Berlin School of Economics and Law
Prof. Dr. Natalie Packham

Summer term 2021
LV-Nr. 425093

Computational Finance and FinTech

Data Visualisation

Contents

3 Data Visualisation

- Further reading: **Py4Fi, Chapter 7**
- We look at the `matplotlib` plotting libraries.
- Interactive 2D plotting is available with `plotly`.

- More information on `plotly` and some introductory examples can be found in the **Python for Finance** book.

3.1 Static 2D Plotting

- Some standard imports and customatisations:

[1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt  # main plotting subpackage
plt.style.use('seaborn')  # sets the plotting style
mpl.rcParams['font.family'] = 'serif'  # set the font to be serif in all plots
%matplotlib inline
# ensures that output of plotting command is displayed within notebook
```

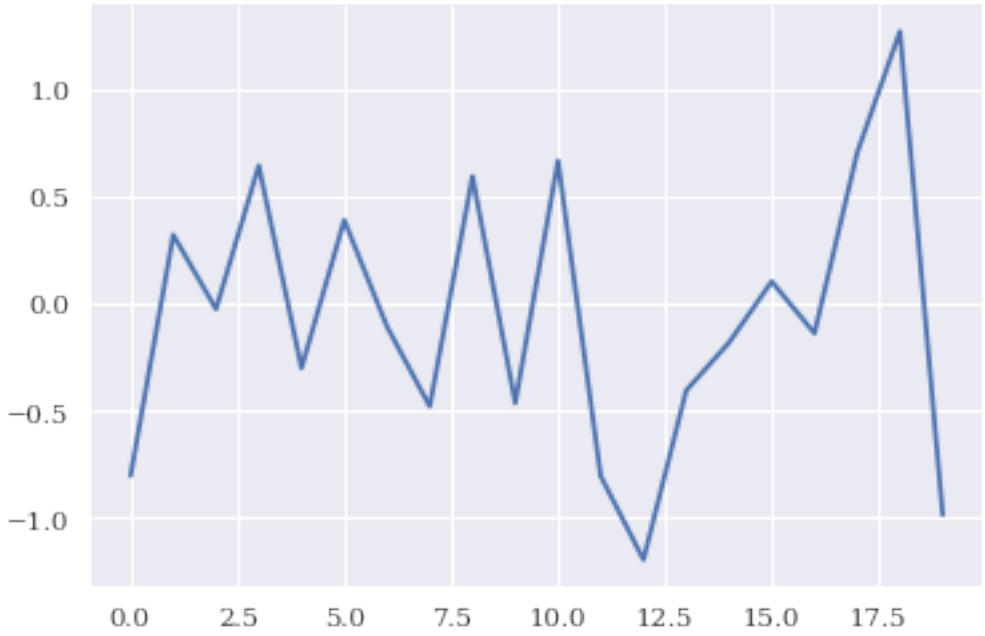
[2]:
```
mpl.__version__  # the version of matplotlib
```

[2]:
```
'3.1.3'
```

Simple plotting

- The standard (and powerful) plotting method is `plt.plot()`.
- It takes as basic argument lists or arrays of x values and y values

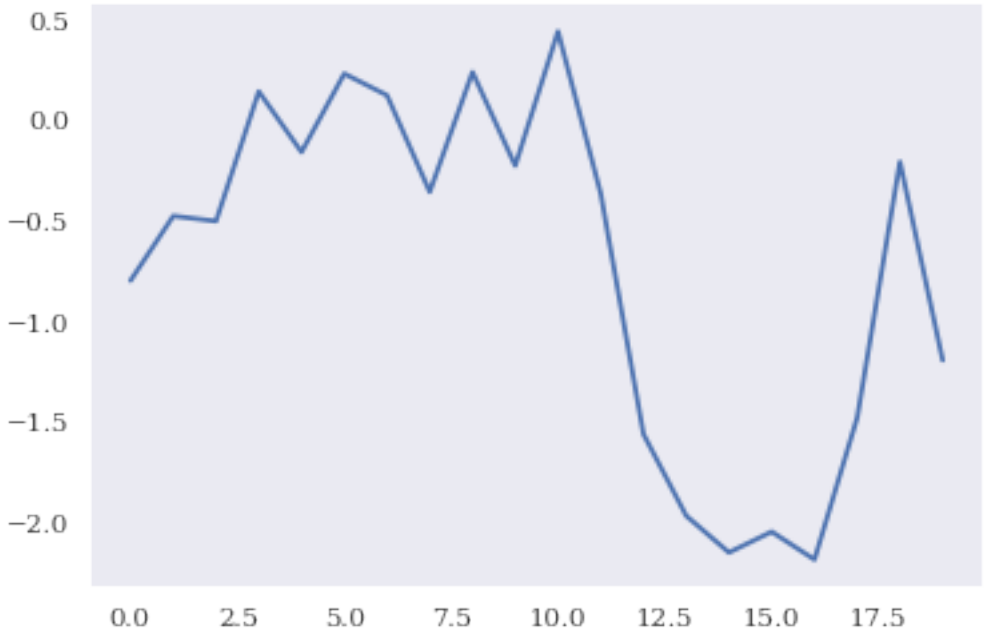
[3]:
```
import numpy as np
np.random.seed(1000)
y=np.random.standard_normal(20) # draw some random numbers
x=np.arange(len(y)) # fix the x axis
plt.plot(x,y); # plot y against x
```



Simple plotting

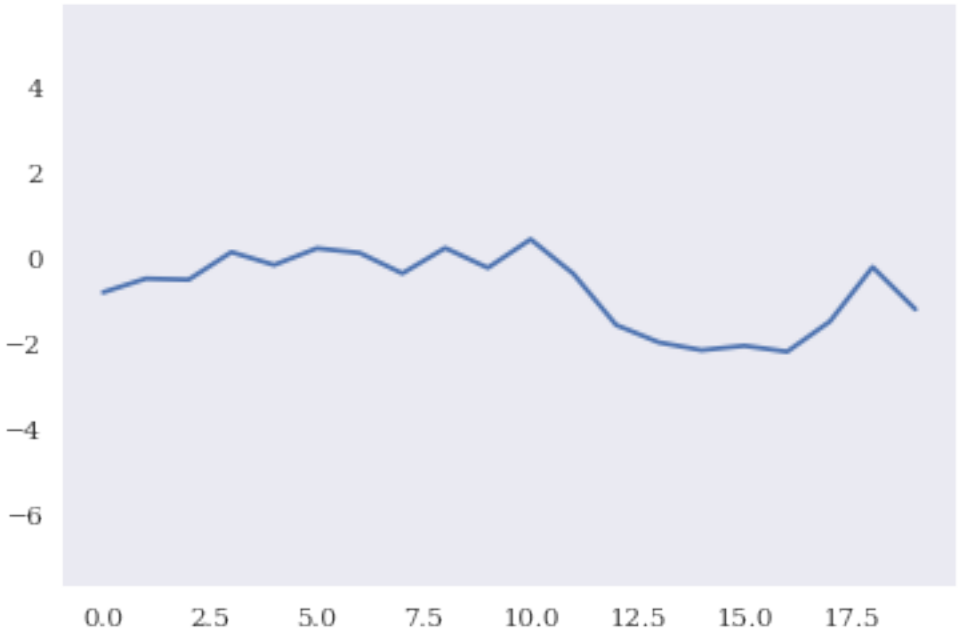
- A number functions are available to customise the plot:

[4]:
```
plt.plot(y.cumsum())
plt.grid(False) #
```



[5]:
```
plt.plot(y.cumsum())
plt.grid(False)
```

```
plt.axis('equal');
```



Simple plotting

- Options for `plt.axis()`:

Table 7-1. Options for plt.axis()

Parameter	Description
Empty	Returns current axis limits
`off`	Turns axis lines and labels off
`equal`	Leads to equal scaling
`scaled`	Produces equal scaling via dimension changes
`tight`	Makes all data visible (tightens limits)
`image`	Makes all data visible (with data limits)
`[xmin, xmax, ymin, ymax]`	Sets limits to given (`list` of) values

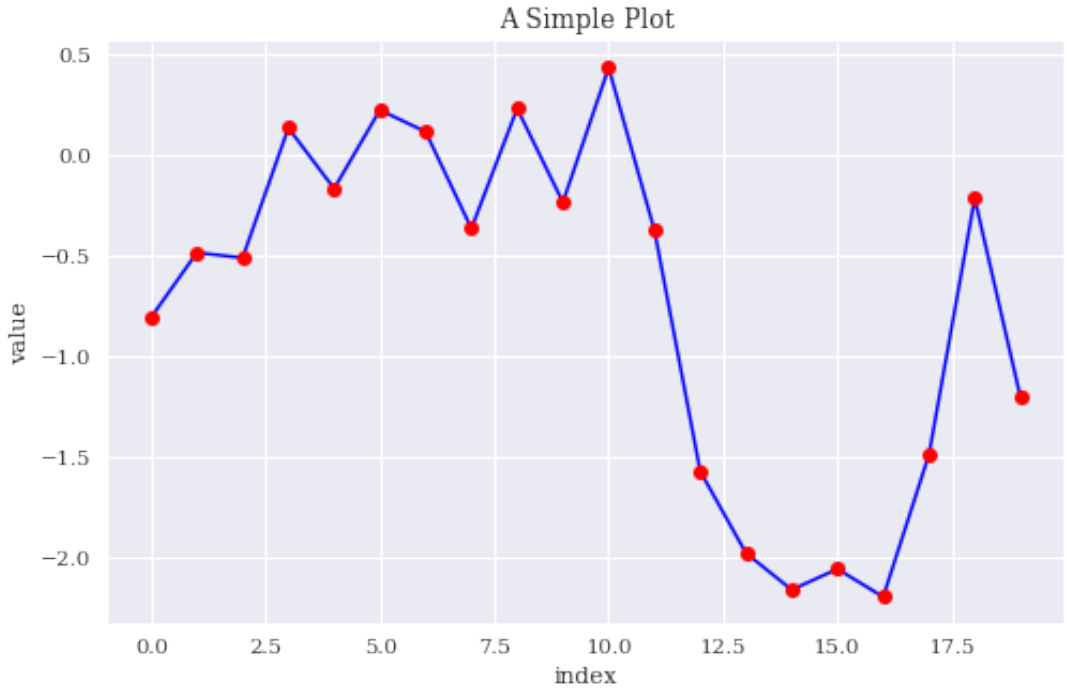
Options for plt.axis

Source: Python for Finance, 2nd ed.

Simple plotting

- Further customisations:

[6]:
```
plt.figure(figsize=(8, 5))  # increase size of figure
plt.plot(y.cumsum(), 'b', lw=1.5)  # plot data in blue with a line width of 1.5
plt.plot(y.cumsum(), 'ro')  # plot the data points as red dots
plt.xlabel('index')  # label of x-axis
plt.ylabel('value')  # label of y-axis
plt.title('A Simple Plot');  # plot title
```



Simple plotting

- Standard colour abbreviations:

Character	Colour
b	blue
g	green
r	red
c	cyan
m	magenta
y	yellow
k	black
w	white

Simple plotting

- Line styles:

Character	Colour
'-'	solid line
'--'	dashed line
'-.	dash-dot line

Character	Colour
':'	dotted line

Simple plotting

- Some marker styles:

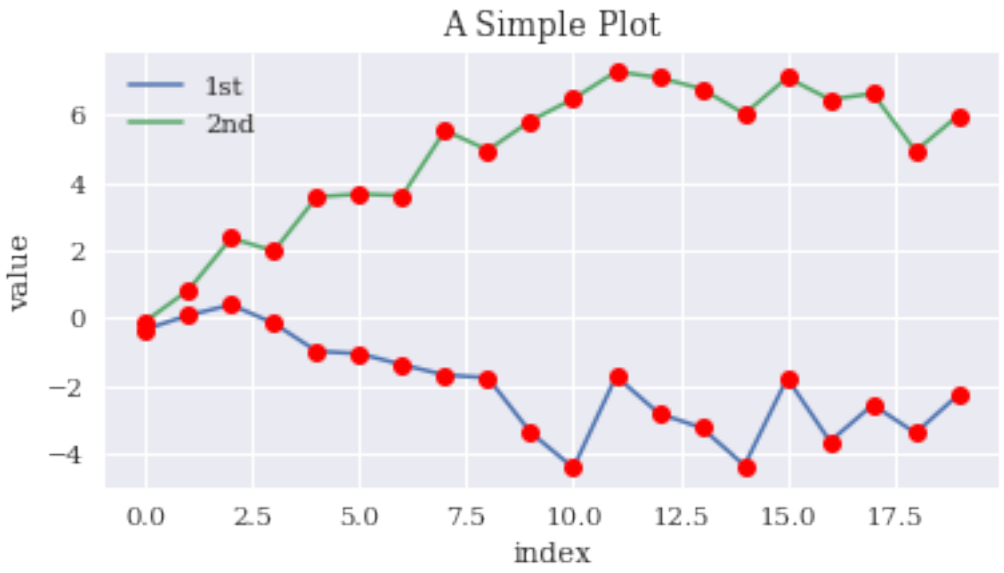
Character	Colour
'.'	point
','	pixel
'o'	circle
'v'	triangle down
'^'	triangle up
'<'	triangle left
'>'	triangle right
'*'	star
'h'	hexagon

- More marker styles are found here and here.

Plotting several data sets

- If the data are arranged in a multi-dimensional array, then `plot()` will automatically plot the columns separately:

[7]:
```
y = np.random.standard_normal((20, 2)).cumsum(axis=0)
plt.figure(figsize=(6, 3))
plt.plot(y[:, 0], lw=1.5, label='1st')  # define a label to be used in the legend
plt.plot(y[:, 1], lw=1.5, label='2nd')
plt.plot(y, 'ro')
plt.legend(loc=0)  # add a legend, consult the legend help to find out about␣
↪locations
plt.xlabel('index')
plt.ylabel('value')
plt.title('A Simple Plot');
```



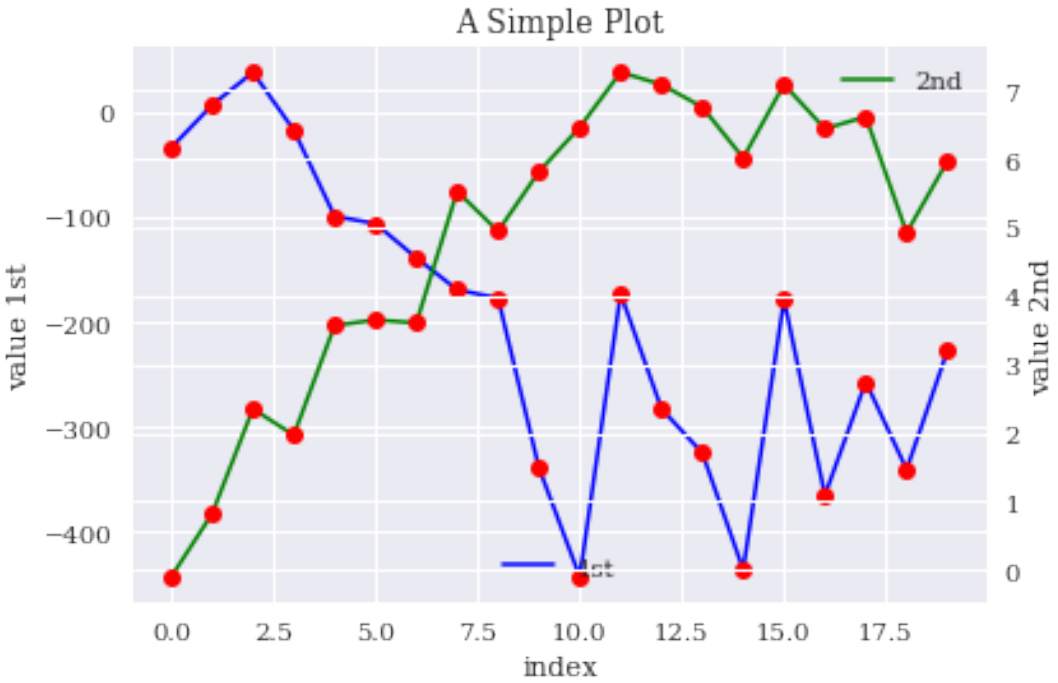
Subplots

- `plt.subplots()` is a powerful method to either combine several plots with separate axes or to produce separate plots.
- In the first example, the plots overlay each other:

Subplots

[8]:
```
y[:,0] = y[:,0] * 100
```

[9]:
```
fig, ax1 = plt.subplots()  # defines  figure and axis objects
plt.plot(y[:, 0], 'b', lw=1.5, label='1st')
plt.plot(y[:, 0], 'ro')
plt.legend(loc=8)
plt.xlabel('index')
plt.ylabel('value 1st')
plt.title('A Simple Plot')
ax2 = ax1.twinx()  # create a second y-axis object
plt.plot(y[:, 1], 'g', lw=1.5, label='2nd')
plt.plot(y[:, 1], 'ro')
plt.legend(loc=0)
plt.ylabel('value 2nd');
```

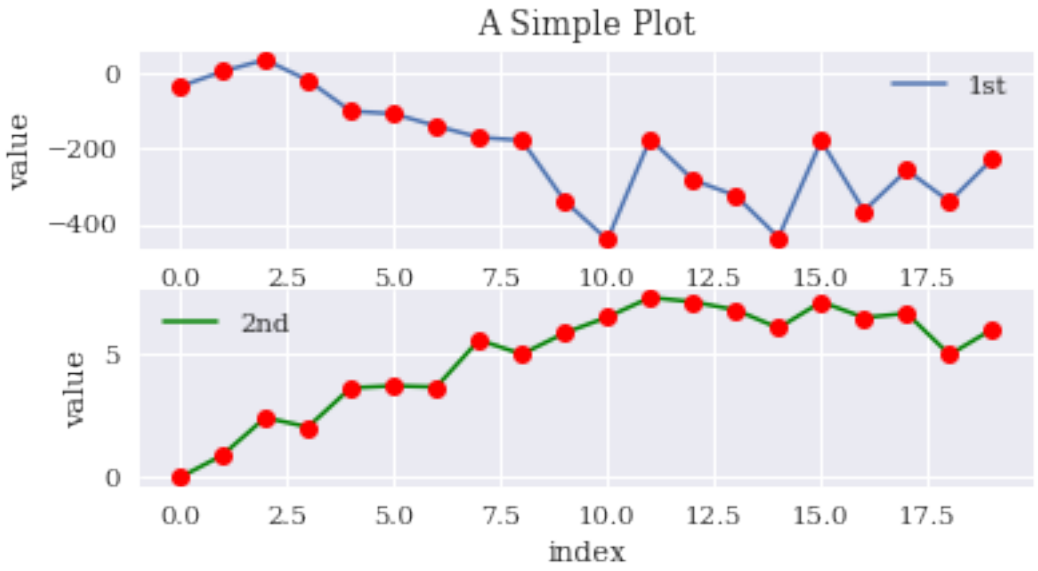


Subplots

- The second example creates two separate plots.
- The main argument to `subplot()` is a 3-digit integer describing the position of the subplot.
- The integers refer to `nrows`, `ncols` and `index`, where `index` starts at 1 in the upper left corner and increases to the right.

Subplots

[10]:
```
plt.figure(figsize=(6, 3))
plt.subplot(211)  # defines the upper plot in a figure with two rows and one column
plt.plot(y[:, 0], lw=1.5, label='1st')
plt.plot(y[:, 0], 'ro')
plt.legend(loc=0)
plt.ylabel('value')
plt.title('A Simple Plot')
plt.subplot(212)  # defines the lower plot
plt.plot(y[:, 1], 'g', lw=1.5, label='2nd')
plt.plot(y[:, 1], 'ro')
plt.legend(loc=0)
plt.xlabel('index')
plt.ylabel('value');
```

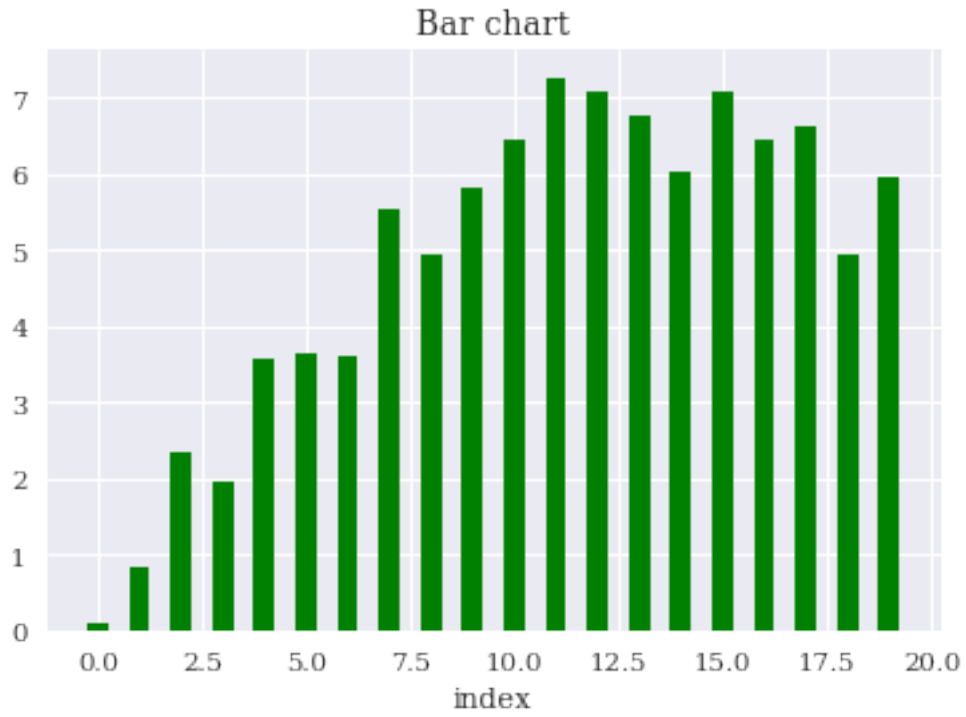


Other plot styles

- The following examples introduce bar charts, scatter plots, histograms and boxplots

Bar chart

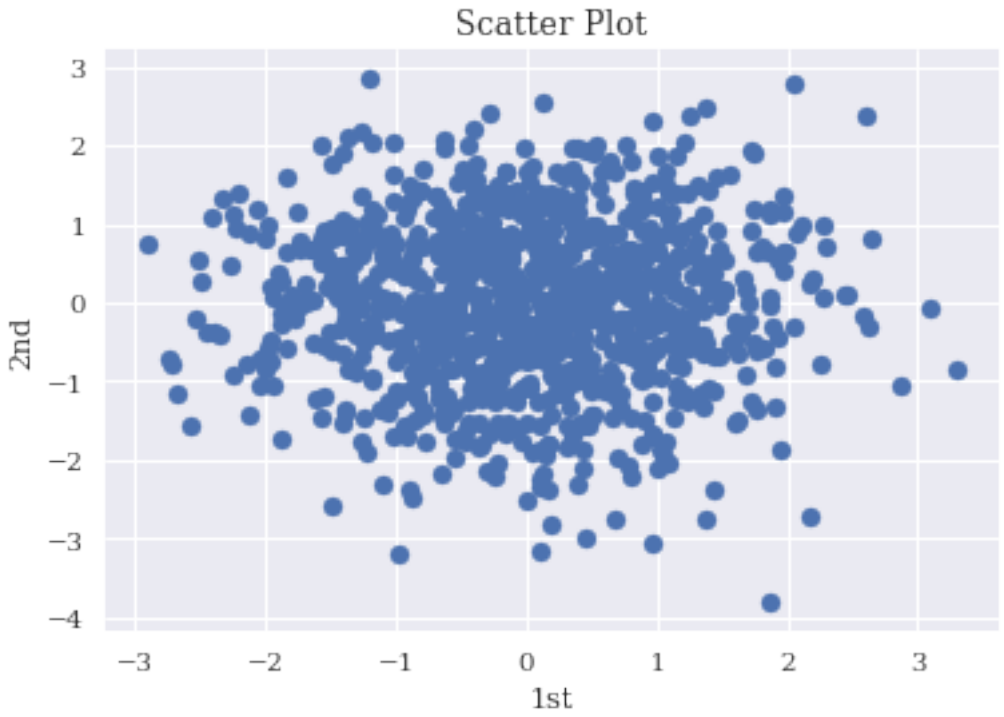
[11]:
```
plt.bar(np.arange(len(y)), abs(y[:, 1]), width=0.5,
        color='g')
plt.xlabel('index')
plt.title('Bar chart');
```



Scatter plot

[12]:
```
y = np.random.standard_normal((1000, 2))
```

[13]:
```
plt.scatter(y[:, 0], y[:, 1], marker='o')
plt.xlabel('1st')
plt.ylabel('2nd')
plt.title('Scatter Plot');
```

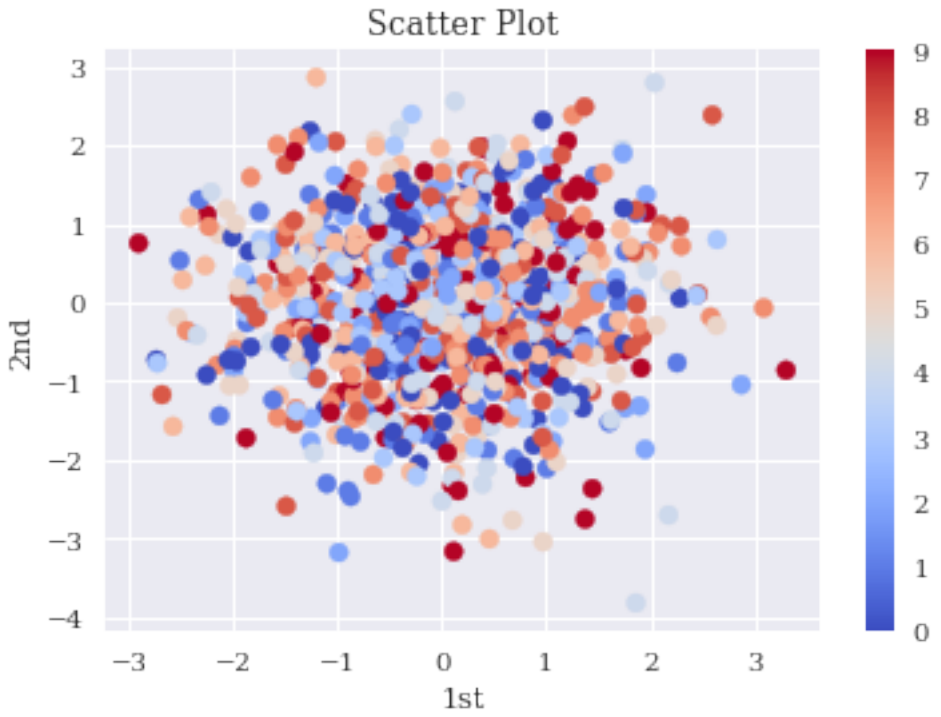


Scatter plot

- Adding a third dimension via a colour map:

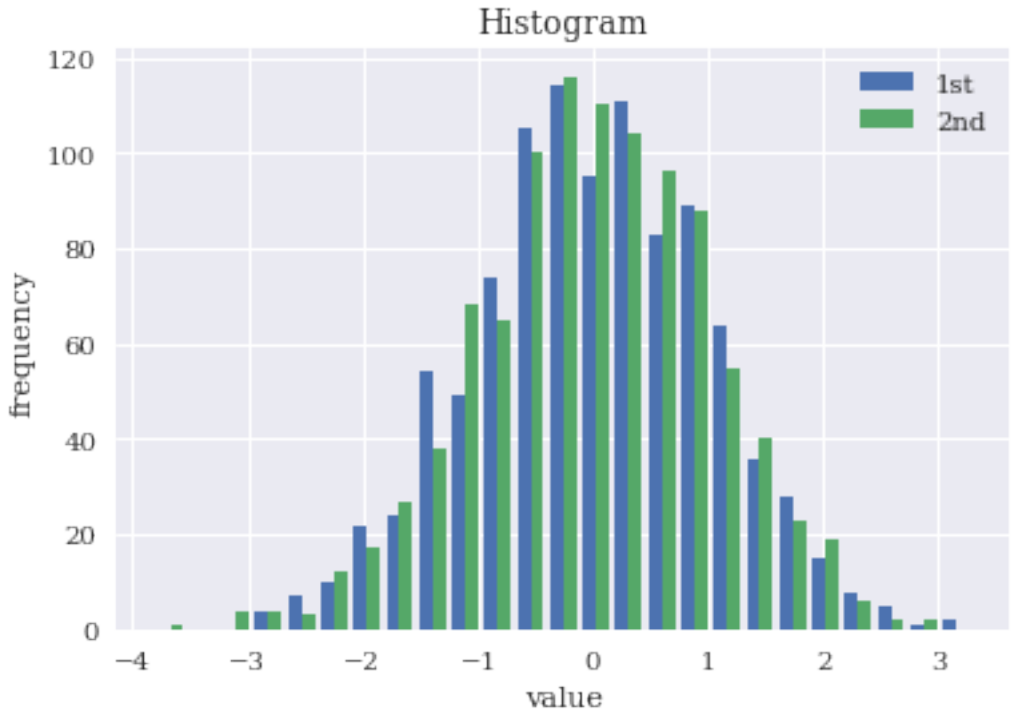
[14]:
```
c = np.random.randint(0, 10, len(y))
```

[15]:
```
plt.scatter(y[:, 0], y[:, 1],
            c=c,
            cmap='coolwarm',
            marker='o')
plt.colorbar()
plt.xlabel('1st')
plt.ylabel('2nd')
plt.title('Scatter Plot');
```



Histogram

[16]:
```
plt.hist(y, label=['1st', '2nd'], bins=25)
plt.legend(loc=0)
plt.xlabel('value')
plt.ylabel('frequency')
plt.title('Histogram');
```



Histogram

- Parameters for `plt.hist()`:

Parameter	Description
`x`	`list` object(s), `ndarray` object
`bins`	Number of bins
`range`	Lower and upper range of bins
`normed`	Norming such that integral value is 1
`weights`	Weights for every value in `x`
`cumulative`	Every bin contains the counts of the lower bins
`histtype`	Options (strings): `bar`, `barstacked`, `step`, `stepfilled`
`align`	Options (strings): `left`, `mid`, `right`
`orientation`	Options (strings): `horizontal`, `vertical`
`rwidth`	Relative width of the bars

Parameters for plt.hist

Parameter	Description
`log`	Log scale
`color`	Color per data set (array-like)
`label`	String or sequence of strings for labels
`stacked`	Stacks multiple data sets

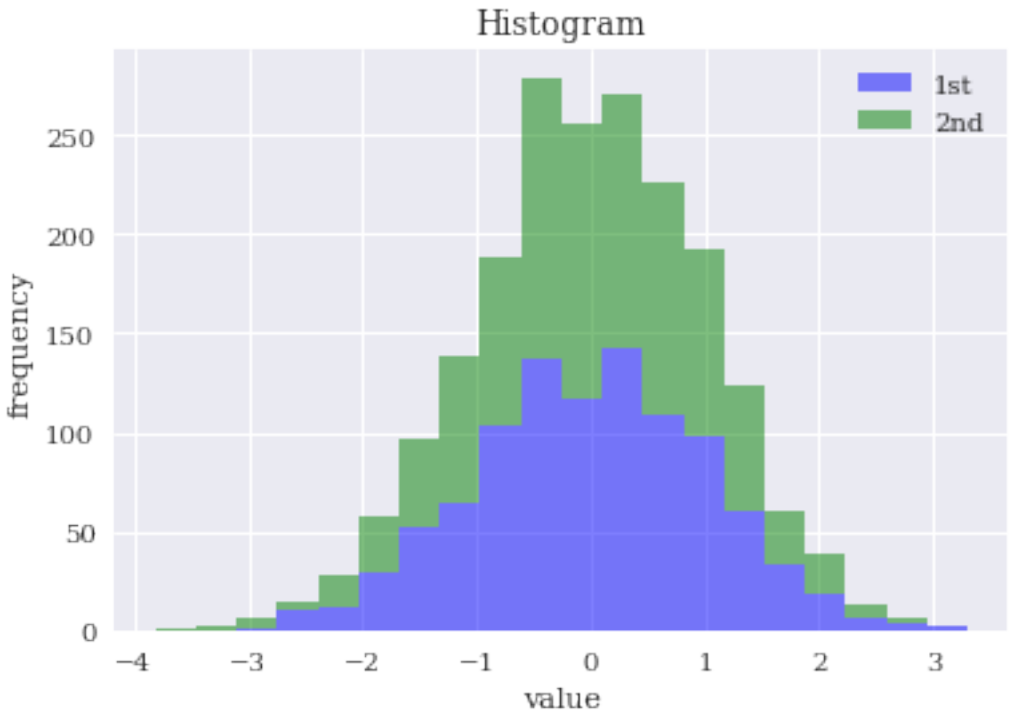
Parameters for plt.hist

Source: Python for Finance, 2nd ed.

Histogram

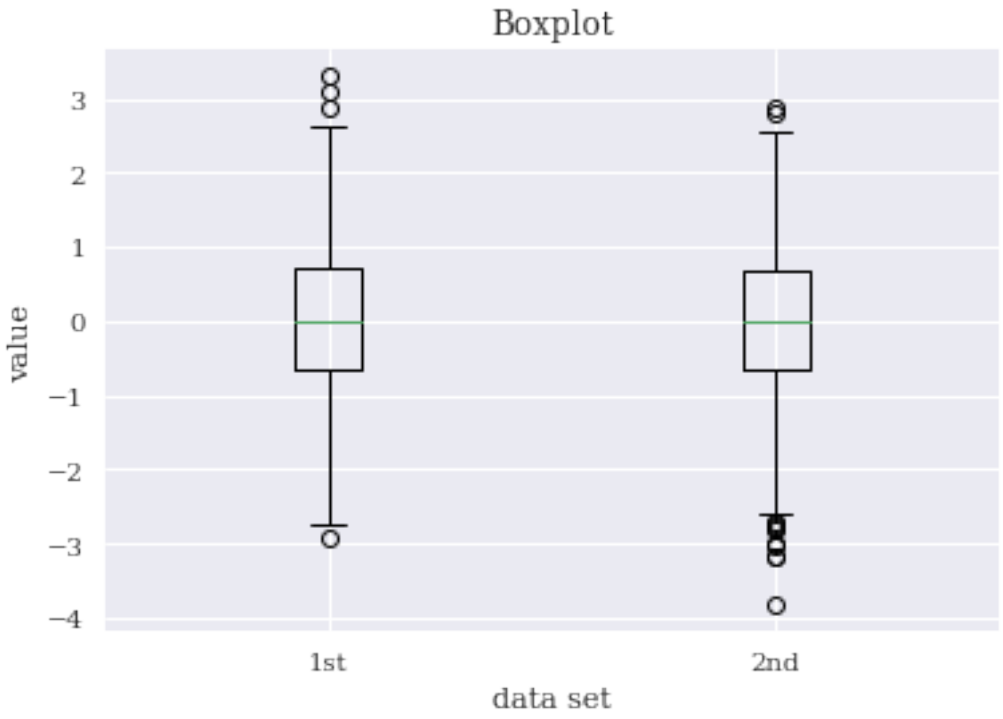
- A stacked histogram:

[17]:
```
plt.hist(y, label=['1st', '2nd'], color=['b', 'g'],
             stacked=True, bins=20, alpha=0.5)
plt.legend(loc=0)
plt.xlabel('value')
plt.ylabel('frequency')
plt.title('Histogram');
```



Boxplot

[18]:
```
fig, ax = plt.subplots()
plt.boxplot(y)
plt.setp(ax, xticklabels=['1st', '2nd'])
plt.xlabel('data set')
plt.ylabel('value')
plt.title('Boxplot');
```



Further examples

- Many more examples are found in the `matploblib` gallery.

3.2 3D plotting

- One application of 3D plots in finance are **volatility surfaces**.

The volatility surface

- In the examples below we plot **volatility surfaces**.
- These refer to prices of call and put option.
- Payoff of a call option: $\max(S_T - K, 0)$ and
- payoff of a put option: $\max(K - S_T, 0)$, where
  - S_T is the stock or index price at maturity T of the option and
  - K is the strike price.
- For some financial instruments, such as the S&P 500 or the DAX indices, liquid markets for simple call and put options exist.
- Their prices are determined by supply and demand, rather than from trading models.

The vol surface

- For these options, traders prefer to look at their **volatility** rather than their price, as this makes the market for options with different maturities and strike prices more easily comparable:
  - Price levels depend strongly on strike and time-to-maturity, even if the market considers the risk levels of options to be the same.
  - (Annualised) volatility is a measure that eliminates distortions from strike levels and time-to-maturity.
- Since there is a one-to-one relationship between an option price and the volatility one would have to plug into the Black-Scholes formula to obtain that price, traders prefer to look at this volatility, called **implied volatility**.

- A volatility surface shows implied volatility as a function of time-to-maturity and option strike.

Plotting the vol surface

- We consider the following parameters:
  - strike values between 50 and 150,
  - time-to-maturity values between 0.5 and 2.5 years.
- The function `nd.meshgrid()` creates a two-dimensional `ndarray` object from the two one-dimensional `ndarray` coordinate systems:

[19]:
```
strike = np.linspace(50, 150, 24)
```

[20]:
```
ttm = np.linspace(0.5, 2.5, 24)
```

[21]:
```
strike, ttm = np.meshgrid(strike, ttm)
```

[22]:
```
strike.ndim, strike.size
```

[22]:
```
(2, 576)
```

[23]:
```
ttm.ndim, ttm.size
```

[23]:
```
(2, 576)
```

Plotting the vol surface

[24]:
```
iv = (strike - 100) ** 2 / (100 * strike) / ttm
```

[25]:
```
iv.ndim, iv.size
```

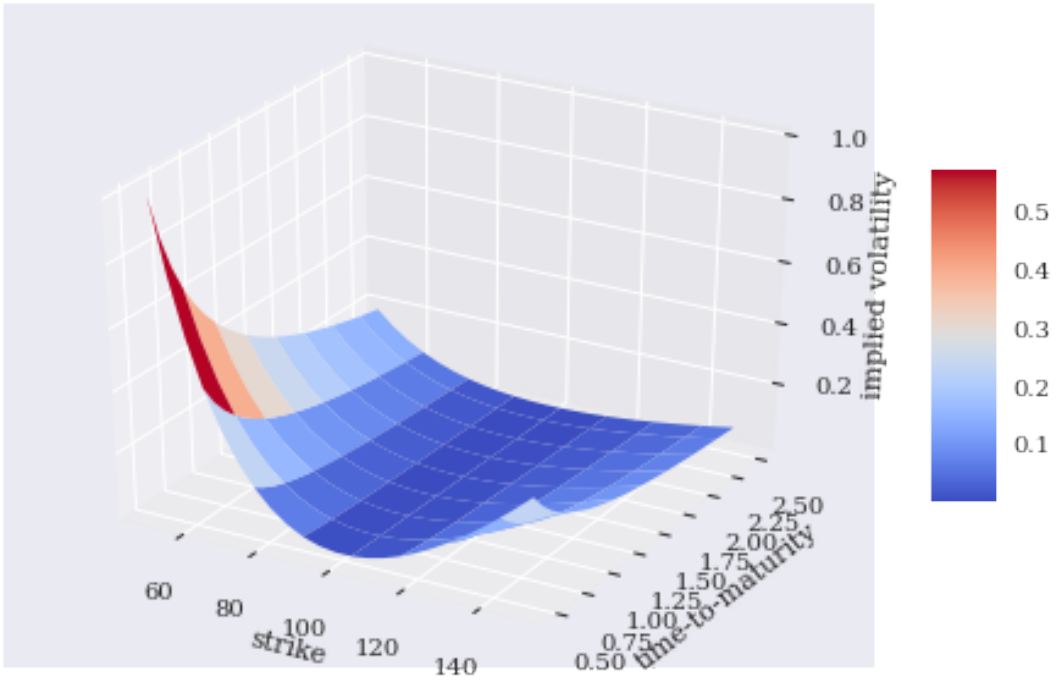
[25]:
```
(2, 576)
```

[26]:
```
iv[:5, :3]
```

[26]:
```
array([[1.        , 0.76695652, 0.58132045],
       [0.85185185, 0.65333333, 0.4951989 ],
       [0.74193548, 0.56903226, 0.43130227],
       [0.65714286, 0.504     , 0.38201058],
       [0.58974359, 0.45230769, 0.34283001]])
```

Plotting the vol surface

[27]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(8, 5))
ax = fig.gca(projection='3d')    # set up canvas for 3D plotting
surf = ax.plot_surface(strike, ttm, iv, rstride=3, cstride=3,
                       cmap=plt.cm.coolwarm, linewidth=0.5,
                       antialiased=True)  # creates 3D plot
ax.set_xlabel('strike')
ax.set_ylabel('time-to-maturity')
ax.set_zlabel('implied volatility')
fig.colorbar(surf, shrink=0.5, aspect=5);
```



Parameters for plot_surface()

Parameter	Description
`X, Y, Z`	Data values as 2D arrays
`rstride`	Array row stride (step size)
`cstride`	Array column stride (step size)
`color`	Color of the surface patches
`cmap`	Color map for the surface patches

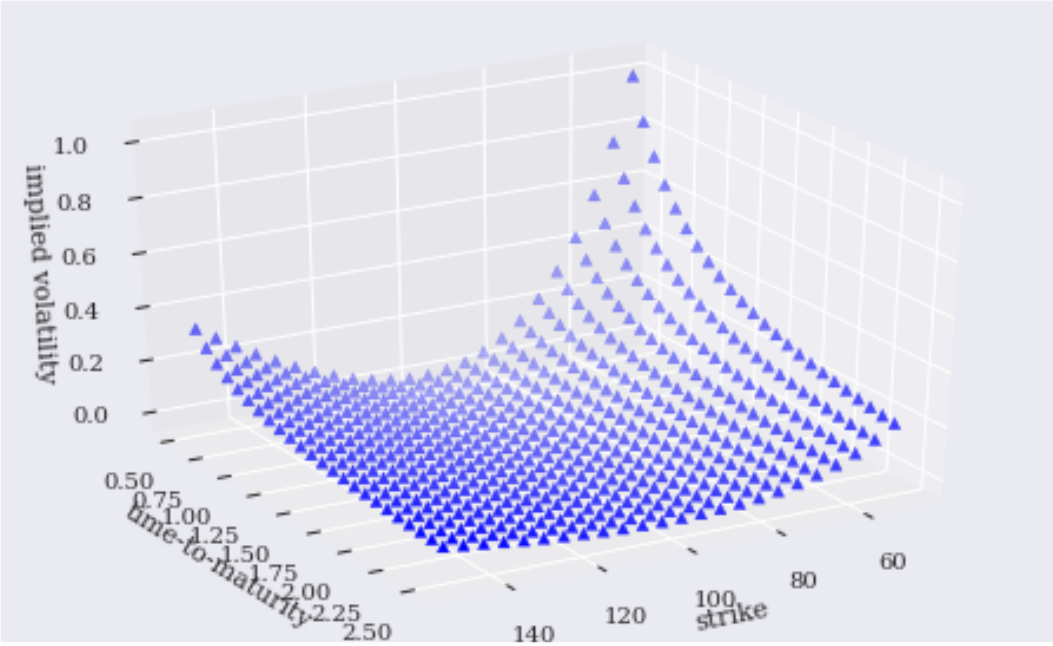
Parameter	Description
`facecolors`	Face colors for the individual patches
`norm`	Instance of `Normalize` to map values to colors
`vmin`	Minimum value to map
`vmax`	Maximum value to map
`shade`	Whether to shade the face colors

Source: Python for Finance, 2nd ed.

Plotting the vol surface

- Changing the style and with a different viewing angle:

[28]:
```
fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(30, 60)
ax.scatter(strike, ttm, iv, zdir='z', s=25,
           c='b', marker='^')
ax.set_xlabel('strike')
ax.set_ylabel('time-to-maturity')
ax.set_zlabel('implied volatility');
```



Further resources

- Matplotlib pyplot tutorial
- Matplotlib mplot3d tutorial