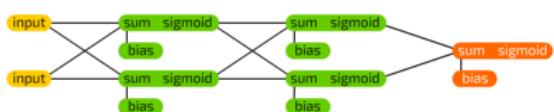


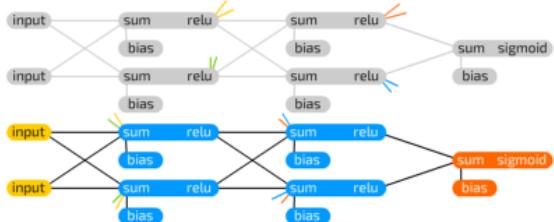
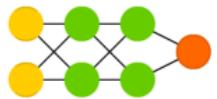
An informative chart to build

Neural Network Graphs

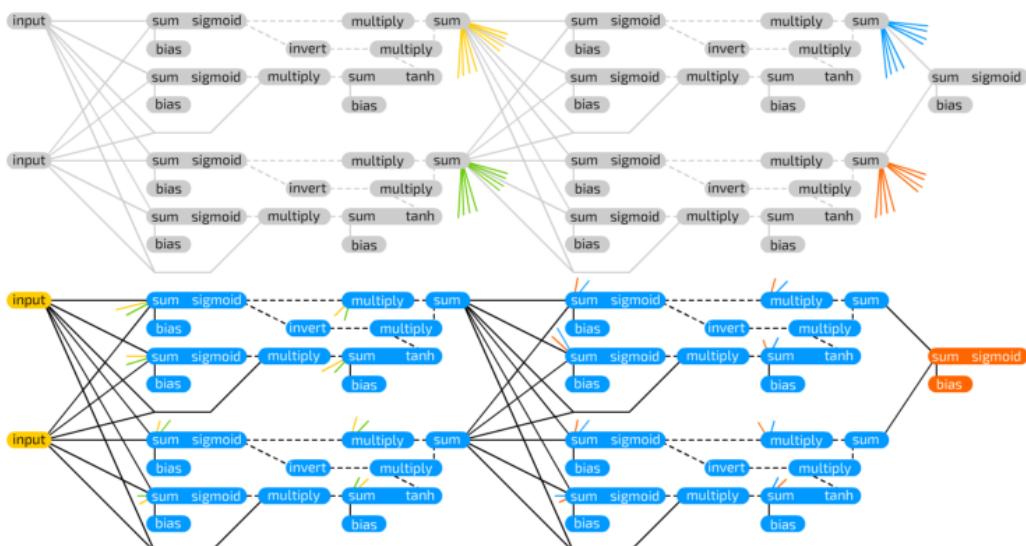
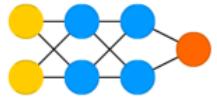
©2016 Fjodor van Veen - asimovinstitute.org



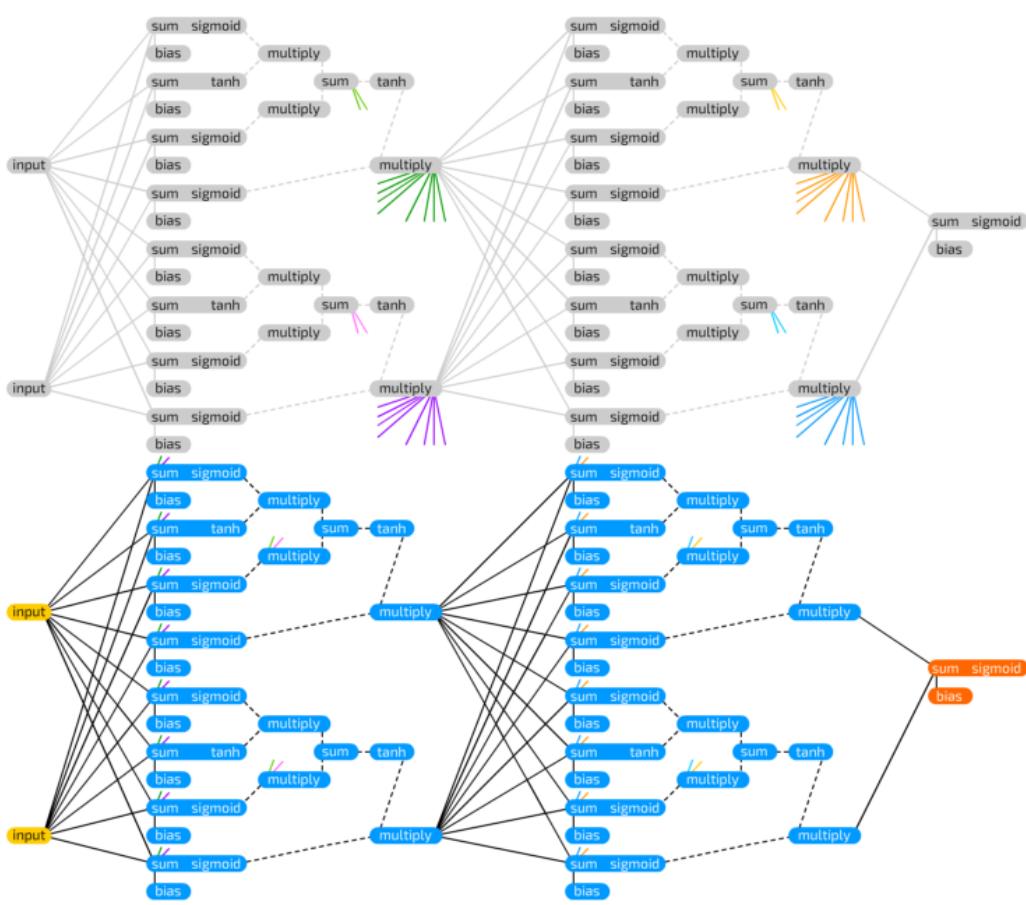
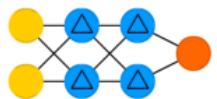
Deep Feed Forward Example



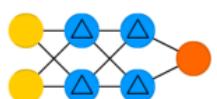
Deep Recurrent Example
(previous iteration)



Deep GRU Example
(previous iteration)



Deep LSTM Example
(previous iteration)



Deep LSTM Example

Linear Vector Spaces:

- Definition:** A linear vector space, X is a set of elements (vectors) defined over a scalar field, F , that satisfies the following conditions:
- 1) if $x \in X$ and $y \in X$ then $x+y \in X$.
 - 2) $x+y = y+x$.
 - 3) $(x+y)+z = x+(y+z)$.
 - 4) There is a unique vector $0 \in X$, such that $x+0=x$ for all $x \in X$.
 - 5) For each vector $x \in X$ there is a unique vector in X , to be called $(-x)$, such that $x+(-x)=0$.
 - 6) multiplication, for all scalars $a \in F$, and all vectors $x \in X$,
 - 7) For any $x \in X$, $1x=x$.
 - 8) For any two scalars $a \in F$ and $b \in F$ and any $x \in X$, $a(bx)=(ab)x$.
 - 9) $(a+b)x=a x+b x$.
 - 10) $a(x+y)=a x+a y$.

Linear Independence: Consider n vectors $\{x_1, x_2, \dots, x_n\}$. If there exists n scalars a_1, a_2, \dots, a_n , at least one of which is nonzero, such that $a_1x_1 + a_2x_2 + \dots + a_nx_n = 0$, then the $\{x_i\}$ are linearly dependent.

Spanning a Space:

Let X be a linear vector space and let $\{u_1, u_2, \dots, u_n\}$ be a subset of vectors in X . This subset spans X if and only if for every vector $x \in X$ there exist scalars x_1, x_2, \dots, x_n such that $x = x_1u_1 + x_2u_2 + \dots + x_nu_n$.

Inner Product: $\langle x, y \rangle$ for any scalar function of x and y .

1. $\langle x, y \rangle = \langle y, x \rangle$
2. $\langle ax_1 + by_1, z \rangle = a \langle x_1, z \rangle + b \langle y_1, z \rangle$
3. $\langle x, x \rangle \geq 0$, where equality holds iff x 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 θ bet. 2 vectors x and y is defined by $\cos \theta = \frac{\langle x, y \rangle}{\|x\| \|y\|}$

Orthogonality: 2 vectors $x, y \in X$ are said to be orthogonal if $\langle x, y \rangle = 0$.

Gram Schmidt Orthogonalization:

Assume that we have n independent vectors y_1, y_2, \dots, y_n . From these vectors we will obtain n orthogonal vectors v_1, v_2, \dots, v_n .

$$v_1 = y_1, \quad v_k = y_k - \sum_{i=1}^{k-1} \frac{\langle v_i, y_k \rangle}{\langle v_i, v_i \rangle} v_i,$$

where $\frac{\langle v_i, y_k \rangle}{\langle v_i, v_i \rangle} v_i$ is the projection of y_k on v_i

Vector Expansions:

$$x = \sum_{i=1}^n x_i v_i = x_1 v_1 + x_2 v_2 + \dots + x_n v_n.$$

$$\text{for orthogonal vectors, } x_j = \frac{\langle v_j, x \rangle}{\langle v_j, v_j \rangle}$$

Reciprocal Basis Vectors:

$$(r_i, v_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}, \quad x_j = (r_j, x)$$

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:

domain: $X = \{x_i\}$, range: $Y = \{y_i\}$, and a rule relating each $x_i \in X$ to an element $y_i \in Y$.

Linear Transformations: transformation A is linear if:

1. for all $x_1, x_2 \in X$, $A(x_1+x_2) = A(x_1) + A(x_2)$
2. for all $x \in X$, $a \in R$, $A(ax) = aA(x)$

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 : $A(x) = y$

The coefficients of the matrix representation are obtained from

$$A(v_j) = \sum_{i=1}^m a_{ij} u_i$$

Change of Basis: $B_t = [t_1 \ t_2 \ \dots \ t_n]$, $B_w = [w_1 \ w_2 \ \dots \ w_n]$

$$A' = [B_w^{-1} A B_t]$$

Eigenvalues & Eigenvectors: $Az = \lambda z$, $|(A - \lambda I)| = 0$

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:

$$a = \text{hardlim}(\mathbf{W}\mathbf{p} + \mathbf{b}), \quad \mathbf{W} = [\mathbf{w}_1^T \ \mathbf{w}_2^T \ \dots \ \mathbf{w}_n^T]^T, \quad a_i = \text{hardlim}(n_i) = \text{hardlim}(\mathbf{t}_i^T \mathbf{p} + b_i)$$

Decision Boundary: $\mathbf{w}^T \mathbf{p} + b_i = 0$

The decision boundary is always orthogonal to the weight vector. Single-layer perceptrons can only classify linearly separable vectors.

Perceptron Learning Rule

$$\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \mathbf{e}\mathbf{p}^T, \quad \mathbf{b}^{\text{new}} = \mathbf{b}^{\text{old}} + \mathbf{e}, \quad \text{where } \mathbf{e} = \mathbf{t} - \mathbf{a}$$

Hebb's Postulate: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Linear Associator: $\mathbf{a} = \text{purelin}(\mathbf{W}\mathbf{p})$

The Hebb Rule: Supervised Form: $\mathbf{w}_{ij}^{\text{new}} = \mathbf{w}_{ij}^{\text{old}} + t_{qi}P_{qi}$

$$\mathbf{W} = \mathbf{t}_1 \mathbf{P}_1^T + \mathbf{t}_2 \mathbf{P}_2^T + \dots + \mathbf{t}_Q \mathbf{P}_Q^T$$

$$\mathbf{W} = [\mathbf{t}_1 \ \mathbf{t}_2 \ \dots \ \mathbf{t}_Q] \begin{bmatrix} \mathbf{P}_1^T \\ \mathbf{P}_2^T \\ \vdots \\ \mathbf{P}_Q^T \end{bmatrix} = \mathbf{T} \mathbf{P}^T$$

Pseudoinverse Rule: $\mathbf{W} = \mathbf{T} \mathbf{P}^+$

When the number, R , of rows of \mathbf{P} is greater than the num ber of columns, Q , of \mathbf{P} and the columns of \mathbf{P} are independent, then the pseudoinverse can be computed by $\mathbf{P}^+ = (\mathbf{P}^T \mathbf{P})^{-1} \mathbf{P}^T$

Variations of Hebbian Learning:

Filtered Learning (Ch.14): $\mathbf{W}^{\text{new}} = (1 - \gamma)\mathbf{W}^{\text{old}} + \alpha \mathbf{t}_q \mathbf{p}_q^T$

Delta Rule (Ch.10): $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha (\mathbf{t}_q - \mathbf{a}_q) \mathbf{p}_q^T$

Unsupervised Hebb (Ch.13): $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha \mathbf{a}_q \mathbf{p}_q^T$

Taylor: $F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^*) \nabla^2 F(\mathbf{x})^T|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) + \dots$

Grad $\nabla F(\mathbf{x}) = \left[\frac{\partial}{\partial x_1} F(\mathbf{x}) \quad \frac{\partial}{\partial x_2} F(\mathbf{x}) \quad \dots \quad \frac{\partial}{\partial x_n} F(\mathbf{x}) \right]^T$

Hessian: $\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial}{\partial x_1 \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial}{\partial x_1 \partial x_n} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial}{\partial x_2^2} F(\mathbf{x}) & \dots & \frac{\partial}{\partial x_2 \partial x_n} F(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_n \partial x_1} F(\mathbf{x}) & \frac{\partial}{\partial x_n \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n^2} F(\mathbf{x}) \end{bmatrix}$

Directional Derivatives:

$$\text{1st Dir.Der.: } \frac{\mathbf{p}^T \nabla F(\mathbf{x})}{\|\mathbf{p}\|}, \quad \text{2nd Dir.Der.: } \frac{\mathbf{p}^T \nabla^2 F(\mathbf{x}) \mathbf{p}}{\|\mathbf{p}\|^2}$$

Minima:

Strong Minimum: if a scalar $\delta > 0$ exists, such that $F(x) < F(x + \Delta x)$ for all Δx such that $\delta > \|\Delta x\| > 0$.

Global Minimum: if $F(x) < F(x + \Delta x)$ for all $\Delta x \neq 0$

Weak Minimum: if it is not a strong minimum, and a scalar $\delta > 0$ exists, such that $F(x) \leq F(x + \Delta x)$ for all Δx such that $\delta > \|\Delta x\| > 0$.

Necessary Conditions for Optimality:

1st-Order Condition: $\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} = 0$ (Stationary Points)

2nd-Order Condition: $\nabla^2 F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^*} \geq 0$ (Positive Semi-definite Hessian Matrix).

Quadratic fn.: $F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{d}^T \mathbf{x} + c$

$$\nabla F(\mathbf{x}) = \mathbf{A} \mathbf{x} + \mathbf{d}, \quad \nabla^2 F(\mathbf{x}) = \mathbf{A}, \quad \lambda_{\min} \leq \frac{\mathbf{p}^T \mathbf{A} \mathbf{p}}{\|\mathbf{p}\|^2} \leq \lambda_{\max}$$

<p>General Minimization Algorithm: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ or $\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$</p> <p>Steepest Descent Algorithm: $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$ where, $\mathbf{g}_k = \nabla F(\mathbf{x}) _{\mathbf{x}=\mathbf{x}_k}$</p> <p>Stable Learning Rate: ($\alpha_k = \alpha$, constant) $\alpha < \frac{2}{\lambda_{max}}$</p> <p>$\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ Eigenvalues of Hessian matrix A</p> <p>Learning Rate to Minimize Along the Line: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \xrightarrow{\text{is}} \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ (For quadratic fn.)</p> <p>After Minimization Along the Line: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$</p> <p>ADALINE: $\mathbf{a} = \text{purelin}(\mathbf{W}\mathbf{p} + \mathbf{b})$</p> <p>Mean Square Error: (for ADALINE it is a quadratic fn.) $F(\mathbf{x}) = E[e^2] = E[(t - a)^2] = E[(t - \mathbf{x}^T \mathbf{z})^2]$</p> <p>$F(\mathbf{x}) = c - 2\mathbf{x}^T \mathbf{h} + \mathbf{x}^T \mathbf{R} \mathbf{x}$,</p> <p>$c = E[t^2]$, $\mathbf{h} = E[t\mathbf{z}]$ and $\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] \Rightarrow \Lambda = 2\mathbf{R}$, $d = -2\mathbf{h}$</p> <p>Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1} \mathbf{h}$,</p> <p>where $\mathbf{x} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$</p> <p>LMS Algorithm: $\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha \mathbf{e}(k) \mathbf{p}^T(k)$</p> <p>$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha \mathbf{e}(k)$</p> <p>Convergence Point: $\mathbf{x}^* = \mathbf{R}^{-1} \mathbf{h}$</p> <p>Stable Learning Rate: $0 < \alpha < 1/\lambda_{max}$ where λ_{max} is the maximum eigenvalue of R</p> <p>Adaptive Filter ADALINE:</p> $a(k) = \text{purelin}(\mathbf{W}\mathbf{p}(k) + b) = \sum_{i=1}^R w_{i,i}y(k-i+1) + b$	<p>*Heuristic Variations of Backpropagation:</p> <p>Batching: The parameters are updated 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 complete, i.e., covers all possible input/output pairs, then the gradient estimate will be exact.)</p> <p>Backpropagation with Momentum (MOBP):</p> $\Delta \mathbf{W}^m(k) = \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T$ $\Delta \mathbf{b}^m(k) = \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m$ <p>Variable Learning Rate Backpropagation (VLBP):</p> <ol style="list-style-type: none"> If the squared error (over the entire training set) increases by more than some set percentage ζ (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $\rho < 1$, and the momentum coefficient γ (if it is used) is set to zero. 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 γ has been previously set to zero, it is reset to its original value. If the squared error increases by less than ζ, then the weight update is accepted but the learning rate and the momentum coefficient are unchanged. <p>Association: $\mathbf{a} = \text{hardlim}(\mathbf{W}^0 \mathbf{P}^0 + \mathbf{W}\mathbf{p} + \mathbf{b})$</p> <p>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.</p> <p>Associative Learning Rules:</p> <p>Unsupervised Hebb Rule:</p> $\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha \mathbf{a}(q) \mathbf{p}^T(q)$ <p>Hebb with Decay:</p> $\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha \mathbf{a}(q) \mathbf{p}^T(q)$ <p>Instar: $\mathbf{a} = \text{hardlim}(\mathbf{W}\mathbf{p} + \mathbf{b})$, $\mathbf{a} = \text{hardlim}({}_1 \mathbf{w}^T \mathbf{p} + b)$</p> <p>The instar is activated for ${}_1 \mathbf{w}^T \mathbf{p} = \ {}_1 \mathbf{w} \ \ \mathbf{p} \ \cos\theta \geq -b$ where θ is the angle between \mathbf{p} and ${}_1 \mathbf{w}$.</p> <p>Instar Rule:</p> ${}_i \mathbf{w}(q) = {}_i \mathbf{w}(q-1) + \alpha a_i(q)(\mathbf{p}(q) - {}_i \mathbf{w}(q-1))$ ${}_i \mathbf{w}(q) = (1-\alpha) {}_i \mathbf{w}(q-1) + \alpha \mathbf{p}(q), \text{ if } (a_i(q) = 1)$ <p>Kohonen Rule:</p> ${}_i \mathbf{w}(q) = {}_i \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - {}_i \mathbf{w}(q-1)) \text{ for } i \in X(q)$ <p>Outstar Rule: $\mathbf{a} = \text{satlim}(\mathbf{W}\mathbf{p})$</p> $\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha (a(q) - \mathbf{w}_j(q-1)) p_j(q)$ <p>Competitive Layer: $\mathbf{a} = \text{compet}(\mathbf{W}\mathbf{p}) = \text{compet}(\mathbf{n})$</p> <p>Competitive Learning with the Kohonen Rule:</p> ${}_{i^*} \mathbf{w}(q) = {}_{i^*} \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - {}_{i^*} \mathbf{w}(q-1))$ $= (1-\alpha) {}_{i^*} \mathbf{w}(q-1) + \alpha \mathbf{p}(q)$ <p>$i^* \mathbf{w}(q) = {}_{i^*} \mathbf{w}(q-1)$, $i \neq i^*$ where i^* is the winning neuron.</p> <p>Self-Organizing with the Kohonen Rule:</p> ${}_{i^*} \mathbf{w}(q) = {}_{i^*} \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - {}_{i^*} \mathbf{w}(q-1))$ $= (1-\alpha) {}_{i^*} \mathbf{w}(q-1) + \alpha \mathbf{p}(q)$ $N_i(d) = \{j, d_{i,j} \leq d\}$ <p>LVO Network: $(w_{k,i}^2 = 1) \Rightarrow$ subclass i is a part of class k</p> $n_i^1 = -\ {}_i \mathbf{w}^1 - \mathbf{p} \ , \mathbf{a}^1 = \text{compet}(\mathbf{n}^1), \mathbf{a}^2 = \mathbf{W}^2 \mathbf{a}^1$ <p>LVQ Network Learning with the Kohonen Rule:</p> ${}_{i^*} \mathbf{w}^1(q) = {}_{i^*} \mathbf{w}^1(q-1) + \alpha (\mathbf{p}(q) - {}_{i^*} \mathbf{w}^1(q-1)),$ $\text{if } a_{k^*}^2 = t_{k^*} = 1$ ${}_{i^*} \mathbf{w}^1(q) = {}_{i^*} \mathbf{w}^1(q-1) - \alpha (\mathbf{p}(q) - {}_{i^*} \mathbf{w}^1(q-1)),$ $\text{if } a_{k^*}^2 = 1 \neq t_{k^*} = 0$
<p>Backpropagation Algorithm:</p> <p>Performance Index:</p> <p>Mean Square error: $F(\mathbf{x}) = E[e^2] = E[(t - a)^2] = E[(t - \mathbf{x}^T \mathbf{z})^2]$</p> <p>Approximate Performance Index: (single sample)</p> <p>$\hat{F}(x) = \mathbf{e}^T(k) \mathbf{e}(k) = (\mathbf{t}(k) - \mathbf{a}(k))^T (\mathbf{t}(k) - \mathbf{a}(k))$</p> <p>Sensitivity: $\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left[\frac{\partial \hat{F}}{\partial n_1^m} \quad \frac{\partial \hat{F}}{\partial n_2^m} \quad \dots \quad \frac{\partial \hat{F}}{\partial n_s^m} \right]^T$</p> <p>Forward Propagation: $\mathbf{a}^0 = \mathbf{p}$,</p> <p>$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1} \mathbf{a}^m + \mathbf{b}^{m+1})$ for $m = 0, 1, \dots, M-1$</p> <p>$\mathbf{a} = \mathbf{a}^M$</p> <p>Backward Propagation: $\mathbf{s}^M = -2\mathbf{f}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$,</p> <p>$\mathbf{s}^m = \mathbf{f}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}$ for $m = M-1, \dots, 2, 1$, where</p> <p>$\mathbf{f}^m(\mathbf{n}^m) = \text{diag}([f^m(n_1^m) \quad f^m(n_2^m) \quad \dots \quad f^m(n_s^m)])$</p> <p>$f^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$</p> <p>Weight Update (Approximate Steepest Descent):</p> <p>$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T$</p> <p>$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$</p> <p>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: $a = \frac{1}{1+e^{-n}}$, tansig: $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$, postlin: $a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$</p> <p>compet: $a = \begin{cases} 1 & \text{neuron with max } n \\ 0 & \text{all other neurons} \end{cases}$, satlin: $a = \begin{cases} 0 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$, satlim: $a = \begin{cases} -1 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$</p> <p>Delay: $a(t) = u(t-1)$, Integrator: $a(t) = \int_0^t u(\tau) d\tau + a(0)$</p>	<p>HINT:</p> $\text{diag}([1 \ 2 \ 3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

MACHINE LEARNING IN EMOJI

SUPERVISED

UNSUPERVISED

REINFORCEMENT

	SUPERVISED	human builds model based on input / output human input, machine output
	UNSUPERVISED	human utilizes if satisfactory human input, machine output
	REINFORCEMENT	human reward/punish, cycle continues

BASIC REGRESSION

	LINEAR	linear_model.LinearRegression() Lots of numerical data	
	LOGISTIC	linear_model.LogisticRegression() Target variable is categorical	or

CLASSIFICATION

	NEURAL NET	neural_network.MLPClassifier() Complex relationships. Prone to overfitting Basically magic.	
	K-NN	neighbors.KNeighborsClassifier() Group membership based on proximity	
	DECISION TREE	tree.DecisionTreeClassifier() If/then/else. Non-contiguous data. Can also be regression	
	RANDOM FOREST	ensemble.RandomForestClassifier() Find best split randomly Can also be regression	
	SVM	svm.SVC() svm.LinearSVC() Maximum margin classifier. Fundamental Data Science algorithm	
	NAIVE BAYES	GaussianNB() MultinomialNB() BernoulliNB() Updating knowledge step by step with new info	

CLUSTER ANALYSIS

	K-MEANS	cluster.KMeans()	
	ANOMALY DETECTION	covariance. EllipticalEnvelope()	

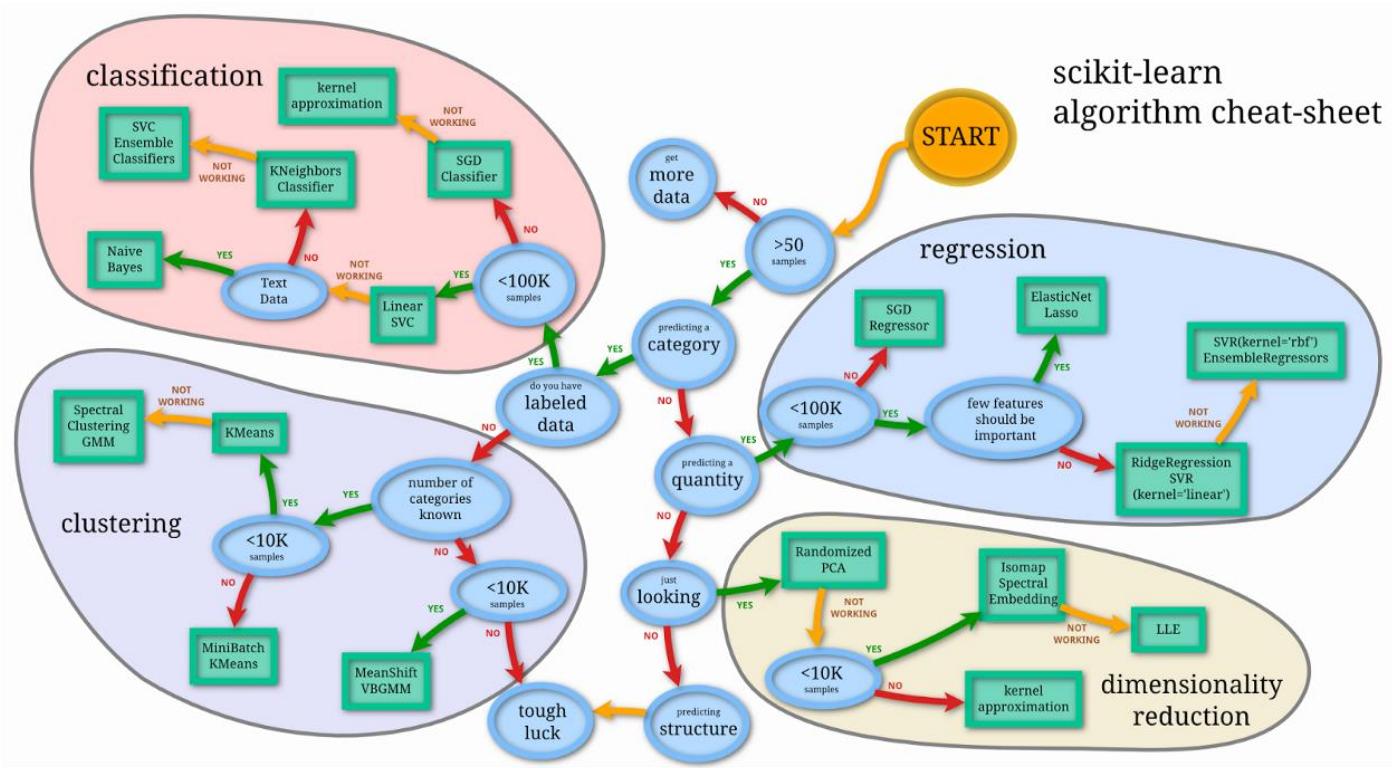
FEATURE REDUCTION

T-DISTRIBUTED STOCHASTIC NEIGHBOR EMBEDDING	manifold.TSNE()	
Visualize high dimensional data. Convert similarity to joint probabilities		
PRINCIPAL COMPONENT ANALYSIS	decomposition.PCA()	
Distill feature space into components that describe greatest variance		
CANONICAL CORRELATION ANALYSIS	decomposition.CCA()	
Making sense of cross-correlation matrices		
LINEAR DISCRIMINANT ANALYSIS	lda.LDA()	
Linear combination of features that separates classes		

OTHER IMPORTANT CONCEPTS

BIAS VARIANCE TRADEOFF	
UNDERFITTING / OVERFITTING	
INERTIA	
ACCURACY FUNCTION	$(TP + TN) / (P + N)$
Precision Function	$TP / (TP + FP)$
Specificity Function	$TN / (FP + TN)$
Sensitivity Function	$TP / (TP + FN)$

@emilyinamillion made this



Python For Data Science Cheat Sheet

Scikit-Learn

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



Scikit-learn

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

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.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=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

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

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

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((2,5)))
>>> y_pred = lr.predict(X_test)
>>> y_pred = knn.predict(X_test)
```

Predict labels

```
Predict labels
Estimate probability of a label
```

Unsupervised Estimators

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

Predict labels in clustering algos

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score <code>>>> from sklearn.metrics import accuracy_score >>> accuracy_score(y_test, y_pred)</code>	Estimator score method Metric scoring functions
Classification Report <code>>>> from sklearn.metrics import classification_report >>> print(classification_report(y_test, y_pred))</code>	Precision, recall, f1-score and support
Confusion Matrix <code>>>> from sklearn.metrics import confusion_matrix >>> print(confusion_matrix(y_test, y_pred))</code>	

Regression Metrics

Mean Absolute Error <code>>>> from sklearn.metrics import mean_absolute_error >>> y_true = [3, -0.5, 2] >>> mean_absolute_error(y_true, y_pred)</code>	
Mean Squared Error <code>>>> from sklearn.metrics import mean_squared_error >>> mean_squared_error(y_true, y_pred)</code>	
R² Score <code>>>> from sklearn.metrics import r2_score >>> r2_score(y_true, y_pred)</code>	

Clustering Metrics

Adjusted Rand Index <code>>>> from sklearn.metrics import adjusted_rand_score >>> adjusted_rand_score(y_true, y_pred)</code>	
Homogeneity <code>>>> from sklearn.metrics import homogeneity_score >>> homogeneity_score(y_true, y_pred)</code>	
V-measure <code>>>> from sklearn.metrics import v_measure_score >>> metrics.v_measure_score(y_true, y_pred)</code>	

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

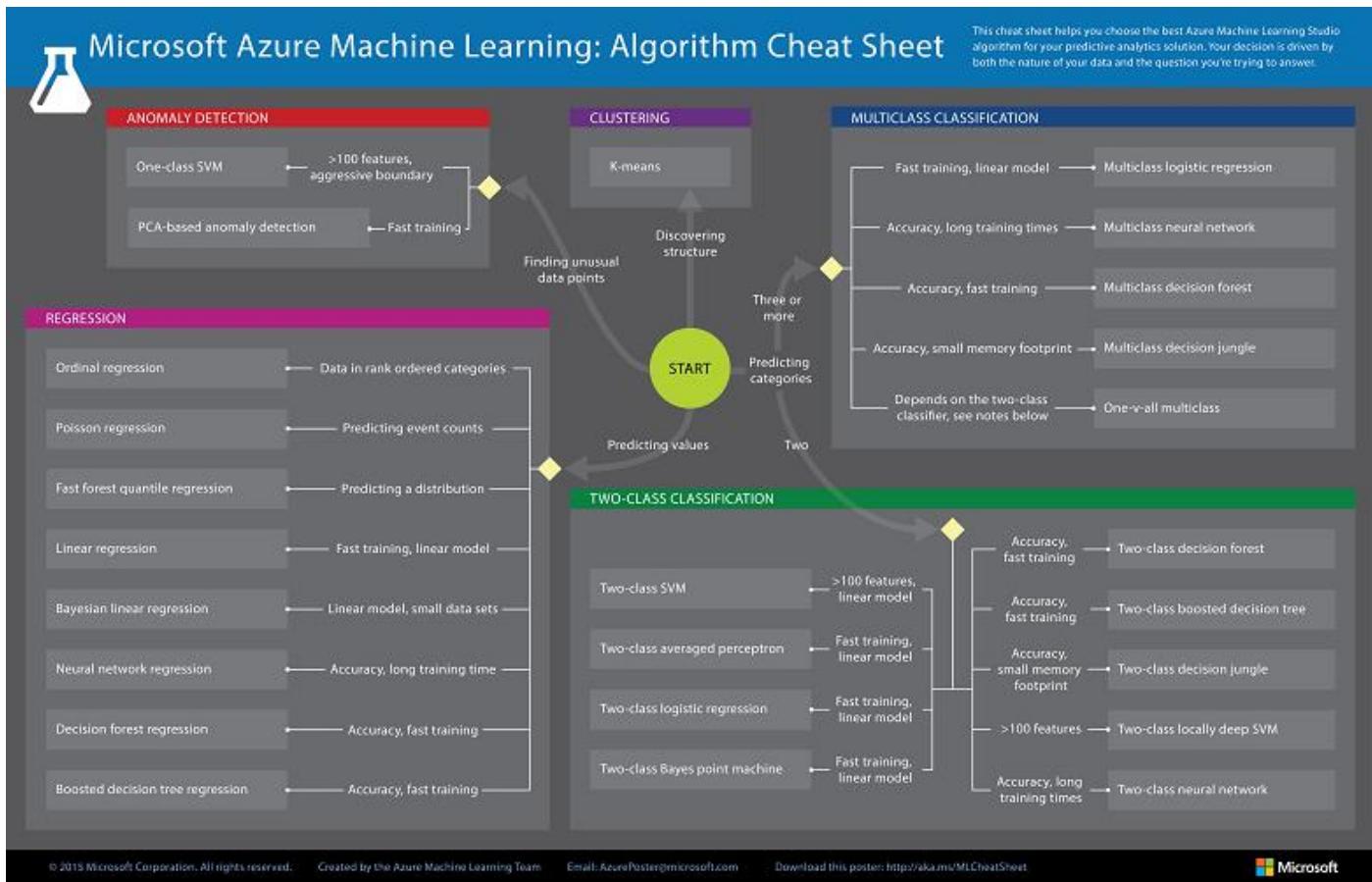
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
...            "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
...                      param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)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

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
...            "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
...                               param_distributions=params,
...                               cv=4,
...                               n_iter=8,
...                               random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science interactively





© 2015 Microsoft Corporation. All rights reserved.

Created by the Azure Machine Learning Team

Email: AzurePoster@microsoft.com

Download this poster: <http://aka.ms/MLCheatSheet>



Python For Data Science Cheat Sheet

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com



Variables and Data Types

Variable Assignment

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

Calculations With Variables

<code>>>> x+2</code>	Sum of two variables
<code>>>> x-2</code>	Subtraction of two variables
<code>>>> x*2</code>	Multiplication of two variables
<code>>>> x**2</code>	Exponentiation of a variable
<code>>>> x%2</code>	Remainder of a variable
<code>>>> x/float(2)</code>	Division of a variable
<code>2.5</code>	

Types and Type Conversion

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

Asking For Help

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

Strings

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

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Init'
'thisStringIsAwesomeInit'
>>> 'm' in my_string
True
```

Lists

Also see NumPy Arrays

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

Selecting List Elements

Index starts at 0

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

List Operations

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

List Methods

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

String Operations

Index starts at 0

String Methods

<code>my_string.upper()</code>	String to uppercase
<code>my_string.lower()</code>	String to lowercase
<code>my_string.count('*')</code>	Count String elements
<code>my_string.replace('*', 'i')</code>	Replace String elements
<code>my_string.strip()</code>	Strip whitespace from ends

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
Selective import
>>> from math import pi
```

pandas

Data analysis

Machine learning

NumPy

Scientific computing

matplotlib

2D plotting

Install Python



Leading open data science platform

powered by Python



Free IDE that is included

with Anaconda



Create and share

documents with live code,

visualizations, text, ...

Numpy Arrays

Also see Lists

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

Selecting Numpy Array Elements

Index starts at 0

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

Numpy Array Operations

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

Numpy Array Functions

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

DataCamp

Learn Python for Data Science Interactively

Python For Data Science Cheat Sheet

Bokeh

Learn Bokeh [Interactively](#) at [www.DataCamp.com](#), taught by Bryan Van de Ven, core contributor



Plotting With Bokeh

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



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 3
>>> output_file("lines.html")    Step 4
>>> show(p)                    Step 5
```

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'])
```

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

Nesting Rows & Columns

```
>>> layout = row(column(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 Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Legends

Legend Location

Inside Plot Area
`p.legend.location = 'bottom_left'`

Outside Plot Area

```
>>> r1 = p2.add_rect(np.array([1,2,3]), np.array([3,2,1]))
>>> r2 = p2.add_line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One", r1), ("Two", r2)], location=(-30, -30))
```

```
>>> p.add_layout(legend, 'right')
```

4 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

>>> show(p1)	>>> save(p1)
>>> show(layout)	>>> save(layout)

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

Nesting Rows & Columns

```
>>> layout = row(column(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 Tabs
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = Tabs(tabs=[tab1, tab2])
```

Legends

Legend Location

Inside Plot Area
`p.legend.location = 'bottom_left'`

Outside Plot Area

```
>>> r1 = p2.add_rect(np.array([1,2,3]), np.array([3,2,1]))
>>> r2 = p2.add_line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One", r1), ("Two", r2)], location=(-30, -30))
```

```
>>> p.add_layout(legend, 'right')
```

4 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

>>> show(p1)	>>> save(p1)
>>> show(layout)	>>> save(layout)

Customized Glyphs

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)
```

Also see Data

Legend Location

Legend Orientation

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

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

Legend Background & Border

```
>>> p.legend.border_line_color = "navy"
```

```
>>> p.legend.background_fill_color = "white"
```

Also see Data

Statistical Charts With Bokeh

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

Bar Chart

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

Box Plot

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

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

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

DataCamp

Learn Python for Data Science [Interactively](#)



About

TensorFlow

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

Skflow

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

Keras

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

Installation

How to install new package in Python:

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

Example: pip install requests

How to install tensorflow?

```
device = cpu/gpu
python_version = cp27/cp34
sudo pip install
https://storage.googleapis.com/tensorflow/linux/$device/tensorflow-0.8.0-$python_version-none-linux_x86_64.whl
```

How to install Skflow

```
pip install sklearn
```

How to install Keras

```
pip install keras
```

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

Helpers

Python helper

Important functions

`type(object)`

Get object type

`help(object)`

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

`dir(object)`

Get list of object attributes (fields, functions)

`str(object)`

Transform an object to string

`object?`

Shows documentations about the object

`globals()`

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

`locals()`

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

`id(object)`

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

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

Other built-in functions

TensorFlow

Main classes

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

Some useful functions

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

TensorFlow Optimizers

```
GradientDescentOptimizer
```

```
AdadeltaOptimizer
```

```
AdagradOptimizer
```

```
MomentumOptimizer
```

```
AdamOptimizer
```

```
FtrlOptimizer
```

```
RMSPropOptimizer
```

Reduction

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

Activation functions

```
tf.nn?
```

```
relu
```

```
relu6
```

```
elu
```

```
softplus
```

```
softsign
```

```
dropout
```

```
bias_add
```

```
sigmoid
```

```
tanh
```

```
sigmoid_cross_entropy_with_logits
```

```
softmax
```

```
log_softmax
```

```
softmax_cross_entropy_with_logits
```

```
sparse_softmax_cross_entropy_with_logits
```

```
weighted_cross_entropy_with_logits
```

etc.

Skflow

Main classes

```
TensorFlowClassifier
```

```
TensorFlowRegressor
```

```
TensorFlowDNNClassifier
```

```
TensorFlowDNNRegressor
```

```
TensorFlowLinearClassifier
```

```
TensorFlowLinearRegressor
```

```
TensorFlowRNNClassifier
```

```
TensorFlowRNNRegressor
```

TensorFlowEstimator

Each classifier and regressor have following fields

`n_classes=0` (Regressor), `n_classes` are expected to be input (Classifiers)

`batch_size=32,`

`steps=200, // except`

`TensorFlowRNNClassifier - there is 50`

`optimizer='Adagrad',`

`learning_rate=0.1,`

Python For Data Science Cheat Sheet

Keras

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



Keras

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

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.randint(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,
    cifar10,
    imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train,y_train),(x_test,y_test) = boston_housing.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

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

Preprocessing

Sequence Padding

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

One-Hot Encoding

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

Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
    input_dim=8,
    kernel_initializer='uniform',
    activation='relu'))
>>> 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(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_train)
>>> standardized_x_train = scaler.transform(x_train)
>>> standardized_x_test = scaler.transform(x_test)
```

Inspect Model

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

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

Compile Model

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

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

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

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

Model Training

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

Evaluate Your Model's Performance

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

Prediction

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

Save / Reload Models

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

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(optimizer=opt,
    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



Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science interactively at www.DataCamp.com



Pandas

The Pandas library is built on NumPy and provides easy-to-use data structures and data analysis tools for the Python programming language.



Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

Series

A one-dimensional labeled array capable of holding any data type



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

DataFrame

Columns

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

```
Index
```

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

```
>>> 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)
```

Also see NumPy Arrays

Selection

Getting

```
>>> s['b']
->
>>> 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
0 Belgium
```

Select single value by row & column

By Label

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

Select single value by row & column labels

By Label/Position

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

Select single row of subset of rows

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

Select a single column of subset of columns

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

Select rows and columns

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

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

Boolean Indexing

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

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

Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index a of Series s to 6

Read and Write to SQL Query or Database Table

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

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

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)

Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index(by='Country')
>>> s.order()
>>> df.rank()
```

Sort by row or column index

Sort a series by its values

Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)

Describe index

Describe DataFrame columns

Info on DataFrame

Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idmin()/df.idmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values

Cumulative sum of values

Minimum/maximum value

Minimum/Maximum index value

Summary statistics

Mean of values

Median of values

Applying Functions

```
>>> f = lambda x: x*x
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function

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'])
>>> s3
  a    10.0
  b    NAN
  c     5.0
  d     7.0
>>> s3.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c     5.0
  d     7.0
>>> s3.sub(s3, fill_value=2)
  a     0.0
  b    -4.0
  c     5.0
  d     7.0
>>> s3.div(s3, fill_value=4)
  a     0.0
  b    -0.5
  c     1.0
  d     1.75
>>> s3.mul(s3, fill_value=3)
```

DataCamp

Learn Python for Data Science interactively



Python For Data Science Cheat Sheet

NumPy Basics

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



NumPy

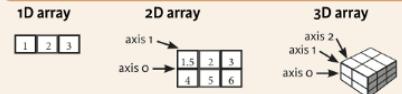
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

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



NumPy Arrays



Creating Arrays

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

Initial Placeholders

```
>>> np.zeros((3, 4))          Create an array of zeros
>>> np.ones((2, 3, 4), dtype=np.int16) Create an array of ones
>>> d = np.arange(10, 25, 5) Create an array of evenly spaced values (start value)
Create an array of evenly spaced values (number of samples)
>>> np.linspace(0, 2, 9) Create an array of evenly spaced values (number of samples)
>>> e = np.full((2, 2), 7) Create a constant array
>>> f = np.eye(2) Create a 2x2 identity matrix
>>> np.random.random((2, 2)) Create an array with random values
>>> np.empty((3, 2)) Create an empty array
```

I/O

Saving & Loading On Disk

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

Saving & Loading Text Files

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

Data Types

```
>>> np.int64          Signed 64-bit integer type
>>> np.float32         Standard double-precision floating point
>>> np.complex         Complex numbers represented by 128 floats
>>> np.bool_           Boolean type storing TRUE and FALSE values
>>> np.object          Python object type
>>> np.string          Fixed-length string type
>>> np.unicode          Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape          Array dimensions
>>> len(a)           Length of array
>>> b.ndim           Number of array dimensions
>>> e.size           Number of array elements
>>> b.dtype          Data type of array elements
>>> b.dtype.name     Name of data type
>>> b.astype(int)    Convert an array to a different type
```

Asking For Help

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

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
>>> array([-0.5, 0., 0.], [-3., -3., -3.])
>>> np.subtract(a,b)
>>> b + a
>>> array([[ 2.5,  4.,  6.],
       [ 5.,  7.,  9.]]])
>>> np.add(b,a)
>>> a / b
>>> array([ 0.66666667,  1.,  1.5,  2.,  3.,  4.], [ 0.25,  0.4,  0.5,  0.6,  0.7,  0.8])
>>> np.divide(a,b)
>>> a * b
>>> array([ 1.5,  4.,  9.], [ 4.,  10.,  18.])
>>> 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.]])
```

Subtraction

Addition

Division

Multiplication

```
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product
```

Comparison

```
>>> a == b
>>> array([[False,  True,  True],
       [False,  False,  False]], dtype=bool)
>>> a < 2
>>> array[[True, False, False], dtype=bool]
>>> np.array_equal(a, b)
```

Element-wise comparison

Element-wise comparison

Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.corrcoef()
>>> np.std(b)
```

Array-wise sum

Array-wise minimum value

Maximum value of an array row

Cumulative sum of the elements

Mean

Median

Correlation coefficient

Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data

Create a copy of the array

Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array

Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see [Lists](#)

Subsetting

1	2	3
13	2	1
4	5	6

Select the element at the 2nd index

Slicing

1	2	3
13	2	1
4	5	6

Select the element at row 0 column 2 (equivalent to b[1][2])

Reversed array

1	2	3
13	2	1
4	5	6

Reversed array a

Fancy Indexing

1	2	3
13	2	1
4	5	6

Select elements from a less than 2

Select elements

1	0	0	1	1	2
13	2	1	0	1	2
4	5	6	1	2	3

Select elements (1,0,0,0,1,1,2) and (0,0)

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

1	2	3
13	2	1
4	5	6

Permute array dimensions

Changing Array Shape

1	2	3
13	2	1
4	5	6

Flatten the array

Adding/Removing Elements

1	2	3
13	2	1
4	5	6

Return a new array with shape (2,6)

Combining Arrays

1	2	3
13	2	1
4	5	6

Append items to an array

Stacking Arrays

1	2	3
13	2	1
4	5	6

Insert items in an array

Deleting Items

1	2	3
13	2	1
4	5	6

Delete items from an array

Concatenating Arrays

1	2	3
13	2	1
4	5	6

Concatenate arrays

Stacking vertically (row-wise)

1	2	3
13	2	1
4	5	6

Stack arrays vertically (row-wise)

Stacking horizontally (column-wise)

1	2	3
13	2	1
4	5	6

Create stacked column-wise arrays

Creating Stacked Column-Wise Arrays

1	2	3
13	2	1
4	5	6

Create stacked column-wise arrays

Splitting Arrays

1	2	3
13	2	1
4	5	6

Split the array horizontally at the 3rd index

Splitting vertically at the 2nd index

1	2	3
13	2	1
4	5	6

Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science **Interactively**



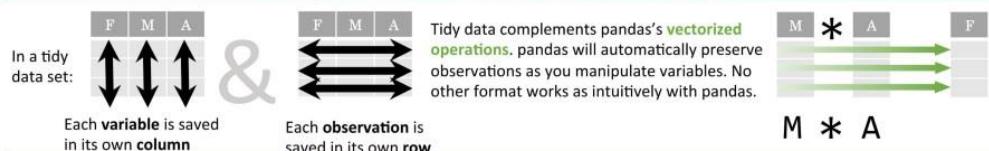
Data Wrangling

with pandas

Cheat Sheet

<http://pandas.pydata.org>

Tidy Data – A foundation for wrangling in pandas



M * A

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	

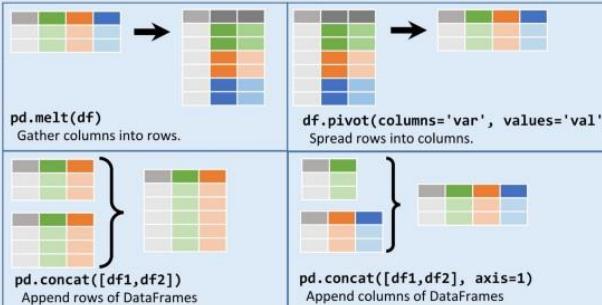
```
df = pd.DataFrame(
    {"a" : [4, 5, 6],
     "b" : [7, 8, 9],
     "c" : [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
        names=['n','v']))
Create DataFrame with a MultiIndex
```

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).

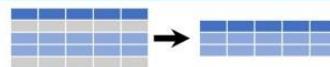
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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)



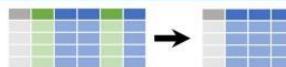
```
df[df.Length > 7]
Extract rows that meet logical criteria.

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

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

regex (Regular Expressions) Examples

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

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

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

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

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

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

Summarize Data

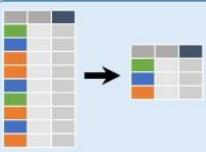
```
df['w'].value_counts()
Count number of rows with each unique value of variable
len(df)
# of rows in DataFrame.
df['w'].nunique()
# of distinct values in a column.
df.describe()
Basic descriptive statistics for each column (or GroupBy)
```



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

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

Group Data



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

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

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

Windows

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

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

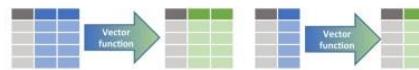
Handling Missing Data

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

Make New Columns



```
df.assign(Area=lambda df: df.Length*df.Height)
Compute and append one or more new columns.
df['Volume'] = df.Length*df.Height*df.Depth
Add single column.
pd.qcut(df.col, n, labels=False)
Bin column into n buckets.
```



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

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

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

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

Plotting

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



Combine Data Sets

adf	x1 x2	bdf	x1 x3
A 1	T	A T	
B 2	F	B F	
C 3	NaN	D T	



Standard Joins

x1 x2	x3	pd.merge(adf, bdf, how='left', on='x1')
A 1	T	Join matching rows from bdf to adf.
B 2	F	
C 3	NaN	

x1 x2	x3	pd.merge(adf, bdf, how='right', on='x1')
A 1.0	T	Join matching rows from adf to bdf.
B 2.0	F	
D NaN	T	

x1 x2	x3	pd.merge(adf, bdf, how='inner', on='x1')
A 1	T	Join data. Retain only rows in both sets.
B 2	F	

x1 x2	x3	pd.merge(adf, bdf, how='outer', on='x1')
A 1	T	Join data. Retain all values, all rows.
B 2	F	
C 3	NaN	

Filtering Joins

x1 x2	adf[adf.x1.isin(bdf.x1)]
A 1	All rows in adf that have a match in bdf.
B 2	

x1 x2	adf[~adf.x1.isin(bdf.x1)]
C 3	All rows in adf that do not have a match in bdf.

ydf	x1 x2	zdf	x1 x2
A 1	T	B 2	
B 2	F	C 3	
C 3	NaN	D 4	



Set-like Operations

x1 x2	pd.merge(ydf, zdf)
B 2	Rows that appear in both ydf and zdf (Intersection).
C 3	

x1 x2	pd.merge(ydf, zdf, how='outer')
A 1	Rows that appear in either or both ydf and zdf (Union).
B 2	
C 3	
D 4	

x1 x2	pd.merge(ydf, zdf, how='outer', indicator=True)
A 1	.query('_merge == "left_only"')
B 2	.drop(['_merge'], axis=1)
C 3	
D 4	Rows that appear in ydf but not zdf (Setdiff).

Data Wrangling

with dplyr and tidyverse

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      0.2 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).

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

devtools::install_github("rstudio/EDAWR") for data sets

Tidy Data - A foundation for wrangling in R

In a tidy data set:



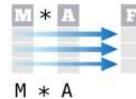
Each variable is saved in its own column

&



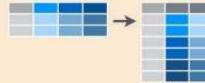
Each observation is saved in its own row

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



$M * A \rightarrow F$

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.

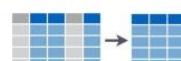
`dplyr::slice(iris, 10:15)`

Select rows by position.

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

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

Subset Variables (Columns)



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

Select columns by name or helper function.

Helper functions for select - ?select

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

Select columns whose name contains a character string.

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

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

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

Select columns whose name matches a regular expression.

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

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

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

Select columns whose names are in a group of names.

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

Select columns whose name starts with a character string.

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

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

`select(iris, -Species)`

Select all columns except Species.

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

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

Learn more with

`browseVignettes(package = c("dplyr", "tidyverse"))` • dplyr 0.4.0 • tidyverse 0.2.0 • Updated: 1/15

Summarise Data



`dplyr::summarise(iris, avg = mean(Sepal.Length))`

Summarise data into single row of values.

`dplyr::summarise_each(iris, funs(mean))`

Apply summary function to each column.

`dplyr::count(iris, Species, wt = Sepal.Length)`

Count number of rows with each unique value of variable (with or without weights).



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

`dplyr::first`

First value of a vector.

`min`

Minimum value in a vector.

`max`

Maximum value in a vector.

`dplyr::nth`

Nth value of a vector.

`mean`

Mean value of a vector.

`dplyr::n`

of values in a vector.

`median`

Median value of a vector.

`dplyr::n_distinct`

of distinct values in 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.



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

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 get 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`

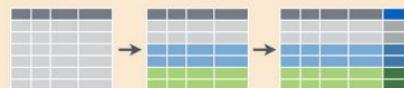
Element-wise max

`pmin`

Element-wise min

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

Compute new variables by group.



`devtools::install_github("rstudio/EDAWR")` for data sets

Combine Data Sets



Mutating Joins

`x1 | x2 | x3`

A 1 | T

B 2 | F

C 3 | NA

D NA | T

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

Join matching rows from b to a.

`x1 | x2 | x3`

A 1 | T

B 2 | F

C 3 | NA

D NA | T

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

Join matching rows from a to b.

`x1 | x2 | x3`

A 1 | T

B 2 | F

C 3 | NA

D NA | T

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

Join data. Retain only rows in both sets.

`x1 | x2 | x3`

A 1 | T

B 2 | F

C 3 | NA

D NA | T

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

Join data. Retain all values, all rows.

Filtering Joins

`x1 | x2`

A 1

B 2

C 3

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

All rows in a that have a match in b.

`x1 | x2`

C 3

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

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

Set Operations

`y | z`

A 1

B 2

C 3

`x1 | x2`

B 2

C 3

D 4

`dplyr::intersect(y, z)`

Rows that appear in both y and z.

`dplyr::union(y, z)`

Rows that appear in either or both y and z.

`dplyr::setdiff(y, z)`

Rows that appear in y but not z.

Binding

`x1 | x2`

A 1

B 2

C 3

`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.

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

Python For Data Science Cheat Sheet

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at 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([(1-5), 2], [(4, 5, 6)])
>>> c = np.array([(1.5, 2, 3), (4, 5, 6)], [(3, 2, 1), (4, 5, 6)])
```

Index Tricks

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

Shape Manipulation

```
>>> np.transpose(b)
Permute array dimensions
Flatens the array
>>> np.hstack((b, c))
Stack arrays horizontally (column-wise)
>>> np.vstack((a, b))
Stack arrays vertically (row-wise)
>>> np.split(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)
>>> q[3:] += np.pi
Unwrap
>>> np.linspace(0,10,3)
Create an array of evenly spaced values along scale
>>> np.select([(a<4), (a>2)],
Return values from a list of arrays depending on
conditions
Factorial
Combine N things taken at k time
Weights for N-point central derivative
Find the n-th derivative of a function at a point
>>> misc.factorial(a)
>>> misc.comb(10,3,exact=True)
>>> misc.central_diff_weights(3)
>>> misc.derivative(myfunc,1.0)
```

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

Basic Matrix Routines

Inverse	Inverse
>>> A.I	>>> linalg.inv(A)
Transposition	Transpose matrix
>>> A.T	>>> A.H
Trace	Conjugate transposition
>>> np.trace(A)	>>> np.trace(A)
Norm	Trace
>>> linalg.norm(A)	Frobenius norm
>>> linalg.norm(A,1)	L1 norm (max column sum)
>>> linalg.norm(A,np.inf)	Linf norm (max row sum)
Rank	Matrix rank
>>> np.linalg.matrix_rank(C)	Determinant
Determinant	Solver for dense matrices
>>> linalg.det(A)	Solver for dense matrices
Solving linear problems	Least-squares solution to linear matrix equation
>>> linalg.solve(A, b)	>>> E = np.mat('a',T) >>> linalg.lstsq(F,E)
Generalized inverse	Compute the pseudo-inverse of a matrix (least-squares solver)
>>> linalg.pinv(C)	>>> linalg.pinv2(C)
	Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

Inverse	Create a 2x2 identity matrix
>>> F = np.eye(3, k=1)	Create a 2x2 identity matrix
Norm	Compressed Sparse Row matrix
>>> G = np.mat(np.identity(2))	Compressed Sparse Column matrix
C	Dictionary Of Keys matrix
>>> H = sparse.csr_matrix(C)	Sparse matrix to full matrix
>>> I = sparse.csc_matrix(D)	Identify sparse matrix
E	
>>> J = sparse dok_matrix(A)	
T.e todense()	
>>> sparse.lisspmatrix_csc(A)	

Sparse Matrix Routines

Inverse	Inverse
>>> sparse.linalg.inv(I)	>>> linalg.inv(H)
Norm	Norm
>>> sparse.linalg.norm(J)	Solver for sparse matrices
Solving linear problems	>>> sparse.linalg.spsolve(H, I)
>>> sparse.linalg.sparsesolve(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	Addition
>>> np.add(A, D)	Subtraction
Subtraction	>>> np.subtract(A,D)
>>> np.subtract(A,D)	Division
>>> np.divide(A,D)	>>> np.divide(A,D)
Multiplication	Multiplication operator
>>> A @ D	(Python 3)
	Multiplication
>>> np.multiply(D,A)	Dot product
>>> np.dot(A,D)	Vector dot product
>>> np.vdot(A,D)	Inner product
>>> np.inner(A,D)	Outer product
>>> np.outer(A,D)	Tensor dot product
>>> np.kron(A,D)	Kronecker product
Exponential Functions	Matrix exponential
>>> linalg.expm(A)	Matrix exponential (Taylor Series)
>>> linalg.expm2(A)	Matrix exponential decomposition
>>> linalg.expm3(D)	
Logarithm Function	Matrix logarithm
>>> linalg.logm(A)	Matrix sine
Trigonometric Functions	Matrix cosine
>>> linalg.sinm(D)	Matrix tangent
>>> linalg.cosm(D)	
>>> linalg.tanm(A)	
Hyperbolic Trigonometric Functions	Hyperbolic matrix sine
>>> linalg.sinhm(D)	Hyperbolic matrix cosine
>>> linalg.coshm(D)	Hyperbolic matrix tangent
>>> linalg.tanhm(A)	
Matrix Sign Function	Matrix sign function
>>> np.sqrtm(A)	
Matrix Square Root	Matrix square root
>>> linalg.sqrtm(A)	
Arbitrary Functions	Evaluate matrix function
>>> linalg.funm(A, lambda x: x**x)	

Decompositions

Eigenvalues and Eigenvectors	Solve ordinary or generalized eigenvalue problem for square matrix
>>> la, v = linalg.eig(A)	Unpack eigenvalues
>>> la, 1, 2 = la	First eigenvector
>>> v[:, 0]	Second eigenvector
>>> v[:, 1]	Unpack eigenvalues
>>> linalg.eigvals(A)	
Singular Value Decomposition	Singular Value Decomposition (SVD)
>>> U, s, Vh = linalg.svd(B)	Construct sigma matrix in SVD
>>> M, N = B.shape	
>>> Sig = linalg.diagsvd(s, M, N)	
LU Decomposition	LU Decomposition
>>> P, L, U = linalg.lu(C)	

Sparse Matrix Decompositions

>>> la, v = sparse.linalg.eigs(F, 1)	Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)	SVD

DataCamp
Learn Python for Data Science Interactively



Python For Data Science Cheat Sheet

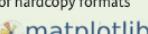
Matplotlib

Learn Python Interactively at www.DataCamp.com



Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.



1 Prepare The Data

Also see Lists & NumPy

1D Data

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

2D Data or Images

```
>>> data = np.random.random((10, 10))
>>> data2 = np.random.random((10, 10))
>>> Y, X = np.meshgrid(np.arange(-3.1:1.00), -3:1.00)
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.image import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

```
>>> import matplotlib.pyplot as plt
Figure
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax = 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

iD Data

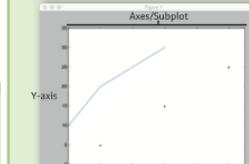
```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x, y)
>>> ax.set_xticks([1, 2, 3, 4, 5])
>>> ax.set_yticks([0.5, 1, 2, 3, 4, 5])
>>> ax.set_xlabel('x-axis')
>>> ax.set_ylabel('y-axis')
>>> ax.set_title('Example Graph', style='italic')
>>> ax.annotate("Sine", xy=(0, 1), xytext=(10, 0),
xycoords='data', textcoords='data',
arrowprops=dict(arrowstyle=">-", connectionstyle="arc3", ))
>>> ax.text(1, 2, "Example Graph", style='italic')
```

Colormapped or RGB arrays

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

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]
>>> y = [10,20,25,30]
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(x, y, color='lightblue', linewidth=3)
>>> ax.scatter([2,4,6], [15,20,25],
color='darkgreen',
marker='^')
>>> 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, x**2, x, x**3)
>>> ax.plot(x, y, alpha=0.4)
>>> ax.plot(x, y, color='red')
>>> 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,ls="-.")
>>> plt.plot(x,y,ls=":")
>>> plt.setp(lines,color='r', linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1, 2, "Example Graph", style='italic')
>>> ax.annotate("Sine", xy=(0, 1), xytext=(10, 0),
xycoords='data', textcoords='data',
arrowprops=dict(arrowstyle=">-", connectionstyle="arc3", ))
>>> ax.text(1, 2, "Example Graph", style='italic')
>>> ax.set_xticks([1, 2, 3, 4, 5])
>>> ax.set_yticks([0.5, 1, 2, 3, 4, 5])
>>> ax.set_xlabel('x-axis')
>>> ax.set_ylabel('y-axis')
>>> ax.set_title('Example Graph', style='italic')
>>> ax.annotate("Sine", xy=(0, 1), xytext=(10, 0),
xycoords='data', textcoords='data',
arrowprops=dict(arrowstyle=">-", connectionstyle="arc3", ))
>>> ax.text(1, 2, "Example Graph", style='italic')
```

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)
```

Data Distributions

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

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2].axlabel(CS)
```

Pseudocolor plot of 2D array

```
>>> axes2[0].pcolor(data2)
>>> axes2[1].pcolor(data2)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].
```

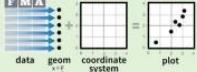
Data Visualization with ggplot2

Cheat Sheet

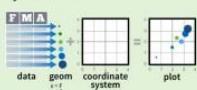


Basics

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 = cyl, 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 = cyl, y = hwy))

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

**ggplot(mpg, aes(hwy, cyl)) +
geom_point(aes(color = cyl)) +
geom_smooth(method = "lm") +
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

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

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

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

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
a + geom_density(kernel = "gaussian")
x, y, alpha, color, fill, linetype, size, weight
a + geom_dotplot()
x, y, alpha, color, fill

a + geom_freqpoly()
x, y, alpha, color, linetype, size
a + geom_histogram(binwidth = 5)
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

c <- ggplot(map, aes(long, lat))

c + geom_polygon(aes(group = group))
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))

**d + geom_path(lineend = "butt",
linejoin = "round", linemitre = 1)**
x, y, alpha, color, linetype, size
**d + geom_ribbon(aes(ymin = unemploy - 900,
ymax = unemploy + 900))**
x, ymax, ymin, alpha, color, fill, linetype, size

e <- ggplot(seals, aes(x = long, y = lat))

**e + geom_segment(aes(
xend = long + delta_long,
yend = lat + delta_lat))**
x, xend, y, yend, alpha, color, linetype, size
**e + geom_rect(aes(xmin = long, ymin = lat,
xmax = long + delta_long,
ymax = lat + delta_lat))**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

f <- ggplot(blank, aes(x = 1, y = 1))

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl")
alpha, color, linetype, size

f + geom_smooth(model = lm)
x, y, alpha, color, fill, linetype, size, weight

f + geom_text(aes(label = ct))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

f + geom_jitter()
x, y, alpha, color, fill, shape, size

f + geom_point()
x, y, alpha, color, fill, shape, size

f + geom_quantile()
x, y, alpha, color, linetype, size, weight

f + geom_rug(sides = "bl")
alpha, color, linetype, size

f + geom_smooth(model = lm)
x, y, alpha, color, fill, linetype, size, weight

C f + geom_text(aes(label = ct))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Continuous Function

j < ggplot(economics, aes(date, unemploy))

j + geom_area()
x, y, alpha, color, fill, linetype, size

j + geom_line()
x, y, alpha, color, linetype, size

j + geom_step(direction = "hv")
x, y, alpha, color, linetype, size

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))

k + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Maps

**data <- data.frame(murder = USArrests\$Murder,
state = tolower(rownames(USArrests)))**
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

**l + geom_map(aes(map_id = state), map = map) +
expand_limits(x = map\$long, y = map\$lat)**
map_id, alpha, color, fill, linetype, size

Three Variables

seals\$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))

m <- ggplot(seals, aes(long, lat))

m + geom_rect(aes(z = z))
x, y, z, alpha, colour, linetype, size, weight

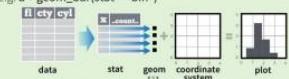
**m + geom_raster(aes(fill = z), hjust = 0.5,
vjust = 0.5, interpolate = FALSE)**
x, y, alpha, fill

m + geom_tile(aes(fill = z))
x, y, alpha, color, fill, linetype, size

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

Stats - An alternative way to build a layer

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



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

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")`



`i + stat_density2d(aes(fill = ..level..), geom = "polygon", n = 100)`

geom for layer | parameters for stat

`a + stat_bin(binwidth = 1, origin = 10)` 1D distributions

`x, y | .count..., .n..., density..., .ndensity...`

`a + stat_bindot(binwidth = 1, binaxis = "x")`

`x, y | .count..., .density...`

`a + stat_density(adjust = 1, kernel = "gaussian")`

`x, y | .count..., .density..., .scaled...`

`f + stat_bin2d(bins = 30, drop = TRUE)` 2D distributions

`x, y, fill | .count..., .density...`

`f + stat_binehd(bins = 30)`

`x, y, fill | .count..., .density...`

`f + stat_density2d(contour = TRUE, n = 100)`

`x, y, color, size | .level...`

`m + stat_contour(aes(z = z))` 3 Variables

`x, y, z, order | .level...`

`m + stat_spoke(aes(radius = z, angle = z))`

`angle, radius, x, xend, y, yend | .x..., .xend..., .y..., .yend...`

`m + stat_summary_hex(aes(z = z), bins = 30, fun = mean)`

`x, y, 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 | .scaled..., .count..., .n..., .violinwidth..., .width...`

`f + stat_ecdf(n = 40)` Functions

`x, y |`

`f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")`

`x, y | .qu 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 = 33), fun = dnorm, n = 101, args = list(sd = 0.5))` General Purpose

`x | .y...`

`f + stat_identity()`

`ggplot() + stat_qf(aes(sample = 1:100), distribution = qt, dparams = list(df = 5))`

`sample, x, y |`

`f + stat_sum()`

`x, y | .size...`

`f + stat_summary(fun.data = "mean_cl_boot")`

`x | .stat...`

`f + stat_unique()`

Scales

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

`n <- b + geom_bar()`

`r + coord_cartesian(xlim = c(0, 5))`

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.

`z + coord_map(projection = "ortho", orientation = c(-74, 0))`

`projection, orientation, xlim, ylim`

Map projections from the mapproj package

(mercator (default), azequalarea, lagrange, etc.)

General Purpose scales

Use with any aesthetic:

alpha, color, fill, linetype, shape, size

`scale_*_continuous()` - map cont values to visual values

`scale_*_discrete()` - map discrete values to visual values

`scale_*_identity()` - use data values as visual values

`scale_*_manual(values = c())` - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

`scale_x_date(labels = date_format("%m/%d/%d"), 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

`n <- b + geom_bar(aes(fill = f))`

`r + scale_fill_brewer(palette = "Blues")`

For palette choices: library(RColorBrewer); display.brewer.all()

`r + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")`

RColorBrewer::brewer.pal()

Shape scales

Manual shape values

0 □ 1 ○ 2 △ 3 ▲ 4 △ 5 ▽ 6 ▾ 7 ▷ 8 ▶ 9 ▷ 10 ▷ 11 ▷ 12 ▷ 13 ▷ 14 ▷ 15 ▷ 16 ● 17 △ 18 ▲ 19 ▲ 20 ▾ 21 ▾ 22 ▾ 23 ▾ 24 ▾ 25 ▾

0 □ 1 ○ 2 △ 3 ▲ 4 △ 5 ▽ 6 ▾ 7 ▷ 8 ▶ 9 ▷ 10 ▷ 11 ▷ 12 ▷ 13 ▷ 14 ▷ 15 ▷ 16 ● 17 △ 18 ▲ 19 ▲ 20 ▾ 21 ▾ 22 ▾ 23 ▾ 24 ▾ 25 ▾

0 □ 1 ○ 2 △ 3 ▲ 4 △ 5 ▽ 6 ▾ 7 ▷ 8 ▶ 9 ▷ 10 ▷ 11 ▷ 12 ▷ 13 ▷ 14 ▷ 15 ▷ 16 ● 17 △ 18 ▲ 19 ▲ 20 ▾ 21 ▾ 22 ▾ 23 ▾ 24 ▾ 25 ▾

0 □ 1 ○ 2 △ 3 ▲ 4 △ 5 ▽ 6 ▾ 7 ▷ 8 ▶ 9 ▷ 10 ▷ 11 ▷ 12 ▷ 13 ▷ 14 ▷ 15 ▷ 16 ● 17 △ 18 ▲ 19 ▲ 20 ▾ 21 ▾ 22 ▾ 23 ▾ 24 ▾ 25 ▾

0 □ 1 ○ 2 △ 3 ▲ 4 △ 5 ▽ 6 ▾ 7 ▷ 8 ▶ 9 ▷ 10 ▷ 11 ▷ 12 ▷ 13 ▷ 14 ▷ 15 ▷ 16 ● 17 △ 18 ▲ 19 ▲ 20 ▾ 21 ▾ 22 ▾ 23 ▾ 24 ▾ 25 ▾

Size scales

Manual shape values

`p <- f + geom_point(aes(shape = f))`

`p + scale_shape(solid = FALSE)`

`p + scale_shape_manual(values = c(3:7))`

Shape values shown in chart on right

`q <- f + geom_point(aes(size = q))`

`q + scale_size_area(max = 6)`

Value mapped to area of circle (not radius)

Position Adjustments

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

`s <- ggplot(mpg, aes(flf, 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_gray()`

Grey background (default theme)

`r + theme_minimal()`

Minimal theme

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 labeller to adjust facet labels

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

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

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

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

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

`c d e p r`

Use scale functions to update legend labels

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")`

All of the above

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.

Zooming

Without clipping (preferred)

`t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))`

With clipping (removes unseen data points)

`t + xlim(0, 100) + ylim(10, 20)`

`t + scale_x_continuous(limits = c(0, 100)) + scale_y_continuous(limits = c(0, 100))`

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

Python For Data Science Cheat Sheet

PySpark Basics

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



Spark

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



Initializing Spark

SparkContext

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

Inspect SparkContext

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

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
...     .setMaster("local")
...     .setAppName("MyApp")
...     .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([(1,2),(1,2),(1,2)])
>>> rdd2 = sc.parallelize([(1,2),(1,2),(1,2),(1,2)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([(1,2,3,4,5,6,7,8,9,10)])
```

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

```
>>> rdd.getNumPartitions()
>>> rdd.count()
3
>>> rdd.countByKey()
defaultdict(<type 'int'>, {'a':2,'b':1})
>>> rdd.collectAsMap()
{('a', 2), ('b', 1)}
>>> rdd.sum()
4950
>>> sc.parallelize([]).isEmpty()
True
```

List the number of partitions
Count RDD instances
Count RDD instances by key
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

Summary

```
>>> rdd3.max()
99
>>> rdd3.min()
0
>>> rdd3.mean()
49.5
>>> rdd3.stdev()
28.866070047722118
>>> rdd3.variance()
833.25
>>> rdd3.histogram(3)
[(0,33,66,99), [33,33,34]]
>>> rdd3.stats()
Summary statistics (count, mean, stdev, max & min)
```

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins
Summary statistics (count, mean, stdev, max & min)

Applying Functions

```
>>> rdd.map(lambda x: x*(x[1],x[0]))
[('a',7,7), ('a', 2, 2), ('a', 2, 2, 'b', 'b')]
>>> rdd2 = rdd.flatMap(lambda x: x*(x[1],x[0]))
>>> rdd3.collect()
[('a',7,7, 'a'), ('a', 2, 2, 'a'), ('b', 2, 2, 'b', 'b')]
>>> rdd4 = rdd3.values(lambda x: x)
>>> rdd4.collect()
[('a', 'a'), ('a', 'b'), ('b', 'a'), ('b', 'b')]
```

Apply a function to each RDD element
Apply a function to each RDD element and flatten the result
Apply a flatMap function to each (key,value) pair of RDD without changing the keys

Selecting Data

Getting	Return a list with all RDD elements
>>> rdd.collect()	Take first 2 RDD elements
>>> rdd.take(2)	Take first RDD element
>>> rdd.top(1)	Take top 2 RDD elements
>>> rdd.top(2)	Return sampled subset of rdd3
Sampling	Filter the RDD
>>> rdd3.sample(False, 0.15, 81).collect()	Return distinct RDD values
>>> rdd.filter(lambda x: "a" in x).collect()	Return (key,value) RDD's keys

Iterating

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

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)
.collect()
[('a',9), ('b',2)]
```

Merge the RDD values for each key

Grouping

```
>>> rdd3.groupBy(lambda x: x % 2)
.collect()
>>> rdd3.mapValues(lambda x: x*2)
.collect()
[('a',9), ('b',2)]
```

Return RDD of grouped values

Aggregating

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

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

Aggregating

```
>>> rdd3.fold(0,add)
```

Aggregate the elements of each partition, and then the results

```
.collect()
[('a',9), ('b',2)]
```

Merge the values for each key

```
>>> rdd3.reduceBy(lambda x: x+x)
.collect()
```

Create tuples of RDD elements by applying a function

Mathematical Operations

Subtract

```
>>> rdd.subtract(rdd2)
.collect()
[('b',2), ('a',7)]
>>> rdd2.subtractByKey(rdd)
.collect()
[('d', 1)]
```

Return each rdd value not contained in rdd2

```
>>> rdd2.subtractByKey(rdd)
.collect()
[('a', 1)]
```

Return each (key,value) pair of rdd2 with no matching key in rdd

```
>>> rdd.cartesian(rdd2).collect()
```

Return the Cartesian product of rdd and rdd2

Sort

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

Sort RDD by given function

Sort (key, value) RDD by key

Repartitioning

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

New RDD with 4 partitions

Decrease the number of partitions in the RDD to 1

Saving

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

Stopping SparkContext

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

Execution

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

DataCamp

Learn Python for Data Science interactively



LEGEND

TIME Complexity vs. SPACE Complexity

Good Fair Bad

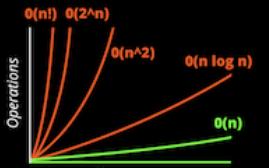
Good Fair Bad

<BIG-O-CHEATSHEET>

DATA STRUCTURE Operations

www.bigcheatsheet.com

DATA Structure	TIME Complexity				SPACE Complexity			
	Average	Worst	Average	Worst	Average	Worst	Average	Worst
Array								
Stack								
Queue								
Singly-Linked List								
Doubly-Linked List								
Skip List								
Hash Table	N/A							
Binary Search Tree								
Cartesian Tree	N/A							
B-Tree								
Red-Black Tree								
Splay Tree	N/A							
AVL Tree								
KD Tree								



ARRAY SORTING Algorithms

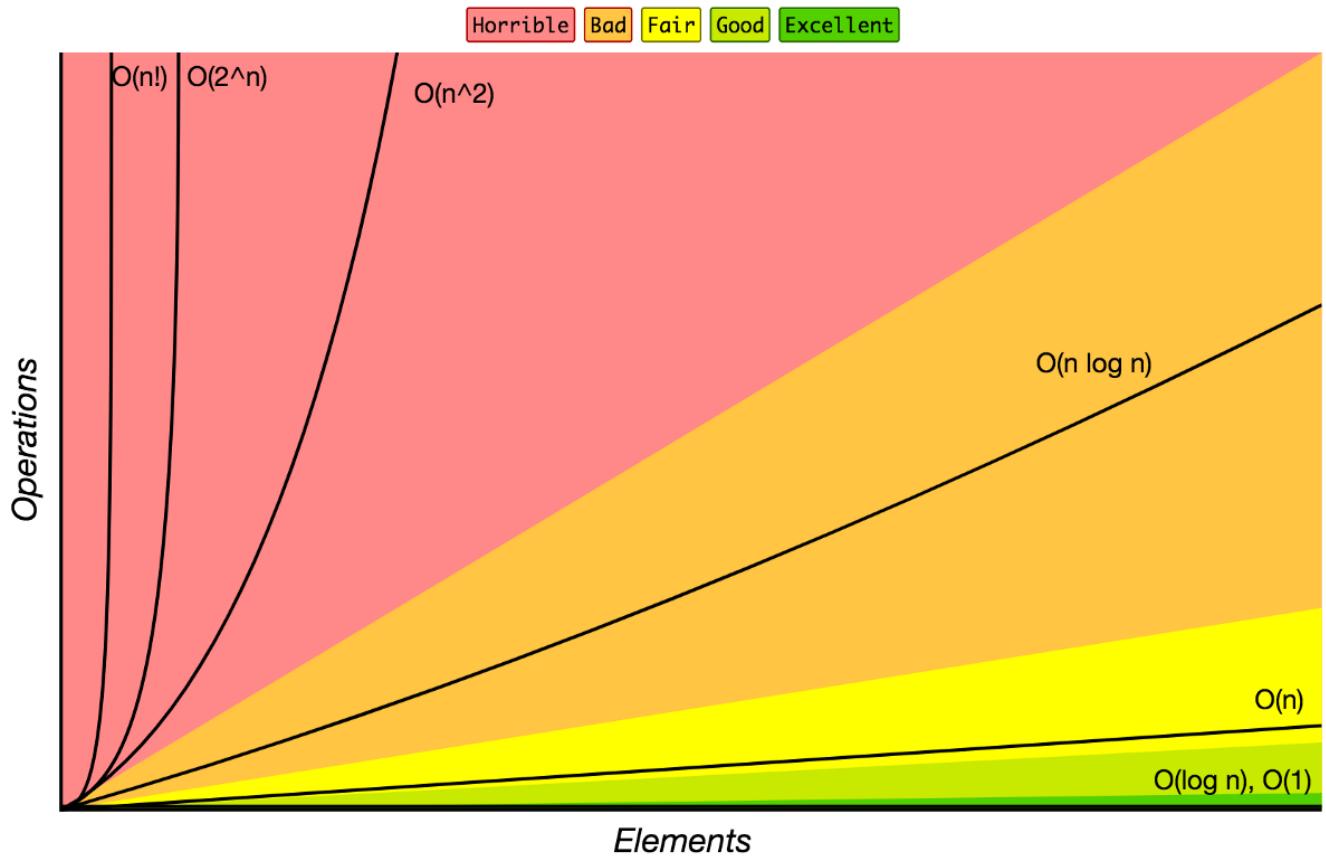
TIME Complexity

QuickSort				
Mergesort				
Timsort				
Heapsort				
Bubble Sort				
Insertion Sort				
Selection Sort				
Tree Sort				
Shell Sort				
Bucket Sort				
Radix Sort				
Counting Sort				
Cubesort				

SPACE Complexity

QuickSort				
Mergesort				
Timsort				
Heapsort				
Bubble Sort				
Insertion Sort				
Selection Sort				
Tree Sort				
Shell Sort				
Bucket Sort				
Radix Sort				
Counting Sort				
Cubesort				

Big-O Complexity Chart



Common Data Structure Operations

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

Array Sorting Algorithms

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