



Stefan Kojouharov [Follow](#)

Founder & Editor of @ChatbotsLife. I help Companies Create Great Chatbots & Share my Insights...

Jul 9 · 7 min read

Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Big Data

The Most Complete List of Best AI Cheat Sheets

Over the past few months, I have been collecting AI cheat sheets. From time to time I share them with friends and colleagues and recently I have been getting asked a lot, so I decided to organize and share the entire collection. To make things more interesting and give context, I added descriptions and/or excerpts for each major topic.

This is the most complete list and the Big-O is at the very end, enjoy...

If you like this list, you can let me know [here](#).

Neural Networks

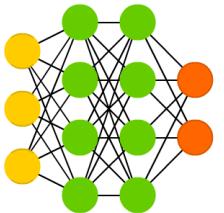
A mostly complete chart of

Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

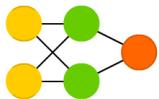
Deep Feed Forward (DFF)



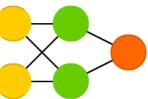
Perceptron (P)



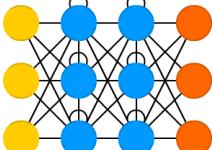
Feed Forward (FF)



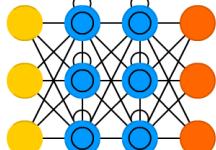
Radial Basis Network (RBF)



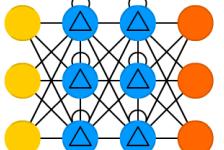
Recurrent Neural Network (RNN)



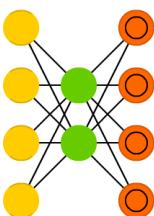
Long / Short Term Memory (LSTM)



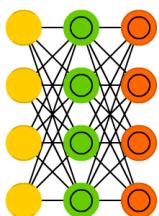
Gated Recurrent Unit (GRU)



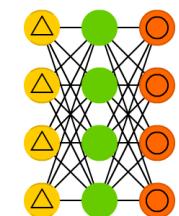
Auto Encoder (AE)



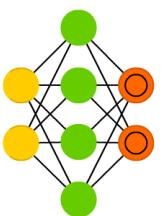
Variational AE (VAE)



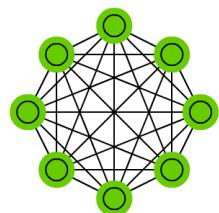
Denoising AE (DAE)



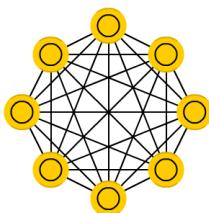
Sparse AE (SAE)



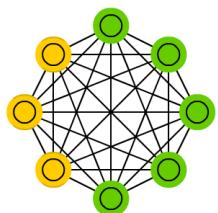
Markov Chain (MC)



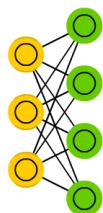
Hopfield Network (HN)



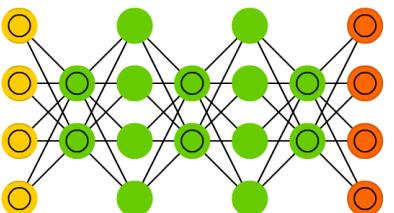
Boltzmann Machine (BM)



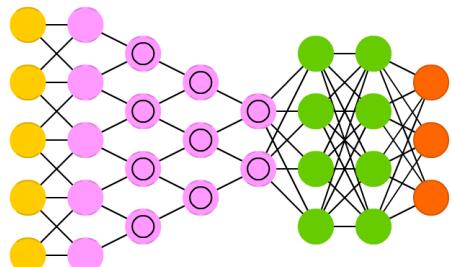
Restricted BM (RBM)



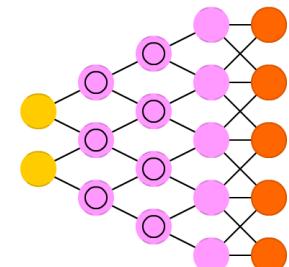
Deep Belief Network (DBN)



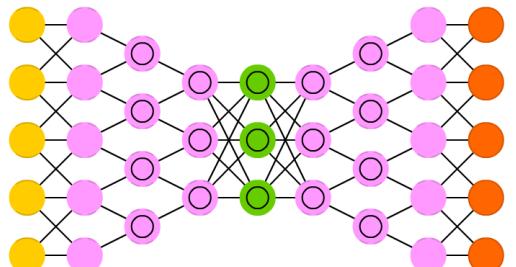
Deep Convolutional Network (DCN)



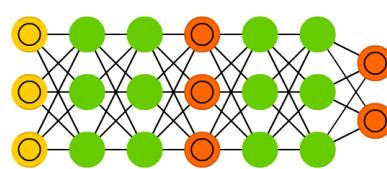
Deconvolutional Network (DN)



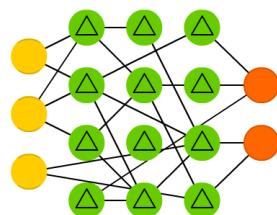
Deep Convolutional Inverse Graphics Network (DCIGN)



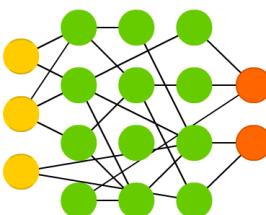
Generative Adversarial Network (GAN)



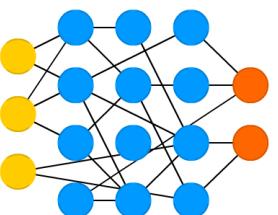
Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



Echo State Network (ESN)



Deep Residual Network (DRN)



Kohonen Network (KN)

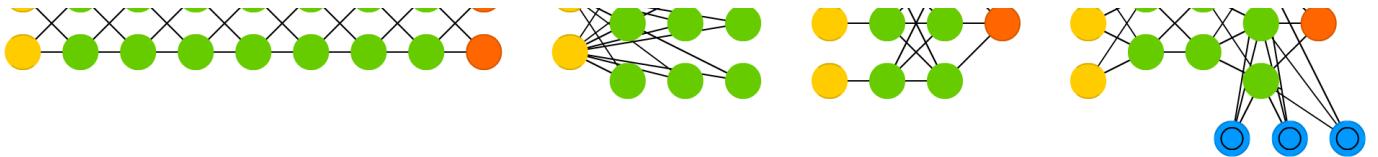


Support Vector Machine (SVM)



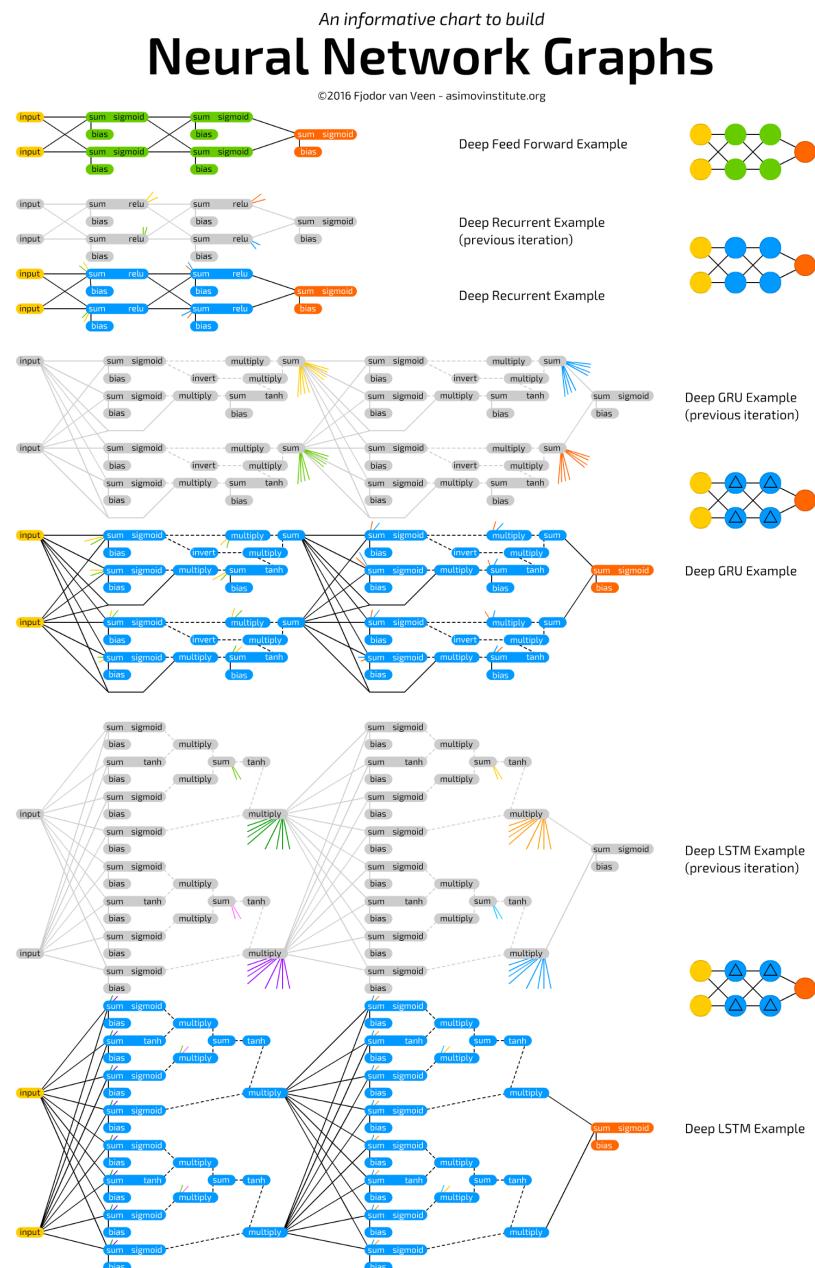
Neural Turing Machine (NTM)





Neural Networks Cheat Sheet

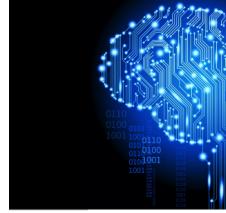
Neural Networks Graphs



Neural Networks Graphs Cheat Sheet

Linear Vector Spaces: Definition: A linear vector space, X , is a set of elements (vectors) defined over a scalar field, \mathbb{F} , that satisfies the following conditions: 1) If $x \in X$ and $y \in X$ then $x+y \in X$. 2) $x \cdot y = y \cdot x$. 3) $(x+y) \cdot z = x \cdot z + y \cdot z$. 4) There is a unique element $0 \in X$ such that $x+0=x$ for all $x \in X$. 5) For each vector $x \in X$ there is a unique vector $-x \in X$, such that $x+(-x)=0$. 6) Multiplication, for all scalars $a \in \mathbb{F}$, and all vectors $x \in X$, such that $a \cdot (x \cdot y) = (ax) \cdot y$. 7) For any $x, y \in X$, $1 \cdot x = x$. 8) For any vectors $x, y \in X$ and any scalar $a \in \mathbb{F}$, $a(x+y) = ax + ay$. Linear Independence: Consider n vectors $\{x_1, x_2, \dots, x_n\}$. If there exists n scalars a_1, a_2, \dots, a_n , at least one of which is nonzero, such that $a_1x_1 + a_2x_2 + \dots + a_nx_n = 0$, then the $\{x_i\}$ are linearly dependent. Spanning a Space: Let X be a linear vector space and let $\{u_1, u_2, \dots, u_n\}$ be a subset of vectors in X . This subset spans X if and only if for every vector $x \in X$ there exist scalars x_1, x_2, \dots, x_n , such that $x = x_1u_1 + x_2u_2 + \dots + x_nu_n$.	Perceptron Architecture: $a = \text{hardlim}(\mathbf{W}\mathbf{p} + b)$, $\mathbf{W} = [\mathbf{w}_1 \mathbf{w}_2 \dots \mathbf{w}_n]^T$, $a_i = \text{hardlim}(a_i) = \text{hardlim}([\mathbf{w}_i^T \mathbf{p} + b_i])^T$. Decision Boundary: $\mathbf{w}^T \mathbf{p} + b = 0$ The decision boundary is always orthogonal to the weight vector. Single-layer perceptrons can only classify linearly separable vectors. Perceptron Learning Rule $\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T, \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e},$ where $\mathbf{e} = \mathbf{t} - \mathbf{a}$	General Minimization Algorithm: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \quad \text{or} \quad \Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$ Steepest Descent Algorithm: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k \quad \text{where } \mathbf{g}_k = \nabla F(\mathbf{x}) _{\mathbf{x}=\mathbf{x}_k}$ Stable Learning Rate: $(\alpha_k = \alpha, \text{ constant}) \alpha < \frac{\lambda_{max}}{\lambda_{min}}$ $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ Eigenvalues of Hessian matrix \mathbf{A} Learning Rate to Minimize Along the Line: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ (For quadratic fn.) After Minimization Along the Line: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k = 0$ ADALINE: $a = \text{purelin}(\mathbf{W}\mathbf{p} + b)$ Mean Square Error: (for ADALINE it is a quadratic fn.) $F(\mathbf{x}) = E[\mathbf{z}^2] = E[(\mathbf{t} - \mathbf{a})^2] = E[(\mathbf{t} - \mathbf{x}^T \mathbf{z})^2]$ $F(\mathbf{x}) = c - 2\mathbf{x}^T \mathbf{h} + \mathbf{x}^T \mathbf{R} \mathbf{x}$, $c = E[\mathbf{t}^2], \mathbf{h} = E[\mathbf{z}] \text{ and } \mathbf{R} = E[\mathbf{z}\mathbf{z}^T] \Rightarrow \mathbf{A} = 2\mathbf{R}, \mathbf{d} = -2\mathbf{h}$ Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$, where $\mathbf{x} = \begin{bmatrix} \mathbf{w} \\ \mathbf{b} \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} \mathbf{t} \\ 1 \end{bmatrix}$ LMS Algorithm: $\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha \mathbf{e}(k) \mathbf{p}^T(k)$ $\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha \mathbf{e}(k)$ Convergence Point: $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$ Stable Learning Rate: $0 < \alpha < 1/\lambda_{max}$ where λ_{max} is the maximum eigenvalue of \mathbf{R} Adaptive Filter ADALINE: $a(k) = \text{purelin}(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^R \mathbf{w}_{i,i} y_i (k-i+1) + b$
---	---	---

Neural Network Cheat Sheet



Ultimate Guide to Leveraging NLP & Machine Learning for your Chatbot

Code Snippets and Github Included
chatbotslife.com

Machine Learning Overview

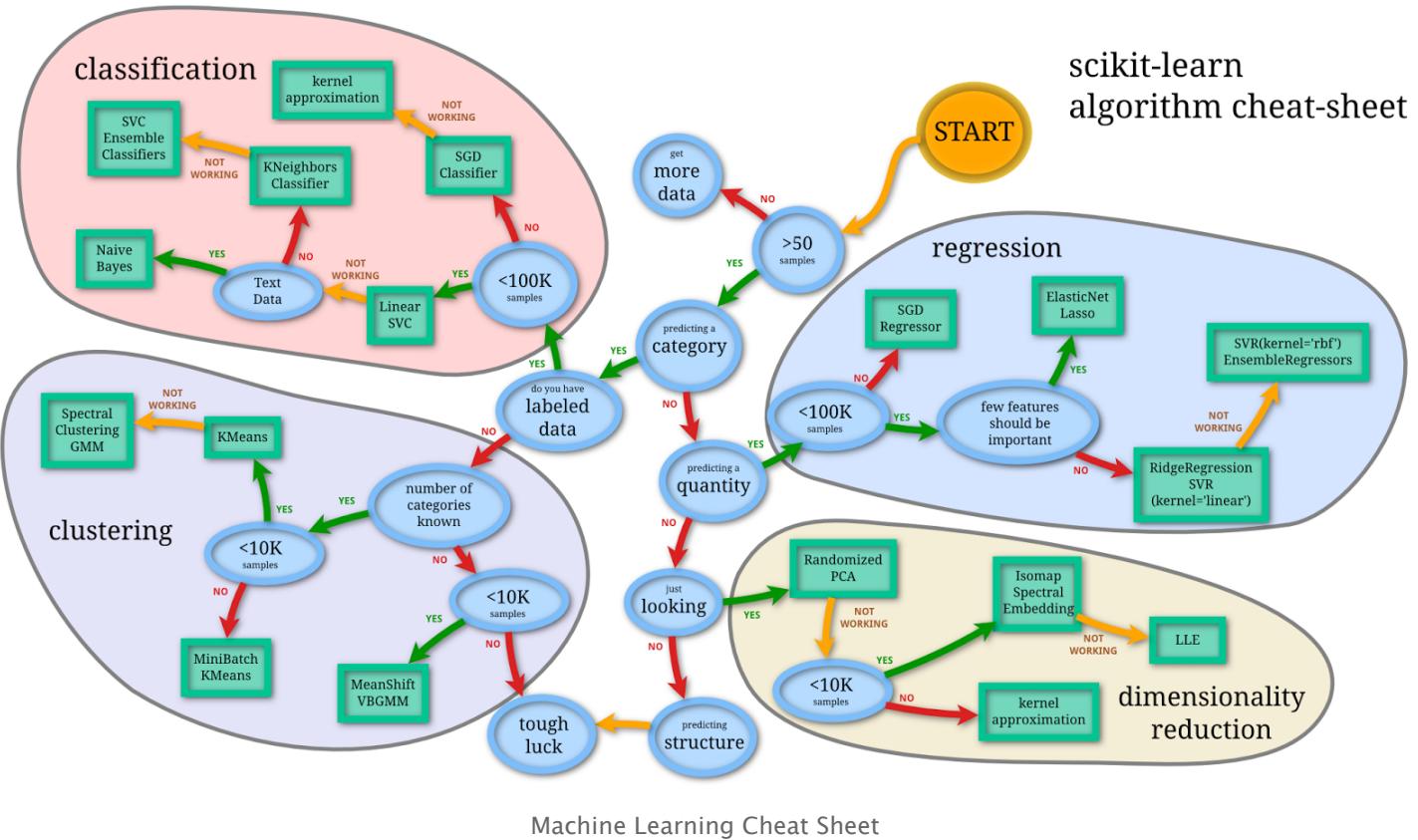


Machine Learning Cheat Sheet

Machine Learning: Scikit-learn algorithm

This machine learning cheat sheet will help you find the right estimator for the job which is the most difficult part. The flowchart will help you check the documentation and rough guide of each estimator that will help you to know more about the problems and how to solve it.

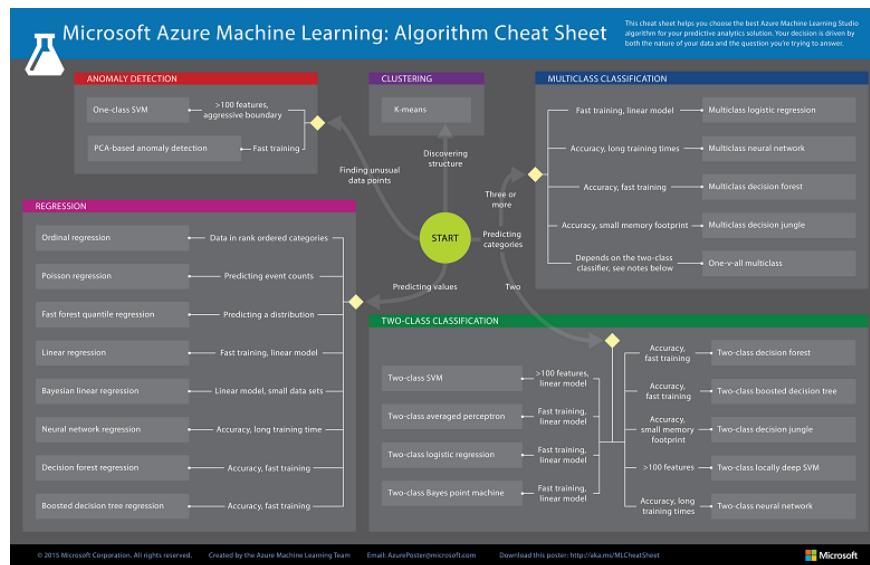
scikit-learn algorithm cheat-sheet



Scikit-Learn

Scikit-learn (formerly **scikits.learn**) is a [free software machine learning library](#) for the Python programming language. It features various [classification](#), [regression](#) and [clustering](#) algorithms including [support vector machines](#), [random forests](#), [gradient boosting](#), [k-means](#) and [DBSCAN](#), and is designed to interoperate with the Python numerical and scientific libraries [NumPy](#) and [SciPy](#).

This machine learning cheat sheet from Microsoft Azure will help you choose the appropriate machine learning algorithms for your predictive analytics solution. First, the cheat sheet will ask you about the data nature and then suggests the best algorithm for the job.



MACHINE LEARNING ALGORITHM CHEAT SHEET

>>> If you like this list, you can let me know [here](#). <<<

Ultimate Guide to Leveraging NLP & Machine Learning for your Chatbot

Code Snippets and Github Included

chatbotslife.com



Python for Data Science

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	<pre>>>> x=5</pre>												
	<pre>>>> x</pre>												
	5												
Calculations With Variables													
	<pre>>>> x+2</pre>												
	Sum of two variables												
	<pre>>>> x-2</pre>												
	Subtraction of two variables												
	<pre>>>> x*2</pre>												
	Multiplication of two variables												
	<pre>>>> x**2</pre>												
	Exponentiation of a variable												
	<pre>>>> x%2</pre>												
	Remainder of a variable												
	<pre>>>> x/float(2)</pre>												
	Division of a variable												
	2.5												
Types and Type Conversion													
	<table border="1"><tr><td><code>str()</code></td><td><code>"5", "3.45", "True"</code></td><td>Variables to strings</td></tr><tr><td><code>int()</code></td><td><code>5, 3, 1</code></td><td>Variables to integers</td></tr><tr><td><code>float()</code></td><td><code>5.0, 1.0</code></td><td>Variables to floats</td></tr><tr><td><code>bool()</code></td><td><code>True, True, True</code></td><td>Variables to booleans</td></tr></table>	<code>str()</code>	<code>"5", "3.45", "True"</code>	Variables to strings	<code>int()</code>	<code>5, 3, 1</code>	Variables to integers	<code>float()</code>	<code>5.0, 1.0</code>	Variables to floats	<code>bool()</code>	<code>True, True, True</code>	Variables to booleans
<code>str()</code>	<code>"5", "3.45", "True"</code>	Variables to strings											
<code>int()</code>	<code>5, 3, 1</code>	Variables to integers											
<code>float()</code>	<code>5.0, 1.0</code>	Variables to floats											
<code>bool()</code>	<code>True, True, True</code>	Variables to booleans											
Asking For Help	<pre>>>> help(str)</pre>												
Strings													
	<pre>>>> my_string = 'thisStringIsAwesome' >>> my_string 'thisStringIsAwesome'</pre>												
String Operations													
	<pre>>>> my_string * 2 'thisStringIsAwesomethisStringIsAwesome' >>> my_string + 'Init' 'thisStringIsAwesomeInit' >>> "m" in my_string True</pre>												
Lists													
	<pre>>>> a = [] >>> a.append("rice") >>> my_list = ["my", "list", a, b] >>> my_list2 = [[4,5,6,7], [3,4,5,6]]</pre>												
Selecting List Elements													
	<pre>>>> my_list[1] >>> my_list[-3] Slice >>> my_list[1:3] >>> my_list[:1] >>> my_list[1:] Subset List of Lists >>> my_list2[1][0] >>> my_list2[1][1:2]</pre>												
	Index starts at 0												
	<pre>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]</pre>												
List Operations													
	<pre>>>> my_list * my_list [[4,5,6,7], [3,4,5,6], [4,5,6,7], [3,4,5,6]] >>> my_list * 2 [[4,5,6,7], [3,4,5,6], [4,5,6,7], [3,4,5,6]] >>> my_list2 * 4 [[4,5,6,7], [3,4,5,6], [4,5,6,7], [3,4,5,6], [4,5,6,7], [3,4,5,6], [4,5,6,7], [3,4,5,6]]</pre>												
List Methods													
	<pre>>>> my_list.index(a) Get the index of an item >>> my_list.count(a) Count an item >>> my_list.append("t") Append an item at a time >>> my_list.remove("t") Remove an item >>> del(my_list[0]) Reverse the list >>> my_list.reverse() Append an item >>> my_list.extend("t") Append an item >>> my_list.pop(-1) Remove an item >>> my_list.insert(0, "t") Insert an item >>> my_list.sort() Sort the list</pre>												
String Operations													
	<pre>>>> my_string[3] Get the index of an item >>> my_string[4:9] Select items at index o and 1</pre>												
String Methods													
	<pre>>>> my_string.upper() String to uppercase >>> my_string.lower() String to lowercase >>> my_string.count("t") Count String elements >>> my_string.replace("t", "i") Replace String elements >>> my_string.strip() Strip whitespace from ends</pre>												
Libraries													
	<p>Import libraries Import numpy Import numpy as np Selective import From math import pi</p>												
Install Python													
	<p>Anaconda Spyder Jupyter</p>												
Numpy Arrays													
	<pre>>>> my_list = [1, 2, 3, 4] >>> my_array = np.array(my_list) >>> my_2darray = np.array([[1,2,3],[4,5,6]])</pre>												
Selecting Numpy Array Elements													
	<pre>>>> my_array[1] Select item at index 1 >>> my_array[0:2] Select items at index o and 1</pre>												
Numpy Array Operations													
	<pre>>>> my_array * 3 array([3, 6, 9, 12]) >>> my_array * 2 array([2, 4, 6, 8]) >>> my_array + np.array([5, 6, 7, 8]) array([6, 10, 12])</pre>												
Numpy Array Functions													
	<pre>>>> my_array.shape Get the dimensions of the array >>> np.append(my_array, other_array) Append items to an array >>> np.insert(my_array, 1, 9) 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</pre>												

Python Data Science Cheat Sheet

Python For Data Science Cheat Sheet

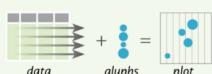
Bokeh

Learn Bokeh interactively at www.DataCamp.com, taught by Bryan Van de Ven, core contributor



Plotting With Bokeh

The Python interactive visualization library **Bokeh** enables high-performance visual presentation of large datasets in modern web browsers.



The basic steps to creating plots with the `bokeh.plotting` interface are:

1. Prepare some data:
`Python lists, NumPy arrays, Pandas DataFrames and other sequences of values`
 2. Create a new plot
 3. Add renderers for your data, with visual customizations
 4. Specify where to generate the output
 5. Show or save the results
- ```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5] # Step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title="simple line example",
 x_axis_label='x',
 y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2) # Step 2
>>> output_file("lines.html") # Step 3
>>> show(p) # Step 4
```

### 1 Data

#### Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9, 4, 65, 'US'], [32.4, 4, 66, 'Asia'], [21.4, 4, 109, 'Europe']]), columns=['mpg', 'cyl', 'hp', 'origin'], index=['Toyota', 'Flat', 'Volvo'])
```

>>> from bokeh.models import ColumnDataSource

>>> cds\_df = ColumnDataSource(df)

### 2 Plotting

```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
 x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

## 3 Renderers & Visual Customizations

### Glyphs

#### Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
 fill_color='white')
>>> p2.square(np.array([[1,5,3,5,5,5]], [1,4,3]),
 color='blue', size=1)
```

#### Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
 pd.DataFrame([[3,4,5],[3,2,1]]), color="blue")
```

#### Rows & Columns Layout

|                                   |                                                           |
|-----------------------------------|-----------------------------------------------------------|
| <b>Rows</b>                       | <code>&gt;&gt;&gt; from bokeh.layouts import row</code>   |
|                                   | <code>&gt;&gt;&gt; layout = row(p1,p2, p3)</code>         |
| <b>Columns</b>                    | <code>&gt;&gt;&gt; layout = column(p1,p2, p3)</code>      |
| <b>Nesting Rows &amp; Columns</b> | <code>&gt;&gt;&gt; layout = row(column(p1,p2), p3)</code> |

#### Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2], [p3]])
```

#### Tabbed Layout

```
>>> from bokeh.models.widgets import Panel, Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

#### Legends

##### Legend Location

###### Inside Plot Area

>>> p.legend.location = 'bottom\_left'

###### Outside Plot Area

>>> r1 = p2.asterisk(np.array([[1,2,3]], np.array([3,2,1]))

>>> r2 = p2.line([1,2,3,4], [3,4,5,6])

>>> legend = Legend(items=[{"One": [r1], "Two": [r2]}], location=(0, -30))

>>> p.add\_layout(legend, "right")

#### Linked Plots

```
>>> p4 = figure(plot_width = 100, tools='box_select,lasso_select')
>>> p4.circle('mpg', 'cyl', source=cds_df)
>>> p5 = figure(plot_width = 200, tools='box_select,lasso_select')
>>> p5.circle('mpg', 'cyl', source=cds_df)
>>> layout = row(p4,p5)
```

#### Linked Axes

>>> p2.x\_range = p1.x\_range

>>> p2.y\_range = p1.y\_range

#### Linked Brushing

>>> p4 = figure(plot\_width = 100, tools='box\_select,lasso\_select')

>>> p4.circle('mpg', 'cyl', source=cds\_df)

>>> p5 = figure(plot\_width = 200, tools='box\_select,lasso\_select')

>>> p5.circle('mpg', 'cyl', source=cds\_df)

>>> layout = row(p4,p5)

#### Legend Orientation

>>> p.legend.orientation = "horizontal"

>>> p.legend.orientation = "vertical"

#### Legend Background & Border

>>> p.legend.border\_line\_color = "navy"

>>> p.legend.background\_fill\_color = "white"

## Statistical Charts With Bokeh

Bokeh's high-level `bokeh.charts` interface is ideal for quickly creating statistical charts

#### Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red','blue'])
```

#### Box Plot

```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
 legend='bottom_right')
```

#### Histogram

```
>>> from bokeh.charts import Histogram
```

>>> p = Histogram(df, title='Histogram')

#### Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y='hp', marker='square',
 xlabel='Miles Per Gallon',
 ylabel='Horsepower')
```

#### DataCamp

Learn Python for Data Science interactively



## Big Data Cheat Sheet

## **TensorFlow**

In May 2017 Google announced the second-generation of the TPU, as well as the availability of the TPUs in [Google Compute Engine](#).<sup>[12]</sup>

The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs provide up to 11.5 petaflops.

## About

### TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

### Skflow

Scikit Flow provides a set of high level model classes that you can use to easily integrate with your existing Scikit-learn pipeline code. Scikit Flow is a simplified interface for TensorFlow, to get people started on predictive analytics and data mining. Scikit Flow has been merged into TensorFlow since version 0.8 and now called TensorFlow Learn.

### Keras

Keras is a minimalist, highly modular neural networks library, written in Python and capable of running on top of either TensorFlow or Theano

## Installation

### How to install new package in Python:

```
pip install <package-name>
```

Example: pip install requests

### How to install tensorflow?

```
device = cpu/gpu
python_version = cp27/cp34
sudo pip install
https://storage.googleapis.com/
tensorflow/linux/$device/tensorflow-
0.8.0-$python_version-none-linux_x86
_64.whl
```

### How to install Skflow

```
pip install sklearn
```

### How to install Keras

```
pip install keras
update ~/.keras/keras.json - replace
"theano" by "tensorflow"
```

## Helpers

### Python helper

#### Important functions

##### `type(object)`

Get object type

##### `help(object)`

Get help for object (list of available methods, attributes, signatures and so on)

##### `dir(object)`

Get list of object attributes  
(fields, functions)

##### `str(object)`

Transform an object to string

##### `object?`

Shows documentations about the object

##### `globals()`

Return the dictionary containing the current scope's global variables.

##### `locals()`

Update and return a dictionary containing the current scope's local variables.

### `id(object)`

Return the identity of an object. This is guaranteed to be unique among simultaneously existing objects.

### `import __builtin__` `dir(__builtin__)`

Other built-in functions

## TensorFlow

### Main classes

```
tf.Graph()
tf.Operation()
tf.Tensor()
tf.Session()
```

### Some useful functions

```
tf.get_default_session()
tf.get_default_graph()
tf.reset_default_graph()
ops.reset_default_graph()
tf.device("/cpu:0")
tf.name_scope(value)
tf.convert_to_tensor(value)
```

### TensorFlow Optimizers

```
GradientDescentOptimizer
AdadeltaOptimizer
AdagradOptimizer
MomentumOptimizer
AdamOptimizer
FtrlOptimizer
RMSPropOptimizer
```

### Reduction

```
reduce_sum
reduce_prod
reduce_min
reduce_max
reduce_mean
reduce_all
reduce_any
accumulate_n
```

### Activation functions

```
tf.nn?
relu
relu6
elu
softplus
softsign
dropout
bias_add
sigmoid
tanh
sigmoid_cross_entropy_with_logits
softmax
log_softmax
softmax_cross_entropy_with_logits
sparse_softmax_cross_entropy_with_logits
weighted_cross_entropy_with_logits
etc.
```

## Skflow

### Main classes

```
TensorFlowClassifier
TensorFlowRegressor
TensorFlowDNNClassifier
TensorFlowDNNRegressor
TensorFlowLinearClassifier
TensorFlowLinearRegressor
TensorFlowRNNClassifier
TensorFlowRNNRegressor
```

### TensorFlowEstimator

#### Each classifier and regressor have following fields

`n_classes=0` (Regressor), `n_classes` are expected to be input (Classifiers)  
`batch_size=32,`  
`steps=200, // except`  
`TensorFlowRNNClassifier - there is 50`  
`optimizer='Adagrad',`  
`learning_rate=0.1,`

## TesorFlow Cheat Sheet

# Keras

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Chollet explained that Keras was conceived to be an interface rather than an end-to-end machine-learning framework. It presents a higher-level, more intuitive set of abstractions that make it easy to configure neural networks regardless of the backend scientific computing library.

## Python For Data Science Cheat Sheet

### Keras

Learn Python for data science interactively at [www.DataCamp.com](#)



#### Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

##### A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
 activation='relu',
 input_dim=100))
>>> model.add(Dense(1,activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
 loss='binary_crossentropy',
 metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

#### Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

#### Keras Data Sets

```
>>> from keras.datasets import boston_housing,
 mnist,
 cifar10,
 imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

#### Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data[:,8]
```

#### Preprocessing

##### Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

##### One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

## Model Architecture

### Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

### Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
 input_dim=8,
 kernel_initializer='uniform',
 activation='relu'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

### Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

### Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Also see NumPy & Scikit-Learn

## Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
 y,
 test_size=0.33,
 random_state=42)
```

## Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

## Inspect Model

```
>>> model.output_shape
>>> model.summary()
>>> model.get_config()
>>> model.get_weights()
```

Model output shape  
Model summary representation  
Model configuration  
List all weight tensors in the model

## Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
 loss='binary_crossentropy',
 metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
 loss='categorical_crossentropy',
 metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
 loss='mse',
 metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
 optimizer='adam',
 metrics=['accuracy'])
```

## Model Training

```
>>> model3.fit(x_train4,
 y_train4,
 batch_size=32,
 epochs=15,
 verbose=1,
 validation_data=(x_test4,y_test4))
```

## Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
 y_test,
 batch_size=32)
```

## Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4, batch_size=32)
```

## Save/Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

## Model Fine-tuning

### Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(100.0, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
 optimizer=opt,
 metrics=['accuracy'])
```

## Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
 y_train4,
 batch_size=32,
 epochs=15,
 validation_data=(x_test4,y_test4),
 callbacks=[early_stopping_monitor])
```



# Numpy

NumPy targets the [CPython](#) reference implementation of Python, which is a non-optimizing [bytecode](#) interpreter. Mathematical algorithms written for this version of Python often run much slower than [compiled](#) equivalents. NumPy address the slowness problem partly by providing multidimensional arrays and functions and operators that operate efficiently on arrays, requiring rewriting some code, mostly inner loops using NumPy.

## Python For Data Science Cheat Sheet

### NumPy Basics

Learn Python for Data Science [Interactively](#) at [www.DataCamp.com](#)

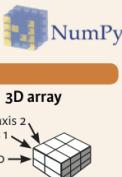


### NumPy

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))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)
>>> np.linspace(0,2,9)
>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros  
Create an array of ones  
Create an array of evenly spaced values (step value)  
Create an array of evenly spaced values (number of samples)  
Create a constant array  
Create a 2x2 identity matrix  
Create an array with random values  
Create an empty array

### I/O

#### Saving & Loading On Disk

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

|                          |                                                                      |
|--------------------------|----------------------------------------------------------------------|
| <code>np.int64</code>    | Signed 64-bit integer types                                          |
| <code>np.float32</code>  | Standard double-precision floating point                             |
| <code>np.complex</code>  | Complex numbers represented by 128 floats                            |
| <code>np.bool</code>     | Boolean type storing <code>TRUE</code> and <code>FALSE</code> values |
| <code>np.object</code>   | Python object type                                                   |
| <code>np.string_</code>  | Fixed-length string type                                             |
| <code>np.unicode_</code> | Fixed-length unicode type                                            |

### Inspecting Your Array

|                            |                                      |
|----------------------------|--------------------------------------|
| <code>a.shape</code>       | Array dimensions                     |
| <code>a.len()</code>       | Length of array                      |
| <code>a.ndim</code>        | Number of array dimensions           |
| <code>a.size</code>        | Number of array elements             |
| <code>a.dtype</code>       | Data type of array elements          |
| <code>a.dtype.name</code>  | Name of data type                    |
| <code>a.astype(int)</code> | Convert an array to a different type |

### Asking For Help

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

### Array Mathematics

#### Arithmetic Operations

|                                                             |                                |
|-------------------------------------------------------------|--------------------------------|
| <code>a - b</code>                                          | Subtraction                    |
| <code>np.subtract(a,b)</code>                               | Addition                       |
| <code>a + b</code>                                          | Division                       |
| <code>np.add(b,a)</code>                                    | Multiplication                 |
| <code>a / b</code>                                          | Exponentiation                 |
| <code>np.divide(a,b)</code>                                 | Square root                    |
| <code>a * b</code>                                          | Print sines of an array        |
| <code>np.multiply(a,b)</code>                               | Element-wise cosine            |
| <code>np.sqrt(b)</code>                                     | Element-wise natural logarithm |
| <code>np.sin(a)</code>                                      | Dot product                    |
| <code>np.cos(b)</code>                                      |                                |
| <code>np.log(a)</code>                                      |                                |
| <code>e.dot(f)</code>                                       |                                |
| <code>array([[ 1.5,  4.,  9.,  1.,  4., 10., 18. ]])</code> |                                |

#### Comparison

|                                                                              |                         |
|------------------------------------------------------------------------------|-------------------------|
| <code>a == b</code>                                                          | Element-wise comparison |
| <code>array([[False, True, True], [False, False, False]], dtype=bool)</code> |                         |
| <code>a &lt; 2</code>                                                        | Element-wise comparison |
| <code>array([True, False, False], dtype=bool)</code>                         |                         |
| <code>np.array_equal(a, b)</code>                                            | Array-wise comparison   |

#### Aggregate Functions

|                               |                                |
|-------------------------------|--------------------------------|
| <code>a.sum()</code>          | Array-wise sum                 |
| <code>a.min()</code>          | Array-wise minimum value       |
| <code>b.max(axis=0)</code>    | Maximum value of an array row  |
| <code>b.cumsum(axis=1)</code> | Cumulative sum of the elements |
| <code>a.mean()</code>         | Mean                           |
| <code>b.median()</code>       | Median                         |
| <code>a.correlcoef()</code>   | Correlation coefficient        |
| <code>np.std(b)</code>        | Standard deviation             |

### Copying Arrays

|                           |                                               |
|---------------------------|-----------------------------------------------|
| <code>h = a.view()</code> | Create a view of the array with the same data |
| <code>np.copy(a)</code>   | Create a copy of the array                    |
| <code>h = a.copy()</code> | Create a deep copy of the array               |

### Sorting Arrays

|                             |                                      |
|-----------------------------|--------------------------------------|
| <code>a.sort()</code>       | Sort an array                        |
| <code>c.sort(axis=0)</code> | Sort the elements of an array's axis |

### Subsetting, Slicing, Indexing

Also see [Lists](#)

|                                                         |                                                                            |
|---------------------------------------------------------|----------------------------------------------------------------------------|
| <code>a[2]</code>                                       | Select the element at the 2nd index                                        |
| <code>3</code>                                          | Select the element at row 0 column 2 (equivalent to <code>b[1][2]</code> ) |
| <code>b[1,2]</code>                                     | Select items at index 0 and 1                                              |
| <code>6,0</code>                                        | Select items at rows 0 and 1 in column 1                                   |
| <code>b[1,:]</code>                                     | Select all items at row 0 (equivalent to <code>b[0:1, :]</code> )          |
| <code>array([1, 2, 3, 4, 5, 6])</code>                  | Same as <code>[1, :, :]</code>                                             |
| <code>a[1,...]</code>                                   | Reversed array <code>a</code>                                              |
| <code>array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])</code> | Select elements from <code>a</code> less than 2                            |
| <code>a[&lt;1]</code>                                   | Select elements <code>(1,0, 0,1, (1,2) and (0,0)</code>                    |
| <code>b[[1, 0, 1, 0][1], [0, 1, 2, 0]]</code>           | Select a subset of the matrix's rows and columns                           |

### Array Manipulation

|                                                                                                        |                                               |
|--------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| <code>i = np.transpose(b)</code>                                                                       | Permute array dimensions                      |
| <code>i.T</code>                                                                                       | Permute array dimensions                      |
| <code>b.ravel()</code>                                                                                 | Flatten the array                             |
| <code>g.reshape(3,-2)</code>                                                                           | Reshape, but don't change data                |
| <code>h.resize((2,6))</code>                                                                           | Return a new array with shape (2,6)           |
| <code>h.append(g)</code>                                                                               | Append items to an array                      |
| <code>h.insert(a, 1, 5)</code>                                                                         | Insert items in an array                      |
| <code>h.delete(a, [1])</code>                                                                          | Delete items from an array                    |
| <code>np.concatenate((a,d),axis=0)</code>                                                              | Concatenate arrays                            |
| <code>array([ 1,  2,  3, 10, 15, 20])</code>                                                           | Stack arrays vertically (row-wise)            |
| <code>np.vstack((a,b))</code>                                                                          | Stack arrays vertically (row-wise)            |
| <code>array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])</code> | Stack arrays horizontally (column-wise)       |
| <code>np.column_stack((a,d))</code>                                                                    | Create stacked column-wise arrays             |
| <code>array([[ 1, 10], [ 2, 20], [ 3, 20]])</code>                                                     | Create stacked column-wise arrays             |
| <code>np.c_[a,d]</code>                                                                                | Split the array horizontally at the 3rd index |
| <code>np.hsplit(a,3)</code>                                                                            | Split the array vertically at the 2nd index   |

DataCamp  
Learn Python for Data Science [Interactively](#)

# Pandas

The name ‘Pandas’ is derived from the term “[panel data](#)”, an [econometrics](#) term for multidimensional structured data sets.

## 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

Columns

|   | Country | Capital   | Population |
|---|---------|-----------|------------|
| 1 | Belgium | Brussels  | 11190846   |
| 2 | India   | New Delhi | 1303171035 |
| 3 | Brazil  | Brasilia  | 207847528  |

Index

A two-dimensional labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
 'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
 'Population': [11190846, 1303171035, 207847528]}
```

```
>>> df = pd.DataFrame(data,
 columns=['Country', 'Capital', 'Population'])
```

### I/O

#### Read and Write to CSV

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

#### Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
 Read multiple sheets from the same file
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

### Asking For Help

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

### Selection

Also see [NumPy Arrays](#)

#### Getting

```
>>> s['b']
-5
>>> df[1]
 Country Capital Population
1 India New Delhi 1303171035
2 Brazil Brasilia 207847528
```

Get one element

Get subset of a DataFrame

### Selecting, Boolean Indexing & Setting

#### By Position

```
>>> df.iloc[[0], [0]]
'Belgium'
>>> df.iat[[0], [0]]
'Belgium'
```

Select single value by row & column

#### By Label

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

Select single value by row & column labels

#### By Label/Position

```
>>> df.ix[2]
 Country Brazil
 Capital Brasilia
 Population 207847528
```

Select single row of subset of rows

```
>>> df.ix[:, 'Capital']
```

Select a single column of subset of columns

```
>>> df.ix[1, 'Capital']
```

Select rows and columns

#### Boolean Indexing

```
>>> s[-(s > 1)]
>>> s[(s < -1) | (s > 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')
>>> s.order_()
>>> df.rank()
```

Sort by row or column index  
Sort a series by its values  
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()
Cumulative 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)
```

DataCamp

Learn Python for Data Science [Interactively](#)



## Pandas Cheat Sheet

# Data Wrangling

The term “data wrangler” is starting to infiltrate pop culture. In the 2017 movie [Kong: Skull Island](#), one of the characters, played by actor [Marc Evan Jackson](#) is introduced as “Steve Woodward, our data wrangler”.

# Data Wrangling

## with pandas

### Cheat Sheet

<http://pandas.pydata.org>

#### Syntax – Creating DataFrames

|   | a | b | c  |
|---|---|---|----|
| 1 | 4 | 7 | 10 |
| 2 | 5 | 8 | 11 |
| 3 | 6 | 9 | 12 |

```
df = pd.DataFrame(
 {"a": [4, 5, 6],
 "b": [7, 8, 9],
 "c": [10, 11, 12]},
 index = [1, 2, 3])
Specify values for each column.
```

```
df = pd.DataFrame(
 [[4, 7, 10],
 [5, 8, 11],
 [6, 9, 12]],
 index=[1, 2, 3],
 columns=['a', 'b', 'c'])
Specify values for each row.
```

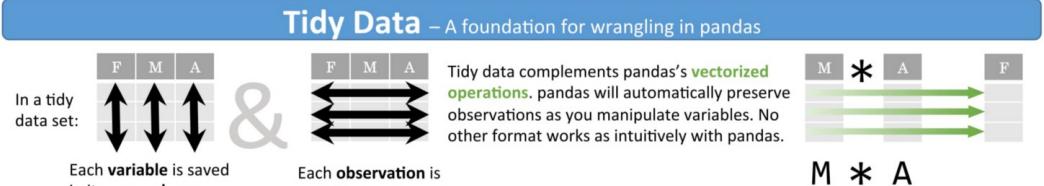
|   | a | b | c  |
|---|---|---|----|
| n | v |   |    |
| d | 1 | 4 | 7  |
| e | 2 | 5 | 10 |
| f | 2 | 6 | 9  |
| g | 2 | 7 | 11 |

```
df = pd.DataFrame(
 {"a": [4, 5, 6],
 "b": [7, 8, 9],
 "c": [10, 11, 12]},
 index = pd.MultiIndex.from_tuples(
 [('d',1),('d',2),('e',2)],
 names=['n','v']))
Create DataFrame with a MultiIndex
```

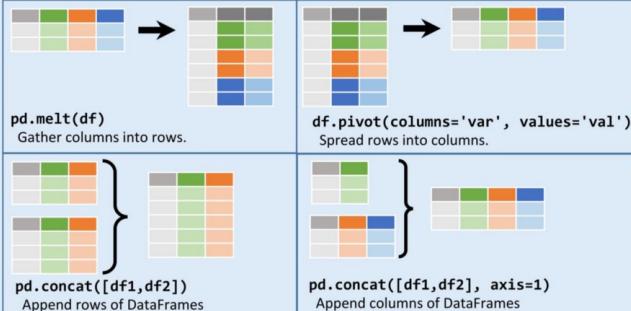
#### Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
 .rename(columns={'variable': 'var',
 'value': 'val'})
 .query('val >= 200')
)
```



#### Reshaping Data – Change the layout of a data set



```
df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
Order rows by values of a column (high to low).

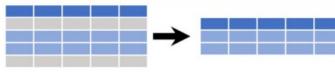
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(['Length', 'Height'], axis=1)
Drop columns from DataFrame
```

#### Subset Observations (Rows)



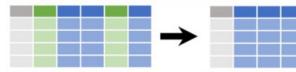
```
df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.
```

#### Subset Variables (Columns)



```
df[['width', 'length', 'species']]
Select multiple columns with specific names.

df['width'] or df.width
Select single column with specific name.

df.filter(regex='regex')
Select columns whose name matches regular expression regex.
```

##### regex (Regular Expressions) Examples

|                      |                                                              |
|----------------------|--------------------------------------------------------------|
| '.'                  | Matches strings containing a period.'                        |
| 'Length\$'           | Matches strings ending with word 'Length'                    |
| '^Sepal'             | Matches strings beginning with the word 'Sepal'              |
| '^x[1-5]\$'          | Matches strings beginning with 'x' and ending with 1,2,3,4,5 |
| '^^(?!Species\$).*'' | Matches strings except the string 'Species'                  |

```
df.loc[:, 'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:, [1, 2, 5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns.
```

#### Logic in Python (and pandas)

|    |                        |                                |                                     |
|----|------------------------|--------------------------------|-------------------------------------|
| <  | Less than              | !=                             | Not equal to                        |
| >  | Greater than           | df.column.isin(values)         | Group membership                    |
| == | Equals                 | pd.isnull(obj)                 | Is NaN                              |
| <= | Less than or equals    | pd.notnull(obj)                | Is not NaN                          |
| >= | Greater than or equals | &,  , ~, ^, df.any(), df.all() | Logical and, or, not, xor, any, all |

#### Data Wrangling Cheat Sheet

<http://pandas.pydata.org/> This cheat sheet inspired by RStudio Data Wrangling Cheatsheet (<https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>) Written by Irv Lustig, Princeton Consultants

## Summarize Data

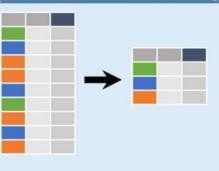
```
df['w'].value_counts()
Count number of rows with each unique value of variable
len(df)
of rows in DataFrame.
df['w'].nunique()
of distinct values in a column.
df.describe()
Basic descriptive statistics for each column (or GroupBy)
```



pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

```
sum() Sum values of each object.
count() Count non-NA/null values of each object.
median() Median value of each object.
quantile([0.25,0.75]) Quantiles of each object.
apply(function) Apply function to each object.
```

## Group Data



```
df.groupby(by="col")
Return a GroupBy object, grouped by values in column named "col".
df.groupby(level="ind")
Return a GroupBy object, grouped by values in index level named "ind".
```

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

```
size() Size of each group.
agg(function) Aggregate group using function.
```

## Windows

```
df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.
```

## Handling Missing Data

```
df.dropna()
Drop rows with any column having NA/null data.
df.fillna(value)
Replace all NA/null data with value.
```

## Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`  
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`  
Add single column.  
`pd.qcut(df.col, n, labels=False)`  
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

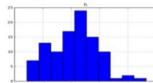
```
max(axis=1) min(axis=1)
Element-wise max. Element-wise min.
clip(lower=-10,upper=10) abs()
Trim values at input thresholds Absolute value.
```

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

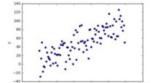
|                                    |                               |
|------------------------------------|-------------------------------|
| <code>shift(1)</code>              | <code>shift(-1)</code>        |
| Copy with values shifted by 1.     | Copy with values lagged by 1. |
| <code>rank(method='dense')</code>  | <code>cumsum()</code>         |
| Ranks with no gaps.                | Cumulative sum.               |
| <code>rank(method='min')</code>    | <code>cummax()</code>         |
| Ranks. Ties get min rank.          | Cumulative max.               |
| <code>rank(pct=True)</code>        | <code>cummin()</code>         |
| Ranks rescaled to interval [0, 1]. | Cumulative min.               |
| <code>rank(method='first')</code>  | <code>cumprod()</code>        |
| Ranks. Ties go to first value.     | Cumulative product.           |

## Plotting

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

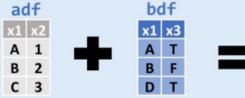


`df.plot.scatter(x='w',y='h')`  
Scatter chart using pairs of points



## Pandas Data Wrangling Cheat Sheet

## Combine Data Sets



Standard Joins

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

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

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

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

### Filtering Joins

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

x1 x2

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



### Set-like Operations

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

x1 x2

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

x1 x2

```
pd.merge(ydf, zdf, how='outer',
 indicator=True)
.query('_merge == "left_only"')
.drop(['_merge'],axis=1)
Rows that appear in ydf but not zdf (Setdiff).
```



## Ultimate Guide to Leveraging NLP & Machine Learning for your Chatbot

Code Snippets and Github Included  
chatbotslife.com

# Data Wrangling with dplyr and tidyr

# Data Wrangling with dplyr and tidyr

Cheat Sheet



## Syntax - Helpful conventions for wrangling

`dplyr::tbl_df(iris)`

Converts data to `tbl` class. `tbl`'s are easier to examine than data frames. R displays only the data that fits onscreen:

```
Source: local data frame [150 x 5]
 Sepal.Length Sepal.Width Petal.Length
1 5.1 3.5 1.4
2 4.9 3.0 1.4
3 4.7 3.2 1.3
4 4.6 3.1 1.5
5 5.0 3.6 1.4
.. ...
Variables not shown: Petal.Width (dbl), Species (fctr)
```

`dplyr::glimpse(iris)`

Information dense summary of `tbl` data.

`utils::View(iris)`

View data set in spreadsheet-like display (note capital V).

|    | Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|----|--------------|-------------|--------------|-------------|---------|
| 1  | 5.1          | 3.5         | 1.4          | 0.2         | setosa  |
| 2  | 4.9          | 3.0         | 1.4          | 0.2         | setosa  |
| 3  | 4.7          | 3.2         | 1.3          | 0.2         | setosa  |
| 4  | 4.6          | 3.1         | 1.5          | 0.2         | setosa  |
| 5  | 5.0          | 3.6         | 1.4          | 0.2         | setosa  |
| .. | ...          | ...         | ...          | ...         | ...     |
| 8  | 5.0          | 3.4         | 1.5          | 0.2         | setosa  |

`dplyr::%>%`

Passes object on left hand side as first argument (or . argument) of function on righthand side.

```
x %>% f(y) is the same as f(x, y)
y %>% f(x, ., z) is the same as f(x, y, z)
```

"Piping" with `%>%` makes code more readable, e.g.

```
iris %>%
 group_by(Species) %>%
 summarise(avg = mean(Sepal.Width)) %>%
 arrange(avg)
```

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

## Tidy Data - A foundation for wrangling in R

In a tidy data set:

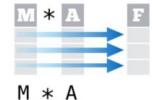


Each variable is saved in its own column

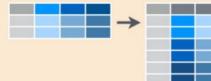


Each observation is saved in its own row

Tidy data complements R's **vectorized operations**. R will automatically preserve observations as you manipulate variables. No other format works as intuitively with R.



## Reshaping Data - Change the layout of a data set



`tidy::gather(cases, "year", "n", 2:4)`

Gather columns into rows.



`tidy::spread(pollution, size, amount)`

Spread rows into columns.



`tidy::separate(storms, date, c("y", "m", "d"))`

Separate one column into several.



`tidy::unite(data, col, ..., sep)`

Unite several columns into one.

`dplyr::data_frame(a = 1:3, b = 4:6)`

Combine vectors into data frame (optimized).

`dplyr::arrange(mtcars, mpg)`

Order rows by values of a column (low to high).

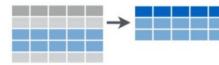
`dplyr::arrange(mtcars, desc(mpg))`

Order rows by values of a column (high to low).

`dplyr::rename(tb, y = year)`

Rename the columns of a data frame.

## Subset Observations (Rows)



`dplyr::filter(iris, Sepal.Length > 7)`

Extract rows that meet logical criteria.

`dplyr::distinct(iris)`

Remove duplicate rows.

`dplyr::sample_frac(iris, 0.5, replace = TRUE)`

Randomly select fraction of rows.

`dplyr::sample_n(iris, 10, replace = TRUE)`

Randomly select n rows.

`dplyr::slice(iris, 10:15)`

Select rows by position.

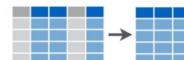
`dplyr::top_n(storms, 2, date)`

Select and order top n entries (by group if grouped data).

### Logic in R - ?Comparison, ?base::Logic

|                    |                          |                                      |                   |
|--------------------|--------------------------|--------------------------------------|-------------------|
| <                  | Less than                | <code>!=</code>                      | Not equal to      |
| >                  | Greater than             | <code>%in%</code>                    | Group membership  |
| <code>==</code>    | Equal to                 | <code>is.na</code>                   | Is NA             |
| <code>&lt;=</code> | Less than or equal to    | <code>!is.na</code>                  | Is not NA         |
| <code>&gt;=</code> | Greater than or equal to | <code>&amp;, !, xor, any, all</code> | Boolean operators |

## Subset Variables (Columns)



`dplyr::select(iris, Sepal.Width, Petal.Length, Species)`

Select columns by name or helper function.

### Helper functions for select - ?select

`select(iris, contains("."))`

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches("t.*"))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of(c("Species", "Genus")))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepal"))`

Select columns whose name starts with a character string.

`select(iris, Sepal.Length:Petal.Width)`

Select all columns between Sepal.Length and Petal.Width (inclusive).

`select(iris, -Species)`

Select all columns except Species.

## Data Wrangling with dplyr and tidyr Cheat Sheet

devtools::install\_github("rstudio/EDAWR") for data sets

Learn more with [browseVignettes\(package = c\("dplyr", "tidyr"\)\)](#) • dplyr 0.4.0 • tidyr 0.2.0 • Updated: 1/15

### Summarise Data

`dplyr::summarise(iris, avg = mean(Sepal.Length))`  
Summarise data into single row of values.

`dplyr::summarise_each(iris, funs(mean))`  
Apply summary function to each column.

`dplyr::count(iris, Species, wt = Sepal.Length)`  
Count number of rows with each unique value of variable (with or without weights).

Summarise uses **summary functions**, functions that take a vector of values and return a single value, such as:

|                                   |                                 |
|-----------------------------------|---------------------------------|
| <code>dplyr::first</code>         | <code>min</code>                |
| First value of a vector.          | Minimum value in a vector.      |
| <code>dplyr::last</code>          | <code>max</code>                |
| Last value of a vector.           | Maximum value in a vector.      |
| <code>dplyr::nth</code>           | <code>mean</code>               |
| Nth value of a vector.            | Mean value of a vector.         |
| <code>dplyr::n</code>             | <code>median</code>             |
| # of values in a vector.          | Median value of a vector.       |
| <code>dplyr::n_distinct</code>    | <code>var</code>                |
| # of distinct values in a vector. | Variance of a vector.           |
| <code>IQR</code>                  | <code>sd</code>                 |
| IQR of a vector.                  | Standard deviation of a vector. |

### Make New Variables

`dplyr::mutate(iris, sepal = Sepal.Length + Sepal.Width)`  
Compute and append one or more new columns.

`dplyr::mutate_each(iris, funs(min_rank))`  
Apply window function to each column.

`dplyr::transmute(iris, sepal = Sepal.Length + Sepal.Width)`  
Compute one or more new columns. Drop original columns.

Mutate uses **window functions**, functions that take a vector of values and return another vector of values, such as:

|                                  |                               |
|----------------------------------|-------------------------------|
| <code>dplyr::lead</code>         | <code>dplyr::cumall</code>    |
| Copy with values shifted by 1.   | Cumulative <code>all</code>   |
| <code>dplyr::lag</code>          | <code>dplyr::cumany</code>    |
| Copy with values lagged by 1.    | Cumulative <code>any</code>   |
| <code>dplyr::dense_rank</code>   | <code>dplyr::cummean</code>   |
| Ranks with no gaps.              | Cumulative <code>mean</code>  |
| <code>dplyr::min_rank</code>     | <code>cumsum</code>           |
| Ranks. Ties get min rank.        | Cumulative <code>sum</code>   |
| <code>dplyr::percent_rank</code> | <code>cummax</code>           |
| Ranks rescaled to [0, 1].        | Cumulative <code>max</code>   |
| <code>dplyr::row_number</code>   | <code>cummin</code>           |
| Ranks. Ties got to first value.  | Cumulative <code>min</code>   |
| <code>dplyr::ntile</code>        | <code>cumprod</code>          |
| Bin vector into n buckets.       | Cumulative <code>prod</code>  |
| <code>dplyr::between</code>      | <code>pmax</code>             |
| Are values between a and b?      | Element-wise <code>max</code> |
| <code>dplyr::cume_dist</code>    | <code>pmin</code>             |
| Cumulative distribution.         | Element-wise <code>min</code> |

### Combine Data Sets

`dplyr::left_join(a, b, by = "x1")`  
Join matching rows from b to a.

`dplyr::right_join(a, b, by = "x1")`  
Join matching rows from a to b.

`dplyr::inner_join(a, b, by = "x1")`  
Join data. Retain only rows in both sets.

`dplyr::full_join(a, b, by = "x1")`  
Join data. Retain all values, all rows.

**Mutating Joins**

|                           |                   |                  |                                   |
|---------------------------|-------------------|------------------|-----------------------------------|
| <code>x1   x2   x3</code> | <code>a</code>    | <code>+ b</code> | <code>=</code>                    |
| A 1 T<br>B 2 F<br>C 3 NA  | A 1<br>B 2<br>C 3 |                  | A 1 T<br>B 2 F<br>C 3 NA<br>D 4 T |

**Filtering Joins**

|                      |                   |                  |                          |
|----------------------|-------------------|------------------|--------------------------|
| <code>x1   x2</code> | <code>y</code>    | <code>+ z</code> | <code>=</code>           |
| A 1<br>B 2           | A 1<br>B 2<br>C 3 |                  | A 1<br>B 2<br>C 3<br>D 4 |

**Set Operations**

|                      |                          |                |                          |
|----------------------|--------------------------|----------------|--------------------------|
| <code>y</code>       | <code>z</code>           | <code>+</code> | <code>=</code>           |
| <code>x1   x2</code> | <code>x1   x2</code>     |                |                          |
| A 1<br>B 2<br>C 3    | A 1<br>B 2<br>C 3<br>D 4 |                | A 1<br>B 2<br>C 3<br>D 4 |

`dplyr::intersect(y, z)`  
Rows that appear in both y and z.

`dplyr::union(y, z)`  
Rows that appear in either or both y and z.

`dplyr::setdiff(y, z)`  
Rows that appear in y but not z.

**Binding**

|                      |                          |                  |                          |
|----------------------|--------------------------|------------------|--------------------------|
| <code>x1   x2</code> | <code>y</code>           | <code>+ z</code> | <code>=</code>           |
| A 1<br>B 2<br>C 3    | A 1<br>B 2<br>C 3<br>D 4 |                  | A 1<br>B 2<br>C 3<br>D 4 |

`dplyr::bind_rows(y, z)`  
Append z to y as new rows.

`dplyr::bind_cols(y, z)`  
Append z to y as new columns.  
Caution: matches rows by position.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

devtools::install\_github("rstudio/EDAWR") for data sets

Learn more with `browseVignettes(package = c("dplyr", "tidy"))` • dplyr 0.4.0 • tidy 0.2.0 • Updated: 1/15

## Data Wrangling with dplyr and tidyR Cheat Sheet

# Scipy

SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy, and an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as MATLAB, GNU Octave, and Scilab. The NumPy stack is also sometimes referred to as the SciPy stack.[3]

## Python For Data Science Cheat Sheet

### SciPy - Linear Algebra

Learn More Python for Data Science [Interactively](#) at [www.datacamp.com](http://www.datacamp.com)



#### SciPy

The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.



#### Interacting With NumPy

[Also see NumPy](#)

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j),2j,3j], [4j,5j,6j])
>>> c = np.array([(1,5,2,3), (4,5,6)], [(3,2,1), (4,5,6)])
```

#### Index Tricks

```
>>> np.mgrid[0:5,0:5]
>>> np.ogrid[0:2,0:2]
>>> np.r_[3,[0]*5,-1:1:10j]
>>> np.c_[b,c]
```

Create a dense meshgrid  
Create an open meshgrid  
Stack arrays vertically (row-wise)  
Create stacked column-wise arrays

#### Shape Manipulation

```
>>> np.transpose(b)
>>> b.flatten()
>>> np.vstack((b,c))
>>> np.vstack((a,b))
>>> np.hsplit(c,2)
>>> np.vsplit(d,4)
```

Permute array dimensions  
Flatten the array  
Stack arrays horizontally (column-wise)  
Stack arrays vertically (row-wise)  
Split the array horizontally at the 2nd index  
Split the array vertically at the 2nd index

#### Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])
```

Create a polynomial object

#### Vectorizing Functions

```
>>> def myfunc(a):
... if a < 0:
... return a**2
... else:
... return a/2
>>> np.vectorize(myfunc)
```

Vectorize functions

#### Type Handling

```
>>> np.real(b)
Return the real part of the array elements
>>> np.imag(b)
Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)
Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)
```

Cast object to a data type

#### Other Useful Functions

```
>>> np.angle(b,deg=True)
Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)
Create an array of evenly spaced values
(number of samples)
>>> g[3:] += np.pi
>>> np.unwrap(g)
Unwrap
>>> np.logspace(0,10,3)
Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])
Return values from a list of arrays depending on
conditions
>>> misc.factorial(a)
Factorial
>>> misc.comb(10,3,exact=True)
Combine N things taken at k time
>>> misc.central_diff_weights(3)
Weights for N-point central derivative
>>> misc.derivative(myfunc,i,0)
Find the n-th derivative of a function at a point
```

Return the angle of the complex argument  
Create an array of evenly spaced values  
(number of samples)  
Unwrap  
Create an array of evenly spaced values (log scale)  
Return values from a list of arrays depending on  
conditions  
Factorial  
Combine N things taken at k time  
Weights for N-point central derivative  
Find the n-th derivative of a function at a point

## Linear Algebra

You'll use the linalg and sparse modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

#### Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(B)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

#### Basic Matrix Routines

|         |         |
|---------|---------|
| Inverse | Inverse |
|---------|---------|

```
>>> A.I
>>> linalg.inv(A)
```

#### Transposition

```
>>> A.T
```

```
>>> A.H
```

#### Trace

```
>>> np.trace(A)
```

#### Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

#### Rank

```
>>> np.linalg.matrix_rank(C)
```

#### Determinant

```
>>> linalg.det(A)
```

#### Solving linear problems

```
>>> linalg.solve(A,b)
```

```
>>> E = np.mat(a).T
```

```
>>> linalg.lstsq(F,E)
```

#### Generalized inverse

```
>>> linalg.pinv(C)
```

```
>>> linalg.pinv2(C)
```

Solver for dense matrices  
Solver for dense matrices  
Least-squares solution to linear matrix equation

Compute the pseudo-inverse of a matrix  
(least-squares solver)

Compute the pseudo-inverse of a matrix  
(SVD)

#### Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)
```

```
>>> G = np.mat(np.identity(2))
```

```
>>> C[C > 0.5] = 0
```

```
>>> H = sparse.csr_matrix(C)
```

```
>>> I = sparse.csc_matrix(D)
```

```
>>> J = sparse.dok_matrix(A)
```

```
>>> E.todense()
```

```
>>> sparse.isspmatrix_csc(A)
```

Create a 2x2 identity matrix  
Create a 2x2 identity matrix

Compressed Sparse Row matrix

Compressed Sparse Column matrix

Dictionary Of Keys matrix

Sparse matrix to full matrix

Identify sparse matrix

#### Sparse Matrix Routines

##### Inverse

```
>>> sparse.linalg.inv(I)
```

##### Norm

```
>>> sparse.linalg.norm(I)
```

##### Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Solver for sparse matrices

#### Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)
```

Sparse matrix exponential

#### Asking For Help

```
>>> help(scipy.linalg.diagsvd)
```

```
>>> np.info(np.matrix)
```

[Also see NumPy](#)

## Matrix Functions

### Addition

```
>>> np.add(A,D)
```

### Subtraction

```
>>> np.subtract(A,D)
```

### Division

```
>>> np.divide(A,D)
```

### Multiplication

```
>>> A @ D
```

```
>>> np.multiply(D,A)
```

```
>>> np.dot(A,D)
```

```
>>> np.vdot(A,D)
```

```
>>> np.inner(A,D)
```

```
>>> np.outer(A,D)
```

```
>>> np.tensordot(A,D)
```

```
>>> np.kron(A,D)
```

### Exponential Functions

```
>>> linalg.expm(A)
```

```
>>> np.expm2(A)
```

```
>>> np.expm3(D)
```

### Logarithm Function

```
>>> linalg.logm(A)
```

### Trigonometric Functions

```
>>> linalg.sinm(D)
```

```
>>> linalg.cosm(D)
```

```
>>> linalg.tanm(A)
```

### Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
```

```
>>> linalg.coshm(D)
```

```
>>> linalg.tanhm(A)
```

### Matrix Sign Function

```
>>> np.sigmm(A)
```

### Matrix Square Root

```
>>> linalg.sqrtm(A)
```

### Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)
```

### Decompositions

#### Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
```

```
>>> l1, l2 = la
```

```
>>> v[:,0]
```

```
>>> v[:,1]
```

```
>>> linalg.eigvals(A)
```

#### Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
```

```
>>> M,N = B.shape
```

```
>>> Sg = linalg.diagsvd(s,M,N)
```

#### LU Decomposition

```
>>> P,L,U = linalg.lu(C)
```

#### LU Decomposition

### Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)
```

```
>>> sparse.linalg.svds(H, 2)
```

Eigenvalues and eigenvectors

SVD

**DataCamp**  
Learn Python for Data Science [Interactively](#)



## Scipy Cheat Sheet

# Matplotlib

**matplotlib** is a plotting library for the **Python** programming language and its numerical mathematics extension **NumPy**. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like **Tkinter**, **wxPython**, **Qt**, or **GTK+**. There is also a procedural “pylab” interface based on a state machine (like OpenGL), designed to closely resemble that of **MATLAB**, though its use is discouraged.[2] **SciPy** makes use of **matplotlib**.

**pyplot** is a **matplotlib** module which provides a **MATLAB-like** interface.[6] **matplotlib** is designed to be as usable as **MATLAB**, with the ability to use **Python**, with the advantage that it is free.

## Python For Data Science Cheat Sheet

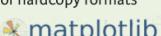
### Matplotlib

Learn Python Interactively at [www.DataCamp.com](http://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))
>>> X, Y = np.mgrid[-1:1:100j, -3:1:100j]
>>> 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

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> scatter(X,Y)
>>> axes[0,0].fill([(1,2,3), [3,4,5)])
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them  
Draw unconnected points, scaled or colored  
Plot vertical rectangles (constant width)  
Plot horizontal rectangles (constant height)  
Draw a horizontal line across axes  
Draw a vertical line across axes  
Draw filled polygons  
Fill between y-values and o

Draw points with lines or markers connecting them  
Draw unconnected points, scaled or colored  
Plot vertical rectangles (constant width)  
Plot horizontal rectangles (constant height)  
Draw a horizontal line across axes  
Draw a vertical line across axes  
Draw filled polygons  
Fill between y-values and o

#### Vector Fields

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

Add an arrow to the axes  
Plot a 2D field of arrows  
Plot a 2D field of arrows

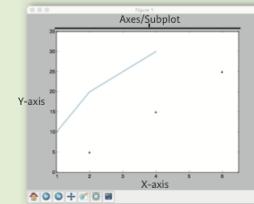
#### Data Distributions

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

Plot a histogram  
Make a box and whisker plot  
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] # Step 2
>>> fig = plt.figure() # Step 3
>>> 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='^') # Step 4
>>> ax.set_xlim(1, 6.5) # Step 4
>>> plt.savefig('foo.png') # Step 5
>>> plt.show() # Step 6
```

Figure

4) Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha=.4)
>>> ax.plot(x, y, c='r')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img, cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".") # Step 1
>>> ax.plot(x,y,marker="o") # Step 2
```

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"))
```

Subplot Spacing

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

Axis Spines

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

Mathtext

```
>>> plt.title(r'$\sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling	<pre>>>> ax.margins(x=0,y=0.1)</pre>	Add padding to a plot
	<pre>>>> ax.axis('equal')</pre>	Set the aspect ratio of the plot to 1
	<pre>>>> ax.set(xlim=[0,10.5], ylim=[-1.5,1.5])</pre>	Set limits for x and y-axis
	<pre>>>> ax.set_xlim(0,10.5)</pre>	Set limits for x-axis

Legends

```
>>> ax.set(title='An Example Axes', xlabel='Y-Axis', ylabel='X-Axis')
```

```
>>> ax.legend(loc='best')
```

Ticks	<pre>>>> ax.xaxis.set(ticks=range(1,5), ticklabels=[3,100,-12,"foo"])</pre>	Manually set x-ticks
	<pre>>>> ax.tick_params(axis='y', direction='inout', length=10)</pre>	Make y-ticks longer and go in and out

Subplot Spacing

	<pre>>>> fig3.subplots_adjust(wspace=0.5, hspace=0.3,</pre>	Adjust the spacing between subplots
	<pre>left=0.125, right=0.9, top=0.9, bottom=0.1)</pre>	

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible

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

Clear an axis

Clear the entire figure

Close a window

DataCamp

Learn Python for Data Science [Interactively](http://www.DataCamp.com)

Matplotlib Cheat Sheet

>>> If you like this list, you can let me know [here](#). <<<

Data Visualization

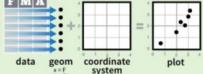
Data Visualization with ggplot2

Cheat Sheet

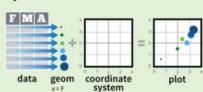


Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data** set, a set of **geoms**—visual marks that represent data points, and a **coordinate system**.



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, and **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

aesthetic mappings **data** **geom**
qplot(x = cty, y = hwy, color = cyl, data = mpg, geom = "point")
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

ggplot(data = mpg, aes(x = cty, y = hwy))

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

```
data
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  geom_smooth(method = "lm") +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()
```

add layers, elements with +
layer = geom +
default stat +
layer specific mappings
additional elements

Add a new layer to a plot with a **geom_***() or **stat_***() function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

last_plot()

Returns the last plot

ggsave("plot.png", width = 5, height = 5)

Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

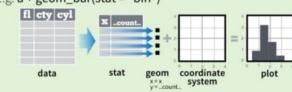
Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.		
<h4>One Variable</h4> <p>Continuous</p> <pre>a <- ggplot(mpg, aes(hwy)) a + geom_area(stat = "bin") x, y, alpha, color, fill, linetype, size b + geom_area(aes(y = ..density..), stat = "bin") a + geom_density(kernel = "gaussian") x, y, alpha, color, fill, linetype, size, weight b + geom_density(aes(y = ..count..)) a + geom_dotplot() x, y, alpha, color, fill a + geom_freqpoly() x, y, alpha, color, linetype, size b + geom_freqpoly(aes(y = ..density..)) a + geom_histogram(binwidth = 5) x, y, alpha, color, fill, linetype, size, weight b + geom_histogram(aes(y = ..density..))</pre> <p>Discrete</p> <pre>b <- ggplot(mpg, aes(f1)) b + geom_bar() x, alpha, color, fill, linetype, size, weight</pre>	<h4>Two Variables</h4> <p>Continuous X, Continuous Y</p> <pre>f + geom_blank() f + geom_jitter() x, y, alpha, color, fill, shape, size f + geom_point() x, y, alpha, color, fill, shape, size f + geom_quantile() x, y, alpha, color, linetype, size, weight f + geom_rug(sides = "bl") alpha, color, linetype, size f + geom_smooth(model = lm) x, y, alpha, color, fill, linetype, size, weight C f + geom_text(aes(label = cty)) x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust</pre> <p>Discrete X, Continuous Y</p> <pre>g <- ggplot(mpg, aes(class, hwy)) g + geom_bar(stat = "identity") x, y, alpha, color, fill, linetype, size, weight</pre> <p>Discrete X, Discrete Y</p> <pre>h <- ggplot(diamonds, aes(cut, color)) h + geom_jitter() x, y, alpha, color, fill, shape, size</pre>	<p>Continuous Bivariate Distribution</p> <pre>i <- ggplot(movies, aes(year, rating)) i + geom_bin2d(binwidth = c(5, 0.5)) xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight</pre> <p>Continuous Function</p> <pre>j <- ggplot(economics, aes(date, unemploy)) j + geom_area() x, y, alpha, color, fill, linetype, size j + geom_line() x, y, alpha, color, linetype, size j + geom_step(direction = "hv") x, y, alpha, color, linetype, size</pre> <p>Visualizing error</p> <pre>d1 <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2) k <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se)) k + geom_crossbar(fatten = 2) x, y, ymax, ymin, alpha, color, fill, linetype, size k + geom_errorbar() x, ymax, ymin, alpha, color, linetype, size, width (also geom_errorbarh()) k + geom_linerange() x, ymin, ymax, alpha, color, linetype, size k + geom_pointrange() x, y, ymin, ymax, alpha, color, fill, linetype, shape, size</pre> <p>Maps</p> <pre>data <- data.frame(murder = USArrests\$Murder, state = tolower(rrownames(USArrests))) map <- map_data("state") l <- ggplot(data, aes(murder)) l + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat) map_id, alpha, color, fill, linetype, size</pre>
<h4>Graphical Primitives</h4> <p>c <- ggplot(map, aes(long, lat))</p> <p>c + geom_polygon(aes(group = group)) x, y, alpha, color, fill, linetype, size</p> <p>d < ggplot(economics, aes(date, unemploy))</p> <p>d + geom_path(lineend="butt", linejoin="round", linemt=1) x, y, alpha, color, linetype, size</p> <p>d + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900)) x, ymax, ymin, alpha, color, fill, linetype, size</p> <p>e <- ggplot(seals, aes(x = long, y = lat))</p> <p>e + geom_segment(aes(xend = long + delta_long, yend = lat + delta_lat)) x, end, y, end, alpha, color, linetype, size</p> <p>e + geom_rect(aes(xmin = long, ymin = lat, xmax = long + delta_long, ymax = lat + delta_lat)) xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size</p>	<p>Three Variables</p> <pre>seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)) m <- ggplot(seals, aes(long, lat)) m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE) x, y, alpha, fill</pre> <p>m + geom_contour(aes(z = z)) x, y, z, alpha, colour, linetype, size, weight</p>	

Data Visualization Cheat Sheet

Learn more at docs.ggplot2.org • ggplot2 0.9.3.1 • Updated: 3/15

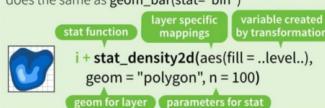
Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. `a + geom_bar(stat = "bin")`



Each stat creates additional variables to map aesthetics to. These variables use a common `.name..` syntax.

stats functions and geom functions both combine a stat with a geom to make a layer, i.e. `stat_bin(geom = "bar")` does the same as `geom_bar(stat = "bin")`



```
a + stat_bin(binwidth = 1, origin = 10)          1D distributions
x, y | .count, .n, .density, .ndensity...
a + stat_bindot(binwidth = 1, binaxis = "x")
x, y | .count, .n, ...
a + stat_density(adjust = 1, kernel = "gaussian")
x, y | .density, .scaled...
```

```
f + stat_bin2d(bins = 30, drop = TRUE)           2D distributions
x, y, fill | .count, ..., density...
f + stat_hex(bins = 30)
x, y, fill | .count, ..., density...
f + stat_contour(contour = TRUE, n = 100)
x, y, color, size | .level...
```

```
m + stat_contour(aes(z = z))                  3 Variables
x, y, z, order | .level...
m + stat_spoke(aes(radius = z, angle = z))
angle, radius, x, yend, y, yend | .x, .yend, ...
m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | .value...
m + stat_summary2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | .value...
```

```
g + stat_boxplot(coef = 1.5)                   Comparisons
x, y | .lower, .middle, .upper, .outliers...
g + stat_ydensity(adjust = 1, kernel = "gaussian", scale = "area")
x, y | .density, .scaled, .count, .n, .violinwidth, .width...
```

```
f + stat_ecdf(n = 40)                         Functions
x, y | .x, .y...
f + stat_quantile(quartiles = c(0.25, 0.5, 0.75), formula = y ~ log(x),
method = "rq")
x, y | .quartile, .x, .y...
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80,
fullrange = FALSE, level = 0.95)
x, y | .se, .x, .y, .ymin, .ymax...
```

```
ggplot() + stat_function(fun = dnorm, n = 101, args = list(sd=0.5))    General Purpose
x | .y...
f + stat_identity()
ggplot() + stat_qq(residuals = 1:100, distribution = qt,
dparams = list(d=5))
sample, x, y | .x, .y...
f + stat_sum()
x, y, size | .size...
f + stat_summary(fun.data = "mean_cl_boot")
f + stat_unique()
```

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com

Scales

Scales control how a plot maps data values to the visual values of an aesthetic. To change the mapping, add a custom scale.



General Purpose scales

Use with any aesthetic:
alpha, color, fill, linetype, shape, size

`scale_*`(continuous) - map cont' values to visual values
`scale_*`(discrete) - map discrete values to visual values
`scale_*`(identity) - use data values as visual values
`scale_*`(manual)(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

`scale_x_date`(labels = date_format("%m/%d"),
breaks = date_breaks("2 weeks")) - treat x values as dates. Use `tz` argument to specify time zone.

`scale_x_datetime`() - treat x values as date times. Use same arguments as `scale_x_date`().

`scale_x_log10`() - Plot x on log10 scale

`scale_x_reverse`() - Reverse direction of x axis

`scale_x_sqrt`() - Plot x on square root scale

Color and fill scales

Discrete 	Continuous
<code>n <- b + geom_bar(aes(fill = f))</code> <code>n + scale_fill_brewer(palette = "Blues")</code> <code>n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")</code>	<code>o <- a + geom_dotplot(aes(fill = ..x..))</code> <code>o + scale_fill_gradient(low = "red", high = "blue")</code> <code>o + scale_fill_gradient2(low = "red", high = "blue", mid = "white", midpoint = 25)</code> <code>o + scale_fill_gradientn(colors = terrain.colors(6))</code> <code>o + scale_fill_jitter()</code>

Shape scales

Manual shape values 	Shape scales
<code>p <- f + geom_point(aes(shape = f))</code> <code>p + scale_shape(solid = FALSE)</code> <code>p + scale_shape_manual(values = c(3:7))</code>	<code>o + scale_size_area(max = 6)</code> <code>o + scale_size_continuous(limits = c(0, 100), breaks = 10)</code>

Size scales

Size scales 	Size scales
<code>q <- f + geom_point(aes(size = cyl))</code> <code>q + scale_size_area(max = 6)</code>	<code>q + scale_size_continuous(limits = c(0, 100), breaks = 10)</code>

Coordinate Systems

`r <- b + geom_bar()`
`r + coord_cartesian(xlim = c(0, 5), ylim = c(0, 5))`

The default cartesian coordinate system
`r + coord_fixed(ratio = 1/2)`
Cartesian coordinates with fixed aspect ratio between x and y units

`r + coord_flip()`
Flipped Cartesian coordinates

`r + coord_polar(theta = "x", direction = 1)`
Polar coordinates

`r + coord_trans(ytrans = "sqrt")`
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

`r + coord_map(projection = "ortho", orientation = c(41, -74, 0))`
projection, orientation, xlim, ylim



Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

`s <- ggplot(mpg, aes(f, fill = drv))`

`s + geom_bar(position = "dodge")`
Arrange elements side by side

`s + geom_bar(position = "fill")`
Stack elements on top of one another, normalize height

`s + geom_bar(position = "stack")`
Stack elements on top of one another

`f + geom_point(position = "jitter")`
Add random noise to X and Y position of each element to avoid overplotting

Each position adjustment can be recast as a function with manual `width` and `height` arguments

`s + geom_bar(position = position_dodge(width = 1))`

Themes 	Themes
<code>r + theme_bw()</code> White background with grid lines	<code>r + theme_classic()</code> White background no gridlines
<code>r + theme_grey()</code> Grey background	<code>r + theme_minimal()</code> Minimal theme

ggthemes - Package with additional ggplot2 themes

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

`t + ggplot(mpg, aes(cty, hwy)) + geom_point()`

<code>t + facet_grid(~ fl)</code> facet into columns based on fl	<code>t + facet_grid(~ .)</code> facet into rows based on year
<code>t + facet_grid(year ~ fl)</code> facet into both rows and columns	<code>t + facet_wrap(~ fl)</code> wrap facets into a rectangular layout

Set `scales` to let axis limits vary across facets

`t + facet_grid(~ x, scales = "free")`
x and y axis limits adjust to individual facets

- "free_x" - x axis limits adjust
- "free_y" - y axis limits adjust

Set `labeler` to adjust facet labels

<code>t + facet_grid(~ fl, labeler = label_both)</code>	<code>fl: c fl: d fl: e fl: p fl: r</code>
<code>t + facet_grid(~ fl, labeler = label_bquote(alpha ^ .(x)))</code>	<code>alpha^c alpha^d alpha^e alpha^p alpha^r</code>
<code>t + facet_grid(~ fl, labeler = label_parsed)</code>	<code>c d e p r</code>

Labels

<code>t + ggtitle("New Plot Title")</code> Add a main title above the plot	<code>t + xlab("New X label")</code> Change the label on the X axis
<code>t + ylab("New Y label")</code> Change the label on the Y axis	<code>t + labs(title = "New title", x = "New x", y = "New y")</code> All of the above

Use scale functions to update legend labels

Legends

<code>t + theme(legend.position = "bottom")</code> Place legend at "bottom", "top", "left", or "right"	<code>t + guides(color = "none")</code> Set legend type for each aesthetic: colorbar, legend, or none (no legend)
<code>t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))</code> Set legend title and labels with a scale function.	

Zooming Without clipping (preferred)	<code>t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))</code>
With clipping (removes unseen data points)	<code>t + scale_x_continuous(limits = c(0, 100), breaks = 10)</code> <code>t + scale_y_continuous(limits = c(0, 100), breaks = 10)</code>

Learn more at docs.ggplot2.org • ggplot2 0.9.3.1 • Updated: 3/15

ggplot cheat sheet

PySpark

Python For Data Science Cheat Sheet

PySpark Basics

Learn Python for data science interactively at www.DataCamp.com



Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python



Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

<code>>>> sc.version</code>	Retrieve SparkContext version
<code>>>> sc.pythonVer</code>	Retrieve Python version
<code>>>> sc.master</code>	Master URL to connect to
<code>>>> str(sc.sparkHome)</code>	Path where Spark is installed on worker nodes
<code>>>> str(sc.sparkUser())</code>	Username of the Spark User running SparkContext
<code>>>> sc.appName</code>	Return application name
<code>>>> sc.applicationId</code>	Retrieve application ID
<code>>>> sc.defaultParallelism</code>	Return default level of parallelism
<code>>>> sc.defaultMinPartitions</code>	Default minimum number of partitions for RDDs

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
...     .setMaster("local")
...     .setAppName("My app")
...     .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Select which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',1),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([('a',[x,y,z]), 
...     ("b",[p,q,r])])
```

External Data

Read either one text file from HDFS, a local file system or any

Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

<code>>>> rdd.getNumPartitions()</code>	List the number of partitions
<code>>>> rdd.count()</code>	Count RDD instances
<code>>>> rdd.countByKey()</code>	Count RDD instances by key
<code>>>> rdd.countByValue()</code>	Count RDD instances by value
<code>>>> rdd.collectAsMap()</code>	Return (key,value) pairs as a dictionary
<code>(['a': 2, 'b': 2) >>> rdd3.sum() 4950 >>> sc.parallelize([]).isEmpty() True</code>	Sum of RDD elements
	Check whether RDD is empty

Summary

<code>>>> rdd3.max() 99 >>> rdd3.min() 0 >>> rdd3.mean() 49.5 >>> rdd3.stdev() 28.866070047722118 >>> rdd3.variance() 833.0 >>> rdd3.histogram(3) ([0, 33, 66, 99], [33, 33, 34]) >>> rdd3.stats()</code>	Maximum value of RDD elements Minimum value of RDD elements Mean value of RDD elements Standard deviation of RDD elements Compute variance of RDD elements Compute histogram by bins Summary statistics (count, mean, stdev, max & min)
--	---

Applying Functions

<code>>>> rdd.map(lambda x: x+(x[1],x[0]))collect() ... [('a',7,'a'), ('a',2,2,'a'), ('b',2,2,'b')] >>> rdd5.collect() ... [('a',7,'a','a',2,2,'a','b',2,2,'b')collect() ... [('a','z','y'),('a','z'),('b','p'),('b','r')]]</code>	Apply a function to each RDD element Apply a function to each RDD element and flatten the result Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys
--	--

Selecting Data

<code>>>> rdd.collect() ... [('a', 7), ('a', 2), ('b', 2)] ... ('a', 7, 'a', 2) ... ('a', 7, 'a', 2) ... rdd.first() ... ('a', 7) ... rdd.top(2) ... ('b', 2), ('a', 7)] ... Sampling ... rdd3.sample(False, 0.15, 81).collect() ... [3, 4, 27, 31, 40, 41, 42, 43, 60, 76, 79, 80, 86, 97]</code>	Return a list with all RDD elements Take first 2 RDD elements Take first RDD element Take top 2 RDD elements Return sampled subset of rdd3 Filter the RDD Return distinct RDD values Return (key,value) RDD's keys
---	---

Iterating

<code>>>> def g(x): print(x) ... rdd.foreach(g) ... ('a', 7) ... ('b', 2) ... ('a', 2)</code>	Apply a function to all RDD elements
--	--------------------------------------

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x, y : x+y)
...     .collect()
... [('a',9), ('b',1)]
```

`>>> rdd.reduce(lambda a, b: a + b)`

Grouping

```
>>> rdd3.groupByKey()
...     .mapValues(list)
...     .collect()
... [(('a',9), ('b',2))]
```

Aggregating

```
>>> seqOp = (lambda x, y: (x[0]+y,x[1]+1))
... combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
... rdd3.aggregate((0,0),seqOp,combOp)
... (4950,100)
... >>> rdd3.aggregateByKey((0,0),seqOp,combOp)
...     .collect()
... [(('a',9,2), ('b',2,1))]
... >>> rdd3.fold(0,add)
... 4950
... >>> rdd3.foldByKey(0,add)
...     .collect()
... [(('a',9), ('b',2))]
... >>> rdd3.keyBy(lambda x: x+x)
...     .collect()
```

Merge the rdd values for each key
Merge the rdd values

Return RDD of grouped values

Group rdd by key

Aggregate RDD elements of each partition and then the results
Aggregate values of each RDD key

Create tuples of RDD elements by applying a function

Mathematical Operations

<code>>>> rdd.subtract(rdd2)collect() ... [(('b',2),('a',7))] ... >>> rdd2.subtractByKey(rdd)collect() ... [(('d',1),)] ... >>> rdd.cartesian(rdd2).collect()</code>	Return each rdd value not contained in rdd2 Return each (key,value) pair of rdd2 with no matching key in rdd Return the Cartesian product of rdd and rdd2
---	---

Sort

```
>>> rdd2.sortBy(lambda x: x[1])
...     .collect()
... [(('d',1),('b',1),('a',2))]
... >>> rdd2.sortByKey()
...     .collect()
... [(('a',2),('b',1),('d',1))]
```

Sort RDD by given function

Sort (key, value) RDD by key

Repartitioning

```
>>> rdd.repartition(4)
... >>> rdd.coalesce(1)
```

New RDD with 4 partitions

Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
... >>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
...     'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

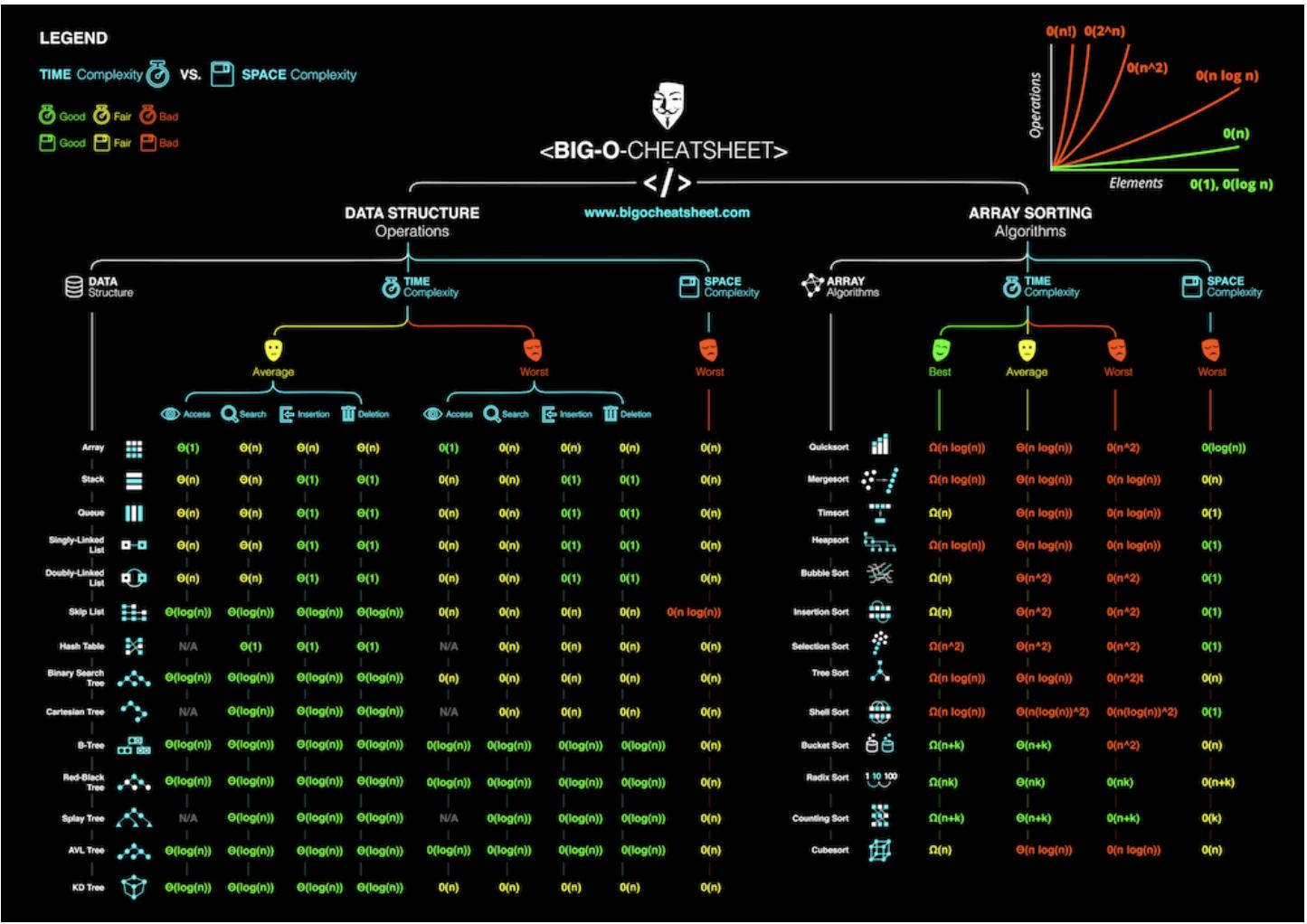
```
$ ./bin/spark-submit examples/src/main/python/pi.py
```

DataCamp

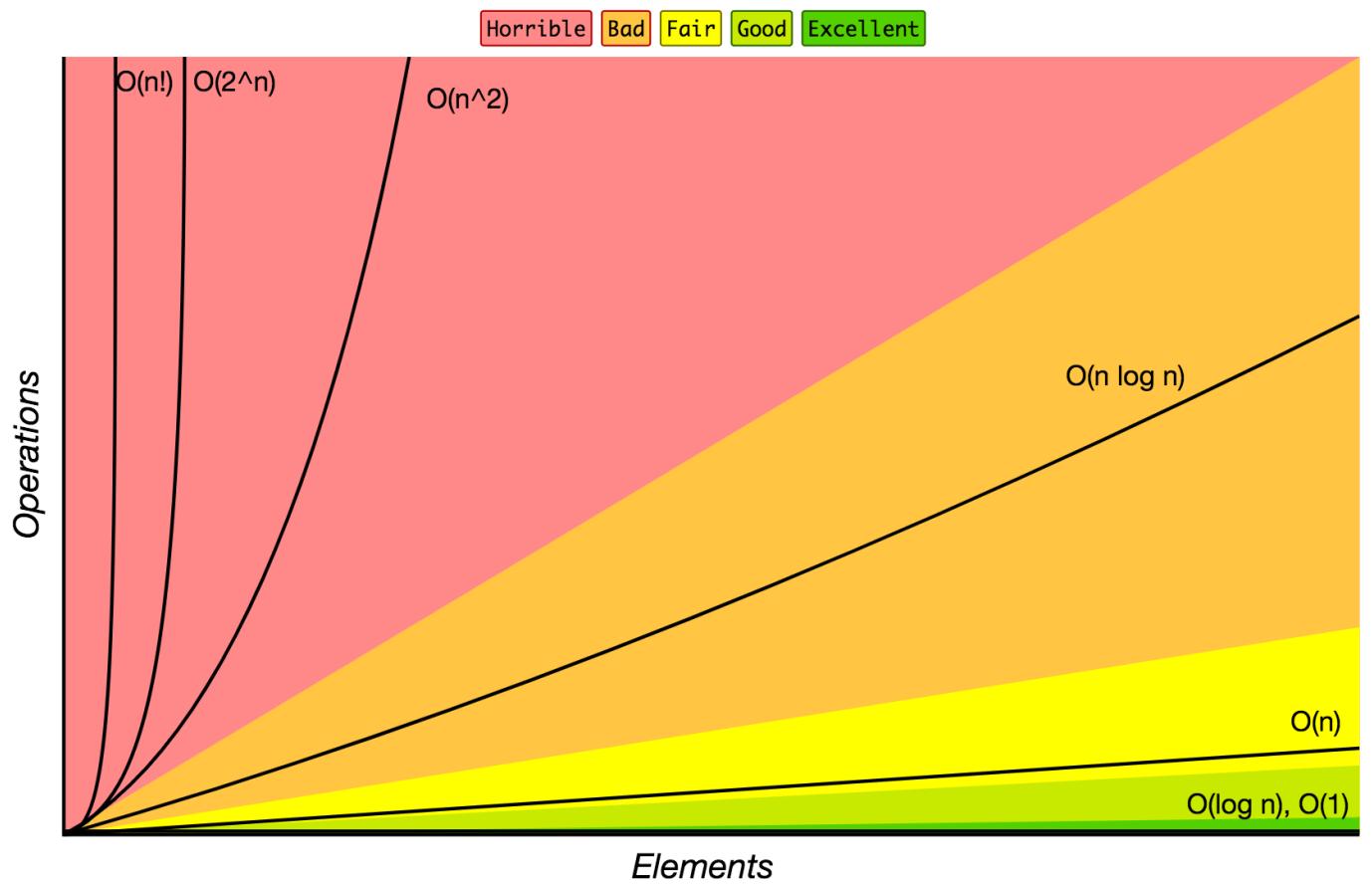
Learn Python for Data Science interactively

Big-O

Pyspark Cheat Sheet



Big-O Complexity Chart



Big-O Algorithm Complexity Chart

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity	
	Average				Worst					
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion		
Array	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	
Stack	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	
Queue	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	
Singly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	
Doubly-Linked List	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$	
Skip List	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n \log(n))$	
Hash Table	N/A	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	N/A	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	
Binary Search Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	
Cartesian Tree	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	N/A	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	
B-Tree	$\Theta(\log(n))$	$\Theta(n)$								
Red-Black Tree	$\Theta(\log(n))$	$\Theta(n)$								
Splay Tree	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	N/A	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(n)$	
AVL Tree	$\Theta(\log(n))$	$\Theta(n)$								
KD Tree	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(\log(n))$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	

BIG-O Algorithm Data Structure Operations

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	
Quicksort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
Mergesort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Timsort	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
Heapsort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
Tree Sort	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
Shell Sort	$\Omega(n \log(n))$	$\Theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
Bucket Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$
Radix Sort	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
Counting Sort	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$
Cubesort	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

Big-O Array Sorting Algorithms

About Stefan

Stefan is the founder of [Chatbot's Life](#), a Chatbot media and consulting firm. Chatbot's Life has grown to over 150k views per month and has become the premium place to learn about Bots & AI online. Chatbot's Life has also consulted many of the top Bot companies like Swelly, Instavest, OutBrain, NearGroup and a number of Enterprises.

I only write 2-3 articles per month.

yourname@example.com

Sign up

176

29



...



Join the Community



Subscribe



Apply To Be A Writer

Resources

Big-O Algorithm Cheat Sheet: <http://bigocheatsheet.com/>

Bokeh Cheat Sheet:

https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Python_Bokeh_Cheat_Sheet.pdf

Data Science Cheat Sheet:

<https://www.datacamp.com/community/tutorials/python-data-science-cheat-sheet-basics>

Data Wrangling Cheat Sheet: <https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf>

Data Wrangling: https://en.wikipedia.org/wiki/Data_wrangling

Ggplot Cheat Sheet: <https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

Keras Cheat Sheet:

<https://www.datacamp.com/community/blog/keras-cheat-sheet#gs.DRKeNMs>

Keras: <https://en.wikipedia.org/wiki/Keras>

Machine Learning Cheat Sheet: <https://ai.icymi.email/new-machinelearning-cheat-sheet-by-emily-barry-abdsc/>

Machine Learning Cheat Sheet: <https://docs.microsoft.com/en-in/azure/machine-learning/machine-learning-algorithm-cheat-sheet>

ML Cheat Sheet:: <http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html>

Matplotlib Cheat Sheet:

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

Matplotlib: <https://en.wikipedia.org/wiki/Matplotlib>

Neural Networks Cheat Sheet:

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

Neural Networks Graph Cheat Sheet:

<http://www.asimovinstitute.org/blog/>

Neural Networks: <https://www.quora.com/Where-can-find-a-cheat-sheet-for-neural-network>

Numpy Cheat Sheet:

<https://www.datacamp.com/community/blog/python-numpy-cheat-sheet#gs.AK5ZBgE>

NumPy: <https://en.wikipedia.org/wiki/NumPy>

Pandas Cheat Sheet:

<https://www.datacamp.com/community/blog/python-pandas-cheat-sheet#gs.oundfxM>

Pandas: [https://en.wikipedia.org/wiki/Pandas_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))

Pandas Cheat Sheet:

<https://www.datacamp.com/community/blog/pandas-cheat-sheet-python#gs.HPFoRIC>

Pyspark Cheat Sheet:

<https://www.datacamp.com/community/blog/pyspark-cheat-sheet-python#gs.L=J1zxQ>

Scikit Cheat Sheet:

<https://www.datacamp.com/community/blog/scikit-learn-cheat-sheet>

Scikit-learn: <https://en.wikipedia.org/wiki/Scikit-learn>

Scikit-learn Cheat Sheet: <http://peekaboo-vision.blogspot.com/2013/01/machine-learning-cheat-sheet-for-scikit.html>

Scipy Cheat Sheet:

<https://www.datacamp.com/community/blog/python-scipy-cheat-sheet#gs.JDSg3OI>

SciPy: <https://en.wikipedia.org/wiki/SciPy>

TesorFlow Cheat Sheet: <https://www.altoros.com/tensorflow-cheat-sheet.html>

Tensor Flow: <https://en.wikipedia.org/wiki/TensorFlow>

