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Given how rapidly the Machine Learning space is evolving, I imagine these will go out of date quickly, but at least as of June 1, 2017, they are pretty current.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

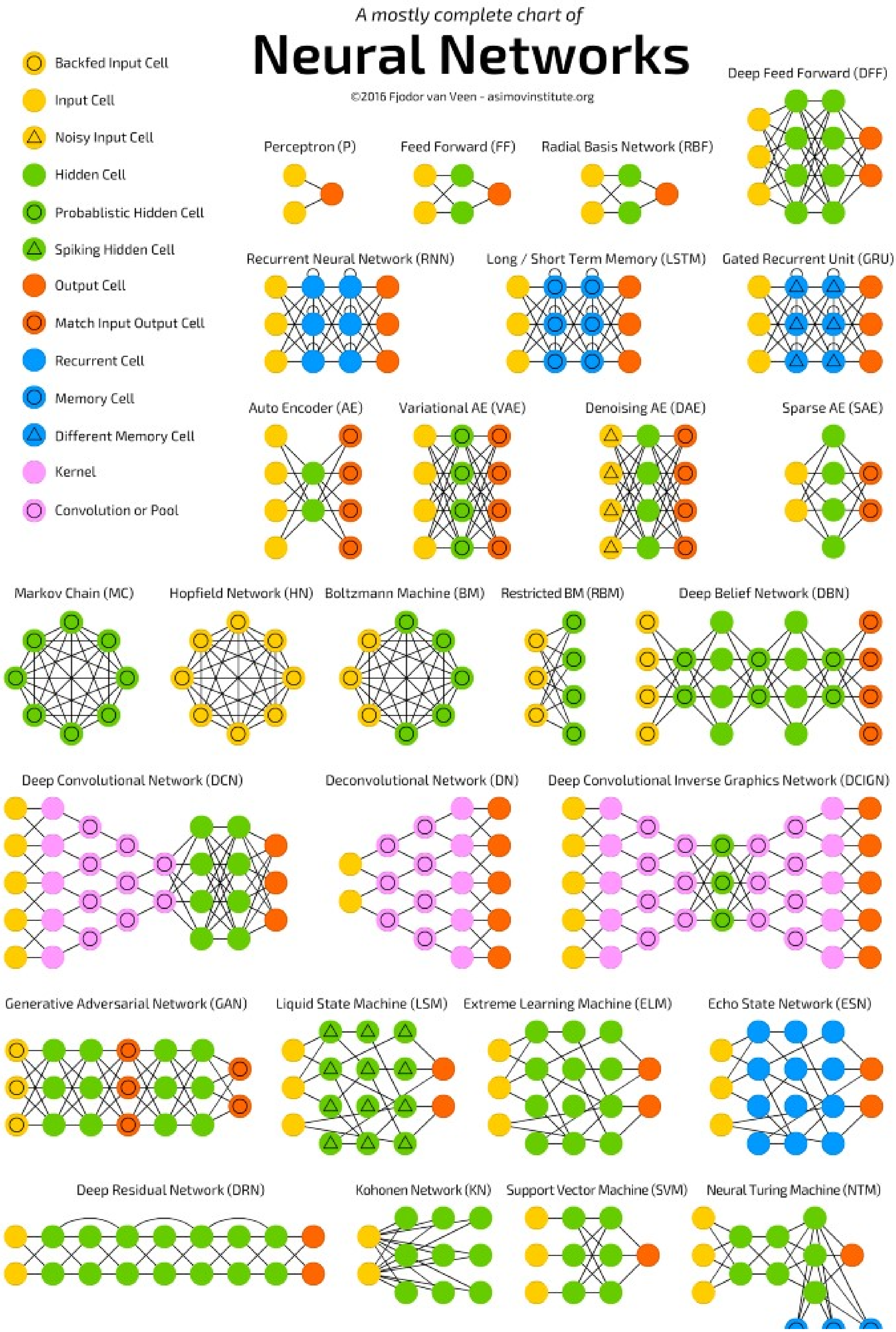
If you like this post, give it a ❤ below.

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.

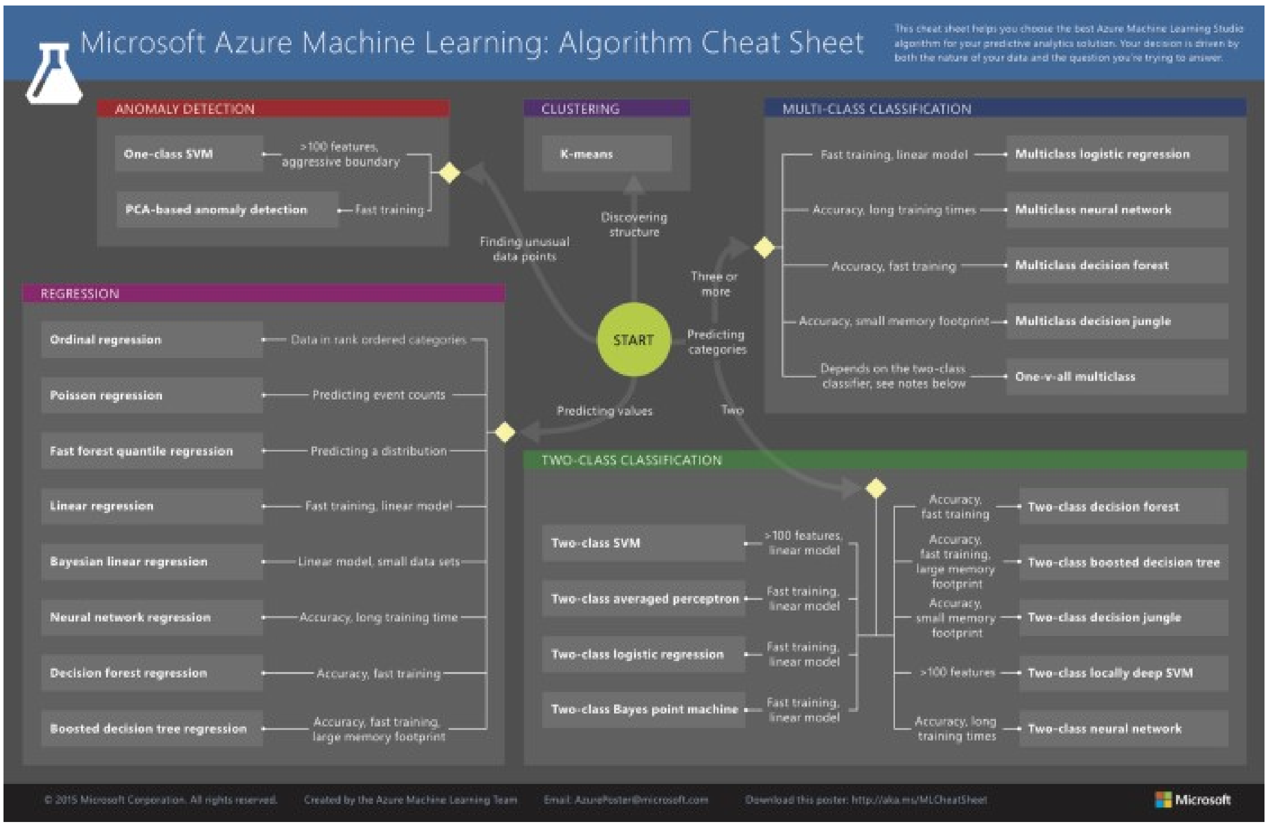
Neural Network Architectures

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


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Microsoft Azure Algorithm Flowchar

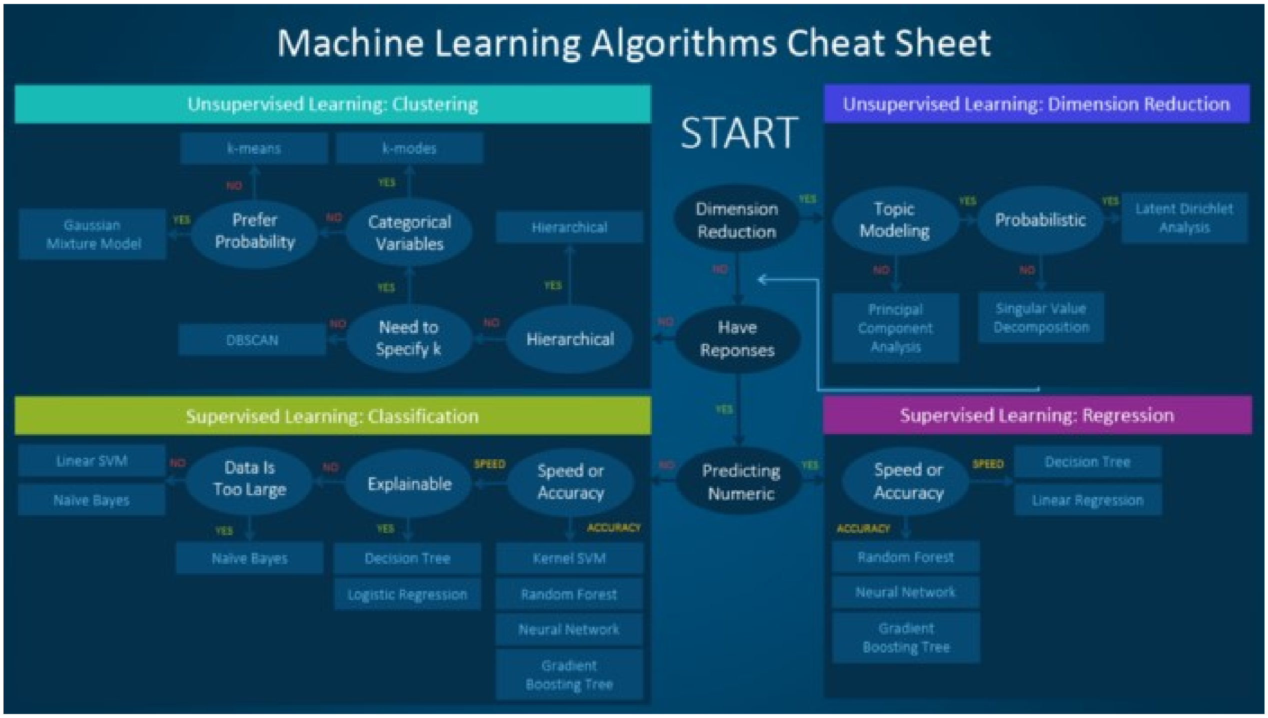
Source: https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-cheat-sheet



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

think big data

regression

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

bayesian

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

regularization

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

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

instance based

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

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

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)

dimesionality 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

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)

associated rule

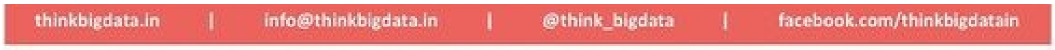
Apriori
Eclat
FP-Growth

ensemble

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

...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 dataiku

TOP PREDICTION ALGORITHMS

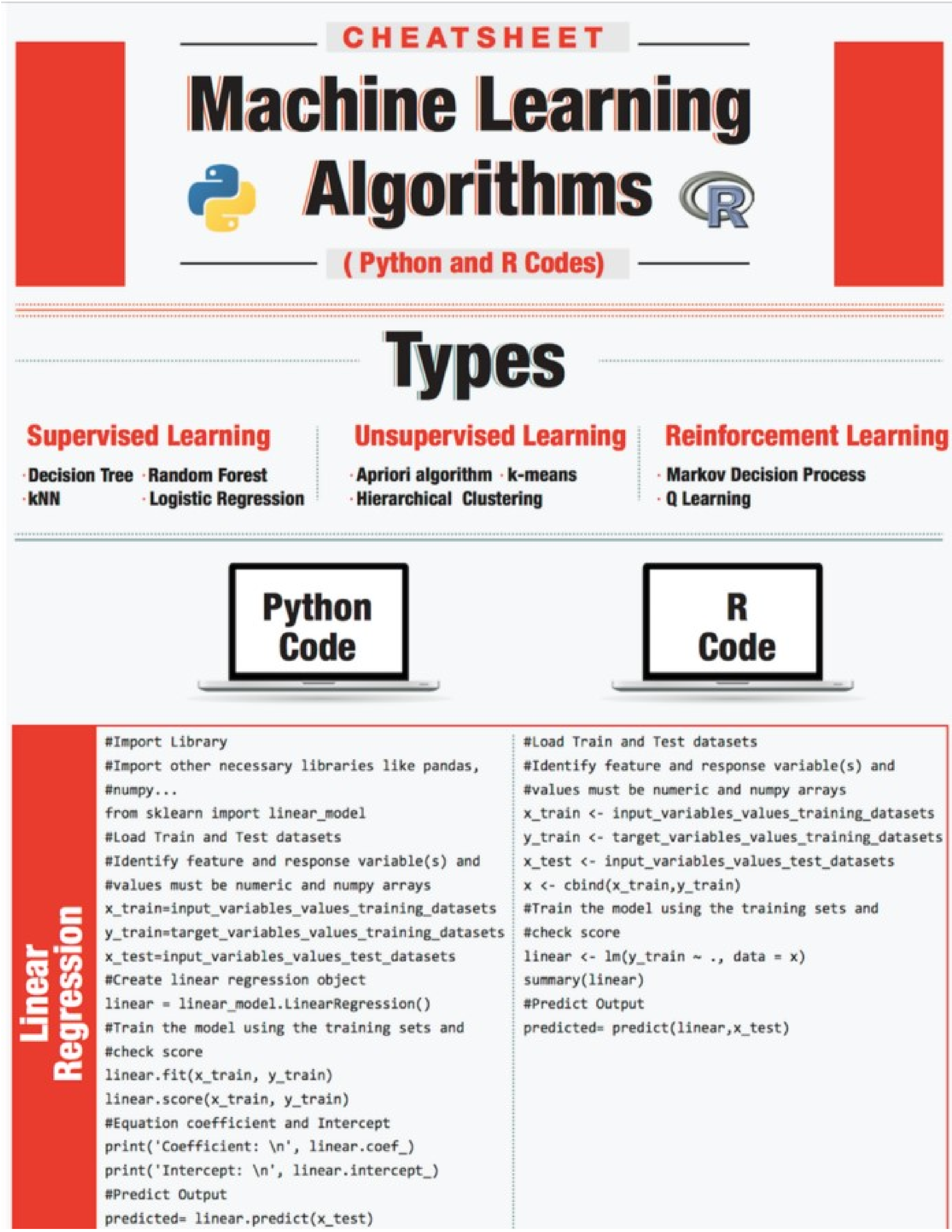
TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Linear	Logistic regression	The adaptation of **linear regression** to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	X Sometimes too simple to capture complex relationships between variables. X Tendency for the model to "overfit".
Tree-based	Decision tree	A graph that uses a **branching method** to match all possible outcomes of a decision.	Easy to understand and implement.	X Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.
Tree-based	Random Forest	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.	X Can be slow to output predictions relative to other algorithms. X Not easy to understand predictions.
Tree-based	Gradient Boosting	Uses even weaker decision trees, that are increasingly **focused on "hard" examples.**	High-performing.	X A small change in the feature set or training set can create radical changes in the model. X Not easy to understand predictions.
Neural networks	Neural networks	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.	X Very, very slow to train, because they have so many layers. Require a lot of power. X 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.

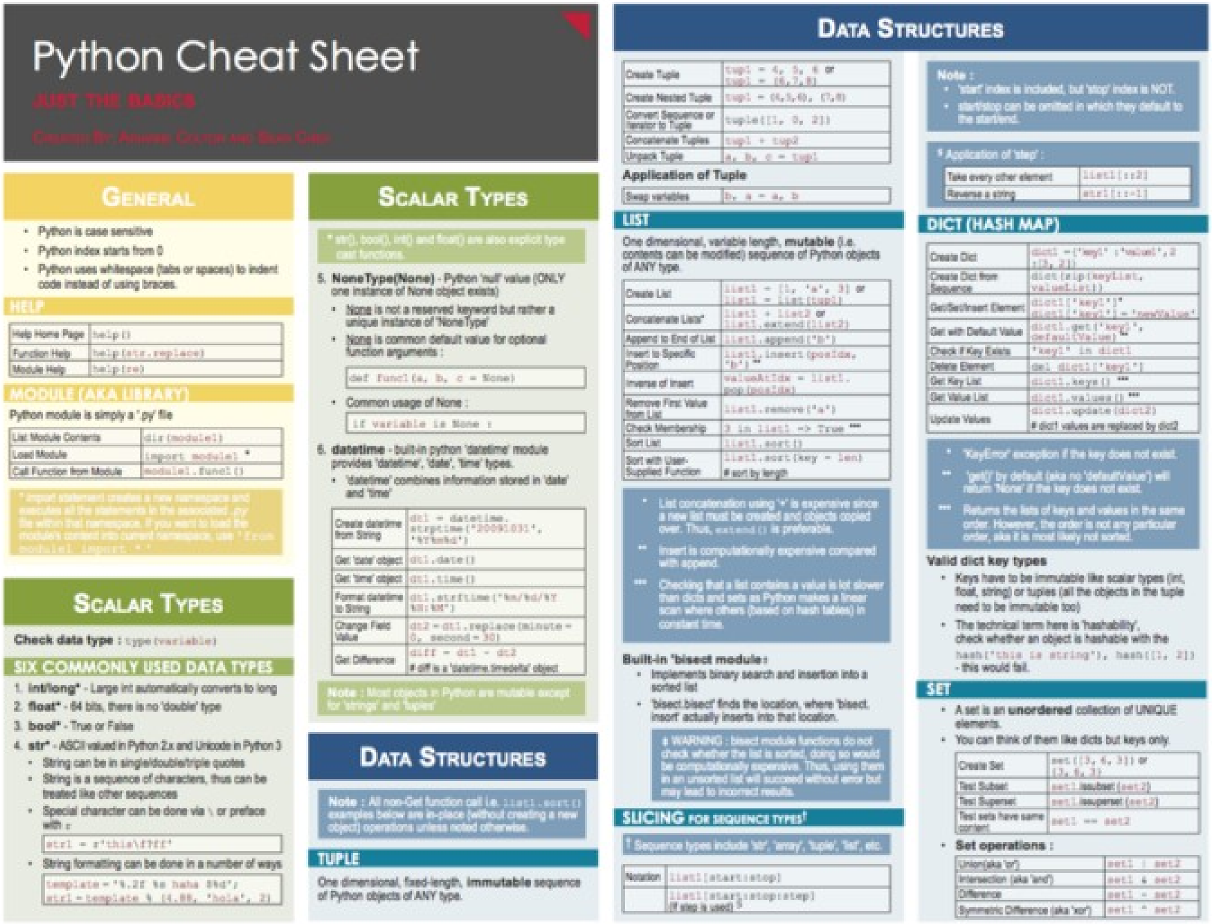
Algorithms

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

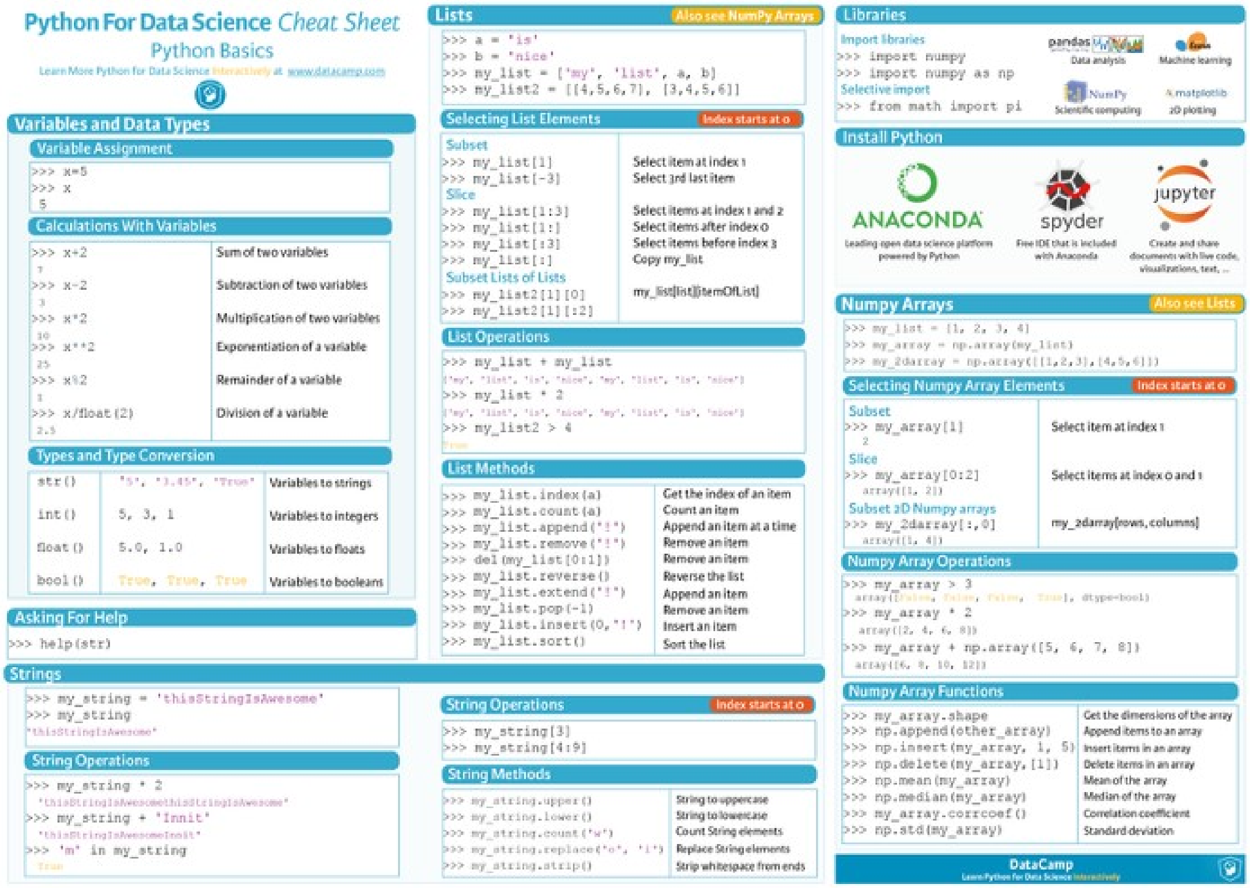


Python Basics

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



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.datacamp.com

Variables and Data Types

Variable Assignment

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

Calculations With Variables

`>>> x+2` `7`	Sum of two variables
`>>> x-2` `3`	Subtraction of two variables
`>>> x*2` `10`	Multiplication of two variables
`>>> x**2` `25`	Exponentiation of a variable
`>>> x%2` `1`	Remainder of a variable
`>>> x/float(2)` `2.5`	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists — Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements — Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]` `>>> my_list2[1][:2]`	my_list[list][itemOfList]

List Operations

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

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Strings

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

String Operations

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

String Operations — Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespace from ends

Libraries

Import libraries

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

Selective import

```
>>> from math import pi
```

pandas — Data analysis; learn — Machine learning; NumPy — Scientific computing; matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays — Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements — Index starts at 0

Subset	
`>>> my_array[1]` `2`	Select item at index 1
Slice	
`>>> my_array[0:2]` `array([1, 2])`	Select items at index 0 and 1
Subset 2D Numpy arrays	
`>>> my_2darray[:,0]` `array([1, 4])`	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp

Learn Python for Data Science Interactively

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 `5`th 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 `5`th power

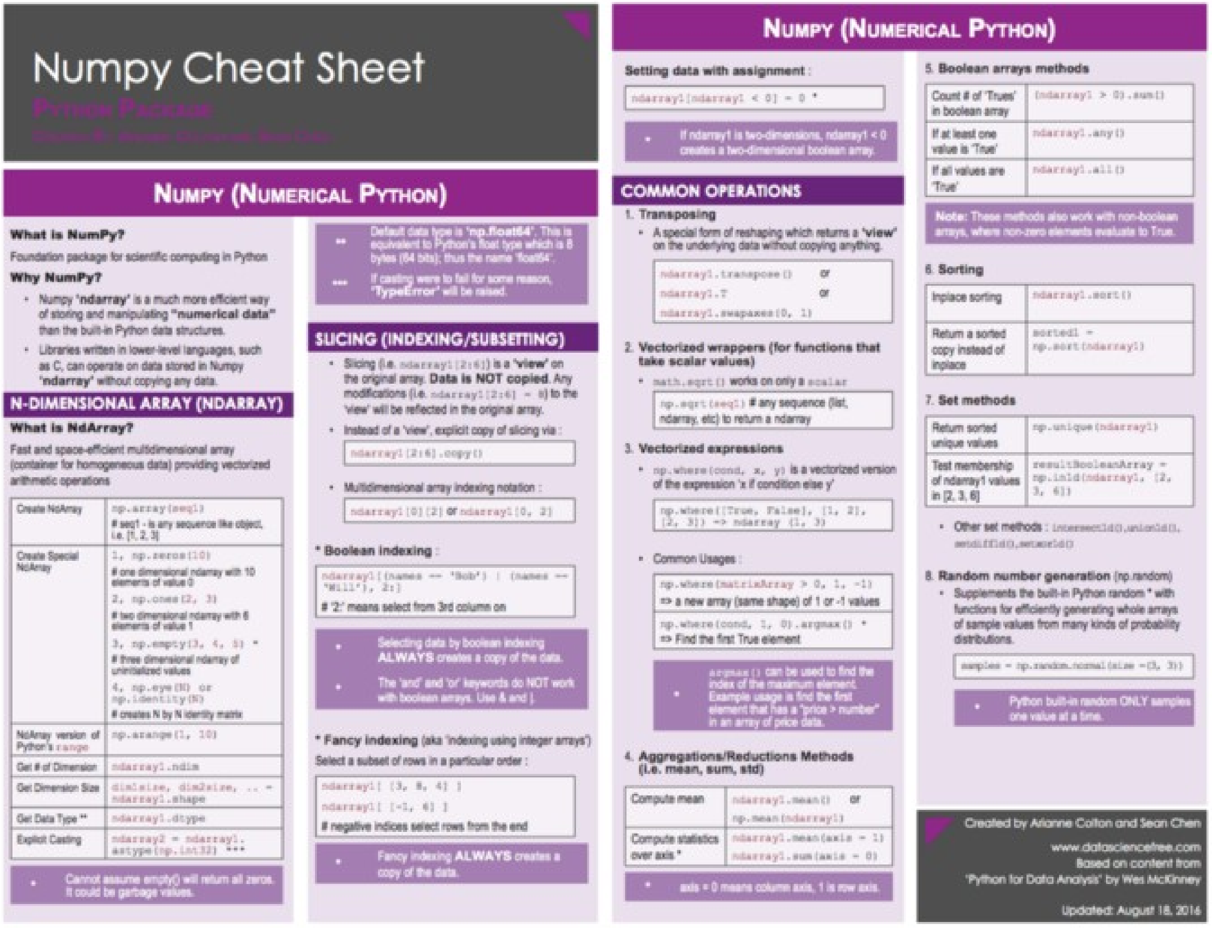
VECTOR MATH

- `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

Source: http://datasciencefree.com/numpy.pdf



Numpy Cheat Sheet

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	`np.array(seq1)` # seq1 - is any sequence like object, i.e. [1, 2, 3]
Create Special NdArray	`1, np.zeros(10)` # one dimensional ndarray with 10 elements of value 0 `2, np.ones(2, 3)` # two dimensional ndarray with 6 elements of value 1 `3, np.empty(3, 4, 5)` * # three dimensional ndarray of uninitialized values `4, np.eye(N)` or `np.identity(N)` # creates N by N identity matrix
NdArray version of Python's range	`np.arange(1, 10)`
Get # of Dimension	`ndarray1.ndim`
Get Dimension Size	`dim1size, dim2size, .. = ndarray1.shape`
Get Data Type **	`ndarray1.dtype`
Explicit Casting	`ndarray2 = ndarray1.astype(np.int32)` ***

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

** Default data type is **'np.float64'**. This is equivalent to Python's float type which is 8 bytes (64 bits); thus the name 'float64'.

*** If casting were to fail for some reason, **'TypeError'** will be raised.

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

COMMON OPERATIONS

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

```
ndarray1.transpose()    or
ndarray1.T              or
ndarray1.swapaxes(0, 1)
```

2. **Vectorized wrappers (for functions that take scalar values)**
   - `math.sqrt()` works on only a `scalar`

```
np.sqrt(seq1) # any sequence (list, ndarray, etc) to return a ndarray
```

3. **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(matrixArray > 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	`ndarray1.mean()` or `np.mean(ndarray1)`
Compute statistics over axis *	`ndarray1.mean(axis = 1)` `ndarray1.sum(axis = 0)`

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

5. **Boolean arrays methods**

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

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

6. **Sorting**

Inplace sorting	`ndarray1.sort()`
Return a sorted copy instead of inplace	`sorted1 = np.sort(ndarray1)`

7. **Set methods**

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

   - 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.

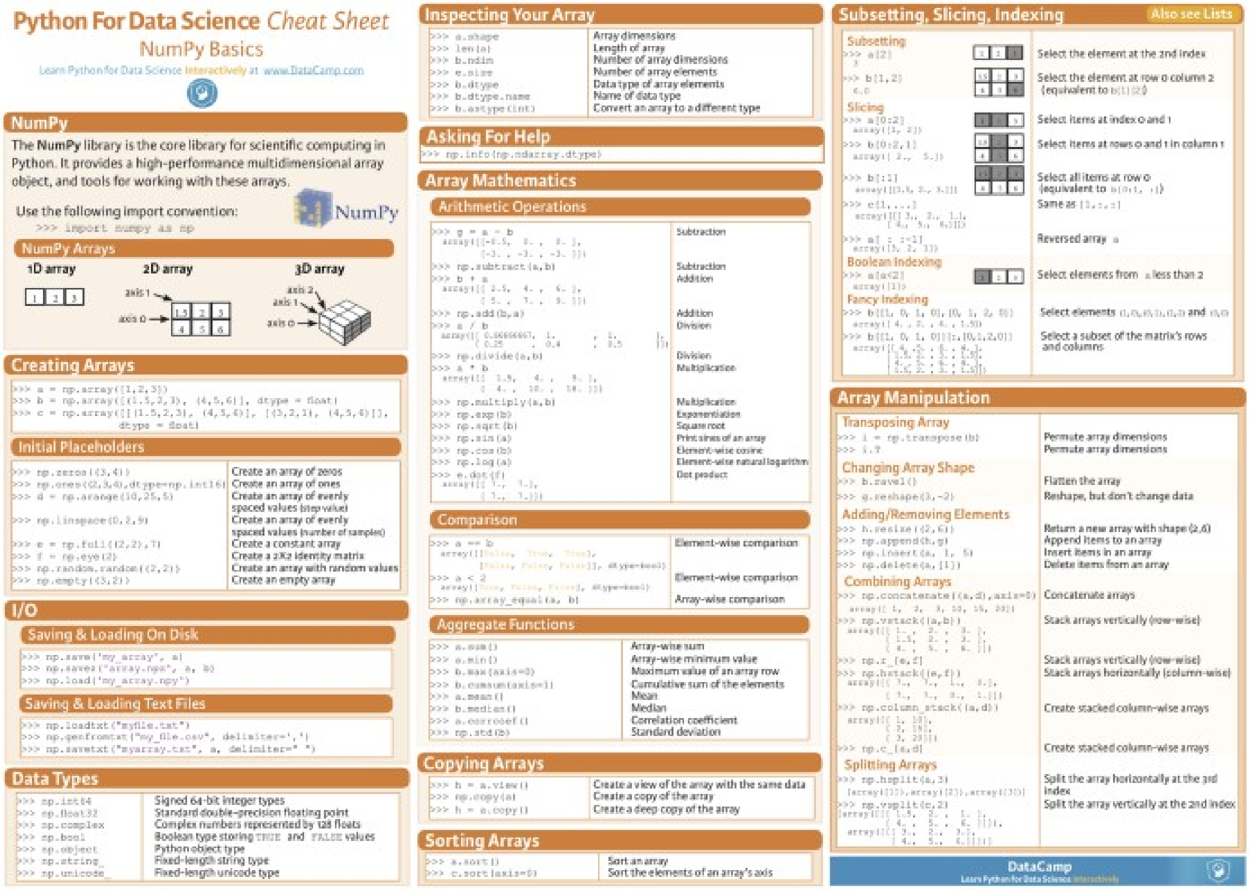
```
samples = np.random.normal(size =(3, 3))
```

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

Created by Arianne Colton and Sean Chen
www.datasciencefree.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



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)
```

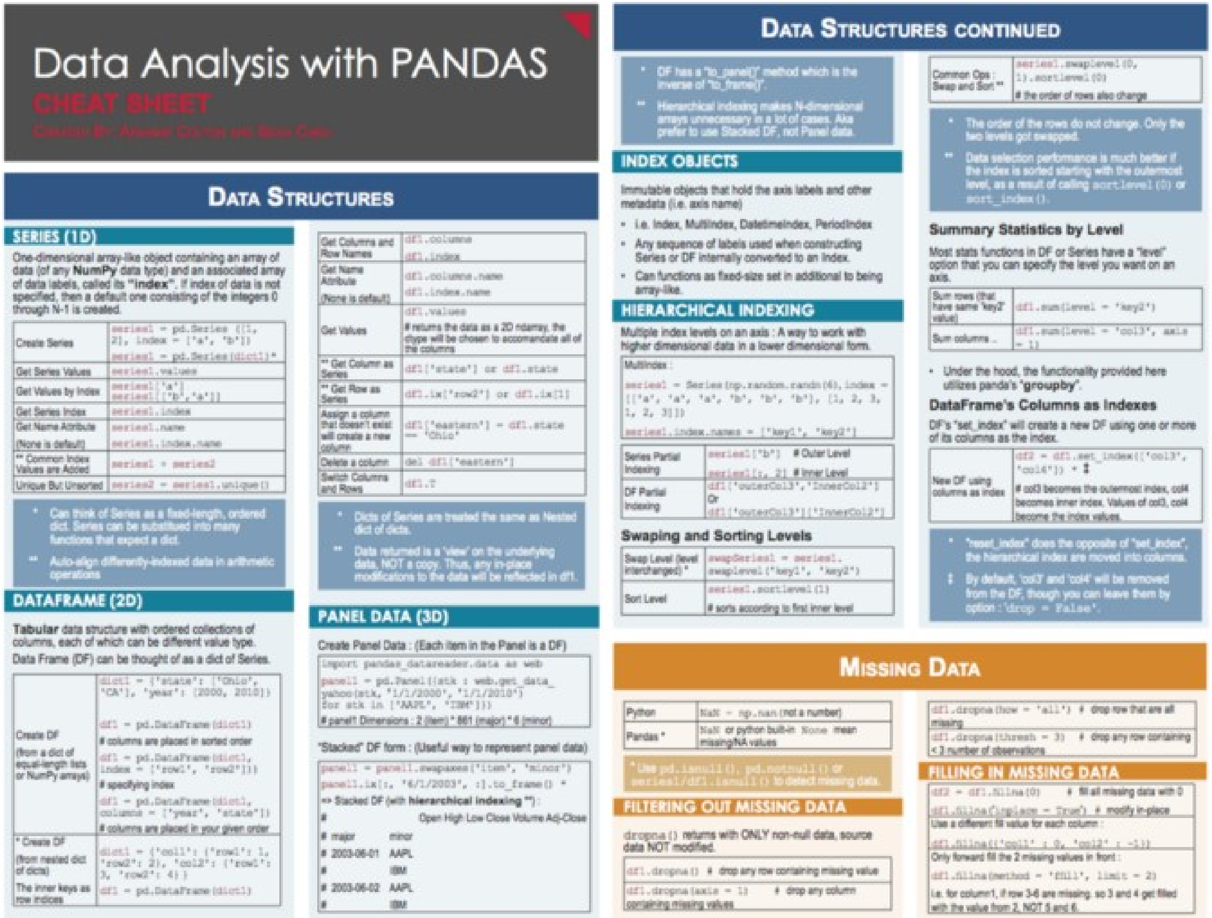
```
[1 2 3]
(3,)
int64
```

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

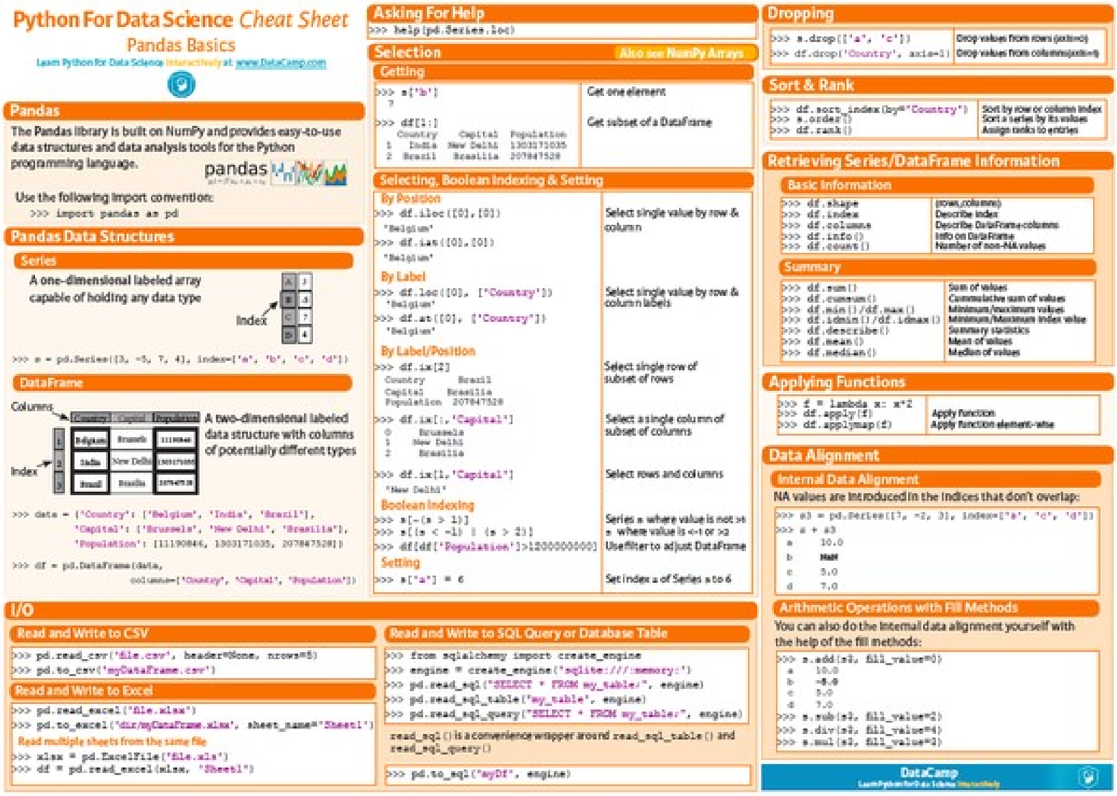
```
[[0 2 4]
 [1 3 5]]
(2, 3)
int64
```

Pandas

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



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', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

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

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']                 Get one element
  -5
>>> df[1:]                 Get subset of a DataFrame
     Country    Capital   Population
  1  India    New Delhi   1303171035
  2  Brazil   Brasília    207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]       Select single value by row & column
  'Belgium'
>>> df.iat([0],[0])
  'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']]   Select single value by row & column labels
  'Belgium'
>>> df.at([0], ['Country'])
  'Belgium'
```

By Label/Position

```
>>> df.ix[2]                   Select single row of subset of rows
  Country      Brazil
  Capital      Brasília
  Population   207847528
>>> df.ix[:,'Capital']         Select a single column of subset of columns
  0    Brussels
  1    New Delhi
  2    Brasília
>>> df.ix[1,'Capital']         Select rows and columns
  'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)]                           Series s where value is not >1
>>> s[(s < -1) | (s > 2)]                 s where value is <-1 or >2
>>> df[df['Population']>1200000000]       Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6                            Set index a of Series s to 6
```

Dropping

```
>>> s.drop(['a', 'c'])                 Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)         Drop values from columns(axis=1)
```

Sort & Rank

```
>>> df.sort_index(by="Country")   Sort by row or column index
>>> s.order()                     Sort a series by its values
>>> df.rank()                     Assign ranks to entries
```

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape         (rows,columns)
>>> df.index         Describe index
>>> df.columns       Describe DataFrame columns
>>> df.info()        Info on DataFrame
>>> df.count()       Number of non-NA values
```

Summary

```
>>> df.sum()                          Sum of values
>>> df.cumsum()                       Cummulative sum of values
>>> df.min()/df.max()                 Minimum/maximum values
>>> df.idxmin()/df.idxmax()           Minimum/Maximum index value
>>> df.describe()                     Summary statistics
>>> df.mean()                         Mean of values
>>> df.median()                       Median of values
```

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)          Apply function
>>> df.applymap(f)       Apply function element-wise
```

Data Alignment

Internal Data Alignment

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

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a    10.0
  b     NaN
  c     5.0
  d     7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c     5.0
  d     7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

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

Read and Write to Excel

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

Read multiple sheets from the same file

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

Read and Write to SQL Query or Database Table

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

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

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

DataCamp

Learn Python for Data Science Interactively

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.

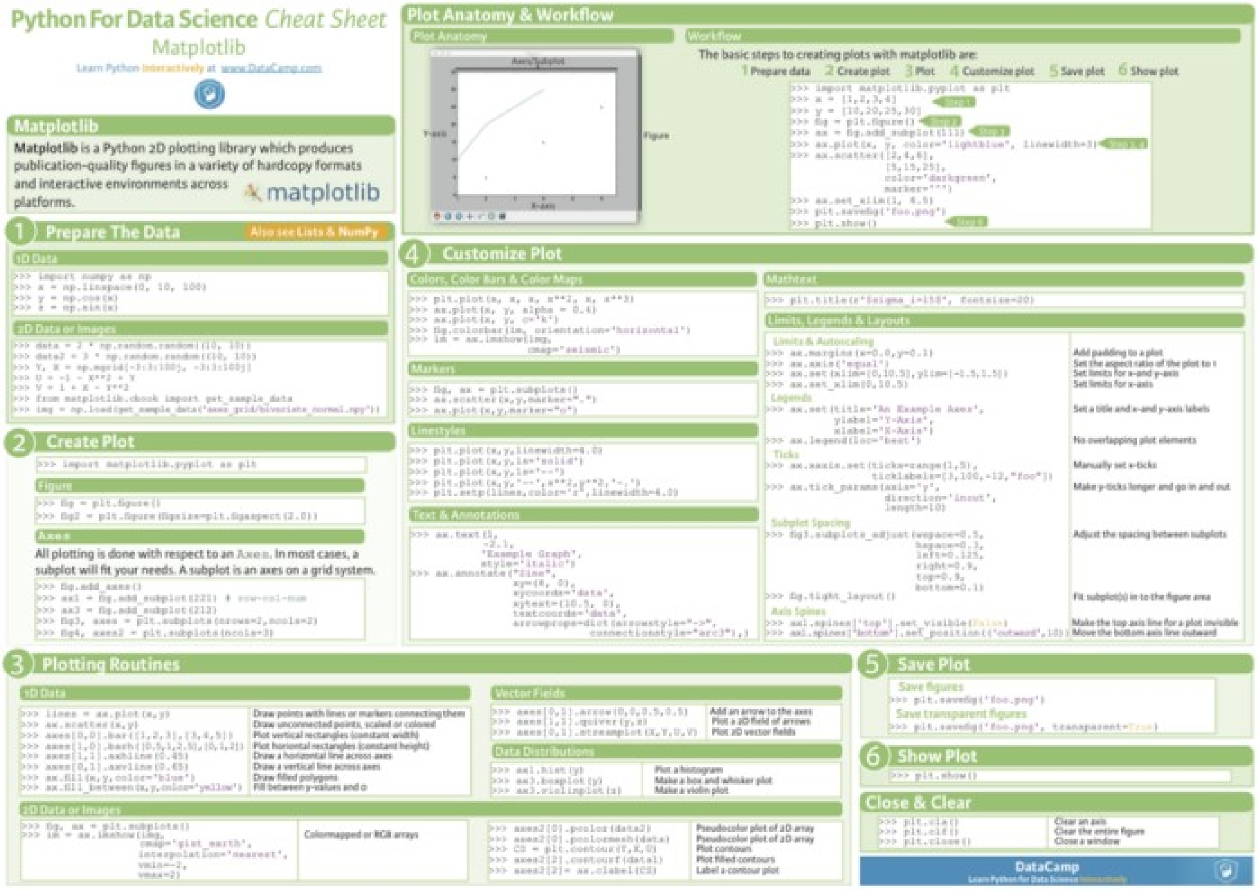
Create a Series:

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



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