

# Python For Data Science Cheat Sheet

## Python Basics

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



### Variables and Data Types

#### Variable Assignment

```
>>> x=5
>>> x
5
```

#### Calculations With Variables

>>> x+2 7	Sum of two variables
>>> x-2 3	Subtraction of two variables
>>> x*2 10	Multiplication of two variables
>>> x**2 25	Exponentiation of a variable
>>> x%2 1	Remainder of a variable
>>> x/float(2) 2.5	Division of a variable

#### Types and Type Conversion

str()	'5', '3.45', 'True'	Variables to strings
int()	5, 3, 1	Variables to integers
float()	5.0, 1.0	Variables to floats
bool()	True, True, True	Variables to booleans

### Asking For Help

```
>>> help(str)
```

### Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

#### String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

### Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

#### Selecting List Elements

Index starts at 0

##### Subset

```
>>> my_list[1]
>>> my_list[-3]
```

Select item at index 1  
Select 3rd last item

##### Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2  
Select items after index 0  
Select items before index 3  
Copy my\_list

##### Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my\_list[list][itemOfList]

#### List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

error

#### List Methods

>>> my_list.index(a)	Get the index of an item
>>> my_list.count(a)	Count an item
>>> my_list.append('!')	Append an item at a time
>>> my_list.remove('!')	Remove an item
>>> del(my_list[0:1])	Remove an item
>>> my_list.reverse()	Reverse the list
>>> my_list.extend('!')	Append an item
>>> my_list.pop(-1)	Remove an item
>>> my_list.insert(0, '!')	Insert an item
>>> my_list.sort()	Sort the list

#### String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

#### String Methods

>>> my_string.upper()	String to uppercase
>>> my_string.lower()	String to lowercase
>>> my_string.count('w')	Count String elements
>>> my_string.replace('e', 'i')	Replace String elements
>>> my_string.strip()	Strip whitespaces

### Libraries

#### Import libraries

```
>>> import numpy
>>> import numpy as np
Selective import
>>> from math import pi
```

pandas Data analysis	Machine learning
NumPy Scientific computing	matplotlib 2D plotting

### Install Python

ANACONDA Leading open data science platform powered by Python	spyder Free IDE that is included with Anaconda	jupyter Create and share documents with live code, visualizations, text, ...
---	--	---

### NumPy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

#### Selecting Numpy Array Elements

Index starts at 0

##### Subset

```
>>> my_array[1]
2
```

Select item at index 1

##### Slice

```
>>> my_array[0:2]
array([1, 2])
```

Select items at index 0 and 1

##### Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
array([1, 4])
```

my\_2darray[rows, columns]

#### NumPy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

#### NumPy Array Functions

>>> my_array.shape	Get the dimensions of the array
>>> np.append(other_array)	Append items to an array
>>> np.insert(my_array, 1, 5)	Insert items in an array
>>> np.delete(my_array, [1])	Delete items in an array
>>> np.mean(my_array)	Mean of the array
>>> np.median(my_array)	Median of the array
>>> <del>my_array.corrcorrf()</del>	Correlation coefficient
>>> np.std(my_array)	Standard deviation

DataCamp

Learn Python for Data Science Interactively

np.corrcorrf(my\_array)

# Python For Data Science Cheat Sheet

## Jupyter Notebook

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



### Saving/Loading Notebooks

Create new notebook

Make a copy of the current notebook

Save current notebook and record checkpoint

Preview of the printed notebook

Close notebook & stop running any scripts

Open an existing notebook

Rename notebook

Revert notebook to a previous checkpoint

Download notebook as

- IPython notebook
- Python
- HTML
- Markdown
- reST
- LaTeX
- PDF

### Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

#### Edit Cells

Cut currently selected cells to clipboard

Paste cells from clipboard above current cell

Paste cells from clipboard on top of current cell

Revert "Delete Cells" invocation

Merge current cell with the one above

Move current cell up

Adjust metadata underlying the current notebook

Remove cell attachments

Paste attachments of current cell

Copy cells from clipboard to current cursor position

Paste cells from clipboard below current cell

Delete current cells

Split up a cell from current cursor position

Merge current cell with the one below

Move current cell down

Find and replace in selected cells

Copy attachments of current cell

Insert image in selected cells

#### Insert Cells

Add new cell above the current one

Add new cell below the current one

### Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IP[y]:  
IPython

R  
IRkernel

IJ[.j]:  
JJulia

Installing Jupyter Notebook will automatically install the IPython kernel.

Restart kernel

Restart kernel & run all cells

Restart kernel & run all cells

Interrupt kernel

Interrupt kernel & clear all output

Connect back to a remote notebook

Run other installed kernels

### Command Mode:

### Edit Mode:

### Executing Cells

Run selected cell(s)

Run current cells down and create a new one above

Run all cells above the current cell

Change the cell type of current cell

toggle, toggle scrolling and clear all output

Run current cells down and create a new one below

Run all cells

Run all cells below the current cell

toggle, toggle scrolling and clear current outputs

### View Cells

Toggle display of Jupyter logo and filename

Toggle line numbers in cells

Toggle display of toolbar

Toggle display of cell action icons:

- None
- Edit metadata
- Raw cell format
- Slideshow
- Attachments
- Tags

### Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

Download serialized state of all widget models in use

Save notebook with interactive widgets

Embed current widgets

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

### Asking For Help

Walk through a UI tour

Edit the built-in keyboard shortcuts

Description of markdown available in notebook

Python help topics

NumPy help topics

Matplotlib help topics

Pandas help topics

List of built-in keyboard shortcuts

Notebook help topics

Information on unofficial Jupyter Notebook extensions

IPython help topics

SciPy help topics

SymPy help topics

About Jupyter Notebook



# 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

```
1 2 3
```

#### 2D array

axis 1  
axis 0

```
1.5 2 3  
4 5 6
```

#### 3D array

axis 2  
axis 1  
axis 0

### 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  
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() np.median(b)  
>>> a.corrcoef() np.corrcoef(a)  
>>> 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 0 column 2 (equivalent to b[1][2])

#### Slicing

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

Select items at index 0 and 1  
Select items at rows 0 and 1 in column 1  
Select all items at row 0 (equivalent to b[0:1, :])  
Same as [1, :, :]

#### Boolean Indexing

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

Reversed array a  
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],  
       [ 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

## SciPy - Linear Algebra

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



### SciPy

The **SciPy** library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.



### Interacting With NumPy

[Also see NumPy](#)

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)])
```

### Index Tricks

<pre>&gt;&gt;&gt; np.mgrid[0:5,0:5] &gt;&gt;&gt; np.ogrid[0:2,0:2] &gt;&gt;&gt; np.r_[[3,[0]*5,-1:1:10j]] &gt;&gt;&gt; np.c_[b,c]</pre>	Create a dense meshgrid Create an open meshgrid Stack arrays vertically (row-wise) Create stacked column-wise arrays
---	---

### Shape Manipulation

<pre>&gt;&gt;&gt; np.transpose(b) &gt;&gt;&gt; b.flatten() &gt;&gt;&gt; np.hstack((b,c)) &gt;&gt;&gt; np.vstack((a,b)) &gt;&gt;&gt; np.hsplit(c,2) &gt;&gt;&gt; np.vpsplit(d,2)</pre>	Permute array dimensions Flatten the array Stack arrays horizontally (column-wise) Stack arrays vertically (row-wise) Split the array horizontally at the 2nd index Split the array vertically at the 2nd index
---	--

### Polynomials

<pre>&gt;&gt;&gt; from numpy import polyld &gt;&gt;&gt; p = polyld([3,4,5])</pre>	Create a polynomial object
---	----------------------------

### Vectorizing Functions

<pre>&gt;&gt;&gt; def myfunc(a):     if a &lt; 0:         return a*2     else:         return a/2 &gt;&gt;&gt; np.vectorize(myfunc)</pre>	Vectorize functions
---	---------------------

### Type Handling

<pre>&gt;&gt;&gt; np.real(c) &gt;&gt;&gt; np.imag(c) &gt;&gt;&gt; np.real_if_close(c,tol=1000) &gt;&gt;&gt; np.cast['f'](np.pi)</pre>	Return the real part of the array elements Return the imaginary part of the array elements Return a real array if complex parts close to 0 Cast object to a data type
---	--

### Other Useful Functions

<pre>&gt;&gt;&gt; np.angle(b,deg=True) &gt;&gt;&gt; g = np.linspace(0,np.pi,num=5) &gt;&gt;&gt; g[3:] += np.pi &gt;&gt;&gt; np.unwrap(g) &gt;&gt;&gt; np.logspace(0,10,3) &gt;&gt;&gt; np.select([c&lt;4],[c*2])  &gt;&gt;&gt; misc.factorial(a) &gt;&gt;&gt; misc.comb(10,3,exact=True) &gt;&gt;&gt; misc.central_diff_weights(3) &gt;&gt;&gt; misc.derivative(myfunc,1.0)</pre>	Return the angle of the complex argument Create an array of evenly spaced values (number of samples) Unwrap Create an array of evenly spaced values (log scale) Return values from a list of arrays depending on conditions Factorial Combine N things taken at k time Weights for Np-point central derivative Find the n-th derivative of a function at a point
---	--

## Linear Algebra

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

### Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

### Basic Matrix Routines

#### Inverse

```
>>> A.I
>>> linalg.inv(A)
>>> A.T
>>> A.H
>>> np.trace(A)
```

Inverse  
Inverse  
Transpose matrix  
Conjugate transposition  
Trace

#### Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

Frobenius norm  
L1 norm (max column sum)  
L inf norm (max row sum)

#### Rank

```
>>> np.linalg.matrix_rank(C)
```

Matrix rank

#### Determinant

```
>>> linalg.det(A)
```

Determinant

#### Solving linear problems

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(D,E)
```

Solver for dense matrices  
Solver for dense matrices  
Least-squares solution to linear matrix equation

#### Generalized inverse

```
>>> linalg.pinv(C)
>>> linalg.pinv2(C)
```

Compute the pseudo-inverse of a matrix (least-squares solver)  
Compute the pseudo-inverse of a matrix (SVD)

### Creating Sparse Matrices

<pre>&gt;&gt;&gt; F = np.eye(3, k=1) &gt;&gt;&gt; G = np.mat(np.identity(2)) &gt;&gt;&gt; C[C &gt; 0.5] = 0 &gt;&gt;&gt; H = sparse.csr_matrix(C) &gt;&gt;&gt; I = sparse.csc_matrix(D) &gt;&gt;&gt; J = sparse.dok_matrix(A) &gt;&gt;&gt; E.todense() &gt;&gt;&gt; sparse.isspmatrix_csc(A)</pre>	Create a 2X2 identity matrix Create a 2x2 identity matrix  Compressed Sparse Row matrix Compressed Sparse Column matrix Dictionary Of Keys matrix Sparse matrix to full matrix Identify sparse matrix
--	--

### Sparse Matrix Routines

#### Inverse

```
>>> sparse.linalg.inv(I)
```

Inverse

#### Norm

```
>>> sparse.linalg.norm(I)
```

Norm

#### Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Solver for sparse matrices

### Sparse Matrix Functions

<pre>&gt;&gt;&gt; sparse.linalg.expm(I)</pre>	Sparse matrix exponential
---	---------------------------

### Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

[Also see NumPy](#)

### Matrix Functions

#### Addition

```
>>> np.add(A,D)
```

Addition

#### Subtraction

```
>>> np.subtract(A,D)
```

Subtraction

#### Division

```
>>> np.divide(A,D)
```

Division

#### Multiplication

```
>>> np.multiply(D,A)
>>> np.dot(A,D)
>>> np.vdot(A,D)
>>> np.inner(A,D)
>>> np.outer(A,D)
>>> np.tensordot(A,D)
>>> np.kron(A,D)
```

Multiplication  
Dot product  
Vector dot product  
Inner product  
Outer product  
Tensor dot product  
Kronecker product

#### Exponential Functions

```
>>> linalg.expm(A)
>>> linalg.expm2(A)
>>> linalg.expm3(D)
```

Matrix exponential  
Matrix exponential (Taylor Series)  
Matrix exponential (eigenvalue decomposition)

#### Logarithm Function

```
>>> linalg.logm(A)
```

Matrix logarithm

#### Trigonometric Functions

```
>>> linalg.sinm(D)
>>> linalg.cosm(D)
>>> linalg.tanm(A)
```

Matrix sine  
Matrix cosine  
Matrix tangent

#### Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
>>> linalg.coshm(D)
>>> linalg.tanhm(A)
```

Hyperbolic matrix sine  
Hyperbolic matrix cosine  
Hyperbolic matrix tangent

#### Matrix Sign Function

```
>>> np.sigm(A)
```

Matrix sign function

#### Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Matrix square root

#### Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)
```

Evaluate matrix function

### Decompositions

#### Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
>>> l1, l2 = la
>>> v[:,0]
>>> v[:,1]
>>> linalg.eigvals(A)
```

Solve ordinary or generalized eigenvalue problem for square matrix  
Unpack eigenvalues  
First eigenvector  
Second eigenvector  
Unpack eigenvalues

#### Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)
```

Singular Value Decomposition (SVD)  
Construct sigma matrix in SVD

#### LU Decomposition

```
>>> P,L,U = linalg.lu(C)
```

LU Decomposition

### Sparse Matrix Decompositions

<pre>&gt;&gt;&gt; la, v = sparse.linalg.eigs(F,1) &gt;&gt;&gt; sparse.linalg.svds(H, 2)</pre>	Eigenvalues and eigenvectors SVD
---	-------------------------------------

DataCamp

Learn Python for Data Science [Interactively](#)





# Python For Data Science Cheat Sheet

## Pandas Basics

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



### Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis tools** for the Python programming language.



Use the following import convention:

```
>>> import pandas as pd
```

### Pandas Data Structures

#### Series

A **one-dimensional** labeled array capable of holding any data type

a	3
b	-5
c	7
d	4

Index

```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

#### DataFrame

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528

A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
           'Capital': ['Brussels', 'New Delhi', 'Brasília'],
           'Population': [11190846, 1303171035, 207847528]}
```

```
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

### I/O

#### Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> df.to_csv('myDataFrame.csv')
```

#### Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')

Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

### Asking For Help

```
>>> help(pd.Series.loc)
```

### Selection

Also see NumPy Arrays

#### Getting

```
>>> s['b']
-5

>>> df[1:]
   Country  Capital  Population
1   India  New Delhi  1303171035
2  Brazil  Brasília  207847528
```

Get one element

Get subset of a DataFrame

#### Selecting, Boolean Indexing & Setting

##### By Position

```
>>> df.iloc([0], [0])
'Belgium'

>>> df.iat([0], [0])
'Belgium'
```

Select single value by row & column

##### By Label

```
>>> df.loc([0], ['Country'])
'Belgium'

>>> df.at([0], ['Country'])
'Belgium'
```

Select single value by row & column labels

##### By Label/Position

```
>>> df.ix[2]
Country      Brazil
Capital      Brasília
Population    207847528
```

Select single row of subset of rows

```
>>> df.ix[:, 'Capital']
0      Brussels
1    New Delhi
2    Brasilia
```

Select a single column of subset of columns

```
>>> df.ix[1, 'Capital']
'New Delhi'
```

Select rows and columns

##### Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population'] > 1200000000]
```

Series **s** where value is not >1  
**s** where value is <-1 or >2  
Use filter to adjust DataFrame

##### Setting

```
>>> s['a'] = 6
```

Set index **a** of Series **s** to 6

### Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)  
Drop values from columns(axis=1)

### Sort & Rank

```
>>> df.sort_index()
>>> df.sort_values(by='Country')
>>> df.rank()
```

Sort by labels along an axis  
Sort by the values along an axis  
Assign ranks to entries

### Retrieving Series/DataFrame Information

#### Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)  
Describe index  
Describe DataFrame columns  
Info on DataFrame  
Number of non-NA values

#### Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values  
Cumulative sum of values  
Minimum/maximum values  
Minimum/Maximum index value  
Summary statistics  
Mean of values  
Median of values

### Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function  
Apply function element-wise

### Data Alignment

#### Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
a      10.0
b      NaN
c       5.0
d       7.0
```

#### Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
a      10.0
b     -5.0
c       5.0
d       7.0

>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

#### Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

DataCamp

Learn Python for Data Science Interactively



# Python For Data Science Cheat Sheet

## Scikit-Learn

Learn Python for data science [Interactively](#) at [www.DataCamp.com](#)



### Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.



#### A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

#### Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','F','F','F'])
>>> X[X < 0.7] = 0
```

#### Training And Test Data

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    random_state=0)
```

### Preprocessing The Data

#### Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

#### Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

#### Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

### Create Your Model

#### Supervised Learning Estimators

```
Linear Regression
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)

Support Vector Machines (SVM)
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')

Naive Bayes
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()

KNN
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

#### Unsupervised Learning Estimators

```
Principal Component Analysis (PCA)
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)

K Means
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

### Model Fitting

<b>Supervised learning</b> >>> lr.fit(X, y) >>> knn.fit(X_train, y_train) >>> svc.fit(X_train, y_train)	Fit the model to the data
<b>Unsupervised Learning</b> >>> k_means.fit(X_train) >>> pca_model = pca.fit_transform(X_train)	Fit the model to the data Fit to data, then transform it

### Prediction

<b>Supervised Estimators</b> >>> y_pred = svc.predict(np.random.random((2,5))) >>> y_pred = lr.predict(X_test) >>> y_pred = knn.predict_proba(X_test)	Predict labels Predict labels Estimate probability of a label
<b>Unsupervised Estimators</b> >>> y_pred = k_means.predict(X_test)	Predict labels in clustering algos

### Evaluate Your Model's Performance

#### Classification Metrics

<b>Accuracy Score</b> >>> knn.score(X_test, y_test) >>> from sklearn.metrics import accuracy_score >>> accuracy_score(y_test, y_pred)	Estimator score method Metric scoring functions
<b>Classification Report</b> >>> from sklearn.metrics import classification_report >>> print(classification_report(y_test, y_pred))	Precision, recall, f1-score and support
<b>Confusion Matrix</b> >>> from sklearn.metrics import confusion_matrix >>> print(confusion_matrix(y_test, y_pred))	

#### Regression Metrics

<b>Mean Absolute Error</b> >>> from sklearn.metrics import mean_absolute_error >>> y_true = [3, -0.5, 2] >>> mean_absolute_error(y_true, y_pred)	
<b>Mean Squared Error</b> >>> from sklearn.metrics import mean_squared_error >>> mean_squared_error(y_test, y_pred)	
<b>R<sup>2</sup> Score</b> >>> from sklearn.metrics import r2_score >>> r2_score(y_true, y_pred)	

#### Clustering Metrics

<b>Adjusted Rand Index</b> >>> from sklearn.metrics import adjusted_rand_score >>> adjusted_rand_score(y_true, y_pred)	
<b>Homogeneity</b> >>> from sklearn.metrics import homogeneity_score >>> homogeneity_score(y_true, y_pred)	
<b>V-measure</b> >>> from sklearn.metrics import v_measure_score >>> metrics.v_measure_score(y_true, y_pred)	

#### Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

### Tune Your Model

#### Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,5),
            "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                      param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

#### Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
            "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                               param_distributions=params,
                               cv=4,
                               n_iter=8,
                               random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

#### Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

#### Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

#### Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```



# 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

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them  
Draw unconnected points, scaled or colored  
Plot vertical rectangles (constant width)  
Plot horizontal rectangles (constant height)  
Draw a horizontal line across axes  
Draw a vertical line across axes  
Draw filled polygons  
Fill between y-values and 0

#### 2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
                  cmap='gist_earth',
                  interpolation='nearest',
                  vmin=-2,
                  vmax=2)
```

Colormapped or RGB arrays

#### Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)
>>> axes[1,1].quiver(y,z)
>>> axes[0,1].streamplot(X,Y,U,V)
```

Add an arrow to the axes  
Plot a 2D field of arrows  
Plot a 2D field of arrows

#### Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

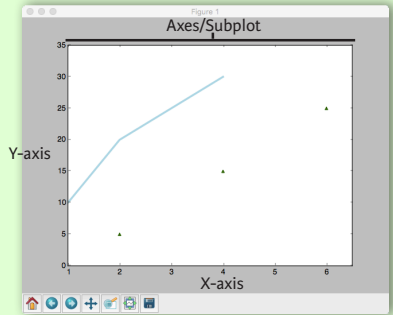
Plot a histogram  
Make a box and whisker plot  
Make a violin plot

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2] = ax.clabel(CS)
```

Pseudocolor plot of 2D array  
Pseudocolor plot of 2D array  
Plot contours  
Plot filled contours  
Label a contour plot

### 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





Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on **matplotlib** and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

- 1. Prepare some data
- 2. Control figure aesthetics
- 3. Plot with Seaborn
- 4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> sns.set_style("whitegrid")
>>> g = sns.lmplot(x="tip", y="total_bill", data=tips, aspect=2)
>>> g = (g.set_axis_labels("Tip", "Total bill (USD)")).set(xlim=(0,10),ylim=(0,100))
>>> plt.title("title")
>>> plt.show(g)
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101), 'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```

Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks", {'xtick.major.size':8, 'ytick.major.size':8})
>>> sns.axes_style("whitegrid")
```

(Re)set the seaborn default Set the matplotlib parameters Set the matplotlib parameters

Return a dict of params or use with to temporarily set the style

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic, col="survived", row="sex")
>>> g = g.map(plt.hist, "age")
>>> sns.factorplot(x="pclass", y="survived", hue="sex", data=titanic)
>>> sns.lmplot(x="sepal_width", y="sepal_length", hue="species", data=iris)
```

Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
>>> i = sns.JointGrid(x="x", y="y", data=data)
>>> i = i.plot(sns.regplot, sns.distplot)
>>> sns.jointplot("sepal_length", "sepal_width", data=iris, kind='kde')
```

Subplot grid for plotting pairwise relationships Plot pairwise bivariate distributions Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Categorical Plots

```
>>> sns.stripplot(x="species", y="petal_length", data=iris)
>>> sns.swarmplot(x="species", y="petal_length", data=iris)
```

Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

```
>>> sns.barpplot(x="sex", y="survived", hue="class", data=titanic)
```

Show point estimates and confidence intervals with scatterplot glyphs

```
>>> sns.countplot(x="deck", data=titanic, palette="Greens_d")
```

Show count of observations

```
>>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette={"male": "g", "female": "m"}, markers=["^", "o"], linestyle=["-", "--"])
```

Show point estimates and confidence intervals as rectangular bars

```
>>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic)
>>> sns.boxplot(data=iris, orient="h")
```

Boxplot

Boxplot with wide-form data

```
>>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)
```

Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width", y="sepal_length", data=iris, ax=ax)
```

Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y, kde=False, color="b")
```

Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data, vmin=0, vmax=1)
```

Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived", "Sex")
>>> h.set(xlim=(0,5), ylim=(0,5), xticks=[0,2.5,5], yticks=[0,2.5,5])
```

Remove left spine Set the labels of the y-axis Set the tick labels for x Set the axis labels

Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax, yticks=[0,5])
>>> plt.tight_layout()
```

Add plot title Adjust the label of the y-axis Adjust the label of the x-axis Adjust the limits of the y-axis Adjust the limits of the x-axis Adjust a plot property Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

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

Show the plot Save the plot as a figure Save transparent figure

Close & Clear

Also see Matplotlib

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

Clear an axis Clear an entire figure Close a window





## Bokeh

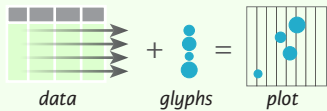
Learn Bokeh **Interactively** at [www.DataCamp.com](https://www.datacamp.com),  
taught by Bryan Van de Ven, core contributor



## Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:
2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5]
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",
>>>             x_axis_label='x',
>>>             y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)
>>> output_file("lines.html")
>>> show(p)
```

## 1 Data

Also see [Lists, NumPy & Pandas](#)

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9, 4, 65, 'US'],
>>>                             [32.4, 4, 66, 'Asia'],
>>>                             [21.4, 4, 109, 'Europe']]
>>>                  columns=['mpg', 'cyl', 'hp', 'origin'],
>>>                  index=['Toyota', 'Fiat', 'Volvo'])
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

## 2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
>>>             x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

## 3 Renderers & Visual Customizations

### Glyphs

**Scatter Markers**

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
>>>           fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
>>>           color='blue', size=1)
```

**Line Glyphs**

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
>>>               pd.DataFrame([[3,4,5],[3,2,1]]),
>>>               color="blue")
```

### Customized Glyphs

Also see [Data](#)

**Selection and Non-Selection Glyphs**

```
>>> p = figure(tools='box_select')
>>> p.circle('mpg', 'cyl', source=cds_df,
>>>          selection_color='red',
>>>          nonselection_alpha=0.1)
```

**Hover Glyphs**

```
>>> from bokeh.models import HoverTool
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```

**Colormapping**

```
>>> from bokeh.models import CategoricalColorMapper
>>> color_mapper = CategoricalColorMapper(
>>>               factors=['US', 'Asia', 'Europe'],
>>>               palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
>>>           color=dict(field='origin',
>>>                       transform=color_mapper),
>>>           legend='Origin')
```

### Legend Location

**Inside Plot Area**

```
>>> p.legend.location = 'bottom_left'
```

**Outside Plot Area**

```
>>> from bokeh.models import Legend
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1]))
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One", [p1, r1]), ("Two", [r2])],
>>>                  location=(0, -30))
>>> p.add_layout(legend, 'right')
```

### Legend Orientation

```
>>> p.legend.orientation = "horizontal"
>>> p.legend.orientation = "vertical"
```

### Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
>>> p.legend.background_fill_color = "white"
```

### Rows & Columns Layout

**Rows**

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2,p3)
```

**Columns**

```
>>> from bokeh.layouts import columns
>>> layout = column(p1,p2,p3)
```

**Nesting Rows & Columns**

```
>>> layout = row(column(p1,p2), p3)
```

### Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

### Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

### Linked Plots

#### Linked Axes

```
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range
```

#### Linked Brushing

```
>>> p4 = figure(plot_width = 100,
>>>             tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200,
>>>             tools='box_select,lasso_select')
>>> p5.circle('mpg', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

## 4 Output & Export

### Notebook

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

### HTML

#### Standalone HTML

```
>>> from bokeh.embed import file_html
>>> from bokeh.resources import CDN
>>> html = file_html(p, CDN, "my_plot")
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

#### Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

### PNG

```
>>> from bokeh.io import export_png
>>> export_png(p, filename="plot.png")
```

### SVG

```
>>> from bokeh.io import export_svgs
>>> p.output_backend = "svg"
>>> export_svgs(p, filename="plot.svg")
```

## 5 Show or Save Your Plots

>>> show(p1)	>>> show(layout)
>>> save(p1)	>>> save(layout)

