matplotlib

Lecture 10

Dr. Colin Rundel

matplotlib & pyplot

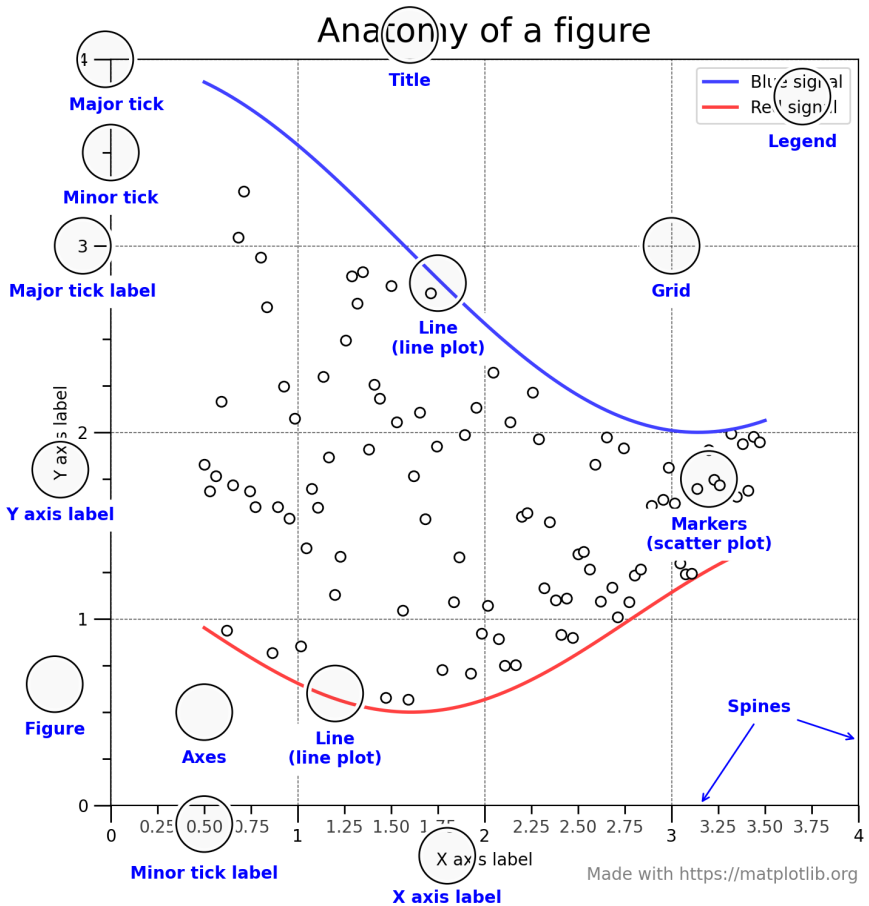
> matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

```
1 import matplotlib as mpl
2 import matplotlib.pyplot as plt
```

Why do we usually import only pyplot then?

> Matplotlib is the whole package; matplotlib.pyplot is a module in matplotlib; and pylab is a module that gets installed alongside matplotlib.
>
> Pyplot provides the state-machine interface to the underlying object-oriented plotting library. The state-machine implicitly and automatically creates figures and axes to achieve the desired plot.

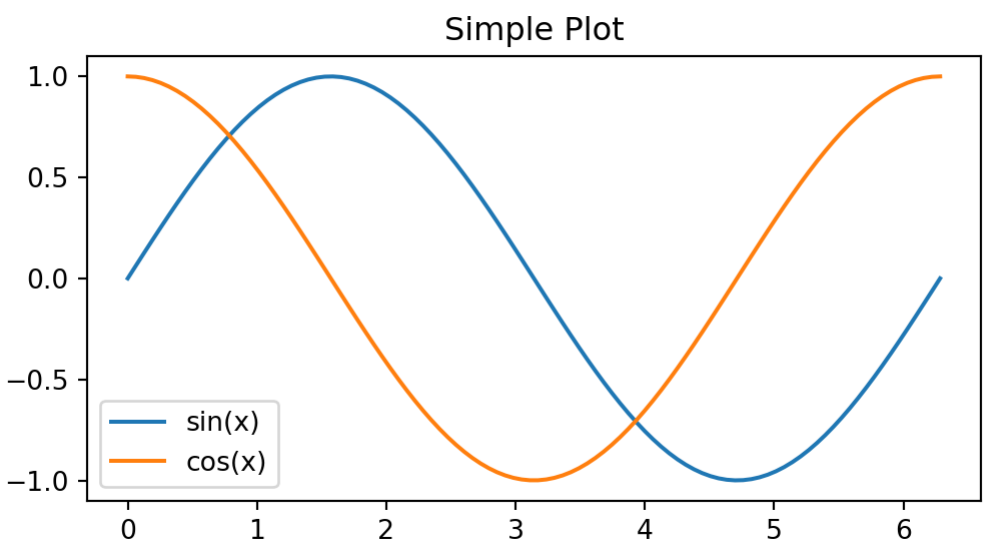
Plot anatomy



- **Figure** - The entire plot (including subplots)
- **Axes** - Subplot attached to a figure, contains the region for plotting data and x & y axis
- **Axis** - Set the scale and limits, generate ticks and ticklabels
- **Artist** - Everything visible on a figure: text, lines, axis, axes, etc.

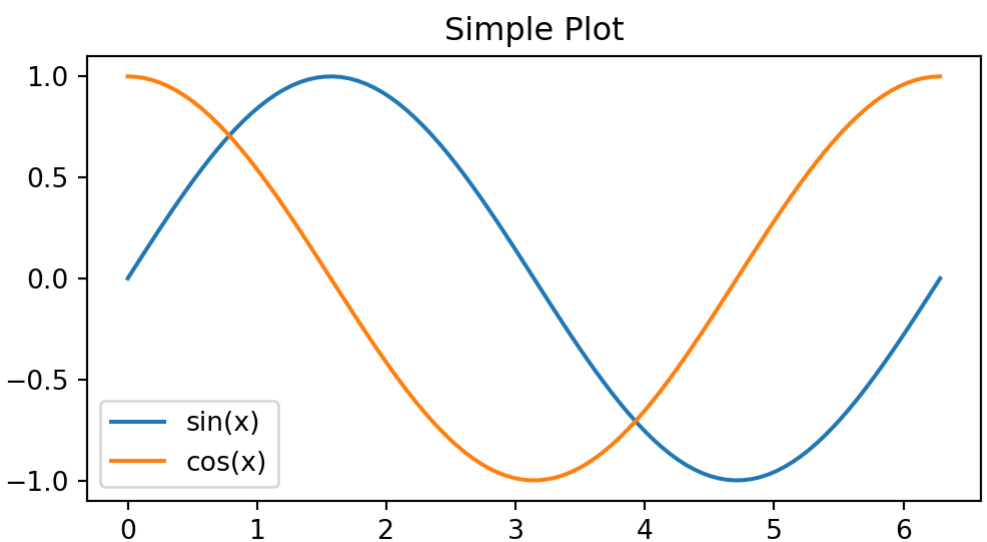
Basic plot - OO style

```
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

fig, ax = plt.subplots(figsize=(6, 3))
ax.plot(x, y1, label="sin(x)")
ax.plot(x, y2, label="cos(x)")
ax.set_title("Simple Plot")
ax.legend()
```



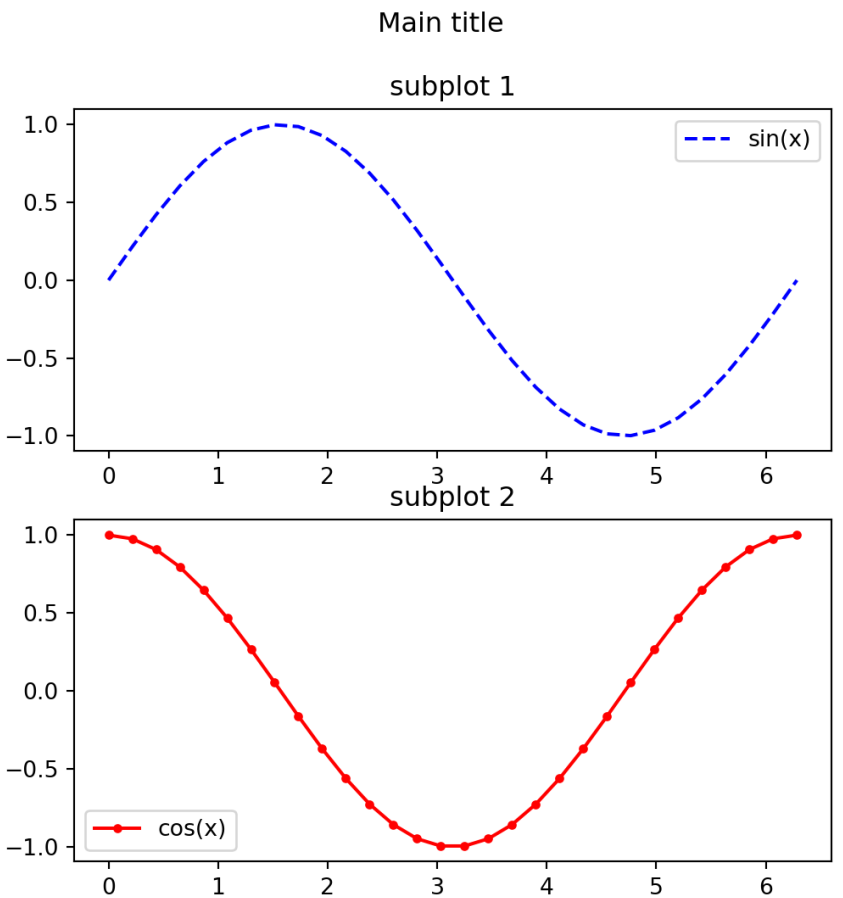
Basic plot - pyplot style

```
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(6, 3))
plt.plot(x, y1, label="sin(x)")
plt.plot(x, y2, label="cos(x)")
plt.title("Simple Plot")
plt.legend()
```



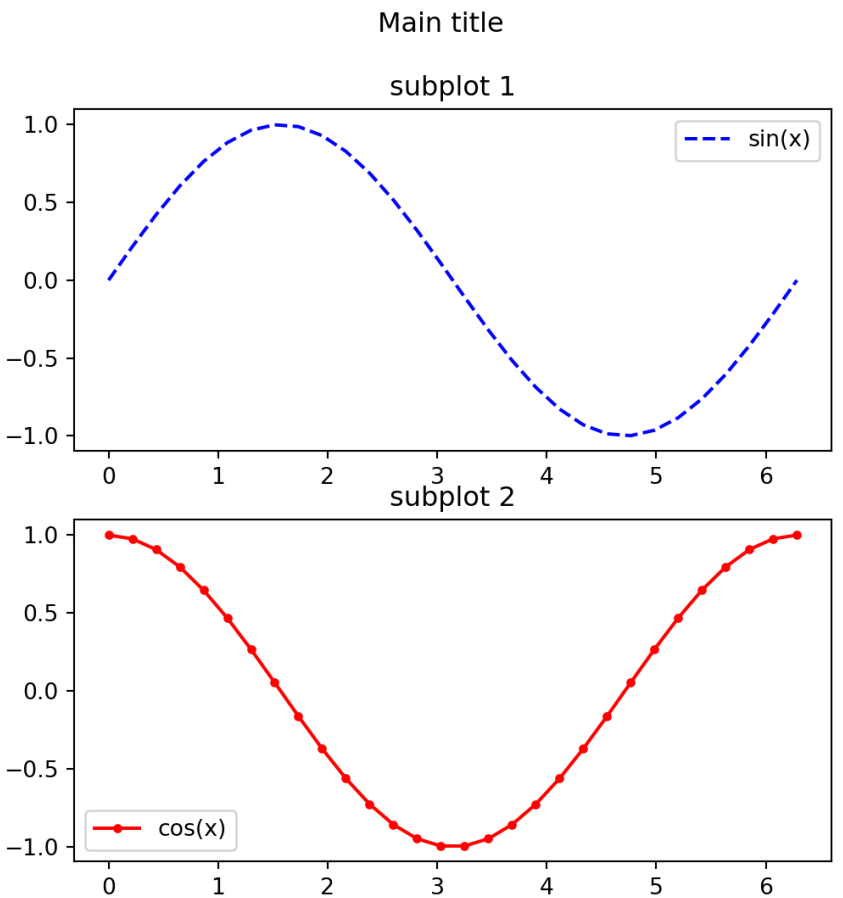
Subplots (OO)

```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.sin(x)
y2 = np.cos(x)

fig, (ax1, ax2) = plt.subplots(
  2, 1, figsize=(6, 6)
)

fig.suptitle("Main title")

ax1.plot(x, y1, "--b", label="sin(x)")
ax1.set_title("subplot 1")
ax1.legend()

ax2.plot(x, y2, ".-r", label="cos(x)")
ax2.set_title("subplot 2")
ax2.legend()
```



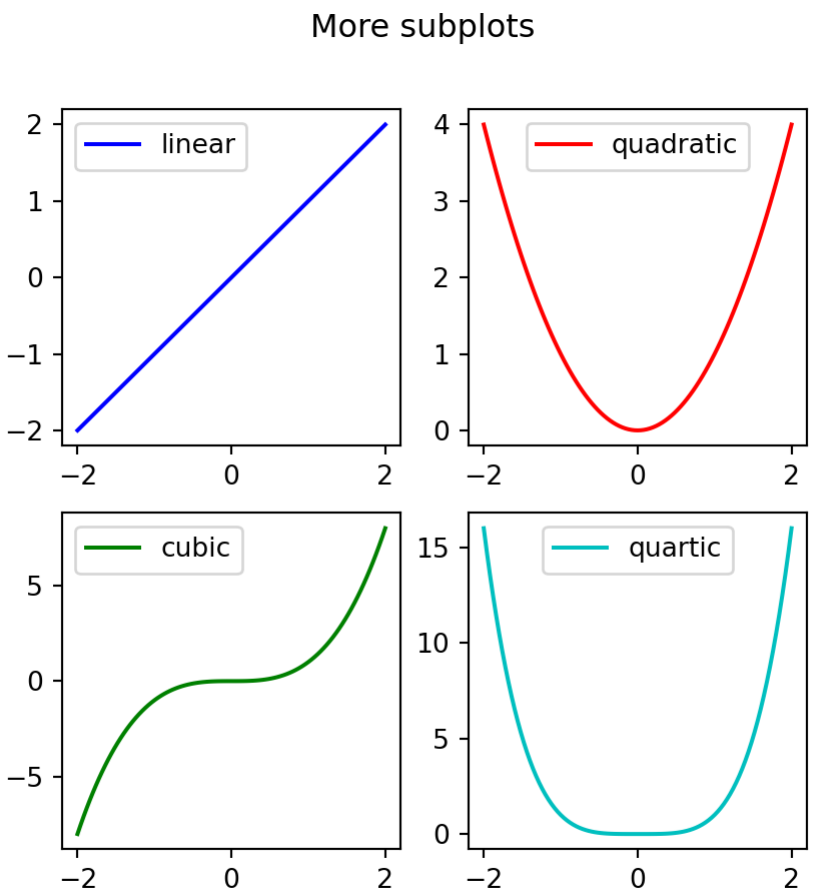
Subplots (pyplot)

```
1  x = np.linspace(0, 2*np.pi, 30)
2  y1 = np.sin(x)
3  y2 = np.cos(x)
4
5  plt.figure(figsize=(6, 6))
6
7  plt.suptitle("Main title")
8
9  plt.subplot(211)
10 plt.plot(x, y1, "--b", label="sin(x)")
11 plt.title("subplot 1")
12 plt.legend()
13
14 plt.subplot(2,1,2)
15 plt.plot(x, y2, ".-r", label="cos(x)")
16 plt.title("subplot 2")
17 plt.legend()
18
19 plt.show()
```



More subplots

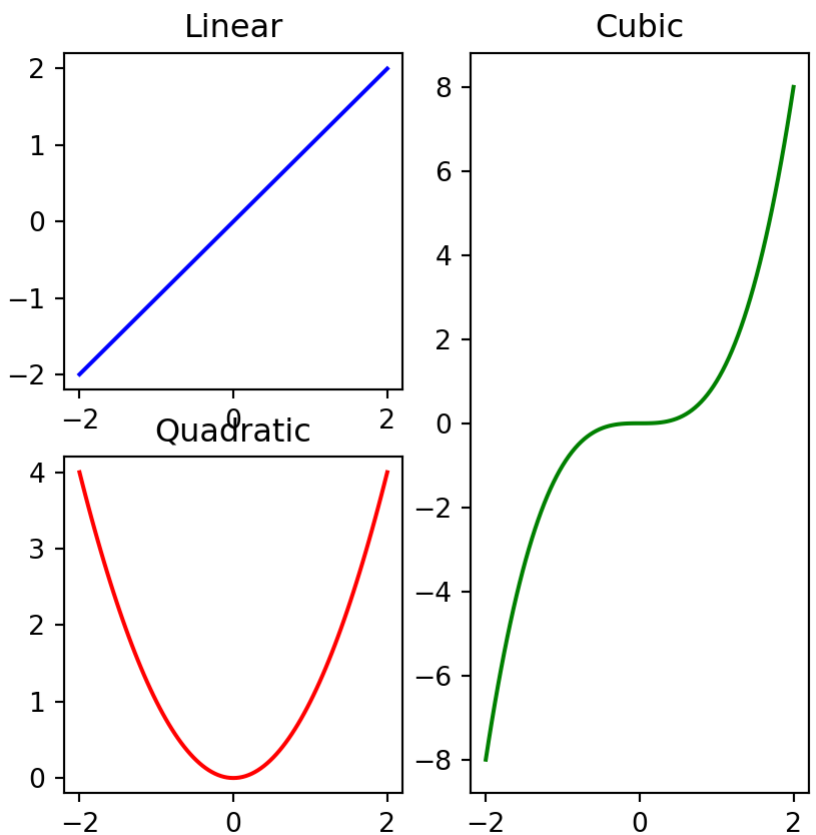
```
1  x = np.linspace(-2, 2, 101)
2
3  fig, axs = plt.subplots(2, 2, figsize=(5, 5))
4
5  fig.suptitle("More subplots")
6
7  axs[0,0].plot(x, x, "b", label="linear")
8  axs[0,1].plot(x, x**2, "r", label="quadratic")
9  axs[1,0].plot(x, x**3, "g", label="cubic")
10 axs[1,1].plot(x, x**4, "c", label="quartic")
11
12 [ax.legend() for row in axs for ax in row]
```



`axs` here is a 2x2 numpy array of axes

Fancy subplots (mosaic)

```
1  x = np.linspace(-2, 2, 101)
2
3  fig, axd = plt.subplot_mosaic(
4    [['upleft', 'right'],
5     ['lowleft', 'right']],
6    figsize=(5, 5)
7  )
8
9  axd['upleft' ].plot(x, x,    "b", label="linear"
10 axd['lowleft'].plot(x, x**2, "r", label="quadrat
11 axd['right'  ].plot(x, x**3, "g", label="cubic")
12
13 axd['upleft'].set_title("Linear")
14 axd['lowleft'].set_title("Quadratic")
15 axd['right'].set_title("Cubic")
```



`axs` here is a dictionary of axes

Format strings

For quick formating of plots (scatter and line) format strings are a useful shorthand, generally they use the format `'[marker][line][color]'`,

character	shape
`.`	point
`,`	pixel
`o`	circle
`v`	triangle down
`^`	triangle up
`<`	triangle left
`>`	triangle right
...	+ more

character	line style
`-`	solid
`--`	dashed
`-.`	dash-dot
`:`	dotted

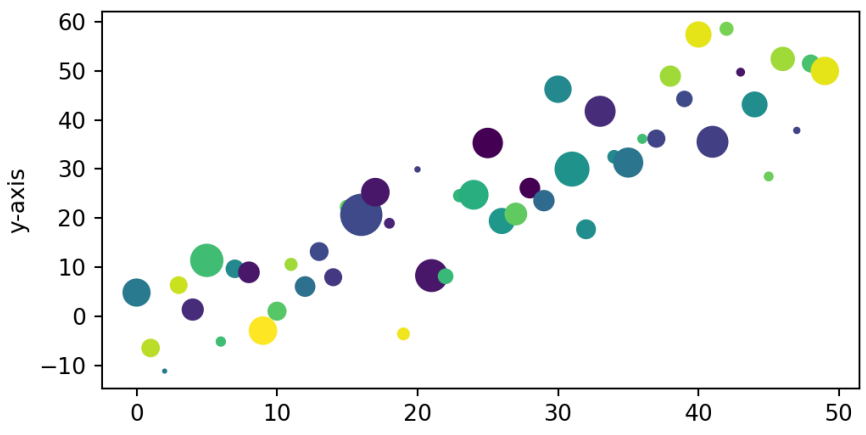
character	color
`b`	blue
`g`	green
`r`	red
`c`	cyan
`m`	magenta
`y`	yellow
`k`	black
`w`	white

See Notes section of pyplot.plot docs

Plotting data

Beyond creating plots for arrays (and lists), addressable objects like dicts and DataFrames can be used via `data`,

```
np.random.seed(19680801)
d = {'x': np.arange(50),
     'color': np.random.randint(0, 50, 50),
     'size': np.abs(np.random.randn(50)) * 100}
d['y'] = d['x'] + 10 * np.random.randn(50)


plt.figure(figsize=(6, 3))
plt.scatter(
  'x', 'y', c='color', s='size',
  data=d
)
plt.xlabel("x-axis")
plt.ylabel("y-axis")

plt.show()
```

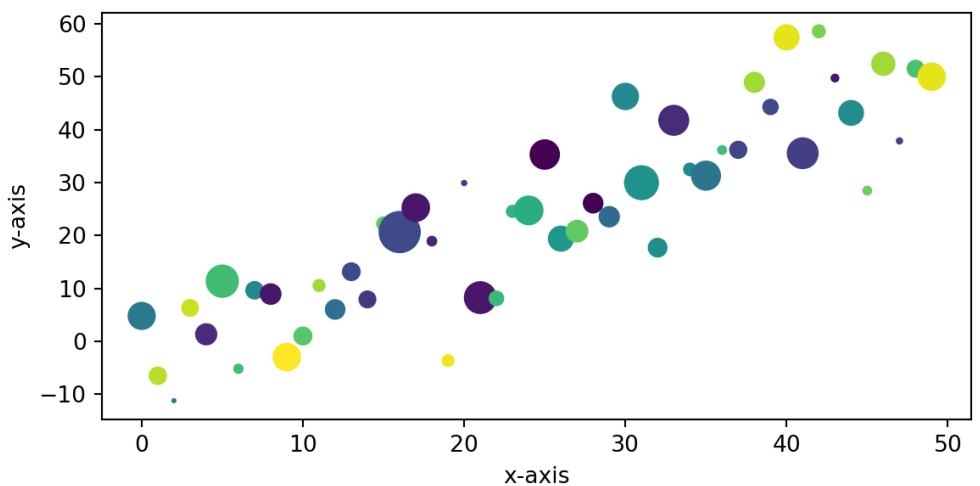


Example from Basic Usage guide

Constrained layout

To fix the legend clipping we can use the “contrained” layout to adjust automatically,

```
np.random.seed(19680801)
d = {'x': np.arange(50),
     'color': np.random.randint(0, 50, 50),
     'size': np.abs(np.random.randn(50)) * 100}
d['y'] = d['x'] + 10 * np.random.randn(50)


plt.figure(
  figsize=(6, 3),
  layout="constrained"
)
plt.scatter(
  'x', 'y', c='color', s='size',
  data=d
)
plt.xlabel("x-axis")
plt.ylabel("y-axis")

plt.show()
```

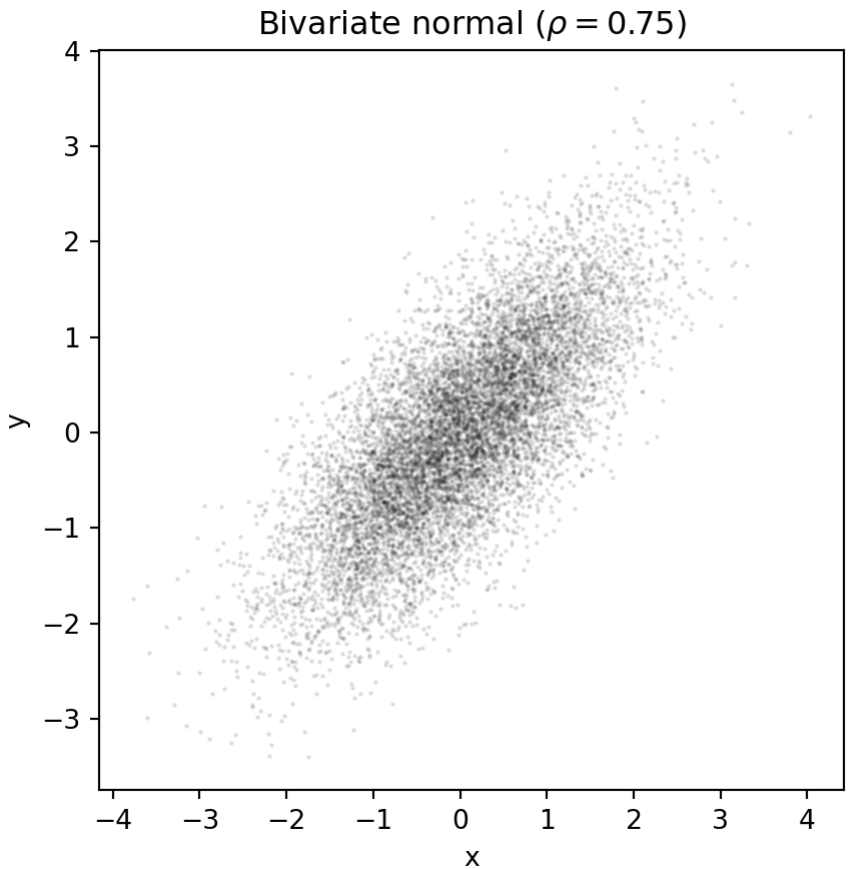


pyplot w/ pandas data

Data can also come from DataFrame objects or series,

```
df = pd.DataFrame({
  "x": np.random.normal(size=10000)
}).assign(
  y = lambda d: np.random.normal(0.75*d.x, np.sqrt(1-0.75**2), size=10000)
)

fig, ax = plt.subplots(figsize=(5,5))

ax.scatter('x', 'y', c='k', data=df, alpha=0.1, s=0.5)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title("Bivariate normal ($\\rho=0.75$)")
```

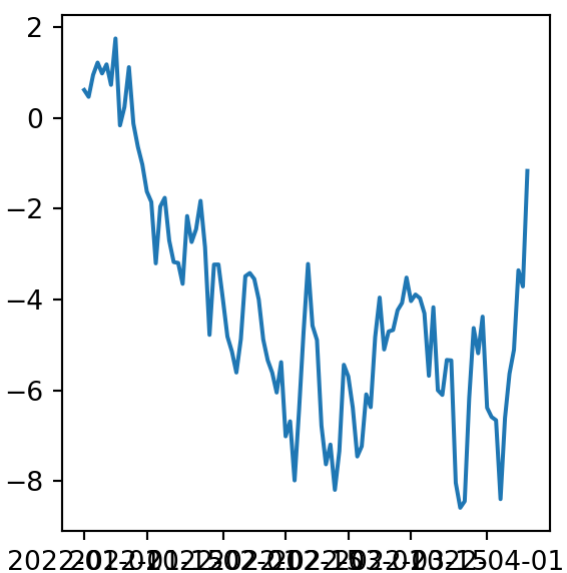
pyplot w/ pandas data



pyplot w/ pandas series

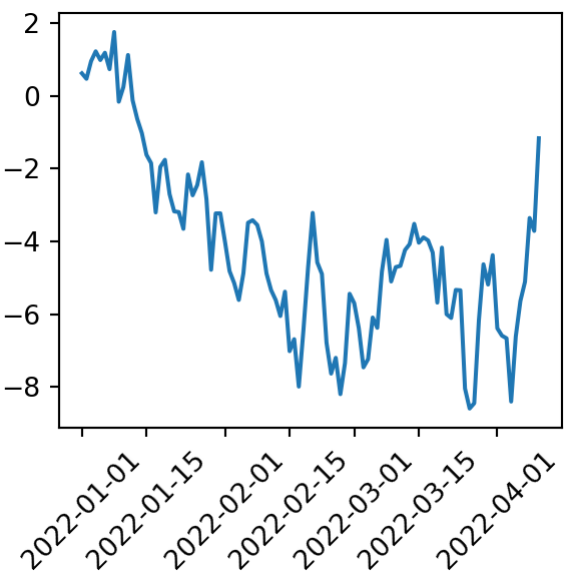
Series objects can also be plotted directly, the index is used as the x axis labels,

```
s = pd.Series(
  np.cumsum( np.random.normal(size=100) ),
  index = pd.date_range("2022-01-01",
                        periods=100, freq="D")
)
plt.figure(figsize=(3, 3), layout="constrained")
plt.plot(s)
plt.show()
```




```
plt.figure(figsize=(3, 3), layout="constrained")
plt.plot(s.index, s.values)
plt.xticks(rotation=45)
```

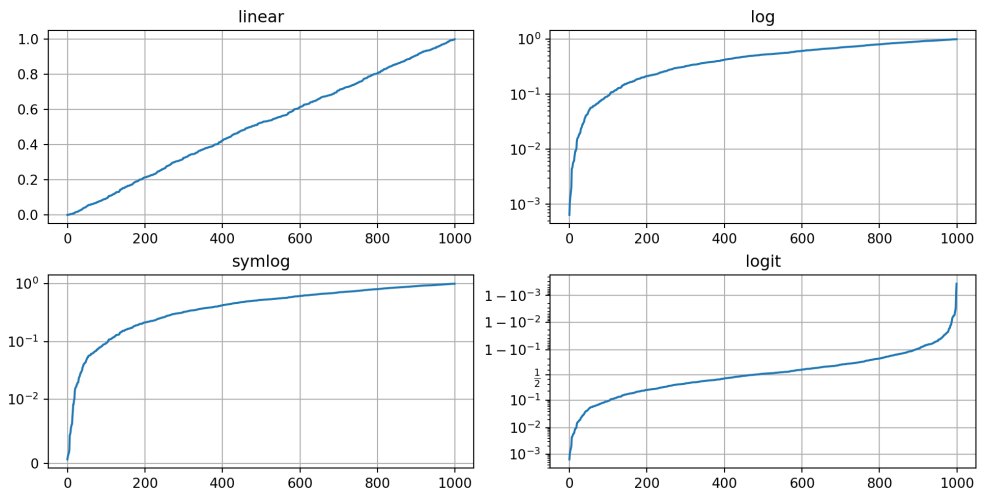
```
plt.show()
```



Scales

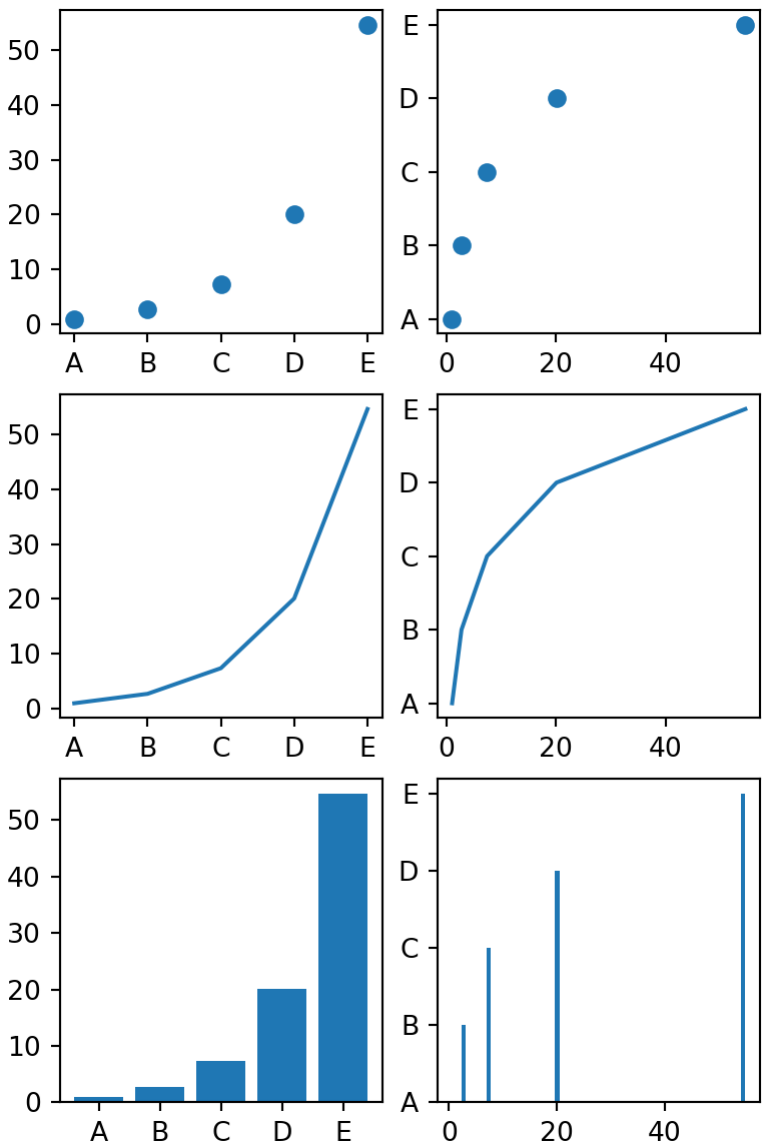
Axis scales can be changed via `plt.xscale()`, `plt.yscale()`, `ax.set_xscale()`, or `ax.set_yscale()`, supported values are “linear”, “log”, “symlog”, and “logit”.

```
y = np.sort( np.random.sample(size=1000) )
x = np.arange(len(y))

plt.figure(layout="constrained")

scales = ['linear', 'log', 'symlog', 'logit']
for i, scale in zip(range(4), scales):
  plt.subplot(221+i)
  plt.plot(x, y)
  plt.grid(True)
  if scale == 'symlog':
    plt.yscale(scale, linthresh=0.01)
  else:
    plt.yscale(scale)
  plt.title(scale)


plt.show()
```



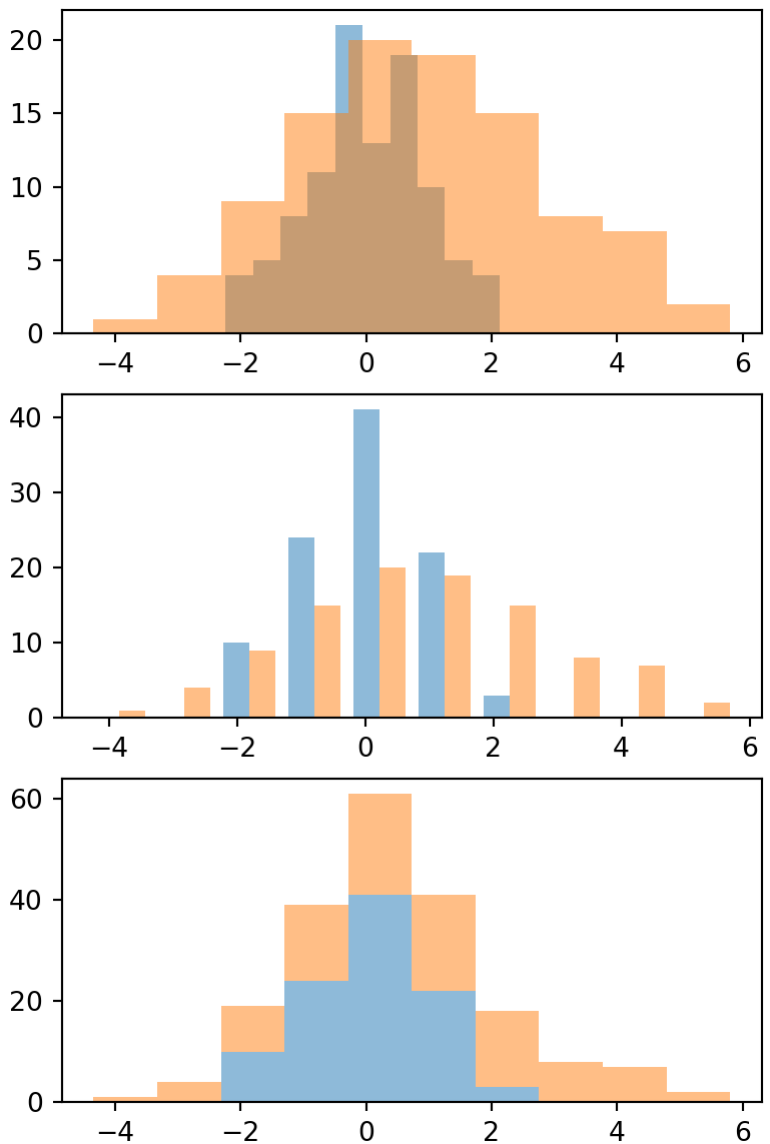
Categorical data

```
df = pd.DataFrame({
  "cat": ["A", "B", "C", "D", "E"],
  "value": np.exp(range(5))
})

plt.figure(figsize=(4, 6), layout="constrained")

plt.subplot(321)
plt.scatter("cat", "value", data=df)
plt.subplot(322)
plt.scatter("value", "cat", data=df)

plt.subplot(323)
plt.plot("cat", "value", data=df)
plt.subplot(324)
plt.plot("value", "cat", data=df)

plt.subplot(325)
b = plt.bar("cat", "value", data=df)
plt.subplot(326)
b = plt.bar("value", "cat", data=df)

plt.show()
```



Histograms

```
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,2, size=100)
})

plt.figure(figsize=(4, 6), layout="constrained")

plt.subplot(311)
h = plt.hist("x1", bins=10, data=df, alpha=0.5)
h = plt.hist("x2", bins=10, data=df, alpha=0.5)

plt.subplot(312)
h = plt.hist(df, alpha=0.5)

plt.subplot(313)
h = plt.hist(df, stacked=True, alpha=0.5)

plt.show()
```



Boxplots

```
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,2, size=100),
  "x3": np.random.normal(-1,3, size=100)
}).melt()

df
```

```
    variable     value
0         x1  0.085670
1         x1  1.660256
2         x1  1.596326
3         x1 -1.167331
4         x1  0.221311
..       ...       ...
295       x3 -0.822684
296       x3  2.081603
297       x3  2.082767
298       x3 -0.046562
299       x3  0.373482

[300 rows x 2 columns]
```

```
plt.figure(figsize=(4, 4), layout="constrained")

plt.boxplot("value", positions="variable", data=
```

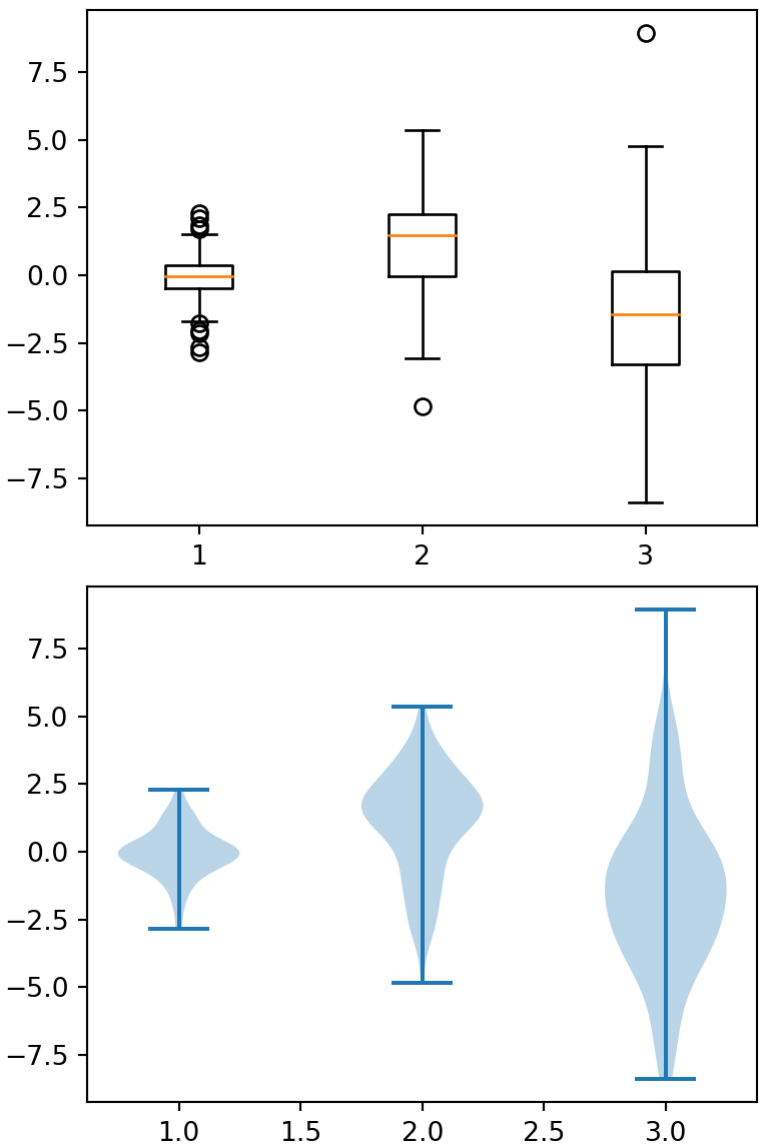
```
Error: ValueError: List of boxplot statistics and
`positions` values must have same the length
```

```
plt.boxplot(df.value, positions=df.variable)
```

```
Error: ValueError: List of boxplot statistics and
`positions` values must have same the length
```

Boxplots (cont.)

```
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,2, size=100),
  "x3": np.random.normal(-1,3, size=100)
})

plt.figure(figsize=(4, 6), layout="constrained")

plt.subplot(211)
b = plt.boxplot(df)

plt.subplot(212)
v = plt.violinplot(df)

plt.show()
```

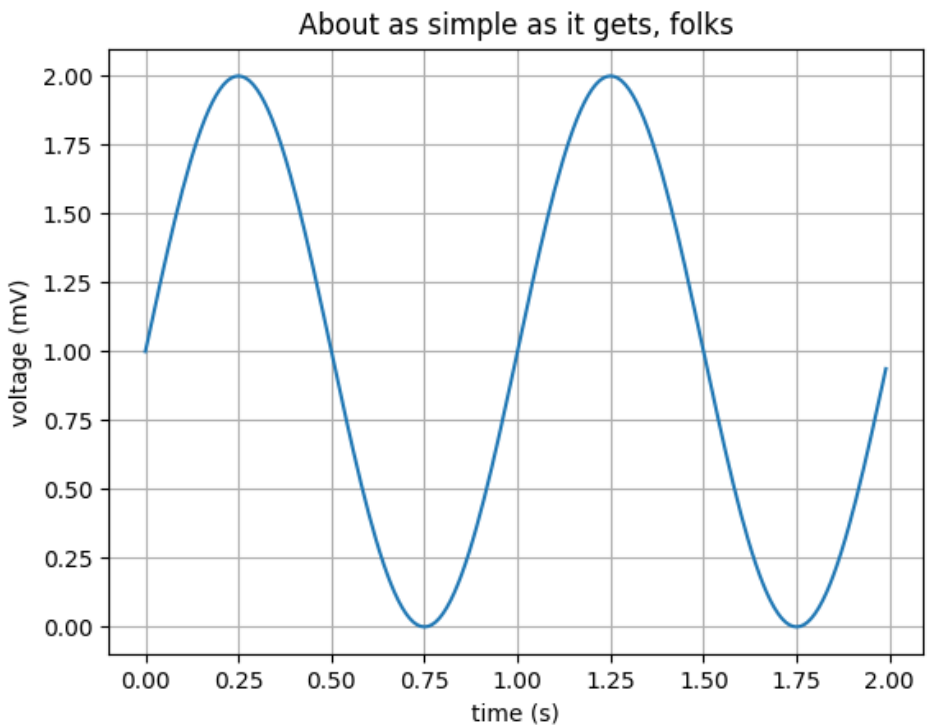


Other Plot Types

matplotlib.org/stable/plot_types/

Exercise 1

To the best of your ability recreate the following plot,



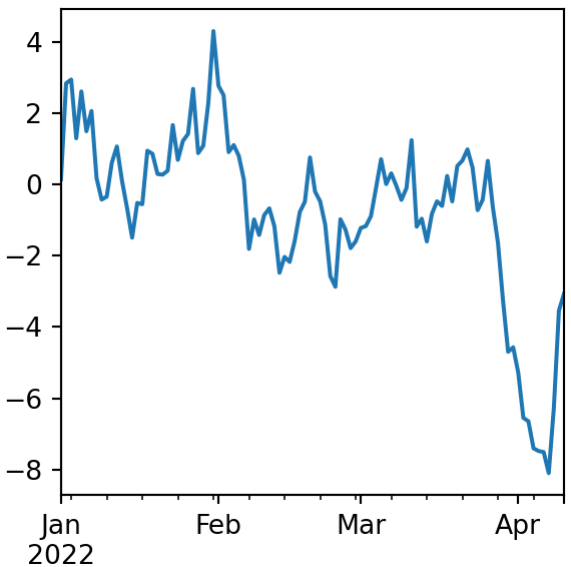
From matplotlib examples

Plotting with pandas

plot methods

Both Series and DataFrame objects have a plot method which can be used to create visualizations - dtypes determine the type of plot produced.

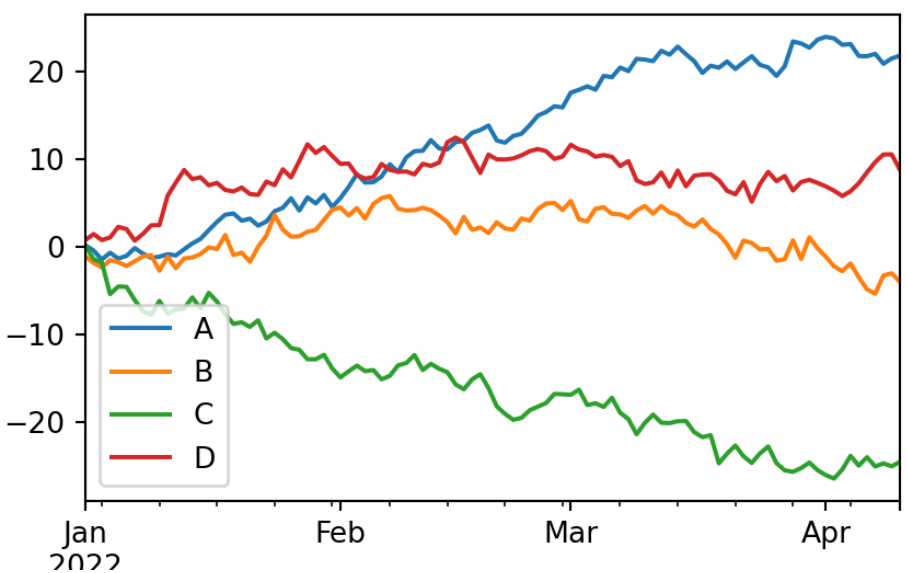
```
s = pd.Series(
  np.cumsum( np.random.normal(size=100) ),
  index = pd.date_range("2022-01-01",
                        periods=100, freq="D")
)

plt.figure(figsize=(3,3), layout="constrained")
s.plot()
plt.show()
```



These methods are just using pyplot to create plots, and so you can use pyplot functiomns and methods to modify

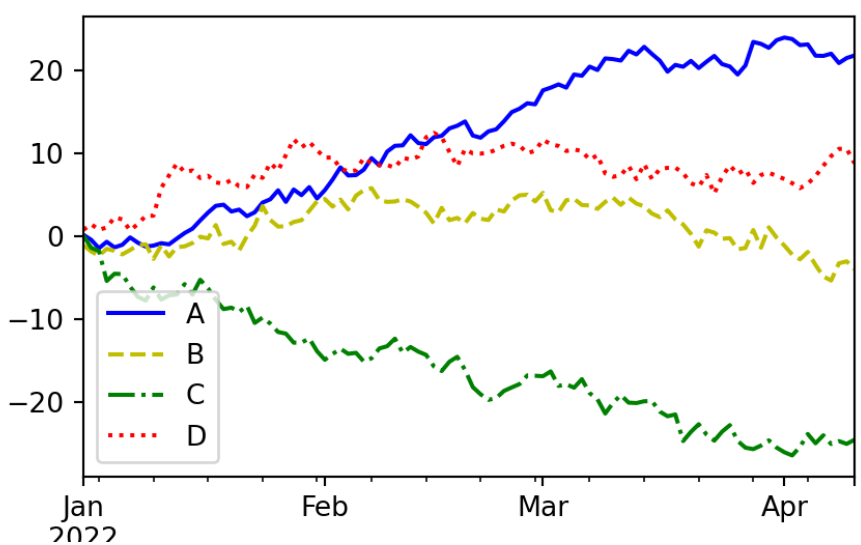
DataFrame plot

```
df = pd.DataFrame(
  np.cumsum( np.random.normal(size=(100,4)), axis=0),
  index = pd.date_range("2022-01-01", periods=100, freq="D"),
  columns = list("ABCD")
)

plt.figure(layout="constrained")
df.plot(figsize=(5,3))
plt.show()
```



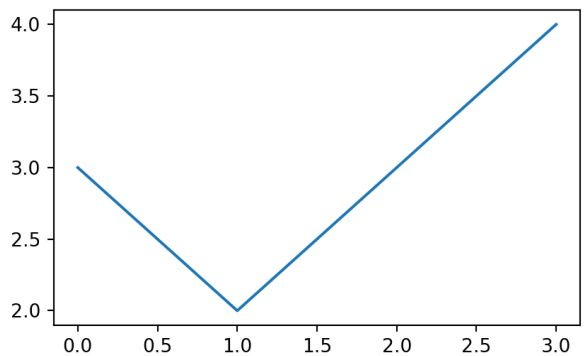
DataFrame line styles

```
df.plot(
  figsize=(5,3),
  style = {
    "A": "-b",
    "B": "--y",
    "C": "-.g",
    "D": ":r"
  }
)
```

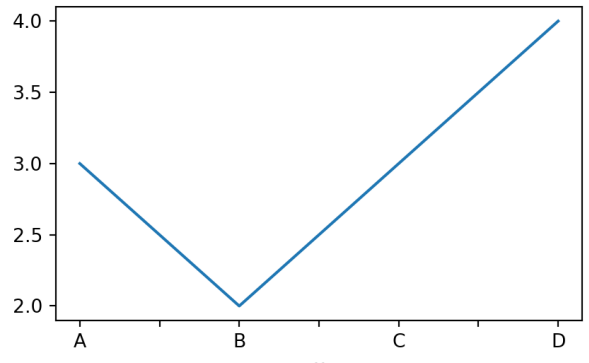


DataFrame plot - categorical

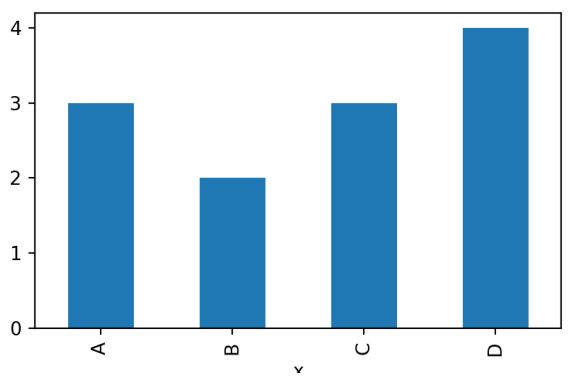
```
df = pd.DataFrame({
 "x": list("ABCD"),
 "y": np.random.poisson(lam=2, size=4)
})

df.plot(figsize=(5,3), legend=False)
```




```
df.set_index("x").plot(figsize=(5,3),legend=Fals
```



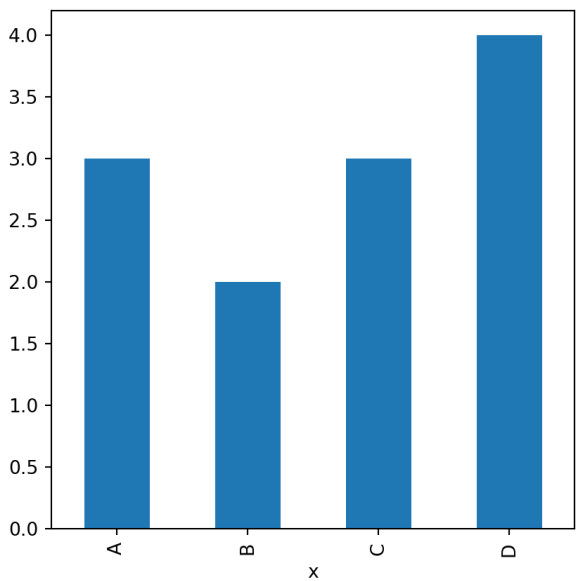

```
df.set_index("x").plot(
  figsize=(5,3), kind="bar", legend=False
)
```



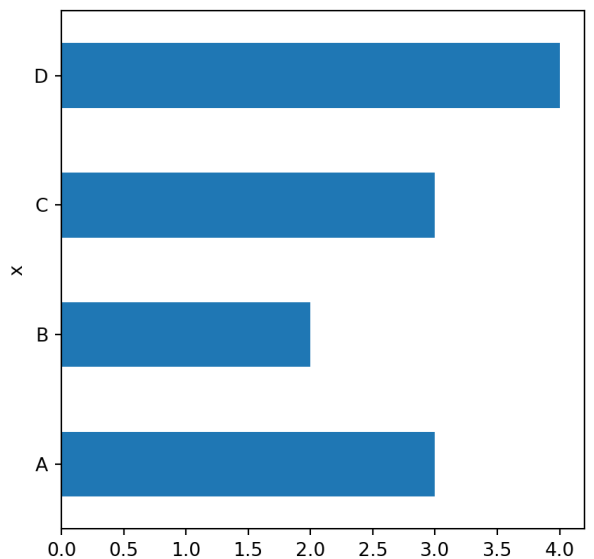
Other plot types

Plot types can be changed via the `kind` argument or using one of the `DataFrame.plot.<kind>` method,

```
df.set_index("x").plot.bar(
  legend=False, figsize=(5,5)
)
```

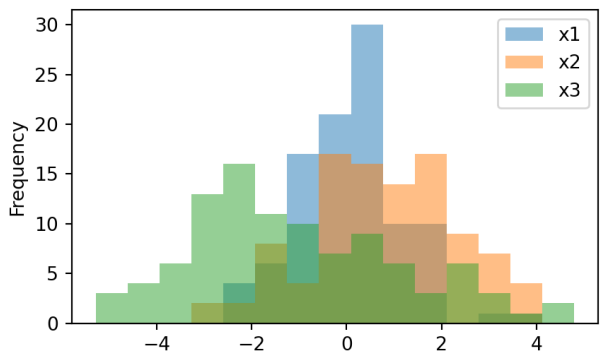



```
df.set_index("x").plot.barh(
  legend=False, figsize=(5,5)
)
```

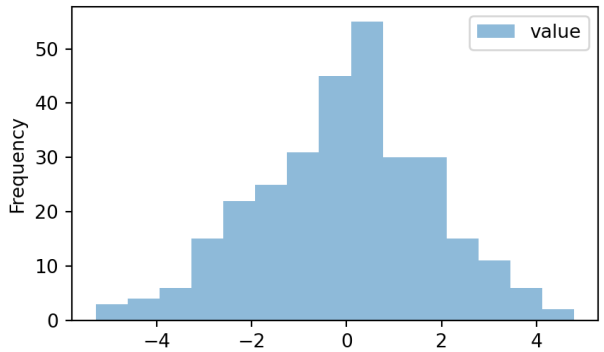


Wide vs long - histograms

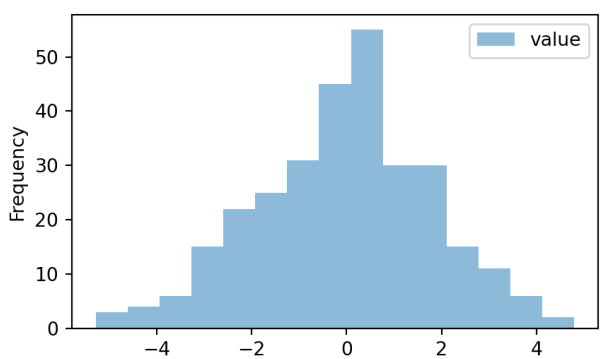
```python
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,1.5, size=100),
  "x3": np.random.normal(-1,2, size=100)
})

df.plot.hist(figsize=(5,3), alpha=0.5, bins=15)
```




```python
df_wide = df.melt()
df_wide.plot.hist(figsize=(5,3), alpha=0.5, bins
```




```python
df_wide.set_index("variable").plot.hist(
  figsize=(5,3), alpha=0.5, bins=15
)
```

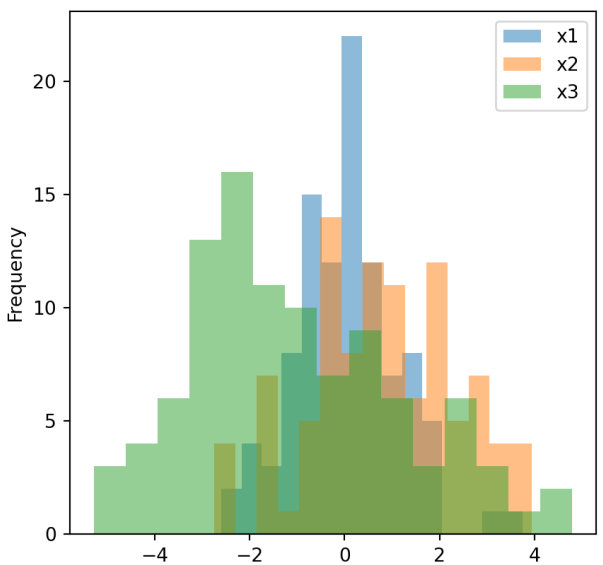


plot and groupby

```
df_wide
```

```
    variable     value
0         x1  1.471225
1         x1 -0.178315
2         x1  0.156732
3         x1  0.291983
4         x1  1.593502
..       ...       ...
295       x3 -3.202525
296       x3 -4.066616
297       x3  0.095091
298       x3 -2.446253
299       x3 -1.810505

[300 rows x 2 columns]
```

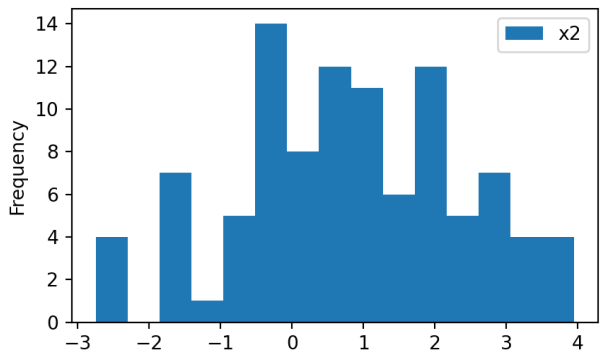
```
plt.figure(figsize=(5,5))

h = ( df_wide
  .groupby("variable")["value"]
  .plot.hist(
    alpha=0.5, legend=True, bins=15
  )
)

plt.show()
```



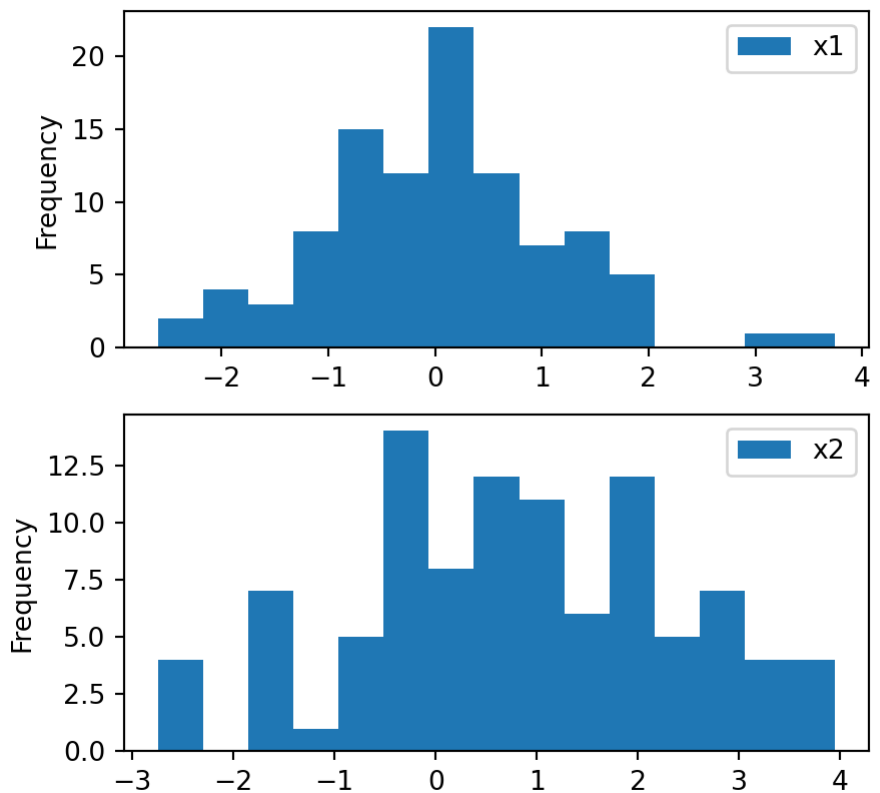
Here we are plotting Series objects hence the need to use

pandas and subplots

```
plt.figure(figsize=(5,3))
plt.subplot(211)
df[["x1"]].plot.hist(bins=15, figsize=(5,3))
plt.subplot(212)
df[["x2"]].plot.hist(bins=15, figsize=(5,3))

plt.show()
```

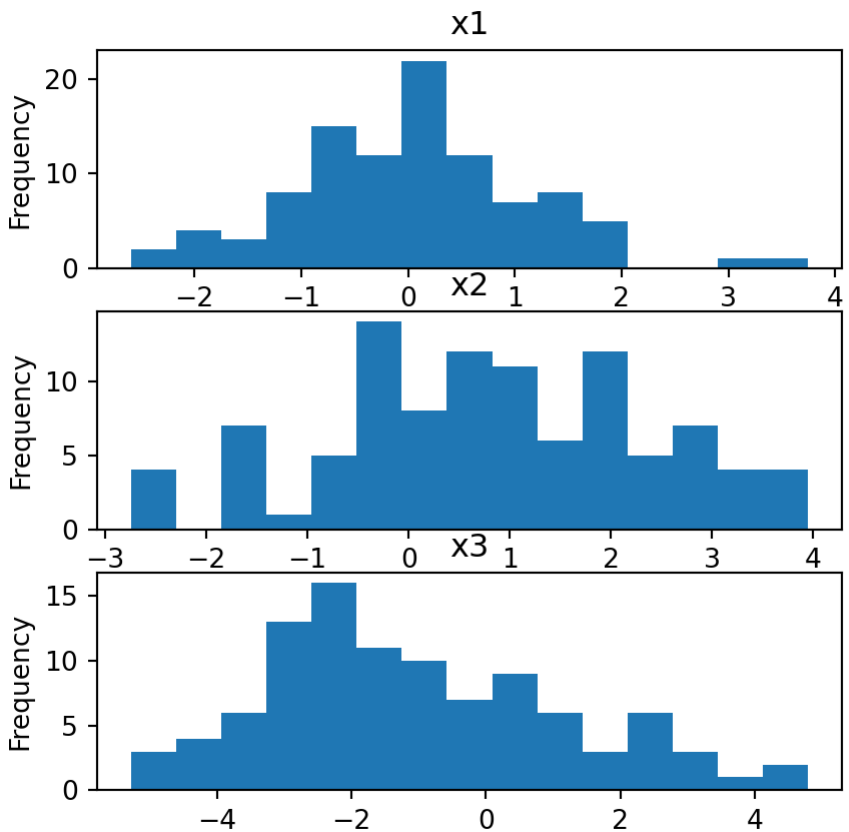



```
fig, (ax1, ax2) = plt.subplots(2,1, figsize=(5,5

df[["x1"]].plot.hist(ax = ax1, bins=15)
df[["x2"]].plot.hist(ax = ax2, bins=15)

plt.show()
```



Using by

```
1  ax = df_wide.plot.hist(bins=15, by="variable", legend=False, figsize=(5,5))
2  plt.show()
```



Note the `by` argument is not common to most of the other plotting functions - only `box` also has it.

Higher level plots - pair plot

The pandas library also provides the `plotting` submodule with several useful higher level plots,

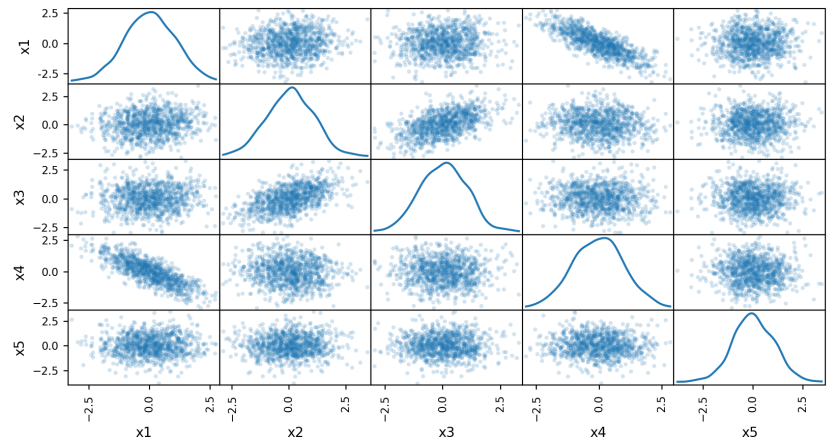
```python
cov = np.identity(5)
cov[1,2] = cov[2,1] = 0.5
cov[3,0] = cov[0,3] = -0.8

df = pd.DataFrame(
  np.random.multivariate_normal(
    mean=[0]*5, cov=cov, size=1000
  ).round(3),
  columns = ["x1","x2","x3","x4","x5"]
)

df
```

	x1	x2	x3	x4	x5
0	-0.676	-0.073	0.536	-0.481	0.829
1	0.868	-0.100	0.015	-1.404	0.466
2	0.028	-1.573	-2.680	-1.031	-0.655
3	0.435	-0.571	0.447	-0.424	-1.337
4	0.321	0.295	0.835	-0.262	-0.648
..	...	...	...	...	...
995	-0.643	1.501	0.245	0.473	0.445
996	1.482	0.903	1.271	-1.003	-0.817
997	0.001	1.001	-0.196	-0.430	-0.767
998	-2.009	0.979	-0.347	1.501	0.670
999	1.703	0.235	1.582	-0.722	0.334

[1000 rows x 5 columns]

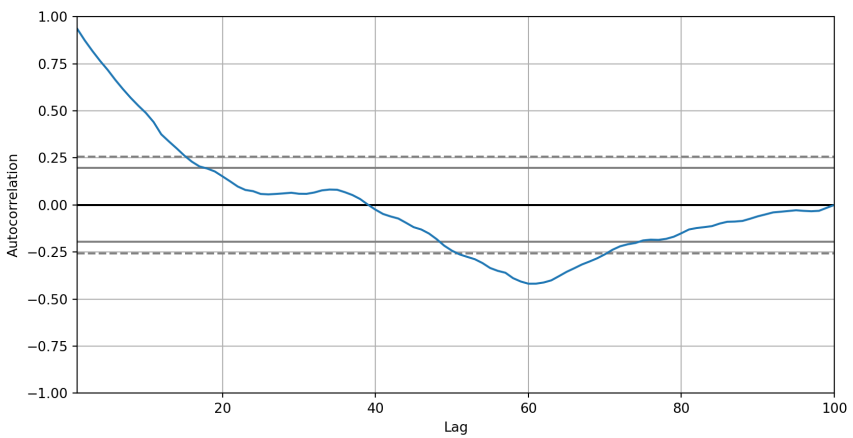
```python
pd.plotting.scatter_matrix(
  df, alpha=0.2, diagonal="kde"
)
```

```python
plt.show()
```



Autocorrelation plots

```
rw = pd.Series(
  np.cumsum( np.random.normal(size=100) ),
)

pd.plotting.autocorrelation_plot(rw)
plt.show()
```




```
wn = pd.Series(
  np.random.normal(size=100),
)

pd.plotting.autocorrelation_plot(wn)
plt.show()
```