

# Python For Data Science Cheat Sheet

## NumPy Basics

Learn Python for Data Science Interactively at [www.DataCamp.com](https://www.datacamp.com)



### NumPy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:



NumPy

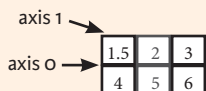
```
>>> import numpy as np
```

### NumPy Arrays

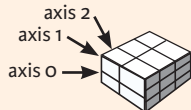
#### 1D array



#### 2D array



#### 3D array



### Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
dtype = float)
```

### Initial Placeholders

```
>>> np.zeros((3,4))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros  
Create an array of ones  
Create an array of evenly spaced values (step value)  
Create an array of evenly spaced values (number of samples)  
Create a constant array  
Create a 2X2 identity matrix  
Create an array with random values  
Create an empty array

### I/O

#### Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

#### Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

### Data Types

```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool
>>> np.object
>>> np.string_
>>> np.unicode_
```

Signed 64-bit integer types  
Standard double-precision floating point  
Complex numbers represented by 128 floats  
Boolean type storing TRUE and FALSE values  
Python object type  
Fixed-length string type  
Fixed-length unicode type

### Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions  
Length of array  
Number of array dimensions  
Number of array elements  
Data type of array elements  
Name of data type  
Convert an array to a different type

### Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

### Array Mathematics

#### Arithmetic Operations

```
>>> g = a - b
array([[ -0.5,  0. ,  0. ],
       [ -3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
array([[ 2.5,  4. ,  6. ],
       [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
array([[ 0.66666667,  1. ,  1. ],
       [ 0.25 ,  0.4 ,  0.5 ]])
>>> np.divide(a,b)
>>> a * b
array([[ 1.5,  4. ,  9. ],
       [ 4. , 10. , 18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
array([[ 7. ,  7. ],
       [ 7. ,  7.]])
```

Subtraction  
Subtraction  
Addition  
Addition  
Division  
Division  
Multiplication  
Multiplication  
Exponentiation  
Square root  
Print sines of an array  
Element-wise cosine  
Element-wise natural logarithm  
Dot product

#### Comparison

```
>>> a == b
array([[False,  True,  True],
       [False, False, False]], dtype=bool)
>>> a < 2
array([[True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison  
Element-wise comparison  
Array-wise comparison

#### Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum  
Array-wise minimum value  
Maximum value of an array row  
Cumulative sum of the elements  
Mean  
Median  
Correlation coefficient  
Standard deviation

### Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data  
Create a copy of the array  
Create a deep copy of the array

### Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array  
Sort the elements of an array's axis

### Subsetting, Slicing, Indexing

Also see Lists

#### Subsetting

```
>>> a[2]
3
>>> b[1,2]
6.0
```

Select the element at the 2nd index  
Select the element at row 1 column 2 (equivalent to b[1][2])

#### Slicing

```
>>> a[0:2]
array([1, 2])
>>> b[0:2,1]
array([ 2.,  5.])
```

Select items at index 0 and 1  
Select items at rows 0 and 1 in column 1

```
>>> b[:1]
array([[1.5, 2., 3.]])
>>> c[1,...]
array([[ 3.,  2.,  1.],
       [ 4.,  5.,  6.]])
```

Select all items at row 0 (equivalent to b[0:1, :])  
Same as [1, :, :]

Reversed array a

#### Boolean Indexing

```
>>> a[a<2]
array([1])
```

Select elements from a less than 2

#### Fancy Indexing

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]
array([ 4. ,  2. ,  6. , 1.5])
>>> b[[1, 0, 1, 0]][:[0,1,2,0]]
array([[ 4.,  5.,  6.,  4. ],
       [ 1.5,  2.,  3., 1.5]])
```

Select elements (1,0), (0,1), (1,2) and (0,0)  
Select a subset of the matrix's rows and columns

### Array Manipulation

#### Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions  
Permute array dimensions

#### Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array  
Reshape, but don't change data

#### Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a, [1])
```

Return a new array with shape (2,6)  
Append items to an array  
Insert items in an array  
Delete items from an array

#### Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
array([[ 1. ,  2. ,  3. ],
       [ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
array([[ 7.,  7.,  1.,  0.],
       [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
array([[ 1, 10],
       [ 2, 15],
       [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays  
Stack arrays vertically (row-wise)  
Stack arrays vertically (row-wise)  
Stack arrays horizontally (column-wise)  
Create stacked column-wise arrays  
Create stacked column-wise arrays

#### Splitting Arrays

```
>>> np.hsplit(a,3)
[array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
[array([[ 1.5,  2. ,  1. ],
       [ 4. ,  5. ,  6. ]]),
 array([[ 3.,  2.,  3.],
       [ 4. ,  5. ,  6.]])]
```

Split the array horizontally at the 3rd index  
Split the array vertically at the 2nd index

DataCamp

Learn Python for Data Science Interactively



# Python For Data Science Cheat Sheet

## Matplotlib

Learn Python Interactively at [www.DataCamp.com](https://www.datacamp.com)



### Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.



### 1 Prepare The Data

Also see Lists & NumPy

#### 1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

#### 2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

### 2 Create Plot

```
>>> import matplotlib.pyplot as plt
```

#### Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

#### Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

### 3 Plotting Routines

#### 1D Data

<pre>&gt;&gt;&gt; lines = ax.plot(x,y) &gt;&gt;&gt; ax.scatter(x,y) &gt;&gt;&gt; axes[0,0].bar([1,2,3],[3,4,5]) &gt;&gt;&gt; axes[1,0].barh([0.5,1,2.5],[0,1,2]) &gt;&gt;&gt; axes[1,1].axhline(0.45) &gt;&gt;&gt; axes[0,1].axvline(0.65) &gt;&gt;&gt; ax.fill(x,y,color='blue') &gt;&gt;&gt; ax.fill_between(x,y,color='yellow')</pre>	<p>Draw points with lines or markers connecting them</p> <p>Draw unconnected points, scaled or colored</p> <p>Plot vertical rectangles (constant width)</p> <p>Plot horizontal rectangles (constant height)</p> <p>Draw a horizontal line across axes</p> <p>Draw a vertical line across axes</p> <p>Draw filled polygons</p> <p>Fill between y-values and 0</p>
--	--

#### 2D Data or Images

<pre>&gt;&gt;&gt; fig, ax = plt.subplots() &gt;&gt;&gt; im = ax.imshow(img,     cmap='gist_earth',     interpolation='nearest',     vmin=-2,     vmax=2)</pre>	Colormapped or RGB arrays
--	---------------------------

#### Vector Fields

<pre>&gt;&gt;&gt; axes[0,1].arrow(0,0,0.5,0.5) &gt;&gt;&gt; axes[1,1].quiver(y,z) &gt;&gt;&gt; axes[0,1].streamplot(X,Y,U,V)</pre>	<p>Add an arrow to the axes</p> <p>Plot a 2D field of arrows</p> <p>Plot 2D vector fields</p>
--	---

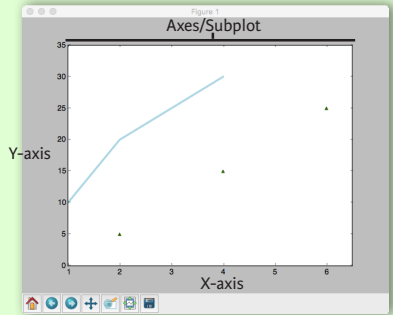
#### Data Distributions

<pre>&gt;&gt;&gt; ax1.hist(y) &gt;&gt;&gt; ax3.boxplot(y) &gt;&gt;&gt; ax3.violinplot(z)</pre>	<p>Plot a histogram</p> <p>Make a box and whisker plot</p> <p>Make a violin plot</p>
--	--

<pre>&gt;&gt;&gt; axes2[0].pcolor(data2) &gt;&gt;&gt; axes2[0].pcolormesh(data) &gt;&gt;&gt; CS = plt.contour(Y,X,U) &gt;&gt;&gt; axes2[2].contourf(data1) &gt;&gt;&gt; axes2[2] = ax.clabel(CS)</pre>	<p>Pseudocolor plot of 2D array</p> <p>Pseudocolor plot of 2D array</p> <p>Plot contours</p> <p>Plot filled contours</p> <p>Label a contour plot</p>
--	--

### Plot Anatomy & Workflow

#### Plot Anatomy



#### Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare data
- 2 Create plot
- 3 Plot
- 4 Customize plot
- 5 Save plot
- 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]
>>> y = [10,20,25,30]
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(x, y, color='lightblue', linewidth=3)
>>> ax.scatter([2,4,6],
    [5,15,25],
    color='darkgreen',
    marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()
```

### 4 Customize Plot

#### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
    cmap='seismic')
```

#### Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

#### Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

#### Text & Annotations

```
>>> ax.text(1,
    -2.1,
    'Example Graph',
    style='italic')
>>> ax.annotate("Sine",
    xy=(8, 0),
    xycoords='data',
    xytext=(10.5, 0),
    textcoords='data',
    arrowprops=dict(arrowstyle="->",
        connectionstyle="arc3"),)
```

#### Mathtext

```
>>> plt.title(r'$\sigma_i=15$', fontsize=20)
```

#### Limits, Legends & Layouts

##### Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

##### Legends

```
>>> ax.set(title='An Example Axes',
    ylabel='Y-Axis',
    xlabel='X-Axis')
>>> ax.legend(loc='best')
```

##### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
    ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
    direction='inout',
    length=10)
```

##### Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,
    hspace=0.3,
    left=0.125,
    right=0.9,
    top=0.9,
    bottom=0.1)
```

```
>>> fig.tight_layout()
```

##### Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position(('outward',10))
```

Add padding to a plot

Set the aspect ratio of the plot to 1

Set limits for x-and y-axis

Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible

Move the bottom axis line outward

### 5 Save Plot

#### Save figures

```
>>> plt.savefig('foo.png')
```

#### Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

### 6 Show Plot

```
>>> plt.show()
```

### Close & Clear

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```

Clear an axis

Clear the entire figure

Close a window

