

matplotlib

Lecture 10

Dr. Colin Rundel

matplotlib & pyplot

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

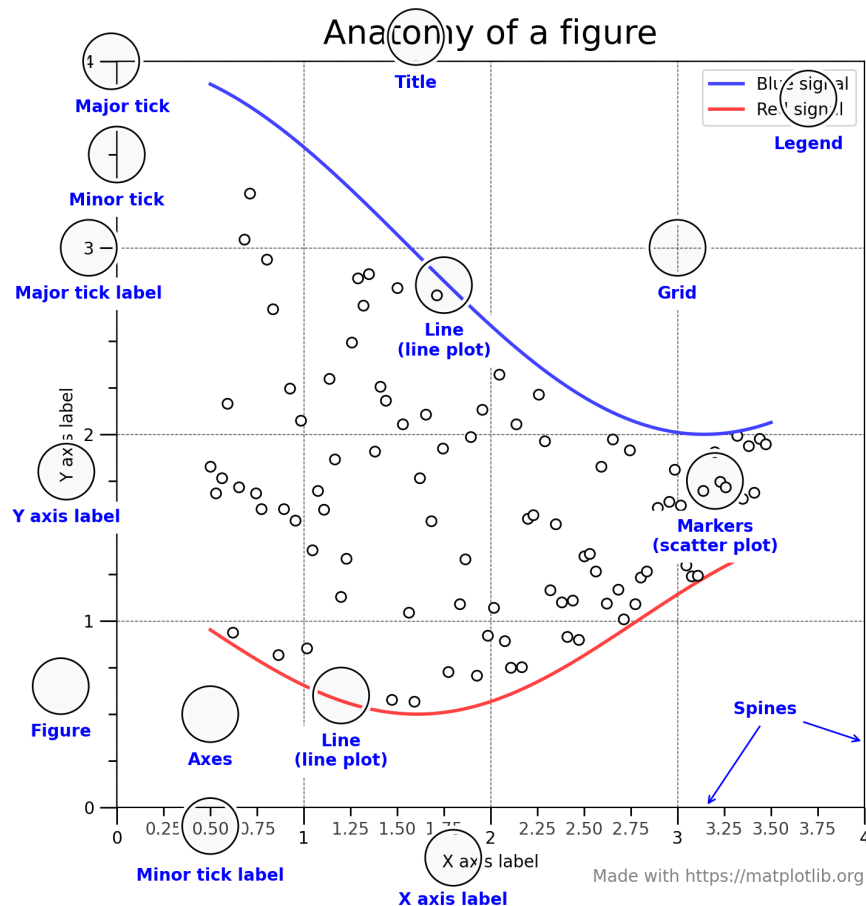
```
1 import matplotlib as mpl
2 import matplotlib.pyplot as plt
```

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.

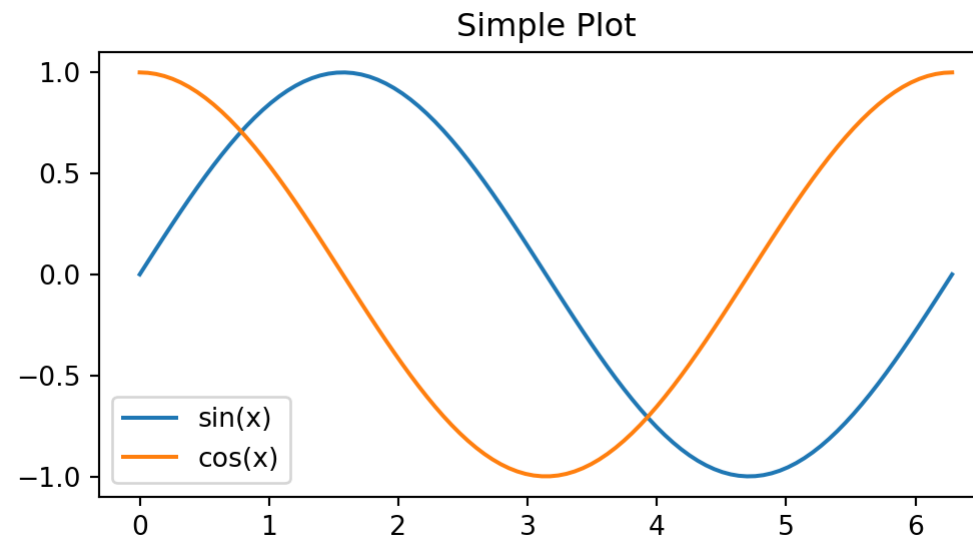
Plot anatomy



- **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.

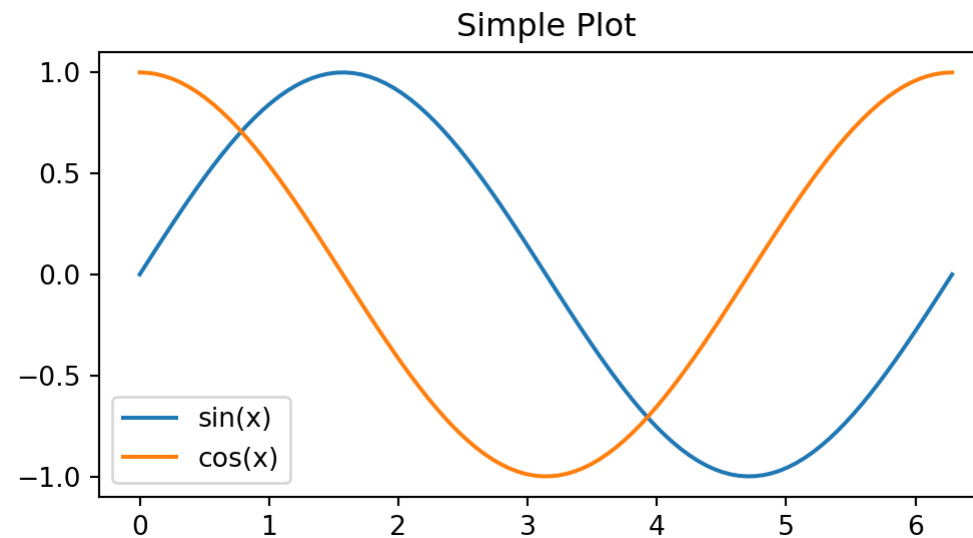
Basic plot - OO style

```
1 x = np.linspace(0, 2*np.pi, 100)
2 y1 = np.sin(x)
3 y2 = np.cos(x)
4
5 fig, ax = plt.subplots(figsize=(6, 3))
6 ax.plot(x, y1, label="sin(x)")
7 ax.plot(x, y2, label="cos(x)")
8 ax.set_title("Simple Plot")
9 ax.legend()
```



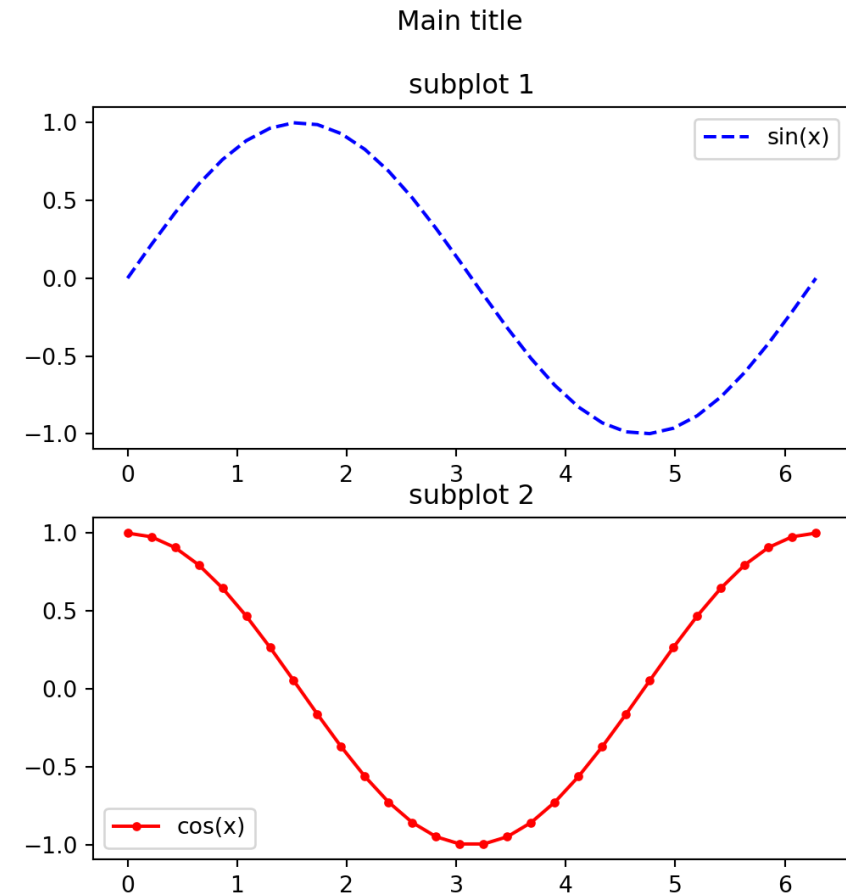
Basic plot - pyplot style

```
1 x = np.linspace(0, 2*np.pi, 100)
2 y1 = np.sin(x)
3 y2 = np.cos(x)
4
5 plt.figure(figsize=(6, 3))
6 plt.plot(x, y1, label="sin(x)")
7 plt.plot(x, y2, label="cos(x)")
8 plt.title("Simple Plot")
9 plt.legend()
```



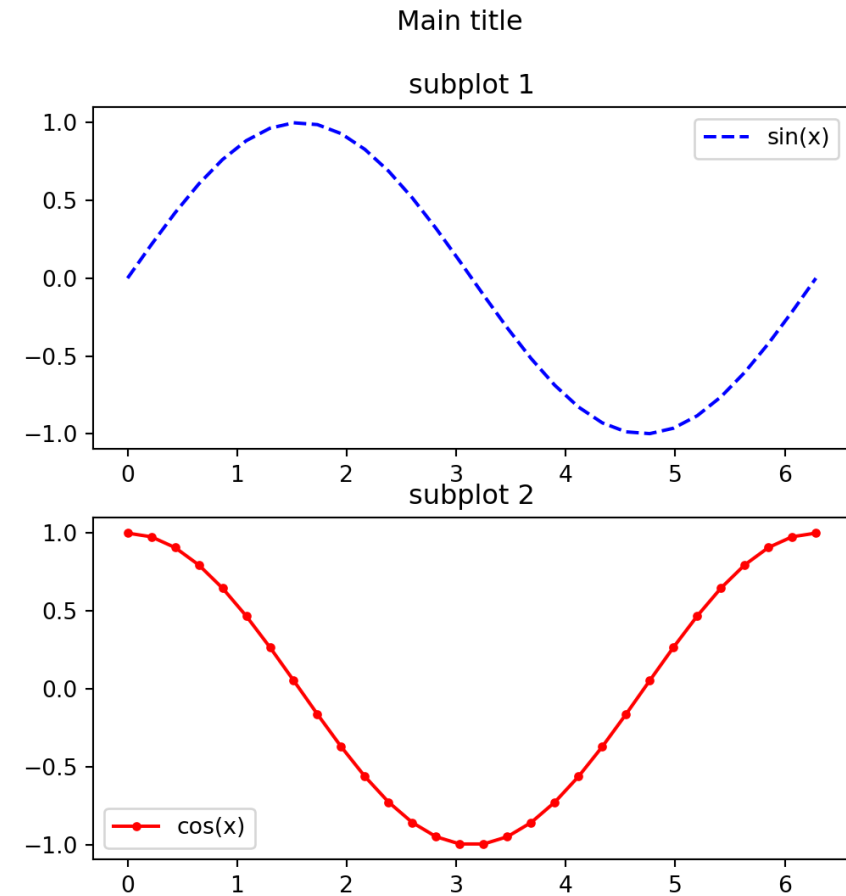
Subplots (OO)

```
1 x = np.linspace(0, 2*np.pi, 30)
2 y1 = np.sin(x)
3 y2 = np.cos(x)
4
5 fig, (ax1, ax2) = plt.subplots(
6     2, 1, figsize=(6, 6)
7 )
8
9 fig.suptitle("Main title")
10
11 ax1.plot(x, y1, "--b", label="sin(x)")
12 ax1.set_title("subplot 1")
13 ax1.legend()
14
15 ax2.plot(x, y2, "-.r", label="cos(x)")
16 ax2.set_title("subplot 2")
17 ax2.legend()
```



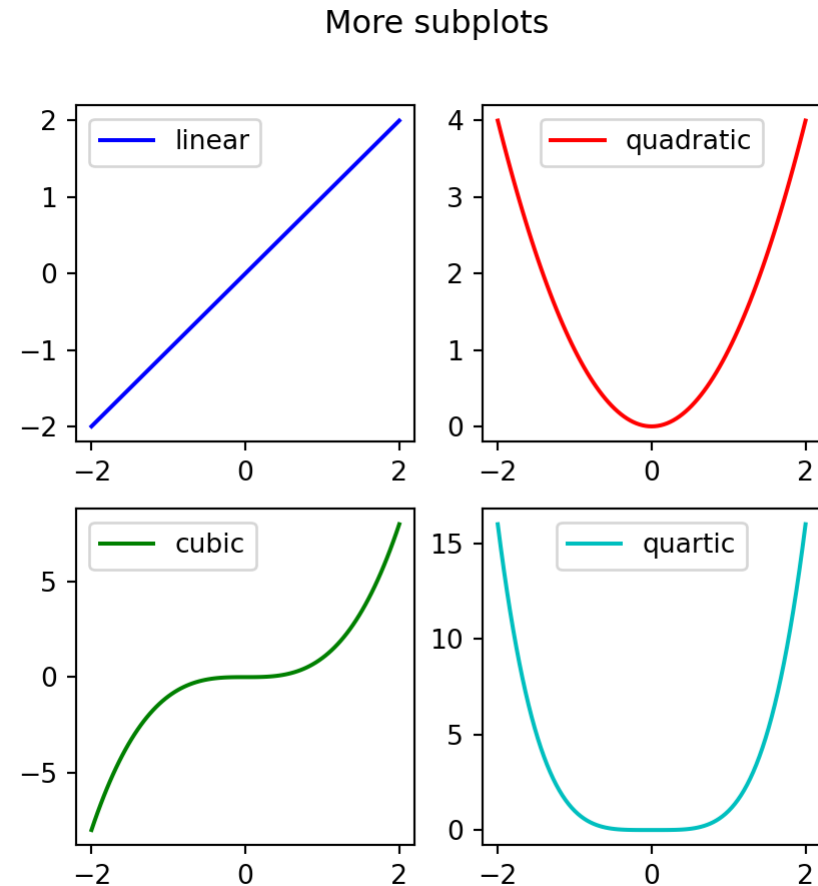
Subplots (pyplot)

```
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()
```



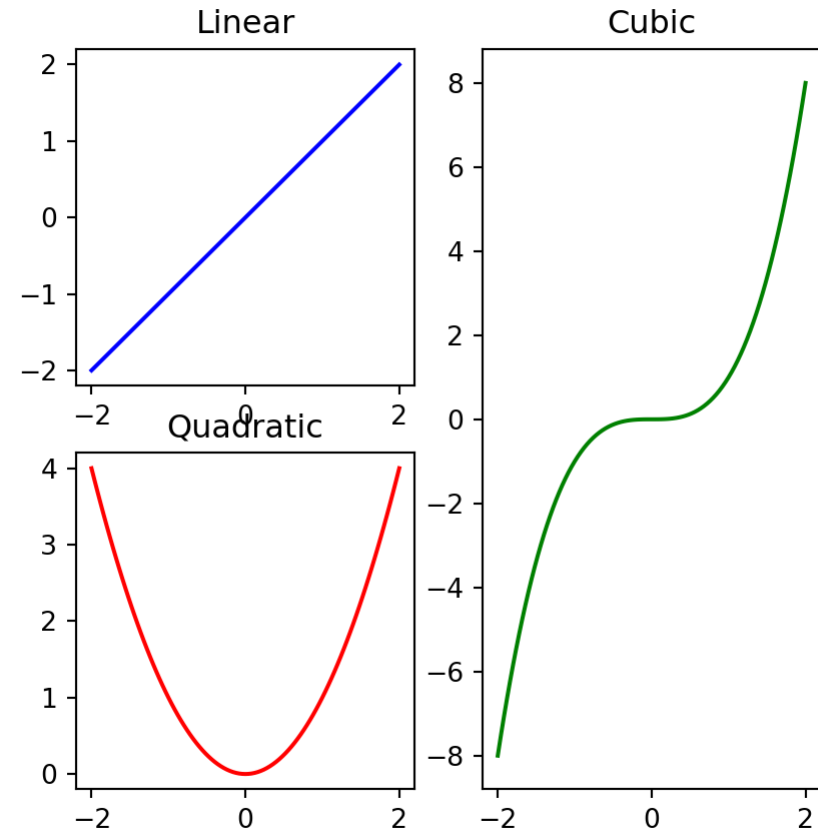
More subplots

```
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]
```



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="quadratic")
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")
```



Format strings

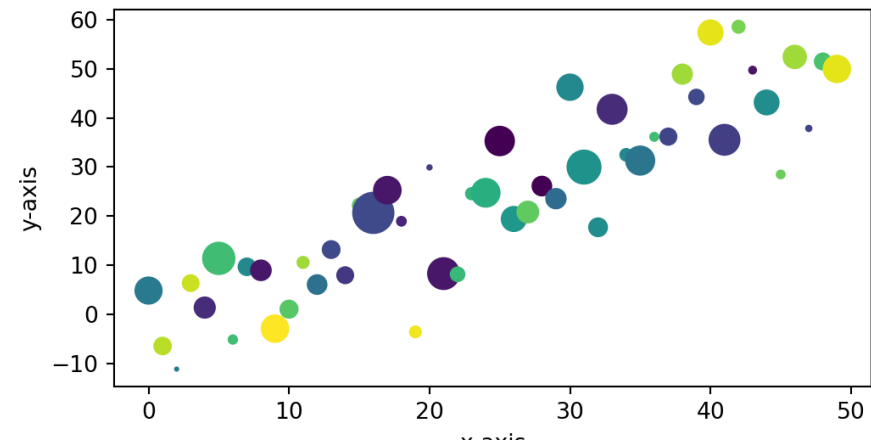
For quick formatting of plots (scatter and line) format strings are a useful shorthand, generally they use the format '`[marker][line][color]`',

character	shape	character	line style	character	color
.	point	—	solid	b	blue
,	pixel	--	dashed	g	green
o	circle	-.	dash-dot	r	red
v	triangle down	:	dotted	c	cyan
^	triangle up			m	magenta
<	triangle left			y	yellow
>	triangle right			k	black
...	+ more			w	white

Plotting data

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

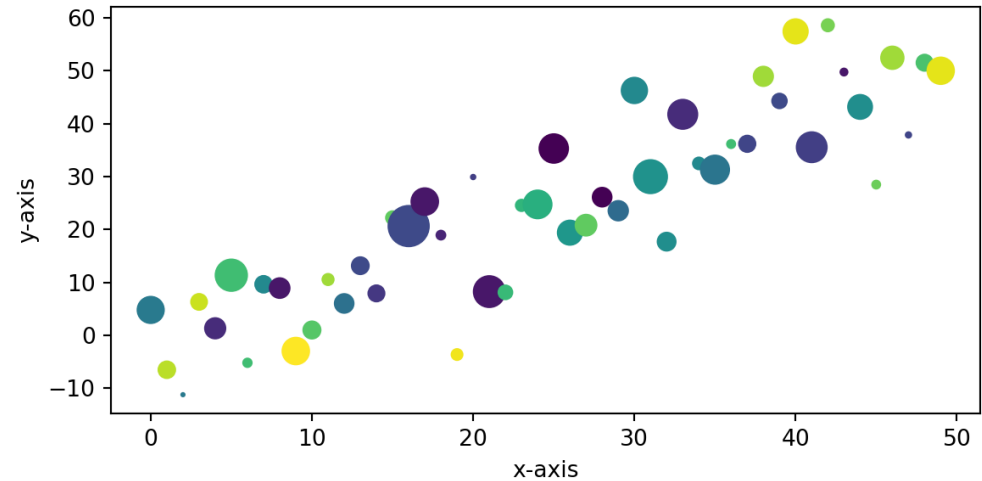
```
1 np.random.seed(19680801)
2 d = {'x': np.arange(50),
3      'color': np.random.randint(0, 50, 50),
4      'size': np.abs(np.random.randn(50)) * 100}
5 d['y'] = d['x'] + 10 * np.random.randn(50)
6
7
8 plt.figure(figsize=(6, 3))
9 plt.scatter(
10     'x', 'y', c='color', s='size',
11     data=d
12 )
13 plt.xlabel("x-axis")
14 plt.ylabel("y-axis")
15
16 plt.show()
```



Constrained layout

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

```
1 np.random.seed(19680801)
2 d = {'x': np.arange(50),
3      'color': np.random.randint(0, 50, 50),
4      'size': np.abs(np.random.randn(50)) * 100}
5 d['y'] = d['x'] + 10 * np.random.randn(50)
6
7
8 plt.figure(
9     figsize=(6, 3),
10    layout="constrained"
11 )
12 plt.scatter(
13     'x', 'y', c='color', s='size',
14     data=d
15 )
16 plt.xlabel("x-axis")
17 plt.ylabel("y-axis")
18
19 plt.show()
```

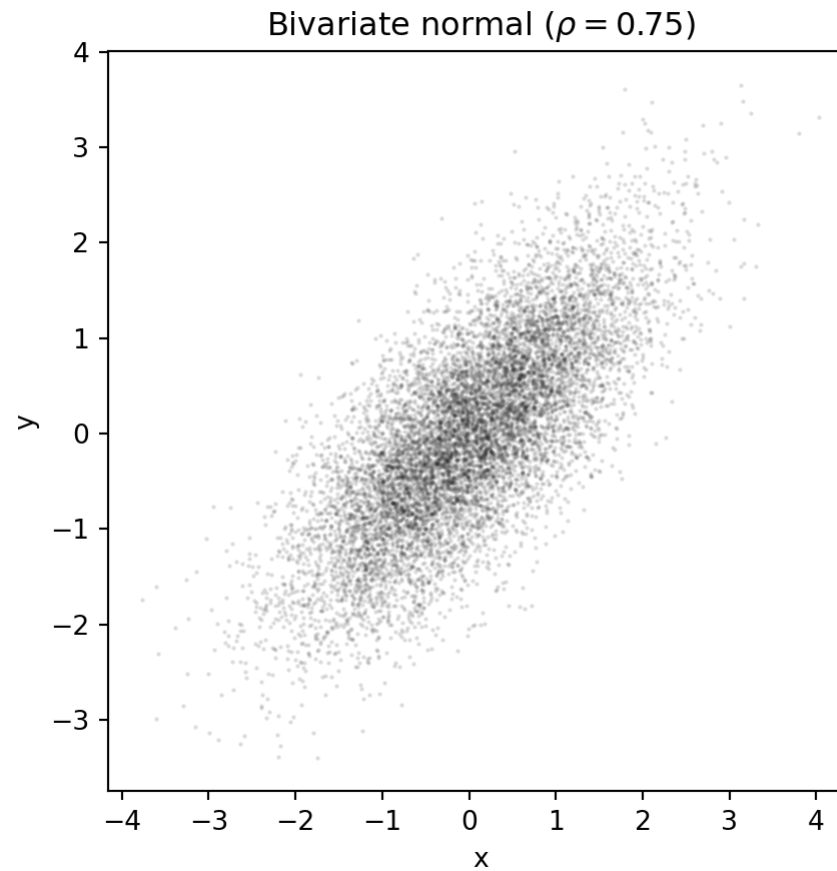


pyplot w/ pandas data

Data can also come from DataFrame objects or series,

```
1 df = pd.DataFrame({
2     "x": np.random.normal(size=10000)
3 }).assign(
4     y = lambda d: np.random.normal(0.75*d.x, np.sqrt(1-0.75**2), size=10000)
5 )
6
7 fig, ax = plt.subplots(figsize=(5,5))
8
9 ax.scatter('x', 'y', c='k', data=df, alpha=0.1, s=0.5)
10
11 ax.set_xlabel('x')
12 ax.set_ylabel('y')
13 ax.set_title("Bivariate normal ( $\rho=0.75$ )")
```

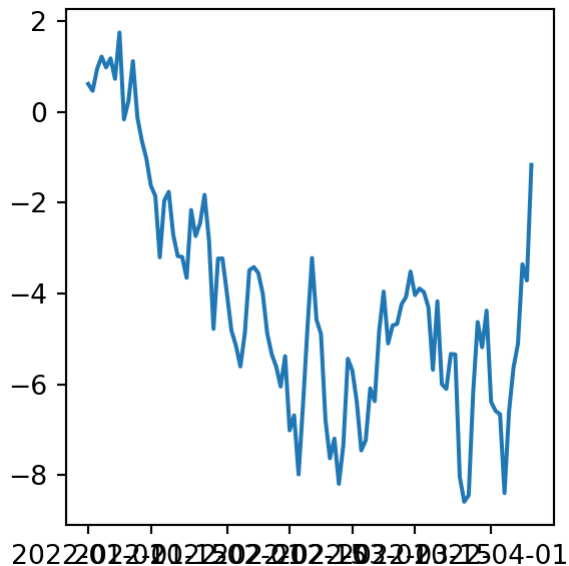
pyplot w/ pandas data



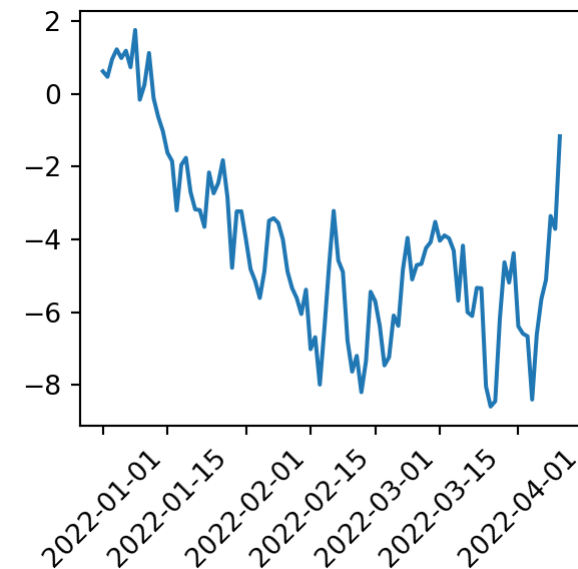
pyplot w/ pandas series

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

```
1 s = pd.Series(  
2     np.cumsum( np.random.normal(size=100) ),  
3     index = pd.date_range("2022-01-01",  
4                             periods=100, freq="D")  
5 )  
6 plt.figure(figsize=(3, 3), layout="constrained")  
7 plt.plot(s)  
8 plt.show()
```



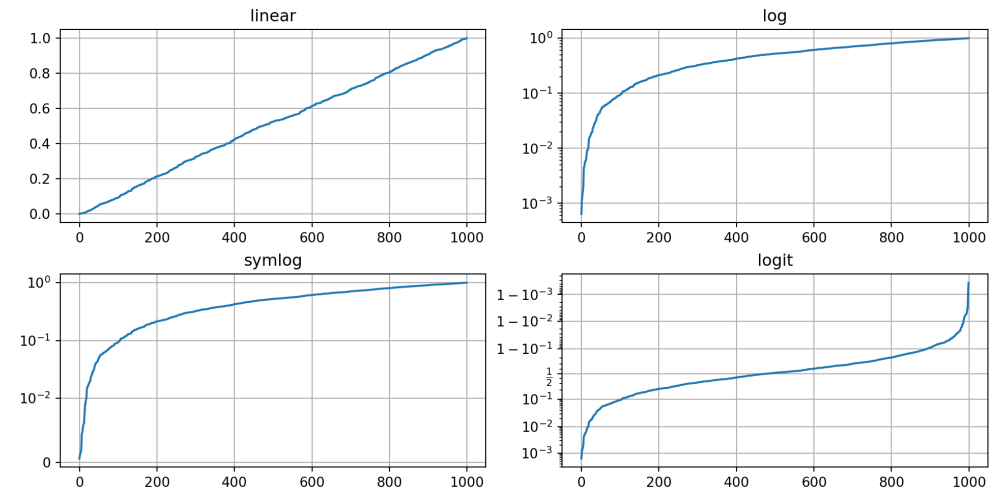
```
1 plt.figure(figsize=(3, 3), layout="constrained")  
2 plt.plot(s.index, s.values)  
3 plt.xticks(rotation=45)  
  
1 plt.show()
```



Scales

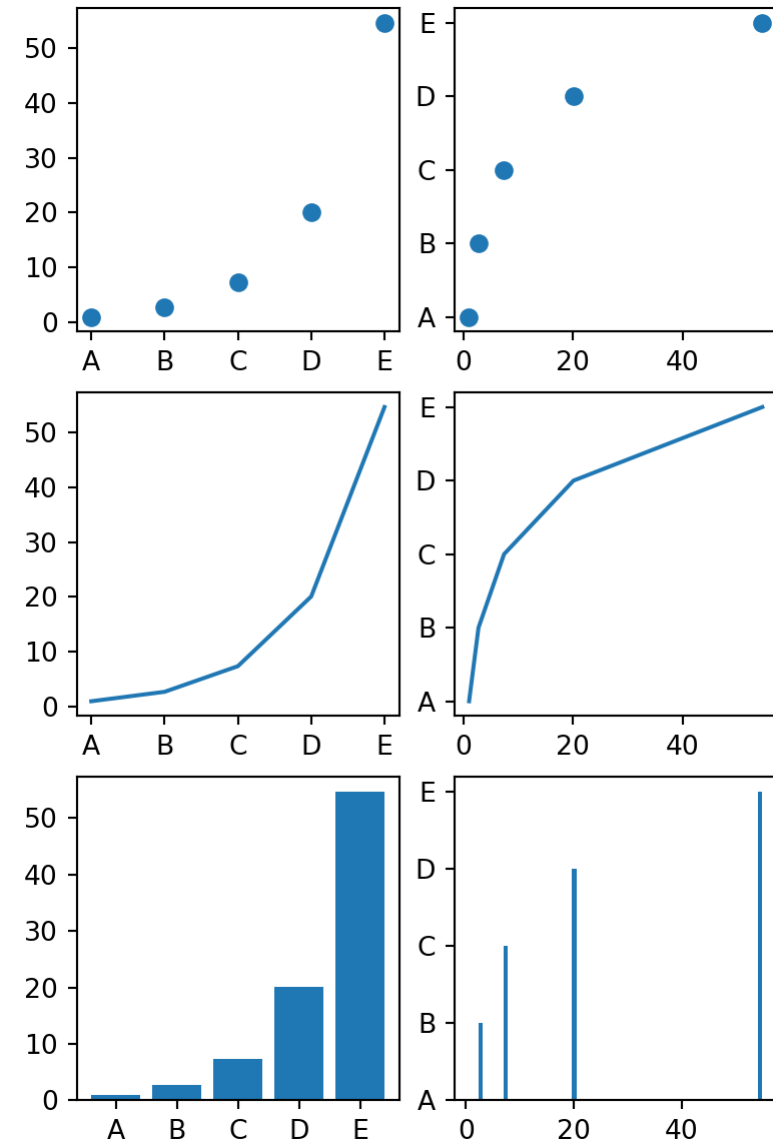
Axis scales can be changed via `plt.xscale()`, `plt.yscale()`, `ax.set_xscale()`, or `ax.set_yscale()`, supported values are “linear”, “log”, “symlog”, and “logit”.

```
1 y = np.sort( np.random.sample(size=1000) )
2 x = np.arange(len(y))
3
4 plt.figure(layout="constrained")
5
6 scales = ['linear', 'log', 'symlog', 'logit']
7 for i, scale in zip(range(4), scales):
8     plt.subplot(221+i)
9     plt.plot(x, y)
10    plt.grid(True)
11    if scale == 'symlog':
12        plt.yscale(scale, linthresh=0.01)
13    else:
14        plt.yscale(scale)
15    plt.title(scale)
16
17
18 plt.show()
```



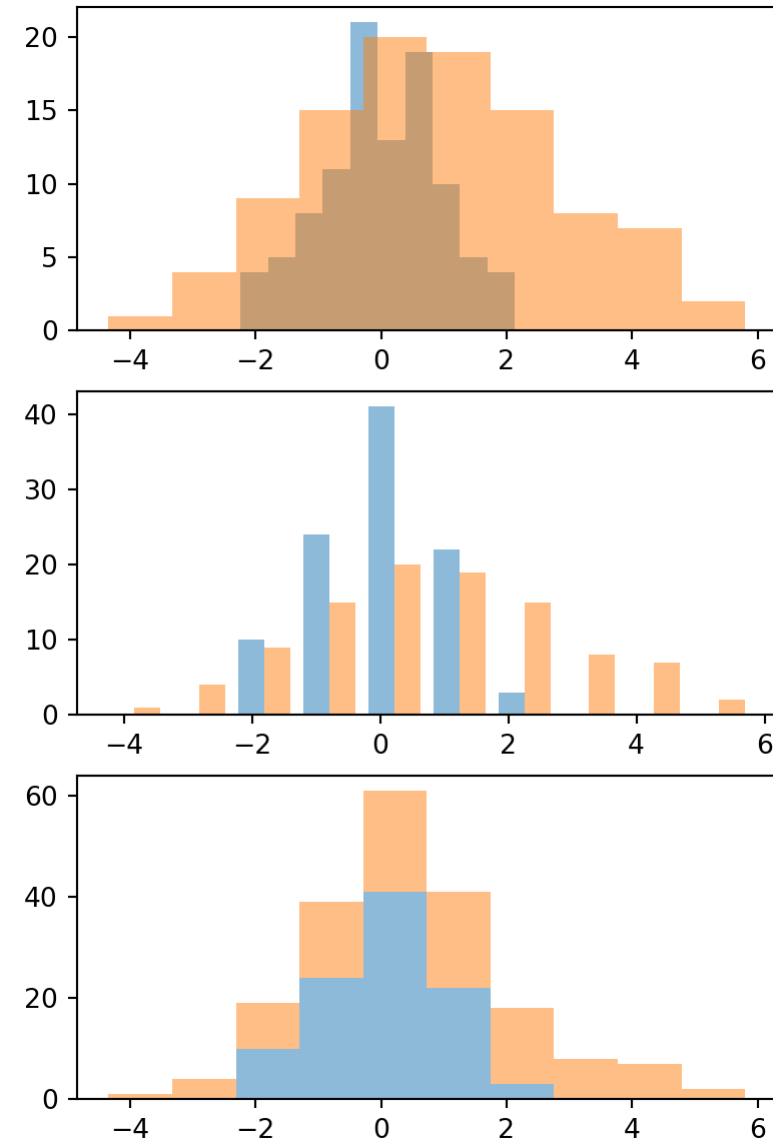
Categorical data

```
1 df = pd.DataFrame({
2     "cat": ["A", "B", "C", "D", "E"],
3     "value": np.exp(range(5))
4 })
5
6 plt.figure(figsize=(4, 6), layout="constrained")
7
8 plt.subplot(321)
9 plt.scatter("cat", "value", data=df)
10 plt.subplot(322)
11 plt.scatter("value", "cat", data=df)
12
13 plt.subplot(323)
14 plt.plot("cat", "value", data=df)
15 plt.subplot(324)
16 plt.plot("value", "cat", data=df)
17
18 plt.subplot(325)
19 b = plt.bar("cat", "value", data=df)
20 plt.subplot(326)
21 b = plt.bar("value", "cat", data=df)
22
23 plt.show()
```



Histograms

```
1 df = pd.DataFrame({
2     "x1": np.random.normal(size=100),
3     "x2": np.random.normal(1,2, size=100)
4 })
5
6 plt.figure(figsize=(4, 6), layout="constrained")
7
8 plt.subplot(311)
9 h = plt.hist("x1", bins=10, data=df, alpha=0.5)
10 h = plt.hist("x2", bins=10, data=df, alpha=0.5)
11
12 plt.subplot(312)
13 h = plt.hist(df, alpha=0.5)
14
15 plt.subplot(313)
16 h = plt.hist(df, stacked=True, alpha=0.5)
17
18 plt.show()
```



Boxplots

```
1 df = pd.DataFrame({
2     "x1": np.random.normal(size=100),
3     "x2": np.random.normal(1,2, size=100),
4     "x3": np.random.normal(-1,3, size=100)
5 }).melt()
6
7 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]

```
1 plt.figure(figsize=(4, 4), layout="constrained")
2
3 plt.boxplot("value", positions="variable", data=
```

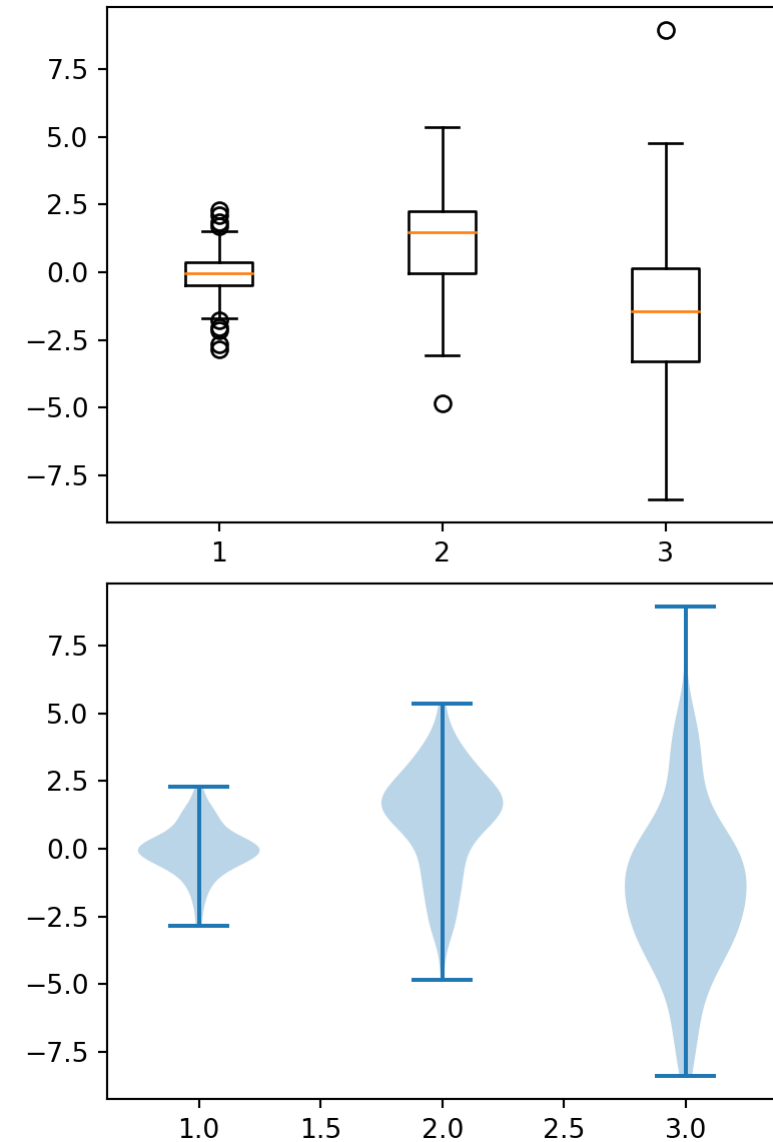
Error: ValueError: List of boxplot statistics and
`positions` values must have same the length

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

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

Boxplots (cont.)

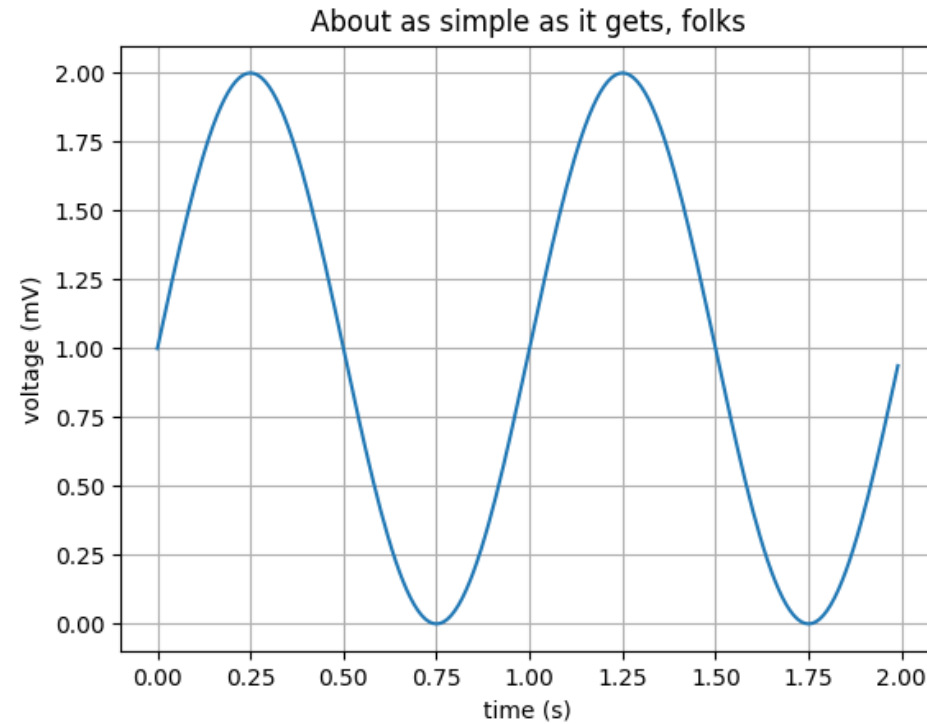
```
1 df = pd.DataFrame({
2     "x1": np.random.normal(size=100),
3     "x2": np.random.normal(1,2, size=100),
4     "x3": np.random.normal(-1,3, size=100)
5 })
6
7 plt.figure(figsize=(4, 6), layout="constrained")
8
9 plt.subplot(211)
10 b = plt.boxplot(df)
11
12 plt.subplot(212)
13 v = plt.violinplot(df)
14
15 plt.show()
```



Other Plot Types

Exercise 1

To the best of your ability recreate the following plot,

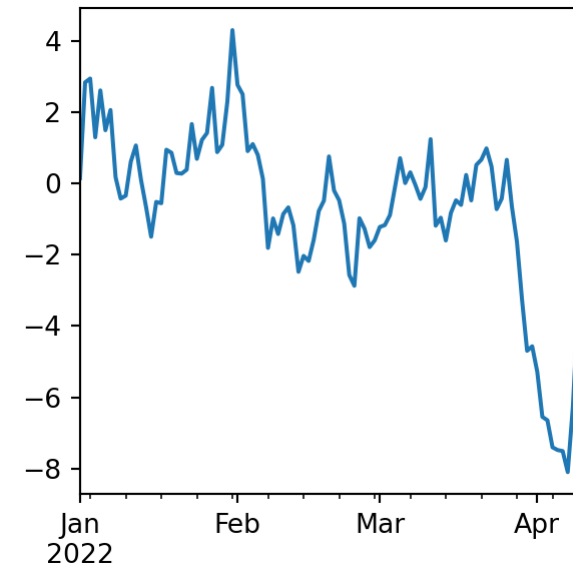


Plotting with pandas

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.

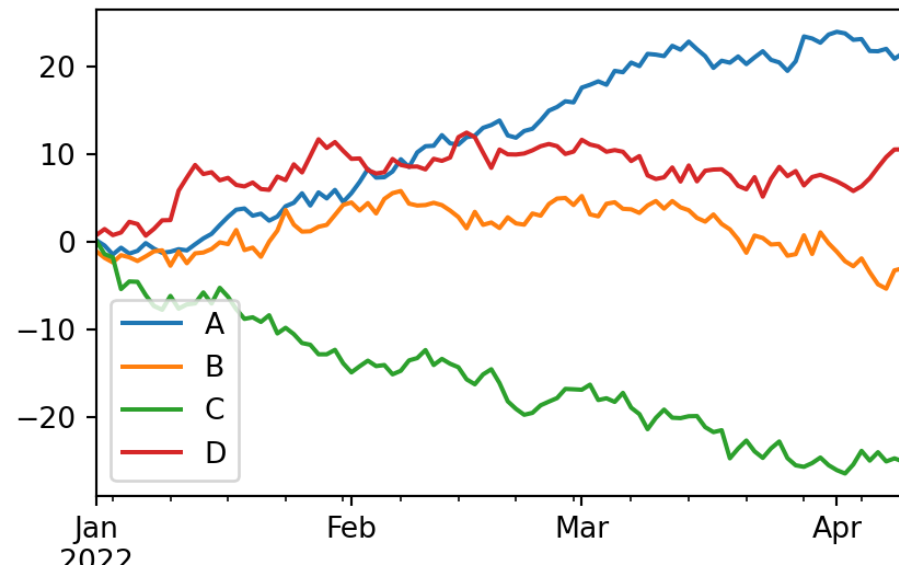
```
1 s = pd.Series(  
2     np.cumsum( np.random.normal(size=100) ),  
3     index = pd.date_range("2022-01-01",  
4                             periods=100, freq="D")  
5 )  
6  
7 plt.figure(figsize=(3,3), layout="constrained")  
8 s.plot()  
9 plt.show()
```



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

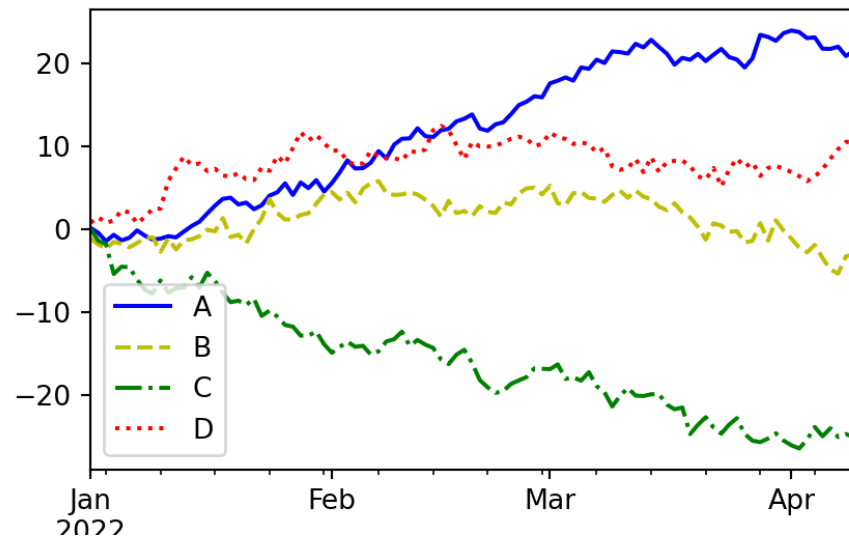
DataFrame plot

```
1 df = pd.DataFrame(  
2     np.cumsum( np.random.normal(size=(100,4)), axis=0),  
3     index = pd.date_range("2022-01-01", periods=100, freq="D"),  
4     columns = list("ABCD")  
5 )  
6  
7 plt.figure(layout="constrained")  
8 df.plot(figsize=(5,3))  
9 plt.show()
```



DataFrame line styles

```
1 df.plot(  
2     figsize=(5,3),  
3     style = {  
4         "A": "-b",  
5         "B": "--y",  
6         "C": "-.g",  
7         "D": ":r"  
8     }  
9 )
```



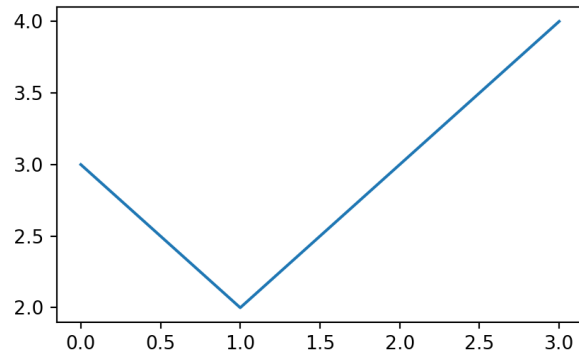
Sta 663 - Spring 2023

DataFrame plot - categorical

```

1 df = pd.DataFrame({
2     "x": list("ABCD"),
3     "y": np.random.poisson(lam=2, size=4)
4 })
5
6 df.plot(figsize=(5,3), legend=False)

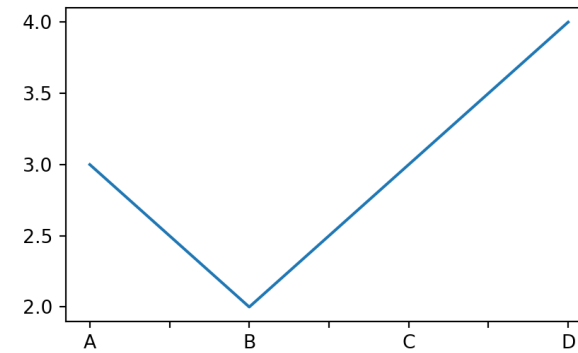
```



```

1 df.set_index("x").plot(figsize=(5,3), legend=False)

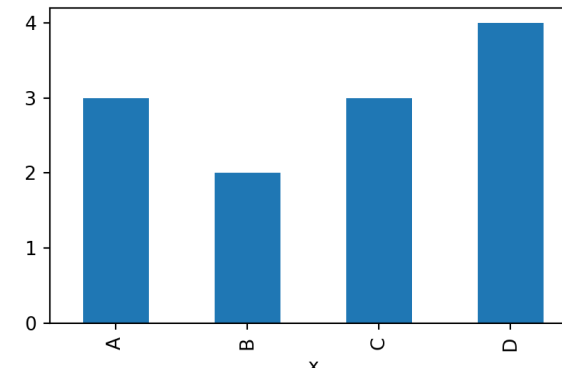
```



```

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

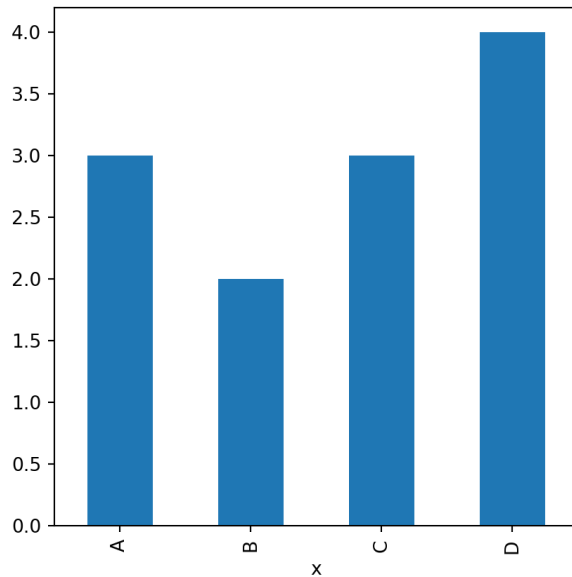
```



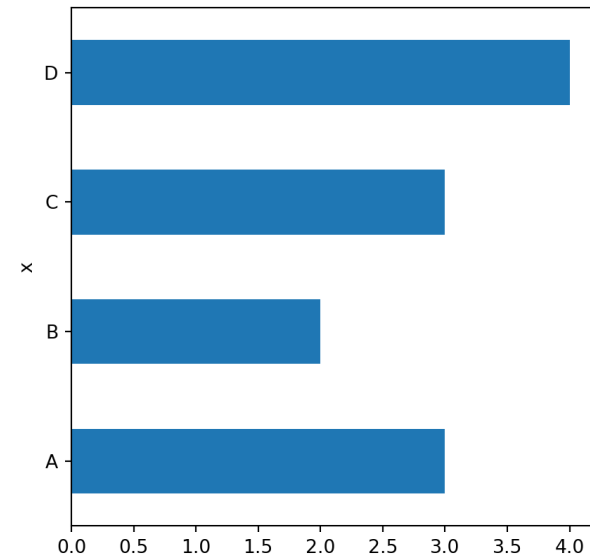
Other plot types

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

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



```
1 df.set_index("x").plot.barh(  
2     legend=False, figsize=(5,5)  
3 )
```

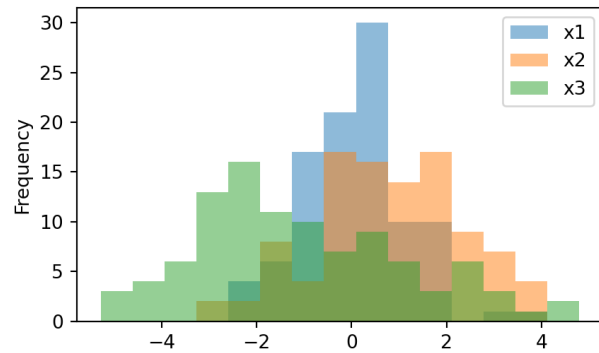


Wide vs long - histograms

```

1 df = pd.DataFrame({
2     "x1": np.random.normal(size=100),
3     "x2": np.random.normal(1,1.5, size=100),
4     "x3": np.random.normal(-1,2, size=100)
5 })
6
7 df.plot.hist(figsize=(5,3), alpha=0.5, bins=15)

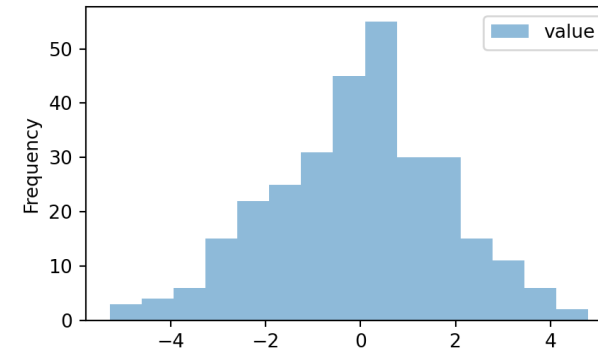
```



```

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

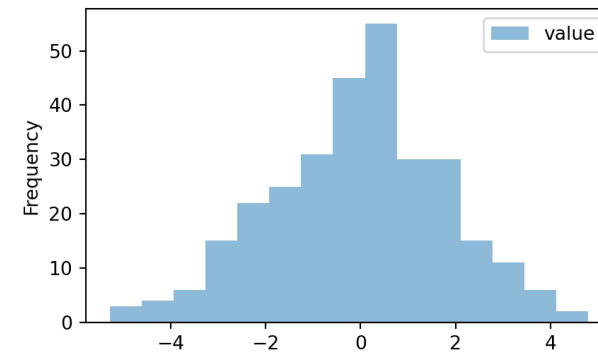
```



```

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

```



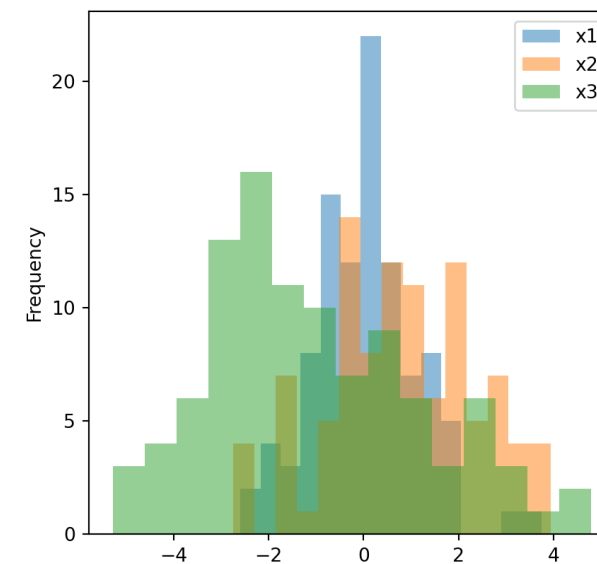
plot and groupby


```
1 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]
```

```
1 plt.figure(figsize=(5,5))
2
3 h = ( df_wide
4       .groupby("variable")["value"]
5       .plot.hist(
6         alpha=0.5, legend=True, bins=15
7       )
8     )
9
10 plt.show()
```



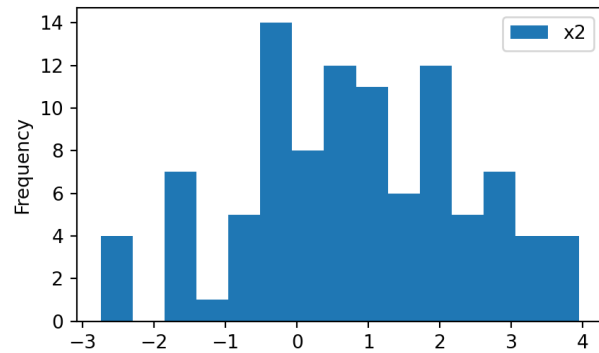
Here we are plotting Series objects hence the need to use

pandas and subplots

```

1 plt.figure(figsize=(5,3))
2 plt.subplot(211)
3 df[["x1"]].plot.hist(bins=15, figsize=(5,3))
4 plt.subplot(212)
5 df[["x2"]].plot.hist(bins=15, figsize=(5,3))
6
7 plt.show()

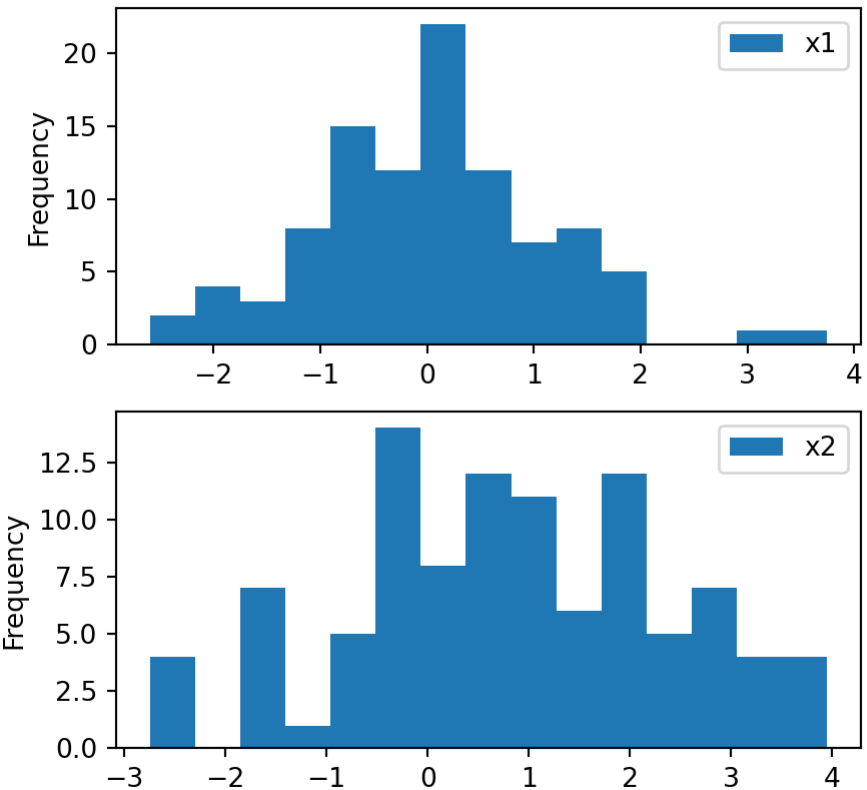
```



```

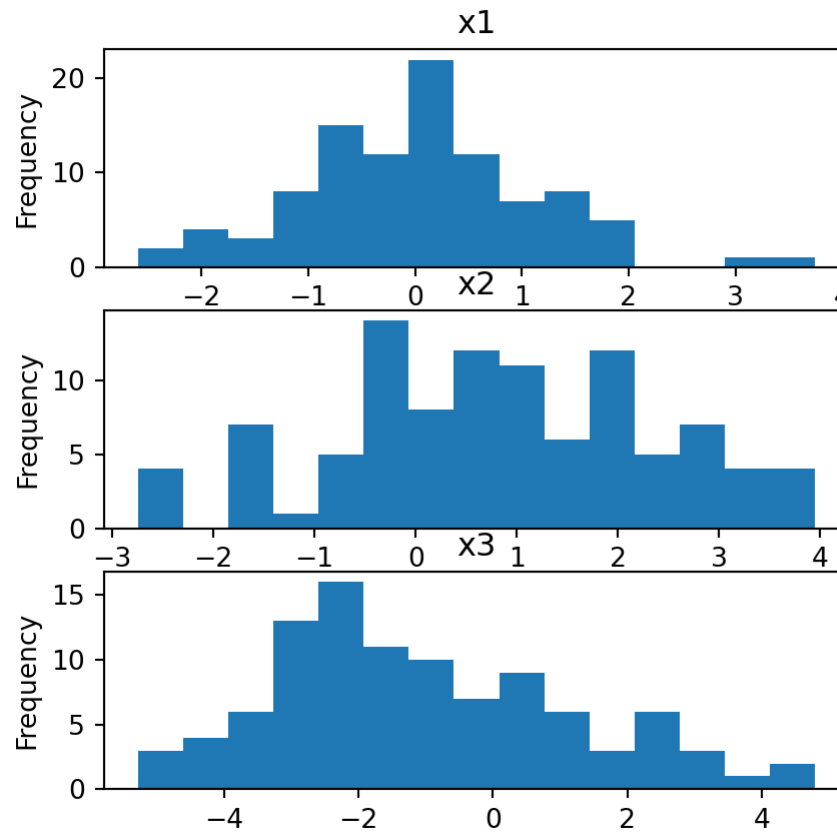
1 fig, (ax1, ax2) = plt.subplots(2,1, figsize=(5,5))
2
3 df[["x1"]].plot.hist(ax = ax1, bins=15)
4 df[["x2"]].plot.hist(ax = ax2, bins=15)
5
6 plt.show()

```



Using by

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



Higher level plots - pair plot

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

```

1 cov = np.identity(5)
2 cov[1,2] = cov[2,1] = 0.5
3 cov[3,0] = cov[0,3] = -0.8
4
5 df = pd.DataFrame(
6     np.random.multivariate_normal(
7         mean=[0]*5, cov=cov, size=1000
8     ).round(3),
9     columns = ["x1", "x2", "x3", "x4", "x5"]
10 )
11
12 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

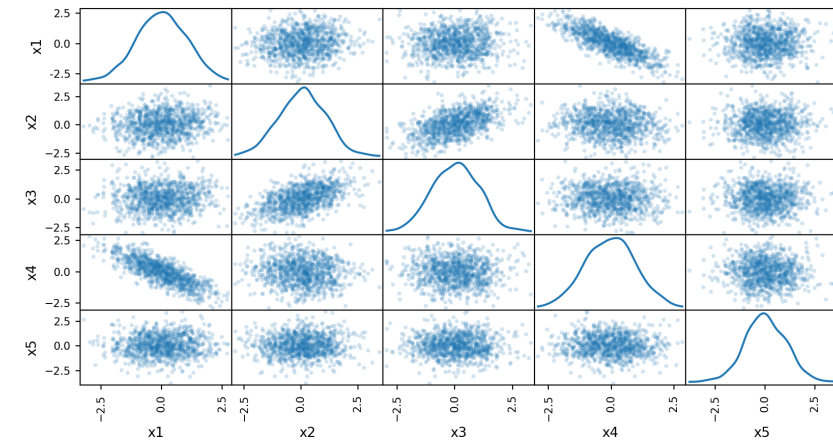
[1000 rows x 5 columns]

```

1 pd.plotting.scatter_matrix(
2     df, alpha=0.2, diagonal="kde"
3 )

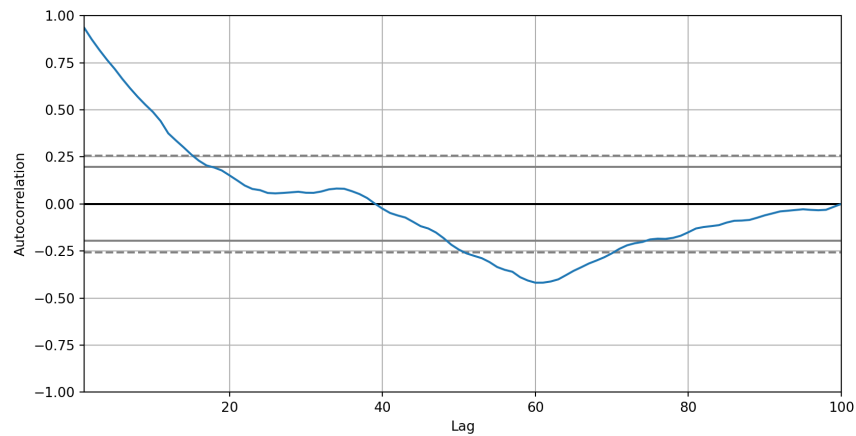
```

```
1 plt.show()
```



Autocorrelation plots

```
1 rw = pd.Series(  
2     np.cumsum( np.random.normal(size=100) ),  
3 )  
4  
5 pd.plotting.autocorrelation_plot(rw)  
6 plt.show()
```



```
1 wn = pd.Series(  
2     np.random.normal(size=100),  
3 )  
4  
5 pd.plotting.autocorrelation_plot(wn)  
6 plt.show()
```

