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

medium.com/machine-learning-in-practice/cheat-sheet-of-machine-learning-and-python-and-math-cheat-sheets-a4afe4e791b6

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Given how rapidly the Machine Learning space is evolving, I imagine these will go out of date quickly, but at least as of June 1, 2017, they are pretty current.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

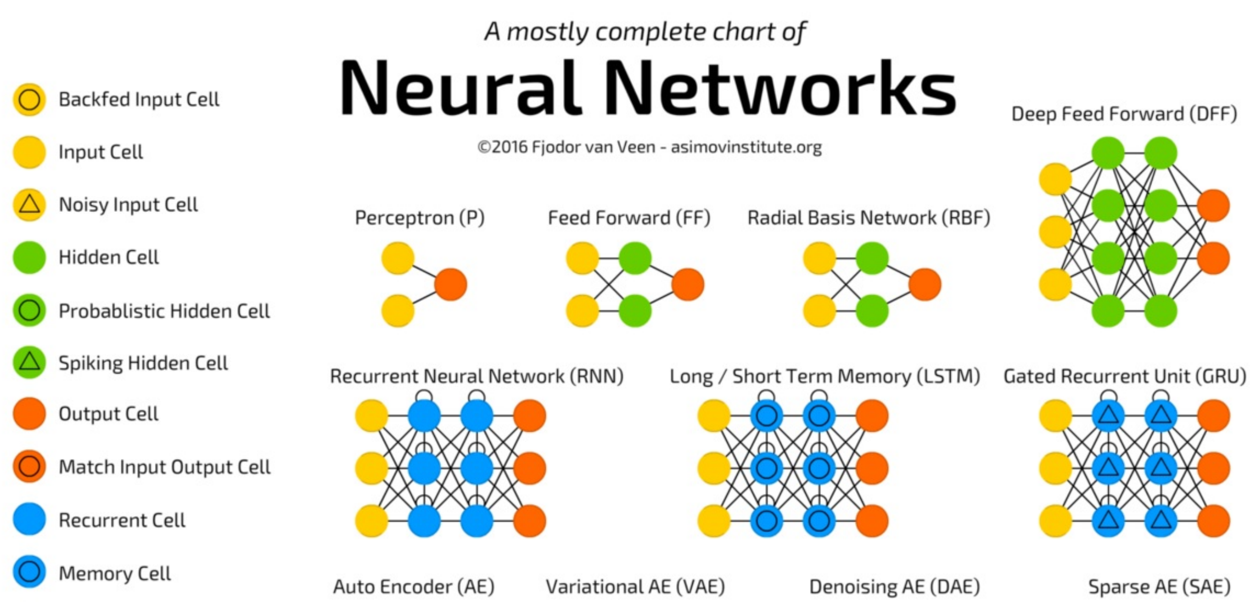
If you like this post, give it a ❤ below.

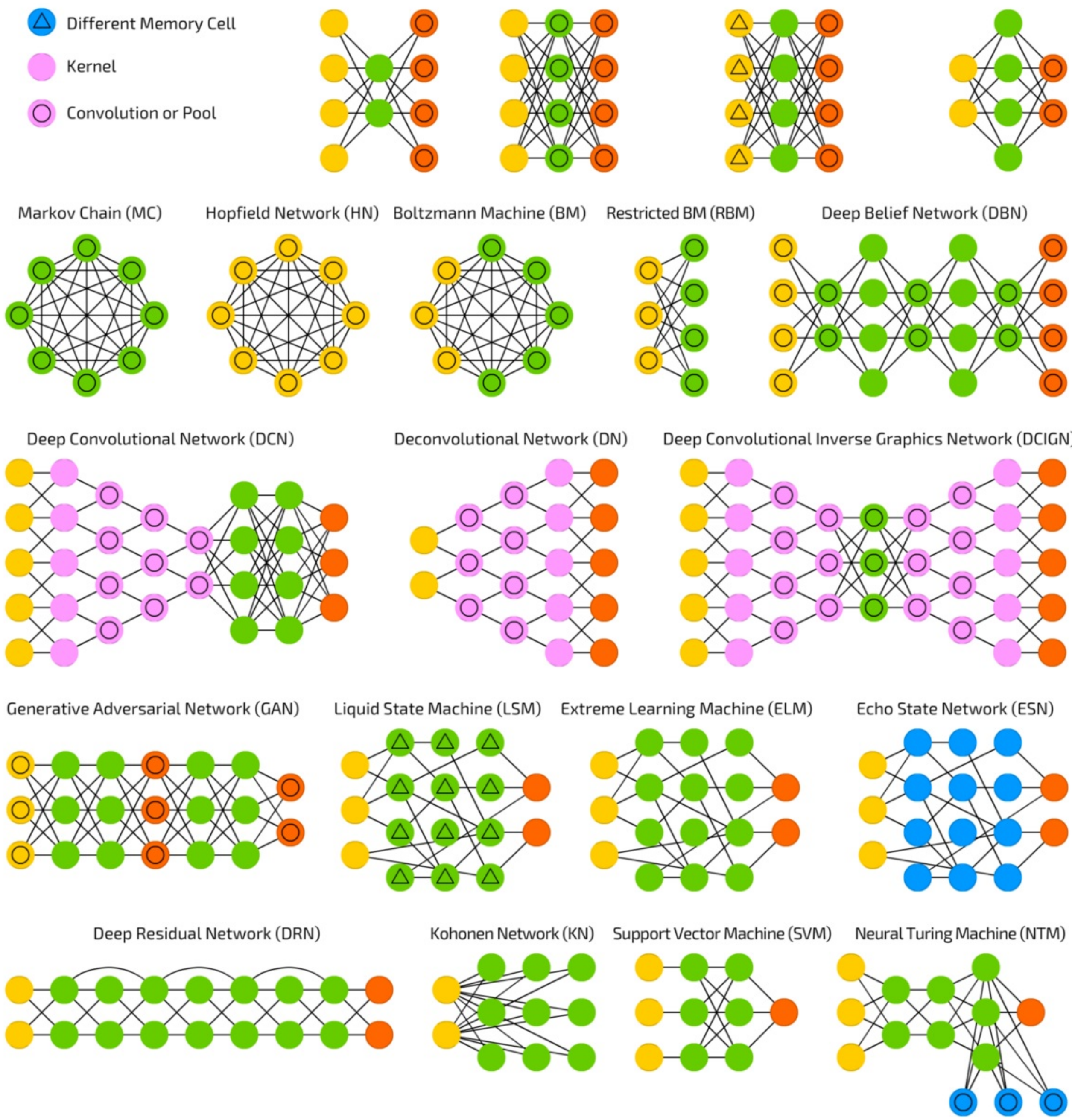
Machine Learning

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/

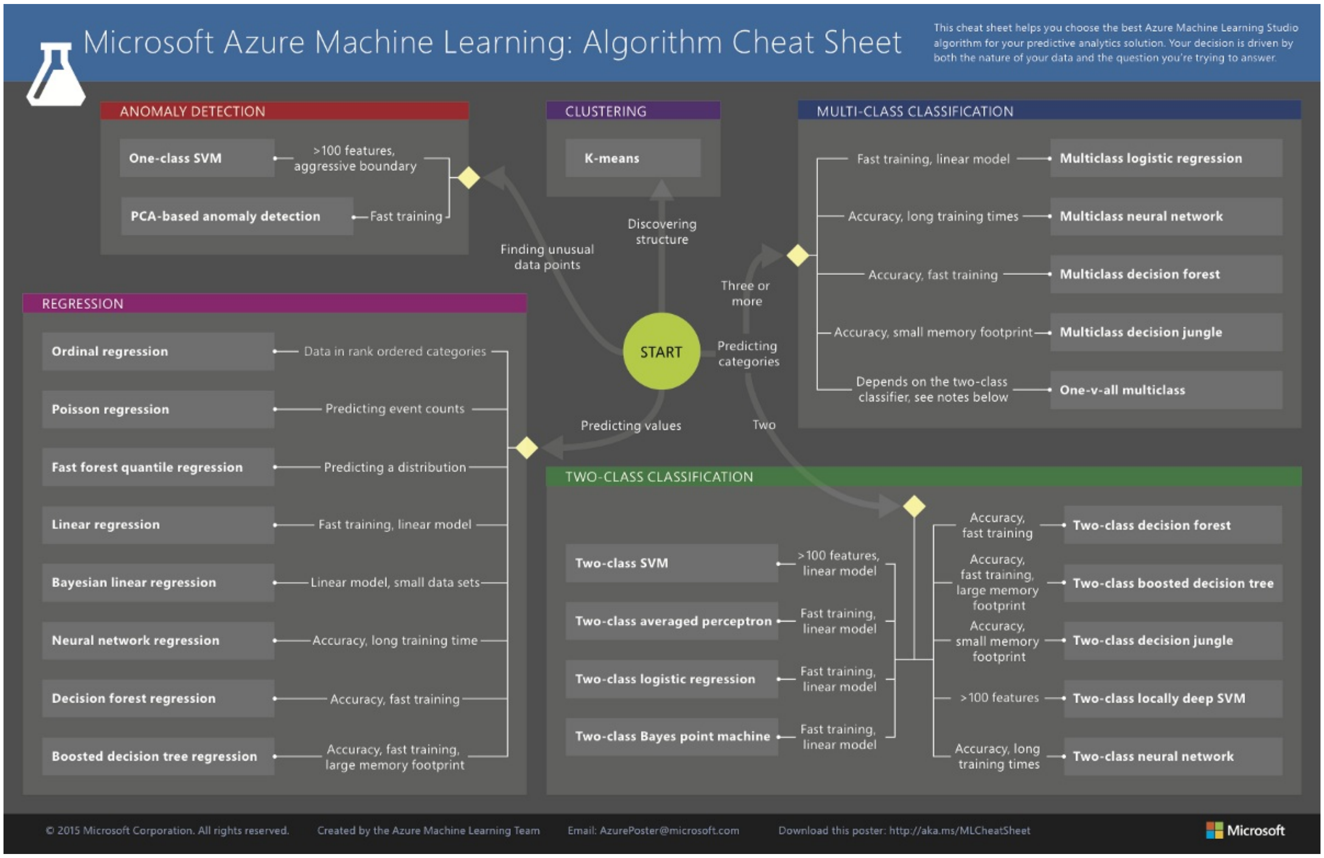




The Neural Network Zoo

Microsoft Azure Algorithm Flowchart

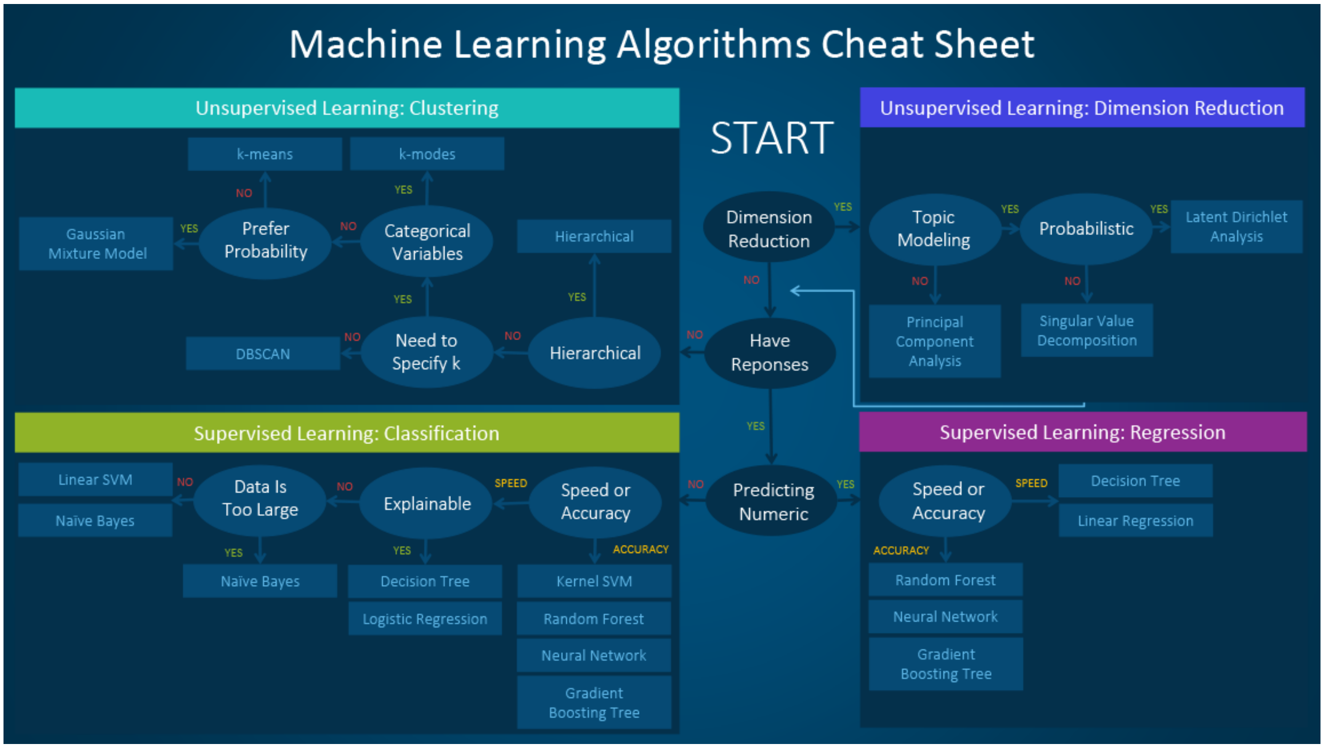
Source: https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-cheat-sheet



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/



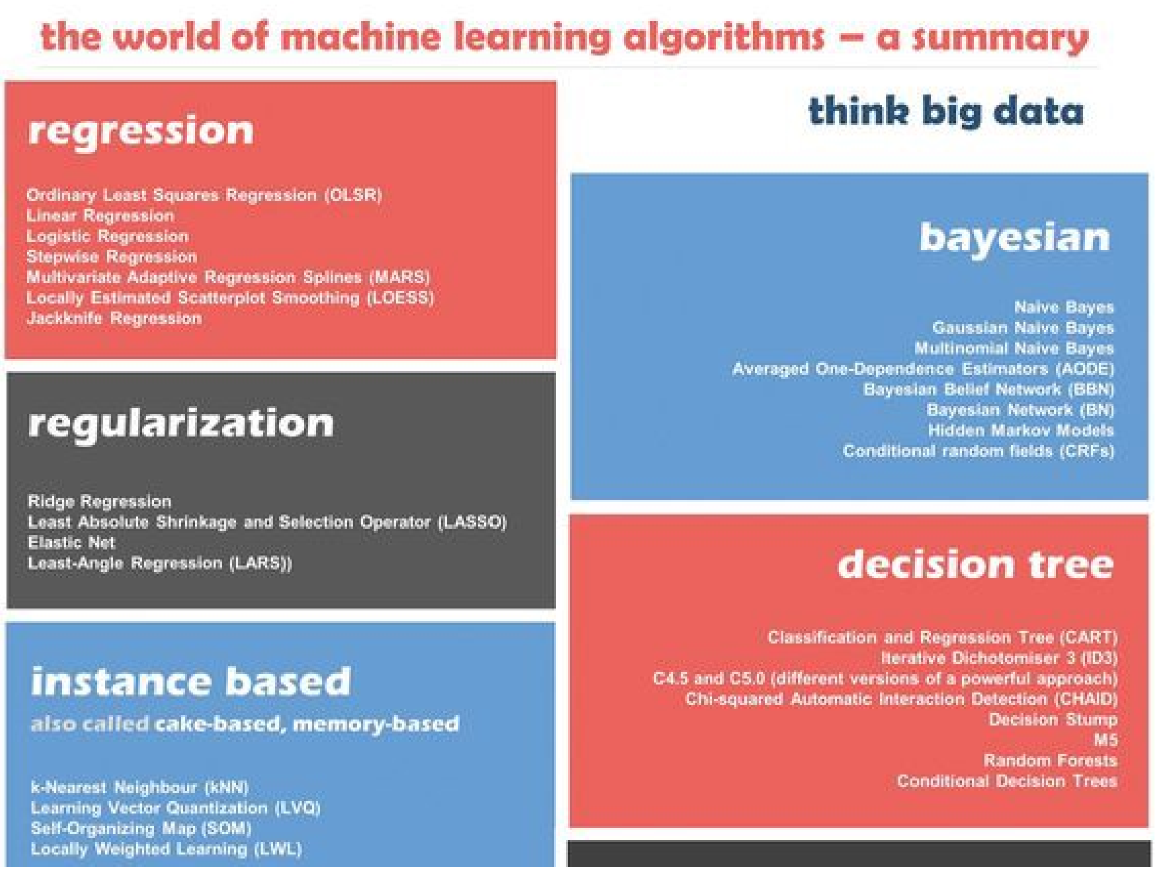
SAS: Which machine learning algorithm should I use?

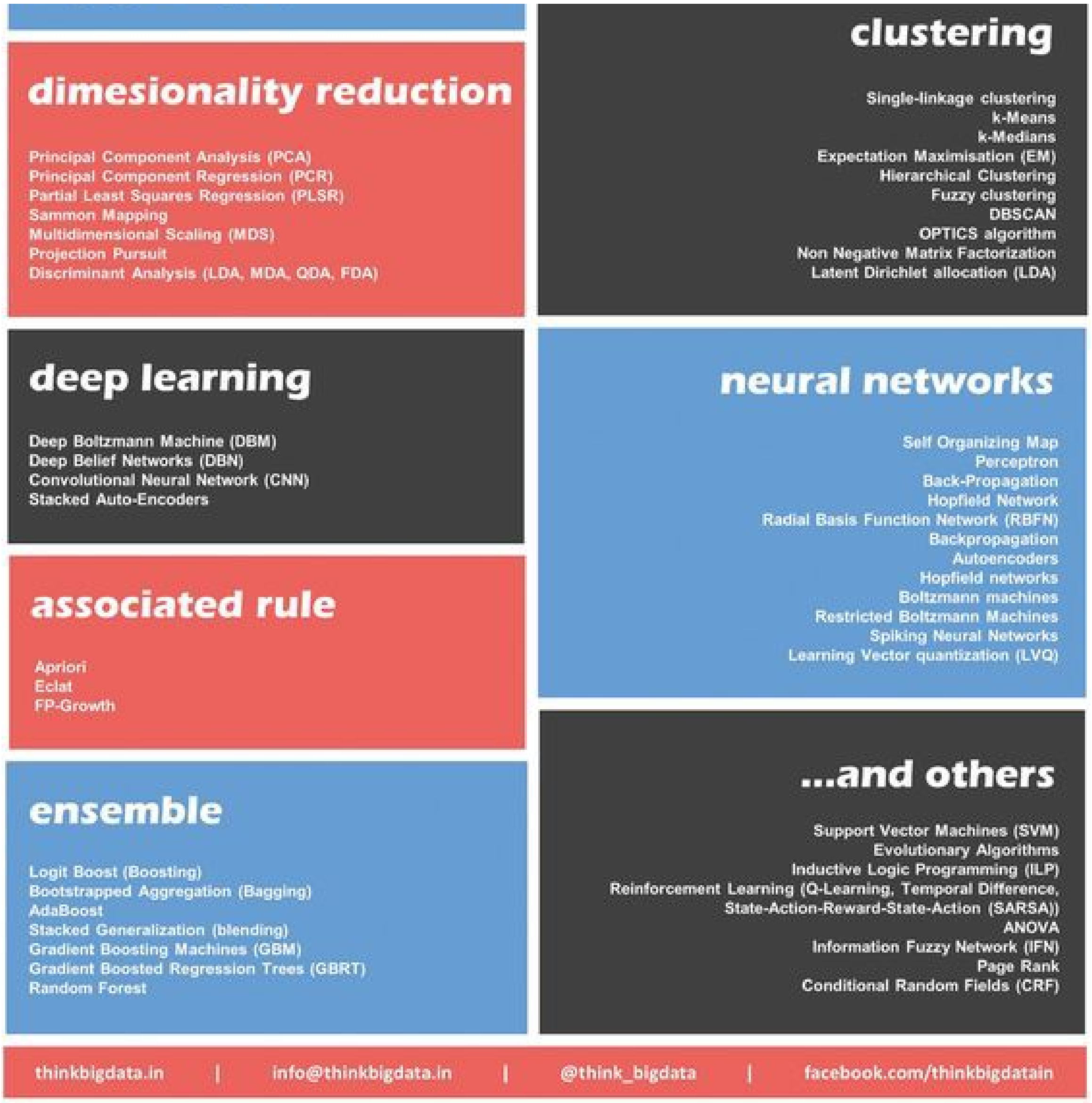
Algorithm Summary

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/





Which are the best known machine learning algorithms?

Algorithm Pro/Con

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

TOP PREDICTION ALGORITHMS

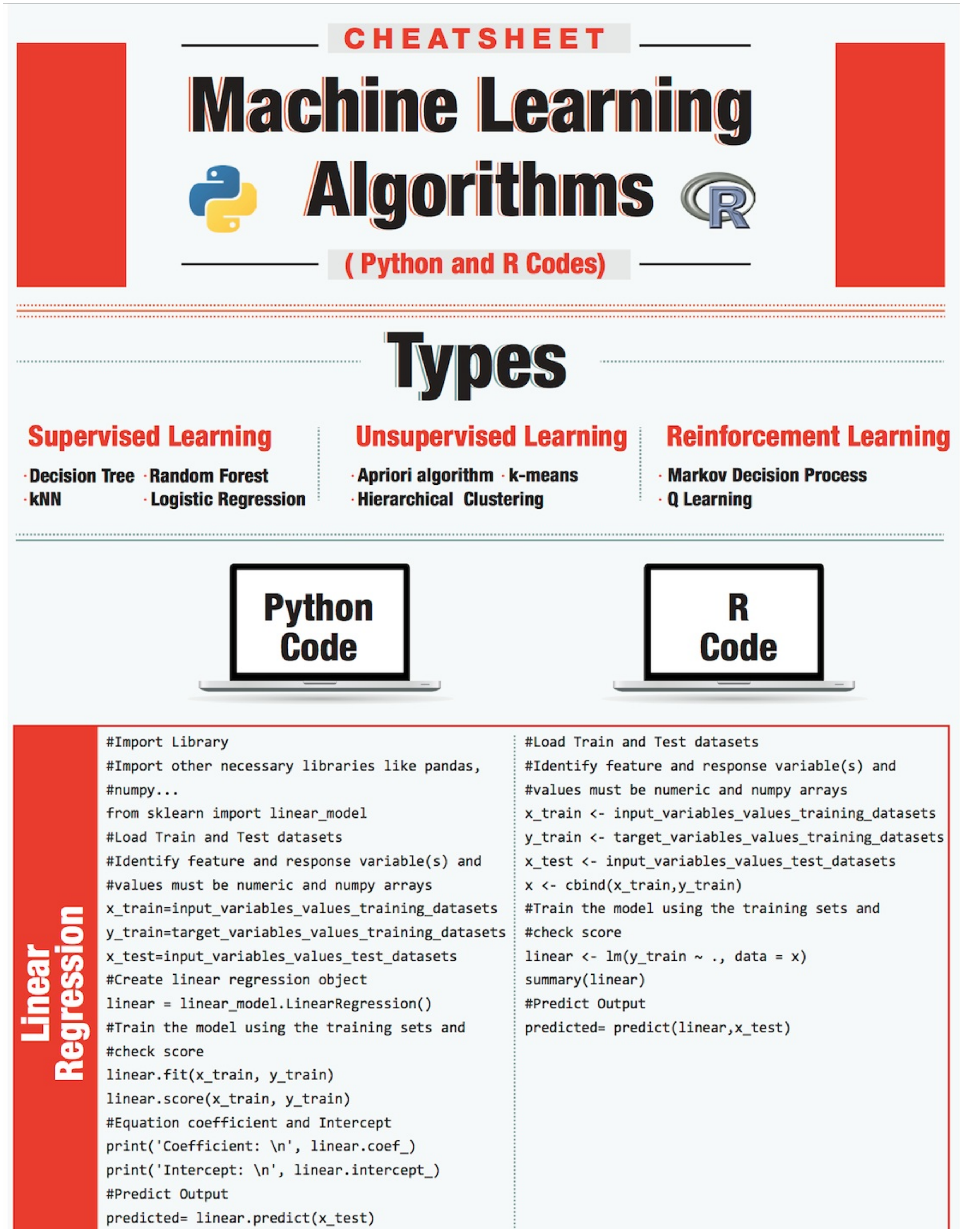
TYPE	NAME	DESCRIPTION	ADVANTAGES	DISADVANTAGES
Linear	Linear regression	**The "best fit" line** through all data points. Predictions are numerical.	Easy to understand -- you clearly see what the biggest drivers of the model are.	**X** Sometimes too simple to capture complex relationships between variables. **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/



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

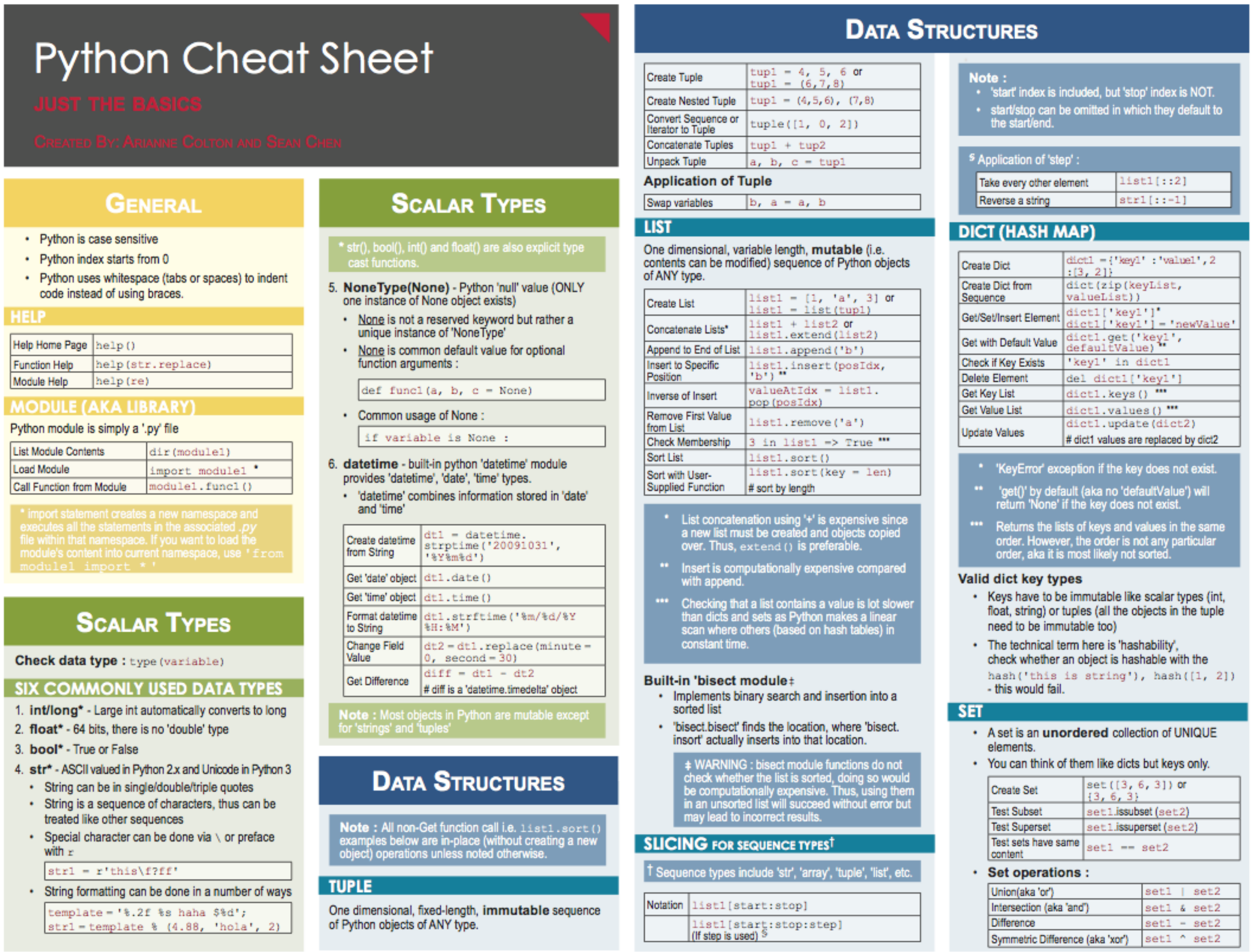
Linear Regression

```python
#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)
```

```r
#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: ARIANNE COLTON AND SEAN CHEN

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	`help()`
Function Help	`help(str.replace)`
Module Help	`help(re)`

MODULE (AKA LIBRARY)

Python module is simply a '.py' file

List Module Contents	`dir(module1)`
Load Module	`import module1` *
Call Function from Module	`module1.func1()`

* 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 module1 import * '

SCALAR TYPES

Check data type : `type(variable)`

SIX COMMONLY USED DATA TYPES

1. **int/long*** - Large int automatically converts to long
2. **float*** - 64 bits, there is no 'double' type
3. **bool*** - True or False
4. **str*** - ASCII valued in Python 2.x 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

   ```
   str1 = r'this\f?ff'
   ```

   - String formatting can be done in a number of ways

   ```
   template = '%.2f %s haha $%d';
   str1 = template % (4.88, 'hola', 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)
   ```

   - Common usage of 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	`dt1 = datetime.strptime('20091031', '%Y%m%d')`
Get 'date' object	`dt1.date()`
Get 'time' object	`dt1.time()`
Format datetime to String	`dt1.strftime('%m/%d/%Y %H:%M')`
Change Field Value	`dt2 = dt1.replace(minute = 0, second = 30)`
Get Difference	`diff = dt1 - dt2` # 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. `list1.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	`tup1 = 4, 5, 6` or `tup1 = (6,7,8)`
Create Nested Tuple	`tup1 = (4,5,6), (7,8)`
Convert Sequence or Iterator to Tuple	`tuple([1, 0, 2])`
Concatenate Tuples	`tup1 + tup2`
Unpack Tuple	`a, b, c = tup1`

Application of Tuple

Swap variables	`b, a = a, b`

LIST

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

Create List	`list1 = [1, 'a', 3]` or `list1 = list(tup1)`
Concatenate Lists*	`list1 + list2` or `list1.extend(list2)`
Append to End of List	`list1.append('b')`
Insert to Specific Position	`list1.insert(posIdx, 'b')` **
Inverse of Insert	`valueAtIdx = list1.pop(posIdx)`
Remove First Value from List	`list1.remove('a')`
Check Membership	`3 in list1 => True` ***
Sort List	`list1.sort()`
Sort with User-Supplied Function	`list1.sort(key = len)` # 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.insort' 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	`list1[start:stop]`
	`list1[start:stop:step]` (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	`list1[::2]`
Reverse a string	`str1[::-1]`

DICT (HASH MAP)

Create Dict	`dict1 ={'key1' :'value1',2 :[3, 2]}`
Create Dict from Sequence	`dict(zip(keyList, valueList))`
Get/Set/Insert Element	`dict1['key1']`* `dict1['key1'] = 'newValue'`
Get with Default Value	`dict1.get('key1', defaultValue)` **
Check if Key Exists	`'key1' in dict1`
Delete Element	`del dict1['key1']`
Get Key List	`dict1.keys()` ***
Get Value List	`dict1.values()` ***
Update Values	`dict1.update(dict2)` # dict1 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

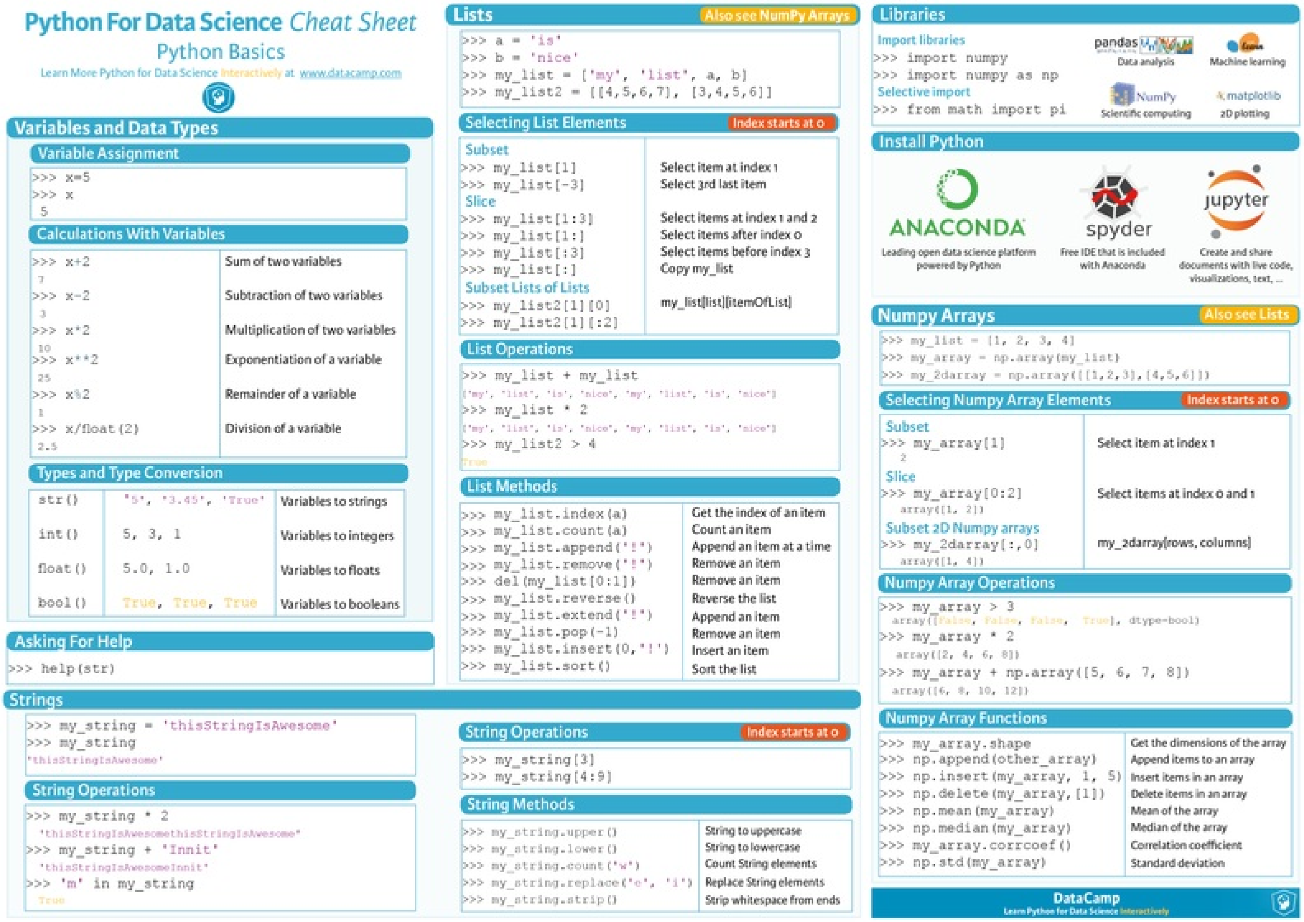
- A set is an **unordered** collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set	`set([3, 6, 3])` or `{3, 6, 3}`
Test Subset	`set1.issubset(set2)`
Test Superset	`set1.issuperset(set2)`
Test sets have same content	`set1 == set2`

- **Set operations :**

Union(aka 'or')	`set1 | set2`
Intersection (aka 'and')	`set1 & set2`
Difference	`set1 - set2`
Symmetric Difference (aka 'xor')	`set1 ^ set2`

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)
```

Strings

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

String Operations

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

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_list[list][itemOfList]
`>>> my_list2[1][:2]`	

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

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; 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 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

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

Numpy Cheat Sheet

Python Package

Created By: Arianne Colton and 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	`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

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

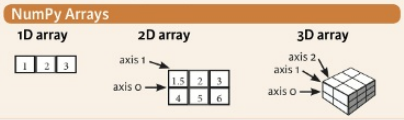
NumPy

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



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`
`array([[-0.5, 0. , 0. ],`
`[-3. , -3. , -3. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5, 4. , 6. ],`
`[ 5. , 7. , 9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667, 1. , 1. ],`
`[ 0.25 , 0.4 , 0.5 ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[ 1.5, 4. , 9. ],`
`[ 4. , 10. , 18. ]])`
	Multiplication

`>>> 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)`
`array([[ 7., 7.],`
`[ 7., 7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False, True, True],`
`[False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2` `array([True, False, False], dtype=bool)`	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
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting	
`>>> a[2]` `3`	Select the element at the 2nd index
`>>> b[1,2]` `6.0`	Select the element at row 0 column 2 (equivalent to `b[1][2]`)
Slicing	
`>>> a[0:2]` `array([1, 2])`	Select items at index 0 and 1
`>>> b[0:2,1]` `array([ 2., 5.])`	Select items at rows 0 and 1 in column 1
`>>> b[:1]` `array([[1.5, 2., 3.]])`	Select all items at row 0 (equivalent to `b[0:1, :]`)

`>>> c[1,...]`
`array([[[ 3., 2., 1.],`
`[ 4., 5., 6.]]])`
	Same as `[1,:,:]`

`>>> a[ : :-1]` `array([3, 2, 1])`	Reversed array `a`
Boolean Indexing	
`>>> a[a<2]` `array([1])`	Select elements from `a` less than 2
Fancy Indexing	
`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]` `array([ 4. , 2. , 6. , 1.5])`	Select elements `(1,0)`,`(0,1)`,`(1,2)` and `(0,0)`

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

Array Manipulation

Transposing Array	
`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions
Changing Array Shape	
`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data
Adding/Removing Elements	
`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array
Combining Arrays	
`>>> np.concatenate((a,d),axis=0)` `array([ 1, 2, 3, 10, 15, 20])`	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. , 2. , 3. ],`
`[ 1.5, 2. , 3. ],`
`[ 4. , 5. , 6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7., 7., 1., 0.],`
`[ 7., 7., 0., 1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`[ 2, 15],`
`[ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays
Splitting Arrays	
`>>> np.hsplit(a,3)` `[array([1]),array([2]),array([3])]`	Split the array horizontally at the 3rd index

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

DataCamp
Learn Python for Data Science Interactively

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: ARIANNE COLTON AND SEAN CHEN

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	`series1 = pd.Series ([1, 2], index = ['a', 'b'])` `series1 = pd.Series(dict1)*`
Get Series Values	`series1.values`
Get Values by Index	`series1['a']` `series1[['b','a']]`
Get Series Index	`series1.index`
Get Name Attribute (None is default)	`series1.name` `series1.index.name`
** Common Index Values are Added	`series1 + series2`
Unique But Unsorted	`series2 = series1.unique()`

* Can think of Series as a fixed-length, ordered dict. Series can be substitued into many functions that expect a dict.

** Auto-align differently-indexed data in arithmetic operations

Get Columns and Row Names	`df1.columns` `df1.index`
Get Name Attribute (None is default)	`df1.columns.name` `df1.index.name`
Get Values	`df1.values` # returns the data as a 2D ndarray, the dtype will be chosen to accomandate all of the columns
** Get Column as Series	`df1['state']` or `df1.state`
** Get Row as Series	`df1.ix['row2']` or `df1.ix[1]`
Assign a column that doesn't exist will create a new column	`df1['eastern'] = df1.state == 'Ohio'`
Delete a column	`del df1['eastern']`
Switch Columns and Rows	`df1.T`

* Dicts of Series are treated the same as Nested dict of dicts.

** Data returned is a 'view' on the underlying data, NOT a copy. Thus, any in-place modificatons to the data will be reflected in df1.

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)	`dict1 = {'state': ['Ohio', 'CA'], 'year': [2000, 2010]}`
	`df1 = pd.DataFrame(dict1)`
	# columns are placed in sorted order
	`df1 = pd.DataFrame(dict1, index = ['row1', 'row2']))`
	# specifying index
	`df1 = pd.DataFrame(dict1, columns = ['year', 'state'])`
	# columns are placed in your given order

* Create DF (from nested dict of dicts) The inner keys as row indices	`dict1 = {'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 3, 'row2': 4} }` `df1 = pd.DataFrame(dict1)`

PANEL DATA (3D)

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

```
import pandas_datareader.data as web
panel1 = pd.Panel({stk : web.get_data_
yahoo(stk, '1/1/2000', '1/1/2010')
for stk in ['AAPL', 'IBM']})
# panel1 Dimensions : 2 (item) * 861 (major) * 6 (minor)
```

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

```
panel1 = panel1.swapaxes('item', 'minor')
panel1.ix[:, '6/1/2003', :].to_frame() *
=> Stacked DF (with hierarchical indexing **) :
#                    Open High Low Close Volume Adj-Close
# major      minor
# 2003-06-01 AAPL
#            IBM
# 2003-06-02 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. Aka 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 functions as fixed-size set in additional 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[:, 2]` # Inner Level

DF Partial Indexing	`df1['outerCol3','InnerCol2']`
	Or
	`df1['outerCol3']['InnerCol2']`

Swaping and Sorting Levels

Swap Level (level interchanged) *	`swapSeries1 = series1.swaplevel('key1', 'key2')`
Sort Level	`series1.sortlevel(1)` # sorts according to first inner level

Common Ops : Swap and Sort **	`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)	`df1.sum(level = 'key2')`
Sum columns ..	`df1.sum(level = 'col3', axis = 1)`

- Under the hood, the functionality provided here utilizes panda'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	`df2 = df1.set_index(['col3', 'col4']) * ‡` # 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	`NaN = np.nan` (not a number)
Pandas *	`NaN` or python built-in `None` mean missing/NA values

* Use `pd.isnull()`, `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.

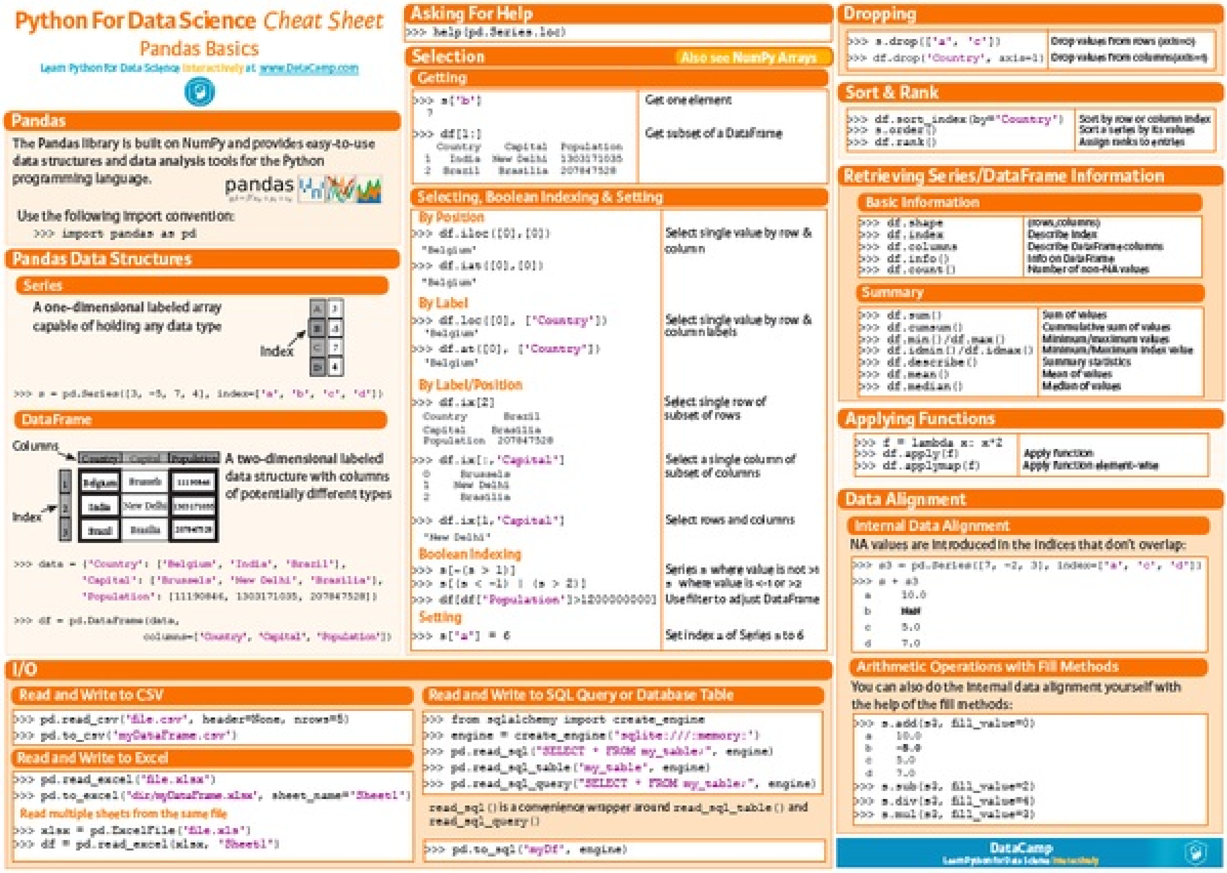
`df1.dropna()` # drop any row containing missing value
`df1.dropna(axis = 1)` # drop any column containing missing values

`df1.dropna(how = 'all')` # drop row that are all missing
`df1.dropna(thresh = 3)` # 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 3-6 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', '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.idmin()/df.idmax()         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

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)
```

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,
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

Colormapped or RGB arrays

`>>> 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

Vector Fields

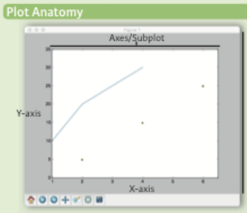
`>>> axes[0,1].arrow(0,0,0.5,0.5)`	Add an arrow to the axes
`>>> axes[1,1].quiver(y,z)`	Plot a 2D field of arrows
`>>> axes[0,1].streamplot(X,Y,U,V)`	Plot 2D vector fields

Data Distributions

`>>> ax1.hist(y)`	Plot a histogram
`>>> ax3.boxplot(y)`	Make a box and whisker plot
`>>> ax3.violinplot(z)`	Make a violin plot

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',
           ylabel='Y-Axis',
           xlabel='X-Axis')
```

Set a title and x-and y-axis labels

`>>> ax.legend(loc='best')`	No overlapping plot elements

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
```

Manually set x-ticks

```
>>> ax.tick_params(axis='y',
                   direction='inout',
                   length=10)
```

Make y-ticks longer and go in and out

Subplot Spacing

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

Adjust the spacing between subplots

`>>> 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

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

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

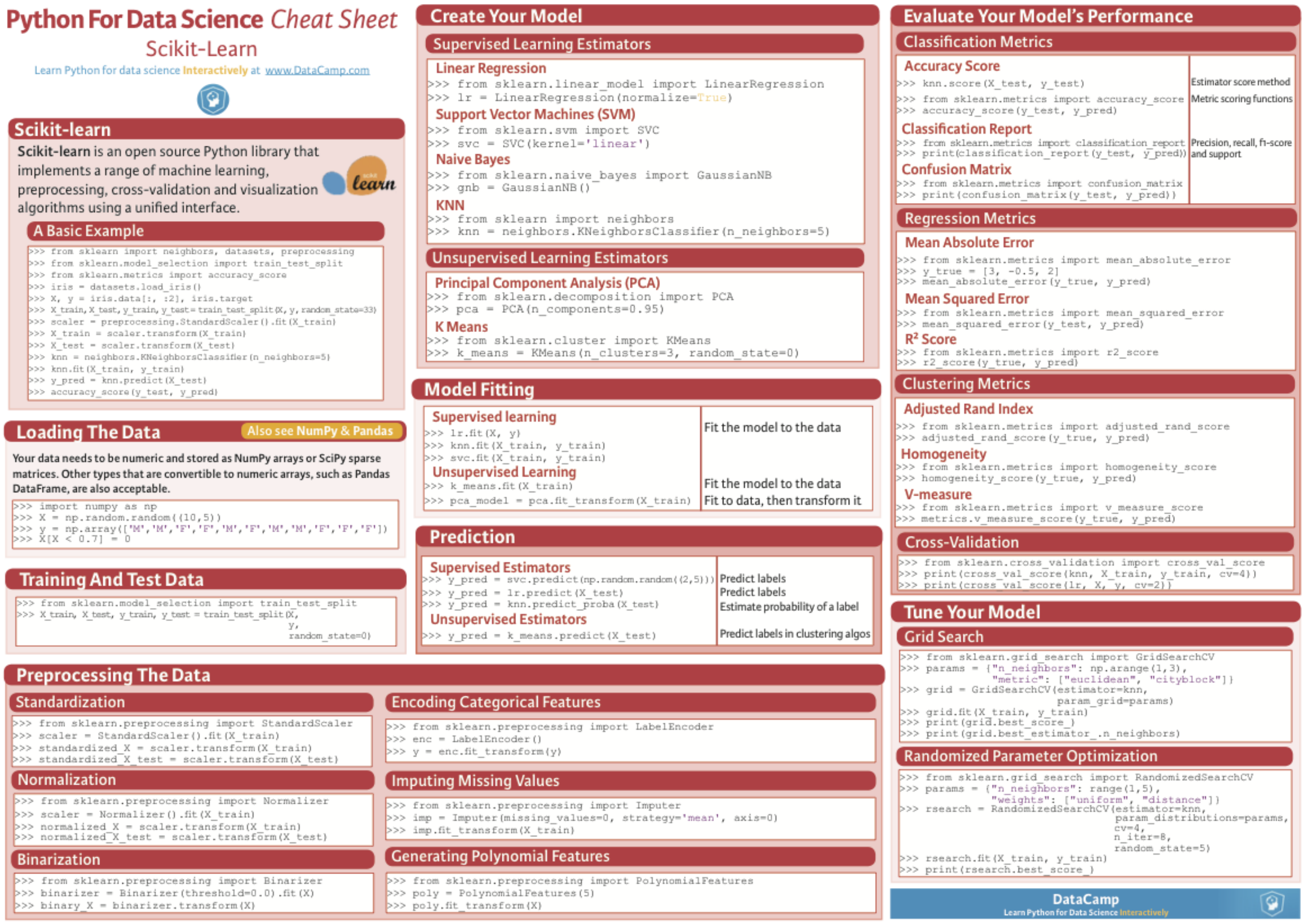
In [3]:
```
x = np.linspace(0, 2, 10)

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



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)
```

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

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)
```

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)
```

R^2 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

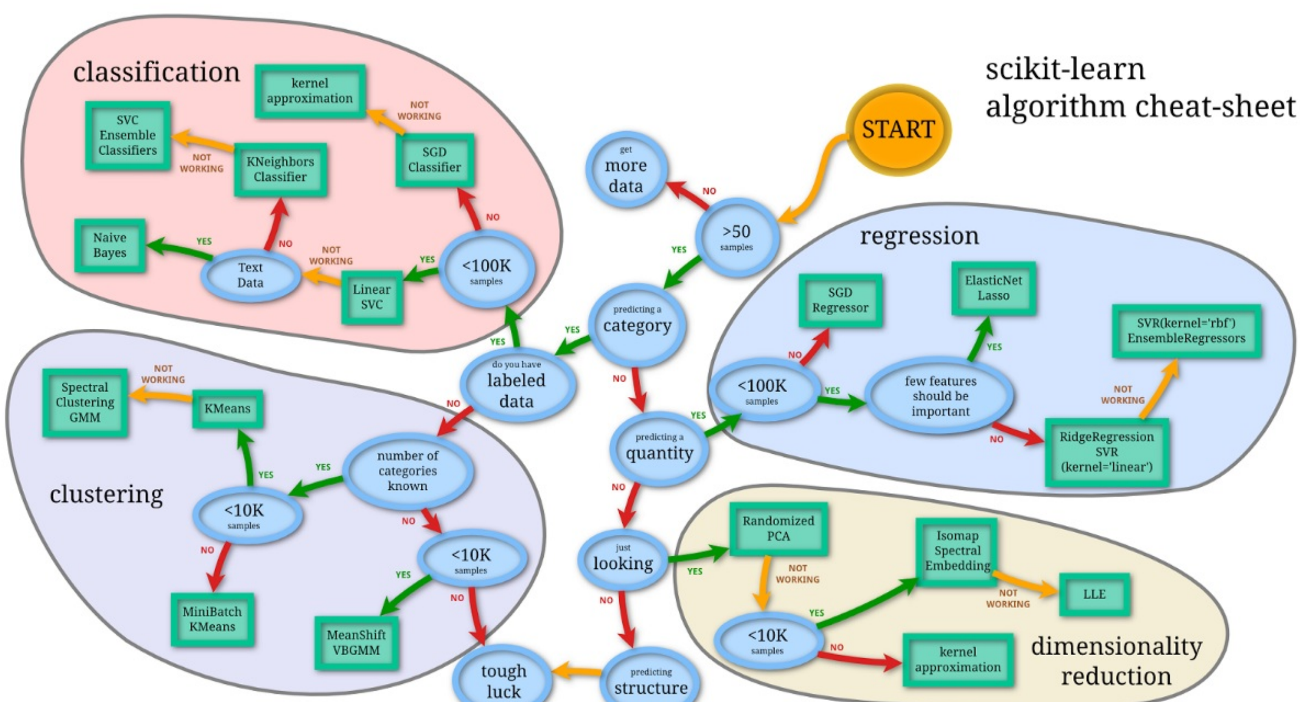
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

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_)
```

DataCamp
Learn Python for Data Science Interactively

Source: http://peekaboo-vision.blogspot.de/2013/01/machine-learning-cheat-sheet-for-scikit.html



Source:
https://github.com/rcompton/ml_cheat_sheet/blob/master/supervised_learning.ipynb

```
In [1]: import sklearn
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import sklearn.datasets
        %pylab inline

        #sklearn two moons generator makes lots of these...
        import warnings
        warnings.filterwarnings("ignore", category=DeprecationWarning)

        Populating the interactive namespace from numpy and matplotlib

In [2]: """
        Build some datasets that I'll demo the models on
        """
        Xs = []
        ys = []

        #low noise, plenty of samples, should be easy
        X0, y0 = sklearn.datasets.make_moons(n_samples=1000, noise=.05)
        Xs.append(X0)
        ys.append(y0)

        #more noise, plenty of samples
        X1, y1 = sklearn.datasets.make_moons(n_samples=1000, noise=.3)
        Xs.append(X1)
        ys.append(y1)

        #less noise, few samples
        X2, y2 = sklearn.datasets.make_moons(n_samples=200, noise=.05)
        Xs.append(X2)
        ys.append(y2)

        #more noise, less samples, should be hard
        X3, y3 = sklearn.datasets.make_moons(n_samples=200, noise=.3)
        Xs.append(X3)
```

Tensorflow

Source: https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/1_Introduction/basic_operations.ipynb

In [1]:
```
# Basic Operations example using TensorFlow library.
# Author: Aymeric Damien
# Project: https://github.com/aymericdamien/TensorFlow-Examples/
```

In [2]:
```
import tensorflow as tf
```

In [3]:
```
# Basic constant operations
# The value returned by the constructor represents the output
# of the Constant op.
a = tf.constant(2)
b = tf.constant(3)
```

In [4]:
```
# Launch the default graph.
with tf.Session() as sess:
    print "a: %i" % sess.run(a), "b: %i" % sess.run(b)
    print "Addition with constants: %i" % sess.run(a+b)
    print "Multiplication with constants: %i" % sess.run(a*b)
```

```
a=2, b=3
Addition with constants: 5
Multiplication with constants: 6
```

In [5]:
```
# Basic Operations with variable as graph input
# The value returned by the constructor represents the output
# of the Variable op. (define as input when running session)
# tf Graph input
a = tf.placeholder(tf.int16)
b = tf.placeholder(tf.int16)
```

In [6]:
```
# Define some operations
add = tf.add(a, b)
mul = tf.multiply(a, b)
```

Pytorch

Source: https://github.com/bfortuner/pytorch-cheatsheet

Pytorch Cheatsheet

Imports

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.init as init
import torch.optim as optim
import torch.nn.functional as F
from torch.autograd import Variable
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torchvision.utils as tv_utils
from torch.utils.data import DataLoader
import torchvision.models as models
import torch.backends.cudnn as cudnn
import torchvision
import torch.autograd as autograd
from PIL import Image
import imp
import os
import sys
import math
import time
import random
import shutil
import cv2
import scipy.misc
from glob import glob
import sklearn
import logging

from tqdm import tqdm
import numpy as np
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
plt.style.use('bmh')

%matplotlib inline
```

Basics

- http://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- http://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Datasets

File Management

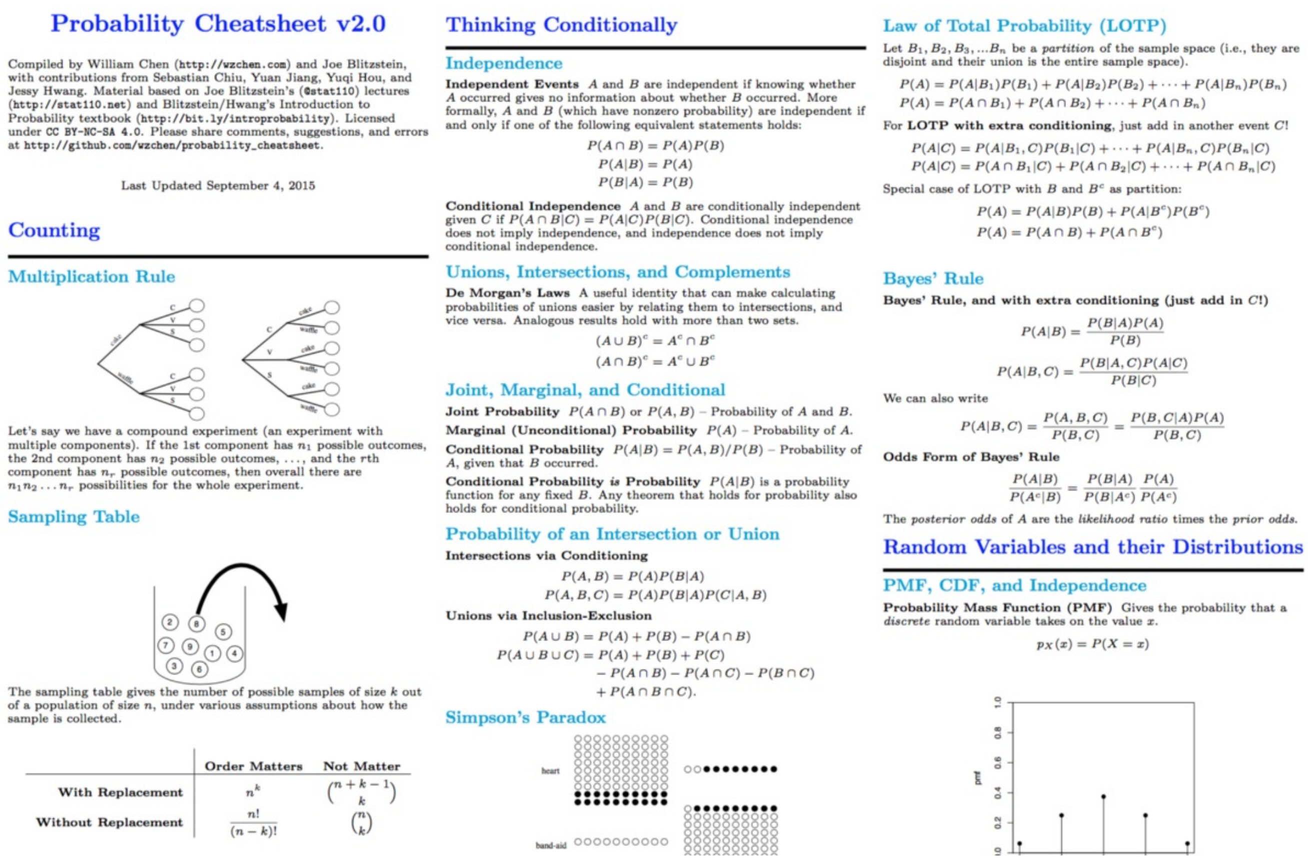
In [ ]:
```
random.seed(1)
torch.manual_seed(1)
DATA_PATH='/media/bfortuner/bigguy/data/'
```

Math

If you really want to understand Machine Learning, you need a solid understanding of Statistics (especially Probability), Linear Algebra, and some Calculus. I minored in Math during undergrad, but I definitely needed a refresher. These cheat sheets provide most of what you need to understand the Math behind the most common Machine Learning algorithms.

Probability

Source: http://www.wzchen.com/s/probability_cheatsheet.pdf



Probability Cheatsheet v2.0

Compiled by William Chen (http://wzchen.com) and Joe Blitzstein, with contributions from Sebastian Chiu, Yuan Jiang, Yuqi Hou, and Jessy Hwang. Material based on Joe Blitzstein's (@stat110) lectures (http://stat110.net) and Blitzstein/Hwang's Introduction to Probability textbook (http://bit.ly/introprobability). Licensed under CC BY-NC-SA 4.0. Please share comments, suggestions, and errors at http://github.com/wzchen/probability_cheatsheet.

Last Updated September 4, 2015

Counting

Multiplication Rule

Let's say we have a compound experiment (an experiment with multiple components). If the 1st component has n_1 possible outcomes, the 2nd component has n_2 possible outcomes, …, and the rth component has n_r possible outcomes, then overall there are $n_1 n_2 \dots n_r$ possibilities for the whole experiment.

Sampling Table

The sampling table gives the number of possible samples of size k out of a population of size n, under various assumptions about how the sample is collected.

	Order Matters	Not Matter
With Replacement	n^k	$\binom{n+k-1}{k}$
Without Replacement	$\frac{n!}{(n-k)!}$	$\binom{n}{k}$

Thinking Conditionally

Independence

Independent Events A and B are independent if knowing whether A occurred gives no information about whether B occurred. More formally, A and B (which have nonzero probability) are independent if and only if one of the following equivalent statements holds:

$$P(A \cap B) = P(A)P(B)$$
$$P(A|B) = P(A)$$
$$P(B|A) = P(B)$$

Conditional Independence A and B are conditionally independent given C if $P(A \cap B|C) = P(A|C)P(B|C)$. Conditional independence does not imply independence, and independence does not imply conditional independence.

Unions, Intersections, and Complements

De Morgan's Laws A useful identity that can make calculating probabilities of unions easier by relating them to intersections, and vice versa. Analogous results hold with more than two sets.

$$(A \cup B)^c = A^c \cap B^c$$
$$(A \cap B)^c = A^c \cup B^c$$

Joint, Marginal, and Conditional

Joint Probability $P(A \cap B)$ or $P(A, B)$ – Probability of A and B.

Marginal (Unconditional) Probability $P(A)$ – Probability of A.

Conditional Probability $P(A|B) = P(A,B)/P(B)$ – Probability of A, given that B occurred.

Conditional Probability *is* Probability $P(A|B)$ is a probability function for any fixed B. Any theorem that holds for probability also holds for conditional probability.

Probability of an Intersection or Union

Intersections via Conditioning

$$P(A, B) = P(A)P(B|A)$$
$$P(A, B, C) = P(A)P(B|A)P(C|A, B)$$

Unions via Inclusion-Exclusion

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$
$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C).$$

Simpson's Paradox

Law of Total Probability (LOTP)

Let $B_1, B_2, B_3, \dots B_n$ be a *partition* of the sample space (i.e., they are disjoint and their union is the entire sample space).

$$P(A) = P(A|B_1)P(B_1) + P(A|B_2)P(B_2) + \dots + P(A|B_n)P(B_n)$$
$$P(A) = P(A \cap B_1) + P(A \cap B_2) + \dots + P(A \cap B_n)$$

For **LOTP with extra conditioning**, just add in another event C!

$$P(A|C) = P(A|B_1, C)P(B_1|C) + \dots + P(A|B_n, C)P(B_n|C)$$
$$P(A|C) = P(A \cap B_1|C) + P(A \cap B_2|C) + \dots + P(A \cap B_n|C)$$

Special case of LOTP with B and B^c as partition:

$$P(A) = P(A|B)P(B) + P(A|B^c)P(B^c)$$
$$P(A) = P(A \cap B) + P(A \cap B^c)$$

Bayes' Rule

Bayes' Rule, and with extra conditioning (just add in C!)

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$
$$P(A|B, C) = \frac{P(B|A, C)P(A|C)}{P(B|C)}$$

We can also write

$$P(A|B, C) = \frac{P(A, B, C)}{P(B, C)} = \frac{P(B, C|A)P(A)}{P(B, C)}$$

Odds Form of Bayes' Rule

$$\frac{P(A|B)}{P(A^c|B)} = \frac{P(B|A)}{P(B|A^c)} \frac{P(A)}{P(A^c)}$$

The *posterior odds* of A are the *likelihood ratio* times the *prior odds*.

Random Variables and their Distributions

PMF, CDF, and Independence

Probability Mass Function (PMF) Gives the probability that a *discrete* random variable takes on the value x.

$$p_X(x) = P(X = x)$$

Probability Cheatsheet v2.0

Linear Algebra

Source: https://minireference.com/static/tutorials/linear_algebra_in_4_pages.pdf

Linear algebra explained in four pages

Excerpt from the NO BULLSHIT GUIDE TO LINEAR ALGEBRA by Ivan Savov

***Abstract*—This document will review the fundamental ideas of linear algebra. We will learn about matrices, matrix operations, linear transformations and discuss both the theoretical and computational aspects of linear algebra. The tools of linear algebra open the gateway to the study of more advanced mathematics. A lot of *knowledge buzz* awaits you if you choose to follow the path of *understanding*, instead of trying to memorize a bunch of formulas.**

I. INTRODUCTION

Linear algebra is the math of vectors and matrices. Let n be a positive integer and let $\mathbb{R}$ denote the set of real numbers, then $\mathbb{R}^n$ is the set of all n-tuples of real numbers. A vector $\vec{v} \in \mathbb{R}^n$ is an n-tuple of real numbers. The notation "$\in S$" is read "element of S." For example, consider a vector that has three components:

$$\vec{v} = (v_1, v_2, v_3) \ \in \ (\mathbb{R}, \mathbb{R}, \mathbb{R}) \equiv \mathbb{R}^3.$$

A matrix $A \in \mathbb{R}^{m \times n}$ is a rectangular array of real numbers with m rows and n columns. For example, a 3×2 matrix looks like this:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \in \begin{bmatrix} \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} \\ \mathbb{R} & \mathbb{R} \end{bmatrix} \equiv \mathbb{R}^{3\times 2}.$$

The purpose of this document is to introduce you to the mathematical operations that we can perform on vectors and matrices and to give you a feel of the power of linear algebra. Many problems in science, business, and technology can be described in terms of vectors and matrices so it is important that you understand how to work with these.

Prerequisites

The only prerequisite for this tutorial is a basic understanding of high school math concepts[1] like numbers, variables, equations, and the fundamental arithmetic operations on real numbers: addition (denoted $+$), subtraction (denoted $-$), multiplication (denoted implicitly), and division (fractions).

B. Matrix operations

We denote by A the matrix as a whole and refer to its entries as a_{ij}. The mathematical operations defined for matrices are the following:

- addition (denoted $+$)

$$C = A + B \qquad \Leftrightarrow \qquad c_{ij} = a_{ij} + b_{ij}.$$

- subtraction (the inverse of addition)
- matrix product. The product of matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n \times \ell}$ is another matrix $C \in \mathbb{R}^{m \times \ell}$ given by the formula

$$C = AB \qquad \Leftrightarrow \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{32}b_{22} \end{bmatrix}$$

- matrix inverse (denoted A^{-1})
- matrix transpose (denoted $^{\mathsf{T}}$):

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \beta_1 & \beta_2 & \beta_3 \end{bmatrix}^{\mathsf{T}} = \begin{bmatrix} \alpha_1 & \beta_1 \\ \alpha_2 & \beta_2 \\ \alpha_3 & \beta_3 \end{bmatrix}.$$

- matrix trace: $\mathrm{Tr}[A] \equiv \sum_{i=1}^{n} a_{ii}$
- determinant (denoted $\det(A)$ or $|A|$)

Note that the matrix product is not a commutative operation: $AB \neq BA$.

C. Matrix-vector product

The matrix-vector product is an important special case of the matrix-matrix product. The product of a 3×2 matrix A and the 2×1 column vector $\vec{x}$ results in a 3×1 vector $\vec{y}$ given by:

Linear algebra explained in four pages

Statistics

Source: http://web.mit.edu/~csvoss/Public/usabo/stats_handout.pdf

Statistics Cheat Sheet

Population

The entire group one desires information about

Sample

A subset of the population taken because the entire population is usually too large to analyze
Its characteristics are taken to be representative of the population

Mean

Also called the arithmetic mean or average
The sum of all the values in the sample divided by the number of values in the sample/population
μ is the mean of the population; $\overline{x}$ is the mean of the sample

Median

The value separating the higher half of a sample/population from the lower half
Found by arranging all the values from lowest to highest and taking the middle one (or the mean of the middle two if there are an even number of values)

Variance

Measures dispersion around the mean
Determined by averaging the squared differences of all the values from the mean
Variance of a population is σ^2

$$\sigma^2 = \frac{\sum (x-\mu)^2}{n}$$

Can be calculated by subtracting the square of the mean from the average of the squared scores:

$$\sigma^2 = \frac{\sum x^2}{n} - \mu^2$$

Statistics Cheat Sheet

Calculus

Source: http://tutorial.math.lamar.edu/getfile.aspx?file=B,41,N

Calculus Cheat Sheet

Limits

Definitions

Precise Definition : We say $\lim_{x\to a} f(x) = L$ if for every $\varepsilon > 0$ there is a $\delta > 0$ such that whenever $0 < |x-a| < \delta$ then $|f(x) - L| < \varepsilon$.

"Working" Definition : We say $\lim_{x\to a} f(x) = L$ if we can make $f(x)$ as close to L as we want by taking x sufficiently close to a (on either side of a) without letting $x = a$.

Right hand limit : $\lim_{x\to a^+} f(x) = L$. This has the same definition as the limit except it requires $x > a$.

Left hand limit : $\lim_{x\to a^-} f(x) = L$. This has the same definition as the limit except it requires $x < a$.

Limit at Infinity : We say $\lim_{x\to\infty} f(x) = L$ if we can make $f(x)$ as close to L as we want by taking x large enough and positive.

There is a similar definition for $\lim_{x\to-\infty} f(x) = L$ except we require x large and negative.

Infinite Limit : We say $\lim_{x\to a} f(x) = \infty$ if we can make $f(x)$ arbitrarily large (and positive) by taking x sufficiently close to a (on either side of a) without letting $x = a$.

There is a similar definition for $\lim_{x\to a} f(x) = -\infty$ except we make $f(x)$ arbitrarily large and negative.

Relationship between the limit and one-sided limits

$\lim_{x\to a} f(x) = L \Rightarrow \lim_{x\to a^+} f(x) = \lim_{x\to a^-} f(x) = L$ $\qquad$ $\lim_{x\to a^+} f(x) = \lim_{x\to a^-} f(x) = L \Rightarrow \lim_{x\to a} f(x) = L$

$\lim_{x\to a^+} f(x) \neq \lim_{x\to a^-} f(x) \Rightarrow \lim_{x\to a} f(x)$ Does Not Exist

Properties

Assume $\lim_{x\to a} f(x)$ and $\lim_{x\to a} g(x)$ both exist and c is any number then,

1. $\lim_{x\to a}[cf(x)] = c\lim_{x\to a} f(x)$
2. $\lim_{x\to a}[f(x) \pm g(x)] = \lim_{x\to a} f(x) \pm \lim_{x\to a} g(x)$
3. $\lim_{x\to a}[f(x)g(x)] = \lim_{x\to a} f(x)\ \lim_{x\to a} g(x)$
4. $\lim_{x\to a}\left[\frac{f(x)}{g(x)}\right] = \frac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)}$ provided $\lim_{x\to a} g(x) \neq 0$
5. $\lim_{x\to a}[f(x)]^n = \left[\lim_{x\to a} f(x)\right]^n$
6. $\lim_{x\to a}\left[\sqrt[n]{f(x)}\right] = \sqrt[n]{\lim_{x\to a} f(x)}$

Basic Limit Evaluations at $\pm\infty$

Note : $\operatorname{sgn}(a) = 1$ if $a > 0$ and $\operatorname{sgn}(a) = -1$ if $a < 0$.

1. $\lim_{x\to\infty} \mathbf{e}^x = \infty$ & $\lim_{x\to-\infty} \mathbf{e}^x = 0$

5. n even : $\lim_{x\to\pm\infty} x^n = \infty$

Calculus Cheat Sheet

Evaluation Techniques

Continuous Functions

If $f(x)$ is continuous at a then $\lim_{x\to a} f(x) = f(a)$

Continuous Functions and Composition

$f(x)$ is continuous at b and $\lim_{x\to a} g(x) = b$ then

$\lim_{x\to a} f(g(x)) = f\left(\lim_{x\to a} g(x)\right) = f(b)$

Factor and Cancel

$$\lim_{x\to 2}\frac{x^2+4x-12}{x^2-2x} = \lim_{x\to 2}\frac{(x-2)(x+6)}{x(x-2)} = \lim_{x\to 2}\frac{x+6}{x} = \frac{8}{2} = 4$$

Rationalize Numerator/Denominator

$$\lim_{x\to 9}\frac{3-\sqrt{x}}{x^2-81} = \lim_{x\to 9}\frac{3-\sqrt{x}}{x^2-81}\,\frac{3+\sqrt{x}}{3+\sqrt{x}} = \lim_{x\to 9}\frac{9-x}{(x^2-81)(3+\sqrt{x})} = \lim_{x\to 9}\frac{-1}{(x+9)(3+\sqrt{x})} = \frac{-1}{(18)(6)} = -\frac{1}{108}$$

Combine Rational Expressions

$$\lim_{h\to 0}\frac{1}{h}\left(\frac{1}{x+h}-\frac{1}{x}\right) = \lim_{h\to 0}\frac{1}{h}\left(\frac{x-(x+h)}{x(x+h)}\right) = \lim_{h\to 0}\frac{1}{h}\left(\frac{-h}{x(x+h)}\right) = \lim_{h\to 0}\frac{-1}{x(x+h)} = -\frac{1}{x^2}$$

L'Hospital's Rule

If $\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{0}{0}$ or $\lim_{x\to a}\frac{f(x)}{g(x)} = \frac{\pm\infty}{\pm\infty}$ then,

$\lim_{x\to a}\frac{f(x)}{g(x)} = \lim_{x\to a}\frac{f'(x)}{g'(x)}$ a is a number, ∞ or $-\infty$

Polynomials at Infinity

$p(x)$ and $q(x)$ are polynomials. To compute $\lim_{x\to\pm\infty}\frac{p(x)}{q(x)}$ factor largest power of x in $q(x)$ out of both $p(x)$ and $q(x)$ then compute limit.

$$\lim_{x\to-\infty}\frac{3x^2-4}{5x-2x^2} = \lim_{x\to-\infty}\frac{x^2\left(3-\frac{4}{x^2}\right)}{x^2\left(\frac{5}{x}-2\right)} = \lim_{x\to-\infty}\frac{3-\frac{4}{x^2}}{\frac{5}{x}-2} = -\frac{3}{2}$$

Piecewise Function

$\lim_{x\to-2} g(x)$ where $g(x) = \begin{cases} x^2+5 & \text{if } x < -2 \\ 1-3x & \text{if } x \geq -2 \end{cases}$

Compute two one sided limits,

$\lim_{x\to-2^-} g(x) = \lim_{x\to-2^-} x^2+5 = 9$

$\lim_{x\to-2^+} g(x) = \lim_{x\to-2^+} 1-3x = 7$

One sided limits are different so $\lim_{x\to-2} g(x)$ doesn't exist. If the two one sided limits had been equal then $\lim_{x\to-2} g(x)$ would have existed and had the same value.

Some Continuous Functions

Partial list of continuous functions and the values of x for which they are continuous.

1. Polynomials for all x.
2. Rational function, except for x's that give division by zero.
3. $\sqrt[n]{x}$ (n odd) for all x.
4. $\sqrt[n]{x}$ (n even) for all $x \geq 0$.
5. $\mathbf{e}^x$ for all x.
6. $\ln x$ for $x > 0$.
7. $\cos(x)$ and $\sin(x)$ for all x.
8. $\tan(x)$ and $\sec(x)$ provided $x \neq \cdots, -\frac{3\pi}{2}, -\frac{\pi}{2}, \frac{\pi}{2}, \frac{3\pi}{2}, \cdots$
9. $\cot(x)$ and $\csc(x)$ provided $x \neq \cdots, -2\pi, -\pi, 0, \pi, 2\pi, \cdots$

Calculus Cheat Sheet