

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

The Most Complete List of Best AI Cheat Sheets



Stefan Kojouharov [Follow](#)

Jul 9, 2017 · 7 min read

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](#).

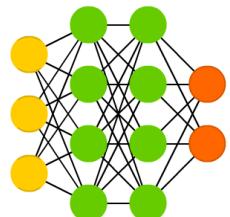
## Neural Networks

- 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

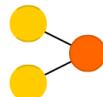
# A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

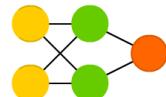
Deep Feed Forward (DFF)



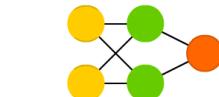
Perceptron (P)



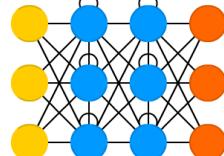
Feed Forward (FF)



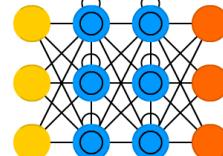
Radial Basis Network (RBF)



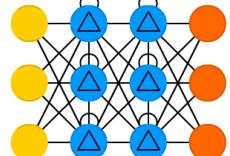
Recurrent Neural Network (RNN)



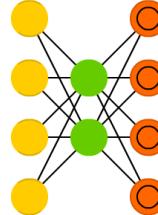
Long / Short Term Memory (LSTM)



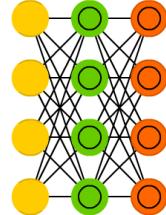
Gated Recurrent Unit (GRU)



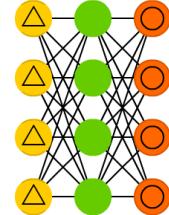
Auto Encoder (AE)



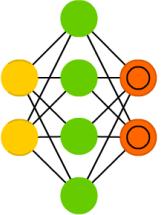
Variational AE (VAE)



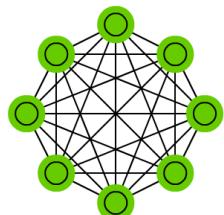
Denoising AE (DAE)



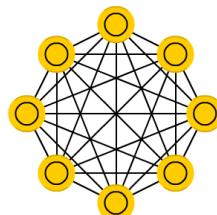
Sparse AE (SAE)



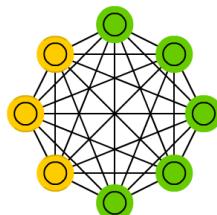
Markov Chain (MC)



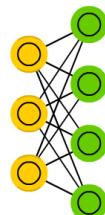
Hopfield Network (HN)



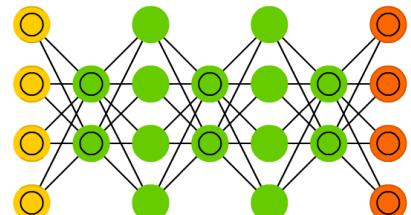
Boltzmann Machine (BM)



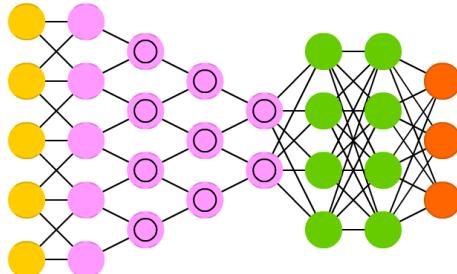
Restricted BM (RBM)



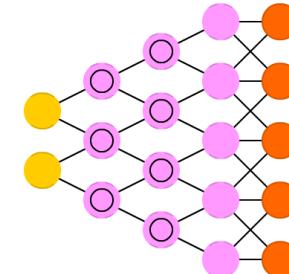
Deep Belief Network (DBN)



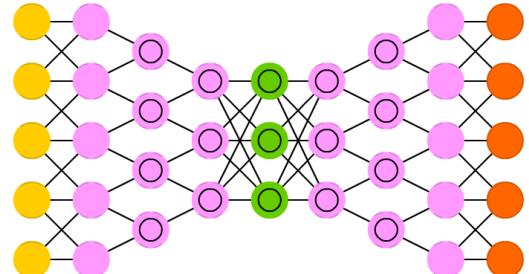
Deep Convolutional Network (DCN)



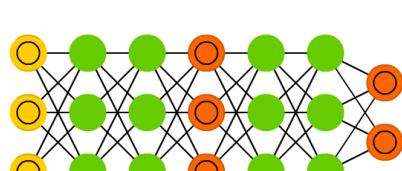
Deconvolutional Network (DN)



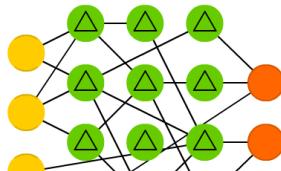
Deep Convolutional Inverse Graphics Network (DCIGN)



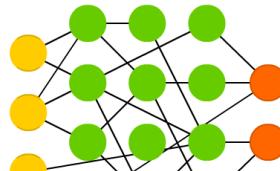
Generative Adversarial Network (GAN)



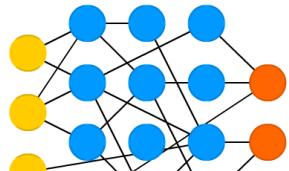
Liquid State Machine (LSM)

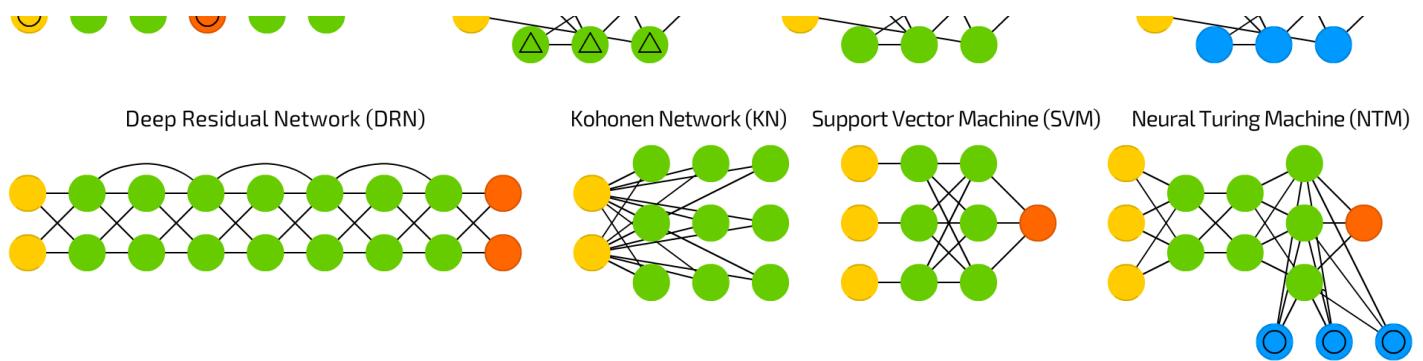


Extreme Learning Machine (ELM)



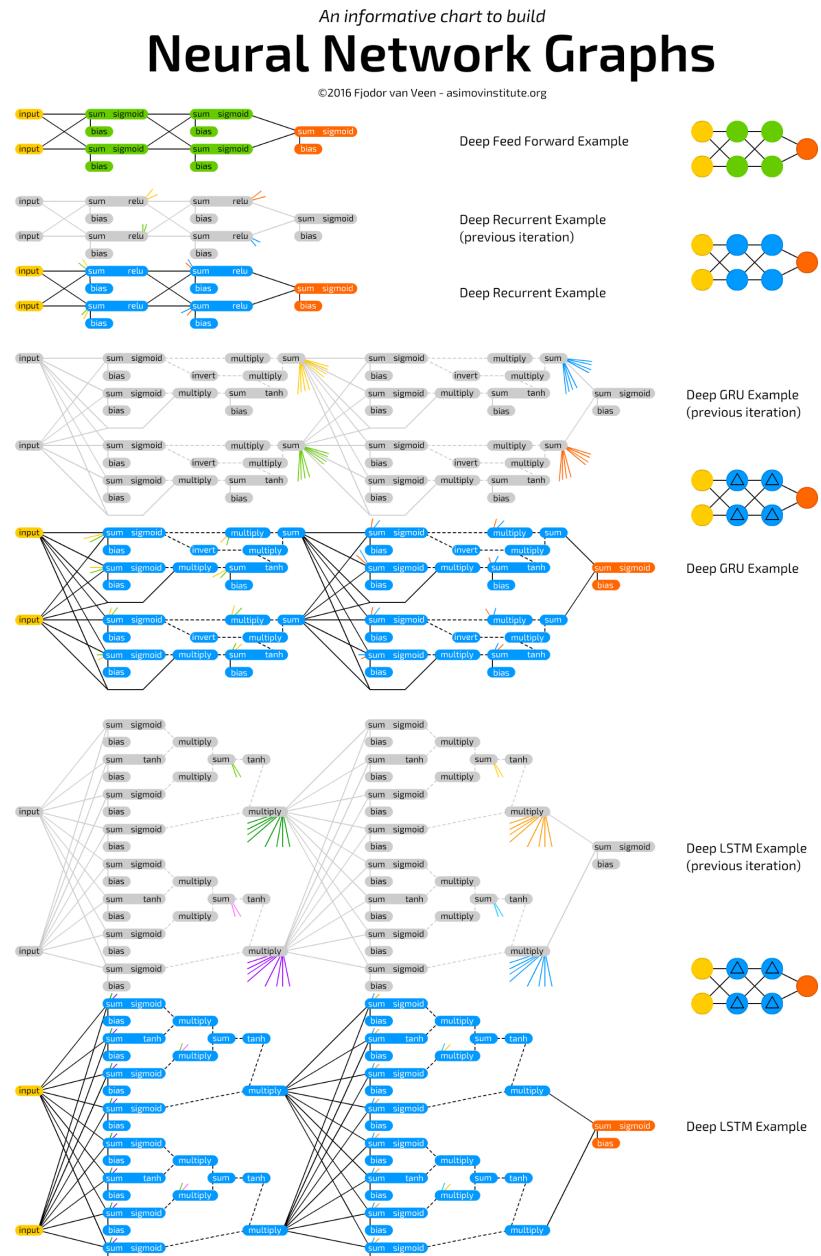
Echo State Network (ESN)





Neural Networks Cheat Sheet

## Neural Networks Graphs



| Linear Vector Spaces:  |  |
|--|--|
| Definition: A linear vector space, $X$ , is a set of elements (vectors) defined over a field, $F$ , that satisfies the following conditions:   |  |
| 1) If $x \in X$ and $y \in X$ then $x+y \in X$ .    2) $\alpha \cdot x = y \in X$ .  |  |
| 3) There is a unique vector $0 \in X$ , such that $x+0=x$ for all $x \in X$ .  |  |
| 4) For every vector $x \in X$ , there exists a vector in $X$ , to be called $(x)$ , such that $x+(x)=0$ .  |  |
| 5) For all scalars $\alpha \in F$ and $x \in X$ , $\alpha x$ is called $(\alpha)x$ .   |  |
| 6) Multiplication, for all scalars $\alpha \in F$ , and all vectors $x \in X$ .  |  |
| 7) For any $x \in X$ , $1 \cdot x = x$ .   |  |
| 8) For any two scalars $a \in F$ and $b \in F$ and any $x \in X$ , $a(bx) = (ab)x$ .   |  |
| 9) $(a+b)x = ax + bx$ .  |  |
| 10) $a(x+y) = ax + ay$ .   |  |
| Linear Independence:   |  |
| Consider a 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 $X$ . Then if every vector $x \in X$ can be expressed as a linear combination of the $\{u_i\}$ , we say that $\{u_1, u_2, \dots, u_n\}$ spans $X$ .  |  |
| Inner Product:   |  |
| $\langle x, y \rangle$ for any scalar function of $x$ and $y$ .  |  |
| 1. $\langle x, y \rangle = \langle y, x \rangle$ 2. $\langle x, y + z \rangle = \langle x, y \rangle + \langle x, z \rangle$ 3. $\langle x, y \rangle = 0$ , where equality holds if $y$ is the zero vector.   |  |
| Norm:  |  |
| A scalar function $\ x\ $ is called a norm if it satisfies:  |  |
| 1. $\ x\  \geq 0$  |  |
| 2. $\ x\  = 0$ if and only if $x = 0$ .  |  |
| 3. $\ ax\  =  a \ x\ $   |  |
| 4. $\ x + y\  \leq \ x\  + \ y\ $  |  |
| Angle:   |  |
| The angle $\theta$ between two vectors $x$ and $y$ is defined by $\cos \theta = \frac{\langle x, y \rangle}{\ x\  \ y\ }$ .  |  |
| Orthogonality:   |  |
| 2 vectors $x$ and $y$ are said to be orthogonal if $\langle x, y \rangle = 0$ .  |  |
| Gram-Schmidt Orthogonalization:  |  |
| Assume that we have $n$ independent vectors $v_1, v_2, \dots, v_n$ . From these vectors we will obtain $n$ orthogonal vectors $v_1, v_2, \dots, v_n$ .   |  |
| $v_1 = y_1, \quad v_k = y_k - \sum_{i=1}^{k-1} \langle y_k, v_i \rangle v_i,$<br>where $\langle v_i, v_j \rangle$ is the projection of $y_k$ on $v_i$  |  |
| Vector Expansions:   |  |
| $x = \sum_{i=1}^n x_i v_i = x_1 v_1 + x_2 v_2 + \dots + x_n v_n,$<br>for orthogonal vectors, $x_j = \frac{\langle v_j, x \rangle}{\langle v_j, v_j \rangle}$   |  |
| Reciprocal Basis Vectors:  |  |
| $\langle r_i, v_j \rangle = \begin{cases} 1 & i=j \\ 0 & i \neq j \end{cases}, \quad r_j = \langle r_j, x \rangle$<br>To compute the reciprocal basis vectors: set $B = [v_1 \ v_2 \ \dots \ v_n]$ , $R = [r_1 \ r_2 \ \dots \ r_n]$ , $R^T = B^{-1}$ . In matrix form: $x^R = B^{-1} x^V$ |  |
| Transformations:   |  |
| A transformation consists of three parts:<br>domain: $X = \{x\}$ , range: $Y = \{y\}$ , and a rule relating each $x \in X$ to an element $y \in Y$ .   |  |
| Linear Transformations:  |  |
| transformation $A$ is linear if:<br>1. $A(x_1 + x_2) = A(x_1) + A(x_2)$<br>2. for all $x \in X$ , $a \in R$ , $A(ax) = a(A(x))$  |  |
| Matrix Representations:  |  |
| Let $\{v_1, v_2, \dots, v_n\}$ be a basis for vector space $X$ , and let $\{u_1, u_2, \dots, u_n\}$ be a basis for vector space $Y$ . Let $A$ be a linear transformation with domain $X$ and range $Y$ : $A(x) = y$ .<br>The coefficients of the matrix representation are obtained from   |  |
| $A(v_j) = \sum_{i=1}^m a_{ij} u_i$<br><b>Change of Basis:</b> $B_1 = [t_1 \ t_2 \ \dots \ t_m], \quad B_2 = [w_1 \ w_2 \ \dots \ w_n]$<br>$A' = [B_2^{-1} A B_1]$  |  |
| Eigenvalues & Eigenvectors:  |  |
| $Ax = \lambda x, \quad  [A - \lambda I]  = 0$  |  |
| Diagonalization:   |  |
| $B = [z_1 \ z_2 \ \dots \ z_n]$ , where $\{z_1, z_2, \dots, z_n\}$ are the eigenvectors of a square matrix $A$ , $[B^{-1} A B] = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$  |  |

| Perceptron Architecture:   |   |
|--|---|
| $a = \text{hardlim}(Wp + b)$ , $W = [1 \ w^T], w^T = [w_1 \ w_2 \ \dots \ w_n]^T$  | $a_k = \text{hardlim}(p_k)$   |
| $a_i = \text{hardlim}(p_i)$ , $\text{hardlim}(x) = \text{sign}(x)$   | $\text{hardlim}(p_k) = (x_k - \bar{x}) / \sigma$                                      |
| Decision Boundary:   |   |
| $w^T p + b = 0$  | The decision boundary is always orthogonal to the weight vector.                      |
| Single-layer perceptrons can only classify linearly separable vectors.   |   |
| Perceptron Learning Rule   |   |
| $W^{\text{new}} = W^{\text{old}} + \eta p^T, \quad b^{\text{new}} = b^{\text{old}} + \eta$ ,<br>where $t = t - a$  |   |
| Hebb's Postulate:  |   |
| "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or more cells such that A's become stronger. As one of the cells firing is increased."  |   |
| Linear Associator:   |   |
| $W^{\text{new}} = \text{purelin}(Wp + b)$  |   |
| The Hebb Rule:   |   |
| $\text{Supervised Form: } W_{ij}^{\text{new}} = w_{ij}^{\text{old}} + t_{ij} p_q^T$  | $\text{For quadratic fn.: } W_{ij}^{\text{new}} = w_{ij}^{\text{old}} + t_{ij} p_q^T$ |
| $W = t_1 P_1^T + t_2 P_2^T + \dots + t_q P_q^T$  | $P = \begin{bmatrix} P_1^T \\ P_2^T \\ \vdots \\ P_q^T \end{bmatrix} = TP^T$          |
| Pseudoinverse Rule: $W = TP^T$   |   |
| When the number, $R$ , of rows of $P$ is greater than the number of columns, $Q$ , of $P$ and the columns of $P$ are independent, then the pseudoinverse can be computed by $P^+ = (P^T)^{-1} P^T$   |   |
| Variations of Hebbian Learning:  |   |
| Filtered Learning (Ch.10): $W^{\text{new}} = W^{\text{old}} + \alpha(t_q - a_q)p_q^T$  |   |
| Delta Rule (Ch.10): $W^{\text{new}} = W^{\text{old}} + \alpha(t_q - a_q)p_q^T$   |   |
| Unsupervised Hebb (Ch.13): $W^{\text{new}} = W^{\text{old}} + \alpha a_q p_q^T$  |   |
| Taylor's $F(x) = F(x^*) + \nabla F(x^*)^T(x - x^*) + \frac{1}{2}(x - x^*)^T \nabla^2 F(x^*)^T(x - x^*) + \dots$  |   |
| Grad $\nabla F(x) = \left[ \frac{\partial}{\partial x_1} F(x) \quad \frac{\partial}{\partial x_2} F(x) \quad \dots \quad \frac{\partial}{\partial x_n} F(x) \right]^T$   |   |
| Hessian: $\nabla^2 F(x) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(x) & \frac{\partial^2}{\partial x_1 \partial x_2} F(x) & \dots & \frac{\partial^2}{\partial x_1 \partial x_n} F(x) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(x) & \frac{\partial^2}{\partial x_2^2} F(x) & \dots & \frac{\partial^2}{\partial x_2 \partial x_n} F(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} F(x) & \frac{\partial^2}{\partial x_n \partial x_2} F(x) & \dots & \frac{\partial^2}{\partial x_n^2} F(x) \end{bmatrix}$ |   |
| Directional Derivatives:   |   |
| 1 <sup>st</sup> Dir. Der.: $\frac{p^T \nabla F(x)}{\ p\ }, \quad 2^{\text{nd}}$ Dir. Der.: $\frac{p^T \nabla^2 F(x) p}{\ p\ ^2}$   |   |
| Minima:  |   |
| <b>Strong Minimum:</b> if a scalar $\delta > 0$ exists, such that $F(x) < F(x+\Delta x)$ for all $\Delta x$ such that $\delta > \ \Delta x\  > 0$ .  |   |
| <b>Global Minimum:</b> if $F(x) < F(x+\Delta x)$ for all $\Delta x \neq 0$   |   |
| <b>Weak Minimum:</b> if it is not a strong minimum, and a scalar $\delta > 0$ exists, such that $F(x) \leq F(x+\Delta x)$ for all $\Delta x$ such that $\delta > \ \Delta x\  > 0$ .   |   |
| Necessary Conditions for Optimality:   |   |
| <b>1<sup>st</sup>-Order Condition:</b> $\nabla F(x) _{x=x^*} = 0$ (Stationary Points)  |   |
| <b>2<sup>nd</sup>-Order Condition:</b> $\nabla^2 F(x) _{x=x^*} \geq 0$ (Positive Semi-definite Hessian Matrix).  |   |
| Quadratic fn.: $F(x) = \frac{1}{2} x^T A x + d^T x + c$  |   |
| $\nabla F(x) = Ax + d, \quad \nabla^2 F(x) = A, \quad \lambda_{\min} \leq \frac{p^T A p}{\ p\ ^2} \leq \lambda_{\max}$   |   |

| General Minimization Algorithm:   |  |
|---|--|
| $x_{k+1} = x_k + \alpha_k p_k$ or $\Delta x_k = (x_{k+1} - x_k) = \alpha_k p_k$   |  |
| Steepest Descent Algorithm:   |  |
| $x_{k+1} = x_k - \alpha_k g_k$ where, $g_k = \nabla F(x) _{x=x_k}$  |  |
| Stable Learning Rate:   |  |
| $(\alpha_k) \in [0, 1]$ multiplication, for all scalars $\alpha \in F$ , and all vectors $x \in X$ .  | $\alpha < \frac{2}{\lambda_{\max}}$ ( $\lambda_1, \lambda_2, \dots, \lambda_n$ ) Eigenvalues of Hessian matrix $A$ |
| Learning Rate to Minimize Along the Line:   |  |
| $x_{k+1} = x_k + \alpha_k p_k$ $\rightarrow$ $\frac{g_k^T p_k}{p_k^T p_k}$ (For quadratic fn.)  |  |
| After Minimization Along the Line:  |  |
| $x_{k+1} = x_k + \alpha_k p_k \rightarrow g_k^T p_k = 0$  |  |
| ADALINE: $a = \text{purelin}(Wp + b)$   |  |
| Mean Square Error: (for ADALINE it is a quadratic fn.)  |  |
| $F(x) = E[e^2] = E[(t - a)^2] = E[(t - x^T z)^2]$   |  |
| $F(x) = c - 2x^T h + x^T R x$ ,   |  |
| $c = E[t^2], \quad h = E[tz]$ and $R = E[zz^T] \Rightarrow A = 2R, \quad d = -2h$   |  |
| Unique minimum, if it exists, is $x^* = R^{-1}h$ ,  |  |
| where $x = \begin{bmatrix} 1 \\ w \end{bmatrix}$ and $z = \begin{bmatrix} 1 \\ p \end{bmatrix}$   |  |
| LMS Algorithm: $W(k+1) = W(k) + 2\alpha e(k)p^T(k)$   |  |
| $b(k+1) = b(k) + 2\alpha e(k)$  |  |
| Convergence Point: $x^* = R^{-1}h$  |  |
| Stable Learning Rate: $0 < \alpha < 1/\lambda_{\max}$ where $\lambda_{\max}$ is the maximum eigenvalue of $R$   |  |
| Adaptive Filter ADALINE:  |  |
| $a(k) = \text{purelin}(Wp(k) + b) = \sum_{i=1}^R w_i y(i-1) + b$  |  |
| Backpropagation Algorithm:  |  |
| Performance Index:  |  |
| Mean Square Error: $F(x) = E[e^2] = E[(t - a)^2]$   |  |
| Approximate Performance Index: (single sample)  |  |
| $\hat{F}(x) = e^T(k)e(k) = (t(k) - a(k))^T(t(k) - a(k))$  |  |
| Sensitivity: $s^m = \frac{\partial \hat{F}}{\partial w^m} = \left[ \frac{\partial \hat{F}}{\partial w_1^m} \quad \frac{\partial \hat{F}}{\partial w_2^m} \quad \dots \quad \frac{\partial \hat{F}}{\partial w_n^m} \right]^T$ |  |
| Forward Propagation: $a^0 = p$ ,  |  |
| $a^{m+1} = f^{m+1}(w^{m+1}a^m + b^{m+1})$ for $m = 0, 1, \dots, M-1$  |  |
| a <sup>M</sup> :  |  |
| Backward Propagation: $s^M = -2\hat{F}(m^N)(t - a)$ ,   |  |
| $s^m = F^m(n^m)(W^{m+1}s^{m+1})$ for $m = M-1, \dots, 2, 1$ , where $F^m(n^m) = \text{diag}([f^m(n_1^m), f^m(n_2^m), \dots, f^m(n_s^m)])$   |  |
| $f^m(n_i^m) = \frac{\partial f^m(n_i^m)}{\partial n_i^m}$   |  |
| Weight Update (Approximate Stochastic Descent):   |  |
| $W^m(k+1) = W^m(k) - \alpha s^m(a^{m-1})$   |  |
| $b^m(k+1) = b^m(k) - \alpha s^m$  |  |
| hardlim: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$  |  |
| hardlims: $a = \begin{cases} -1 & n < 0 \\ 1 & n \geq 0 \end{cases}$  |  |
| purelin: $a = n$  |  |
| logsig: $a = \frac{e^{-x}}{1+e^{-x}}$   |  |
| tansig: $a = \frac{e^x - e^{-x}}{e^x + e^{-x}}$   |  |
| postin: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$   |  |
| compit: $a = \begin{cases} 1 & \text{one neuron with max } n \\ 0 & \text{all other neurons} \end{cases}$   |  |
| satlin: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 1 \\ -1 & 0 \leq n \leq 1 \end{cases}$   |  |
| LVQ Network Learning with the Kohonen Rule:   |  |
| $t^W(q) = t^W(q-1) + \alpha(p(q) - t^W(q-1))$   |  |
| $t^W(q) = t^W(q-1) + \alpha(p(q) - t^W(q-1))$   |  |
| Self-Organizing with the Kohonen Rule:  |  |
| $t^W(q) = t^W(q-1) + \alpha(p(q) - t^W(q-1))$   |  |
| if $a_k^2 = 1 \neq t_k = 0$   |  |
| if $a_k^2 = 1 \neq t_k = 0$   |  |
| delay: $a(t) = u(t-1)$  |  |
| integrator: $a(t) = \int_0^t u(\tau) d\tau + a(0)$  |  |

| *Heuristic Variations of Backpropagation:   |  |
|---|--|
| Batching: the gradients are calculated only after the entire training set has been presented. The gradients calculated for each training example are averaged together to produce a more accurate estimate of the gradient. If the training set is completed, i.e., covers all possible input/output pairs, then the gradient estimate will be exact. |  |
| Backpropagation with Momentum (MOBP):   |  |
| $\Delta w^m(k) = \gamma \Delta w^m(k-1) - (1-\gamma)\alpha s^m(a^{m-1})$  |  |
| $\Delta b^m(k) = \gamma \Delta b^m(k-1) - (1-\gamma)\alpha s^m$   |  |
| Variable Learning Rate Backpropagation (VLBP)   |  |
| 1. If the squared error (over the entire training set) increases by more than some set percentage $\epsilon$ (typically one to five percent) after a weight update, then the weight update is rejected, the learning rate is multiplied by some factor $\beta$ , and the momentum coefficient $\gamma$ is set to zero.                                |  |
| 2. If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$ . If $\gamma$ has been previously set to zero, it is reset to its original value.   |  |
| Associative: $a = \text{hardlim}(W^T p + b)$  |  |
| An association is a link between the inputs and outputs of a network so that when a stimulus $A$ is presented to the network, it will output a response $B$ .   |  |
| Associative Learning Rules:   |  |
| Unsupervised Hebb Rule:   |  |
| $W(q) = W(q-1) + \alpha a(q)p^T(q)$   |  |
| Hebb with Decay:  |  |
| $W(q) = (1-\gamma)W(q-1) + \alpha a(q)p^T(q)$   |  |
| Instar: $a = \text{hardlin}(Wp + b)$  |  |
| The instar is activated for $ w  \cdot p = \ w\  \cdot \ p\  \cos \theta \geq -b$   |  |
| where $\theta$ is the angle between $p$ and $w$ .   |  |
| Kohonen Rule:   |  |
| $w(q) = (1-\alpha)w(q-1) + \alpha(p(q) - w(q-1))$   |  |
| Outstar Rule: $a = \text{satlin}(Wp)$   |  |
| $w_j(q) = w_j(q-1) + \alpha(a_j(q) - w_j(q-1))p_j(q)$   |  |
| Competitive Layer: $a = \text{compet}(Wp) = \text{compet}(n)$   |  |
| Competitive Learning with the Kohonen Rule:   |  |
| $t^W(q) = t^W(q-1) + \alpha(p(q) - t^W(q-1))$   |  |
| $= (1-\alpha)t^W(q-1) + \alpha p(q), \quad t^W(q) \neq d$   |  |
| LVO Network: $(w_d^2 = 1) \Rightarrow$ subclass $d$ is a part of class $k$  |  |
| $n_d^2 = -\ w^1 - p\ ^2, \quad a^1 = \text{compet}(n^1), \quad a^2 = W^2 a^1$   |  |
| LVQ Network Learning with the Kohonen Rule:   |  |
| $t^W(q) = t^W(q-1) + \alpha(p(q) - t^W(q-1))$   |  |
| if $a_k^2 = 1 \neq t_k = 1$   |  |
| if $a_k^2 = 1 \neq t_k = 0$   |  |
| delay: $a(t) = u(t-1)$  |  |
| integrator: $a(t) = \int_0^t u(\tau) d\tau + a(0)$  |  |

## Neural Network Cheat Sheet



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

Code Snippets and Github Included

chatbotslife.com

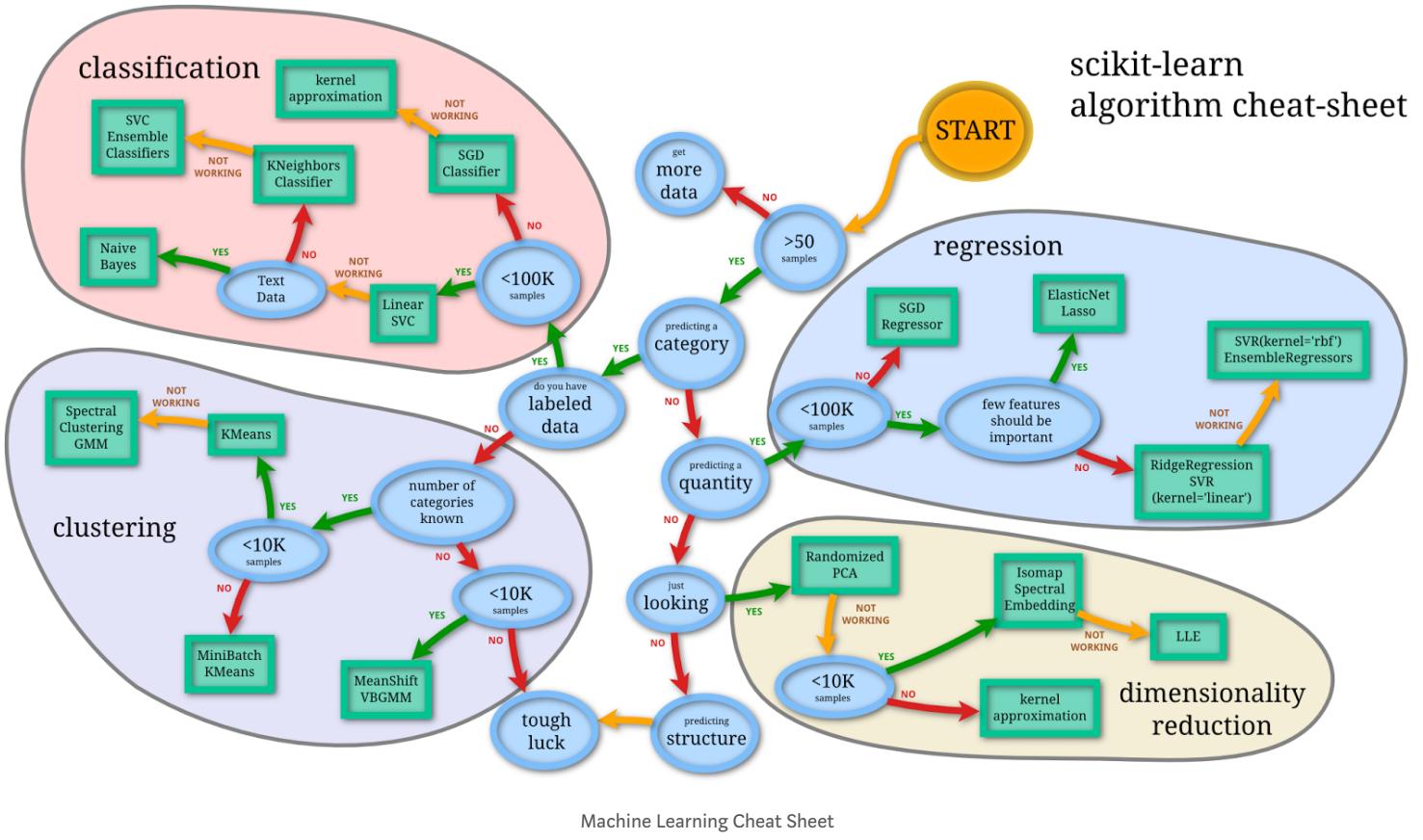
## Machine Learning Overview



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

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.

## Python For Data Science Cheat Sheet

### Scikit-Learn

Learn Python for data science interactively at [www.DataCamp.com](http://www.DataCamp.com)



### Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.



#### A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

#### Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F'])
>>> X[X < 0.7] = 0
```

#### Training And Test Data

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

#### Preprocessing The Data

##### Standardization

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

##### Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

##### Binarization

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

## Create Your Model

### Supervised Learning Estimators

**Linear Regression**  
`>>> from sklearn.linear_model import LinearRegression  
>>> lr = LinearRegression(normalize=True)`  
**Support Vector Machines (SVM)**  
`>>> from sklearn.svm import SVC  
>>> svc = SVC(kernel='linear')`  
**Naive Bayes**  
`>>> from sklearn.naive_bayes import GaussianNB  
>>> gnb = GaussianNB()`  
**KNN**  
`>>> from sklearn import neighbors  
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)`

### Unsupervised Learning Estimators

**Principal Component Analysis (PCA)**  
`>>> from sklearn.decomposition import PCA  
>>> pca = PCA(n_components=0.95)`  
**K Means**  
`>>> from sklearn.cluster import KMeans  
>>> k_means = KMeans(n_clusters=3, random_state=0)`

## Model Fitting

### Supervised learning

```
>>> lr.fit(X, y)
>>> knn.fit(X_train, y_train)
>>> svc.fit(X_train, y_train)
```

### Unsupervised Learning

```
>>> pca.fit(X_train)
```

Fit the model to the data

Fit the model to the data

Fit to data, then transform it

## Prediction

### Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict_proba(X_test)
```

### Unsupervised Estimators

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

Predict labels  
Predict labels  
Estimate probability of a label

Predict labels in clustering algos

## Scikit-Learn Cheat Sheet

# I only write 2-3 articles per month.

Email

Sign up



I agree to leave Medium and submit this information, which will be collected and used according to [Upscribe's](#) privacy policy.

## Evaluate Your Model's Performance

### Classification Metrics

**Accuracy Score**  
`>>> knn.score(X_test, y_test)`  
**Classification Report**  
`>>> from sklearn.metrics import classification_report  
>>> print(classification_report(y_test, y_pred))`

Estimator score method  
Metric scoring functions  
Precision, recall, f1-score and support

### Confusion Matrix

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

Precision, recall, f1-score and support

### Regression Metrics

**Mean Absolute Error**  
`>>> from sklearn.metrics import mean_absolute_error  
>>> y_true = [13, -0.5, 2]`  
**Mean Squared Error**  
`>>> from sklearn.metrics import mean_squared_error  
>>> mean_squared_error(y_true, y_pred)`

Precision, recall, f1-score and support

### R<sup>2</sup> Score

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

### Clustering Metrics

**Adjusted Rand Index**  
`>>> from sklearn.metrics import adjusted_rand_score  
>>> adjusted_rand_score(y_true, y_pred)`  
**Homogeneity**  
`>>> from sklearn.metrics import homogeneity_score  
>>> homogeneity_score(y_true, y_pred)`

Precision, recall, f1-score and support

### V-measure

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

### Cross-Validation

`>>> from sklearn.cross_validation import cross_val_score  
>>> print(cross_val_score(knn, X_train, y_train, cv=4))`

Precision, recall, f1-score and support

### Tune Your Model

#### Grid Search

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

Precision, recall, f1-score and support

#### Randomized Parameter Optimization

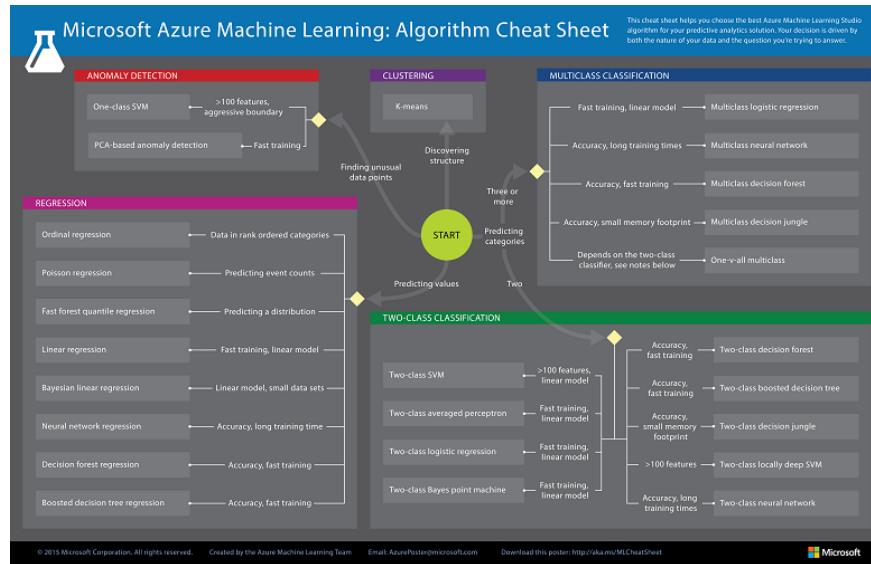
`>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
... "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
... param_distributions=params,
... n_iter=4,
... random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)`

DataCamp  
Learn Python for Data Science interactively



# MACHINE LEARNING : ALGORITHM CHEAT SHEET

*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.*



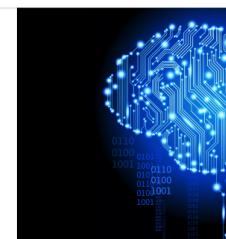
MACHINE LEARNING ALGORITHM CHEAT SHEET

>>> 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](http://chatbotslife.com)

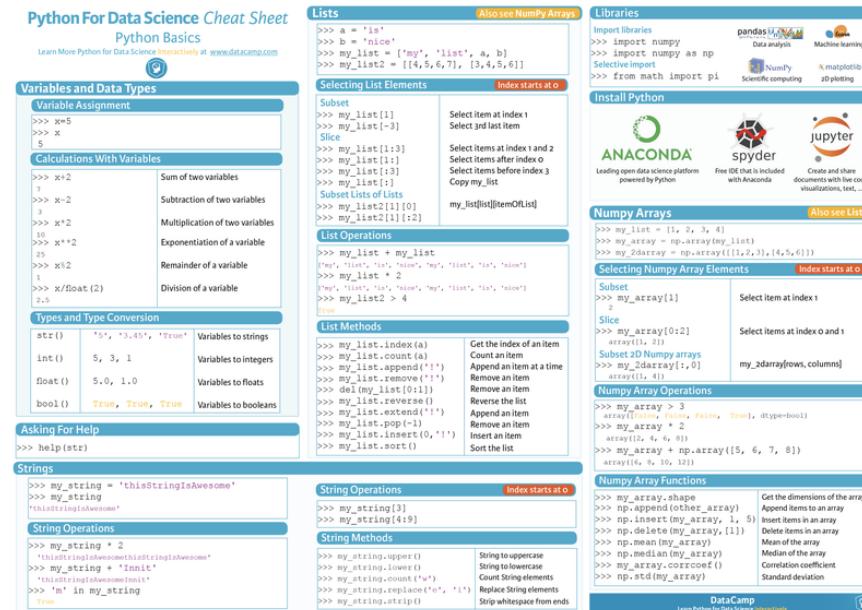


## Python for Data Science

**Python For Data Science Cheat Sheet**

Python Basics

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



This cheat sheet provides a quick reference for Python's data science ecosystem. It includes sections on Variables and Data Types, Lists, Libraries, Numpy Arrays, and String Operations. Each section contains code examples and descriptions of common operations.

### Variables and Data Types

| Variable Assignment                 |
|-------------------------------------|
| >>> a = 'is'<br>>>> b = 'nice'<br>5 |

### Calculations With Variables

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

### Types and Type Conversion

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

### Asking For Help

|               |
|---------------|
| >>> help(str) |
|---------------|

### Strings

|   |
|---|
| >>> my_string = 'thisStringIsAwesome'<br>>>> my_string<br>'thisStringIsAwesome' |
|---|

### String Operations

|   |
|---|
| >>> my_string * 2<br>'thisStringIsAwesome>thisStringIsAwesome'<br>>>> my_string + 'Init'<br>'thisStringIsAwesomeInit'<br>>>> 'm' in my_string<br>True |
|---|

### Lists

Also see NumPy Arrays

|  |
|--|
| >>> a = [1]<br>>>> b = ['nice']<br>>>> my_list = ['my', 'list', a, b]<br>>>> my_list2 = [[4,5,6,7], [3,4,5,6]] |
|--|

### Selecting List Elements

Index starts at 0

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

### List Operations

|   |
|---|
| >>> my_list + my_list<br>['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice'] |
| >>> my_list * 2<br>['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']       |
| >>> my_list2 * 4<br>[[4,5,6,7], [3,4,5,6], [4,5,6,7], [3,4,5,6]]                  |

### List Methods

|   |   |
|---|---|
| >>> my_list.index('a')<br>>>> my_list.count('a')<br>>>> my_list.append('c')<br>>>> my_list.remove('b')<br>>>> del(my_list[0:1])<br>>>> my_list.reverse()<br>>>> my_list.extend('d')<br>>>> my_list.insert(0, 'e')<br>>>> my_list.sort()<br>Sort the list. | Get the index of an item<br>Count an item<br>Append an item at once<br>Remove an item<br>Remove an item<br>Reverse the list<br>Append an item<br>Remove an item<br>Insert an item<br>Sort the list. |
|---|---|

### Operations

Index starts at 0

|  |
|--|
| >>> my_string[3]<br>>>> my_string[4:9] |
|--|

### String Methods

|   |  |
|---|--|
| >>> my_string.upper()<br>>>> my_string.lower()<br>>>> my_string.replace('a', 'z')<br>>>> my_string.replace('a', 'z')<br>>>> my_string.strip() | String touppercase<br>String tolowercase<br>Count string elements<br>Replace String elements<br>Strip whitespace from ends |
|---|--|

### Libraries

|            |                      |
|------------|----------------------|
| pandas     | Data analysis        |
| NumPy      | Scientific computing |
| matplotlib | 2D plotting          |

### Install Python

|          |  |
|----------|--|
| ANACONDA | Leading open data science platform powered by Python                 |
| spyder   | Free IDE that is included with Anaconda                              |
| jupyter  | Create and share documents with live code, visualizations, text, ... |

### Numpy Arrays

Also see Lists

|   |
|---|
| >>> my_list = [1, 2, 3, 4]<br>>>> my_array = np.array(my_list)<br>>>> my_2darray = np.array([[1,2,3], [4,5,6]]) |
|---|

### Selecting Numpy Array Elements

Index starts at 0

|                         |                               |
|-------------------------|-------------------------------|
| Subset                  | Select item at index 1        |
| >>> my_array[1]         |                               |
| Slice                   | Select items at index 0 and 1 |
| >>> my_array[0:2]       | array([1, 2])                 |
| Subsets of Numpy Arrays | Select items at index 1, 0    |
| >>> my_2darray[1,0]     | array([1, 4])                 |

### Numpy Array Operations

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

### Numpy Array Functions

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

DataCamp

Learn Python for Data Science Interactively

## Python Data Science Cheat Sheet

## Python For Data Science Cheat Sheet

### Bokeh

Learn Bokeh interactively at [www.DataCamp.com](http://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.

Bokeh's mid-level general purpose `bokeh.plotting` interface is centered around two main components: data and glyphs.



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', 'Fiat', '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

#### Rows

```
>>> from bokeh.layouts import row
>>> layout = row(p1,p2, p3)
>>> layout = column(p1,p2,p3)
>>> layout = row(column(p1,p2), p3)
```

#### Nesting Rows & Columns

```
>>> layout = row(column(p1,p2), p3)
```

#### Columns

```
>>> from bokeh.layouts import columns
>>> layout = columns(p1,p2,p3)
```

#### 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  | <code>p.legend.location = 'bottom_left'</code>   |
| Outside Plot Area | <code>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": [p1, r1]}, {"Two": [r2]}], location=(-0, -30))</code> |
|                   | <code>p.add_layout(legend, "right")</code>   |

##### Linked Plots

|                 |   |
|-----------------|---|
| Linked Axes     | <code>p2.x_range = p1.x_range p2.y_range = p1.y_range</code>  |
| Linked Brushing | <code>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', 'hp', source=cds_df) layout = row(p4,p5)</code> |

#### Output

##### Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file("my_bar_chart.html", mode='cdn')
Notebook Output
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

#### Embedding

##### Standalone HTML

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, "my_plot")
Components
>>> from bokeh.embed import components
>>> script, div = components(p)
```

### 5 Show or Save Your Plots

|  |  |
|--|--|
| <code>&gt;&gt;&gt; show(p1)</code>     | <code>&gt;&gt;&gt; save(p1)</code>     |
| <code>&gt;&gt;&gt; show(layout)</code> | <code>&gt;&gt;&gt; save(layout)</code> |

### Customized Glyphs

#### Also see Data

##### Selection and Non-Selection Glyphs

```
>>> p.circle('mpg', 'cyl', source=cds_df,
    selection_color='red',
    nonselection_alpha=0.1)
```

##### Hover Glyphs

```
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p.add_tools(hover)
```

##### Colormapping

```
>>> color_mapper = CategoricalColorMapper(
    factors=['Europe', 'Asia', 'US'],
    palette=['red', 'green', 'blue'])
>>> p.circle('mpg', 'cyl', source=cds_df,
    color=dict(field='origin',
    transform=color_mapper),
    legend='Origin')
```

#### Also see Data

### Linked Plots

#### 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', 'hp', source=cds_df)
>>> layout = row(p4,p5)
```

### Statistical Charts With Bokeh

#### Also see Data

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

## 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

### 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
```

### 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,`

**globals()**

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

**locals()**

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

TENSORFLOWREGRESSOR  
TensorFlowDNNClassifier  
TensorFlowDNNRegressor  
TensorFlowLinearClassifier  
TensorFlowLinearRegressor  
TensorFlowRNNClassifier  
TensorFlowRNNRegressor

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](http://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 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'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

#### 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

|   |                                      |
|---|--------------------------------------|
| <code>&gt;&gt;&gt; model.output_shape</code>  | Model output shape                   |
| <code>&gt;&gt;&gt; model.summary()</code>     | Model summary representation         |
| <code>&gt;&gt;&gt; model.get_config()</code>  | Model configuration                  |
| <code>&gt;&gt;&gt; model.get_weights()</code> | List all weight tensors in the model |

## Compile Model

|  |   |
|--|---|
| <b>MLP: Binary Classification</b>      | <code>&gt;&gt;&gt; model.compile(optimizer='adam',     loss='binary_crossentropy',     metrics=['accuracy'])</code>         |
| <b>MLP: Multi-Class Classification</b> | <code>&gt;&gt;&gt; model.compile(optimizer='rmsprop',     loss='categorical_crossentropy',     metrics=['accuracy'])</code> |
| <b>MLP: Regression</b>                 | <code>&gt;&gt;&gt; model.compile(optimizer='rmsprop',     loss='mse',     metrics=['mae'])</code>                           |
| <b>Recurrent Neural Network</b>        | <code>&gt;&gt;&gt; model3.compile(loss='binary_crossentropy',     optimizer='adam',     metrics=['accuracy'])</code>        |

## 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.h5')
>>> my_model = load_model('my_model.h5')
```

## Model Fine-tuning

### Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, 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])
```

## DataCamp

Learn Python for Data Science interactively



## Keras Cheat Sheet

# 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

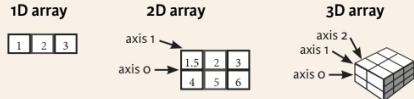
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

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))
```

### I/O

#### Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

#### Saving & Loading Text Files

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

### Data Types

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

### Inspecting Your Array

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

### Asking For Help

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

### Array Mathematics

#### Arithmetic Operations

```
>>> g = a - b
>>> array([[-0.5,  0.,  0.],
           [-3., -3., -3.]])
```

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

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

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

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

```
           [ 0.25,   1. ]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>>> np.sin(a)
```

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

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

```
>>> e.dot(f)
>>> array([[ 7.,  7.],
           [ 7.,  7.]])
```

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

```
>>> np.exp(b)
```

```
>>> np.sqrt(b)
```

```
>&
```

## 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

|  |  |    |
|--|--|----|
|  |  | 3  |
|  |  | -5 |
|  |  | 7  |
|  |  | 4  |

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

#### DataFrame

Columns

Index

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

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
>>> df.ix[:, 'Capital']
0    Brussels
1   New Delhi
2    Brasilia
>>> df.ix[1, 'Capital']
'New Delhi'
```

Select single row of subset of rows

#### Boolean Indexing

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

Series s where value is not >1  
s where value is <-1 or >2  
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()
```

### 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 | 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.
```

| n | v | a | b | c  |
|---|---|---|---|----|
| d | 1 | 4 | 7 | 10 |
| e | 2 | 5 | 8 | 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
```

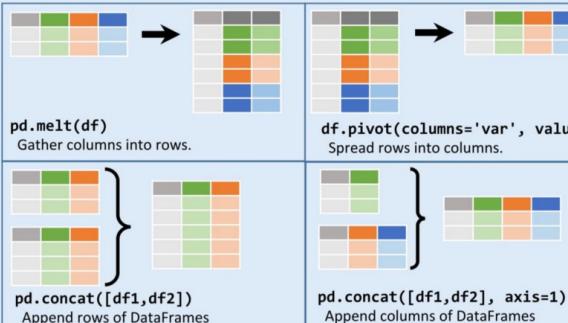
## Method Chaining

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).

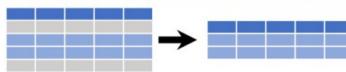
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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']
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 |
| '^*(?iSpecies\$).*' | 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.
```

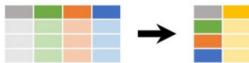
## Logic in Python (and pandas)

|    |                        |                             |                                     |
|----|------------------------|-----------------------------|-------------------------------------|
| <  | 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

## 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:

|                              |  |
|------------------------------|--|
| <b>sum()</b>                 | Sum values of each object.               |
| <b>count()</b>               | Count non-NA/null values of each object. |
| <b>median()</b>              | Median value of each object.             |
| <b>quantile([0.25,0.75])</b> | Quantiles of each object.                |
| <b>apply(function)</b>       | Apply function to each object.           |
| <b>min()</b>                 | Minimum value in each object.            |
| <b>max()</b>                 | Maximum value in each object.            |
| <b>mean()</b>                | Mean value of each object.               |
| <b>var()</b>                 | Variance of each object.                 |
| <b>std()</b>                 | Standard deviation of each object.       |

## Group Data



|                                |  |
|--------------------------------|--|
| <b>df.groupby(by='col')</b>    | Return a GroupBy object, grouped by values in column named "col".      |
| <b>df.groupby(level='ind')</b> | 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:

|                      |                                 |
|----------------------|---------------------------------|
| <b>size()</b>        | Size of each group.             |
| <b>agg(function)</b> | Aggregate group using function. |

## Windows

|                       |  |
|-----------------------|--|
| <b>df.expanding()</b> | Return an Expanding object allowing summary functions to be applied cumulatively.        |
| <b>df.rolling(n)</b>  | Return a Rolling object allowing summary functions to be applied to windows of length n. |

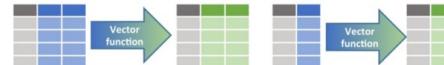
## Handling Missing Data

```
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:

|                                 |                   |
|---------------------------------|-------------------|
| <b>max(axis=1)</b>              | Element-wise max. |
| <b>clip(lower=-10,upper=10)</b> | Element-wise min. |

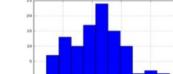
|              |                 |
|--------------|-----------------|
| <b>abs()</b> | Absolute value. |
|--------------|-----------------|

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.

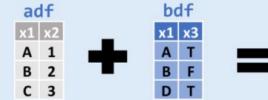
|                             |                                    |
|-----------------------------|------------------------------------|
| <b>shift(1)</b>             | Copy with values shifted by 1.     |
| <b>rank(method='dense')</b> | Rank with no gaps.                 |
| <b>rank(method='min')</b>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

## Plotting

|                                     |                                     |
|-------------------------------------|-------------------------------------|
| <b>df.plot.hist()</b>               | Histogram for each column           |
| <b>df.plot.scatter(x='w',y='h')</b> | Scatter chart using pairs of points |



## Combine Data Sets



### Standard Joins

|            |            |
|------------|------------|
| <b>adf</b> | <b>bdf</b> |
| x1   x2    | x1   x3    |
| A 1        | A T        |
| B 2        | B F        |
| C 3        | D T        |

```
pd.merge(adf, bdf,
        how='left', on='x1')
```

Join matching rows from bdf to adf.

|            |            |
|------------|------------|
| <b>adf</b> | <b>bdf</b> |
| x1   x2    | x1   x3    |
| A 1.0      | A T        |
| B 2.0      | F          |
| D NaN      | T          |

```
pd.merge(adf, bdf,
        how='right', on='x1')
```

Join matching rows from adf to bdf.

|            |            |
|------------|------------|
| <b>adf</b> | <b>bdf</b> |
| x1   x2    | x1   x3    |
| A 1        | T          |
| B 2        | F          |
| C 3        | NaN        |
| D NaN      | T          |

```
pd.merge(adf, bdf,
        how='inner', on='x1')
```

Join data. Retain only rows in both sets.

|            |            |
|------------|------------|
| <b>adf</b> | <b>bdf</b> |
| x1   x2    | x1   x3    |
| A 1        | T          |
| B 2        | F          |
| C 3        | NaN        |
| D NaN      | T          |

```
pd.merge(adf, bdf,
        how='outer', on='x1')
```

Join data. Retain all values, all rows.

|            |            |
|------------|------------|
| <b>adf</b> | <b>bdf</b> |
| x1   x2    | x1   x2    |
| A 1        | T          |
| B 2        | F          |
| C 3        | NaN        |

```
adf[adֆ.x1.isin(bdf.x1)]
```

All rows in adf that have a match in bdf.

|            |            |
|------------|------------|
| <b>adf</b> | <b>bdf</b> |
| x1   x2    | x1   x2    |
| C 3        |            |

```
adf[~adf.x1.isin(bdf.x1)]
```

All rows in adf that do not have a match in bdf.



### Set-like Operations

|            |            |
|------------|------------|
| <b>ydf</b> | <b>zdf</b> |
| x1   x2    | x1   x2    |
| A 1        | B 2        |
| B 2        | C 3        |
| C 3        |            |

```
pd.merge(ydf, zdf)
```

Rows that appear in both ydf and zdf (Intersection).

|            |            |
|------------|------------|
| <b>ydf</b> | <b>zdf</b> |
| x1   x2    | x1   x2    |
| A 1        |            |
| B 2        |            |
| C 3        |            |
| D 4        |            |

```
pd.merge(ydf, zdf, how='outer')
```

Rows that appear in either or both ydf and zdf (Union).

|            |            |
|------------|------------|
| <b>ydf</b> | <b>zdf</b> |
| x1   x2    | x1   x2    |
| A 1        |            |

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
```

```
.query('_merge == "left_only")
```

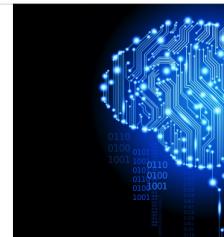
```
.drop(['_merge'], axis=1)
```

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 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      0.4   setosa
..          ...
..          ...
..          ...
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).

| iris |              |             |              |             |         |
|------|--------------|-------------|--------------|-------------|---------|
|      | 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          | 0.4         | setosa  |
| 6    | 5.4          | 3.9         | 1.7          | 0.4         | setosa  |
| 7    | 4.6          | 3.4         | 1.4          | 0.3         | 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:



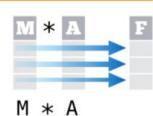
Each variable is saved in its own column

&

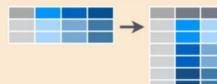


Each observation is saved in its own row

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).

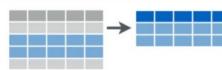
`dplyr::arrange(mtcars, desc(mpg))`

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.

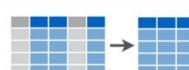
`dplyr::top_n(storms, 2, date)`

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

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

|    |                          |                        |                   |
|----|--------------------------|------------------------|-------------------|
| <  | Less than                | !=                     | Not equal to      |
| >  | Greater than             | %in%                   | Group membership  |
| == | Equal to                 | is.na                  | Is NA             |
| <= | Less than or equal to    | !is.na                 | Is not NA         |
| >= | Greater than or equal to | &,  , !, xor, any, all | 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("x"))`

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

## 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:

`dplyr::first`

First value of a vector.

`dplyr::last`

Last value of a vector.

`dplyr::nth`

Nth value of a vector.

`dplyr::n`

# of values in a vector.

`dplyr::n_distinct`

# of distinct values in a vector.

`IQR`

IQR of a vector.

`min`

Minimum value in a vector.

`max`

Maximum value in a vector.

`mean`

Mean value of a vector.

`median`

Median value of a vector.

`var`

Variance of a vector.

`sd`

Standard deviation of a vector.

## Group Data

`dplyr::group_by(iris, Species)`

Group data into rows with the same value of Species.

`dplyr::ungroup(iris)`

Remove grouping information from data frame.

`iris %>% group_by(Species) %>% summarise(...)`

Compute separate summary row for each group.



## 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:

`dplyr::lead`

Copy with values shifted by 1.

`dplyr::lag`

Copy with values lagged by 1.

`dplyr::dense_rank`

Ranks with no gaps.

`dplyr::min_rank`

Ranks. Ties get min rank.

`dplyr::percent_rank`

Ranks rescaled to [0, 1].

`dplyr::row_number`

Ranks. Ties got to first value.

`dplyr::ntile`

Bin vector into n buckets.

`dplyr::between`

Are values between a and b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::cumall`

Cumulative all

`dplyr::cumany`

Cumulative any

`dplyr::cummean`

Cumulative mean

`cumsum`

Cumulative sum

`cummax`

Cumulative max

`cummin`

Cumulative min

`cumprod`

Cumulative prod

`pmax`

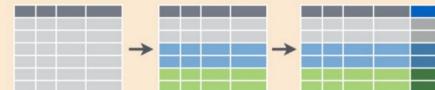
Element-wise max

`pmin`

Element-wise min

`iris %>% group_by(Species) %>% mutate(...)`

Compute new variables by group.



## Combine Data Sets



### Mutating Joins

`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.

### Filtering Joins

`dplyr::semi_join(a, b, by = "x1")`

All rows in a that have a match in b.

`dplyr::anti_join(a, b, by = "x1")`

All rows in a that do not have a match in b.



### Set Operations

`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

`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.

| x1 | x2 |
|----|----|
| A  | 1  |
| B  | 2  |
| C  | 3  |

| x1 | x2 | x3 |
|----|----|----|
| A  | 1  | T  |
| B  | 2  | F  |
| C  | 3  | NA |
| D  | 4  | T  |

## Data Wrangling with dplyr and tidyR Cheat Sheet

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", "tidyR")) • dplyr 0.4.0 • tidyR 0.2.0 • Updated: 1/15

# 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]        | Create a dense meshgrid            |
| >>> np.ogrid[0:2,0:2]        | Create an open meshgrid            |
| >>> np.r_[:3,[0]*5,-1:1:10j] | Stack arrays vertically (row-wise) |
| >>> np.c_[b,c]               | Create stacked column-wise arrays  |

#### Shape Manipulation

|                      |   |
|----------------------|---|
| >>> np.transpose(b)  | Permute array dimensions                      |
| >>> b.flatten()      | Flatten the array                             |
| >>> np.hstack((b,c)) | Stack arrays horizontally (column-wise)       |
| >>> np.vstack((a,b)) | Stack arrays vertically (row-wise)            |
| >>> np.hsplit(c,2)   | Split the array horizontally at the 2nd index |
| >>> np.vsplit(d,2)   | 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[3:1] += np.pi
Unwrap
>>> np.unwrap(g)
>>> np.logspace(0,10,3)
Create an array of evenly spaced values (logscale)
>>> 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,1.0)
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`.

[Also see NumPy](#)

#### Creating Matrices

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

#### Basic Matrix Routines

##### 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)
```

#### Creating Sparse Matrices

##### Inverse

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

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

`>>> C[0,0] = 0`

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

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

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

`>>> E.todense()`

`>>> sparse.isspmatrix_csc(A)`

##### Compressed Sparse Row 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)
```

#### Sparse Matrix Functions

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

`>>> sparse matrix exponential`

#### Asking For Help

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

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

#### Matrix Functions

##### Addition

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

##### Subtraction

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

##### Division

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

##### Multiplication

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

##### Inverse

```
>>> 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)`

`>>> linalg.expm2(A)`

`>>> linalg.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.signm(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`

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

##### LU Decomposition

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

#### Sparse Matrix Decompositions

##### Eigenvalues and Eigenvectors

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

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

## Scipy Cheat Sheet

### DataCamp

Learn Python for Data Science [Interactively](#)



## 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](#)

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

#### 2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> X, Y = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

### 2) Create Plot

```
>>> import matplotlib.pyplot as plt
```

#### Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.rcParams['figure.figsize'])
```

#### 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)
>>> ax.scatter(x, y)
>>> axes[0,0].bar([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 0

#### 2D Data or Images

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

Colormapped or RGB arrays

#### 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)
```

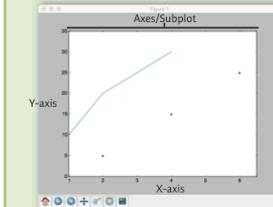
Plot a histogram  
Make a box and whisker plot  
Make a violin plot

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].contour(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data2)
>>> axes2[2] = ax.clabel(CS)
```

Pseudocolor plot of 2D array  
Pseudocolor plot of 2D array  
Plot contours  
Plot filled contours  
Label a contour plot

## Plot Anatomy & Workflow

### Plot Anatomy



Figure

### 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]
>>> y = [10,20,25,30]                                Step 1
>>> fig = plt.figure()                                Step 2
>>> ax = fig.add_subplot(111)                           Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)     Step 4
>>> ax.scatter([15, 16], [5, 15],                   Step 5
                color='darkgreen',
                marker='^')                                 Step 6
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()
```

### 4) Customize Plot

#### Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha=0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                  cmap='seismic')
```

#### Markers

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

#### LineStyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'-.',x**2,y**2,'.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

#### Text & Annotations

```
>>> ax.text(1, -2.1, 'Example Graph', style='italic')
>>> ax.annotate("sin", xy=(8, 0), xycoords='data',
               textcoords='data',
               arrowprops=dict(arrowstyle=">",
                               connectionstyle="arc3"),)
```

#### 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

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].contour(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data2)
>>> axes2[2] = ax.clabel(CS)
```

Pseudocolor plot of 2D array  
Pseudocolor plot of 2D array  
Plot contours  
Plot filled contours  
Label a contour plot

#### Mathtext

```
>>> plt.title(r'$\Sigma_{i=1}^n i = 15$', fontsize=20)
```

#### Limits, Legends & Layouts

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

Add padding to a plot  
Set the aspect ratio of the plot to 1  
Set limits for x and y axis  
Set limits for x-axis

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

Set a title and x and y axis labels  
No overlapping plot elements

```
>>> ax.xaxis.set(ticks=range(1,5),
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',
                  direction='inout',
                  length=10)
```

Manually set x-ticks  
Make y-ticks longer and go in and out  
Adjust the spacing between subplots

#### Subplot Spacing

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

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

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

Clear an axis  
Clear the entire figure  
Close a window

**DataCamp**

Learn Python for Data Science Interactively

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

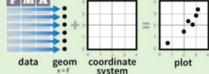
## Data Visualization

# Data Visualization with ggplot2 Cheat Sheet

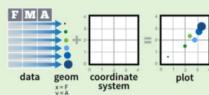


## Basics

**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()**

**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().

```
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(color = cyl)) +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()

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.

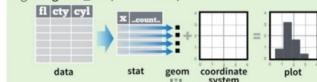
RStudio® is a trademark of RStudio, Inc. • [CC BY RStudio](#) • [info@rstudio.com](#) • 844-448-1212 • [rstudio.com](#)

| <b>Geoms</b> - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>One Variable</b> <ul style="list-style-type: none"> <li><b>Continuous</b> <ul style="list-style-type: none"> <li><code>a + geom_area(stat = "bin")</code><br/></li> <li><code>a + geom_density(kernel = "gaussian")</code><br/></li> <li><code>a + geom_dotplot()</code><br/></li> <li><code>a + geom_freqpoly()</code><br/></li> <li><code>a + geom_histogram(binwidth = 5)</code><br/></li> </ul> </li> <li><b>Discrete</b> <ul style="list-style-type: none"> <li><code>b + geom_bar()</code><br/></li> </ul> </li> </ul> | <b>Two Variables</b> <ul style="list-style-type: none"> <li><b>Continuous X, Continuous Y</b> <ul style="list-style-type: none"> <li><code>f + geom_blank()</code><br/></li> <li><code>f + geom_jitter()</code><br/></li> <li><code>f + geom_point()</code><br/></li> <li><code>f + geom_quantile()</code><br/></li> <li><code>f + geom_rug(sides = "bl")</code><br/></li> <li><code>f + geom_smooth(model = lm)</code><br/></li> <li><code>C f + geom_text(aes(label = cyl))</code><br/></li> </ul> </li> <li><b>Discrete X, Continuous Y</b> <ul style="list-style-type: none"> <li><code>g + geom_bar(stat = "identity")</code><br/></li> <li><code>g + geom_boxplot()</code><br/></li> <li><code>g + geom_dotplot(binaxis = "y", stackdir = "center")</code><br/></li> <li><code>g + geom_violin(scale = "area")</code><br/></li> </ul> </li> <li><b>Discrete X, Discrete Y</b> <ul style="list-style-type: none"> <li><code>h + geom_jitter()</code><br/></li> </ul> </li> </ul> | <b>Continuous Bivariate Distribution</b> <ul style="list-style-type: none"> <li><code>i &lt;- ggplot(movies, aes(year, rating))</code><br/></li> <li><code>i + geom_hex()</code><br/></li> </ul>                                                                                                                                                                                            |
| <b>Graphical Primitives</b> <ul style="list-style-type: none"> <li><b>Continuous Function</b> <ul style="list-style-type: none"> <li><code>j &lt;- ggplot(economics, aes(date, unemploy))</code><br/></li> <li><code>j + geom_area()</code><br/></li> <li><code>j + geom_line()</code><br/></li> <li><code>j + geom_step(direction = "hv")</code><br/></li> </ul> </li> </ul>                                                                                                                                                   | <b>Visualizing error</b> <ul style="list-style-type: none"> <li><code>df &lt;- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)</code><br/><code>k &lt;- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))</code><br/></li> <li><code>k + geom_crossbar(fatten = 2)</code><br/></li> <li><code>k + geom_errorbar()</code><br/></li> <li><code>k + geom_linerange()</code><br/></li> <li><code>k + geom_pointrange()</code><br/></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | <b>Maps</b> <ul style="list-style-type: none"> <li><code>data &lt;- data.frame(murder = USAArrests\$Murder, state = tolower(rownames(USAArrests)))</code><br/><code>map &lt;- map_data("state")</code><br/><code>l &lt;- ggplot(data, aes(fill = murder))</code><br/><code>l + geom_map(aes(map_id = state), map = map) + expand_limits(x = map\$long, y = map\$lat)</code><br/></li> </ul> |
| <b>Three Variables</b> <ul style="list-style-type: none"> <li><code>sealsSz &lt;- with(seals, sqrt(delta_long^2 + delta_lat^2))</code><br/><code>m &lt;- ggplot(seals, aes(long, lat))</code><br/></li> <li><code>m + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE)</code><br/></li> <li><code>m + geom_contour(aes(z = z))</code><br/></li> <li><code>m + geom_tile(aes(fill = z))</code><br/></li> </ul>                                                                                                | <p>Learn more at <a href="#">docs.ggplot2.org</a> • ggplot2 0.9.3.1 • Updated: 3/15</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                             |

## Data Visualization Cheat Sheet

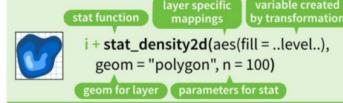
## Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. We 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.

stat 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..., .ncount..., density..., density...
a + stat_bindot(binwidth = 1, binaxis = "x")
x, y, | .count..., .ncount...
a + stat_bindot(binwidth = 1, kernel = "gaussian")
x, y, | .count..., .density..., .scaled...
  
```

```

f + stat_bin2d(bins = 30, drop = TRUE)           2D distributions
x, y | .fill | .count..., .density...
f + stat_hexbin(bins = 30)
x, y, fill | .count..., .density...
f + stat_hexbin2d(contour = TRUE, n = 100)
x, y, color, size | .level...
  
```

```

m + stat_contour(aes(z = z))                   3 Variables
x, y, z, order | .level...
mt + stat_spoke(aes(radius = z, angle = z))
angle, radius, x, end, y, end | .x..., .end..., .y..., .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(r = 40)                          Functions
x, y | .x..., .y...
f + stat_quantile(quartiles = c(0.25, 0.5, 0.75), formula = y ~ log(x),
method = "rc")
x, y | .quantile..., .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(aes(x = 3:3))          General Purpose
fun = dnorm, n = 101, args = list(sd=0.5)
x | .y...
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt,
dparams = list(df=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>` - map discrete values to manually chosen visual values

`scale_x_date`(labels = date\_format("%m/%d%Y"), breaks = date\_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.  
`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



`scale_fill_dodge`(start = 0.2, end = 0.8, na.value = "red")

`scale_shape` Manual shape values

`scale_size` Manual size values

`scale_size_area`(max = 6)

Value mapped to area of circle (not radius)

## Coordinate Systems

`r <- b + geom_bar()`

xlim, ylim

The default cartesian coordinate system

`r + coord_fixed(ratio = 1/2)`

ratio, xlim, ylim

Cartesian coordinates with fixed aspect ratio between x and y units

`r + coord_flip()`

xlim, ylim

Flipped Cartesian coordinates

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

theta, start, direction

Polar coordinates

`r + coord_trans(xtrans = "sqrt")`

xtrans, ytrans, xlim, ylim

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), aequalarea, lagrange, etc.)

## Position Adjustments

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

`s <- ggplot(mpg, aes(fl, 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

`r + theme_bw()`

White background with grid lines

`r + theme_classic()`

White background no gridlines

`r + theme_grey()`

Grey background (default theme)

`r + theme_minimal()`

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()`

`t + facet_grid(~ fl)`

facet into columns based on fl

`t + facet_grid(~ year)`

facet into rows based on year

`t + facet_grid(year ~ fl)`

facet into both rows and columns

`t + facet_wrap(~ fl)`

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

`t + facet_grid(~ fl, labeler = label_both)`

`fl: c fl: d fl: e fl: p fl: r`

`t + facet_grid(~ fl, labeler = label_bquote(alpha ^ .(x)))`

`alpha^c alpha^d alpha^e alpha^p alpha^r`

`t + facet_grid(~ fl, labeler = label_parsed)`

`c d e p r`

## Labels

`t + ggtitle("New Plot Title")`

Add a main title above the plot

`t + xlab("New X label")`

Change the label on the X axis

`t + ylab("New Y label")`

Change the label on the Y axis

`t + labs(title = "New title", x = "New x", y = "New y")`

Use scale functions to update legend labels

## Legends

`t + theme(legend.position = "bottom")`

Place legend at "bottom", "top", "left", or "right"

`t + guides(color = "none")`

Set legend type for each aesthetic: colorbar, legend, or none (no legend)

`t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))`

Set legend title and labels with a scale function.

## ggplot cheat sheet

# PySpark

## Python For Data Science Cheat Sheet

### PySpark Basics

Learn Python for data science interactively at [www.DataCamp.com](http://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>&gt;&gt;&gt; sc.version</code>              | Retrieve SparkContext version                 |
| <code>&gt;&gt;&gt; sc.pythonVer</code>            | Retrieve Python version                       |
| <code>&gt;&gt;&gt; sc.master</code>               | Master URL to connect to                      |
| <code>&gt;&gt;&gt; sc(slackHome)</code>           | Path where Spark is installed on worker nodes |
| <code>&gt;&gt;&gt; sc(slackUser())</code>         | SparkContext                                  |
| <code>&gt;&gt;&gt; sc appName</code>              | Return application name                       |
| <code>&gt;&gt;&gt; sc.applicationId</code>        | Return application ID                         |
| <code>&gt;&gt;&gt; sc.defaultParallelism</code>   | Return default level of parallelism           |
| <code>&gt;&gt;&gt; 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
```

Set 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', 1), ('d', 1), ('b', 1)])
>>> rdd3 = sc.parallelize(range(10))
>>> 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>&gt;&gt;&gt; rdd.getNumPartitions()</code>                                                | List the number of partitions            |
| <code>&gt;&gt;&gt; rdd.count()</code>                                                           | Count RDD instances                      |
| <code>&gt;&gt;&gt; rdd.countByKey()</code>                                                      | Count RDD instances by key               |
| <code>&gt;&gt;&gt; rdd.defaultDict(&lt;type 'int'&gt;, ('a':2,'b':1))</code>                    | Count RDD instances by value             |
| <code>&gt;&gt;&gt; rdd.defaultDict(&lt;type 'int'&gt;, ('b',2):1, ('a',2):1, ('a',1):1))</code> | Return (key,value) pairs as a dictionary |
| <code>&gt;&gt;&gt; rdd.collectAsMap()</code>                                                    | Sum of RDD elements                      |
| <code>&gt;&gt;&gt; rdd2.sum()</code>                                                            | Check whether RDD is empty               |
| <code>&gt;&gt;&gt; 4950</code>                                                                  |                                          |
| <code>&gt;&gt;&gt; sc.parallelize([]).isEmpty()</code>                                          |                                          |
| <code>True</code>                                                                               |                                          |

### Summary

|                                                           |                                                    |
|-----------------------------------------------------------|----------------------------------------------------|
| <code>&gt;&gt;&gt; rdd3.max()</code>                      | Maximum value of RDD elements                      |
| <code>&gt;&gt;&gt; 99</code>                              | Minimum value of RDD elements                      |
| <code>&gt;&gt;&gt; rdd3.mean()</code>                     | Mean value of RDD elements                         |
| <code>&gt;&gt;&gt; 49.5</code>                            | Standard deviation of RDD elements                 |
| <code>&gt;&gt;&gt; rdd3.stdev()</code>                    | Compute variance of RDD elements                   |
| <code>&gt;&gt;&gt; 28.85607004772218</code>               | Compute histogram by bins                          |
| <code>&gt;&gt;&gt; rdd3.variance()</code>                 | Summary statistics (count, mean, stdev, max & min) |
| <code>&gt;&gt;&gt; 833.25</code>                          |                                                    |
| <code>&gt;&gt;&gt; rdd3.histogram(3)</code>               |                                                    |
| <code>&gt;&gt;&gt; ([0, 33, 66, 99], [33, 33, 34])</code> |                                                    |
| <code>&gt;&gt;&gt; rdd3.stats()</code>                    |                                                    |

## Applying Functions

|                                                                       |                                      |
|-----------------------------------------------------------------------|--------------------------------------|
| <code>&gt;&gt;&gt; rdd.map(lambda x: x*(x[1],x[0]))</code>            | Apply a function to each RDD element |
| <code>&gt;&gt;&gt; .collect()</code>                                  |                                      |
| <code>&gt;&gt;&gt; [(('a', 7, 'a'), ('a', 2), ('b', 2)),</code>       |                                      |
| <code>&gt;&gt;&gt; rdd5 = rdd.flatMap(lambda x: x*(x[1],x[0]))</code> |                                      |

```
>>> rdd5.collect()
```

```
>>> [(('a', 7, 'a'), ('a', 2, 'b', 2), ('b', 2, 'b'))]
```

```
>>> rdd4 = rdd.flatMapValues(lambda x: x)
```

```
>>> .collect()
```

```
>>> [('a', 'x'), ('a', 'y'), ('a', 'z'), ('b', 'p'), ('b', 'r')])
```

```
>>> Sampling
```

```
>>> rdd3.sample(False, 0.15, 81).collect()
```

```
>>> [(3, 4, 27, 31, 40, 41, 42, 43, 60, 76, 79, 80, 86, 97)]
```

```
>>> Filtering
```

```
>>> rdd.filter(lambda x: "a" in x)
```

```
>>> [(('a', 1), ('a', 2))]
```

```
>>> rdd5.distinct().collect()
```

```
>>> [('a', 2), ('b', 7)]
```

```
>>> rdd.keys().collect()
```

```
>>> ['a', 'a', 'b']
```

Return sampled subset of rdd3

Filter the RDD

Return distinct RDD values

Return (key,value) RDD's keys

Return a list with all RDD elements

Take first 2 RDD elements

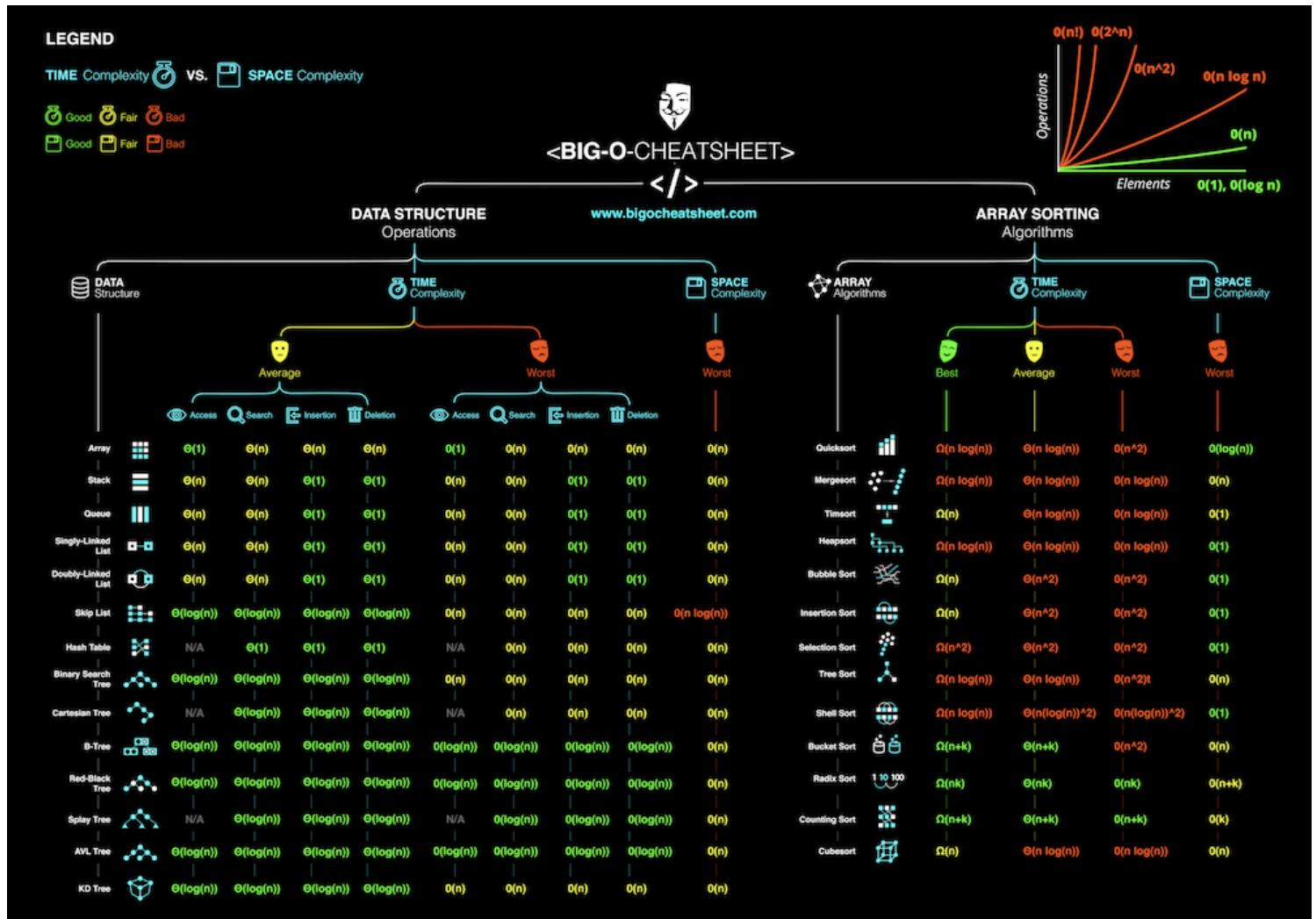
Take first RDD element

Take top 2 RDD elements

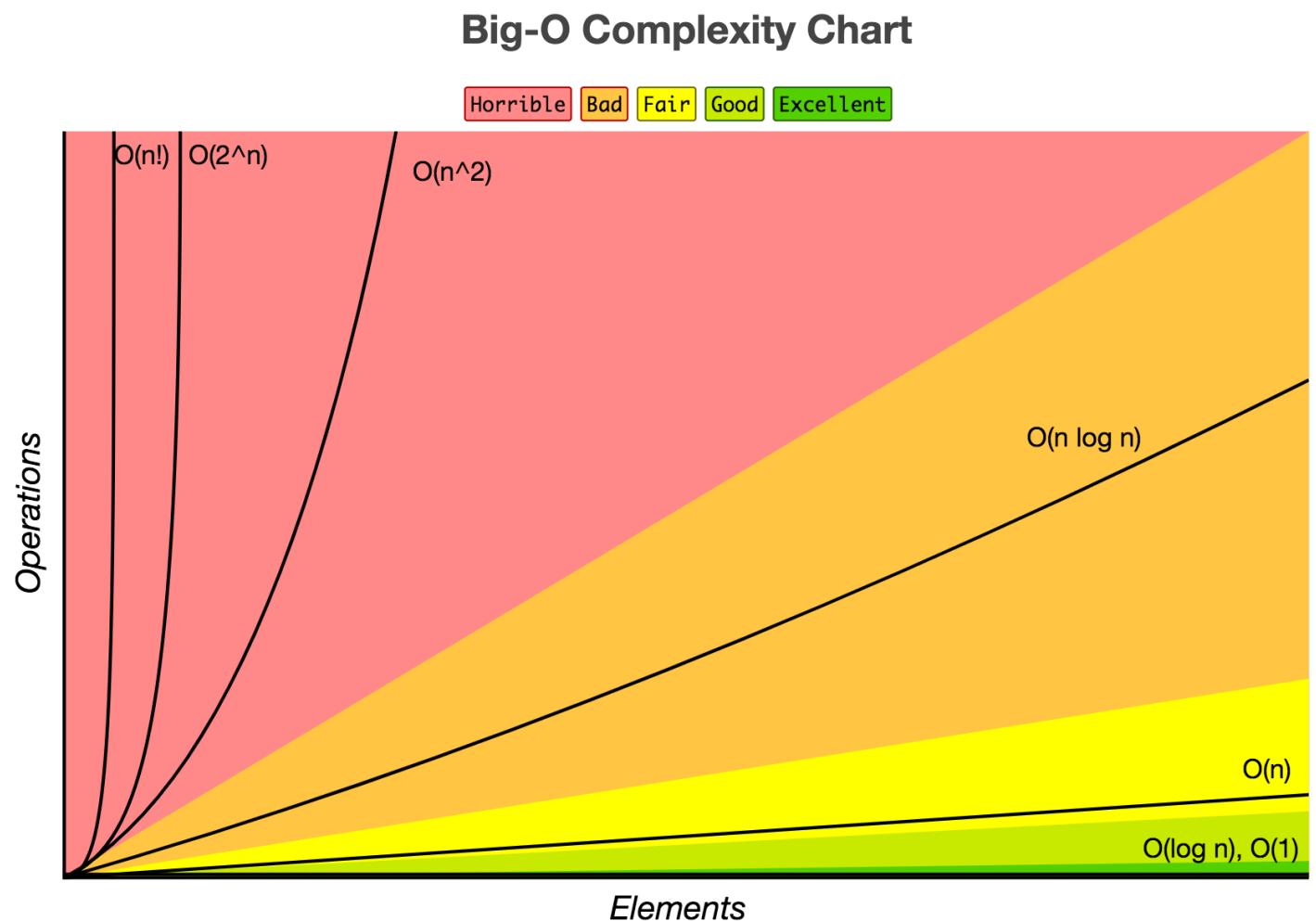
Return RDD by given function

Sort RDD by key

Sort RDD by key and value



Big-O Algorithm Cheat Sheet



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             |                  |
| <u>Quicksort</u>      | $\Omega(n \log(n))$ | $\Theta(n \log(n))$    | $O(n^2)$          | $O(\log(n))$     |
| <u>Mergesort</u>      | $\Omega(n \log(n))$ | $\Theta(n \log(n))$    | $O(n \log(n))$    | $O(n)$           |
| <u>Timsort</u>        | $\Omega(n)$         | $\Theta(n \log(n))$    | $O(n \log(n))$    | $O(n)$           |
| <u>Heapsort</u>       | $\Omega(n \log(n))$ | $\Theta(n \log(n))$    | $O(n \log(n))$    | $O(1)$           |
| <u>Bubble Sort</u>    | $\Omega(n)$         | $\Theta(n^2)$          | $O(n^2)$          | $O(1)$           |
| <u>Insertion Sort</u> | $\Omega(n)$         | $\Theta(n^2)$          | $O(n^2)$          | $O(1)$           |
| <u>Selection Sort</u> | $\Omega(n^2)$       | $\Theta(n^2)$          | $O(n^2)$          | $O(1)$           |
| <u>Tree Sort</u>      | $\Omega(n \log(n))$ | $\Theta(n \log(n))$    | $O(n^2)$          | $O(n)$           |
| <u>Shell Sort</u>     | $\Omega(n \log(n))$ | $\Theta(n(\log(n))^2)$ | $O(n(\log(n))^2)$ | $O(1)$           |
| <u>Bucket Sort</u>    | $\Omega(n+k)$       | $\Theta(n+k)$          | $O(n^2)$          | $O(n)$           |
| <u>Radix Sort</u>     | $\Omega(nk)$        | $\Theta(nk)$           | $O(nk)$           | $O(n+k)$         |
| <u>Counting Sort</u>  | $\Omega(n+k)$       | $\Theta(n+k)$          | $O(n+k)$          | $O(k)$           |
| <u>Cubesort</u>       | $\Omega(n)$         | $\Theta(n \log(n))$    | $O(n \log(n))$    | $O(n)$           |

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.

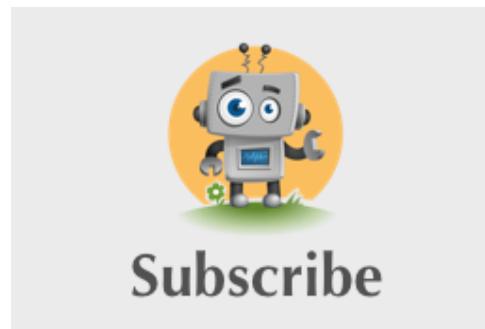
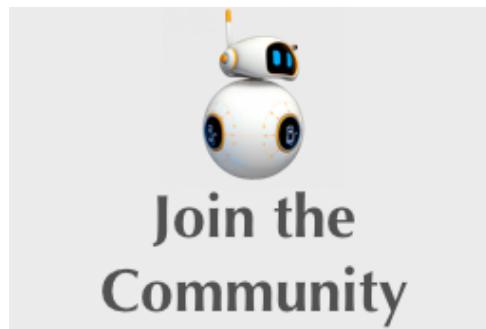
## I only write 2-3 articles per month.

[Sign up](#)

- I agree to leave Medium and submit this information, which will be collected and used according to [Upscribe's privacy policy](#).

 176  29

...



## 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](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](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>



