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

```
1  2  3
```

2D array

```
axis 1
axis 0 →  1.5  2   3
          4    5   6
```

3D array

```
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
  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
>>> np.median(a)     Median
>>> np.corrcoef(a)   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

Subsetting

```
>>> a[2]                                 Select the element at the 2nd index
  3
>>> b[1,2]                               Select the element at row 1 column 2
  6.0                                    (equivalent to b[1][2])
```

Slicing

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

Boolean Indexing

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

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]         Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4., 2., 6., 1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]         Select a subset of the matrix's rows
  array([[ 4.,5., 6., 4.],               and columns
         [ 1.5,2., 3., 1.5],
         [ 4., 5., 6., 4.],
         [ 1.5,2., 3., 1.5]])
```

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
  [array([1]),array([2]),array([3])]     index
>>> 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.]]])]
```

Data Wrangling
with pandas
Cheat Sheet
http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

&

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.
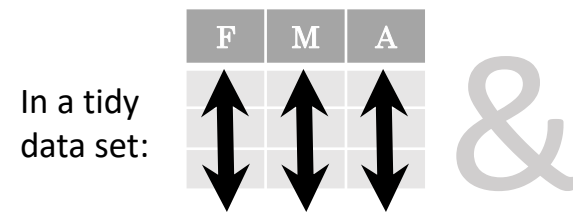
Each **variable** is saved in its own **column**
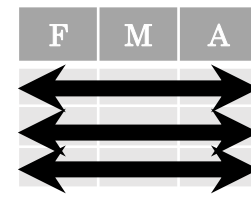
Each **observation** is saved in its own **row**

M * A

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')
     )
```
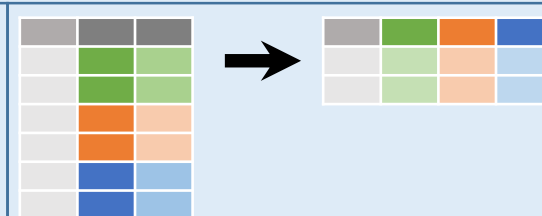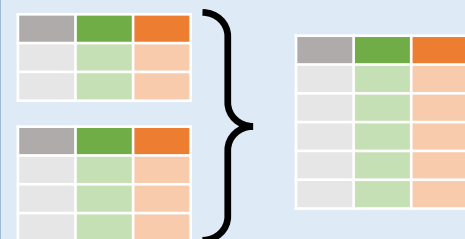
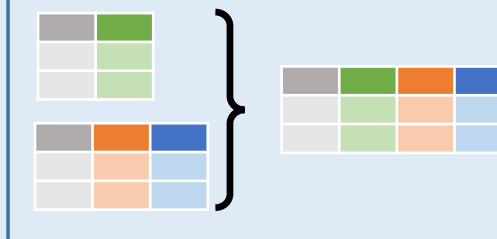## Reshaping Data – Change the layout of a data set
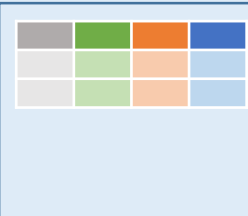
pd.melt(df)
Gather columns into rows.

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

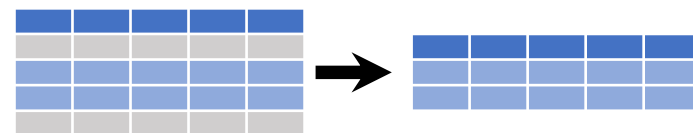**df.sort_index()**
Sort the index of a DataFrame

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.

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

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 .

Summarize Data

`df['w'].value_counts()`
  Count number of rows with each unique value of variable
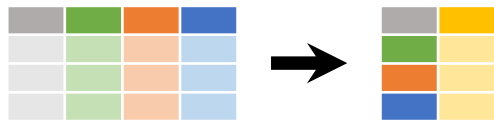`len(df)`
  # of rows in DataFrame.
`df['w'].nunique()`
  # of distinct values in a column.
`df.describe()`
  Basic descriptive statistics for each column (or GroupBy)

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.
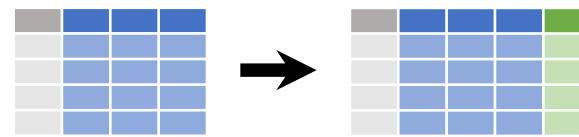
Handling Missing Data

`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.
`clip(lower=-10,upper=10)`
  Trim values at input thresholds

`min(axis=1)`
  Element-wise min.
`abs()`
  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.
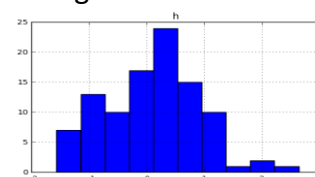`cummax()`
  Cumulative max.
`cummin()`
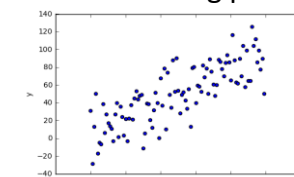 Cumulative min.
`cumprod()`
  Cumulative product.

Plotting

`df.plot.hist()`
Histogram for each column

`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points

Combine Data Sets

Standard Joins

`pd.merge(adf, bdf,`
`        how='left', on='x1')`
  Join matching rows from bdf to adf.

`pd.merge(adf, bdf,`
`        how='right', on='x1')`
  Join matching rows from adf to bdf.

`pd.merge(adf, bdf,`
`        how='inner', on='x1')`
  Join data. Retain only rows in both sets.

`pd.merge(adf, bdf,`
`        how='outer', on='x1')`
  Join data. Retain all values, all rows.

Filtering Joins

`adf[adf.x1.isin(bdf.x1)]`
  All rows in adf that have a match in bdf.

`adf[~adf.x1.isin(bdf.x1)]`
  All rows in adf that do not have a match in bdf.

Set-like Operations

`pd.merge(ydf, zdf)`
  Rows that appear in both ydf and zdf (Intersection).

`pd.merge(ydf, zdf, how='outer')`
  Rows that appear in either or both ydf and zdf (Union).

`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

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)
```
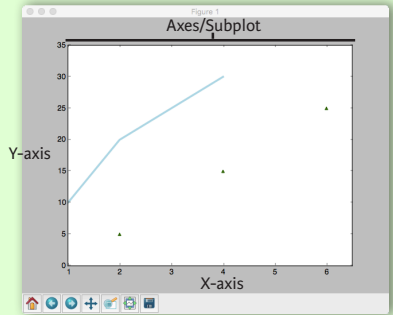
Plot Anatomy & Workflow

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

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

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
```

```
>>> 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*
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)
```

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')
```

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

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)
```

R² Score
```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

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

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

V-measure
```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation
```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "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_)
```

scikit-learn algorithm cheat-sheet

classification

kernel approximation

SVC Ensemble Classifiers

KNeighbors Classifier

SGD Classifier

Naive Bayes

Linear SVC

Text Data

<100K samples

NOT WORKING

YES / NO

START

get more data

>50 samples

regression

SGD Regressor

ElasticNet Lasso

SVR(kernel='rbf') EnsembleRegressors

<100K samples

few features should be important

RidgeRegression SVR (kernel='linear')

predicting a category

do you have labeled data

predicting a quantity

clustering

Spectral Clustering GMM

KMeans

number of categories known

<10K samples

MiniBatch KMeans

MeanShift VBGMM

<10K samples

just looking

Randomized PCA

Isomap Spectral Embedding

LLE

<10K samples

kernel approximation

dimensionality reduction

tough luck

predicting structure