Matplotlib Notes

Computer Engineering for Scientific Computing (Technische Universiteit Delft)

Introductory Notes: Matplotlib

Preliminaries

Start by importing these Python modules

```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
```

Which Application Programming Interface?

The two worlds of Matplotlib
There are two broad ways of using matplotlib's pyplot:

1. The first (and most common) way is not pythonic. It relies on global functions to build and display a global figure using matplotlib as a global state machine. (This is easy for interactive use).

2. The second way is pythonic and object oriented and it is the best approach for programmatic use:
   a. obtain an empty Figure from a global factory, then build the plot using the methods of the Figure and Axes classes; or
   b. obtain a populated Axes (and its Figure container) using a plot method from the pandas data analysis library, which you can then pretty-up and save to file.

These notes focus on the 2a approach. For 2b, see the pandas cheat sheet (http://bit.ly/python_cs). First, let's look at using pyplot as a global state machine.

Using matplotlib as a global state machine

1. Get data – we will fake up a monthly time series

```
x = pd.period_range('1980-01-01', periods=450,
    freq='M').to_timestamp().to_pydatetime()
y = np.random.randn(len(x)).cumsum()
```

2. Plot the data

```
plt.plot(x, y, label='FDI')
```

3. Add your labels and pretty-up the plot

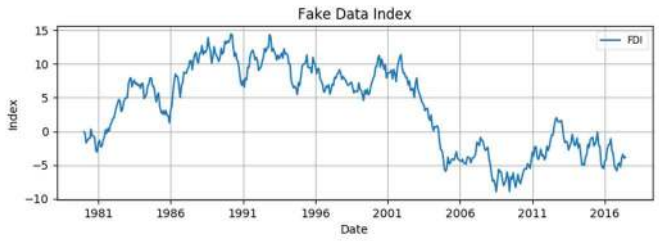
```
plt.gcf().set_size_inches(8, 3)
plt.title('Fake Data Index')
plt.xlabel('Date')
plt.ylabel('Index')
plt.grid(True)
plt.legend(loc='best', framealpha=0.5,
    prop={'size':'small'})
plt.tight_layout(pad=1)
```

4. Save the figure and close it

```
plt.savefig('filename.png')
plt.close()
```

Alternatively, show the figure
With iPython, follow steps 1 to 3 above then

```
plt.show()    # Note: also closes the figure
```



Introducing the Figure and Axes classes

The Figure
Figure is the top-level container for everything on a canvas. It was obtained from the global Figure factory.

```
fig = plt.figure(num=None, figsize=None,
      dpi=None, facecolor=None,
      edgecolor=None)
```

num – integer or string identifier of figure
    if num exists, it is selected
    if num is None, a new one is allocated
figsize – tuple of (width, height) in inches
dpi – dots per inch
facecolor – background; edgecolor – border

Iterating over the open figures

```
for i in plt.get_fignums():
    fig = plt.figure(i) # get the figure
    print (fig.number)  # do something
```

Close a figure

```
plt.close(fig.number) # close figure
plt.close()      # close the current figure
plt.close(i)     # close figure numbered i
plt.close(name)  # close figure by str name
plt.close('all') # close all figures
```

An Axes or Subplot (a subclass of Axes)
An Axes is a container class for a specific plot. A figure may contain many Axes and/or Subplots. Subplots are laid out in a grid within the Figure. Axes can be placed anywhere on the Figure. There are a number of methods that yield an Axes, including:

```
ax = fig.add_subplot(2,2,1) # row-col-num
ax = fig.add_axes([0.1,0.1,0.8,0.8])
```

All at once
Use the subplots factory to get a Figure and Axes.

```
fig, ax = plt.subplots()
fig,(ax1,ax2,ax3) = plt.subplots(nrows=3,
    ncols=1, sharex=True, figsize=(8,4))
```

Iterating the Axes within a Figure

```
for ax in fig.get_axes():
    pass # do something
```

From an Axes get its Figure

```
fig = ax.figure
```

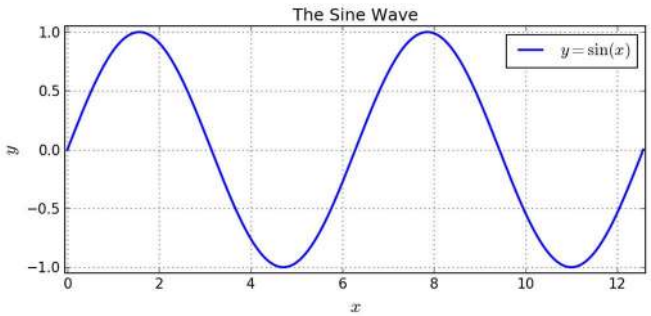
Remove an Axes from a Figure

```
fig.delaxes(ax)
```

Build a line plot with Matplotlib and raw (*x*, *y*) data

A single line plot in matplotlib

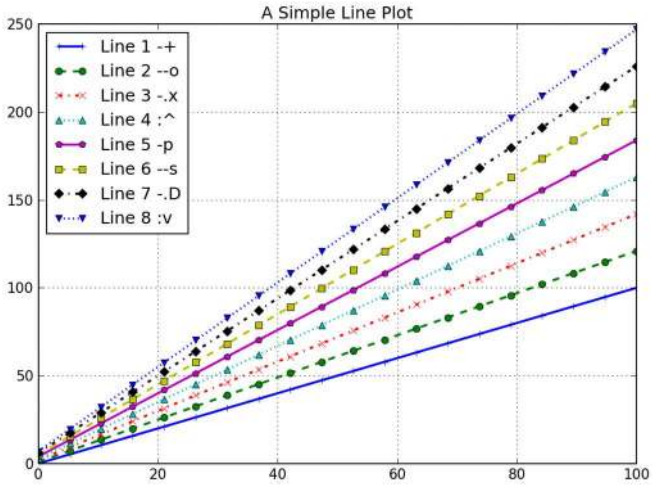
```
# --- fake up some data
x = np.linspace(0, 4*np.pi, 800)
y = np.sin(x)

# --- Select a style
plt.style.use('classic')

# --- get an empty Figure and add an Axes
fig = plt.figure(figsize=(8,4))
ax = fig.add_subplot(1,1,1) # row-col-num

# --- line plot data on the Axes
ax.plot(x, y, 'b-', linewidth=2,
    label=r'$y=\sin(x)$')

# --- add title and axis labels
ax.set_title('The Sine Wave')
ax.set_ylabel(r'$y$', fontsize=16)
ax.set_xlabel(r'$x$', fontsize=16)

# --- change the default plot limits
ax.set_xlim((-0.05, 4*np.pi + 0.05,))
ax.set_ylim((-1.05, 1.05))

# --- plot a legend in the best location
ax.legend(loc='best')

# --- add grid – not in default classic style
ax.grid(True)

# --- improve the layout
fig.tight_layout(pad=1)

# --- save and close
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Build a multi-line plot from raw data in matplotlib

A multi-line plot with markers and line-styles

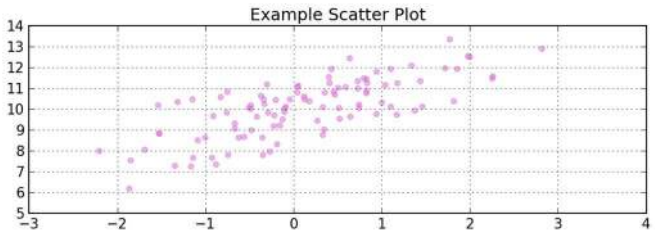
```
# --- select a style
plt.style.use('classic')

# --- get the Figure and Axes all at once
fig, ax = plt.subplots(figsize=(8,6))

# --- plot some lines
N = 8 # the number of lines we will plot
styles =  ['-', '--', '-.', ':']
markers = list('+ox^psDv')
x = np.linspace(0, 100, 20)
for i in range(N): # add line-by-line
    y = x + x/5*i + i
    s = styles[i % len(styles)]
    m = markers[i % len(markers)]
    ax.plot(x, y,
      label='Line '+str(i+1)+' '+s+m,
      marker=m, linewidth=2, linestyle=s)

# --- add grid, legend, title and save
ax.grid(True)
ax.legend(loc='best', prop={'size':'large'})
ax.set_title('A Simple Line Plot')
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Scatter plots – using matplotlib's ax.scatter()

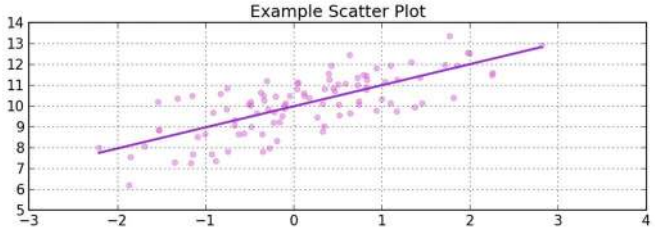
A simple scatter plot

```
x = np.random.randn(100)
y = x + np.random.randn(100) + 10
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 3))
ax.scatter(x, y, alpha=0.5, color='orchid')
ax.set_title('Example Scatter Plot')
ax.grid(True)
fig.tight_layout(pad=1)
fig.savefig('filename1.png', dpi=125)
```



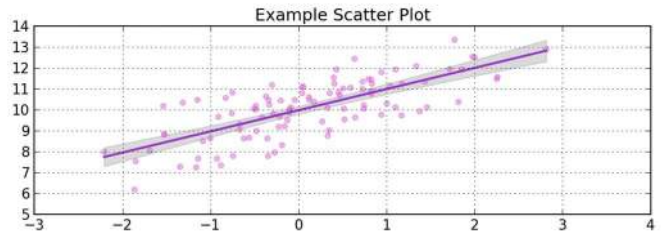
Add a regression line (using statsmodels)

```
import statsmodels.api as sm
x = sm.add_constant(x) # intercept
# Model: y ~ x + c
model = sm.OLS(y, x)
fitted = model.fit()
x_pred = np.linspace(x.min(), x.max(), 50)
x_pred2 = sm.add_constant(x_pred)
y_pred = fitted.predict(x_pred2)
ax.plot(x_pred, y_pred, '-',
    color='darkorchid', linewidth=2)
fig.savefig('filename2.png', dpi=125)
```



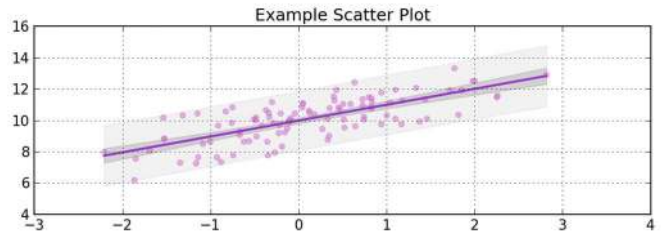
Add confidence bands for the regression line

```
y_hat = fitted.predict(x)
y_err = y - y_hat
mean_x = x.T[1].mean()
n = len(x)
dof = n - fitted.df_model - 1
from scipy import stats
t = stats.t.ppf(1-0.025, df=dof) # 2-tail
s_err = np.sum(np.power(y_err, 2))
conf = t * np.sqrt((s_err/(n-2))*(1.0/n +
    (np.power((x_pred-mean_x),2) /
    ((np.sum(np.power(x_pred,2))) -
    n*(np.power(mean_x,2))))))
upper = y_pred + abs(conf)
lower = y_pred - abs(conf)
ax.fill_between(x_pred, lower, upper,
    color='#888888', alpha=0.3)
fig.savefig('filename3.png', dpi=125)
```



Add a prediction interval for the regression line

```
from statsmodels.sandbox.regression.predstd\
    import wls_prediction_std
sdev, lower, upper =
wls_prediction_std(fitted,
    exog=x_pred2, alpha=0.05)
ax.fill_between(x_pred, lower, upper,
    color='#888888', alpha=0.1)
fig.savefig('filename4.png', dpi=125)
plt.close('all')
```

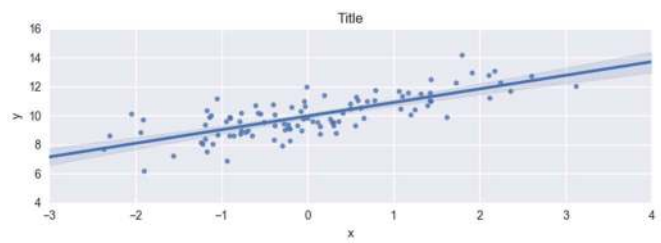


Note: The confidence interval relates to the location of the regression line. The predication interval relates to the location of data points around the regression line.

Quick scatter and regression plots using Seaborn

Use seaborn

```
import seaborn as sns

x = np.random.randn(100)
y = x + np.random.randn(100) + 10
df = DataFrame([x, y], index=['x', 'y']).T

g = sns.lmplot(x='x', y='y', data=df,
    fit_reg=True)
ax = g.axes[0, 0]
ax.set_title('Title')
fig = ax.figure
fig.set_size_inches(8, 3)
fig.tight_layout(pad=1)
fig.savefig("filename.png")
plt.close('all')
```

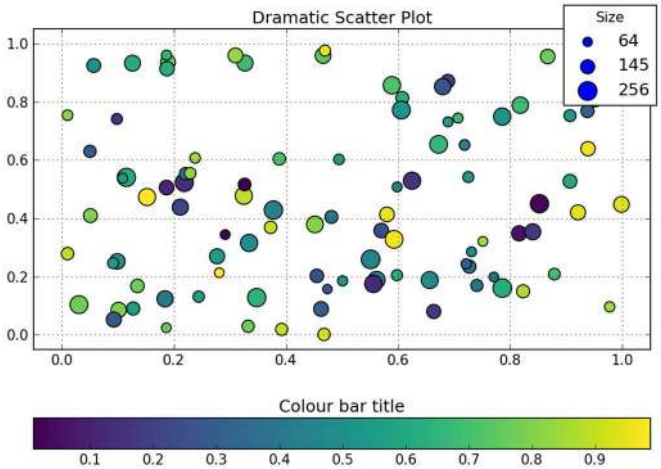


Scatter plots – more options

Changing the marker size and colour

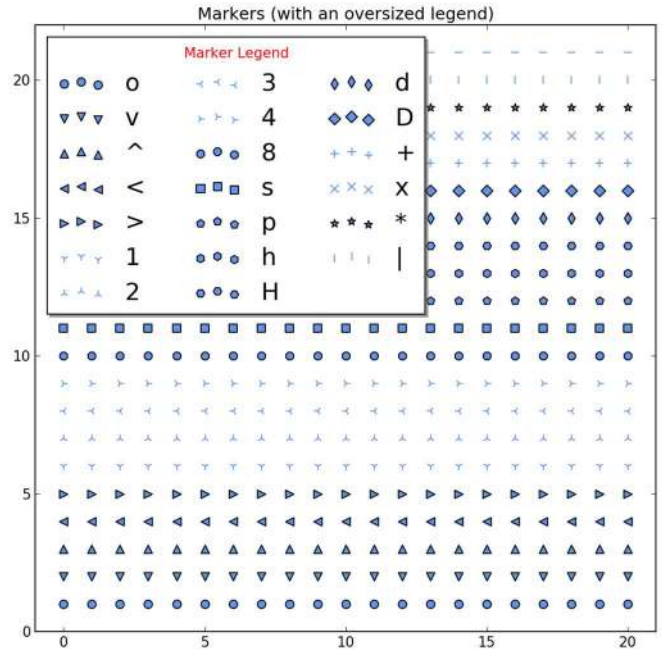
```
# --- import a colour map
import matplotlib.cm as cm

# --- get some data
N = 100
x = np.random.rand(N)
y = np.random.rand(N)
size = ((np.random.rand(N) + 1) * 8) ** 2
colour = np.random.rand(N)

# --- plot
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 6))
l = ax.scatter(x, y, s=size, c=colour,
    cmap=cm.viridis) # choose a colormap

# --- attach the colour bar
clb = fig.colorbar(l,
    orientation='horizontal')
clb.ax.set_title('Colour bar title')

# --- do the size legend by hand ...
sz = [size.min(), size.mean(), size.max()]
handles = [ax.scatter([],[],s=sz[i]) for i in
    range(len(sz))]
labels = [str(int(round(x, 0))) for x in sz]
fig.legend(handles=handles, labels=labels,
    loc='upper right', scatterpoints=1,
    title='Size')

# --- grid, plot-limits, title and save
ax.grid(True)

ax.set_xlim((-0.05, 1.05))
ax.set_ylim((-0.05, 1.05))

ax.set_title('Dramatic Scatter Plot')

fig.tight_layout(pad=1);
fig.savefig('filename6.png', dpi=125)
plt.close('all')
```

Note: there are many colormaps to choose from.



Changing the marker symbol

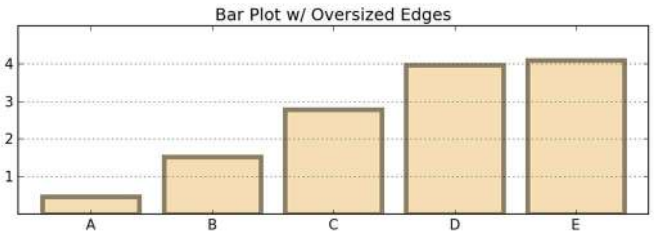
```
# --- get the Figure and Axes classes
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8,8))

# --- add scatter plots
markers = list('ov^<>12348sphHdD+x*|_')
N = len(markers)
for i, m in enumerate(markers):
    x = np.arange(N)
    y = np.repeat(i+1, N)
    ax.scatter(x, y, marker=m, label=m,
        s=50, c='cornflowerblue')

# --- tidy up and save to file
ax.set_xlim((-1,N))
ax.set_ylim((0,len(markers)+1))
ax.legend(loc='upper left', ncol=3,
    prop={'size':'xx-large'},
    shadow=True, title='Marker Legend')
ax.get_legend().get_title().set_color("red")
ax.set_title('Markers ' +
    '(with an oversized legend)')
fig.tight_layout(pad=1);
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Bar plots – using ax.bar() and ax.barh()

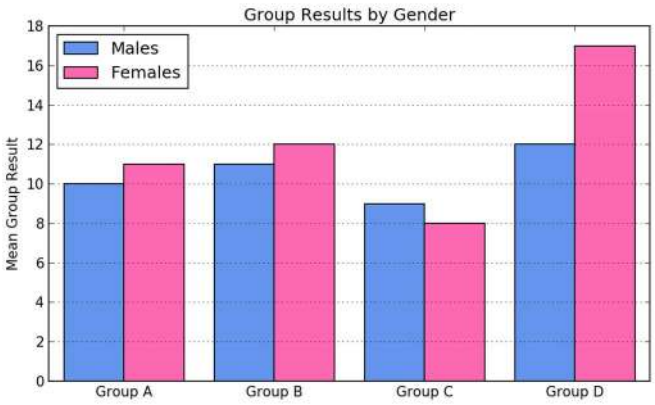
A simple bar chart

```
# --- get the data
N = 5
labels = list('ABCDEFGHIJKLM'[0:N])
data = np.array(range(N)) + np.random.rand(N)

# --- plot the data
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 3))
width = 0.8
tickLocations = np.arange(N)
ax.bar(tickLocations, data, width,
    color='wheat', edgecolor='#8B7E66',
    linewidth=4.0, align='center')

# --- pretty-up the plot
ax.set_xticks(ticks= tickLocations)
ax.set_xticklabels(labels)
ax.set_xlim(min(tickLocations)-0.6,
    max(tickLocations)+0.6)
ax.set_yticks(range(N)[1:])
ax.set_ylim((0,N))
ax.yaxis.grid(True)

# --- title and save
ax.set_title("Bar Plot w/ Oversized Edges")
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



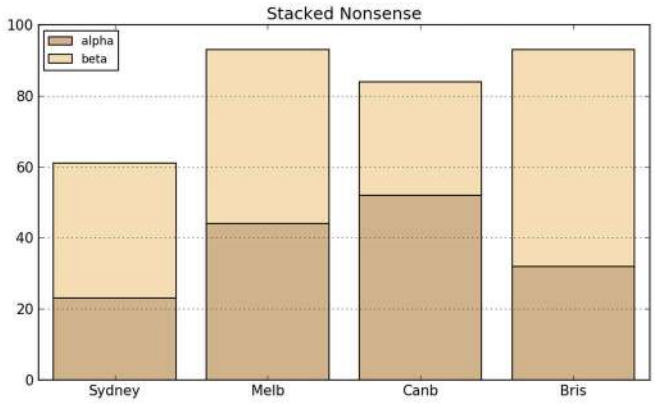
Side by side bar chart

```
# --- get the data
before = np.array([10, 11, 9, 12])
after = np.array([11, 12, 8, 17])
labels=['Group '+x for x in list('ABCD')]
# --- the plot – left then right
fig, ax = plt.subplots(figsize=(8, 5))
width = 0.4 # bar width
xlocs = np.arange(len(before))
ax.bar(xlocs-width, before, width,
     color='cornflowerblue', label='Males')
ax.bar(xlocs, after, width,
     color='hotpink', label='Females')
# --- labels, grids and title, then save
ax.set_xticks(ticks=range(len(before)))
ax.set_xticklabels(labels)
ax.yaxis.grid(True)
ax.legend(loc='best')
ax.set_ylabel('Mean Group Result')
ax.set_title('Group Results by Gender')
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



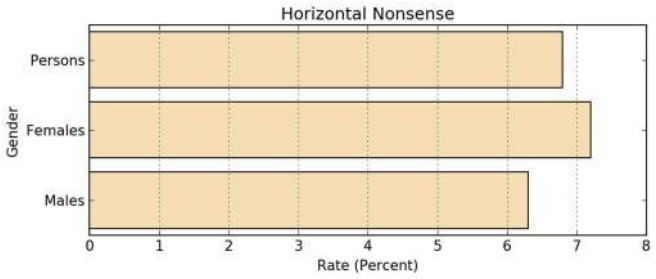
Stacked bar

```
# --- get some data
alphas = np.array( [23, 44, 52, 32] )
betas = np.array( [38, 49, 32, 61] )
labels = ['Sydney', 'Melb', 'Canb', 'Bris']

# --- the plot
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 5))
width = 0.8
xlocations = np.array(range(len(alphas)))
ax.bar(xlocations, alphas, width,
    label='alpha', color='tan',
    align='center')
ax.bar(xlocations, betas, width,
    label='beta', color='wheat',
align='center',
    bottom=alphas)

# --- pretty-up and save
ax.set_xticks(ticks=xlocations)
ax.set_xticklabels(labels)
ax.yaxis.grid(True)
ax.legend(loc='best', prop={'size':'small'})
ax.set_title("Stacked Nonsense")
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Horizontal bar charts

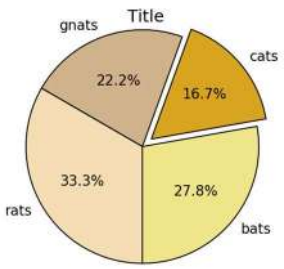
```
# --- the data
labels = ['Males', 'Females', 'Persons']
data = [6.3, 7.2, 6.8]
y = np.arange(len(data))

# --- plot
width = 0.8
fig, ax = plt.subplots(figsize=(8, 3.5))
ax.barh(y, data, width, color='wheat',
    align='center')

# --- tidy-up and save
ax.set_yticks(y)
ax.set_yticklabels(labels)
ax.xaxis.grid(True)
ax.set_ylabel('Gender');
ax.set_xlabel('Rate (Percent)')
ax.set_title("Horizontal Nonsense")
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Pie Chart – using ax.pie()

As nice as pie

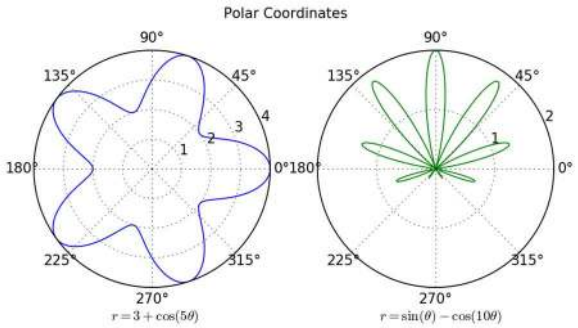
```
# --- get some data
data = np.array([5,3,4,6])
labels = ['bats', 'cats', 'gnats', 'rats']
explode = (0, 0.1, 0, 0) # explode cats
colrs=['khaki', 'goldenrod', 'tan', 'wheat']

# --- the plot – then tidy-up and save
fig, ax = plt.subplots(figsize=(8, 3.5))
ax.pie(data, explode=explode,
    labels=labels, autopct='%1.1f%%',
    startangle=270, colors=colrs)
ax.axis('equal') # keep it a circle
ax.set_title("Title")
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Polar – using ax.plot()

Polar coordinates

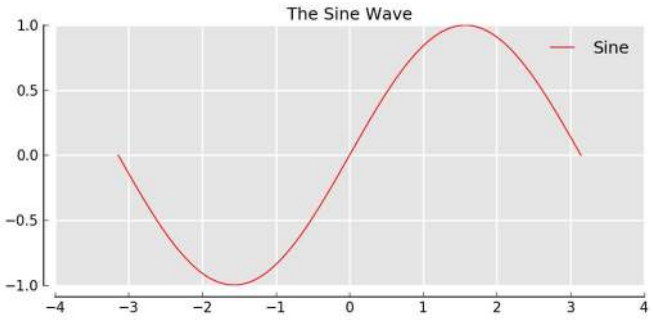
```
# --- theta
theta = np.linspace(-np.pi, np.pi, 800)

# --- get us a Figure
fig = plt.figure(figsize=(8,4))

# --- left hand plot
ax = fig.add_subplot(1,2,1, polar=True)
r = 3 + np.cos(5*theta)
ax.plot(theta, r)
ax.set_yticks([1,2,3,4])

# --- right hand plot
ax = fig.add_subplot(1,2,2, polar=True)
r = (np.sin(theta)) - (np.cos(10*theta))
ax.plot(theta, r, color='green')
ax.set_yticks([1,2])

# --- title, explanatory text and save
fig.suptitle('Polar Coordinates')
fig.text(x=0.24, y=0.05,
   s=r'$r = 3 + \cos(5 \theta)$')
fig.text(x=0.64, y=0.05,
   s=r'$r = \sin(\theta) - \cos(10' +
       r'\theta)$')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Plot spines

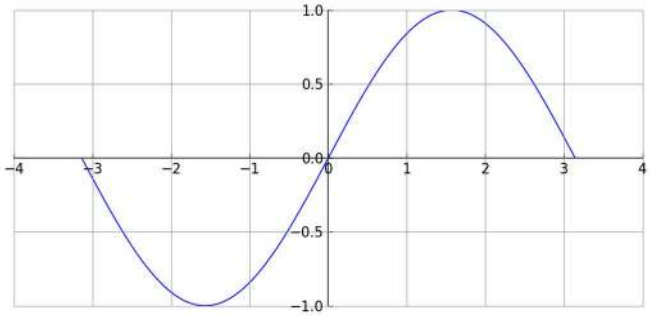
Hiding the top and right spines

```
# --- the data
x = np.linspace(-np.pi, np.pi, 800)
y = np.sin(x)

# --- the plot
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x, y, label='Sine', color='red')

# --- background and spines
ax.set_facecolor('#e5e5e5')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['left'].set_position(
    ('outward',10))
ax.spines['bottom'].set_position(
    ('outward',10))
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

# --- tidy up and save
# do the ax.grid() after setting ticks
ax.grid(b=True, which='both',
    color='white', linestyle='-',
    linewidth=1.5)
ax.set_axisbelow(True)
ax.legend(loc='best', frameon=False)
ax.set_title('The Sine Wave')
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Spines in the middle

```
# --- the data
x = np.linspace(-np.pi, np.pi, 800)
y = np.sin(x)

# --- the plot
fig, ax = plt.subplots(figsize=(8, 4))

# --- the spines
ax.plot(x, y, label='Sine')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position((
    'data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position((
    'data',0))
ax.grid(b=True, which='both',
     color='#888888', linestyle='-',
     linewidth=0.5)

# --- tidy-up and save
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



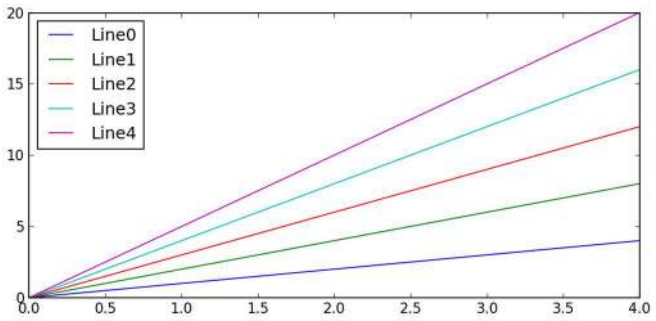
Legends

Legend within the plot

Use the 'loc' argument to place the legend

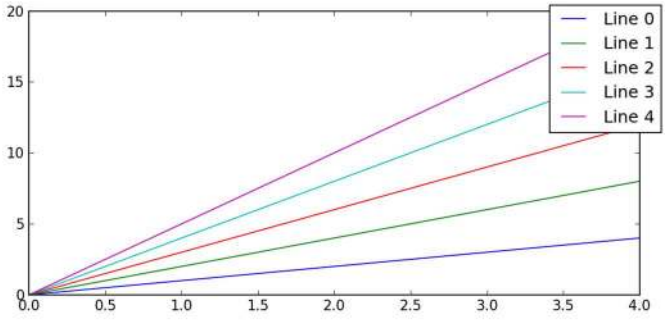
```
# --- get the empty Figure and Axes classes
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))

# --- plot something
N = 5
x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1), label='Line'+str(j))

# --- place the legend
ax.legend(loc='upper left')

# --- save and close
fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```

Note: for legend placement, often loc='best' works perfectly as expected.



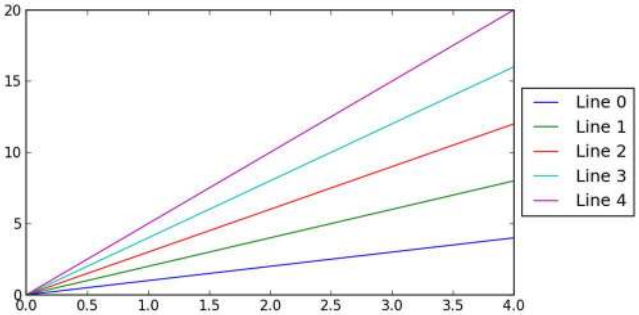
Legend slightly outside of the plot

```
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))
N = 5
x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1),
        label='Line '+str(j))

ax.legend(bbox_to_anchor=(1.05, 1.05))

fig.tight_layout(pad=1)
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



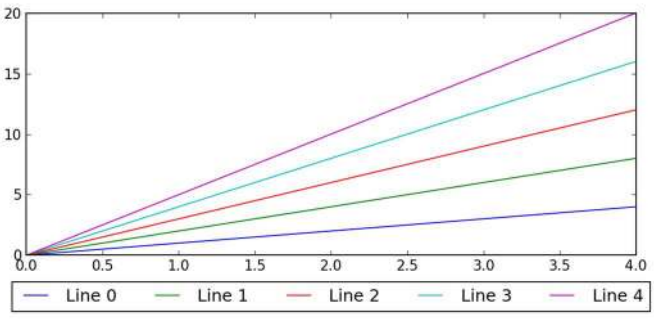
Legend to the right of the plot

```
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))

x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1),
        label='Line '+str(j))

fig.tight_layout(pad=1)
box = ax.get_position()  # Shrink plot
ax.set_position([box.x0, box.y0,
    box.width * 0.8, box.height])
ax.legend(bbox_to_anchor=(1, 0.5),
    loc='center left')   # Put legend
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Legend below the plot

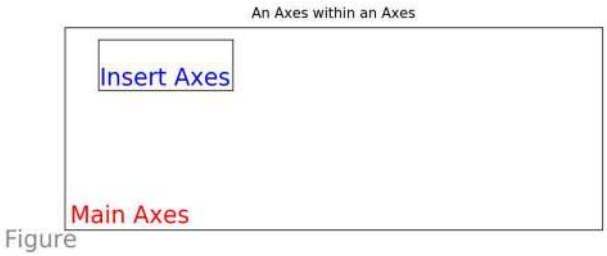
```
plt.style.use('classic')
fig, ax = plt.subplots(figsize=(8, 4))

N = 5
x = np.arange(N)
for j in range(5):
    ax.plot(x, x*(j+1),
        label='Line '+str(j))

fig.tight_layout(pad=1)
box = ax.get_position()
ax.set_position([box.x0,
    box.y0 + box.height * 0.15,
    box.width, box.height * 0.85])
ax.legend(bbox_to_anchor=(0.5, -0.075),
    loc='upper center', ncol=N)

fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Multiple plots on a canvas

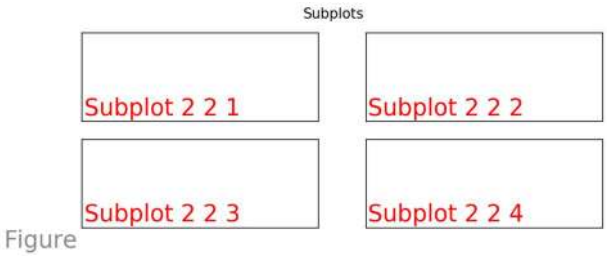
Using Axes to place a plot within a plot

```
fig = plt.figure(figsize=(8,3))
fig.text(x=0.01, y=0.01, s='Figure',
    color='#888888', ha='left',
    va='bottom', fontsize=20)

# --- Main Axes
ax = fig.add_axes([0.1,0.1,0.8,0.8])
ax.text(x=0.01, y=0.01, s='Main Axes',
    color='red', ha='left', va='bottom',
    fontsize=20)
ax.set_xticks([]); ax.set_yticks([])

# --- Insert Axes
ax= fig.add_axes([0.15,0.65,0.2,0.2])
ax.text(x=0.01, y=0.01, s='Insert Axes',
    color='blue', ha='left', va='bottom',
    fontsize=20)
ax.set_xticks([]); ax.set_yticks([])
fig.suptitle('An Axes within an Axes')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



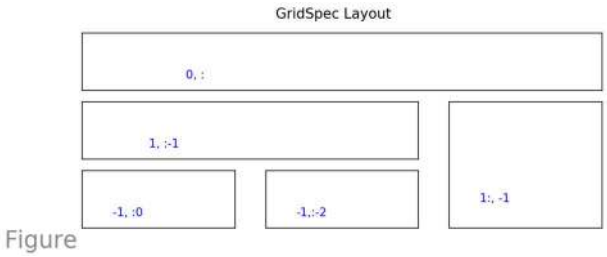
Simple subplot grid layouts

```
fig = plt.figure(figsize=(8,3))
fig.text(x=0.01, y=0.01, s='Figure',
    color='#888888', ha='left',
    va='bottom', fontsize=20)

for i in range(4):
    # fig.add_subplot(nrows, ncols, num)
    ax = fig.add_subplot(2, 2, i+1)
    ax.text(x=0.01, y=0.01,
        s='Subplot 2 2 '+str(i+1),
        color='red', ha='left',
        va='bottom', fontsize=20)
    ax.set_xticks([]); ax.set_yticks([])

ax.set_xticks([]); ax.set_yticks([])
fig.suptitle('Subplots')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Using GridSpec layouts (like list slicing)

```
import matplotlib.gridspec as gs
gs = gs.GridSpec(3, 3) # nrows, ncols
fig = plt.figure(figsize=(8,3))
fig.text(x=0.01, y=0.01, s='Figure',
    color='#888888', ha='left',
    va='bottom', fontsize=20)
ax1 = fig.add_subplot(gs[0, :]) # row,col
ax1.text(x=0.2,y=0.2,s='0, :', color='b')
ax2 = fig.add_subplot(gs[1,:-1])
ax2.text(x=0.2,y=0.2,s='1, :-1', color='b')
ax3 = fig.add_subplot(gs[1:, -1])
ax3.text(x=0.2,y=0.2, s='1:, -1', color='b')
ax4 = fig.add_subplot(gs[-1,0])
ax4.text(x=0.2,y=0.2, s='-1, :0', color='b')
ax5 = fig.add_subplot(gs[-1,-2])
ax5.text(x=0.2,y=0.2, s='-1,:-2', color='b')
for a in fig.get_axes():
    a.set_xticks([])
    a.set_yticks([])

fig.suptitle('GridSpec Layout')
fig.savefig('filename.png', dpi=125)
plt.close('all')
```



Style

Matplotlib has many styles
There are many different styles to choose from.

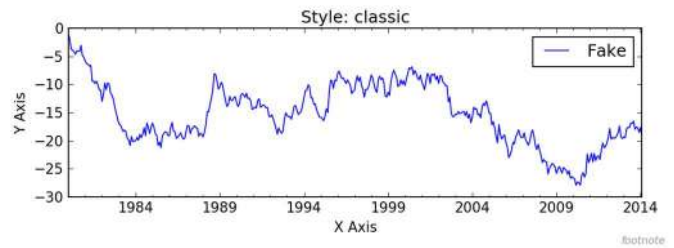
```
print(plt.style.available)
```

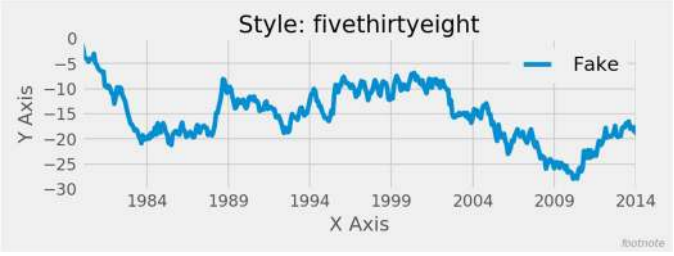
So let's choose one.

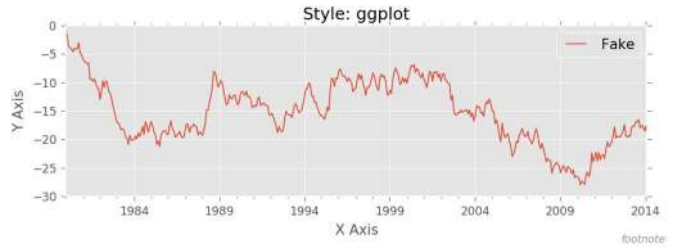
```
plt.style.use('ggplot')
```

Example line chart in three different styles

```
# --- fake up some data
x = pd.period_range('1980-01-01',
    periods=410, freq='M')
y = np.random.randn(len(x)).cumsum()
df = pd.DataFrame(y, index=x,
    columns=['Fake'])

# --- plot data using different styles
styles = ['classic', 'ggplot',
    'fivethirtyeight']
for style in styles:
    plt.style.use(style)
    ax = df.plot()
    ax.set_title('Style: ' + style)
    ax.set_xlabel('X Axis')
    ax.set_ylabel('Y Axis')
    fig = ax.figure
    fig.set_size_inches(8, 3)
    fig.tight_layout(pad=1)
    fig.text(0.99, 0.01, 'footnote',
        ha='right', va='bottom',
        fontsize=9, fontstyle='italic',
        color='#999999')
    fig.savefig('f-'+style+'.png', dpi=125)
    plt.close('all')
```







Roll your own style sheet

The Mark Graph Style Sheet

```
font.family: sans-serif
font.style: normal
font.variant: normal
font.weight: medium
font.stretch: normal
font.sans-serif: Helvetica, Arial, Avant
Garde, sans-serif
font.size: 14.0

lines.linewidth: 2.0
lines.solid_capstyle: butt

legend.fancybox: true
legend.fontsize: x-small
legend.framealpha: 0.5

axes.prop_cycle: cycler('color', ['ef5962',
    '5a9dd2', '7dc16d', 'f9a560', '9d69aa',
    'cd6f5a', 'd680b3', '737373'])
axes.facecolor: white
axes.titlesize: large  # fontsize of the axes
title
axes.labelsize: medium # x and y label size
axes.labelcolor: 111111
axes.axisbelow: true
axes.grid: true
axes.edgecolor: white
axes.linewidth: 0.01

patch.edgecolor: white
patch.linewidth: 0.01

svg.fonttype: path

grid.linestyle: -
grid.linewidth: 0.5
grid.color: e7e7e7

xtick.major.size: 0
xtick.minor.size: 0
xtick.labelsize: small
xtick.color: 333333
ytick.major.size: 0
ytick.minor.size: 0
ytick.labelsize: small
ytick.color: 333333

figure.figsize: 8, 4 # inches
figure.facecolor: white

text.color: black

savefig.edgecolor: white
savefig.facecolor: white
```

Which is saved as "markgraph.mplstyle"

Using your own style sheet

```
plt.style.use('./markgraph.mplstyle')
```

Cautionary notes

This cheat sheet was cobbled together by bots roaming the dark recesses of the Internet seeking ursine and pythonic myths. There is no guarantee the narratives were captured and transcribed accurately. You use these notes at your own risk. You have been warned.