



CONDA CHEAT SHEET

Command line package and environment manager

Learn to use conda in 30 minutes at bit.ly/tryconda

TIP: Anaconda Navigator is a graphical interface to use conda. Double-click the Navigator icon on your desktop or in a Terminal or at the Anaconda prompt, type `anaconda-navigator`

Conda basics

Verify conda is installed, check version number	<code>conda info</code>
Update conda to the current version	<code>conda update conda</code>
Install a package included in Anaconda	<code>conda install PACKAGENAME</code>
Run a package after install, example Spyder*	<code>spyder</code>
Update any installed program	<code>conda update PACKAGENAME</code>
Command line help	<code>COMMANDNAME --help</code> <code>conda install --help</code>

*Must be installed and have a deployable command, usually PACKAGENAME

Using environments

Create a new environment named py35, install Python 3.5	<code>conda create --name py35 python=3.5</code>
Activate the new environment to use it	WINDOWS: <code>activate py35</code> LINUX, macOS: <code>source activate py35</code>
Get a list of all my environments, active environment is shown with *	<code>conda env list</code>
Make exact copy of an environment	<code>conda create --clone py35 --name py35-2</code>
List all packages and versions installed in active environment	<code>conda list</code>
List the history of each change to the current environment	<code>conda list --revisions</code>
Restore environment to a previous revision	<code>conda install --revision 2</code>
Save environment to a text file	<code>conda list --explicit > bio-env.txt</code>
Delete an environment and everything in it	<code>conda env remove --name bio-env</code>
Deactivate the current environment	WINDOWS: <code>deactivate</code> macOS, LINUX: <code>source deactivate</code>
Create environment from a text file	<code>conda env create --file bio-env.txt</code>
Stack commands: create a new environment, name it bio-env and install the biopython package	<code>conda create --name bio-env biopython</code>

Finding conda packages

Use conda to search for a package	<code>conda search PACKAGENAME</code>
See list of all packages in Anaconda	https://docs.anaconda.com/anaconda/packages/pkg-docs



CONTINUED ON BACK ➡

Installing and updating packages

Install a new package (Jupyter Notebook) in the active environment	<code>conda install jupyter</code>
Run an installed package (Jupyter Notebook)	<code>jupyter-notebook</code>
Install a new package (toolz) in a different environment (bio-env)	<code>conda install --name bio-env toolz</code>
Update a package in the current environment	<code>conda update scikit-learn</code>
Install a package (boltons) from a specific channel (conda-forge)	<code>conda install --channel conda-forge boltons</code>
Install a package directly from PyPI into the current active environment using pip	<code>pip install boltons</code>
Remove one or more packages (toolz, boltons) from a specific environment (bio-env)	<code>conda remove --name bio-env toolz boltons</code>

Managing multiple versions of Python

Install different version of Python in a new environment named py34	<code>conda create --name py34 python=3.4</code>
Switch to the new environment that has a different version of Python	Windows: <code>activate py34</code> Linux, macOS: <code>source activate py34</code>
Show the locations of all versions of Python that are currently in the path NOTE: The first version of Python in the list will be executed.	Windows: <code>where python</code> Linux, macOS: <code>which -a python</code>
Show version information for the current active Python	<code>python --version</code>

Specifying version numbers

Ways to specify a package version number for use with conda create or conda install commands, and in meta.yaml files.

Constraint type	Specification	Result
Fuzzy	<code>numpy=1.11</code>	1.11.0, 1.11.1, 1.11.2, 1.11.18 etc.
Exact	<code>numpy==1.11</code>	1.11.0
Greater than or equal to	<code>"numpy>=1.11"</code>	1.11.0 or higher
OR	<code>"numpy=1.11.1 1.11.3"</code>	1.11.1, 1.11.3
AND	<code>"numpy>=1.8,<2"</code>	1.8, 1.9, not 2.0

NOTE: Quotation marks must be used when your specification contains a space or any of these characters: > < | *

MORE RESOURCES


Free Community Support	groups.google.com/a/continuum.io/forum/#!forum/conda
Online Documentation	conda.io/docs
Command Reference	conda.io/docs/commands
Paid Support Options	anaconda.com/support
Anaconda Onsite Training Courses	anaconda.com/training
Anaconda Consulting Services	anaconda.com/consulting

Follow us on Twitter [@anaconda](https://twitter.com/anaconda) and join the [#AnacondaCrew](https://twitter.com/AnacondaCrew)!


Connect with other talented, like-minded data scientists and developers while contributing to the open source movement. Visit anaconda.com/community



Atom Keyboard Shortcut Cheatsheet

⌘ : Command ← : Left arrow ↓ : Down arrow ⇧ : Shift ⌫ : Escape
 ^ : Control → : Right arrow ⌥ : Option/Alt ⇥ : Tab  : Mouse Click
 ⌫ : Delete ↑ : Up arrow ↵ : Return/Enter _ : Space

Open Command Palette	⌘ ⇧ P	Toggle Autocomplete	^ _
Preferences/Settings	⌘ ,	Toggle Bookmark	⌘ F2
Switch Windows	⌘ `	View All Bookmarks	^ F2
Switch Applications	⌘ ⇥	Jump to Next Bookmark	F2
Switch Tab Left	⌘ ⇧ {	Jump to Previous Bookmark	⇧ F2
Switch Tab Right	⌘ ⇧ }	Clear All Bookmarks	⌘ ⇧ F2
Close Window	⌘ W	Find Matching Bracket	^ M
Close Tab	⌘ ⇧ W	Remove Matching Bracket	^ ⌫
Hide Application	⌘ H	Remove Selected Bracket	^]
Hide Other Applications	⌥ ⌘ H	Fold Code	⌥ ⌘ [
Minimize Application	⌘ M	Unfold Code	⌥ ⌘]
New File	⌘ N	Fold at Indentation (N)	⌘ K ⌘ N
New Window	⌘ ⇧ N	Fold Selected Text	⌥ ⌘ ^ F
Open...	⌘ O	Fold All Code	⌥ ⌘ {
Add Project Folder	⌘ ⇧ O	Unfold All Code	⌥ ⌘ }
Save	⌘ S	Reflow Selection	⌥ ⌘ Q
Save As...	⌘ ⇧ S	Cut to End of Line	^ K
Save All	⌥ ⇧ S	Delete to Start of Word	⌥ ⌫
Quit	⌘ Q	Delete to End of Word	⌥ D

Delete Line	^ ⌵ K
Transpose	^ T
Duplicate Line	⌘ ⌵ D
Go to Line	^ G
Go to Matching Bracket	^ M
Indent Selected Text	⌘]
Outdent Selected Text	⌘ [
Join Lines	⌘ J
Move to Start of Word	⌵ B
Move to End of Word	⌵ F
Move to Start of Line	^ A
Move to End of Line	^ E
Move Line Up	⌘ ^ ↑
Move Line Down	⌘ ^ ↓
Move to Top of File	⌘ ↑
Move to Bottom of File	⌘ ↓
Move to First Character	⌘ ←
Move to Last Character	⌘ →
Move Selection Left	⌘ ^ ←
Move Selection Right	⌘ ^ →
Add Selection Above	^ ⌵ ↑
Add Selection Below	^ ⌵ ↓
Multiple Cursors	⌵ ⌘ 

Split into Lines	⌘ ⌵ L
Single Selection	⌵
Select to Start of Word	⌵ ⌵ ←
Select to End of Word	⌵ ⌵ →
Select Entire Word	^ ⌵ W
Select to First Character	⌘ ⌵ ←
Select to Last Character	⌘ ⌵ →
Select to Top of File	⌘ ⌵ ↑
Select to Bottom of File	⌘ ⌵ ↓
Select All	⌘ A
Select Line	⌘ L
Select Word	^ ⌵ W
Select Inside { } [] ()	^ ⌘ M
Toggle (Line) Comment	⌘ /
Convert Tabs to Spaces	⌵ ⌘ [
Convert Spaces to Tabs	⌵ ⌘]
Convert to Upper Case	⌘ K U
Convert to Lower Case	⌘ K L
Split Panes Vertically	⌘ K ↓
Split Panes Horizontally	⌘ K →
Vertical Navigation Panes	⌘ K ⌘ ↓
Horizontal Navigation Panes	⌘ K ⌘ →
Toggle Full Screen	^ ⌘ F

Toggle GitHub Tab	^ ⇧ 8
Toggle Git Tab	^ ⇧ 9
Toggle Git+ Palette	⌘ ⇧ H
Add (Git+)	^ ⇧ A
Commit (Git+)	^ ⇧ C
Add & Commit (Git+)	^ ⇧ A, C
Add, Commit & Push	^ ⇧ A, Q
Add All & Commit (Git+)	^ ⇧ A, A
Add All, Commit & Push	^ ⇧ A, P
Run (Hydrogen)	⌘ ↵
Run & Move Down	⌘ ⇧
Run Cell (Hydrogen)	⌘ ⇧ ↵
Run Cell & Move Down	⌘ ⇧ ⇧ ↵
Run All (Hydrogen)	^ ⌘ ↵
Clear Results (Hydrogen)	⌘ ⇧ ⌫
Toggle Markdown Preview	^ ⇧ M
Toggle Tree View	⌘ \
Reveal Active File	⌘ ⇧ \
Focus On File	^ 0
Fuzzy Find Files	⌘ T
Find Open File	⌘ B
Find Uncommitted File	⌘ ⇧ B
Find File in Project	⌘ P

Copy Path	^ ⇧ C
Add a File	A
Move a File	M
Delete a File	⌫
Reopen Last File	⌘ ⇧ T
Find String in File	⌘ F
Find String in Project	⇧ ⌘ F
Exit String Search Bar	⌫
Find Next String	⌘ G
Find Previous String	⇧ ⌘ G
Select Next String	⌘ D
Select All Strings	^ ⌘ G
Replace Strings	⌘ ⇧ F
Replace Next String	⌘ ⇧ E
Increase Font Size	⌘ +
Decrease Font Size	⌘ -
Reset Font Size	⌘ 0
Choose Encoding	^ ⇧ U
Choose Grammar	^ ⇧ L
Emojis & Symbols	^ ⌘ ⌵
Search Symbol	⌘ R
Search Symbol in Project	⌘ ⇧ R

Mac OS X keymap:

⌘	Command
⌃	Control
⌥	Option
⇧	Shift
↵	Return
␣	Space
⇥	Tab

Linux/PC keymap:

⌘	Control
⌥	Alt
⇧	Shift
↵	Return
␣	Space
⇥	Tab

Edit mode shortcuts

Esc	switch to command mode
⇥	code completion or indent
⇧⇥	tooltip
⌘]	indent
⌘[dedent
⌘A	select all
⌘Z	undo
⌘⇧Z	redo
⌘Y	redo
⌘M	command mode
Esc	command mode
⌘⇧P	open the command palette
⇧↵	run cell, select below
⌘↵	run selected cells
⌥↵	run cell, insert below
⌘⇧Minus	split cell
⌘S	Save and Checkpoint
↓	move cursor down
↑	move cursor up

Edit mode: OSX only

⌘↑	go to cell start
⌘↓	go to cell end
⌥←	go one word left
⌥→	go one word right
⌥⌫	delete word before
⌥⌭	delete word after

Edit mode: Linux/PC

⌘Home	got to cell start
⌘↑	go to cell start
⌘End	go to cell end
⌘↓	go to cell end
⌘←	go one word left
⌘→	go one word right
⌘⌫	delete word before
⌘⌭	delete word after

Command mode shortcuts

↵	enter edit mode
F	find and replace
⌘⇧P	open the command palette
⇧↵	run cell, select below
⌘↵	run selected cells
⌥↵	run cell, insert below
Y	to code
M	to markdown
R	to raw
1	to heading 1
2	to heading 2
3	to heading 3
4	to heading 4
5	to heading 5
6	to heading 6
K	select cell above
↑	select cell above
↓	select cell below
J	select cell below
⇧K	extend selected cells above
⇧↑	extend selected cells above
⇧↓	extend selected cells below
⇧J	extend selected cells below
A	insert cell above
B	insert cell below
X	cut selected cells
C	copy selected cells
⇧V	paste cells above
V	paste cells below
Z	undo cell deletion
D,D	delete selected cells
⇧M	merge selected cells, or current cell with cell below if only one cell selected
⌘S	Save and Checkpoint
S	Save and Checkpoint
L	toggle line numbers
O	toggle output of selected cells
⇧O	toggle output scrolling of selected cells
H	show keyboard shortcuts
I,I	interrupt kernel
O,O	restart the kernel (with dialog)
Esc	close the pager
Q	close the pager
⇧↵	scroll notebook up
↵	scroll notebook down



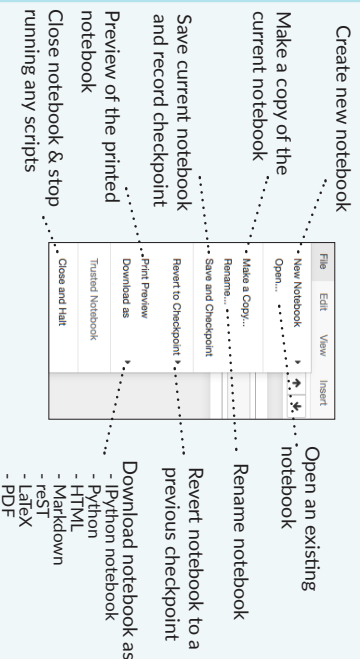
Python For Data Science Cheat Sheet

Jupyter Notebook

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



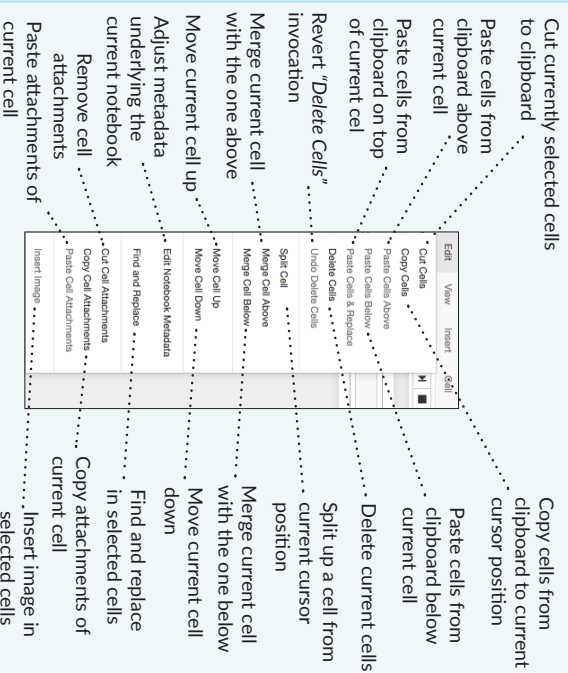
Saving/Loading Notebooks



Writing Code And Text

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

Edit Cells



Insert Cells



Working with Different Programming Languages

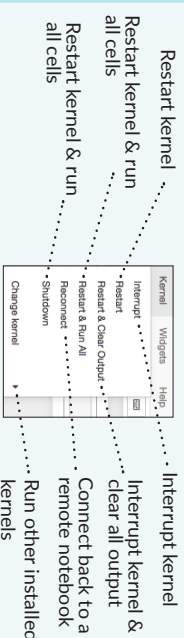
Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPyJ:
IPython

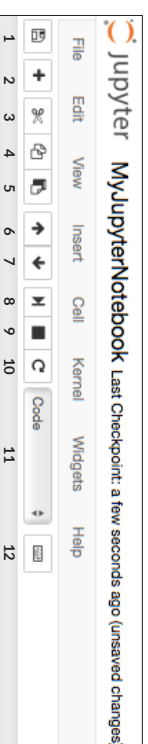
IR
IRkernel

IJ
Julia

Installing Jupyter Notebook will automatically install the IPython kernel.



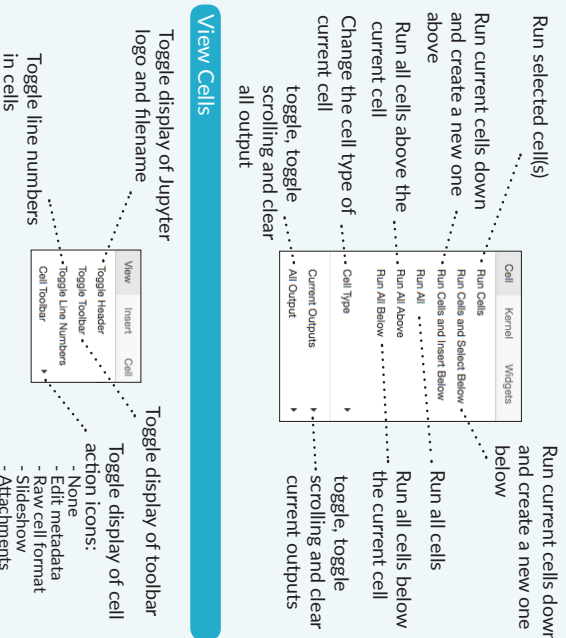
Command Mode:



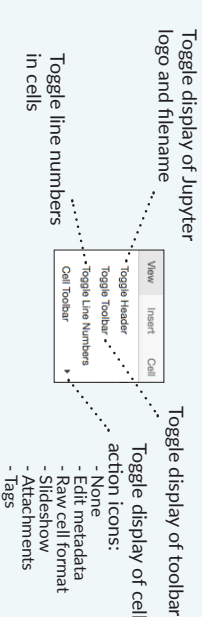
Edit Mode:



Executing Cells



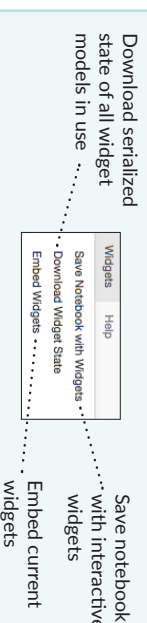
View Cells



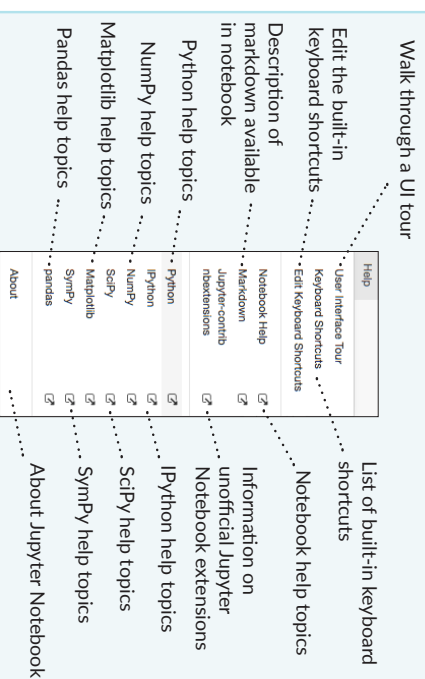
Widgets

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.



Asking For Help



Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit] : [file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

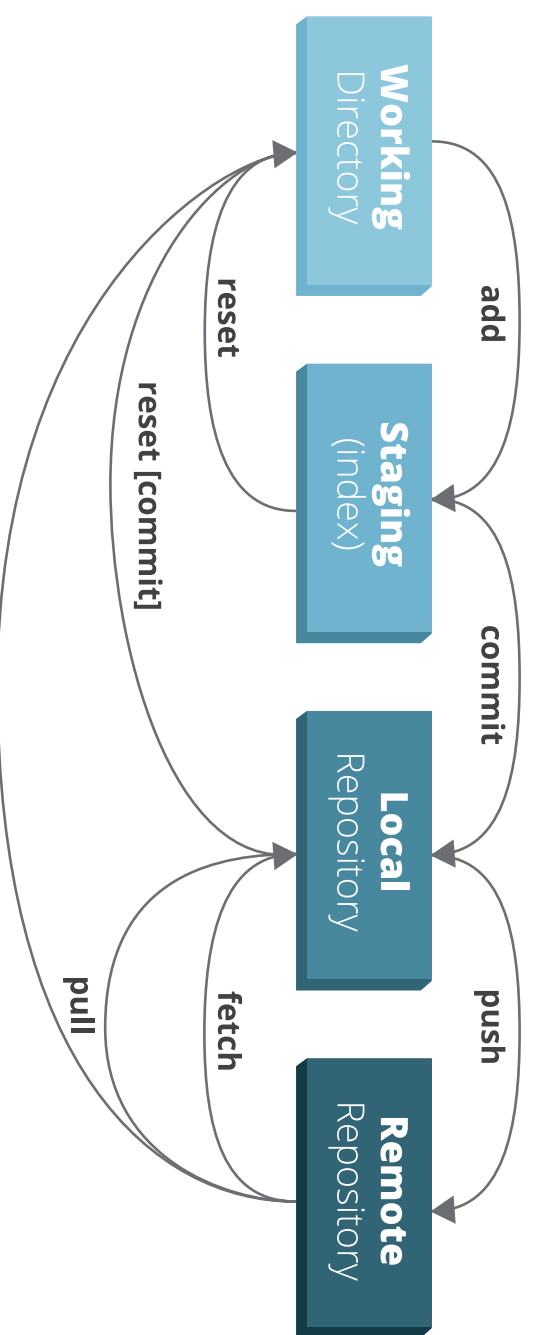
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.



Python For Data Science Cheat Sheet

Python Basics

Learn More Python for Data Science [Interactively](https://www.datacamp.com/interactively-at) at 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() int() float() bool()	'5', '3.45', 'True' 5, 3, 1 5.0, 1.0 True, True, True	Variables to strings Variables to integers Variables to floats Variables to booleans
-------------------------------------	--	---

Asking For Help

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

Strings

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

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Init!'
'thisStringIsAwesomeInit!'
>>> '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]
Slice
```

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
Subset Lists of Lists
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

Select item at index 1
Select 3rd last item
Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list
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
```

List Methods

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

String Operations

Index starts at 0

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

String Methods

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
Strip whitespaces

Libraries

Import libraries

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

pandas Data analysis
NumPy Scientific computing
Machine learning
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

>>> my_array[1] 2 Slice >>> my_array[0:2] array([1, 2]) Subset 2D NumPy arrays >>> my_2darray[:, 0] array([1, 4])	Select item at index 1 Select items at index 0 and 1 my_2darray[rows, columns]
--	--

NumPy Array Operations

```
>>> my_array > 3
array([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 >>> np.append(other_array) >>> np.insert(my_array, 1, 5) >>> np.delete(my_array, [1]) >>> np.mean(my_array) >>> np.median(my_array) >>> my_array.corrcoef() >>> np.std(my_array)	Get the dimensions of the array Append items to an array Insert items in an array Delete items in an array Mean of the array Median of the array Correlation coefficient Standard deviation
--	--



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:



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

Numpy Arrays

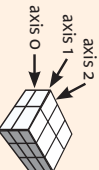
1D array

1	2	3
---	---	---

2D array

axis 1	1.5	2	3
axis 0	4	5	6

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([[ 0.6666667,  1.,  1.5],
       [ 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.]])
```

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

Select the element at the 2nd index
Select the element at row 0 column 2 (equivalent to b[1][2])
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:, :])
Same as [1, :, :]

Slicing

1	2	3
1.5	2	3
4	5	6

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

Boolean Indexing

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

1	2	3
---	---	---

Reversed array a

Select elements from a less than 2

Fancy Indexing

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

Array Manipulation

Transposing Array

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

Changing Array Shape

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

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(b,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.,  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 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.,  3.]])
>>> array([[ 3.,  2.,  6.]])
```

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

DataCamp

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



Python For Data Science Cheat Sheet

SciPy - Linear Algebra

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

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

Shape Manipulation

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

Polynomials

>>> from numpy import poly1d	Create a polynomial object
>>> p = poly1d([3,4,5])	

Vectorizing Functions

>>> def myfunc(a): if a < 0: return a*2 else: return a/2	
>>> np.vectorize(myfunc)	Vectorize functions

Type Handling

>>> np.real(b)	Return the real part of the array elements
>>> np.imag(b)	Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)	Return a real array if complex parts close to 0
>>> np.cast['F'](np.pi)	Cast object to a data type

Other Useful Functions

>>> np.angle(b,deg=True)	Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)	Create an array of evenly spaced values (number of samples)
>>> g[3:] += np.pi	Unwrap
>>> np.unwrap(g)	Create an array of evenly spaced values (log scale)
>>> np.linspace(0,10,3)	Return values from a list of arrays depending on conditions
>>> np.select([c<4],[c*2])	Factorial
>>> misc.factorial(a)	Combine N things taken at k time
>>> misc.comb(10,3,exact=True)	Weights for Np-point central derivative
>>> misc.central_diff_weights(3)	Find the n-th derivative of a function at a point
>>> misc.derivative(myfunc,1,0)	

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)
```

Transposition

```
>>> A.T
>>> A.H
```

Trace

```
>>> np.trace(A)
```

Norm

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

Rank

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

Determinant

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

Solving linear problems

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

Generalized Inverse

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

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)
>>> G = np.mat(np.identity(2))
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)
>>> I = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.ispmatrix_csc(A)
```

Sparse Matrix Routines

Inverse

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

Norm

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

Solving linear problems

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

Sparse Matrix Functions

>>> sparse.linalg.expm(I)	Sparse matrix exponential
---------------------------	---------------------------

Asking For Help

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

Matrix Functions

Addition

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

Subtraction

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

Division

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

Multiplication

```
>>> A @ D
```

```
>>> 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)
```

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

Exponential Functions

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

Logarithm Function

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

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

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

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

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

Trigonometric Functions

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

Hyperbolic Trigonometric Functions

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

Matrix Sign Function

```
>>> np.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

```
>>> linalg.signm(A)
```

Also see [NumPy](#)

Addition

Subtraction

Division

Multiplication operator

(Python 3)

Multiplication

Dot product

Vector dot product

Inner product

Outer product

Tensor dot product

Kronecker product

Matrix exponential

Matrix exponential (Taylor Series)

Matrix exponential (eigenvalue decomposition)

Matrix logarithm

Matrix sine

Matrix cosine

Matrix tangent

Hyperbolic matrix sine

Hyperbolic matrix cosine

Hyperbolic matrix tangent

Matrix sign function

Matrix square root

Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
>>> 11, 12 = la
>>> v[:,0]
>>> v[:,1]
```

```
>>> linalg.eigvals(A)
```

```
>>> U,s,Vh = linalg.svd(B)
```

```
>>> M,N = B.shape
```

```
>>> Sig = linalg.diagsvd(s,M,N)
```

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

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

```
>>> linalg.lu(C)
```

DataCamp

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



Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science [Interactively](https://www.datacamp.com/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
Index	b	-5
	c	7
	d	4

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

DataFrame

	Country	Capital	Population
Index	Belgium	Brussels	11190846
	India	New Delhi	1303171035
	Brazil	Brasilia	207847528

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

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
           'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
           '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
>>> xls = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Asking For Help

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

Selection

Also see NumPy Arrays

Geting

>>> s['b']	Get one element
-5	
>>> df[1:]	Get subset of a DataFrame
Country	Capital
1	India
2	Brazil

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      Brasilia
Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:, 'Capital']
```

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

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)
```

Dropping

>>> s.drop(['a', 'c'])	Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)	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	(rows, columns)
>>> df.index	Describe index
>>> df.columns	Describe DataFrame columns
>>> df.info()	Info on DataFrame
>>> df.count()	Number of non-NA values

Summary

>>> df.sum()	Sum of values
>>> df.cumsum()	Cumulative sum of values
>>> df.min()/df.max()	Minimum/maximum values
>>> df.idxmin()/df.idxmax()	Minimum/Maximum index value
>>> df.describe()	Summary statistics
>>> df.mean()	Mean of values
>>> df.median()	Median of values

Applying Functions

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



Python For Data Science Cheat Sheet

Pandas

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



Reshaping Data

Pivot

```
>>> df3 = df2.pivot(index='Date',
                    columns='Type',
                    values='Value')
```

Date	Type	Value
2016-03-01	a	11.432
2016-03-02	b	13.031
2016-03-01	c	20.784
2016-03-01	c	99.906
2016-03-02	a	1.303
2016-03-02	c	20.784

Date	Type	a	b	c
2016-03-01				
2016-03-01		11.432	NaN	20.784
2016-03-02		1.303	13.031	NaN
2016-03-03		99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,
                        values='Value',
                        index='Date',
                        columns='Type')
```

Stack / Unstack

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

	0	1
1	0.233482	0.390959
2	0.184713	0.237102
3	0.433522	0.429401

	1	5	0
1	0.390959	0.233482	0.237102
2	0.237102	0.184713	0.433522
3	0.433522	0.429401	0.233482

Melt

```
>>> pd.melt(df2,
            id_vars=["Date"],
            value_vars=["Type", "Value"],
            value_name="Observations")
```

Date	Type	Value
2016-03-01	a	11.432
2016-03-01	b	13.031
2016-03-01	c	20.784
2016-03-03	a	99.906
2016-03-02	a	1.303
2016-03-03	c	20.784

Date	Variable	Observations
2016-03-01	Type	a
2016-03-02	Type	b
2016-03-01	Type	c
2016-03-03	Type	a
2016-03-02	Type	c
2016-03-03	Type	a
2016-03-01	Value	11.432
2016-03-02	Value	13.031
2016-03-01	Value	20.784
2016-03-03	Value	99.906
2016-03-02	Value	1.303
2016-03-03	Value	20.784

Iteration

```
>>> df.iteritems()
>>> df.iterrows()
```

(Column-Index, Series) pairs
(Row-Index, Series) pairs

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:, (df3>1).any()]
>>> df3.loc[:, (df3>1).all()]
>>> df3.loc[:, df3.isnull().any()]
>>> df3.loc[:, df3.notnull().all()]
>>> df3.loc[:, df3.notnull().all()]
>>> df[df['country'].isin(df2['Type'])]
>>> df3.filter(items=["a", "b"])
>>> df.select(lambda x: not x%5)
```

Where

```
>>> s.where(s > 0)
```

Query

```
>>> df6.query('second > first')
```

Setting/Resetting Index

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                  columns=('country': "country",
                           "capital": "cptl",
                           "population": "ppltn"))
```

Reindexing

```
>>> s2 = s.reindex(['a', 'c', 'd', 'e', 'b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
>>> s3 = s.reindex(range(5),
               method='ffill')
```

Backward Filling

```
Country    Capital    Population
0    Belgium    Brussels    1
1    India    New Delhi    1303171035
2    Brazil    Brasilia    207847528
3    Brazil    Brasilia    207847528
```

Multindexing

```
>>> arrays = np.array([1,2,3])
>>> np.array([5,4,3])
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                     names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(['Date', "Type"])
```

Duplicate Data

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

Return unique values
Check duplicates
Drop duplicates
Check index duplicates

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date', 'Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a': lambda x: sum(x)/len(x),
                             'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "z")
```

Drop NaN values
Fill NaN values with a predetermined value
Replace values with others

Combining Data

data1		data2	
X1	X2	X1	X3
a	11.432	a	20.784
b	1.303	b	NaN
c	99.906	d	20.784

Merge

```
>>> pd.merge(data1,
             data2,
             how='left',
             on='X1')
```

```
>>> pd.merge(data1,
             data2,
             how='right',
             on='X1')
```

```
>>> pd.merge(data1,
             data2,
             how='inner',
             on='X1')
```

```
>>> pd.merge(data1,
             data2,
             how='outer',
             on='X1')
```

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

```
>>> s.append(s2)
>>> pd.concat([s,s2], axis=1, keys=['One', 'Two'])
>>> pd.concat([data1, data2], axis=1, join='inner')
```

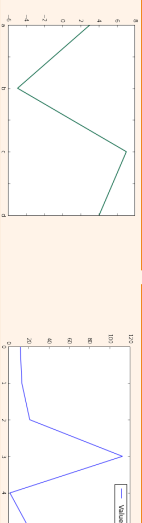
Dates

```
>>> df2['Date'] = pd.to_datetime(df2['Date'])
>>> df2['Date'] = pd.date_range('2000-1-1',
                              periods=6,
                              freq='W')
>>> dates = [datetime(2012, 5, 1), datetime(2012, 5, 2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012, 2, 1), end, freq='BM')
```

Visualization

Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> s.plot()
>>> plt.show()
>>> df2.plot()
>>> plt.show()
```



Python For Data Science Cheat Sheet

Matplotlib

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



matplotlib

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]
>>> V = 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=2)
```

3 Plotting Routines

1D Data

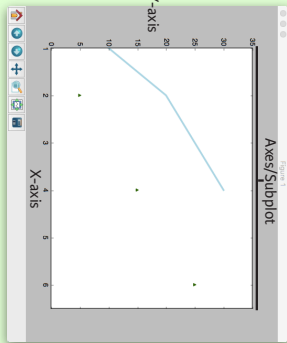
Code	Description
>>> lines = ax.plot(x,y)	Draw points with lines or markers connecting them
>>> ax.scatter(x,y)	Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])	Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1.2,5],[0,1,2])	Plot horizontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)	Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)	Draw a vertical line across axes
>>> ax.fill(x,Y,color='blue')	Draw filled polygons
>>> ax.fill_between(x,Y,color='yellow')	Fill between y-values and o

2D Data or Images

Code	Description
>>> fig, ax = plt.subplots()	
>>> im = ax.imshow(img, cmap='gist_earth', interpolation='nearest', vmin=-2, vmax=2)	Colormapped or RGB arrays

Plot Anatomy & Workflow

Plot Anatomy



Figure

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='v')
>>> 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="r")
>>> 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='--',x**2,y**2,'-.')
>>> 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=(6, 0), xycoords='data', xtext=10.5, ytext=10.5, textcoords='data', arrowprops=dict(arrowstyle="->", connectionstyle="arc3"),)
```

Mathtext

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

Limits, Legends & Layouts

```
>>> ax.margins(x=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', 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)
```

Axis Spines

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

Adjust the spacing between subplots

Manually set x-ticks

Make y-ticks longer and go in and out

direction='inout', length=10

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



Python For Data Science Cheat Sheet

Bokeh

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

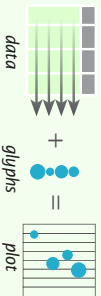


Plotting With Bokeh

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: Python lists, NumPy arrays, Pandas DataFrames and other sequences of values
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)
>>> 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")
```

Rows & Columns Layout

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

Grid Layout

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

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])
```

Legends

Legend Location

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

4 Output

Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Notebook Output

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

Embedding

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, "my_plot")
>>> from bokeh.embed import components
>>> script, div = components(p)
```

5 Show or Save Your Plots

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

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

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

Colormapping

```
>>> 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')
```

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)
```

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"
```

Statistical Charts With Bokeh

Also see [Data](#)

Bokeh's high-level bokeh.charts interface is ideal for quickly creating statistical charts

Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red', 'blue'])
```

Box Plot

```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
>>>              legend='bottom_right')
```

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y='hp', marker='square',
>>>             xlabel='Miles Per Gallon',
>>>             ylabel='Horsepower')
```

DataCamp

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



Python For Data Science Cheat Sheet

Seaborn

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



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

>>> 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",
                  {"tick.major.size":8,
                   "tick.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 with to temporarily set the style

3 Plotting With Seaborn

Axis Grids

>>> g = sns.FacetGrid(titanic, row="survived", col="sex") >>> g = g.map(plt.hist, "age") >>> sns.factorplot(x="class", 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 FaceGrid Plot data and regression model fits across a FaceGrid
---	---

Categorical Plots

Scatterplot >>> sns.stripplot(x="species", y="petal_length", data=iris) >>> sns.swarmplot(x="species", y="petal_length", data=iris) Bar Chart >>> sns.barpplot(x="sex", y="survived", hue="class", data=titanic) Count Plot >>> sns.countplot(x="deck", data=titanic, palette="Greens_d") Point Plot >>> sns.pointplot(x="class", y="survived", hue="sex", data=titanic, palette="magma", markers=["o", "o"], linestyle=["-", "-"]) Boxplot >>> sns.boxplot(x="alive", y="age", hue="adult_male", data=titanic) >>> sns.boxplot(data=iris, orient="h") Violinplot >>> sns.violinplot(x="age", y="sex", hue="survived", data=titanic)	Scatterplot with one categorical variable Categorical scatterplot with non-overlapping points Show point estimates and confidence intervals with scatterplot glyphs Show count of observations Show point estimates and confidence intervals as rectangular bars Boxplot with wide-form data Violin plot
--	--

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                  font_scale=1.5,
                  rc={"lines.linewidth":2.5})
```

Color Palette

```
>>> sns.set_palette("husl", 3)
>>> sns.color_palette("husl")
>>> flatui = ["#959595", "#3498db", "#955555", "#e74c3c", "#34495e", "#2ecc71"]
>>> sns.set_palette(flatui)
```

Define the color palette

Use with with to temporarily set palette scale font elements and override param mapping

Also see [Matplotlib](#)

>>> 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
--	--

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(uni_form_data, vmin=0, vmax=1)
```

Heatmap

4 Further Customizations

Also see [Matplotlib](#)

Axisgrid Objects

>>> g.deapline(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), ticks=[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

>>> plt.clf() >>> plt.close() >>> plt.close()	Clear an axis Clear an entire figure Close a window
---	---

Also see [Matplotlib](#)



Python For Data Science Cheat Sheet

Scikit-Learn

Learn Python for data science [Interactively at www.DataCamp.com](https://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', 'M', 'F', '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')
```

Naïve 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

Supervised learning

```
>>> lr.fit(X, y)
```

```
>>> knn.fit(X_train, y_train)
```

```
>>> svc.fit(X_train, y_train)
```

Unsupervised Learning

```
>>> k_means.fit(X_train)
```

```
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data

Fit the model to the data

Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
```

```
>>> y_pred = lr.predict(X_test)
```

```
>>> y_pred = knn.predict_proba(X_test)
```

Unsupervised Estimators

```
>>> y_pred = k_means.predict(X_test)
```

Predict labels

Predict labels

Estimate probability of a label

Predict labels in clustering algos

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test)
```

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R² Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> 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),
"n_jobs": [-1, 1], "distance": 1}
>>> rsearch = RandomizedSearchCV(estimator=knn,
param_distributions=params,
n_iter=8,
random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp

Learn Python for Data Science Interactively



Python For Data Science Cheat Sheet

Keras

Learn Python for data science [Interactively at www.DataCamp.com](https://www.datacamp.com/interactively-at)



Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
activation='relu',
input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
mnist,
cifar10,
imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/plima-indians-diabetes/
plima-indians-diabetes.data"),delimiter=",")
>>> x = data[:,0:8]
>>> y = data[:,8]
```

Preprocessing

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> y_train = to_categorical(y_train,num_classes)
>>> y_test = to_categorical(y_test,num_classes)
>>> y_train3 = to_categorical(y_train3,num_classes)
>>> y_test3 = to_categorical(y_test3,num_classes)
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
input_dim=8,
kernel_initializer='uniform',
activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3),padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Also see NumPy & Scikit-Learn

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> x_train3,x_test3,y_train3,y_test3 = train_test_split(x,
y_train3,
test_size=0.33,
random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_x = scaler.transform(x_train2)
>>> standardized_x_test = scaler.transform(x_test2)
```

Inspect Model

>>> model.output_shape	Model output shape
>>> model.summary()	Model summary representation
>>> model.get_config()	Model configuration
>>> model.get_weights()	List all weight tensors in the model

Compile Model

```
>>> model.compile(optimizer='adam',
loss='binary_crossentropy',
metrics=['accuracy'])
>>> model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',
metrics=['accuracy'])
>>> model.compile(optimizer='rmsprop',
loss='mse',
metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
optimizer='adam',
metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
y_train4,
batch_size=32,
epochs=15,
verbose=1,
validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
y_test,
batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4,batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_h1e.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001,decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
optimizer=opt,
metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
y_train4,
batch_size=32,
epochs=15,
validation_data=(x_test4,y_test4),
callbacks=[early_stopping_monitor])
```

DataCamp

Learn Python for Data Science Interactively



Python For Data Science Cheat Sheet

PySpark - RDD Basics

Learn Python for data science *Interactively* at www.DataCamp.com



Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.



Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

```
>>> sc.version
>>> sc.pythonVer
>>> sc.master
>>> str(sc.sparkHome)
>>> str(sc.sparkUser())
>>> sc.appName
>>> sc.applicationId
>>> sc.defaultParallelism
>>> sc.defaultMinPartitions
>>> sc.defaultMaxPartitions
```

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
>>>         .setMaster("local*")
>>>         .setAppName("My app"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python `.zip`, `.egg` or `.py` files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([(('a', 7), ('a', 2), ('b', 2))])
>>> rdd2 = sc.parallelize([(('a', 2), ('a', 1), ('b', 1))])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([(("a", "x", "y", "z"),
>>>                        ("b", "p", "q", "r"))])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFiles = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

>>> rdd.getNumPartitions()	List the number of partitions
>>> rdd.count()	Count RDD instances
>>> rdd.countByKey()	Count RDD instances by key
>>> defaultDict(lambda x: 1, ('a', 2), ('a', 7))	Count RDD instances by value
>>> rdd.countByValue()	Return (key,value) pairs as a dictionary
>>> rdd.collect()	Sum of RDD elements
>>> rdd.sum()	Check whether RDD is empty
>>> sc.parallelize(1).isEmpty()	

Summary

>>> rdd3.max()	Maximum value of RDD elements
>>> rdd3.min()	Minimum value of RDD elements
>>> rdd3.mean()	Mean value of RDD elements
>>> rdd3.stdev()	Standard deviation of RDD elements
>>> rdd3.variance()	Compute variance of RDD elements
>>> rdd3.histogram()	Compute histogram by bins
>>> rdd3.stats()	Summary statistics (count, mean, stdev, max & min)

Applying Functions

>>> rdd.map(lambda x: x*(x[1], x[0]))	Apply a function to each RDD element
>>> rdd3.map(lambda x: x*(x[1], x[0]))	Apply a function to each RDD element and flatten the result
>>> rdd5.collect()	Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys

Selecting Data

>>> rdd.collect()	Return a list with all RDD elements
>>> rdd.take(2)	Take first 2 RDD elements
>>> rdd.first()	Take first RDD element
>>> rdd.top(2)	Take top 2 RDD elements
>>> rdd3.sample(False, 0.15, 81).collect()	Return sampled subset of rdd3
>>> rdd3.distinct().collect()	Filter the RDD
>>> rdd.filter(lambda x: "a" in x)	Return distinct RDD values
>>> rdd.keys().collect()	Return (key,value) RDD's keys

Iterating

>>> def g(x): print(x)	Apply a function to all RDD elements
>>> rdd.foreach(g)	

Reshaping Data

Reducing

>>> rdd.reduceByKey(lambda x, y: x+y)	Merge the rdd values for each key
>>> rdd.reduce(lambda a, b: a + b)	Merge the rdd values
>>> rdd3.groupBy(lambda x: x % 2).mapValues(list)	Return RDD of grouped values
>>> rdd.groupByKey().mapValues(list)	Group rdd by key

Aggregating

>>> seqOp = (lambda x, y: (x[0]+y, x[1]+1))	Aggregate RDD elements of each partition, and then the results
>>> combOp = (lambda x, y: (x[0]+y[0], x[1]+y[1]))	Aggregate values of each RDD key
>>> rdd.aggregate((0,0), seqOp, combOp)	Aggregate the elements of each partition, and then the results
>>> rdd.foldByKey(0, add)	Merge the values for each key
>>> rdd.fold(0, add)	Create tuples of RDD elements by applying a function

Mathematical Operations

>>> rdd.subtract(rdd2)	Return each rdd value not contained in rdd2
>>> rdd2.subtractByKey(rdd)	Return each (key,value) pair of rdd2 with no matching key in rdd
>>> rdd.cartesian(rdd2).collect()	Return the Cartesian product of rdd and rdd2

Sort

>>> rdd2.sortBy(lambda x: x[1])	Sort RDD by given function
>>> rdd2.sortByKey()	Sort (key, value) RDD by key

Repartitioning

>>> rdd.repartition(4)	New RDD with 4 partitions
>>> rdd.coalesce(1)	Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
>>>                       org.apache.hadoop.mapred.TextInputFormat)
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```



Python For Data Science Cheat Sheet

PySpark - SQL Basics

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



PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.



Initializing SparkSession

A SparkSession can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files.

```
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Creating DataFrames

From RDDs

```
>>> from pyspark.sql.types import *
>>> InferSchema
>>> sc = spark.sparkContext
>>> lines = sc.textFile("people.txt")
>>> parts = lines.map(lambda l: l.split(","))
>>> people = parts.map(lambda p: Row(name=p[0],age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

Specify Schema

```
>>> people = parts.map(lambda p: Row(name=p[0],
    age=int(p[1]).strip()))
>>> schemestring = "name age"
>>> fields = StructField(field name, StringType(), True) for
    field name in schemestring.split()
>>> schema = StructType(fields)
>>> spark.createDataFrame(people, schema).show()
```

```
+----+-----+
|name|age|
+----+-----+
|Mike|28|
|John|30|
+----+-----+
```

From Spark Data Sources

JSON

```
>>> df = spark.read.json("customer.json")
>>> df.show()
+-----+-----+-----+-----+
|address|age|firstName|lastName|phoneNumber|
+-----+-----+-----+-----+
|New York,10021,N...|25|John|Smith|[1312 555-1234,ho...|
|New York,10021,N...|21|Jane|Doe|[1322 888-1234,ho...|
+-----+-----+-----+-----+
```

Parquet files

```
>>> df2 = spark.read.load("people.json", format="json")
>>> df3 = spark.read.load("users.parquet")
>>> df4 = spark.read.text("people.txt")
```

Inspect Data

```
>>> df.types
Return df column names and data types
>>> df.show()
Display the content of df
>>> df.head()
Return first n rows
>>> df.first()
Return first row
>>> df.take(2)
Return the first n rows
>>> df.schema
Return the schema of df
```

Duplicate Values

```
>>> df = df.dropDuplicates()
```

Queries

```
>>> from pyspark.sql import functions as F
>>> Select
>>> df.select("firstName").show()
Show all entries in firstName column
>>> df.select("firstName","lastName") \
    .show()
Show all entries in firstName, age
and type
>>> df.select("firstName",
    explode("phoneNumber") \
    .alias("contactInfo") \
    .select("contactInfo.type",
    "firstName",
    "age") \
    .show()
Show all entries in firstName and age,
add 1 to the entries of age
>>> df.select(df["firstName"],df["age"]+1)
add 1 to the entries of age
>>> df.select(df["age"] > 24).show()
Show all entries where age >24
>>> When
>>> df.select("firstName",
    .show()
Show firstName and 0 or 1 depending
on age >30
>>> df.select(df.firstName.isin("Jane","Boris"])
Show firstName if in the given options
>>> Like
>>> df.select("firstName",
    df.lastName.like("Smith")) \
    .show()
Show firstName, and lastName is
TRUE if lastName is like Smith
>>> df.select("firstName",
    df.lastName \
    .startswith("Sm")) \
    .show()
Show firstName and TRUE if
lastName starts with Sm
>>> df.select(df.lastName.endswith("ch")) \
    .show()
Show last names ending in ch
>>> df.select(df.firstName.substr(1, 3) \
    .alias("name")) \
    .collect()
Return substrings of firstName
>>> df.select(df.age.between(22, 24)) \
    .show()
Show ages values are TRUE if between
22 and 24
```

Add, Update & Remove Columns

Adding Columns

```
>>> df = df.withColumn('city',df.address.city) \
    .withColumn('postalCode',df.address.postalCode) \
    .withColumn('state',df.address.state) \
    .withColumn('streetAddress',df.address.streetAddress) \
    .withColumn('telephoneNumber',
    explode(df.phoneNumber.number) \
    .withColumn('telephoneType',
    explode(df.phoneNumber.type))
```

Updating Columns

```
>>> df = df.withColumnRenamed('telephoneNumber', 'phoneNumber')
```

Removing Columns

```
>>> df = df.drop("address", "phoneNumber")
>>> df = df.drop(df.address).drop(df.phoneNumber)
>>> df.describe().show()
Compute summary statistics
>>> df.columns
Return the columns of df
>>> df.count()
Count the number of rows in df
>>> df.distinct().count()
Count the number of distinct rows in df
>>> df.printSchema()
Print the schema of df
>>> df.explain()
Print the (logical and physical) plans
```

GroupBy

```
>>> df.groupBy("age") \
    .count() \
    .show()
Group by age, count the members
in the groups
```

Filter

```
>>> df.filter(df["age"]>24).show()
Filter entries of age, only keep those
records of which the values are >24
```

Sort

```
>>> peopledf.sort(peopledf.age.desc()).collect()
>>> df.sort("age",ascending=False).collect()
>>> df.orderBy("age","city",ascending=[0,1]) \
    .collect()
```

Missing & Replacing Values

```
>>> df.na.fill(50).show()
Replace null values
>>> df.na.drop().show()
Return new df omitting rows with null values
>>> df.na \
    .replace(10, 20) \
    .show()
Return new df replacing one value with
another
```

Repartitioning

```
>>> df.repartition(10) \
    .rd \
    .getPartitions()
df with 10 partitions
>>> df.coalesce(1).rd.getNumPartitions()
df with 1 partition
```

Running SQL Queries Programmatically

Registering DataFrames as Views

```
>>> peopledf.createGlobalTempView("people")
>>> df.createTempView("customer")
>>> df.createOrReplaceTempView("customer")
```

Query Views

```
>>> df5 = spark.sql("SELECT * FROM customer").show()
>>> peopledf2 = spark.sql("SELECT * FROM global_temp.people") \
    .show()
```

Output

Data Structures

```
>>> rdd1 = df.rdd
>>> df.toJSON().first()
Convert df into an RDD
>>> df.toPandas()
Convert df into a RDD of string
Return the contents of df as Pandas
DataFrame
```

Write & Save to Files

```
>>> df.select("firstName", "city") \
    .write \
    .save("nameAndCity.parquet")
>>> df.select("firstName", "age") \
    .write \
    .save("nameAndAges.json",format="json")
```

Stopping SparkSession

```
>>> spark.stop()
```

