

Cheat Sheet of Machine Learning and Python (and Math) Cheat Sheets



Robbie Allen

Follow



Jun 1, 2017 · 4 min read

*If you like this article, check out another by Robbie:
[My Curated List of AI and Machine Learning Resources](#)*



There are many facets to Machine Learning. As I started brushing up on the

subject, I came across various “cheat sheets” that compactly listed all the key points I needed to know for a given topic. Eventually, I compiled over 20 Machine Learning-related cheat sheets. Some I reference frequently and thought others may benefit from them too. This post contains 27 of the better cheat sheets I’ve found on the web. Let me know if I’m missing any you like.

Given how rapidly the Machine Learning space is evolving, I imagine these will go out of date quickly, but at least as of June 1, 2017, they are pretty current.

If you want all of the cheat sheets without having to download them individually like I did, [I created a zip file containing all 27](#). Enjoy!

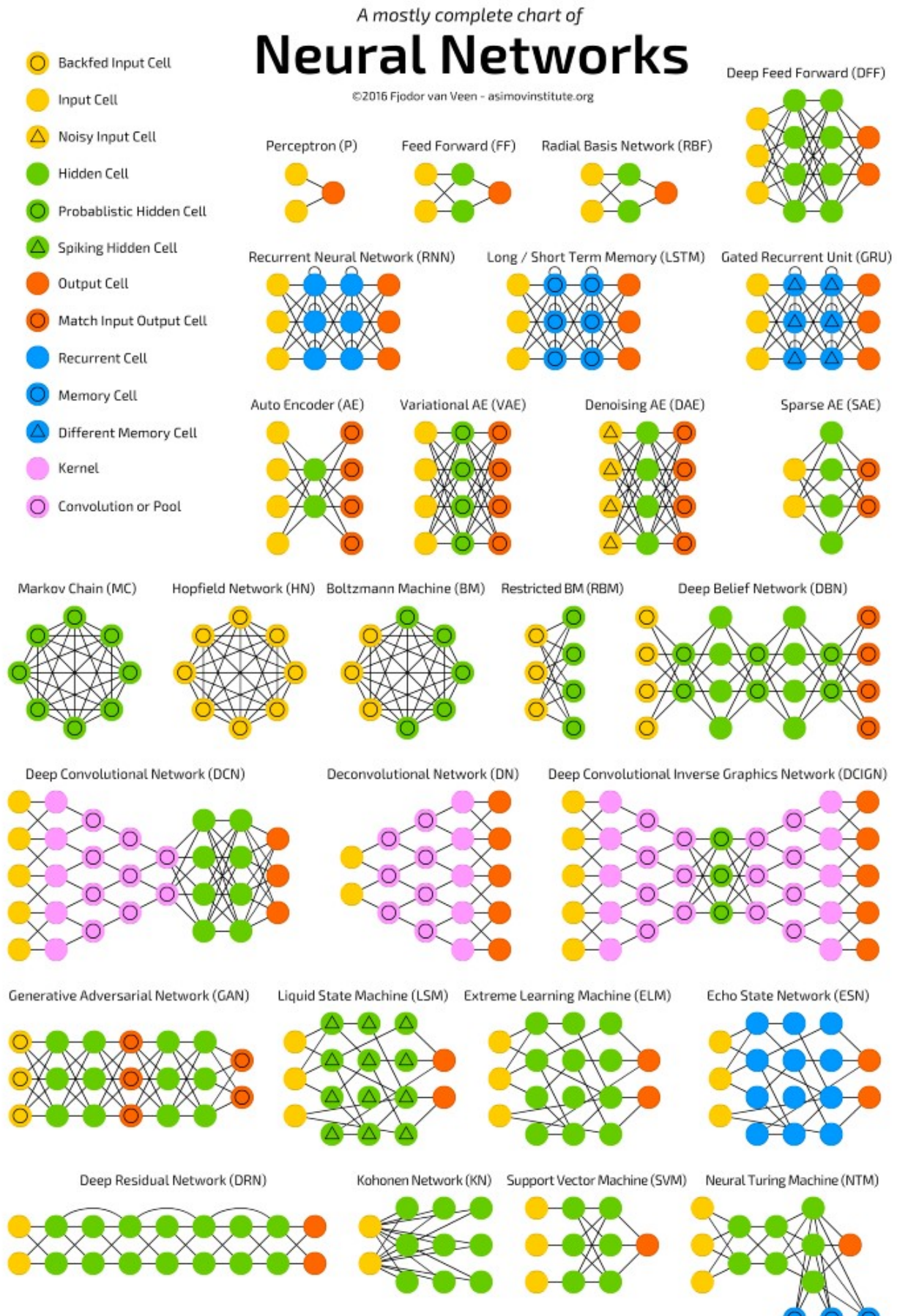
If you like this post, give it a ♥ below.

Machine Learning

There are a handful of helpful flowcharts and tables of Machine Learning algorithms. I’ve included only the most comprehensive ones I’ve found.

Neural Network Architectures

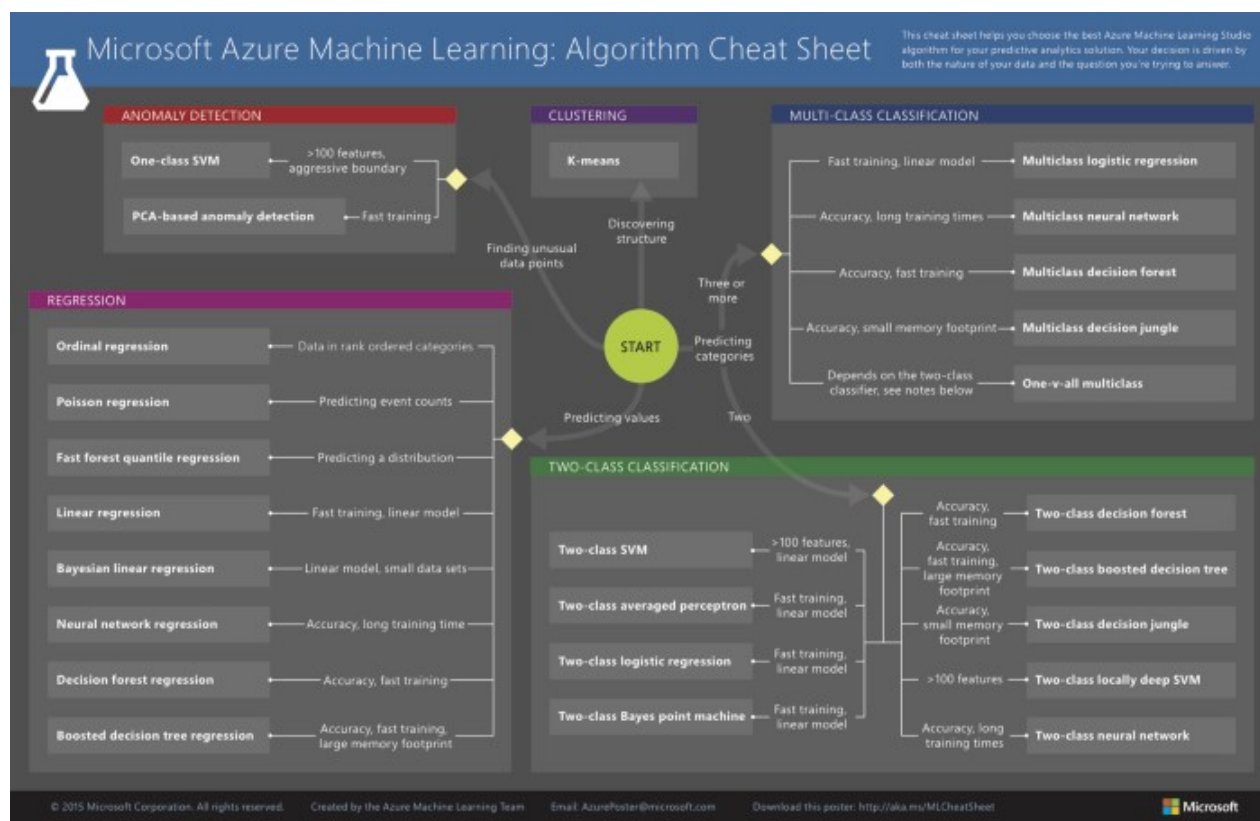
Source: <http://www.asimovinstitute.org/neural-network-zoo/>





Microsoft Azure Algorithm Flowchar

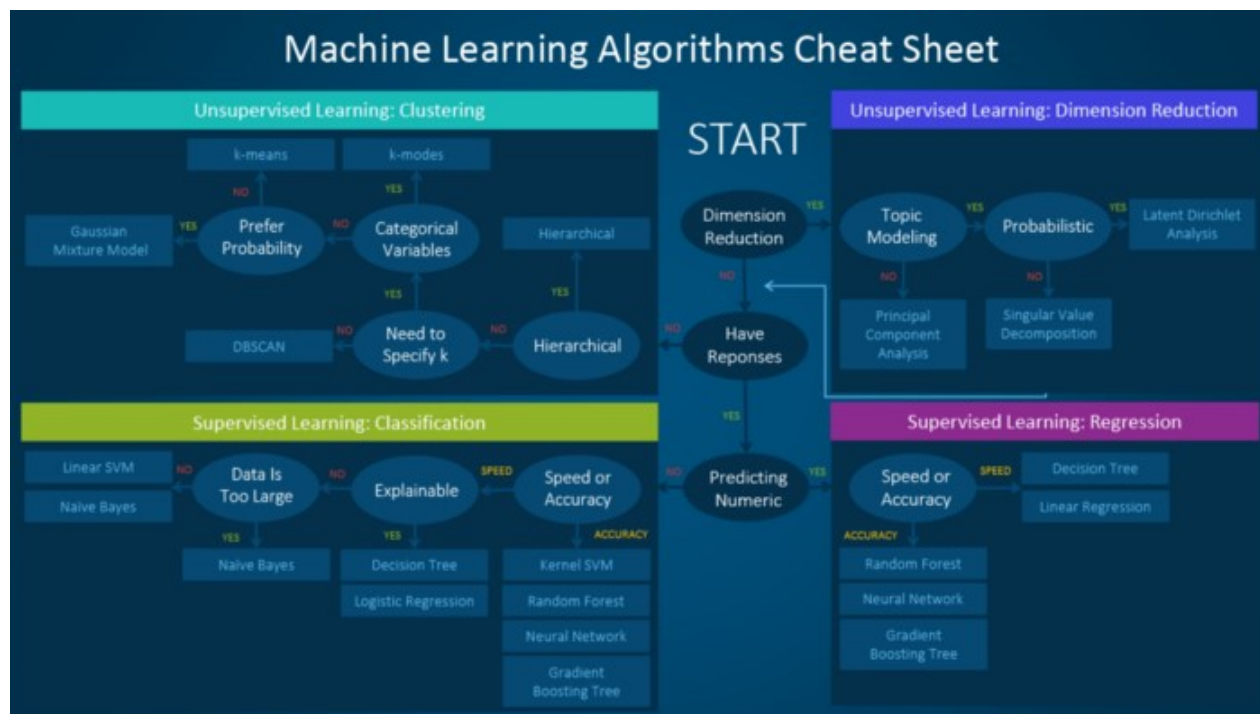
Source: <https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-cheat-sheet>



Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio

SAS Algorithm Flowchart

Source: <http://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/>



SAS: Which machine learning algorithm should I use?

Algorithm Summary

Source: <http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/>



A Tour of Machine Learning Algorithms

Source: <http://thinkbigdata.in/best-known-machine-learning-algorithms-infographic/>

the world of machine learning algorithms – a summary

regression

Ordinary Least Squares Regression (OLSR)
Linear Regression
Logistic Regression
Stepwise Regression
Multivariate Adaptive Regression Splines (MARS)
Locally Estimated Scatterplot Smoothing (LOESS)
Jackknife Regression

regularization

Ridge Regression
Least Absolute Shrinkage and Selection Operator (LASSO)
Elastic Net
Least-Angle Regression (LARS)

instance based

also called **cake-based**, **memory-based**

k-Nearest Neighbour (kNN)
Learning Vector Quantization (LVQ)
Self-Organizing Map (SOM)
Locally Weighted Learning (LWL)

dimensionality reduction

Principal Component Analysis (PCA)
Principal Component Regression (PCR)
Partial Least Squares Regression (PLSR)
Sammon Mapping
Multidimensional Scaling (MDS)
Projection Pursuit
Discriminant Analysis (LDA, MDA, QDA, FDA)

deep learning

Deep Boltzmann Machine (DBM)
Deep Belief Networks (DBN)
Convolutional Neural Network (CNN)
Stacked Auto-Encoders

associated rule

Apriori
Eclat
FP-Growth

ensemble

Logit Boost (Boosting)
Bootstrapped Aggregation (Bagging)
AdaBoost
Stacked Generalization (blending)
Gradient Boosting Machines (GBM)
Gradient Boosted Regression Trees (GBRT)
Random Forest

think big data

bayesian

Naive Bayes
Gaussian Naive Bayes
Multinomial Naive Bayes
Averaged One-Dependence Estimators (AOOE)
Bayesian Belief Network (BBN)
Bayesian Network (BN)
Hidden Markov Models
Conditional random fields (CRFs)

decision tree

Classification and Regression Tree (CART)
Iterative Dichotomiser 3 (ID3)
C4.5 and C5.0 (different versions of a powerful approach)
Chi-squared Automatic Interaction Detection (CHAID)
Decision Stump
M5
Random Forests
Conditional Decision Trees

clustering

Single-linkage clustering
k-Means
k-Medians
Expectation Maximisation (EM)
Hierarchical Clustering
Fuzzy clustering
DBSCAN
OPTICS algorithm
Non Negative Matrix Factorization
Latent Dirichlet allocation (LDA)

neural networks








Self Organizing Map
Perceptron
Back-Propagation
Hopfield Network
Radial Basis Function Network (RBFN)
Backpropagation
Autoencoders
Hopfield networks
Boltzmann machines
Restricted Boltzmann Machines
Spiking Neural Networks
Learning Vector quantization (LVQ)

...and others

Support Vector Machines (SVM)
Evolutionary Algorithms
Inductive Logic Programming (ILP)
Reinforcement Learning (Q-Learning, Temporal Difference, State-Action-Reward-State-Action (SARSA))
ANOVA
Information Fuzzy Network (IFN)
Page Rank
Conditional Random Fields (CRF)

Algorithm Pro/Con

Source: <https://blog.dataiku.com/machine-learning-explained-algorithms-are-your-friend>

<div>  dataiku </div> <div>TOP PREDICTION ALGORITHMS</div>				
TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	 <div>Linear regression</div>	The “best fit” line through all data points. Predictions are numerical.	Easy to understand – you clearly see what the biggest drivers of the model are.	<div>X</div> Sometimes too simple to capture complex relationships between variables. <div>X</div> Tendency for the model to “overfit”.
	 <div>Logistic regression</div>	The adaptation of linear regression to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	<div>X</div> Sometimes too simple to capture complex relationships between variables. <div>X</div> Tendency for the model to “overfit”.
Tree-based	 <div>Decision tree</div>	A graph that uses a branching method to match all possible outcomes of a decision.	Easy to understand and implement.	<div>X</div> Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
	 <div>Random Forest</div>	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but by combining them we get better overall performance.	A sort of “wisdom of the crowd”. Tends to result in very high quality models. Fast to train.	<div>X</div> Can be slow to output predictions relative to other algorithms. <div>X</div> Not easy to understand predictions.
	 <div>Gradient Boosting</div>	Uses even weaker decision trees, that are increasingly focused on “hard” examples.	High-performing.	<div>X</div> A small change in the feature set or training set can create radical changes in the model. <div>X</div> Not easy to understand predictions.
Neural networks	 <div>Neural networks</div>	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several layers of neural networks put one after the other.	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	<div>X</div> Very, very slow to train, because they have so many layers. Require a lot of power. <div>X</div> Almost impossible to understand predictions.

Python

Unsurprisingly, there are a lot of online resources available for Python. For this section, I've only included the best cheat sheets I've come across.

Algorithms

Source: <https://www.analyticsvidhya.com/blog/2015/09/full-cheatsheet-machine-learning-algorithms/>

CHEATSHEET

Machine Learning Algorithms

(Python and R Codes)

Types

Supervised Learning


- Decision Tree
- Random Forest
- kNN
- Logistic Regression

Unsupervised Learning

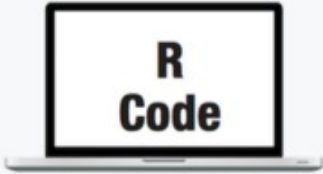
- Apriori algorithm
- k-means
- Hierarchical Clustering

Reinforcement Learning

- Markov Decision Process
- Q Learning



Python Code



R Code

Linear Regression

```

#Import Library
#Import other necessary libraries like pandas,
#numpy...
from sklearn import linear_model
#Load Train and Test datasets
#Identify feature and response variable(s) and
#values must be numeric and numpy arrays
x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_datasets
x_test=input_variables_values_test_datasets
#Create linear regression object
linear = linear_model.LinearRegression()
#Train the model using the training sets and
#check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)
#Equation coefficient and Intercept
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)
#Predict Output
predicted= linear.predict(x_test)
                    
```

```

#Load Train and Test datasets
#Identify feature and response variable(s) and
#values must be numeric and numpy arrays
x_train <- input_variables_values_training_datasets
y_train <- target_variables_values_training_datasets
x_test <- input_variables_values_test_datasets
x <- cbind(x_train,y_train)
#Train the model using the training sets and
#check score
linear <- lm(y_train ~ ., data = x)
summary(linear)
#Predict Output
predicted= predict(linear,x_test)
                    
```

Python Basics

Source: <http://datasciencefree.com/python.pdf>

Python Cheat Sheet

JUST THE BASICS

Created By: Andrew Cunniff and Sam Glick

GENERAL

- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

HELP

Help Home Page	<code>help()</code>
Function Help	<code>help(object.replace)</code>
Module Help	<code>help(sys)</code>

MODULE (AKA LIBRARY)

Python module is simply a '.py' file

List Module Contents	<code>dir(module)</code>
Load Module	<code>import module</code>
Call Function from Module	<code>module.function()</code>

* Import statement creates a new namespace and executes all the statements in the associated .py file within that namespace. If you want to load the module's content into current namespace, use `from module import *`

SCALAR TYPES

Check data type : `type(variable)`

SIX COMMONLY USED DATA TYPES

- int/long** - Large int automatically converts to long
- float** - 64 bits, there is no 'double' type
- bool** - True or False
- str** - ASCII value in Python 2x and Unicode in Python 3
 - String can be in single/double/triple quotes
 - String is a sequence of characters, thus can be treated like other sequences
 - Special character can be done via \ or preface with r

String formatting can be done in a number of ways

```
template = "8.2E to haha %d"
str1 = template % (4.88, "haha", 2)
```

SCALAR TYPES

* str(), bool(), int() and float() are also explicit type cast functions.

5. NoneType(None)

- Python 'null' value (ONLY one instance of None object exists)

- None is not a reserved keyword but rather a unique instance of 'NoneType'
- None is common default value for optional function arguments:

```
def func1(a, b, c = None):
    if variable is None:
```

6. datetime

- built-in python 'datetime' module provides 'datetime', 'date', 'time' types.

- 'datetime' combines information stored in 'date' and 'time'

Create datetime from String	<code>dt1 = datetime.strptime("2009-03-1", "%Y-%m-%d")</code>
Get 'date' object	<code>dt1.date()</code>
Get 'time' object	<code>dt1.time()</code>
Format datetime to String	<code>dt1.strftime("%m/%d/%Y %H:%M")</code>
Change Field Value	<code>dt2 = dt1.replace(minute = 0, second = 30)</code>
Get Difference	<code>diff = dt1 - dt2</code> # diff is a 'datetime.timedelta' object

Note : Most objects in Python are mutable except for 'strings' and 'tuples'

DATA STRUCTURES

Note : All non-Get function call (i.e. list(), sort()) examples below are in-place (without creating a new object) operations unless noted otherwise.

TUPLE

One dimensional, fixed-length, immutable sequence of Python objects of ANY type.

DATA STRUCTURES

Create Tuple	<code>tuple1 = 4, 5, 6 or tuple1 = (4, 5, 6)</code>
Create Nested Tuple	<code>tuple1 = (4, 5, 6), (7, 8)</code>
Convert Sequence or Iterator to Tuple	<code>tuple1 = (1, 0, 2)</code>
Concatenate Tuples	<code>tuple1 + tuple2</code>
Unpack Tuple	<code>a, b, c = tuple1</code>
Swap variables	<code>a, b = b, a</code>

LIST

One dimensional, variable length, mutable (i.e. contents can be modified) sequence of Python objects of ANY type.

Create List	<code>list1 = [1, 'a', 3] or list1 = list(tuple1)</code>
Concatenate List*	<code>list1 + list2 or list1.extend(list2)</code>
Append to End of List	<code>list1.append('a')</code>
Insert in Specific Position	<code>list1.insert(position, 'a')</code>
Inverse of Insert	<code>valueAtIdx = list1.pop(position)</code>
Remove First Value from List	<code>list1.remove('a')</code>
Check Membership	<code>3 in list1 => True</code>
Sort List	<code>list1.sort()</code>
Sort with User-Supplied Function	<code>list1.sort(key = len)</code> # sort by length

- List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, extend() is preferable.
- Insert is computationally expensive compared with append.
- Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

Built-in 'bisect' module

- Implements binary search and insertion into a sorted list
- 'bisect.bisect' finds the location, where 'bisect.insert' actually inserts into that location.

* WARNING : bisect module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them in an unsorted list will succeed without error but may lead to incorrect results.

SLICING FOR SEQUENCE TYPES[†]

† Sequence types include 'str', 'array', 'tuple', 'list', etc.

Notation	<code>list1[start:end:step]</code>
	<code>list1[start:end:step]</code> (if step is used)

NOTE :

- 'start' index is included, but 'stop' index is NOT.
- start/stop can be omitted in which they default to the start/end.

* Application of 'step' :

Take every other element	<code>list1[::2]</code>
Reverse a string	<code>str[::-1]</code>

DICT (HASH MAP)

Create Dict	<code>dict1 = {'key1': 'value1', 2: 1.2, 2.1}</code>
Create Dict from Sequence	<code>dict(zip(keyList, valueList))</code>
Get/Set/Insert Element	<code>dict1['key1'] = 'newValue'</code>
Get with Default Value	<code>dict1.get('key1', defaultValue)</code>
Check if Key Exists	<code>'key1' in dict1</code>
Delete Element	<code>del dict1['key1']</code>
Get Key List	<code>dict1.keys()</code>
Get Value List	<code>dict1.values()</code>
Update Values	<code>dict1.update(dict2)</code> # dict values are replaced by dict2

- 'KeyError' exception if the key does not exist.
- 'get()' by default (aka no 'defaultValue') will return 'None' if the key does not exist.
- Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

Valid dict key types

- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is 'hashability', check whether an object is hashable with the `hash('this is string')`, `hash((1, 2))` - this would fail.

SET

- A set is an unordered collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set	<code>set1 = {3, 6, 3} or set1 = {3, 6}</code>
Test Subset	<code>set1.issubset(set2)</code>
Test Superset	<code>set1.issuperset(set2)</code>
Test sets have same content	<code>set1 == set2</code>

Set operations :

Union (aka 'or')	<code>set1 set2</code>
Intersection (aka 'and')	<code>set1 & set2</code>
Difference	<code>set1 - set2</code>
Symmetric Difference (aka 'xor')	<code>set1 ^ set2</code>

Source: <https://www.datacamp.com/community/tutorials/python-data-science-cheat-sheet-basics#gs.0x1rxEA>

Python For Data Science Cheat Sheet

Python Basics

Learn More Python for Data Science [Interactively at www.dataquest.io](https://www.dataquest.io)

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

>>> x+2	Sum of two variables
>>> x-2	Subtraction of two variables
>>> x*2	Multiplication of two variables
>>> x**2	Exponentiation of a variable
>>> x%2	Remainder of a variable
>>> x/float(2)	Division of a variable

Types and Type Conversion

str()	"5", "3.15", "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)
```

Strings

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

String Operations

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

String Methods

```
>>> my_string.upper()
'THISSTRINGISAWESOME'
>>> my_string.lower()
'thisstringisawesome'
>>> my_string.count('t')
2
>>> my_string.replace('o', 'l')
'thislstringlslsawesomel'
>>> my_string.strip()
'thisStringIsAwesome'
```

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'
>>> my_list[-3]
'list'
>>> my_list[1:3]
['my', 'list']
>>> my_list[1:]
['my', 'list', 'is', 'nice']
>>> my_list[:3]
['my', 'list', 'is']
>>> my_list[:]
['my', 'list', 'is', 'nice']
>>> my_list2[1][0]
4
>>> my_list2[1][:2]
[4, 5]
```

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)
2
>>> my_list.count(a)
1
>>> my_list.append('!')
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice', '!']
>>> my_list.remove('!')
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> del my_list[0:1]
['list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list.reverse()
['nice', 'is', 'list', 'my', 'nice', 'is', 'list']
>>> my_list.extend('!!')
['nice', 'is', 'list', 'my', 'nice', 'is', 'list', '!!']
>>> my_list.pop(-1)
'!'
>>> my_list.insert(0, '!')
['!', 'nice', 'is', 'list', 'my', 'nice', 'is', 'list']
>>> my_list.sort()
['!', 'is', 'list', 'my', 'nice', 'is', 'list']
```




Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
>>> from math import pi
```

 Data analysis
 Machine learning
 Scientific computing
 2D plotting

Install Python

 Leading open data science platform powered by Python
 Free IDE that is included with Anaconda
 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
>>> my_array[0:2]
array([1, 2])
>>> my_2darray[1,0]
4
>>> my_2darray[1,:]
array([4, 5, 6])
```

NumPy Array Operations

```
>>> my_array > 3
array([0, 1, 2, 3])
>>> 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
(4,)
>>> np.append(other_array)
array([0, 1, 2, 3, 4, 5])
>>> np.insert(my_array, 1, 5)
array([0, 5, 1, 2, 3, 4])
>>> np.delete(my_array, [1])
array([0, 2, 3, 4])
>>> np.mean(my_array)
1.5
>>> np.median(my_array)
2.0
>>> my_array.corrcoef()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
>>> np.std(my_array)
1.0
```

DataCamp

Learn Python for Data Science [Interactively](https://www.dataquest.io)

Numpy

Source: <https://www.dataquest.io/blog/numpy-cheat-sheet/>



Data Science Cheat Sheet

Numpy

KEY

We'll use shorthand in this cheat sheet
arr - A numpy Array object

IMPORTS

Import these to start
import numpy as np

IMPORTING/EXPORTING

```
np.loadtxt('file.txt') - From a text file
np.genfromtxt('file.csv', delimiter=',')
    - From a CSV file
np.savetxt('file.txt', arr, delimiter=' ')
    - Writes to a text file
np.savetxt('file.csv', arr, delimiter=',')
    - Writes to a CSV file
```

CREATING ARRAYS

```
np.array([1,2,3]) - One dimensional array
np.array([(1,2,3), (4,5,6)]) - Two dimensional array
np.zeros(3) - 1D array of length 3 all values 0
np.ones((3,4)) - 3x4 array with all values 1
np.eye(5) - 5x5 array of 0 with 1 on diagonal (Identity matrix)
np.linspace(0,100,6) - Array of 6 evenly divided values from 0 to 100
np.arange(0,10,3) - Array of values from 0 to less than 10 with step 3 (eg [0,3,6,9])
np.full((2,3),8) - 2x3 array with all values 8
np.random.rand(4,5) - 4x5 array of random floats between 0-1
np.random.rand(6,7)*100 - 6x7 array of random floats between 0-100
np.random.randint(5, size=(2,3)) - 2x3 array with random ints between 0-4
```

INSPECTING PROPERTIES

```
arr.size - Returns number of elements in arr
arr.shape - Returns dimensions of arr (rows, columns)
arr.dtype - Returns type of elements in arr
arr.astype(dtype) - Convert arr elements to type dtype
arr.tolist() - Convert arr to a Python list
np.info(np.eye) - View documentation for np.eye
```

COPYING/SORTING/RESHAPING

```
np.copy(arr) - Copies arr to new memory
arr.view(dtype) - Creates view of arr elements with type dtype
arr.sort() - Sorts arr
arr.sort(axis=0) - Sorts specific axis of arr
two_d_arr.flatten() - Flattens 2D array two_d_arr to 1D
```

```
arr.T - Transposes arr (rows become columns and vice versa)
arr.reshape(3,4) - Reshapes arr to 3 rows, 4 columns without changing data
arr.resize((5,6)) - Changes arr shape to 5x6 and fills new values with 0
```

ADDING/REMOVING ELEMENTS

```
np.append(arr, values) - Appends values to end of arr
np.insert(arr, 2, values) - Inserts values into arr before index 2
np.delete(arr, 3, axis=0) - Deletes row on index 3 of arr
np.delete(arr, 4, axis=1) - Deletes column on index 4 of arr
```

COMBINING/SPLITTING

```
np.concatenate((arr1, arr2), axis=0) - Adds arr2 as rows to the end of arr1
np.concatenate((arr1, arr2), axis=1) - Adds arr2 as columns to end of arr1
np.split(arr, 3) - Splits arr into 3 sub-arrays
np.hsplit(arr, 5) - Splits arr horizontally on the 5th index
```

INDEXING/SLICING/SUBSETTING

```
arr[5] - Returns the element at index 5
arr[2,5] - Returns the 2D array element on index [2][5]
arr[1]=4 - Assigns array element on index 1 the value 4
arr[1,3]=10 - Assigns array element on index [1][3] the value 10
arr[0:3] - Returns the elements at indices 0,1,2 (On a 2D array: returns rows 0,1,2)
arr[0:3,4] - Returns the elements on rows 0,1,2 at column 4
arr[:2] - Returns the elements at indices 0,1 (On a 2D array: returns rows 0,1)
arr[:,1] - Returns the elements at index 1 on all rows
arr<5 - Returns an array with boolean values (arr1<3) & (arr2>5) - Returns an array with boolean values
~arr - Inverts a boolean array
arr[arr<5] - Returns array elements smaller than 5
```

SCALAR MATH

```
np.add(arr, 1) - Add 1 to each array element
np.subtract(arr, 2) - Subtract 2 from each array element
np.multiply(arr, 3) - Multiply each array element by 3
np.divide(arr, 4) - Divide each array element by 4 (returns np.nan for division by zero)
np.power(arr, 5) - Raise each array element to the 5th power
```

VECTOR MATH

```
np.add(arr1, arr2) - Elementwise add arr2 to arr1
np.subtract(arr1, arr2) - Elementwise subtract arr2 from arr1
np.multiply(arr1, arr2) - Elementwise multiply arr1 by arr2
np.divide(arr1, arr2) - Elementwise divide arr1 by arr2
np.power(arr1, arr2) - Elementwise raise arr1 raised to the power of arr2
np.array_equal(arr1, arr2) - Returns True if the arrays have the same elements and shape
np.sqrt(arr) - Square root of each element in the array
np.sin(arr) - Sine of each element in the array
np.log(arr) - Natural log of each element in the array
np.abs(arr) - Absolute value of each element in the array
np.ceil(arr) - Rounds up to the nearest int
np.floor(arr) - Rounds down to the nearest int
np.round(arr) - Rounds to the nearest int
```

STATISTICS

```
np.mean(arr, axis=0) - Returns mean along specific axis
arr.sum() - Returns sum of arr
arr.min() - Returns minimum value of arr
arr.max(axis=0) - Returns maximum value of specific axis
np.var(arr) - Returns the variance of array
np.std(arr, axis=1) - Returns the standard deviation of specific axis
arr.corrcoef() - Returns correlation coefficient of array
```

Numpy Cheat Sheet

PYTHON PACKAGE
Created by: Arianne Colton, Sean Chen

NUMPY (NUMERICAL PYTHON)

What is NumPy?
 Foundation package for scientific computing in Python

Why NumPy?

- NumPy 'ndarray' is a much more efficient way of storing and manipulating "numerical data" than the built-in Python data structures.
- Libraries written in lower-level languages, such as C, can operate on data stored in NumPy 'ndarray' without copying any data.

N-DIMENSIONAL ARRAY (NDARRAY)

What is NdArray?
 Fast and space-efficient multidimensional array (container for homogeneous data) providing vectorized arithmetic operations

Create NdArray	<code>np.array([1,2,3])</code> # seq1 - is any sequence like object, i.e. [1, 2, 3]
Create Special NdArray	<code>1, np.zeros(10)</code> # one dimensional ndarray with 10 elements of value 0 <code>2, np.ones(2, 3)</code> # two dimensional ndarray with 6 elements of value 1 <code>3, np.empty(3, 4, 5) *</code> # three dimensional ndarray of uninitialized values <code>4, np.eye(N)</code> or <code>np.identity(N)</code> # creates N by N identity matrix
NdArray version of Python's range	<code>np.arange(1, 10)</code>
Get # of Dimension	<code>ndarray1.ndim</code>
Get Dimension Size	<code>dim1size, dim2size, ... = ndarray1.shape</code>
Get Data Type **	<code>ndarray1.dtype</code>
Explicit Casting	<code>ndarray2 = ndarray1.astype(np.float32) ***</code>

* Cannot assume empty() will return all zeros. It could be garbage values.

Slicing (INDEXING/SUBSETTING)

- Slicing (i.e. `ndarray1[2:6]`) is a 'view' on the original array. **Data is NOT copied.** Any modifications (i.e. `ndarray1[2:6] = 8`) to the 'view' will be reflected in the original array.
- Instead of a 'view', explicit copy of slicing via:
`ndarray1[2:6].copy()`
- Multidimensional array indexing notation:
`ndarray1[0][2]` or `ndarray1[0, 2]`

*** Boolean Indexing :**

```
ndarray1[names == "Bob"] : (names == "Will", 2:]
# '2:' means select from 3rd column on
```

- Selecting data by boolean indexing **ALWAYS** creates a copy of the data.
- The 'and' and 'or' keywords do NOT work with boolean arrays. Use & and |.

*** Fancy indexing (aka 'indexing using integer arrays')**
 Select a subset of rows in a particular order:

```
ndarray1[ [3, 8, 4] ]
ndarray1[ [-1, 6] ]
# negative indices select rows from the end
```

- Fancy indexing **ALWAYS** creates a copy of the data.

NUMPY (NUMERICAL PYTHON)

Setting data with assignment :

```
ndarray1[ndarray1 < 0] = 0 *
```

- If ndarray1 is two-dimensional, ndarray1 < 0 creates a two-dimensional boolean array.

COMMON OPERATIONS

- Transposing**
 - A special form of reshaping which returns a 'view' on the underlying data without copying anything.

```
ndarray1.transpose() or ndarray1.T
ndarray1.swapaxes(0, 1)
```
- Vectorized wrappers (for functions that take scalar values)**
 - `math.sqrt()` works on only 1 scalar
 - `np.sqrt(seq1)` # any sequence (list, ndarray, etc) to return a ndarray
- Vectorized expressions**
 - `np.where(cond, x, y)` is a vectorized version of the expression 'x if condition else y'

```
np.where([True, False], [1, 2], [2, 3]) => ndarray([1, 3])
```
- Common Usages :**

```
np.where(matrixNdArray > 0, 1, -1)
=> a new array (same shape) of 1 or -1 values
```

```
np.where(cond, 1, 0).argmax() *
=> Find the first True element
```

 - `argmax()` can be used to find the index of the maximum element. Example usage is find the first element that has a 'price > number' in an array of price data.

4. Aggregations/Reductions Methods (i.e. mean, sum, std)

Compute mean	<code>ndarray1.mean()</code> or <code>np.mean(ndarray1)</code>
Compute statistics over axis *	<code>ndarray1.mean(axis = 1)</code> <code>ndarray1.sum(axis = 0)</code>

 - axis = 0 means column axis, 1 is row axis.

5. Boolean arrays methods

Count # of 'True's in boolean array	<code>(ndarray1 > 0).sum()</code>
If at least one value is 'True'	<code>ndarray1.any()</code>
If all values are 'True'	<code>ndarray1.all()</code>

Note: These methods also work with non-boolean arrays, where non-zero elements evaluate to True.

6. Sorting

Inplace sorting	<code>ndarray1.sort()</code>
Return a sorted copy instead of inplace	<code>sorted1 = np.sort(ndarray1)</code>

7. Set methods

Return sorted unique values	<code>np.unique(ndarray1)</code>
Test membership of ndarray1 values in [2, 3, 6]	<code>resultBooleanArray = np.in1d(ndarray1, [2, 3, 6])</code>

 - Other set methods : `intersect1d()`, `union1d()`, `setdiff1d()`, `setxor1d()`

8. Random number generation (np.random)

 - Supplements the built-in Python random * with functions for efficiently generating whole arrays of sample values from many kinds of probability distributions.

```
samples = np.random.normal(scale = (3, 3))
```

 - Python built-in random ONLY samples one value at a time.

Created by Arianne Colton and Sean Chen
www.data-science-free.com
 Based on content from 'Python for Data Analysis' by Wes McKinney
 Updated: August 18, 2016

Source: <https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs.Nw3V6CE>

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

```
>>> a = np.array([1, 2, 3])
```

2D array

```
>>> b = np.array([[1, 2, 3], [4, 5, 6]])
```

3D array

```
>>> c = np.array([[[1, 2, 3], [4, 5, 6]], [[3, 2, 1], [4, 5, 6]]])
```

Creating Arrays

Initial Placeholders

```
>>> np.zeros(3, 4)
```

Create an array of zeros

```
>>> np.ones(10, 25, 5)
```

Create an array of ones

```
>>> np.linspace(0, 2, 5)
```

Create an array of evenly spaced values (step value)

```
>>> f = np.full((2, 2), 7)
```

Create an array of evenly spaced values (number of samples)

```
>>> e = np.eye(2)
```

Create a constant array

```
>>> np.random.random((2, 2))
```

Create a 2x2 identity matrix

```
>>> np.empty(3, 2)
```

Create an array with random values

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
```

Save an array to disk

```
>>> np.savez('my_array.npz', a, b)
```

Save multiple arrays to disk

```
>>> np.load('my_array.npy')
```

Load an array from disk

Saving & Loading Text Files

```
>>> np.loadtxt('myfile.txt')
```

Load a text file

```
>>> np.genfromtxt('myfile.csv', delimiter=',')
```

Load a CSV file

```
>>> np.savetxt('myarray.txt', a, delimiter=' ')
```

Save an array to a text file

Data Types

```
>>> np.int64
```

Signed 64-bit integer type

```
>>> 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
```

Addition

```
>>> 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)
```

Sine of an array

```
>>> np.cos(b)
```

Element-wise cosine

```
>>> np.log(a)
```

Element-wise natural logarithm

```
>>> np.dot(a, b)
```

Dot product

Comparison

```
>>> a == b
```

Element-wise comparison

```
>>> np.all(a == b)
```

Element-wise comparison

```
>>> a < b
```

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

```
>>> a.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 0 column 2 (equivalent to b[1][2])

Slicing

```
>>> a[0:2]
```

Select items at index 0 and 1

```
>>> np.arange(5, 10)
```

Select items at rows 0 and 1 in column 1

```
>>> b[0:2, 1]
```

Select all items at row 0 (equivalent to b[0], +1)

```
>>> a[1::2]
```

Same as [1, 3, 5]

Boolean Indexing

```
>>> a[a > 2]
```

Reversed array a

```
>>> np.invert(a)
```

Select elements from a less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]
```

Select elements (1, 0, 1, 0) and (0, 1, 2, 0)

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]
```

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> l = np.transpose(b)
```

Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
```

Flatten the array

```
>>> q.reshape(3, -2)
```

Reshape, but don't change data

Adding/Removing Elements

```
>>> b.resize(2, 4)
```

Return a new array with shape (2, 4)

```
>>> np.append(b, q)
```

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, b), axis=0)
```

Concatenate arrays

```
>>> np.vstack((a, b))
```

Stack arrays vertically (row-wise)

```
>>> np.hstack((a, b))
```

Stack arrays horizontally (column-wise)

```
>>> np.c_[a, b]
```

Create stacked column-wise arrays

```
>>> np.r_[a, b]
```

Create stacked row-wise arrays

Splitting Arrays

```
>>> np.hsplit(a, 2)
```

Split the array horizontally at the 2nd index

```
>>> np.vsplit(a, 2)
```

Split the array vertically at the 2nd index

Source: <https://github.com/donnemartin/data-science-ipython-notebooks/blob/master/numpy/numpy.ipynb>

NumPy

Credits: Forked from [Parallel Machine Learning with scikit-learn and IPython](#) by Olivier Grisel

- NumPy Arrays, dtype, and shape
- Common Array Operations
- Reshape and Update In-Place
- Combine Arrays
- Create Sample Data

```
In [1]: import numpy as np
```

NumPy Arrays, dtypes, and shapes

```
In [2]: a = np.array([1, 2, 3])
print(a)
print(a.shape)
print(a.dtype)
```

```
[1 2 3]
(3,)
int64
```

```
In [3]: b = np.array([[0, 2, 4], [1, 3, 5]])
print(b)
print(b.shape)
print(b.dtype)
```

```
[[0 2 4]
 [1 3 5]]
(2, 3)
int64
```

Pandas

Source: <http://datasciencefree.com/pandas.pdf>

Data Analysis with PANDAS

CHEAT SHEET

Created by: [Anurag Choudhary](#) and [Siddhant](#)

DATA STRUCTURES

SERIES (1D)

One-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its "index". If index of data is not specified, then a default one consisting of the integers 0 through N-1 is created.

Create Series	<code>series1 = pd.Series([1, 2], index=['a', 'b'])</code> <code>series1 = pd.Series([1, 2])</code>
Get Series Values	<code>series1.values</code>
Get Values by Index	<code>series1['a']</code> <code>series1[['b', 'a']]</code>
Get Series Index	<code>series1.index</code>
Get Name Attribute (None is default)	<code>series1.name</code> <code>series1.index.name</code>
** Common Index Values are Added	<code>series1 = series2</code>
Unique But Unsorted	<code>series2 = series1.unique()</code>

- * Can think of Series as a fixed-length, ordered dict. Series can be subdivided into many functions that expect a dict.
- ** Auto-align differently-indexed data in arithmetic operations.

DATAFRAME (2D)

Tabular data structure with ordered collections of columns, each of which can be different value type. Data Frame (DF) can be thought of as a dict of Series.

Create DF (from a dict of equal-length lists or NumPy arrays)	<code>dict1 = {'state': ['Ohio', 'CA'], 'year': [2000, 2010]}</code> <code>df1 = pd.DataFrame(dict1)</code> # columns are placed in sorted order <code>df1 = pd.DataFrame(dict1, index=['row1', 'row2'])</code> # specifying index <code>df1 = pd.DataFrame(dict1, columns=['year', 'state'])</code> # columns are placed in your given order
* Create DF (from nested dict of dicts)	<code>dict1 = {'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 3, 'row2': 4}}</code> <code>df1 = pd.DataFrame(dict1)</code>

The inner keys as row indices

PANEL DATA (3D)

Create Panel Data: (Each item in the Panel is a DF)

```
import pandas_datareader.data as web
panel1 = pd.Panel({atk : web.get_data_yahoo(atk, "1/1/2000", "1/1/2010") for atk in ["AAPL", "IBM"]})
# panel1 Dimensions : 2 (items) * 60 (major) * 6 (minor)
```

"Stacked" DF form: (Useful way to represent panel data)

```
panel1 = panel1.swapaxes("item", "minor")
panel1.ix[:, "6/1/2009", :].to_frame() *
=> Stacked DF (with hierarchical indexing "i")
# Open High Low Close Volume Adj-Close
```

# major	minor
# 2009-06-01	AAPL
#	IBM
# 2009-06-02	AAPL
#	IBM

DATA STRUCTURES CONTINUED

* DF has a "to_panel()" method which is the inverse of "to_frame()".

** Hierarchical indexing makes N-dimensional arrays unnecessary in a lot of cases. Also prefer to use Stacked DF, not Panel data.

INDEX OBJECTS

Immutable objects that hold the axis labels and other metadata (i.e. axis name)

- * i.e. Index, MultiIndex, DatetimeIndex, PeriodIndex
- * Any sequence of labels used when constructing Series or DF internally converted to an Index.
- * Can function as fixed-size set in addition to being array-like.

HIERARCHICAL INDEXING

Multiple index levels on an axis: A way to work with higher dimensional data in a lower dimensional form.

MultiIndex:

```
series1 = Series(np.random.randn(6), index=[["a", "a", "a", "b", "b", "b"], [1, 2, 3, 1, 2, 3]])
series1.index.names = ["key1", "key2"]
```

Series Partial Indexing

```
series1["b"] # Outer Level
series1.ix[:, 2] # Inner Level
df1["outerCol3", "innerCol2"]
Or
df1["outerCol3"].ix["innerCol2"]
```

DF Partial Indexing

Swapping and Sorting Levels

Swap Level (level unchanged)	<code>swapLevel1 = series1.swapLevel("key1", "key2")</code>
Sort Level	<code>series1.sortLevel(1)</code> # sorts according to first inner level

Common Ops: `series1.swapLevel(0, 1).sortLevel(0)`
the order of rows also change

* The order of the rows do not change. Only the two levels got swapped.

** Data selection performance is much better if the index is sorted starting with the outermost level, as a result of calling `sortLevel(0)` or `sort_index()`.

Summary Statistics by Level

Most stats functions in DF or Series have a "level" option that you can specify the level you want on an axis.

Sum rows (that have same "key2" value)	<code>df1.sum(level="key2")</code>
Sum columns	<code>df1.sum(level="col3", axis=1)</code>

* Under the hood, the functionality provided here utilizes pandas's "groupby".

DataFrame's Columns as Indexes

DF's "set_index" will create a new DF using one or more of its columns as the index.

New DF using columns as index	<code>df2 = df1.set_index(["col3", "col4"])</code> # col3 becomes the outermost index, col4 becomes inner index. Values of col3, col4 become the index values.
-------------------------------	---

* "reset_index" does the opposite of "set_index", the hierarchical index are moved into columns.

! By default, "col3" and "col4" will be removed from the DF, though you can leave them by option: `drop = False`.

MISSING DATA

Python	<code>NaN</code> or <code>np.nan</code> (not a number)
Pandas *	<code>NaN</code> or python built-in <code>None</code> mean missing/NA values

* Use `pd.isnull()` or `pd.notnull()` or `series1/df1.isnull()` to detect missing data.

FILTERING OUT MISSING DATA

`dropna()` returns with ONLY non-null data, source data NOT modified.

`df1.dropna()` # drop any row containing missing value

`df1.dropna(axis=1)` # drop any column containing missing values

`df1.dropna(how="all")` # drop row that are all missing

`df1.dropna(how="any")` # drop any row containing < 3 number of observations

FILLING IN MISSING DATA

`df2 = df1.fillna(0)` # fill all missing data with 0

`df1.fillna(inplace=True)` # modify in-place

Use a different fill value for each column:

```
df1.fillna({"col1": 0, "col2": -1})
```

Only forward fill the 2 missing values in front:

```
df1.fillna(method="ffill", limit=2)
```

i.e. for column1, if row 3-4 are missing, so 3 and 4 get filled with the value from 2, NOT 5 and 6.

Source: <https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#gs.S4P4T=U>

Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science interactively at www.datacamp.com

Pandas

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

Use the following import convention:

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

Pandas Data Structures

Series

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

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

DataFrame

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'])
```

I/O

Read and Write to CSV

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

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
>>> xlsw = pd.ExcelFile('file.xlsx')
>>> df = pd.read_excel(xlsw, '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_query("SELECT * FROM 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)
```

Asking For Help

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

Selection

Getting

```
>>> s['b']
7
>>> df[1:]
   Country  Capital  Population
1  India    New Delhi  1303171035
2  Brazil   Brasilia  207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0], [0]]
"Belgium"
>>> df.ias[[0], [0]]
"Belgium"
```

By Label

```
>>> df.loc[[0], ['Country']]
"Belgium"
>>> df.at[[0], ['Country']]
"Belgium"
```

By Label/Position

```
>>> df.ix[2]
Country    Brazil
Capital    Brasilia
Population  207847528
>>> df.ix[:, 'Capital']
0    Brussels
1    New Delhi
2    Brasilia
>>> df.ix[1, 'Capital']
"New Delhi"
```

Boolean Indexing

```
>>> s[s > 1]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population'] > 1200000000]
```

Setting

```
>>> s['a'] = 6
```

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
>>> df.drop('Country', axis=0)
```

Sort & Rank

```
>>> df.sort_index(by="Country")
>>> s.sort()
>>> df.rank()
```

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min() / df.max()
>>> df.iatmin() / df.iatmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> a3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> a + a3
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(s2, fill_value=0)
>>> s.sub(s2, fill_value=2)
>>> s.div(s2, fill_value=4)
>>> s.mul(s2, fill_value=3)
```

Source: <https://github.com/donnemartin/data-science-ipython-notebooks/blob/master/pandas/pandas.ipynb>

Pandas

Credits: The following are notes taken while working through [Python for Data Analysis](#) by Wes McKinney

- Series
- DataFrame
- Reindexing
- Dropping Entries
- Indexing, Selecting, Filtering
- Arithmetic and Data Alignment
- Function Application and Mapping
- Sorting and Ranking
- Axis Indices with Duplicate Values
- Summarizing and Computing Descriptive Statistics
- Cleaning Data (Under Construction)
- Input and Output (Under Construction)

```
In [1]: from pandas import Series, DataFrame
import pandas as pd
import numpy as np
```

Series

A Series is a one-dimensional array-like object containing an array of data and an associated array of data labels. The data can be any NumPy data type and the labels are the Series' index.

Create a Series:

```
In [2]: ser_1 = Series([1, 1, 2, -3, -5, 8, 13])
ser_1
```

```
Out[2]: 0    1
        1    1
        2    2
```

Matplotlib

Source: <https://www.datacamp.com/community/blog/python-matplotlib-cheat-sheet>

Python For Data Science Cheat Sheet

Matplotlib

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

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))
>>> V, R = np.meshgrid(3:3:100, ~3:3:100)
>>> V = 1 + R**2 + y
>>> V = 1 + R**2 + y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.loadiget_sample_data('mss_grid/biorate_normal.npy')
```

2 Create Plot

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.gcf().figsize*(2,1))
```

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)
```

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([0.5,1.2,3],[0.5,1,2])
>>> axes[1,1].axhline(0.5)
>>> axes[0,1].axvline(0.5)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,
>>>               cmap=plt.cm.gray,
>>>               interpolation='nearest',
>>>               vmin=-2,
>>>               vmax=2)
```

Colormapped or RGB arrays

```
>>> ax.imshow(img,
>>>               cmap=plt.cm.gray,
>>>               interpolation='nearest',
>>>               vmin=-2,
>>>               vmax=2)
```

Vector Fields

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

Data Distributions

```
>>> ax1.hist(y)
>>> ax2.boxplot(y)
>>> ax3.violinplot(y)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolormesh(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(T,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].ax.contourf(CS)
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, y, 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,linestyle='solid')
>>> plt.plot(x,y,linestyle='dashed')
>>> plt.plot(x,y,linestyle='dotted',color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
>>>        2.3,
>>>        "Example Graph",
>>>        style='italic')
>>> ax.annotate("Data",
>>>            xy=(8, 0),
>>>            xytext=(10, 0),
>>>            textcoords='data',
>>>            arrowprops=dict(arrowstyle='->',
>>>                            connectionstyle='arc3',y))
```

Matplotlib

```
>>> plt.title("Example", fontsize=20)
>>> ax.set_xlabel("Example X-axis",
>>>               ylabel="Y-axis")
>>> ax.set_ylabel("Example Y-axis",
>>>               xlabel="X-axis")
>>> ax.legend(loc='best')
```

Limits & Autoscaling

```
>>> ax.margins(x=0,y=0.1)
>>> ax.axis('equal')
>>> ax.set_xlim(0,10),ylim=[-1.5,1.5])
>>> ax.set_xlim(0,10.5)
```

Legends

```
>>> ax.set_title("An Example Axes",
>>>               ylabel="Y-axis")
>>> ax.legend(loc='best')
```

Ticks

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

Subplot Spacing

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

Axes Spines

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

Close & Clear

```
>>> plt.clf()
>>> plt.cla()
>>> plt.close()
```

DataCamp

Learn Python for Data Science Interactively

Source: <https://github.com/donnemartin/data-science-ipython-notebooks/blob/master/matplotlib/matplotlib.ipynb>

matplotlib

Credits: Content forked from [Parallel Machine Learning with scikit-learn and IPython](#) by Olivier Grisel

- Setting Global Parameters
- Basic Plots
- Histograms
- Two Histograms on the Same Plot
- Scatter Plots

```
In [1]: %matplotlib inline
import pandas as pd
import numpy as np
import pylab as plt
import seaborn
```

Setting Global Parameters

```
In [2]: # Set the global default size of matplotlib figures
plt.rc('figure', figsize=(10, 5))

# Set seaborn aesthetic parameters to defaults
seaborn.set()
```

Basic Plots

```
In [3]: x = np.linspace(0, 2, 10)

plt.plot(x, x, 'o-', label='linear')
plt.plot(x, x ** 2, 'x-', label='quadratic')

plt.legend(loc='best')
plt.title('Linear vs Quadratic progression')
```

Scikit Learn

Source: <https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet#gs.fZ2A1Jk> [About](#) [Write](#) [Help](#) [Legal](#)

