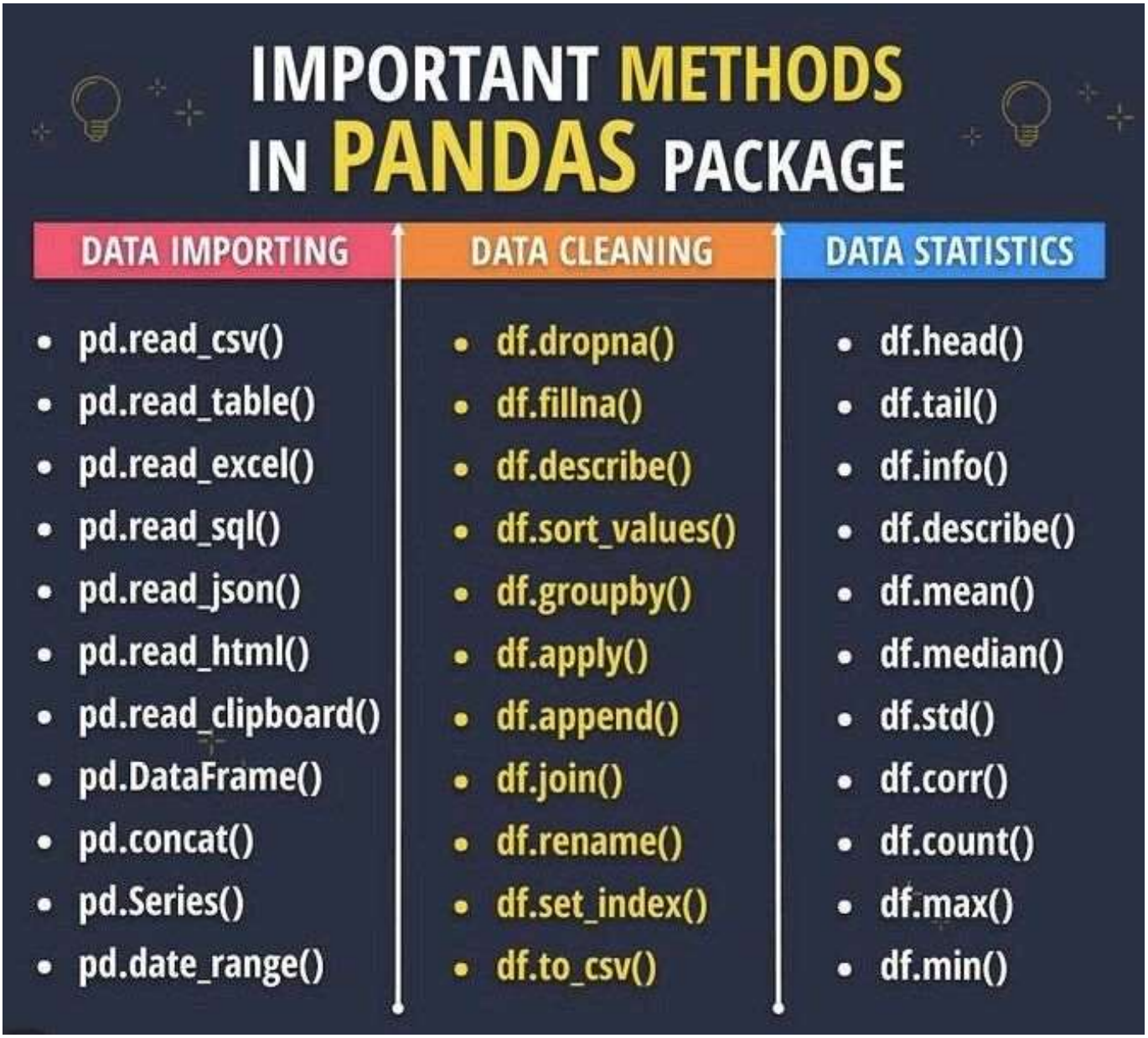
Resources for Pandas, Numpy, Seaborn, and Matplotlib



Pandas Cheat Sheet

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

Index

a	3
b	-5
c	7
d	4

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

DataFrame

Columns

Index

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	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'])
```

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  Brasilia  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  Brasilia`
`Population  207847528`
	Select single row of subset of rows

`>>> df.ix[:,'Capital']`
`0  Brussels`
`1  New Delhi`
`2  Brasilia`
	Select a single column of subset of columns

`>>> df.ix[1,'Capital']` `'New Delhi'`	Select rows and columns
Boolean Indexing	
`>>> s[~(s > 1)]`	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

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)
```

I/O

Read and Write to CSV

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

Read and Write to Excel

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

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

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

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

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

DataCamp
Learn Python for Data Science Interactively

Numpy Cheat Sheet

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

1D array | 2D array | 3D array

axis 1, axis 0

axis 2, axis 1, axis 0

Creating Arrays

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

Initial Placeholders

`>>> np.zeros((3,4))`	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`	Subtraction
`>>> np.subtract(a,b)`	Subtraction
`>>> b + a`	Addition
`>>> np.add(b,a)`	Addition
`>>> a / b`	Division
`>>> np.divide(a,b)`	Division
`>>> a * b`	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)`	Dot product

Comparison

`>>> a == b`	Element-wise comparison
`>>> a < 2`	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]`	Select the element at the 2nd index
`>>> b[1,2]`	Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing

`>>> a[0:2]`	Select items at index 0 and 1
`>>> b[0:2,1]`	Select items at rows 0 and 1 in column 1
`>>> b[:1]`	Select all items at row 0 (equivalent to `b[0:1, :]`)
`>>> c[1,...]`	Same as `[1,:,:]`
`>>> a[ : :-1]`	Reversed array `a`

Boolean Indexing

`>>> a[a<2]`	Select elements from `a` less than 2

Fancy Indexing

`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]`	Select elements `(1,0)`, `(0,1)`, `(1,2)` and `(0,0)`
`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`	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)`	Concatenate arrays
`>>> np.vstack((a,b))`	Stack arrays vertically (row-wise)
`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)
`>>> np.hstack((e,f))`	Stack arrays horizontally (column-wise)
`>>> np.column_stack((a,d))`	Create stacked column-wise arrays
`>>> np.c_[a,d]`	Create stacked column-wise arrays

Splitting Arrays

`>>> np.hsplit(a,3)`	Split the array horizontally at the 3rd index
`>>> np.vsplit(c,2)`	Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science Interactively

IMPORTANT METHODS IN NUMPY PACKAGE

- np.array()
- np.std()
- np.subtract()
- np.add()
- np.divide()
- np.multiply()
- np.exp()
- np.sqrt()
- np.sin()
- np.cos()
- np.log()

- np.copy()
- np.zeros()
- np.ones()
- np.arange()
- np.linspace()
- np.full()
- np.eye()
- np.empty()
- np.save()
- np.load()
- np.loadtxt()

- np.genfromtxt()
- np.savetxt()
- np.append()
- np.insert()
- np.delete()
- np.concatenate()
- np.vstack()
- np.hstack()
- np.hsplit()
- np.vsplit()
- np.random.rand()

IMPORTANT METHODS IN SEABORN PACKAGE

- relplot()
- scatterplot()
- lineplot()
- catplot()
- stripplot()
- swarmplot()
- boxplot()
- violinplot()
- boxenplot()
- pointplot()
- barplot()

- countplot()
- distplot()
- kdeplot()
- rugplot()
- lmplot()
- regplot()
- residplot()
- heatmap()
- clustermap()
- FacetGrid()
- pairplot()

- PairGrid()
- jointplot()
- JointGrid()
- set_palette()
- color_palette()
- load_dataset()
- despine()
- desaturate()
- saturate()
- axes_style()
- set_style()

IMPORTANT METHODS IN MATPLOTLIB PACKAGE

- acorr()
- autoscale()
- axis()
- bar()
- barh()
- boxplot()
- clabel()
- colorbar()
- draw()
- eventplot()
- fill()
- grid()
- hist()
- imread()
- imsave()
- imshow()
- isinteractive()
- legend()
- margins()
- pie()
- plot()
- scatter()
- stackplot()
- streamplot()
- subplot()
- table()
- text()
- title()
- violinplot()
- xlabel()
- xscale()
- ylabel()
- yscale()

Matplotlib Cheat Sheet

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))
```

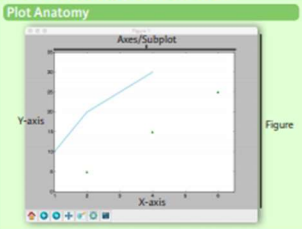
Axes

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 o

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

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

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

DataCamp
Learn Python for Data Science Interactively