Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively 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()`	`'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)
```

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]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	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
```

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

Strings

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

String Operations

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

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

learn — 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
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp

Learn Python for Data Science Interactively

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

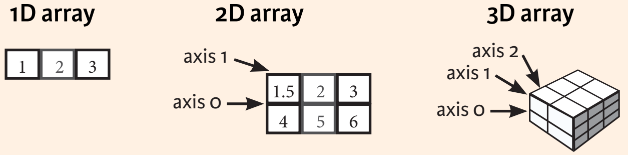
NumPy

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



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))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	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`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing TRUE and FALSE values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	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. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5,  4. ,  6. ],`
`       [ 5. ,  7. ,  9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667,  1.        ,  1.        ],`
`      [ 0.25      ,  0.4       ,  0.5       ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[  1.5,   4. ,   9. ],`
`       [  4. ,  10. ,  18. ]])`
	Multiplication

`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm

`>>> e.dot(f)`
`array([[ 7.,  7.],`
`       [ 7.,  7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False,  True,  True],`
`       [False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2`
`array([True, False, False], dtype=bool)`
	Element-wise comparison

`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

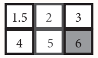
`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

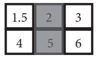
Also see Lists

Subsetting	

`>>> a[2]`
`3`
	Select the element at the 2nd index

`>>> b[1,2]` `6.0` 	Select the element at row 1 column 2 (equivalent to b[1][2])
Slicing	

`>>> a[0:2]`
`array([1, 2])`
	Select items at index 0 and 1

`>>> b[0:2,1]` `array([ 2.,  5.])` 	Select items at rows 0 and 1 in column 1

`>>> b[:1]`
`array([[1.5, 2., 3.]])`
	Select all items at row 0 (equivalent to `b[0:1, :]`)

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

`>>> a[ : :-1]`
`array([3, 2, 1])`
	Reversed array a

Boolean Indexing	

`>>> a[a<2]`
`array([1])`
	Select elements from a less than 2

Fancy Indexing	

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

`>>> 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 a subset of the matrix's rows and columns

Array Manipulation

Transposing Array	
`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions
Changing Array Shape	
`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data
Adding/Removing Elements	
`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array
Combining Arrays	

`>>> np.concatenate((a,d),axis=0)`
`array([ 1,  2,  3, 10, 15, 20])`
	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. ,  2. ,  3. ],`
`       [ 1.5,  2. ,  3. ],`
`       [ 4. ,  5. ,  6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7.,  7.,  1.,  0.],`
`       [ 7.,  7.,  0.,  1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`       [ 2, 15],`
`       [ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays
Splitting Arrays	

`>>> np.hsplit(a,3)`
`[array([1]),array([2]),array([3])]`
	Split the array horizontally at the 3rd index

`>>> np.vsplit(c,2)`
`[array([[[ 1.5,  2. ,  1. ],`
`        [ 4. ,  5. ,  6. ]]]),`
` array([[[ 3.,  2.,  3.],`
`        [ 4.,  5.,  6.]]])]`
	Split the array vertically at the 2nd index

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

Pandas

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

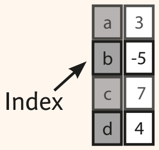
Use the following import convention:

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

Pandas Data Structures

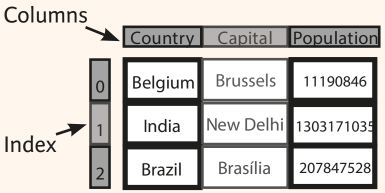
Series

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



```
>>> 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'])
```

Asking For Help

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

Selection

Also see NumPy Arrays

Getting

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

Get one element

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

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     Brasília
```

Select a single column of subset of columns

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

Select rows and columns

Boolean Indexing

`>>> s[~(s > 1)]`	Series s where value is not >1
`>>> s[(s < -1) | (s > 2)]`	s where value is <-1 or >2
`>>> df[df['Population']>1200000000]`	Use filter to adjust DataFrame

Setting

`>>> s['a'] = 6`	Set index a of Series s to 6

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')
>>> df.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')
```

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()`

```
>>> df.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()`	Sort by labels along an axis
`>>> df.sort_values(by='Country')`	Sort by the values along an axis
`>>> df.rank()`	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()`	Cummulative 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)
```

DataCamp

Learn Python for Data Science **Interactively**

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python **Interactively** at 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]
>>> 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))
```

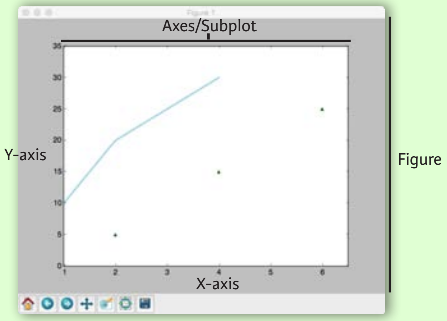
Axes

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)
```

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

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)`	Add padding to a plot
`>>> ax.axis('equal')`	Set the aspect ratio of the plot to 1
`>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])`	Set limits for x-and y-axis
`>>> ax.set_xlim(0,10.5)`	Set limits for x-axis

Legends

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

Set a title and x-and y-axis labels

```
>>> ax.legend(loc='best')
```

No overlapping plot elements

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
```

Manually set x-ticks

```
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

Make y-ticks longer and go in and out

Subplot Spacing

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

Adjust the spacing between subplots

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

Fit subplot(s) in to the figure area

Axis Spines

`>>> ax1.spines['top'].set_visible(False)`	Make the top axis line for a plot invisible
`>>> ax1.spines['bottom'].set_position(('outward',10))`	Move the bottom axis line outward

3 Plotting Routines

1D Data

`>>> 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 horiontal 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 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

`>>> axes2[0].pcolor(data2)`	Pseudocolor plot of 2D array
`>>> axes2[0].pcolormesh(data)`	Pseudocolor plot of 2D array
`>>> CS = plt.contour(Y,X,U)`	Plot contours
`>>> axes2[2].contourf(data1)`	Plot filled contours
`>>> axes2[2]= ax.clabel(CS)`	Label a contour plot

Vector Fields

`>>> axes[0,1].arrow(0,0,0.5,0.5)`	Add an arrow to the axes
`>>> axes[1,1].quiver(y,z)`	Plot a 2D field of arrows
`>>> axes[0,1].streamplot(X,Y,U,V)`	Plot 2D vector fields

Data Distributions

`>>> ax1.hist(y)`	Plot a histogram
`>>> ax3.boxplot(y)`	Make a box and whisker plot
`>>> ax3.violinplot(z)`	Make a violin plot

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

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science **Interactively** at 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")        Step 1
>>> sns.set_style("whitegrid")         Step 2
>>> g = sns.lmplot(x="tip",                Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))
                                         Step 4
>>> plt.title("title")
>>> plt.show(g)                 Step 5
```

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()
```

(Re)set the seaborn default

```
>>> sns.set_style("whitegrid")
```

Set the matplotlib parameters

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

Set the matplotlib parameters

```
>>> sns.axes_style("whitegrid")
```

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

Context Functions

```
>>> sns.set_context("talk")
```

Set context to "talk"

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

Set context to "notebook", scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
```

Define the color palette

```
>>> sns.color_palette("husl")
```

Use with `with` to temporarily set palette

```
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```

Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
```

Subplot grid for plotting conditional relationships

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

Draw a categorical plot onto a Facetgrid

```
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
```

Subplot grid for plotting pairwise relationships

```
>>> sns.pairplot(iris)
```

Plot pairwise bivariate distributions

```
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
```

Grid for bivariate plot with marginal univariate plots

```
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```

Plot bivariate distribution

Categorical Plots

Scatterplot

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

Scatterplot with one categorical variable

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

Categorical scatterplot with non-overlapping points

Bar Chart

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

Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

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

Show count of observations

Point Plot

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

Show point estimates and confidence intervals as rectangular bars

Boxplot

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

Boxplot

```
>>> sns.boxplot(data=iris,orient="h")
```

Boxplot with wide-form data

Violinplot

```
>>> 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)
```

Remove left spine

```
>>> g.set_ylabels("Survived")
```

Set the labels of the y-axis

```
>>> g.set_xticklabels(rotation=45)
```

Set the tick labels for x

```
>>> g.set_axis_labels("Survived",
                      "Sex")
```

Set the axis labels

```
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```

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

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','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)
```

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

Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

Fit the model to the data

Unsupervised Learning

```
>>> 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

Supervised Estimators

```
>>> 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

Unsupervised Estimators

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

Predict labels in clustering algos

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)
```

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)
```

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)
```

Estimator score method

Metric scoring functions

Classification Report

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

Precision, recall, f1-score and support

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)
```

R^2 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(1,3),
              "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_)
```

DataCamp

Learn Python for Data Science **Interactively**

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively 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.r_[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.vpslit(d,2)`	Split the array vertically at the 2nd index

Polynomials

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

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`	
`>>> np.unwrap(g)`	Unwrap
`>>> np.logspace(0,10,3)`	Create an array of evenly spaced values (log scale)
`>>> np.select([c<4],[c*2])`	Return values from a list of arrays depending on conditions
`>>> misc.factorial(a)`	Factorial
`>>> misc.comb(10,3,exact=True)`	Combine N things taken at k time
`>>> misc.central_diff_weights(3)`	Weights for Np-point central derivative
`>>> misc.derivative(myfunc,1.0)`	Find the n-th derivative of a function at a point

Linear Algebra

Also see NumPy

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`	Inverse
`>>> linalg.inv(A)`	Inverse
Transposition	
`>>> A.T`	Tranpose matrix
`>>> A.H`	Conjugate transposition
Trace	
`>>> np.trace(A)`	Trace
Norm	
`>>> linalg.norm(A)`	Frobenius norm
`>>> linalg.norm(A,1)`	L1 norm (max column sum)
`>>> linalg.norm(A,np.inf)`	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)`	Solver for dense matrices
`>>> E = np.mat(a).T`	Solver for dense matrices
`>>> linalg.lstsq(F,E)`	Least-squares solution to linear matrix equation
Generalized inverse	
`>>> linalg.pinv(C)`	Compute the pseudo-inverse of a matrix (least-squares solver)
`>>> linalg.pinv2(C)`	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

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

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

Matrix Functions

Addition	
`>>> np.add(A,D)`	Addition
Subtraction	
`>>> np.subtract(A,D)`	Subtraction
Division	
`>>> np.divide(A,D)`	Division
Multiplication	
`>>> A @ D`	Multiplication operator (Python 3)
`>>> np.multiply(D,A)`	Multiplication
`>>> np.dot(A,D)`	Dot product
`>>> np.vdot(A,D)`	Vector dot product
`>>> np.inner(A,D)`	Inner product
`>>> np.outer(A,D)`	Outer product
`>>> np.tensordot(A,D)`	Tensor dot product
`>>> np.kron(A,D)`	Kronecker product
Exponential Functions	
`>>> linalg.expm(A)`	Matrix exponential
`>>> linalg.expm2(A)`	Matrix exponential (Taylor Series)
`>>> linalg.expm3(D)`	Matrix exponential (eigenvalue decomposition)
Logarithm Function	
`>>> linalg.logm(A)`	Matrix logarithm
Trigonometric Functions	
`>>> linalg.sinm(D)`	Matrix sine
`>>> linalg.cosm(D)`	Matrix cosine
`>>> linalg.tanm(A)`	Matrix tangent
Hyperbolic Trigonometric Functions	
`>>> linalg.sinhm(D)`	Hypberbolic matrix sine
`>>> linalg.coshm(D)`	Hyperbolic matrix cosine
`>>> linalg.tanhm(A)`	Hyperbolic matrix tangent
Matrix Sign Function	
`>>> np.signm(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)`	Solve ordinary or generalized eigenvalue problem for square matrix
`>>> l1, l2 = la`	Unpack eigenvalues
`>>> v[:,0]`	First eigenvector
`>>> v[:,1]`	Second eigenvector
`>>> linalg.eigvals(A)`	Unpack eigenvalues
Singular Value Decomposition	
`>>> U,s,Vh = linalg.svd(B)`	Singular Value Decomposition (SVD)
`>>> M,N = B.shape`	
`>>> Sig = linalg.diagsvd(s,M,N)`	Construct sigma matrix in SVD
LU Decomposition	
`>>> P,L,U = linalg.lu(C)`	LU Decomposition

Sparse Matrix Decompositions

`>>> la, v = sparse.linalg.eigs(F,1)`	Eigenvalues and eigenvectors
`>>> sparse.linalg.svds(H, 2)`	SVD

Asking For Help

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

DataCamp

Learn Python for Data Science Interactively