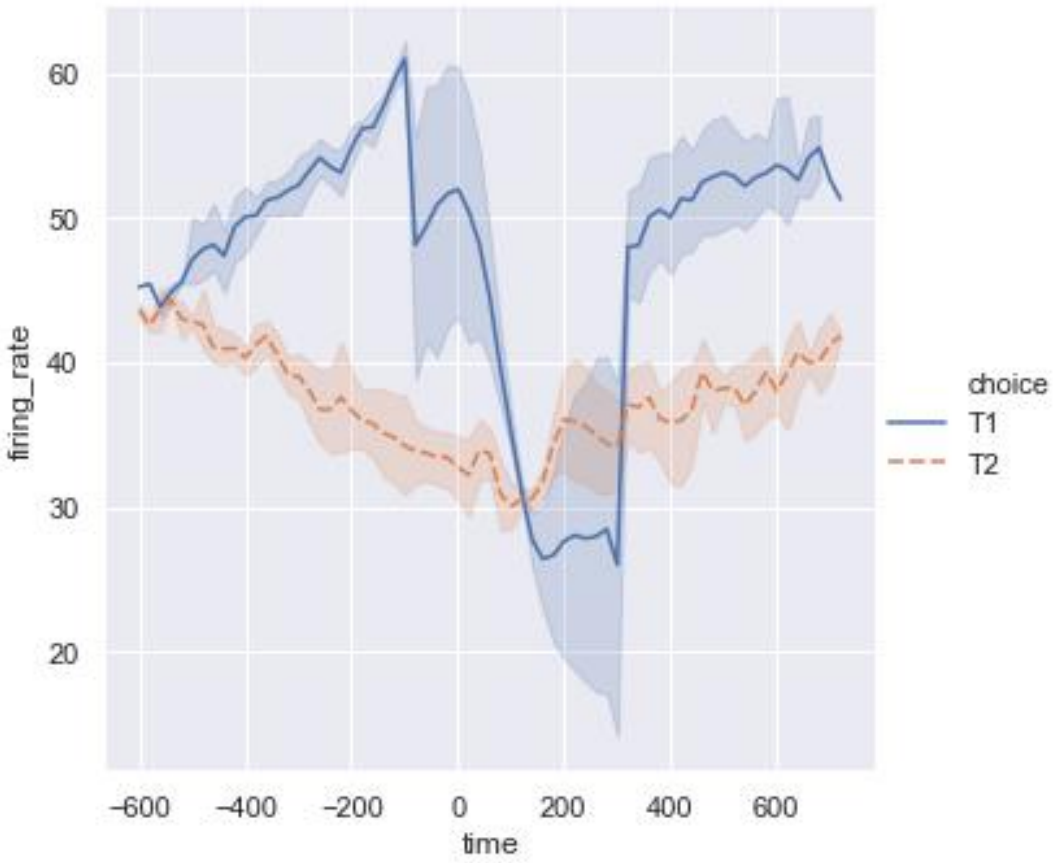
- Notes:
- Scales in all plots are the same
- (except if *facet_kws=dict(sharex=False)*)

relplot: continuous x-axis

sns.relplot(x="time", y="firing_rate", hue="choice", style="choice", **kind="line",** data=dots)

```
dots = sns.load_dataset("dots")
dots.head()
```

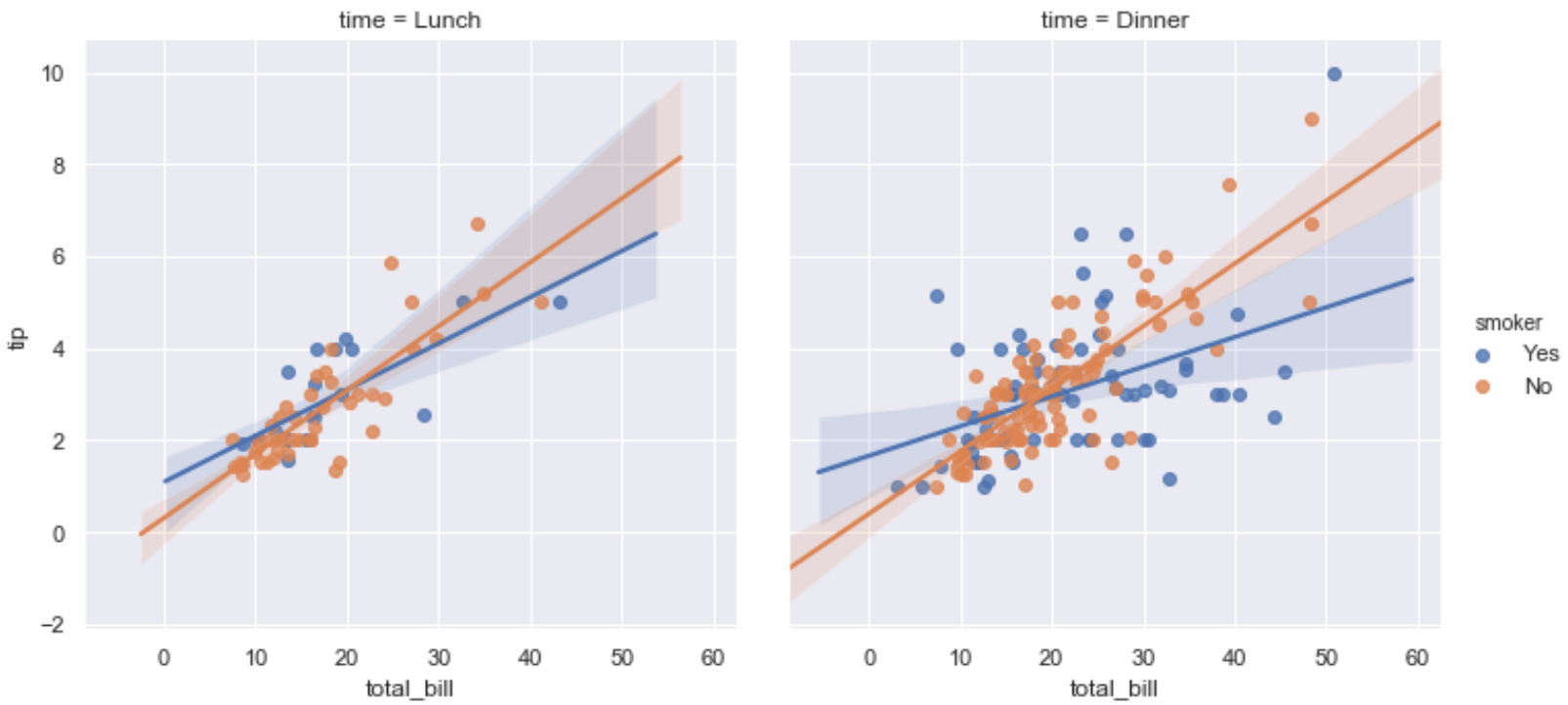
	align	choice	time	coherence	firing_rate
0	dots	T1	-80	0.0	33.189967
1	dots	T1	-80	3.2	31.691726
2	dots	T1	-80	6.4	34.279840
3	dots	T1	-80	12.8	32.631874
4	dots	T1	-80	25.6	35.060487



lmplot: plot with linear models

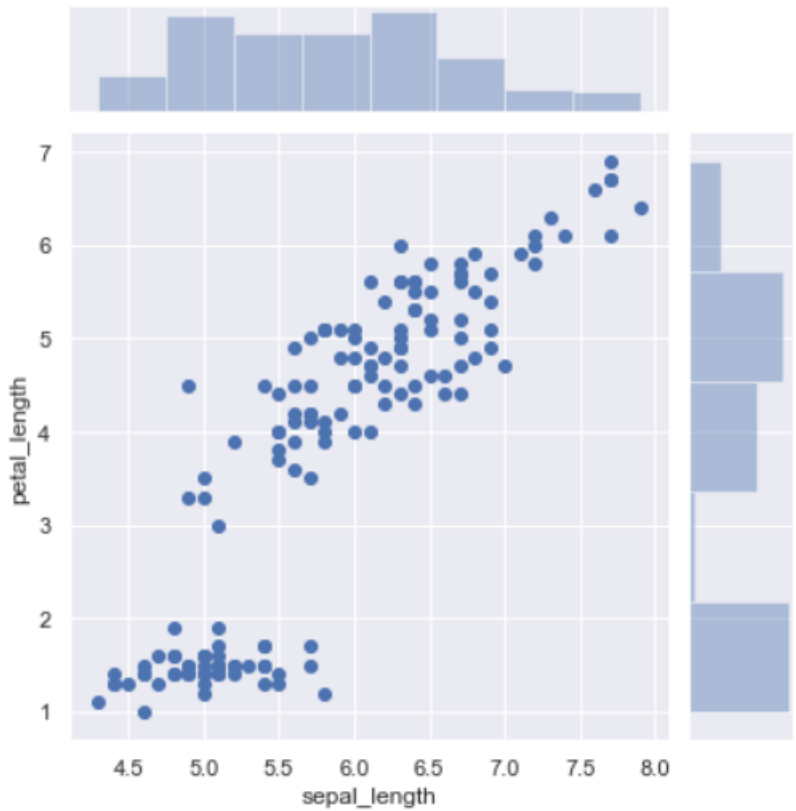
- Useful for displaying linear trends

```
sns.lmplot(x="total_bill", y="tip", col="time", hue="smoker", data=tips)
```



Joint plots

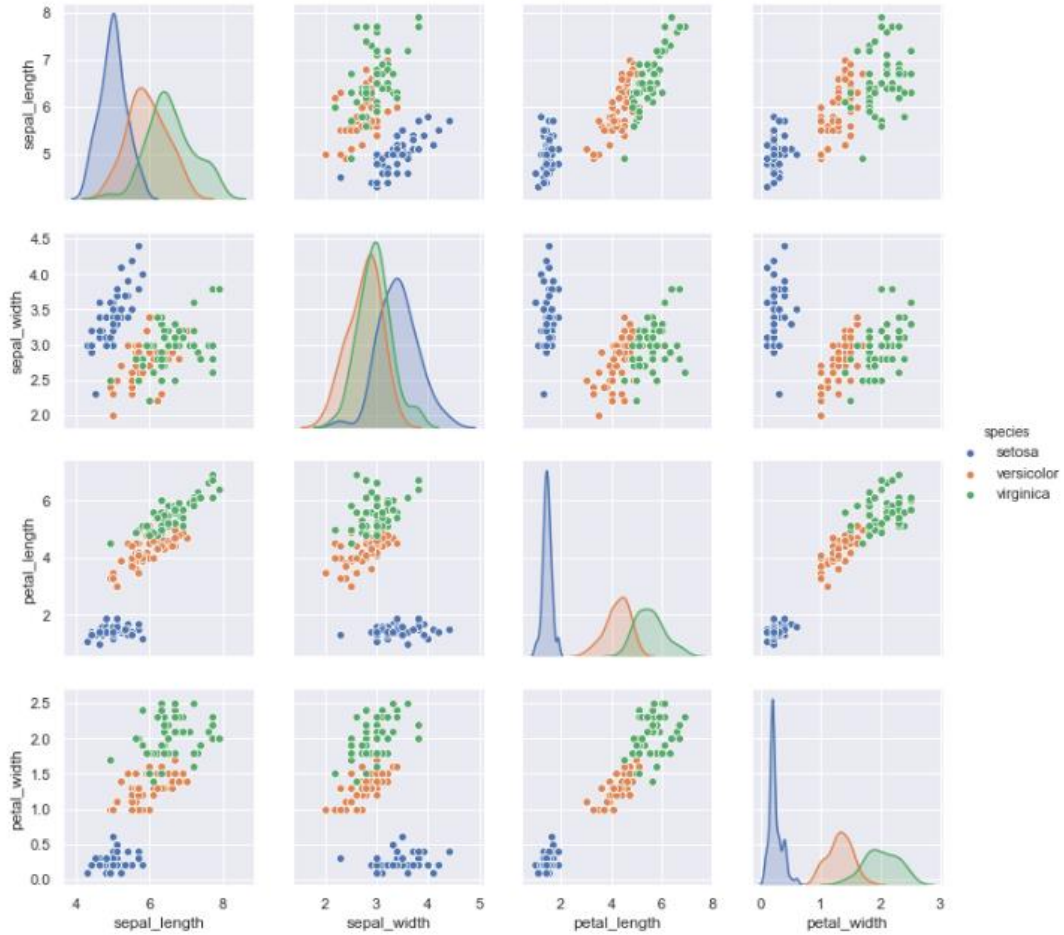
```
iris = sns.load_dataset("iris")
sns.jointplot(x="sepal_length", y="petal_length", data=iris);
```



Relation between two variables

Pair plots

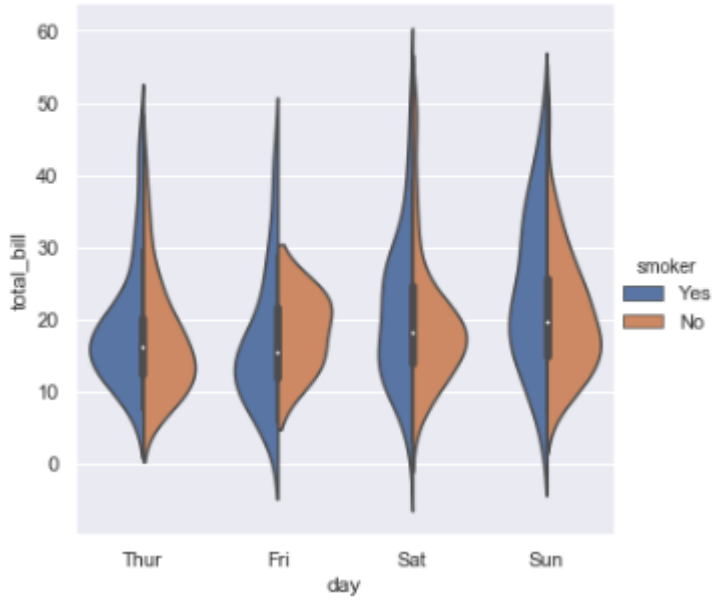
```
sns.pairplot(data=iris, hue="species");
```



Relation between all pairs of variables

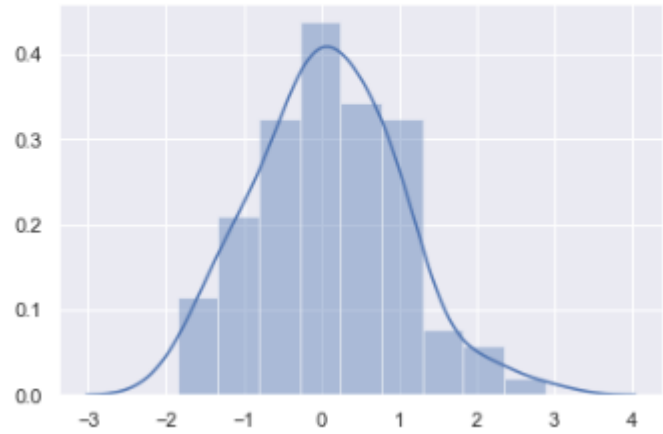
Other plots

```
sns.catplot(x="day", y="total_bill", hue="smoker",
            kind="violin", split=True, data=tips);
```



Day is categorical

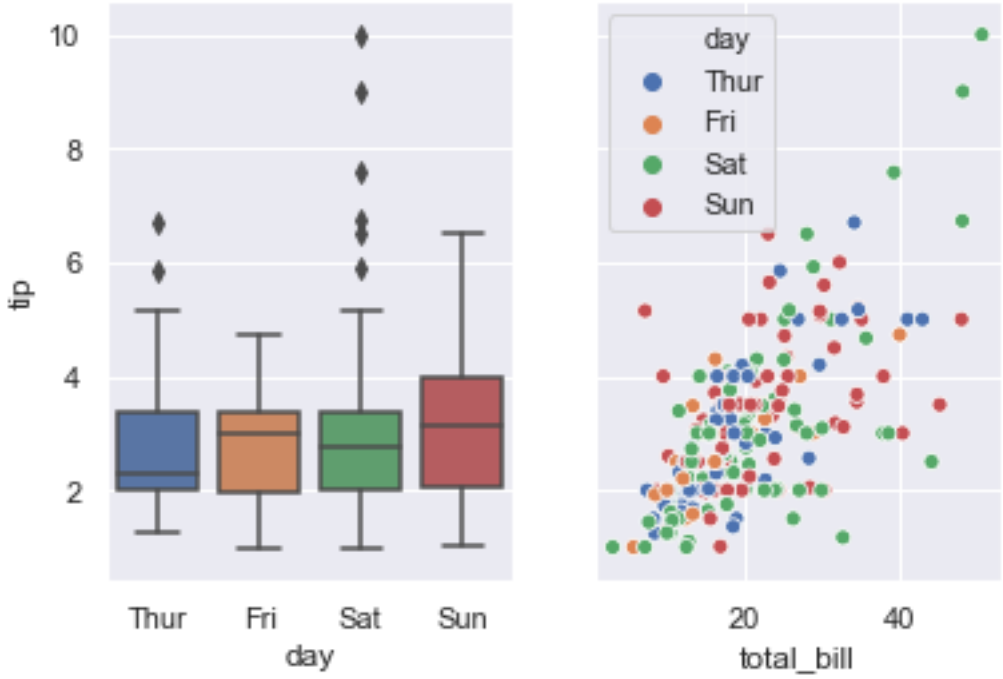
```
x = np.random.normal(size=100)
sns.distplot(x);
```



Histogram / density

Combining different plots using axes-level functions

```
f, axes = plt.subplots(1, 2, sharey=True)
sns.boxplot(x="day", y="tip", data=tips, ax=axes[0])
sns.scatterplot(x="total_bill", y="tip", hue="day",data=tips, ax=axes[1])
```



Organizing datasets: long-form or tidy data

- Seaborn prefers data in long-form
- Long-form:
  - Each observation is a row
  - Each variable is a column:
    - From the point of view of Seaborn, a variable is something that is going to have a role in the plot

Organizing datasets: long-form or tidy data

- Let's suppose that we have the classification results from three machine learning methods along several iterations.
- Let's suppose that we want to plot three lines, one per method
- This dataset is **not in long-format**. It is in **wide-format** because the variable we want to plot (the method) is not in the table.

	Iteration	Method1	Method2	Method3
0	0	0.978094	0.580535	0.267976
1	1	0.078923	0.287646	0.282177
2	2	0.774219	0.908143	0.414082
3	3	0.427651	0.720817	0.305179
4	4	0.462860	0.051145	0.798201
5	5	0.582603	0.065501	0.453163
6	6	0.744782	0.448524	0.499329
7	7	0.620550	0.993869	0.267858
8	8	0.308720	0.855421	0.037727
9	9	0.981836	0.568189	0.238975

From wide-format to long-format

Wide

	Iteration	Method1	Method2	Method3
0	0	0.978094	0.580535	0.267976
1	1	0.078923	0.287646	0.282177
2	2	0.774219	0.908143	0.414082
3	3	0.427651	0.720817	0.305179
4	4	0.462860	0.051145	0.798201
5	5	0.582603	0.065501	0.453163
6	6	0.744782	0.448524	0.499329
7	7	0.620550	0.993869	0.267858
8	8	0.308720	0.855421	0.037727
9	9	0.981836	0.568189	0.238975

Long

	Iteration	Method_ID	Error
0	0	Method1	0.978094
1	1	Method1	0.078923
2	2	Method1	0.774219
3	3	Method1	0.427651
4	4	Method1	0.462860
5	5	Method1	0.582603
6	6	Method1	0.744782
7	7	Method1	0.620550
8	8	Method1	0.308720
9	9	Method1	0.981836
10	0	Method2	0.580535
11	1	Method2	0.287646
12	2	Method2	0.908143
13	3	Method2	0.720817
14	4	Method2	0.051145
15	5	Method2	0.065501
16	6	Method2	0.448524
17	7	Method2	0.993869
18	8	Method2	0.855421
19	9	Method2	0.568189
20	0	Method3	0.267976
21	1	Method3	0.282177
22	2	Method3	0.414082
23	3	Method3	0.305179
24	4	Method3	0.798201
25	5	Method3	0.453163
26	6	Method3	0.499329
27	7	Method3	0.267858
28	8	Method3	0.037727
29	9	Method3	0.238975

- Some variables are kept, they are known as id_vars:
  - iteration
- var_name and value_name are the names of the two new columns (Method_ID, Error)
- value_vars are the variables that will be put together in the same column (Method1, Method2, Method3, in this case)

From wide-format to long-format: melt

```
data_long = pd.melt(data,
    id_vars=['Iteration'],
    value_vars=["Method1", "Method2", "Method3"],
    var_name="Method_ID", value_name="Error")
```

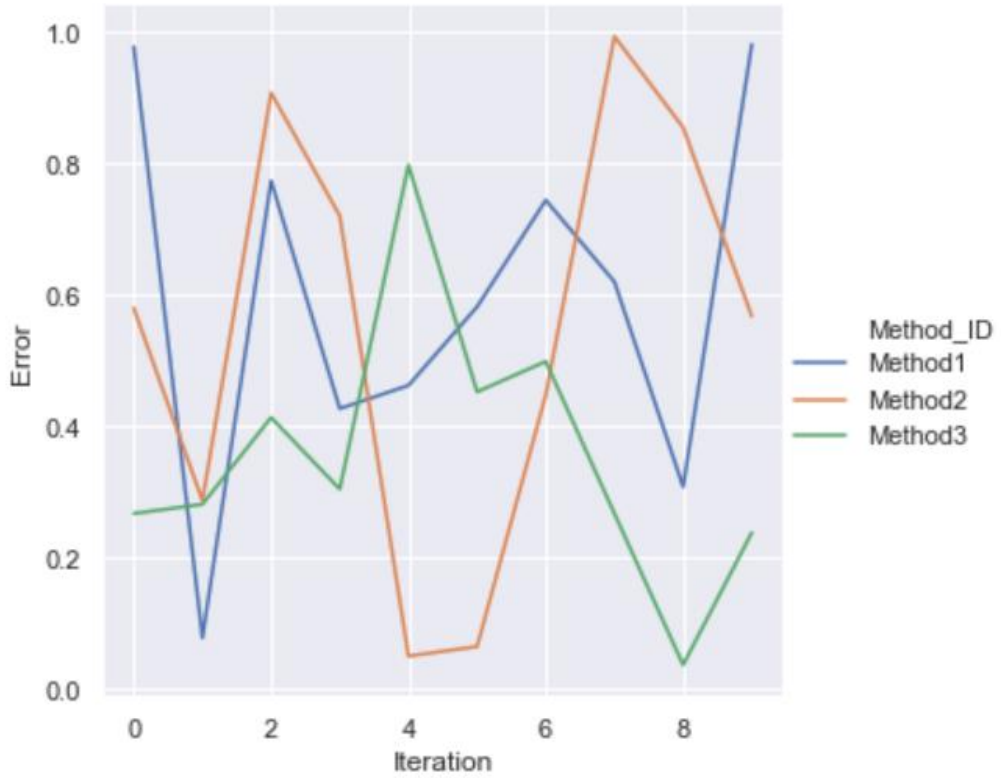
- id_vars = original variables that are kept
- value_vars = the new variable will have those values
- var_name = the name of the new variable
- value_name = the name of the column with values

	Iteration	Method_ID	Error
0	0	Method1	0.978094
1	1	Method1	0.078923
2	2	Method1	0.774219
3	3	Method1	0.427651
4	4	Method1	0.462860
5	5	Method1	0.582603
6	6	Method1	0.744782
7	7	Method1	0.620550
8	8	Method1	0.308720
9	9	Method1	0.981836
10	0	Method2	0.580535
11	1	Method2	0.287646
12	2	Method2	0.908143
13	3	Method2	0.720817
14	4	Method2	0.051145
15	5	Method2	0.065501
16	6	Method2	0.448524
17	7	Method2	0.993869
18	8	Method2	0.855421
19	9	Method2	0.568189
20	0	Method3	0.267976
21	1	Method3	0.282177
22	2	Method3	0.414082
23	3	Method3	0.305179
24	4	Method3	0.798201
25	5	Method3	0.453163
26	6	Method3	0.499329
27	7	Method3	0.267858
28	8	Method3	0.037727
29	9	Method3	0.238975

Seaborn plot in long-format

```
import seaborn as sns
sns.set()
sns.relplot(x="Iteration", y="Error", hue="Method_ID", kind="line", data_long)
```

```
<seaborn.axisgrid.FacetGrid at 0x18aed1e6128>
```



From long-format to wide-format: pivot

```
data_long.pivot(index="Iteration", columns="Method_ID", values="Error")
```

Method_ID	Method1	Method2	Method3
Iteration			
0	0.978094	0.580535	0.267976
1	0.078923	0.287646	0.282177
2	0.774219	0.908143	0.414082
3	0.427651	0.720817	0.305179
4	0.462860	0.051145	0.798201
5	0.582603	0.065501	0.453163
6	0.744782	0.448524	0.499329
7	0.620550	0.993869	0.267858
8	0.308720	0.855421	0.037727
9	0.981836	0.568189	0.238975

- index = identifier
- columns = variable that contains the values that will become columns
- values = contains the values

Exercise: from wide to long

- Creation of data

```
import pandas as pd
d = {}
d['pid'] = ['1', '1', '1', '1', '1', '1', '2', '2', '2', '2']
d['visit'] = ['1', '1', '2', '2', '3', '3', '1', '1', '2', '2']
d['stim'] = ['cmv', 'hiv', 'cmv', 'hiv', 'cmv', 'hiv', 'cmv', 'hiv',
'cmv', 'hiv']
d['tnf'] = [1.0, 2.0, 1.1, 2.1, 1.2, 2.2, 3, 4, 3.1, 4.1]
d['ifn'] = [11.0, 12.0, 11.1, 12.1, 11.2, 12.2, 13, 14, 13.1, 14.1]
d['il2'] = [0.0, 0.0, 0.1, 0.1, 0.2, 0.2, 0.1, 0.3, 0.1, 0.1]
df = pd.DataFrame(d)
```

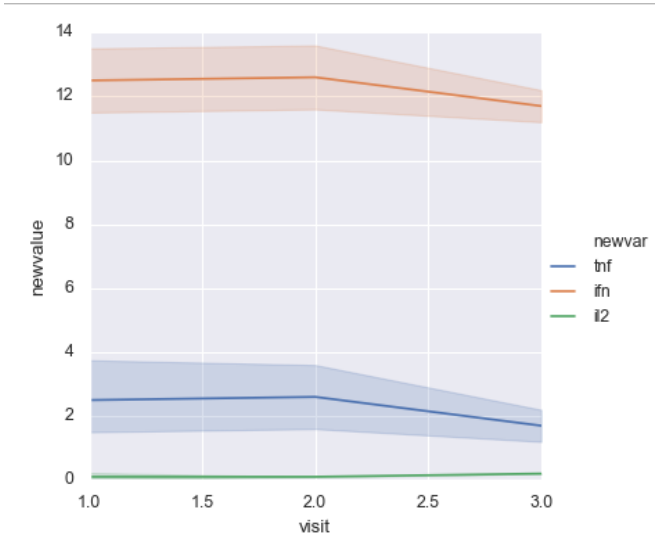
	pid	visit	stim	tnf	ifn	il2
0	1	1	cmv	1.0	11.0	0.0
1	1	1	hiv	2.0	12.0	0.0
2	1	2	cmv	1.1	11.1	0.1
3	1	2	hiv	2.1	12.1	0.1
4	1	3	cmv	1.2	11.2	0.2
5	1	3	hiv	2.2	12.2	0.2
6	2	1	cmv	3.0	13.0	0.1
7	2	1	hiv	4.0	14.0	0.3
8	2	2	cmv	3.1	13.1	0.1
9	2	2	hiv	4.1	14.1	0.1

Exercise: from wide to long

	pid	visit	stim	tnf	ifn	il2
0	1	1	cmv	1.0	11.0	0.0
1	1	1	hiv	2.0	12.0	0.0
2	1	2	cmv	1.1	11.1	0.1
3	1	2	hiv	2.1	12.1	0.1
4	1	3	cmv	1.2	11.2	0.2
5	1	3	hiv	2.2	12.2	0.2
6	2	1	cmv	3.0	13.0	0.1
7	2	1	hiv	4.0	14.0	0.3
8	2	2	cmv	3.1	13.1	0.1
9	2	2	hiv	4.1	14.1	0.1

	pid	stim	visit	newvar	newvalue
0	1	cmv	1	tnf	1.0
1	1	hiv	1	tnf	2.0
2	1	cmv	2	tnf	1.1
3	1	hiv	2	tnf	2.1
4	1	cmv	3	tnf	1.2
5	1	hiv	3	tnf	2.2
6	2	cmv	1	tnf	3.0
7	2	hiv	1	tnf	4.0
8	2	cmv	2	tnf	3.1
9	2	hiv	2	tnf	4.1
10	1	cmv	1	ifn	11.0
11	1	hiv	1	ifn	12.0
12	1	cmv	2	ifn	11.1
13	1	hiv	2	ifn	12.1
14	1	cmv	3	ifn	11.2
15	1	hiv	3	ifn	12.2
16	2	cmv	1	ifn	13.0
17	2	hiv	1	ifn	14.0
18	2	cmv	2	ifn	13.1
19	2	hiv	2	ifn	14.1
20	1	cmv	1	il2	0.0
21	1	hiv	1	il2	0.0
22	1	cmv	2	il2	0.1
23	1	hiv	2	il2	0.1
24	1	cmv	3	il2	0.2
25	1	hiv	3	il2	0.2
26	2	cmv	1	il2	0.1
27	2	hiv	1	il2	0.3
28	2	cmv	2	il2	0.1
29	2	hiv	2	il2	0.1

and plot this (with replot ... kind="line")



Solution

```
long1 = pd.melt(df, id_vars =['stim', 'visit'],
                value_vars = ['tnf', 'ifn', 'il2'],
                var_name = 'newvar',
                value_name = 'newvalue'
                )
```

- Any variable that is not an id is considered a value, therefore this solution would be equivalent:

```
long1 = pd.melt(df, id_vars =['stim', 'visit'],
                var_name = 'newvar',
                value_name = 'newvalue'
                )
```

Solution

```
import seaborn as sns
sns.set()
sns.relplot(x="visit", y="newvalue", hue="newvar", kind="line", data=long1)
```