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
>>> x-2
 3
>>> x*2
 10
>>> x**2
 25
>>> x%2
 1
>>> x/float(2)
 2.5
```

Sum of two variables

Subtraction of two variables

Multiplication of two variables

Exponentiation of a variable

Remainder of a variable

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

Select item at index 1

Select 3rd last item

Slice

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

Select items at index 1 and 2

Select items after index 0

Select items before index 3

Copy my_list

Subset Lists of Lists

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

my_list[list][itemOfList]

List Operations

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

List Methods

```
>>> my_list.index(a)
>>> my_list.count(a)
>>> my_list.append('!')
>>> my_list.remove('!')
>>> del(my_list[0:1])
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> 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 an item

Remove an item

Insert an item

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()
>>> 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

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
>>> 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

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.

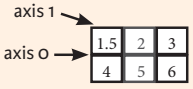
Use the following import convention:

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

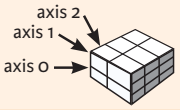
NumPy Arrays

1D array

2D array



3D array



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
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> 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
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> 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 0 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)      Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                  Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                        Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                  Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))            Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                        Create stacked column-wise arrays
```

Splitting Arrays

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

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))
```

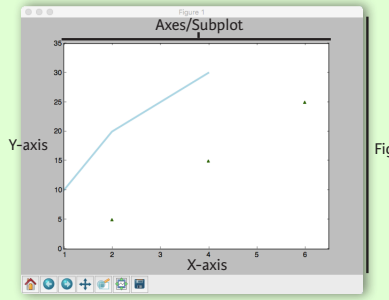
Axes

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',          Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                    No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

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

```
>>> fig, ax = plt.subplots()
>>> 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 a 2D field of arrows
```

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,                      Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> 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
```

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")`	Subplot grid for plotting conditional relationships
`>>> g = g.map(plt.hist,"age")`	
`>>> 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)`	Subplot grid for plotting pairwise relationships
`>>> h = h.map(plt.scatter)`	
`>>> sns.pairplot(iris)`	Plot pairwise bivariate distributions
`>>> i = sns.JointGrid(x="x", y="y", data=data)`	Grid for bivariate plot with marginal univariate plots
`>>> i = i.plot(sns.regplot, sns.distplot)`	
`>>> 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")`	Add plot title
`>>> plt.ylabel("Survived")`	Adjust the label of the y-axis
`>>> plt.xlabel("Sex")`	Adjust the label of the x-axis
`>>> plt.ylim(0,100)`	Adjust the limits of the y-axis
`>>> plt.xlim(0,10)`	Adjust the limits of the x-axis
`>>> plt.setp(ax,yticks=[0,5])`	Adjust a plot property
`>>> plt.tight_layout()`	Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

`>>> plt.show()`	Show the plot
`>>> plt.savefig("foo.png")`	Save the plot as a figure
`>>> plt.savefig("foo.png", transparent=True)`	Save transparent figure

Close & Clear

Also see Matplotlib

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