



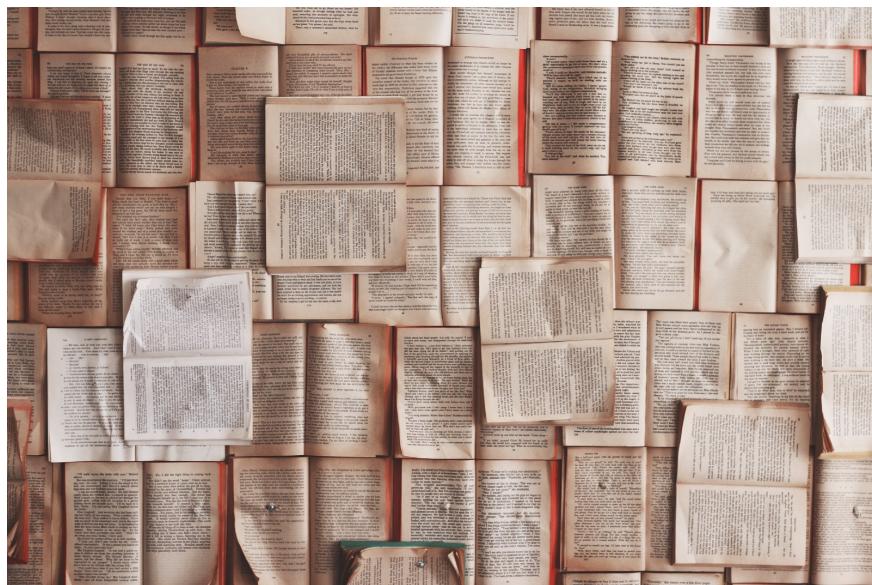
Robbie Allen [Follow](#)

Startup Founder & Author turned PhD Student @UNCCS focused on Artificial Intelligence. Found...
Jun 1 · 4 min read

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

If you like this article, check out another by Robbie:

[My Curated List of AI and Machine Learning Resources](#)



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

Given how rapidly the Machine Learning space is evolving, I imagine these will go out of date quickly, but at least as of June 1, 2017, they are pretty current.

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

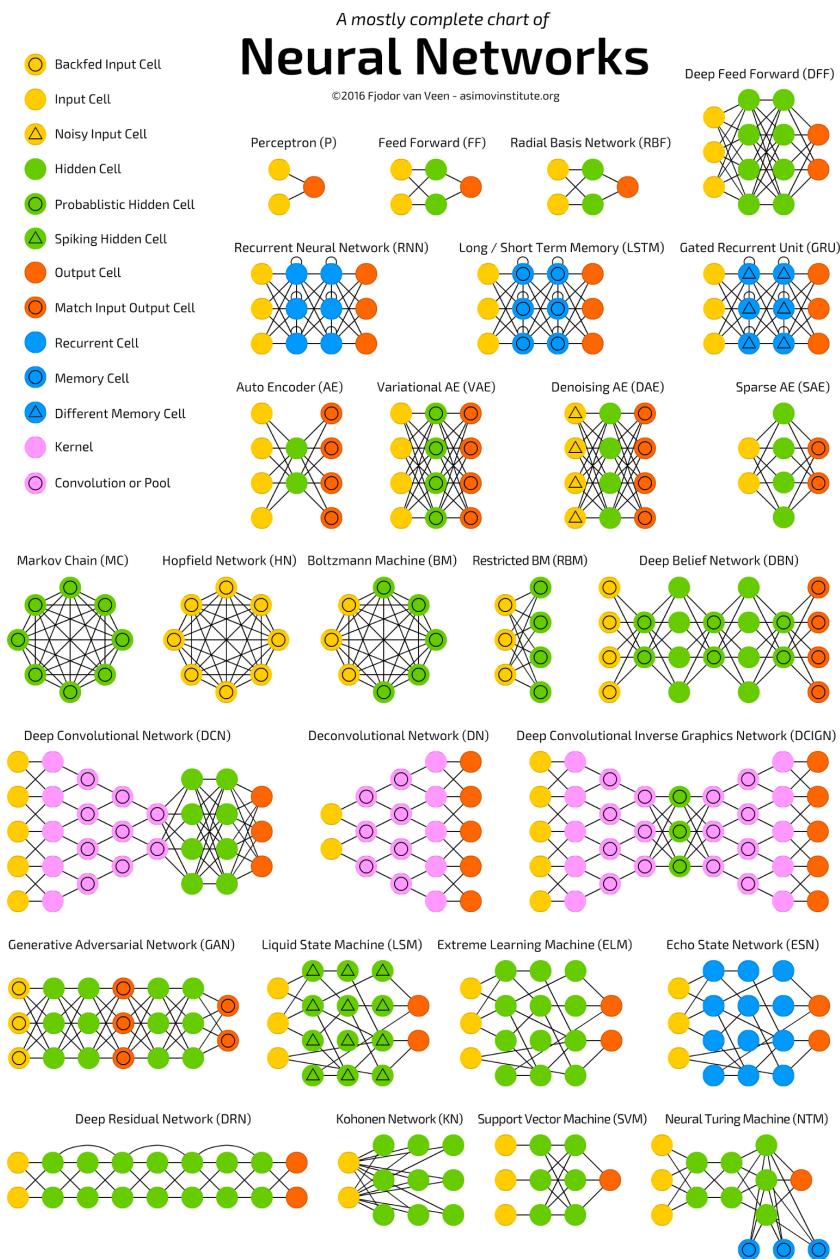
If you like this post, give it a below.

Machine Learning

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

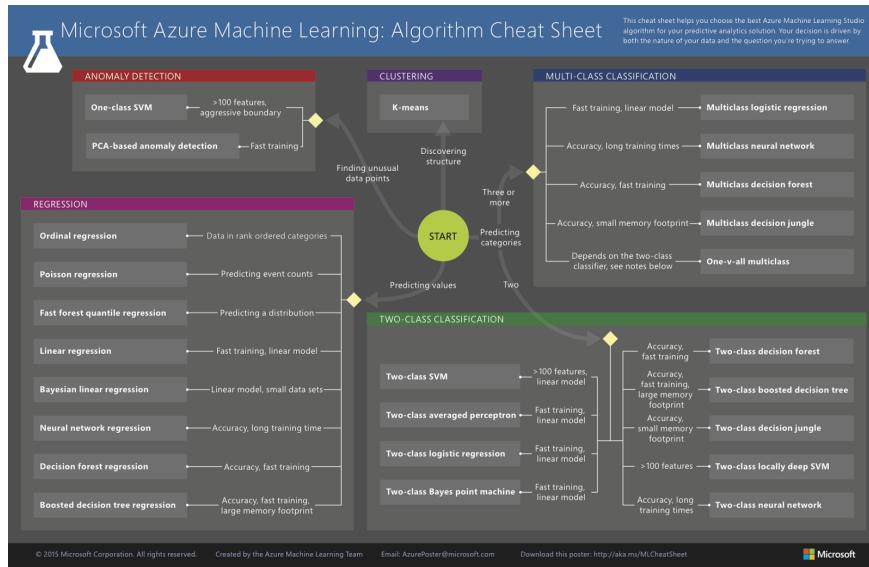
Neural Network Architectures

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



Microsoft Azure Algorithm Flowchart

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

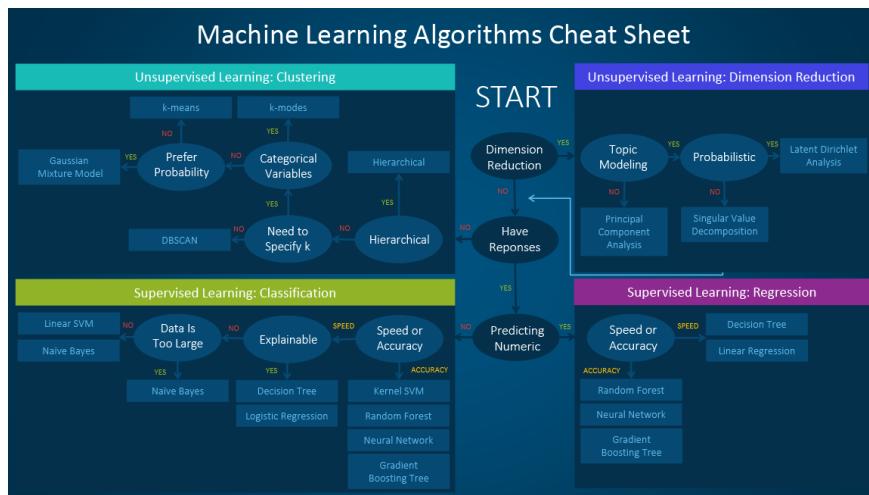


Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio

SAS Algorithm Flowchart

Source:

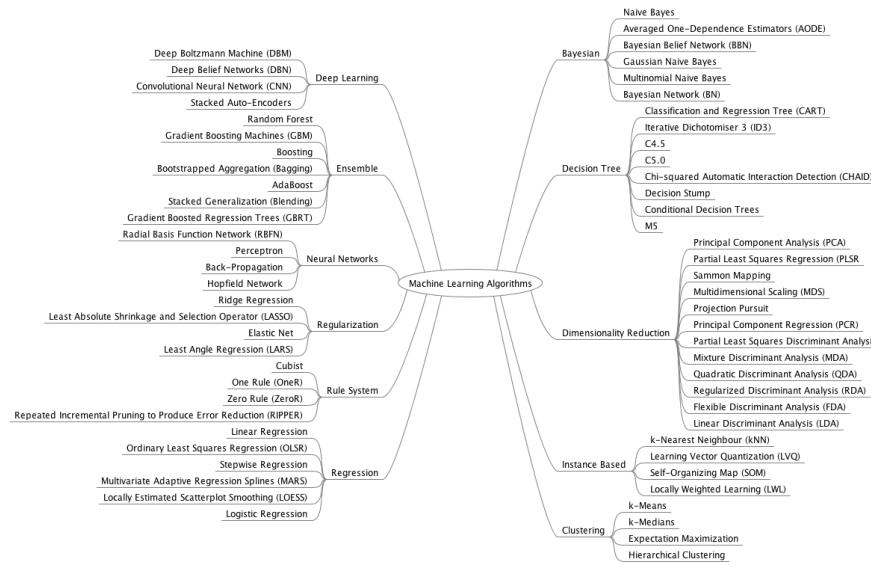
<http://blogs.sas.com/content/subconsciousmusings/2017/04/12/machine-learning-algorithm-use/>



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/>

the world of machine learning algorithms – a summary



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. | <ul style="list-style-type: none">✗ Sometimes too simple to capture complex relationships between variables.✗ Tendency for the model to "overfit". |
| | Logistic regression | The adaptation of linear regression to problems of classification (e.g., yes/no questions, groups, etc.) | Also easy to understand. | <ul style="list-style-type: none">✗ Sometimes too simple to capture complex relationships between variables.✗ 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. | <ul style="list-style-type: none">✗ Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data. |
| | 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. | <ul style="list-style-type: none">✗ Can be slow to output predictions relative to other algorithms.✗ Not easy to understand predictions. |
| | Gradient Boosting | Uses even weaker decision trees, that are increasingly focused on "hard" examples. | High-performing. | <ul style="list-style-type: none">✗ A small change in the feature set or training set can create radical changes in the model.✗ 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. | <ul style="list-style-type: none">✗ Very, very slow to train, because they have so many layers. Require a lot of power.✗ Almost impossible to understand predictions. |



©2017 Dataiku, Inc. | www.dataiku.com | contact@dataiku.com | @dataiku

Python

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

Algorithms

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

CHEATSHEET

Machine Learning Algorithms



(Python and R Codes)

Types

Supervised Learning

- Decision Tree
- Random Forest
- kNN
- Logistic Regression

Unsupervised Learning

- Apriori algorithm
- k-means
- Hierarchical Clustering

Reinforcement Learning

- Markov Decision Process
- Q Learning

Python
Code

R
Code

Linear Regression

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

```
#Load Train and Test datasets
#Identify feature and response variable(s) and
#values must be numeric and numpy arrays
x_train <- input_variables_values_training_datasets
y_train <- target_variables_values_training_datasets
x <- 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 COTTON 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 LIBRARY

Python module is simply a .py file

```
List Module Content | dir(module)
Load Module | import module
Call Function From Module | module.function()
```

* 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 without creating namespaces, use `from module 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` - String, used in `print` and `Unicode` in Python 3
- String can be in single/double quotes
- String is a sequence of characters, thus can be treated like other sequences

- Special character can be done via \ or preface with `\u`
- `\u0000 = "\u0000\000"`

• String formating can be done in a number of ways

```
str1 = "Hello %s" % "World"
str1 = template % (4.88, "Hello", 2)
```

SCALAR TYPES

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

SCALAR TYPES

* `str`, `bool`, `int` and `float` are also explicit type

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

* Common usage of None :

if variable is None :

6. `datetime` - built-in python `datetime` module provides `date`, `time`, `timedelta`.

* `timedelta` combines information stored in `date` and `time`

Create datetime | strptime("2009/01/01", "%Y/%m/%d")
Get date | date()
Get time | time()
Get timedelta | timedelta('00:00:00')
Format datetime | date.strftime("%m/%d/%Y %H:%M")

Change Field | date.replace(minute=10)
Get Difference | diff = dt1 - dt2

Note : `diff` is a `timedelta` (readable object)`

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, extend() is preferable

* Insert is computationally expensive with append.

* Checking that a list contains a value is lot slower than checking if it's in a set. Use sets to scan where others (based on hash tables) in constant time.

Build-in `bisect` module :

* Implements binary search and insertion into a sorted list

* `bisect_left` finds the location, where `bisect.insert` actually inserts into list.

* WARNING : `bisect` module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them on unsorted lists will result in errors and may lead to incorrect results.

DATA STRUCTURES

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

TUPLE

One dimensional, immutable sequence of Python objects of ANY type

Note : Most objects in Python are mutable except for `string` & `tuples`

DATA STRUCTURES

```
Create Tuple | tuple = 4, 5, 6 or
Create Nested Tuple | tuple = (4,5,6), (7,8)
Concatenate Tuples | tuple1 + tuple2
Concatenate Tuples | tuple1 + tuple2
Unpack Tuple | a, b, c = tuple1
```

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 | list = [1, "a", 3]
Create List | list = []
Append to List | list.append("a")
Concatenate Lists | list1 + list2
Extend List | list.extend(list2)
Append to End of List | list.append("b")
Insert to Specific Position | list.insert(index, "b")
Invert of Insert | list.pop(index)
Remove First Value | list.remove("a")
Delete Element | del list[0]
Check Membership | 3 in list == True ***
Sort List | list.sort()
Sort User-defined Function | sorted([1, 2, 3], key=len)
Sort by Length | sorted([1, 2, 3], key=len)
```

Note : `list.remove()` removes first occurrence of value

Note : `del` is a `list` (readable object)`

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, extend() is preferable

* Insert is computationally expensive with append.

* Checking that a list contains a value is lot slower than checking if it's in a set. Use sets to scan where others (based on hash tables) in constant time.

Build-in `bisect` module :

* Implements binary search and insertion into a sorted list

* `bisect_left` finds the location, where `bisect.insert` actually inserts into list.

* WARNING : `bisect` module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them on unsorted lists will result in errors and may lead to incorrect results.

DATA STRUCTURES

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

Slicing

Sequence types include `str`, `array`, `tuple`, `list`, etc.

Notation | list[start:stop]
Notation | list[start:stop]
If step is used |

Note : * 'start' index is included, but 'stop' index is NOT.
* start/stop can be omitted in which they default to 0 and len(list).

* Application of 'step' :
Take every other element | list[1::2]
Reverse a string | list[::-1]

DICT (HASH MAP)

```
Create Dict | dict = {"key": "value", 2: 2}
Create Dict From Sequence | dict = dict.fromkeys(range(3))
Get/Set/Insert Element | dict["key"] = "value"
Get With Default Value | dict.get("key", "not found")
Check If Key Exists | "key" in dict
Delete Element | del dict["key"]
Get Key List | dict.keys() ===
Get Value List | dict.values() ===
Update Values | dict.update(dict2)
Update Value | # dict values are replaced by dict2
```

* `KeyError` exception if the key does not exist.

** `get()` by default (aka no `defaultValue`) will return `None` if the key does not exist.

*** Returns the list of keys and values in the same order. However, the order is not any particular order because they 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`. A scalar value is hashable if the hash(`"this is a string"`, hash(1, 2)) - this would fail

SET

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

Create Set | set((1, 2, 3)) or
set([1, 2, 3])

Test SubSet | set1.issubset(set2)

Test Superset | set1.issuperset(set2)

Test sets have same elements | set1 == set2

SET OPERATIONS

Operations | update(s1) | 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
```

Sum of two variables

```
>>> x-2
```

Subtraction of two variables

```
>>> x*2
```

Multiplication of two variables

```
>>> 10
```

Exponentiation of a variable

```
>>> x**2
```

Remainder of a variable

```
>>> x%2
```

Division of a variable

```
>>> x/float(2)
```

Variables 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 + "Init"
```

"thisStringIsAwesomeInit"

```
>>> "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

Select item at index 1

Select 3rd last item

Select items at index 1 and 2

Select items after index 0

Select items before index 3

Copy my_list

my_list[1][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', 'list', 'is', 'nice')

>>> my_list2[0] > 4

[4, 5, 6, 7]

Subset

slice[0:2]

slice[-1:-3:-1]

slice[1:-1:-1]

slice[1:-1:-2]

slice[1:-1:-3]

slice[1:-1:-4]

slice[1:-1:-5]

slice[1:-1:-6]

slice[1:-1:-7]

slice[1:-1:-8]

slice[1:-1:-9]

slice[1:-1:-10]

slice[1:-1:-11]

slice[1:-1:-12]

slice[1:-1:-13]

slice[1:-1:-14]

slice[1:-1:-15]

slice[1:-1:-16]

slice[1:-1:-17]

slice[1:-1:-18]

slice[1:-1:-19]

slice[1:-1:-20]

slice[1:-1:-21]

slice[1:-1:-22]

slice[1:-1:-23]

slice[1:-1:-24]

slice[1:-1:-25]

slice[1:-1:-26]

slice[1:-1:-27]

slice[1:-1:-28]

slice[1:-1:-29]

slice[1:-1:-30]

slice[1:-1:-31]

slice[1:-1:-32]

slice[1:-1:-33]

slice[1:-1:-34]

slice[1:-1:-35]

slice[1:-1:-36]

slice[1:-1:-37]

slice[1:-1:-38]

slice[1:-1:-39]

slice[1:-1:-40]

slice[1:-1:-41]

slice[1:-1:-42]

slice[1:-1:-43]

slice[1:-1:-44]

slice[1:-1:-45]

slice[1:-1:-46]

slice[1:-1:-47]

slice[1:-1:-48]

slice[1:-1:-49]

slice[1:-1:-50]

slice[1:-1:-51]

slice[1:-1:-52]

slice[1:-1:-53]

slice[1:-1:-54]

slice[1:-1:-55]

slice[1:-1:-56]

slice[1:-1:-57]

slice[1:-1:-58]

slice[1:-1:-59]

slice[1:-1:-60]

slice[1:-1:-61]

slice[1:-1:-62]

slice[1:-1:-63]

slice[1:-1:-64]

slice[1:-1:-65]

slice[1:-1:-66]

slice[1:-1:-67]

slice[1:-1:-68]

slice[1:-1:-69]

slice[1:-1:-70]

slice[1:-1:-71]

slice[1:-1:-72]

slice[1:-1:-73]

slice[1:-1:-74]

slice[1:-1:-75]

slice[1:-1:-76]

slice[1:-1:-77]

slice[1:-1:-78]

slice[1:-1:-79]

slice[1:-1:-80]

slice[1:-1:-81]



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=',')
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 arr 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
```

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([seq])
# seq = any sequence like object, list, tuple, dict, etc.

Create Special NdArray np.zeros(10)
# one dimensional ndarray with 10 elements of value 0
np.zeros((2, 3))
# two dimensional ndarray with 6 elements of value 0
np.zeros((2, 3, 4))
# three dimensional ndarray of uninitialized values
4, np.newaxis() or
4, np.newaxis()
# creates N by 1 identity matrix
np.arange(1, 10)

NdArray version of Python's range
Get # of Dimension np.ndim
Get Dimension Size np.shape, .size, ...
Get Data Type ** ndarray.dtype
Explicit Casting ndarray.astype(np.int32) **

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

```

Slicing (INDEXING/SUBSETTING)

- Slicing (i.e. `ndarray[2:6]`) is a "View" on the underlying array and any changes made to the array via modification (i.e. `ndarray[2:6] = 8`) to the View will be reflected in the original array.
- Instead of a "View", explicit copy of slicing via:

```

ndarray[2:6].copy()

```

Multidimensional array indexing notation:

```

ndarray[0][2] or ndarray[0, 2]

```

Boolean Indexing

```

ndarray[(names == 'Bob') | (names == 'Willie'), 2]
# "2" means select from 3rd column on

```

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

Fancy indexing (aka indexing using integer arrays)

Select a subset of rows in a particular order:

```

ndarray[[3, 6, 4], 1]
ndarray[[-1, 0], 1]
# negative indices select rows from the end

```

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

Setting data with assignment:

```

ndarray[(ndarray < 0) = 0]

```

- If ndarray is two-dimensional, ndarray[1 < 0] creates a two-dimensional boolean array.

COMMON OPERATIONS

- Transposing
 - A special form of reshaping which returns a "view" on the underlying data without copying anything.
- Vectorized wrappers (for functions that take scalar values)
 - `math.sqrt()` works on only a scalar
 - `np.sqrt([seq])` -> any sequence (list, ndarray, etc) to return a ndarray
- Vectorized expressions
 - `np.where(cond, x, y)` is a vectorized version of the expression `x if condition else y`
- Common Usages:
 - `np.where(booleanArray > 0, 1, -1)` -> a new array (same shape) of 1 or -1 values
 - `np.where(cond, 1, 0).argmax()` -> finds the first True element
 - `argmax()` can be used to find the index of the max value
 - Even though argmax is the first element that has a "price > number" in an array of price data.
- Aggregations/Reductions Methods (i.e. mean, sum, std)
 - Compute mean `ndarray1.mean()` or `np.mean(ndarray1)`
 - Compute statistics `ndarray1.mean(axis=1)` over axis *
 - * axis = 0 means column axis, 1 is row axis.

Boolean arrays methods

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

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 ndarray values `resultBooleanArray = np.isin(ndarray1, [2, 3, 6])`

- Other set methods: `isnane(ndarray), isnan(ndarray), isfinite(ndarray)`

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, 5))

```

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

Created by Arianne Colton and Sean Chen
www.datasciencecheats.com
Based on content from "Python for Data Analysis" by Wes McKinney
Updated: August 18, 2016

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

Python For Data Science Cheat Sheet

Numpy Basics
Learn Python for Data Science interactively at: www.DataCamp.com

NumPy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

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

NumPy Arrays

| | | |
|-----------------|-----------------|----------------------------|
| 1D array | 2D array | 3D array |
| | | |
| axis 0 | axis 1 | axis 0 axis 1 axis 2 |

Creating Arrays

```

>>> a = np.array([1,2,3])
>>> b = np.array([(4,5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[1,1,1,2,3], (4,5,6)], dtype = float)
>>> d = np.zeros((2,2))

```

Initial Placeholders

```

>>> np.zeros(3,4)
Create an array of zeros
>>> np.ones((2,3), dtype=np.int16)
Create an array of ones
>>> d = np.arange(10,25,5)
Create an array of even numbers
>>> np.linspace(0,2,5)
Create an array of evenly spaced numbers
>>> e = np.full((2,2),7)
Create a constant array
>>> f = np.eye(2)
Create a 2x2 identity matrix
>>> g = np.random.randint(1,2)
Create random integers
>>> h = np.empty(3,2)
Create an empty array

```

I/O

Saving & Loading On Disk

```

>>> np.savetxt("my_array", a)
>>> np.savetxt("my_array.csv", a, delimiter=",")
>>> np.load("my_array.npy")

```

Saving & Loading Text Files

```

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

```

Data Types

```

>>> np.int64
Signed 64-bit integer type
>>> np.float32
Standard 32-bit precision floating point
>>> np.complex64
Complex numbers represented by 128 bits
>>> np.bool_
Boolean type (TRUE and FALSE values)
>>> np.object_
Fixed-length string type
>>> np.string_
Fixed-length unicode type

```

Inspecting Your Array

```

>>> a.shape
Array dimensions
>>> a.size
Length of array
>>> a.dtype
Numerical type
>>> a.itemsize
Number of array elements
>>> a.nbytes
Data type of array elements
>>> a.base
Base array type
>>> a.strides
Convert an array to a different type

```

Asking For Help

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

Array Mathematics

Arithmetic Operations

```

>>> a = a + b
Addition
>>> a = np.subtract(a,b)
Subtraction
>>> a = a * b
Multiplication
>>> a = np.multiply(a,b)
Exponentiation
>>> a = np.divide(a,b)
Division
>>> a = np.mod(a,b)
Modulo
>>> a = np.fmod(a,b)
Fmod
>>> a = np.add(a,b)
Addition
>>> a = np.subtract(a,b)
Subtraction
>>> a = a + b
Addition
>>> a = np.multiply(a,b)
Multiplication
>>> a = np.divide(a,b)
Division
>>> a = np.mod(a,b)
Modulo
>>> a = np.fmod(a,b)
Fmod

```

Comparison

```

>>> a == b
Element-wise comparison
>>> a != b
Element-wise comparison
>>> a < b
Element-wise comparison
>>> a <= b
Element-wise comparison
>>> np.array_equal(a, b)
Element-wise comparison

```

Aggregate Functions

```

>>> a.sum()
Array-wise sum
>>> a.min()
Array-wise minimum value
>>> a.max()
Maximum value of the array
>>> b.cumsum(axis=1)
Cumulative sum of the array
>>> a.mean()
Mean
>>> a.std()
Standard deviation
>>> a.correlate()
Correlation coefficient

```

Copying Arrays

```

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

```

Sorting Arrays

```

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

```

Subsetting, Slicing, Indexing

| | | |
|-------------------------|-------------------------|--|
| Subsetting | Slicing | Also see Lists |
| <code>a[1]</code> | <code>a[1:2]</code> | Select the element at the 2nd index |
| <code>b[1,2]</code> | <code>b[1,1:2]</code> | Select the element at row 0 column 2 (equivalent to <code>b[1][2]</code>) |
| Indexing | <code>a[1,2,3]</code> | Select items at index 0 and i in column 1 |
| <code>a[1,2,3]</code> | <code>a[1,1,1:2]</code> | Select all items at row 0 (equivalent to <code>a[0,:,:,1]</code>) Same as <code>a[1,1,1]</code> |
| <code>b[1,1]</code> | <code>b[1,1,1:2]</code> | Reversed array = <code>a[::-1]</code> |
| <code>a[1,1,1:2]</code> | <code>a[1,1,1:2]</code> | Select elements from <code>a</code> less than 2 |
| <code>b[1,1,1:2]</code> | <code>b[1,1,1:2]</code> | Select elements (<code>1,0,0,1,1,1,2,0,1</code>) and columns |

Array Manipulation

Transposing Array

```

>>> a = np.transpose(b)
Permute array dimensions

```

Changing Array Shape

```

>>> b.ravel()
Flatten the array

```

Adding/Removing Elements

```

>>> g.reshape(3,-2)
Reshape, but don't change data

```

Combining Arrays

```

>>> np.concatenate((a, b), axis=0)
Return a new array with shape (2,6)
Append items to an array
Insert items to an array
Delete items from an array

```

Stacking Arrays

```

>>> np.vstack((a, b))
Concatenate arrays
Stack arrays vertically (row-wise)

```

Splitting Arrays

```

>>> np.hsplit(a, 3)
Split the array horizontally at the 3rd index
>>> np.vsplit(a, 2)
Split the array vertically at the 2nd index

```

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

NumPy

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

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

In [1]: `import numpy as np`

NumPy Arrays, dtypes, and shapes

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

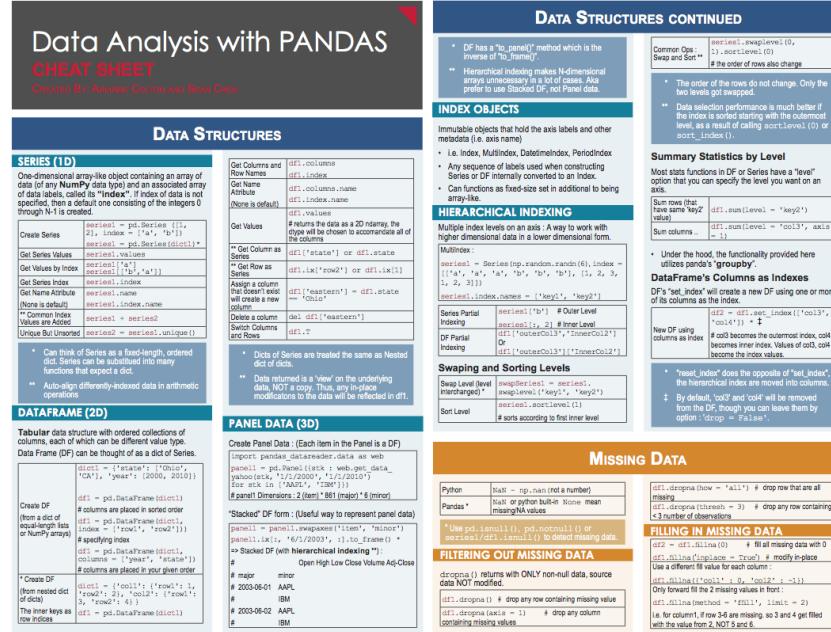
[1 2 3]
(3,)
int64

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

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

Pandas

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



This cheat sheet provides a quick reference for Pandas data structures and operations. It is organized into several sections:

- DATA STRUCTURES**: Includes Series (1D) and DataFrame (2D).
- PANEL DATA (3D)**: Describes Panel Data.
- DATA STRUCTURES CONTINUED**: Discusses Index Objects, Hierarchical Indexing, and Summary Statistics by Level.
- MISSING DATA**: Shows how to handle missing values.
- FILLING IN MISSING DATA**: Provides methods for filling missing data.

The cheat sheet also includes code snippets for creating and manipulating Pandas objects, such as Series and DataFrames, and demonstrates various indexing and selection techniques.

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

| Python For Data Science Cheat Sheet | |
|---|---|
| Pandas Basics | |
| Pandas The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language. | Asking For Help <code>>>> help(pd.Series.loc)</code> |
| Pandas Data Structures Series A one-dimensional labeled array capable of holding any data type | Selection <code>>>> df[1]</code> Get one element <code>>>> df[['Country']]</code> Get subset of a DataFrame |
| DataFrame A two-dimensional labeled data structure with columns of potentially different types | Also see NumPy Arrays <code>>>> df[1:2]</code> Select single value by row & column <code>>>> df.iat[0, 0]</code> Select single value by row & column labels <code>>>> df.ix[2]</code> Select single row or subset of rows <code>>>> df.ix[[0, 1, 2]]</code> Select a single column or subset of columns <code>>>> df.ix[[0, 1], 'Capital']</code> Select rows and columns <code>>>> df['New Delhi']</code> Series a where value is not > x when value is <=0 or >= Population if value>1200000000 Use filter to adjust DataFrame |
| I/O Read and Write to CSV <code>>>> pd.read_csv('file.csv', header=None, nrows=5)</code> <code>>>> pd.to_csv('myDataFrame.csv')</code> | Read and Write to SQL Query or Database Table <code>>>> from sqlalchemy import create_engine</code> <code>>>> engine = create_engine('sqlite:///memory:')</code> <code>>>> pd.read_sql("SELECT * FROM my_table", engine)</code> <code>>>> pd.read_sql_table('my_table', engine)</code> <code>>>> pd.read_sql_query("SELECT * FROM my_table", engine)</code> <code>>>> pd.read_sql('ibisconvenience wrapper around read_sql_table() and read_sql_query()')</code> <code>>>> pd.read_sql('myDF', engine)</code> |
| Read and Write to Excel <code>>>> pd.read_excel('file.xlsx')</code> <code>>>> pd.to_excel('xlsx>DataFrame.xlsx', sheet_name='Sheet1')</code> Read multiple sheets from the same file | Dropping <code>>>> s.drop(['a', 'c'])</code> Drop values from rows (axis=0) <code>>>> df.drop(['Country'], axis=1)</code> Drop values from columns (axis=1) |
| <code>>>> a = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])</code> <code>>>> df = pd.DataFrame(data, columns=['Country', 'Capital', 'Population'])</code> | Sort & Rank <code>>>> df.sort_index(by='Country')</code> Sort by row or column index <code>>>> df.rank()</code> Assign ranks to entries |
| | Retrieving Series/DataFrame Information Basic Information <code>>>> df.shape</code> (rows,columns) <code>>>> df.info()</code> Describe DataFrame/series <code>>>> df.describe()</code> Info on DataFrame <code>>>> df.corr()</code> Numerical or boolean value |
| | Summary <code>>>> df.sum()</code> Sum of values <code>>>> df.min()</code> Minimum value of series <code>>>> df.max()</code> Maximum/maximum values <code>>>> df.idxmax()</code> Index of maximum value <code>>>> df.describe()</code> Summary statistics <code>>>> df.mean()</code> Mean of values <code>>>> df.median()</code> Median of values |
| | Applying Functions <code>>>> a = lambda x: x**2</code> Apply function <code>>>> df.applymap(f)</code> Apply function element-wise |
| | Data Alignment Internal Data Alignment NA values are introduced in the indices that don't overlap: <code>>>> a = pd.Series([1, -3, 0], index=['a', 'b', 'd'])</code> <code>a + a3</code> <code>a + 100</code> <code>a + 0.0</code> <code>a - 7.0</code> |
| | Arithmetic Operations with Fill Methods You can also do the internal data alignment yourself with the help of the fill methods: <code>>>> a = pd.Series([1, -3, 0], index=['a', 'b', 'd'])</code> <code>>>> a.add(a2, fill_value=0)</code> <code>>>> a + a3</code> <code>>>> a + 100</code> <code>>>> a + 0.0</code> <code>>>> a - 7.0</code> <code>>>> a.add(a2, fill_value=2)</code> <code>>>> a + 2</code> <code>>>> a.div(a2, fill_value=4)</code> <code>>>> a * 2</code> <code>>>> a.fill(a2, fill_value=2)</code> |

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

2 Create Plot

3 Plotting Routines

4 Customize Plot

5 Save Plot

6 Show Plot

Close & Clear

Data

Figure

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 = [1,2,3,4,5]
>>> y = [1,4,9,16,25]
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(x, y, color='lightblue', linewidth=3)
>>> ax.scatter([3,15,25], [1,4,9], color='green', marker='*')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()

```

Workflow

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
>>> from sklearn import linear_model
>>> X, y = iris.data, iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
>>> regr = linear_model.LinearRegression()
>>> regr.fit(X_train, y_train)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> knn.score(X_test, y_test)
```

Loading The Data (Also see NumPy & Pandas)

Most objects to be used are 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.array([[1, 2], [3, 4], [5, 6], [7, 8]])
>>> y = np.array([0, 1, 0, 1])
>>> X[0] < 0.5 == 0
```

Training And Test Data

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

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> X_train = scaler.fit_transform(X_train)
>>> X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> normalizer = Normalizer()
>>> normalized_X = normalizer.fit_transform(X_train)
>>> normalized_X[0] == X[0]
```

Binarization

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

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression().fit(X_train, y_train)
```

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
>>> kmeans = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

```
>>> i = int(0)
>>> knn.fit(X_train, y_train)
>>> i += 1
```

Unsupervised Learning

```
>>> kmeans.fit(X_train)
```

Fit the model to the data

Fit the model to the data

Fit the model to the data

Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> i = int(0)
>>> y_pred = lr.predict(X_test)
>>> i += 1
```

Unsupervised Estimators

```
>>> y_pred = kmeans.predict(X_test)
```

Predict labels

Predict labels

Predict labels

Predict labels in clustering algor

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> ann.score(X_test, y_test)
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Classification Report

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

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 = [-0.5, 0.2, 1.0]
>>> y_pred = [-0.3, 0.4, 0.8]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> y_true = [0.5, 1.0, 1.5]
>>> y_pred = [0.7, 1.2, 1.4]
>>> mean_squared_error(y_true, y_pred)
```

R² Score

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

Clustering Metrics

Adjusted Rand Index

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

Homogeneity Score

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

V-measure

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

Cross-Validation

```
>>> from sklearn.model_selection import cross_val_score
>>> print(cross_val_score(X_train, y_train, cv=4))
>>> print(cross_val_score(X, y, cv=4))
```

Tune Your Model

Grid Search

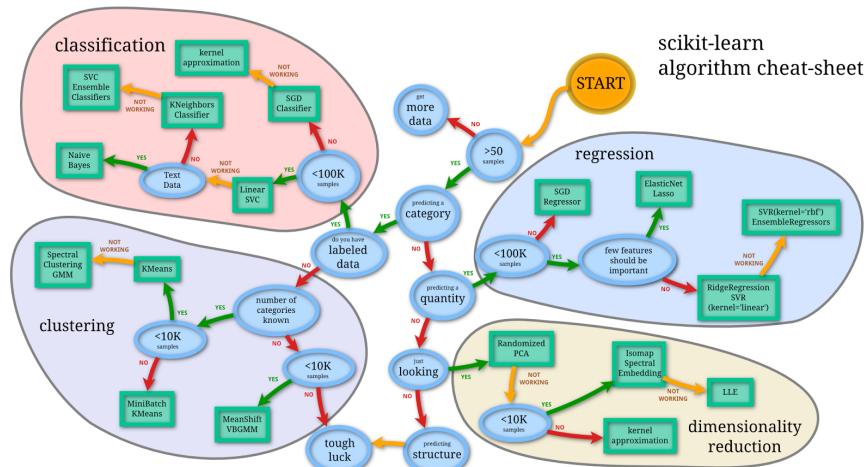
```
>>> from sklearn.grid_search import GridSearchCV
>>> param = {'C': [0.01, 0.1, 1, 10, 100, 1000],
>>>           'metric': ['euclidean', 'manhattan', 'minkowski']}
>>> grid = GridSearchCV(SGDClassifier(), param_grid=param)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_estimator_, grid.best_score_)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> param = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
>>>           'weights': ['uniform', 'distance']}
>>> rsearch = RandomizedSearchCV(KNeighborsClassifier(),
>>>                               param_distributions=param,
>>>                               n_iter=10, scoring='f1',
>>>                               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/bigguys/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

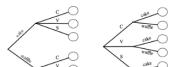
Probability Cheatsheet v2.0

Compiled by William Chen (<https://wchen.com>) and Joe Blitzstein with contributions from Schachar Ohn, Wan-Jie Liu, Yufei Hwang and Jeyu Hwang. Material based on Joe Blitzstein's [Stat110.net](http://stat110.net) and Blitzstein/Hwang's Introduction to Probability (<http://math.mit.edu/~ Blitzstein/IntroductionToProbability.html>). Licensed under CC BY-NC-SA 4.0. Please share, reuse, adapt, and report errors at http://github.com/wchen/probability_cheatsheet.

Last Updated September 4, 2015

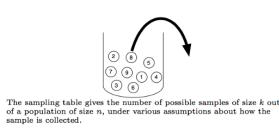
Counting

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 r th component has n_r possible outcomes, then overall there are $n_1 n_2 \dots n_r$ possibilities for the whole experiment.

Sampling Table



Thinking Conditionally

Independence

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

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

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

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.

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

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

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 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

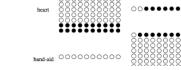
$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

$$P(A \cup B \cup C) = P(A) + P(B) + P(C)$$

$$- P(A \cap B) - P(A \cap C) - P(B \cap C)$$

$$+ P(A \cap B \cap C).$$

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).

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

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' as partition:

$$P(A) = P(A|B)P(B) + P(A|B')P(B')$$

$$P(A) = P(A \cap B) + P(A \cap B')$$

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

$$\frac{P(A|B)}{P(A'|B)} = \frac{P(B|A)}{P(B|A')} \cdot \frac{P(A)}{P(A')}$$

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 fuzz 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 concepts¹ 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 $+$)
- 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

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

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

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

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

- matrix trace: $\text{Tr}[A] \equiv \sum_{i=1}^n a_{ii}$
- 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; \bar{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

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

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

Statistics Cheat Sheet

Calculus

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

| | |
|--|--|
| <p>Calculus Cheat Sheet</p> <p>Limits</p> <p>Definitions</p> <p>Precise Definition : We say $\lim_{x \rightarrow a} f(x) = L$ if for every $\epsilon > 0$ there is a $\delta > 0$ such that whenever $0 < x - a < \delta$ then $f(x) - L < \epsilon$.</p> <p>"Working" Definition : We say $\lim_{x \rightarrow a} f(x) = L$ 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 $x = a$.</p> <p>Right hand limit : $\lim_{x \rightarrow a^+} f(x) = L$. This has the same definition as the limit except it requires $x > a$.</p> <p>Left hand limit : $\lim_{x \rightarrow a^-} f(x) = L$. This has the same definition as the limit except it requires $x < a$.</p> <p>Relationship between the limit and one-sided limits</p> <p>$\lim_{x \rightarrow a} f(x) = L \Rightarrow \lim_{x \rightarrow a^+} f(x) = L \quad \lim_{x \rightarrow a^-} f(x) = L \Rightarrow \lim_{x \rightarrow a} f(x) = L$</p> <p>$\lim_{x \rightarrow a} f(x) \neq \lim_{x \rightarrow a^+} f(x) \Rightarrow \lim_{x \rightarrow a^-} f(x) \text{ Does Not Exist}$</p> <p>Properties</p> <p>Assume $\lim_{x \rightarrow a} f(x)$ and $\lim_{x \rightarrow a} g(x)$ both exist and c is any number then,</p> <ol style="list-style-type: none"> $\lim_{x \rightarrow a} [cf(x)] = c \lim_{x \rightarrow a} f(x)$ $\lim_{x \rightarrow a} [f(x) \pm g(x)] = \lim_{x \rightarrow a} f(x) \pm \lim_{x \rightarrow a} g(x)$ $\lim_{x \rightarrow a} [f(x)g(x)] = \lim_{x \rightarrow a} f(x) \lim_{x \rightarrow a} g(x)$ $\lim_{x \rightarrow a} \left[\frac{f(x)}{g(x)} \right] = \frac{\lim_{x \rightarrow a} f(x)}{\lim_{x \rightarrow a} g(x)}$ provided $\lim_{x \rightarrow a} g(x) \neq 0$ $\lim_{x \rightarrow a} [f(x)g(x)]^n = [\lim_{x \rightarrow a} f(x)]^n$ $\lim_{x \rightarrow a} [\sqrt[n]{f(x)}] = \sqrt[n]{\lim_{x \rightarrow a} f(x)}$ <p>Basic Limit Evaluations at $\pm\infty$</p> <p>Note : $\operatorname{sgn}(a) = 1$ if $a > 0$ and $\operatorname{sgn}(a) = -1$ if $a < 0$.</p> <ol style="list-style-type: none"> $\lim_{x \rightarrow \pm\infty} e^x = \infty$ & $\lim_{x \rightarrow \pm\infty} e^{-x} = 0$ n even : $\lim_{x \rightarrow \pm\infty} x^n = \infty$ | <p>Calculus Cheat Sheet</p> <p>Evaluation Techniques</p> <p>Continuous Functions</p> <p>If $f(x)$ is continuous at a then $\lim_{x \rightarrow a} f(x) = f(a)$</p> <p>If $\lim_{x \rightarrow a} f(x) = 0$ or $\lim_{x \rightarrow a} g(x) = \pm\infty$ then, $\lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \frac{0}{\pm\infty} = 0$</p> <p>Continuous Functions and Composition</p> <p>$f(x)$ is continuous at b and $\lim_{x \rightarrow a} g(x) = b$ then $\lim_{x \rightarrow a} f(g(x)) = f(\lim_{x \rightarrow a} g(x)) = f(b)$</p> <p>Rationalize Numerator/Denominator</p> <p>$\lim_{x \rightarrow \pm\infty} \frac{3-\sqrt{x}}{x^2-81} = \lim_{x \rightarrow \pm\infty} \frac{3-\sqrt{x}}{(x-9)(x+9)} = \lim_{x \rightarrow \pm\infty} \frac{9-x}{(x-9)(3+\sqrt{x})} = \lim_{x \rightarrow \pm\infty} \frac{-1}{(3+\sqrt{x})} = \frac{-1}{18(6)} = -\frac{1}{108}$</p> <p>Factor and Cancel</p> <p>$\lim_{x \rightarrow 2} \frac{x^2+4x-12}{x^2-2x} = \lim_{x \rightarrow 2} \frac{(x-2)(x+6)}{x(x-2)} = \lim_{x \rightarrow 2} \frac{x+6}{x} = 4$</p> <p>Polynomials at Infinity</p> <p>$p(x)$ and $q(x)$ are polynomials. To compute $\lim_{x \rightarrow \pm\infty} \frac{p(x)}{q(x)}$ factor largest power of x in $q(x)$ out of both $p(x)$ and $q(x)$ then compute limit.</p> <p>$\lim_{x \rightarrow \pm\infty} \frac{p(x)}{q(x)} = \frac{\text{factor largest power of } x \text{ in } q(x)}{\text{factor largest power of } x \text{ in } p(x)}$</p> <p>$\lim_{x \rightarrow \pm\infty} \frac{3x^2-4}{5x-2} = \lim_{x \rightarrow \pm\infty} \frac{x^2\left(\frac{3}{x^2}-\frac{4}{x^2}\right)}{x^2\left(\frac{5}{x^2}-\frac{2}{x^2}\right)} = \lim_{x \rightarrow \pm\infty} \frac{3-\frac{4}{x^2}}{\frac{5}{x^2}-2} = -\frac{3}{2}$</p> <p>Piecewise Function</p> <p>$\lim_{x \rightarrow a^-} g(x) \text{ where } g(x) = \begin{cases} x^2+5 & \text{if } x < -2 \\ 1-3x & \text{if } x \geq -2 \end{cases}$</p> <p>Compute two one sided limits,</p> <p>$\lim_{x \rightarrow a^+} g(x) = \lim_{x \rightarrow a^+} x^2+5 = 9$</p> <p>$\lim_{x \rightarrow a^-} g(x) = \lim_{x \rightarrow a^-} 1-3x = 7$</p> <p>One sided limits are different so $\lim_{x \rightarrow a} g(x)$ doesn't exist. If the two one sided limits had been equal then $\lim_{x \rightarrow a} g(x)$ would have existed and had the same value.</p> <p>Some Continuous Functions</p> <p>Partial list of continuous functions and the values of x for which they are continuous.</p> <ol style="list-style-type: none"> Polynomials for all x. Rational function, except for x's that give division by zero. \sqrt{x} (x odd) \sqrt{x} (x even) e^x for all x. $\ln x$ for $x > 0$. $\cos(x)$ and $\sin(x)$ for all x. $\tan(x)$ and $\sec(x)$ provided $x \neq \frac{\pi}{2}$. $\cot(x)$ and $\csc(x)$ provided $x \neq 0$. |
|--|--|

Calculus Cheat Sheet

