

# 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
```

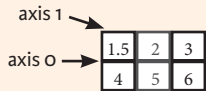


### NumPy Arrays

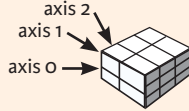
#### 1D array



#### 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))
>>> 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('aFray.npz', a, b)
>>> np.load('my_array.npy')
```

#### Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

### Data Types

```
>>> np.int64
>>> np.float32
>>> np.complex
>>> np.bool
>>> np.object
>>> np.string_
>>> np.unicode_
```

Signed 64-bit integer types  
Standard double-precision floating point  
Complex numbers represented by 128 floats  
Boolean type storing TRUE and FALSE values  
Python object type  
Fixed-length string type  
Fixed-length unicode type

### Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions  
Length of array  
Number of array dimensions  
Number of array elements  
Data type of array elements  
Name of data type  
Convert an array to a different type

### Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

### Array Mathematics

#### Arithmetic Operations

```
>>> g = a - b
array([[ -0.5,  0. ,  0. ],
       [ -3. , -3. , -3. ]])
>>> np.subtract(a,b)
>>> b + a
array([[ 2.5,  4. ,  6. ],
       [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)
>>> a / b
array([[ 0.66666667,  1. ,  1. ],
       [ 0.25 ,  0.4 ,  0.5 ]])
>>> np.divide(a,b)
>>> a * b
array([[ 1.5,  4. ,  9. ],
       [ 4. , 10. , 18. ]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
array([[ 7. ,  7. ],
       [ 7. ,  7.]])
```

Subtraction  
Subtraction  
Addition  
Addition  
Division  
Division  
Division  
Multiplication  
Multiplication  
Exponentiation  
Square root  
Print sines of an array  
Element-wise cosine  
Element-wise natural logarithm  
Dot product

#### Comparison

```
>>> a == b
array([[False,  True,  True],
       [False, False, False]], dtype=bool)
>>> a < 2
array([ True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison  
Element-wise comparison  
Array-wise comparison

#### Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> 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]
3
>>> b[1,2]
6.0
```

Select the element at the 2nd index  
Select the element at row 1 column 2 (equivalent to b[1][2])

#### Slicing

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

Select items at index 0 and 1  
Select items at rows 0 and 1 in column 1  
Select all items at row 0 (equivalent to b[0:1, :])  
Same as [1, :, :]  
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])
>>> b[[1, 0, 1, 0]][:, [0,1,2,0]]
array([[ 4. ,  5. ,  6. ,  4. ],
       [ 1.5,  2. ,  3. ,  1.5],
       [ 4. ,  5. ,  6. ,  4. ],
       [ 1.5,  2. ,  3. ,  1.5]])
```

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

### Array Manipulation

#### Transposing Array

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

Permute array dimensions  
Permute array dimensions

#### Changing Array Shape

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

Flatten the array  
Reshape, but don't change data

#### Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a, [1])
```

Return a new array with shape (2,6)  
Append items to an array  
Insert items in an array  
Delete items from an array

#### Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
array([[ 1. ,  2. ,  3. ],
       [ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
array([[ 7.,  7.,  1.,  0.],
       [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
array([[ 1, 10],
       [ 2, 15],
       [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays  
Stack arrays vertically (row-wise)  
Stack arrays vertically (row-wise)  
Stack arrays horizontally (column-wise)  
Create stacked column-wise arrays  
Create stacked column-wise arrays

#### Splitting Arrays

```
>>> np.hsplit(a,3)
[array([1]),array([2]),array([3])]
>>> np.vsplit(c,2)
[array([[ 1.5,  2. ,  1. ],
       [ 4. ,  5. ,  6. ]]),
 array([[ 3.,  2.,  3.],
       [ 4. ,  5. ,  6.]])]
```

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

DataCamp

Learn Python for Data Science Interactively

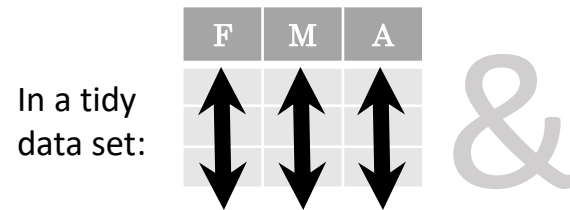


# Data Wrangling

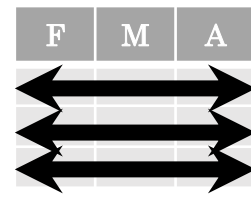
with pandas  
Cheat Sheet

<http://pandas.pydata.org>

## Tidy Data – A foundation for wrangling in pandas

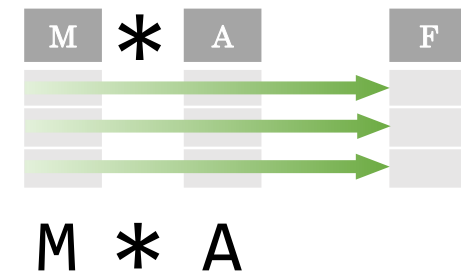


Each **variable** is saved in its own **column**



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



## Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
    {"a" : [4 ,5, 6],
     "b" : [7, 8, 9],
     "c" : [10, 11, 12]},
    index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
    {"a" : [4 ,5, 6],
     "b" : [7, 8, 9],
     "c" : [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
        names=['n','v']))
```

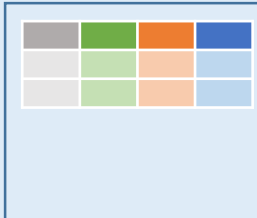
Create DataFrame with a MultiIndex

## Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
     .rename(columns={
         'variable' : 'var',
         'value' : 'val'})
     .query('val >= 200'))
```

## Reshaping Data – Change the layout of a data set



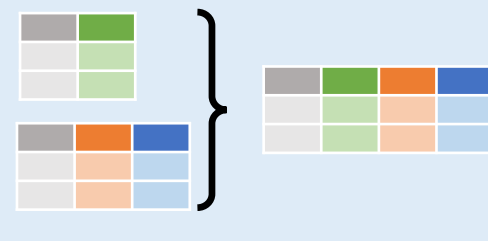
**pd.melt(df)**  
Gather columns into rows.



**df.pivot(columns='var', values='val')**  
Spread rows into columns.



**pd.concat([df1, df2])**  
Append rows of DataFrames



**pd.concat([df1, df2], axis=1)**  
Append columns of DataFrames

```
df.sort_values('mpg')
    Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
    Order rows by values of a column (high to low).

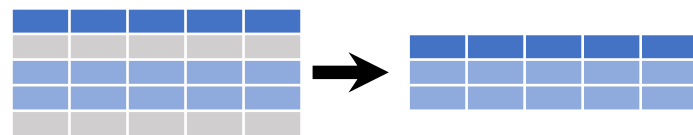
df.rename(columns = {'y':'year'})
    Rename the columns of a DataFrame

df.sort_index()
    Sort the index of a DataFrame

df.reset_index()
    Reset index of DataFrame to row numbers, moving
    index to columns.

df.drop(columns=['Length', 'Height'])
    Drop columns from DataFrame
```

## Subset Observations (Rows)



```
df[df.Length > 7]
    Extract rows that meet logical
    criteria.

df.drop_duplicates()
    Remove duplicate rows (only
    considers columns).

df.head(n)
    Select first n rows.

df.tail(n)
    Select last n rows.
```

```
df.sample(frac=0.5)
    Randomly select fraction of rows.

df.sample(n=10)
    Randomly select n rows.

df.iloc[10:20]
    Select rows by position.

df.nlargest(n, 'value')
    Select and order top n entries.

df.nsmallest(n, 'value')
    Select and order bottom n entries.
```

## Subset Variables (Columns)



```
df[['width', 'length', 'species']]
    Select multiple columns with specific names.

df['width'] or df.width
    Select single column with specific name.

df.filter(regex='regex')
    Select columns whose name matches regular expression regex.
```

### regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length\$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]\$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species\$).*'	Matches strings except the string 'Species'

```
df.loc[:, 'x2': 'x4']
    Select all columns between x2 and x4 (inclusive).

df.iloc[:, [1, 2, 5]]
    Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a', 'c']]
    Select rows meeting logical condition, and only the specific columns.
```

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(values)	Group membership
==	Equals	pd.isnull(obj)	Is NaN
<=	Less than or equals	pd.notnull(obj)	Is not NaN
>=	Greater than or equals	&,  , ~, ^, df.any(), df.all()	Logical and, or, not, xor, any, all

## Summarize Data

**df['w'].value\_counts()**

Count number of rows with each unique value of variable

**len(df)**

# of rows in DataFrame.

**df['w'].nunique()**

# of distinct values in a column.

**df.describe()**

Basic descriptive statistics for each column (or GroupBy)



pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

**sum()**

Sum values of each object.

**count()**

Count non-NA/null values of each object.

**median()**

Median value of each object.

**quantile([0.25,0.75])**

Quantiles of each object.

**apply(function)**

Apply function to each object.

**min()**

Minimum value in each object.

**max()**

Maximum value in each object.

**mean()**

Mean value of each object.

**var()**

Variance of each object.

**std()**

Standard deviation of each object.

## Group Data



**df.groupby(by="col")**

Return a GroupBy object, grouped by values in column named "col".

**df.groupby(level="ind")**

Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

**size()**

Size of each group.

**agg(function)**

Aggregate group using function.

## Windows

**df.expanding()**

Return an Expanding object allowing summary functions to be applied cumulatively.

**df.rolling(n)**

Return a Rolling object allowing summary functions to be applied to windows of length n.

## Handling Missing Data

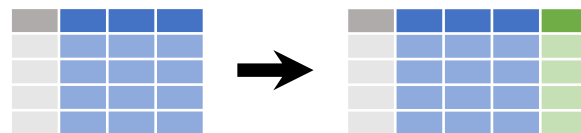
**df.dropna()**

Drop rows with any column having NA/null data.

**df.fillna(value)**

Replace all NA/null data with value.

## Make New Columns



**df.assign(Area=lambda df: df.Length\*df.Height)**

Compute and append one or more new columns.

**df['Volume'] = df.Length\*df.Height\*df.Depth**

Add single column.

**pd.qcut(df.col, n, labels=False)**

Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

**max(axis=1)**

Element-wise max.

**min(axis=1)**

Element-wise min.

**clip(lower=-10,upper=10) abs()**

Trim values at input thresholds Absolute value.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

**shift(1)**

Copy with values shifted by 1.

**rank(method='dense')**

Ranks with no gaps.

**rank(method='min')**

Ranks. Ties get min rank.

**rank(pct=True)**

Ranks rescaled to interval [0, 1].

**rank(method='first')**

Ranks. Ties go to first value.

**shift(-1)**

Copy with values lagged by 1.

**cumsum()**

Cumulative sum.

**cummax()**

Cumulative max.

**cummin()**

Cumulative min.

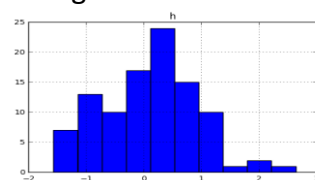
**cumprod()**

Cumulative product.

## Plotting

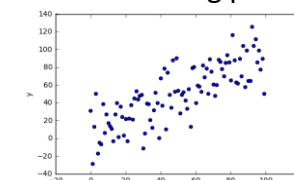
**df.plot.hist()**

Histogram for each column



**df.plot.scatter(x='w',y='h')**

Scatter chart using pairs of points



## Combine Data Sets

**adf**

x1	x2
A	1
B	2
C	3



**bdf**

x1	x3
A	T
B	F
D	T



Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

**pd.merge(adf, bdf, how='left', on='x1')**

Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

**pd.merge(adf, bdf, how='right', on='x1')**

Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

**pd.merge(adf, bdf, how='inner', on='x1')**

Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

**pd.merge(adf, bdf, how='outer', on='x1')**

Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

**adf[adf.x1.isin(bdf.x1)]**

All rows in adf that have a match in bdf.

x1	x2
C	3

**adf[~adf.x1.isin(bdf.x1)]**

All rows in adf that do not have a match in bdf.

**ydf**

x1	x2
A	1
B	2
C	3



**zdf**

x1	x2
B	2
C	3
D	4



Set-like Operations

x1	x2
B	2
C	3

**pd.merge(ydf, zdf)**

Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

**pd.merge(ydf, zdf, how='outer')**

Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

**pd.merge(ydf, zdf, how='outer', indicator=True)**

**.query('\_merge == "left\_only")**

**.drop(columns=['\_merge'])**

Rows that appear in ydf but not zdf (Setdiff).



# Python For Data Science Cheat Sheet

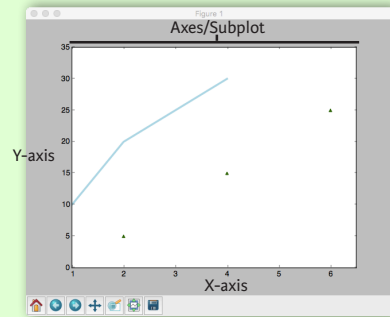
## Matplotlib

Learn Python Interactively at [www.DataCamp.com](http://www.DataCamp.com)



## 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='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()
```

## Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.



## 1 Prepare The Data

Also see Lists & NumPy

### 1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

### 2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

## 2 Create Plot

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

### Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

### Axes

All plotting is done with respect to an Axes. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

## 4 Customize Plot

### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                  cmap='seismic')
```

### Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

### Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

### Text & Annotations

```
>>> ax.text(1,
          -2.1,
          'Example Graph',
          style='italic')
>>> ax.annotate("Sine",
              xy=(8, 0),
              xycoords='data',
              xytext=(10.5, 0),
              textcoords='data',
              arrowprops=dict(arrowstyle="->",
                              connectionstyle="arc3"),)
```

### Mathtext

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

### Limits, Legends & Layouts

#### Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

#### Legends

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

#### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
               ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                  direction='inout',
                  length=10)
```

#### Subplot Spacing

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

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

#### Axis Spines

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

Add padding to a plot  
Set the aspect ratio of the plot to 1  
Set limits for x-and y-axis  
Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible  
Move the bottom axis line outward

## 3 Plotting Routines

### 1D Data

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

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

### Vector Fields

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

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

### Data Distributions

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

Plot a histogram  
Make a box and whisker plot  
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)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2] = ax.clabel(CS)
```

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

## 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

## Scikit-Learn

Learn Python for data science [Interactively](https://www.datacamp.com) 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', 'F', 'M', 'F', 'F', 'F'])
>>> X[X < 0.7] = 0
```

#### Training And Test Data

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

### Preprocessing The Data

#### Standardization

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

#### Normalization

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

#### Binarization

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

#### 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)
```

### 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)
```

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

### 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_true, y_pred)
```

##### R<sup>2</sup> 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),
            "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_)
```



# scikit-learn algorithm cheat-sheet

