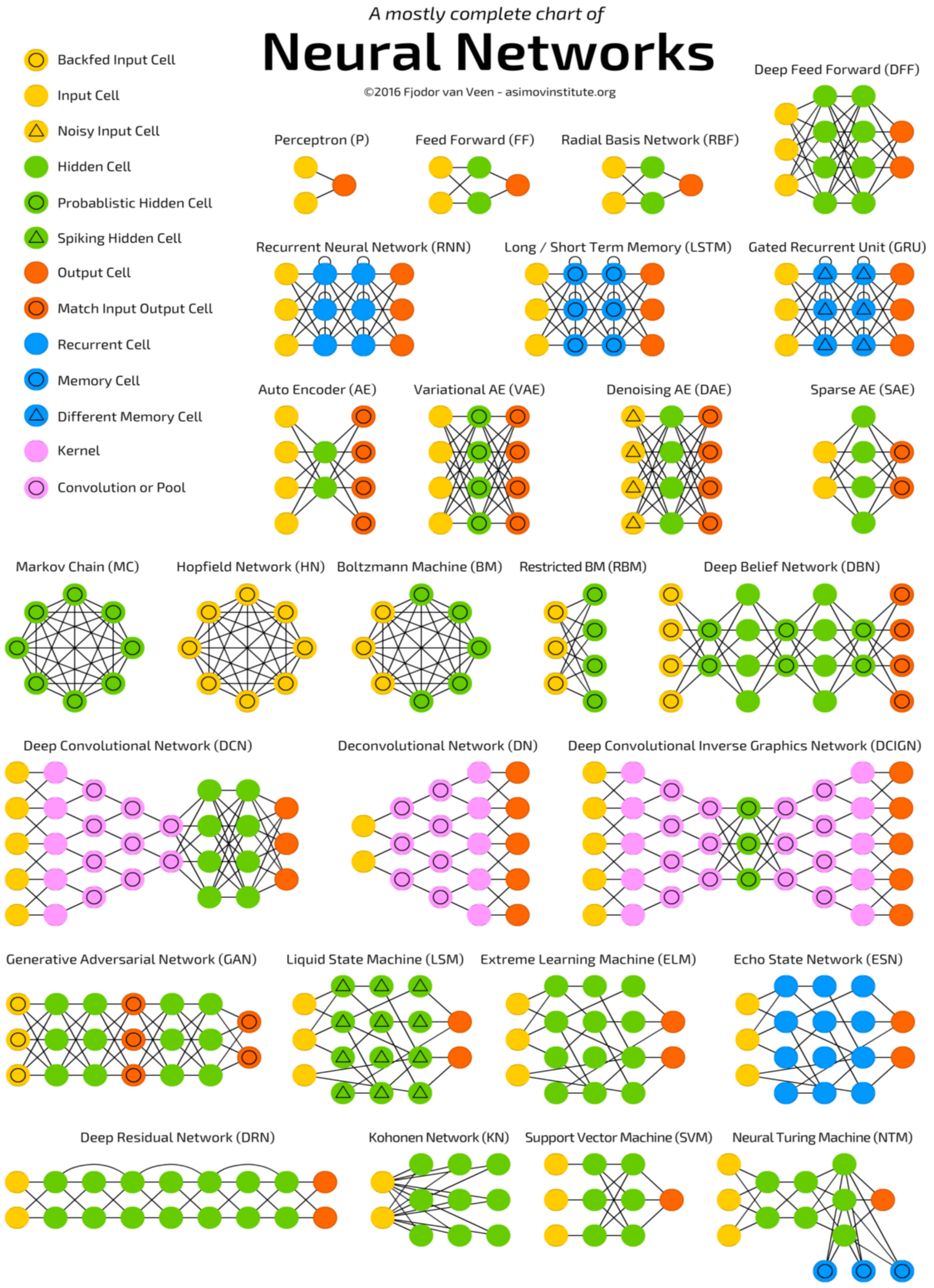
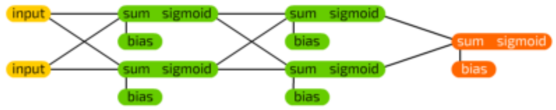
# Cheat Sheets for AI, Neural Networks, Machine Learning, Deep Learning & Big Data



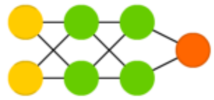
*An informative chart to build*

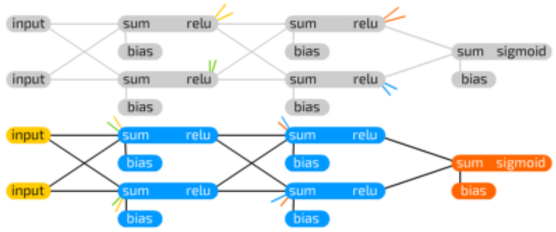
# Neural Network Graphs





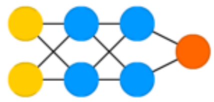
Deep Feed Forward Example

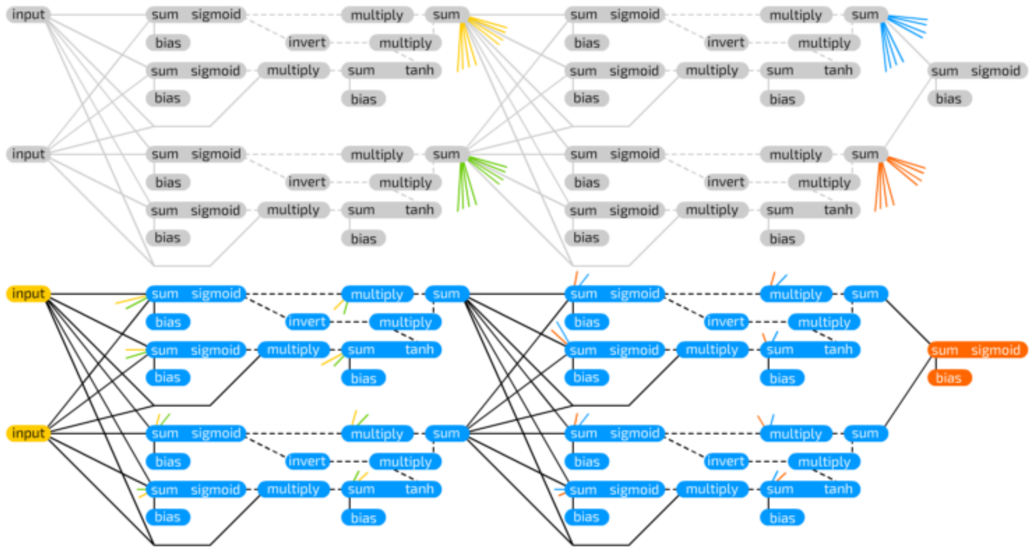




Deep Recurrent Example (previous iteration)

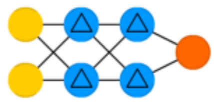
Deep Recurrent Example

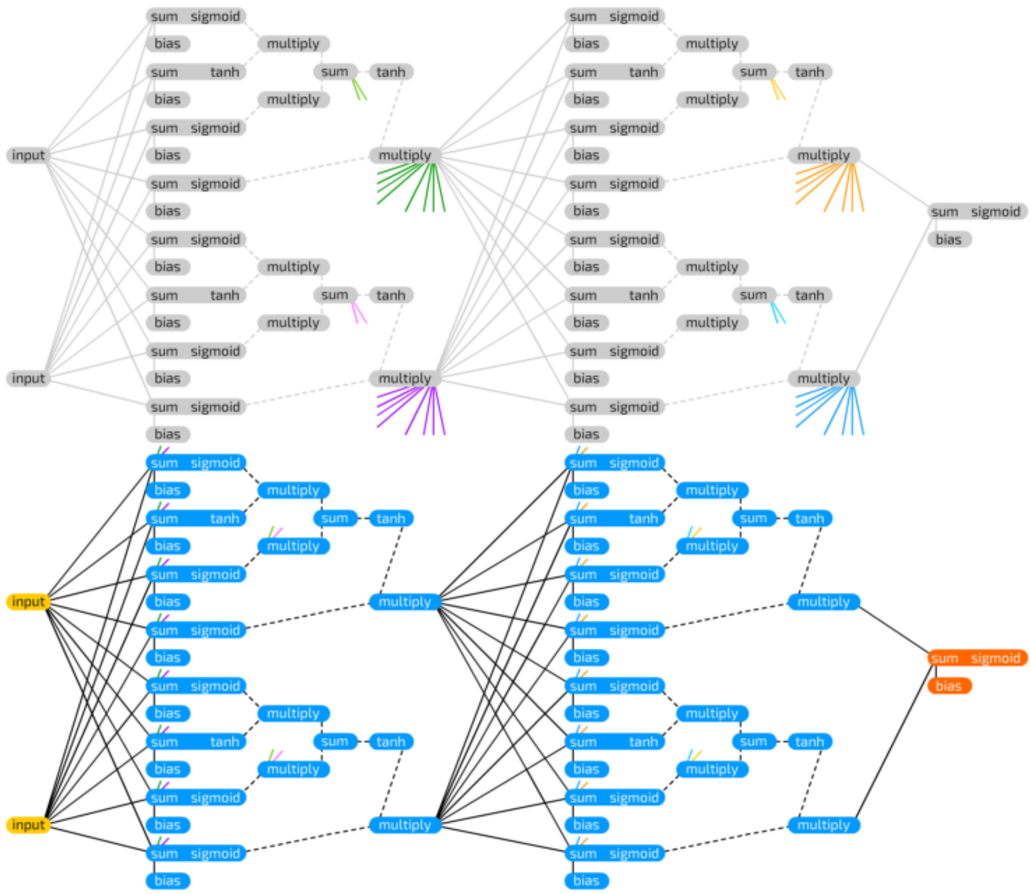




Deep GRU Example (previous iteration)

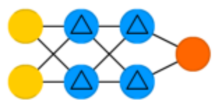
Deep GRU Example





Deep LSTM Example (previous iteration)

Deep LSTM Example



**Linear Vector Spaces:**

**Definition**: A linear vector space, $\mathcal{X}$ is a set of elements (vectors) defined over a scalar field, $F$, that satisfies the following conditions:

**1)** if $\chi \in X$ and $y \in X$ then $\chi + y \in X$. **2)** $\chi + y = y + \chi$ **3)** $(\chi + y) + z = \chi + (y + z)$

**4)** There is a unique vector $0 \in X$, such that $\chi + 0 = \chi$ for all $\chi \in X$.

**5)** For each vector $\chi \in X$ there is a unique vector in X, to be called $(-\chi)$, such that $\chi + (-\chi) = 0$. **6)** multiplication, for all scalars $a \in F$, and all vectors $\chi \in X$,

**7)** For any $\chi \in X$, $1\chi = \chi$ (for scalar 1).

**8)** For any two scalars $a \in F$ and $b \in F$ and any $\chi \in X$, $a(b\chi) = (ab)\chi$.

**9)** $(a + b)\chi = a\chi + b\chi$. **10)** $a(\chi + y) = a\chi + ay$.

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

**Spanning a Space:**

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

**Inner Product:** $(x, y)$ for any scalar function of x and y.

1. $(x,y) = (y,x)$ 2. $(x, ay_1 + by_2) = a(x,y_1) + b(x,y_2)$
3. $(x,x) \geq 0$, 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 $\theta$ bet. 2 vectors $x$ and $y$ is defined by $\cos\theta = \frac{(x,y)}{\|x\|\,\|y\|}$

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

**Gram Schmidt Orthogonalization:**

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

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

$$where\ \frac{(v_i, y_k)}{(v_i, v_i)} v_i\ is\ the\ projection\ of\ y_k\ on\ v_i$$

**Vector Expansions:**

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

$$for\ orthogonal\ vectors, x_j = \frac{(v_j, x)}{(v_j, v_j)}$$

**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 $\mathbf{B} = [\mathrm{v}_1\ \mathrm{v}_2\ \ldots\ \mathrm{v}_n]$,

$\mathbf{R} = [\mathrm{r}_1\ \mathrm{r}_2 \ldots \mathrm{r}_n]$, $\mathbf{R}^T = \mathbf{B}^{-1}$ In matrix form: $\mathbf{x}^v = \mathbf{B}^{-1}\,\mathbf{x}^s$

**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 $\{v_1, v_2, \ldots, v_n\}$ be a basis for vector space $X$, and let $\{u_1, u_2, \ldots, u_n\}$ 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:** $\mathbf{B}_t = [\mathbf{t}_1\ \mathbf{t}_2\ \ldots \mathbf{t}_n]$, $\mathbf{B}_w = [\mathbf{w}_1\ \mathbf{w}_2\ \ldots \mathbf{w}_n]$

$$\mathbf{A}' = [\mathbf{B}_w^{-1}\mathbf{A}\mathbf{B}_t]$$

**Eigenvalues & Eigenvectors:** $\mathbf{A}\mathbf{z} = \lambda\mathbf{z}$, $|[\mathbf{A} - \lambda\mathbf{I}]| = 0$

**Diagonalization:** $\mathbf{B} = [\mathbf{z}_1\ \mathbf{z}_2\ \ldots \mathbf{z}_n]$,

where $\{\boldsymbol{z}, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n\}$ are the eigenvectors of a square matrix A,

$$[\mathbf{B}^{-1}\mathbf{A}\mathbf{B}] = \mathrm{diag}([\lambda_1\ \lambda_2 \ldots \lambda_n])$$

**Perceptron Architecture:**

$\mathbf{a} = hardlim(\mathbf{Wp} + \mathbf{b})$, $\mathbf{W} = [\,_1\mathbf{w}^T\ _2\mathbf{w}^T\ \ldots\ _S\mathbf{w}^T]^T$,

$$a_i = hardlim(n_i) = hardlim(\,_i\mathbf{w}^T\,\mathrm{p} + b_i)$$

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

The decision boundary is always orthogonal to the weight vector.

Single-layer perceptrons can only classify linearly separable vectors.

**Perceptron Learning Rule**

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T, \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e},$$

$$where\ \ \mathbf{e} = \mathbf{t} - \mathbf{a}$$

**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:** $\mathbf{a} = purelin(\mathbf{Wp})$

**The Hebb Rule:** *Supervised Form:* $w_{ij}^{new} = w_{ij}^{old} + t_{qi}P_{qi}$

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

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

**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 $\mathbf{P}^+ = (\mathbf{P}^T\mathrm{P})^{-1}\mathbf{P}^T$

**Variations of Hebbian Learning:**

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

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

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

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

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

**Hessian:** $\nabla^2 F(x) =$

$$\begin{bmatrix} \frac{\partial}{\partial x_1^2}F(\mathbf{x}) & \frac{\partial}{\partial x_1\,\partial x_2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_1\,\partial x_n}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2\,\partial x_1}F(\mathbf{x}) & \frac{\partial}{\partial x_2^2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_2\,\partial x_n}F(\mathbf{x}) \\ \vdots & \vdots & \vdots \\ \frac{\partial}{\partial x_n\,\partial x_1}F(\mathbf{x}) & \frac{\partial}{\partial x_n\,\partial x_2}F(\mathbf{x}) \ldots & \frac{\partial}{\partial x_n^2}F(\mathbf{x}) \end{bmatrix}$$

**Directional Derivatives:**

**1st Dir.Der.:** $\frac{\mathbf{p}^T\,\nabla F(\mathbf{x})}{\|\mathbf{p}\|}$, **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 $\Delta x$ 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 $\Delta x$ 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.:** $F(x) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\,\mathbf{x} + \mathbf{d}^T\mathbf{x} + \mathrm{c}$

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

**General Minimization Algorithm:**
$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ *or* $\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$

**Steepest Descent Algorithm:**
$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$ where, $\mathbf{g}_k = \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}$

**Stable Learning Rate:** ($\alpha_k = \alpha$, constant) $\alpha < \frac{2}{\lambda_{max}}$

$\{\lambda_1\ \lambda_2, \ldots, \lambda_n\}$ Eigenvalues of Hessian matrix A

**Learning Rate to Minimize Along the Line:**
$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \overset{is}{\Rightarrow} \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ (For quadratic fn.)

**After Minimization Along the Line:**
$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$

**ADALINE:** $\mathbf{a} = purelin(\mathbf{Wp} + \mathbf{b})$

**Mean Square Error:** *(for ADALINE it is a quadratic fn.)*
$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t - \mathbf{x}^T\mathbf{z})^2]$
$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$,
$c = E[t^2]$, $\mathbf{h} = E[t\mathbf{z}]$ *and* $\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] \Rightarrow \mathbf{A} = 2\mathbf{R}, \mathbf{d} = -2\mathbf{h}$
Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$,
where $\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$

**LMS Algorithm:** $\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\, \mathbf{e}(k)\, \mathbf{p}^T(k)$
$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\, \mathbf{e}(k)$

**Convergence Point:** $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$

**Stable Learning Rate:** $0 < \alpha < 1/\lambda_{max}$ where $\lambda_{max}$ is the maximum eigenvalue of $\mathbf{R}$

**Adaptive Filter ADALINE:**
$$a(k) = purelin(\mathbf{Wp}(k) + b) = \sum_{i=1}^{R} w_{1,i}\, y(k-i+1) + b$$

**Backpropagation Algorithm:**

**Performance Index:**
Mean Square error: $F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$

**Approximate Performance Index:** (single sample)
$\hat{F}(x) = \mathbf{e}^T(k)\mathbf{e}(k) = \left(\mathbf{t}(k) - \mathbf{a}(k)\right)^T(\mathbf{t}(k) - \mathbf{a}(k))$

**Sensitivity:** $\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left[\frac{\partial \hat{F}}{\partial \mathbf{n}_1^m}\ \frac{\partial \hat{F}}{\partial \mathbf{n}_2^m}\ \cdots\ \frac{\partial \hat{F}}{\partial \mathbf{n}_{S^m}^m}\right]^T$

**Forward Propagation:** $\mathbf{a}^0 = \mathbf{p}$,
$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1})$ *for* $m = 0, 1, \ldots, M-1$
$$\mathbf{a} = \mathbf{a}^M$$

**Backward Propagation:** $\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$,
$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}$ *for* $m = M-1, \ldots, 2, 1$, where
$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \text{diag}([\dot{f}^m(n_1^m)\ \ \dot{f}^m(n_2^m)\ \ldots\ \dot{f}^m(n_{S^m}^m)])$
$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

**Weight Update (Approximate Steepest Descent):**
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T$$
$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

***Heuristic Variations of Backpropagation:**

**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.)

**Backpropagation with Momentum (MOBP):**
$$\Delta\mathbf{W}^m(k) = \gamma\Delta\mathbf{W}^m(k-1) - (1-\gamma)\alpha\, \mathbf{s}^m(\mathbf{a}^{m-1})^T$$
$$\Delta\mathbf{b}^m(k) = \gamma\Delta\mathbf{b}^m(k-1) - (1-\gamma)\alpha\, \mathbf{s}^m$$

**Variable Learning Rate Backpropagation (VLBP)**
1. If the squared error (over the entire training set) increases by more than some set percentage $\zeta$ (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 $\gamma$ (if it is used) is set to zero.
2. If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\eta > 1$. If $\gamma$ has been previously set to zero, it is reset to its original value.
3. If the squared error increases by less than $\zeta$, then the weight update is accepted but the learning rate and the momentum coefficient are unchanged.

**Association:** $\mathbf{a} = hardlim(\mathbf{W}^0\mathbf{P}^0 + \mathbf{Wp} + b)$
An association is a link between the inputs and outputs of a network so that when a stimulus A is presented to the network, it will output a response B.

**Associative Learning Rules:**

**Unsupervised Hebb Rule:**
$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha\, \mathbf{a}(q)\mathbf{p}^T(q)$$

**Hebb with Decay:**
$$\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha\, \mathbf{a}(q)\mathbf{p}^T(q)$$

**Instar:** $\mathbf{a} = hardlim(\mathbf{Wp} + b)$, $\mathbf{a} = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$
The instar is activated for ${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|cos\theta \geq -b$
where $\theta$ is the angle between $\mathbf{p}$ and ${}_1\mathbf{w}$.

**Instar Rule:**
${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\, a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$
${}_i\mathbf{w}(q) = (1-\alpha)\ {}_i\mathbf{w}(q-1) + \alpha\, \mathbf{p}(q)$, $if\ (a_i(q) = 1)$

**Kohonen Rule:**
${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\, (\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$ *for* $i \in X(q)$

**Outstar Rule:** $\mathbf{a} = satlins(\mathbf{Wp})$
$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha\, \left(\mathbf{a}(q) - \mathbf{w}_j(q-1)\right)p_j(q)$

**Competitive Layer:** $\mathbf{a} = compet(\mathbf{Wp}) = compet(\mathbf{n})$

**Competitive Learning with the Kohonen Rule:**
$${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha\, \left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)\right)$$
$$= (1-\alpha)\, {}_{i^*}\mathbf{w}(q-1) + \alpha\, \mathbf{p}(q)$$
${}_{i}\mathbf{w}(q) = {}_{i}\mathbf{w}(q-1)$, $i \neq i^*$ where $i^*$ is the winning neuron.

**Self-Organizing with the Kohonen Rule:**
$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\, \left(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)\right)$$
$= (1-\alpha)\ {}_i\mathbf{w}(q-1) + \alpha\, \mathbf{p}(q)$, $i \in N_{i^*}(d)$
$$N_i(d) = \{j, d_{i,j} \leq d\}$$

**LVQ Network:** $(w_{k,i}^2 = 1) \Rightarrow$ subclass $i$ is a part of class $k$
$n_i^1 = -\|{}_i\mathbf{w}^1 - \mathbf{p}\|$, $\mathbf{a}^1 = compet(\mathbf{n}^1)$, $\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$

**LVQ Network Learning with the Kohonen Rule:**
${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha\, \left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)\right)$,
$$if\ a_{k^*}^2 = t_{k^*} = 1$$
${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha\, \left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)\right)$,
$$if\ a_{k^*}^2 = 1 \neq t_{k^*} = 0$$

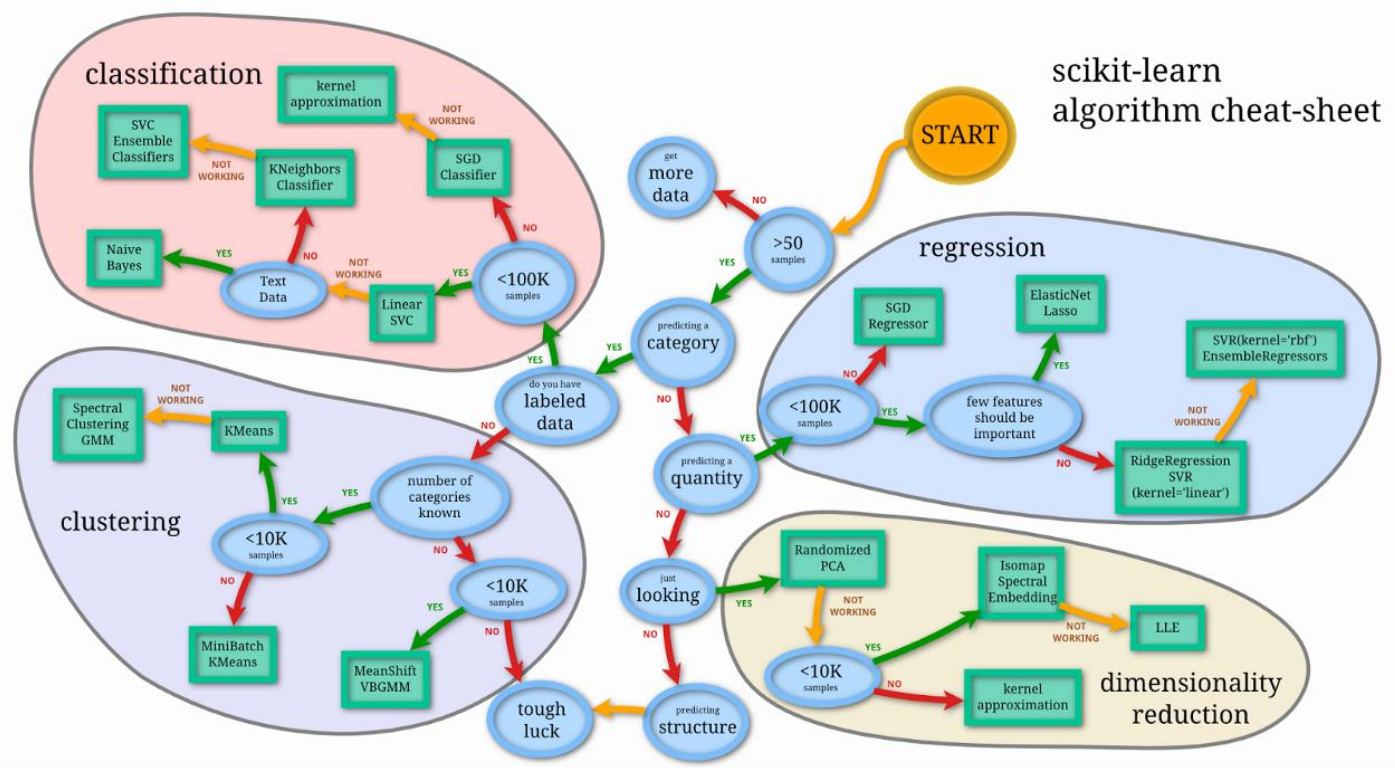
***hardlim***: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$, ***hardlims***: $a = \begin{cases} -1 & n < 0 \\ +1 & n \geq 0 \end{cases}$, ***purelin***: $a = n$, ***Logsig***: $a = \frac{1}{1+e^{-n}}$, ***tansig***: $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$, ***poslin***: $a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$

***compet***: $a = \begin{cases} 1 & \text{neuron with max } n \\ 0 & \text{all other neurons} \end{cases}$, ***satlin***: $a = \begin{cases} 0 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$, ***satlins***: $a = \begin{cases} -1 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$

***Delay***: $a(t) = u(t-1)$, ***Integrator***: $a(t) = \int_0^t u(\tau)d\tau + a(0)$

****HINT:**
$diag([1\ 2\ 3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$


MACHINE LEARNING IN EMOJI
SUPERVISED
UNSUPERVISED
REINFORCEMENT
SUPERVISED human builds model based on input / output
UNSUPERVISED human input, machine output human utilizes if satisfactory
REINFORCEMENT human input, machine output 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()
Similar datum into groups
based on centroids
ANOMALY DETECTION covariance.EllipticalEnvelope()
Finding outliers
through grouping
FEATURE REDUCTION
T-DISTRIB STOCHASTIC NEIB EMBEDDING manifold.TSNE()
Visualize high dimensional data. Convert
similarity to joint probabilities
PRINCIPLE 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.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

## Training And Test Data

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

## Create Your Model

### Supervised Learning Estimators

**Linear Regression**

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

**Support Vector Machines (SVM)**

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

**Naive Bayes**

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

**KNN**

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### Unsupervised Learning Estimators

**Principal Component Analysis (PCA)**

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

**K Means**

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

## Model Fitting

**Supervised learning**

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

Fit the model to the data

**Unsupervised Learning**

```
>>> k_means.fit(X_train)
>>> pca_model = pca.fit_transform(X_train)
```

Fit the model to the data
Fit to data, then transform it

## Prediction

**Supervised Estimators**

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

Predict labels
Predict labels
Estimate probability of a label

**Unsupervised Estimators**

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

Predict labels in clustering algos

## Preprocessing The Data

### Standardization

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

### Normalization

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

### Binarization

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

### Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

### Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

### Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

## Evaluate Your Model's Performance

### Classification Metrics

**Accuracy Score**

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

Estimator score method
Metric scoring functions

**Classification Report**

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

Precision, recall, f1-score and support

**Confusion Matrix**

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

### Regression Metrics

**Mean Absolute Error**

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

**Mean Squared Error**

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

**$R^2$ Score**

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

### Clustering Metrics

**Adjusted Rand Index**

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

**Homogeneity**

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

**V-measure**

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

### Cross-Validation

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

## Tune Your Model

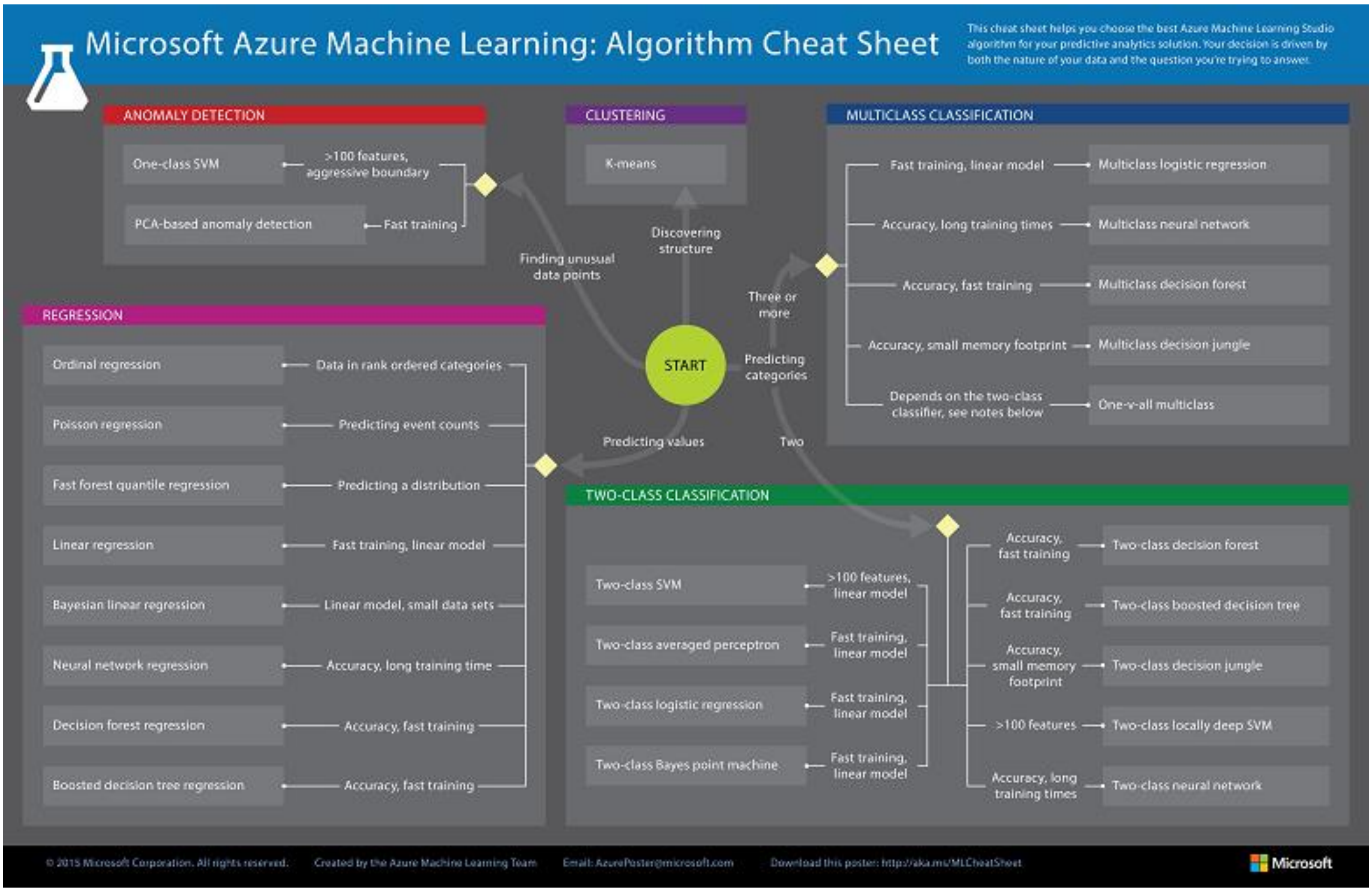
### Grid Search

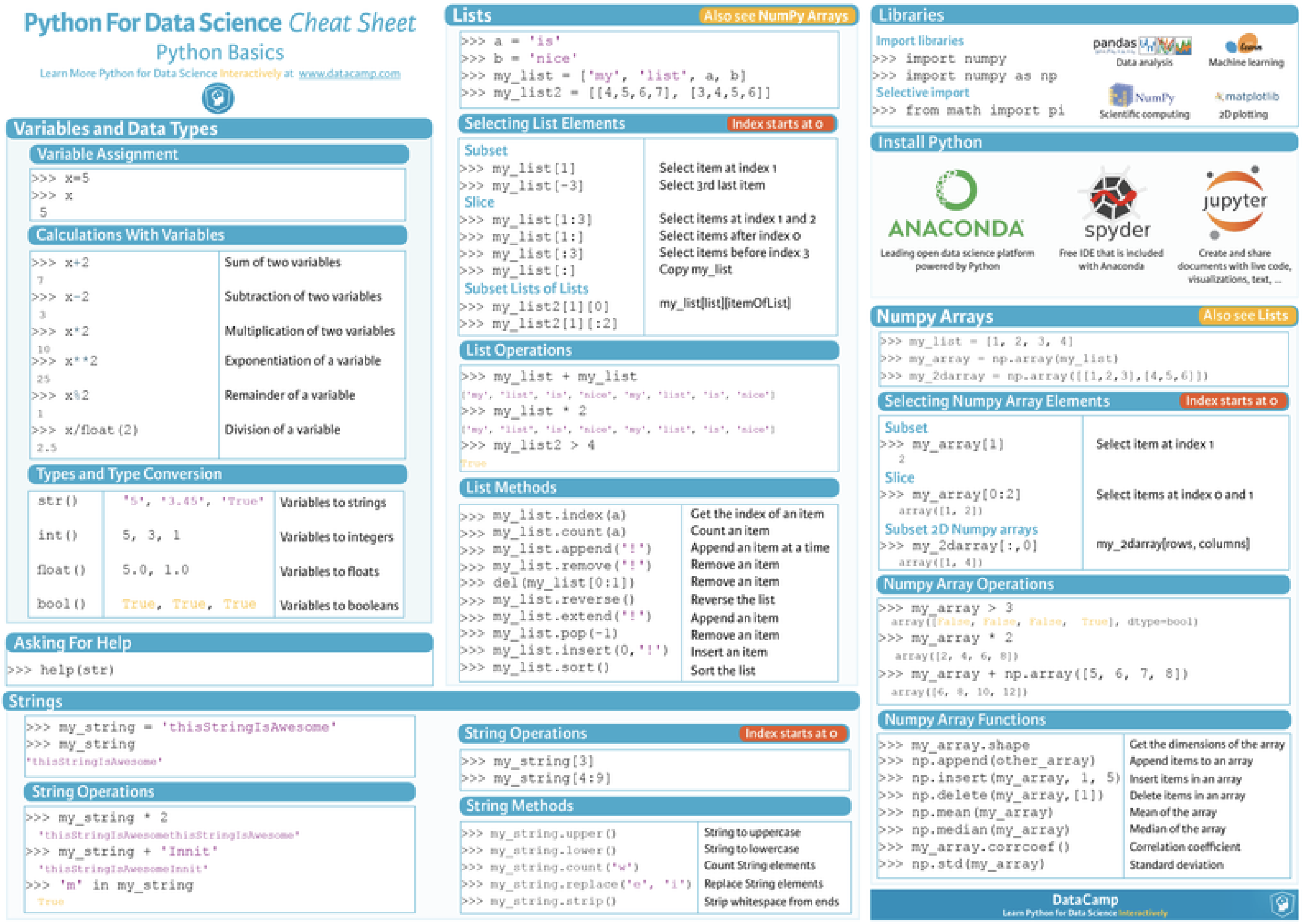
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)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

### Randomized Parameter Optimization

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

DataCamp
Learn Python for Data Science **Interactively**


Microsoft Azure Machine Learning: Algorithm Cheat Sheet
This cheat sheet helps you choose the best Azure Machine Learning Studio algorithm for your predictive analytics solution. Your decision is driven by both the nature of your data and the question you're trying to answer.
ANOMALY DETECTION
One-class SVM
>100 features, aggressive boundary
PCA-based anomaly detection
Fast training
CLUSTERING
K-means
Discovering structure
Finding unusual data points
MULTICLASS CLASSIFICATION
Fast training, linear model
Multiclass logistic regression
Accuracy, long training times
Multiclass neural network
Accuracy, fast training
Multiclass decision forest
Accuracy, small memory footprint
Multiclass decision jungle
Depends on the two-class classifier, see notes below
One-v-all multiclass
Three or more
START
Predicting categories
REGRESSION
Ordinal regression
Data in rank ordered categories
Poisson regression
Predicting event counts
Fast forest quantile regression
Predicting a distribution
Linear regression
Fast training, linear model
Bayesian linear regression
Linear model, small data sets
Neural network regression
Accuracy, long training time
Decision forest regression
Accuracy, fast training
Boosted decision tree regression
Accuracy, fast training
Predicting values
Two
TWO-CLASS CLASSIFICATION
Two-class SVM
>100 features, linear model
Two-class averaged perceptron
Fast training, linear model
Two-class logistic regression
Fast training, linear model
Two-class Bayes point machine
Fast training, linear model
Accuracy, fast training
Two-class decision forest
Accuracy, fast training
Two-class boosted decision tree
Accuracy, small memory footprint
Two-class decision jungle
>100 features
Two-class locally deep SVM
Accuracy, long training times
Two-class neural network
© 2015 Microsoft Corporation. All rights reserved.
Created by the Azure Machine Learning Team
Email: AzurePoster@microsoft.com
Download this poster: http://aka.ms/MLCheatSheet
Microsoft


Python For Data Science Cheat Sheet
Python Basics
Learn More Python for Data Science Interactively at www.datacamp.com
Variables and Data Types
Variable Assignment
>>> x=5
>>> x
5
Calculations With Variables
>>> x+2
7
Sum of two variables
>>> x-2
3
Subtraction of two variables
>>> x*2
10
Multiplication of two variables
>>> x**2
25
Exponentiation of a variable
>>> x%2
1
Remainder of a variable
>>> x/float(2)
2.5
Division of a variable
Types and Type Conversion
str() '5', '3.45', 'True' Variables to strings
int() 5, 3, 1 Variables to integers
float() 5.0, 1.0 Variables to floats
bool() True, True, True Variables to booleans
Asking For Help
>>> help(str)
Strings
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
String Operations
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
Lists
Also see NumPy Arrays
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
Selecting List Elements
Index starts at 0
Subset
>>> my_list[1]
Select item at index 1
>>> my_list[-3]
Select 3rd last item
Slice
>>> my_list[1:3]
Select items at index 1 and 2
>>> my_list[1:]
Select items after index 0
>>> my_list[:3]
Select items before index 3
>>> my_list[:]
Copy my_list
Subset Lists of Lists
>>> my_list2[1][0]
my_list[list][itemOfList]
>>> my_list2[1][:2]
List Operations
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
List Methods
>>> my_list.index(a)
Get the index of an item
>>> my_list.count(a)
Count an item
>>> my_list.append('!')
Append an item at a time
>>> my_list.remove('!')
Remove an item
>>> del(my_list[0:1])
Remove an item
>>> my_list.reverse()
Reverse the list
>>> my_list.extend('!')
Append an item
>>> my_list.pop(-1)
Remove an item
>>> my_list.insert(0,'!')
Insert an item
>>> my_list.sort()
Sort the list
String Operations
Index starts at 0
>>> my_string[3]
>>> my_string[4:9]
String Methods
>>> my_string.upper()
String to uppercase
>>> my_string.lower()
String to lowercase
>>> my_string.count('w')
Count String elements
>>> my_string.replace('e', 'i')
Replace String elements
>>> my_string.strip()
Strip whitespace from ends
Libraries
Import libraries
>>> import numpy
>>> import numpy as np
Selective import
>>> from math import pi
pandas
Data analysis
Machine learning
NumPy
Scientific computing
matplotlib
2D plotting
Install Python
ANACONDA
Leading open data science platform powered by Python
spyder
Free IDE that is included with Anaconda
jupyter
Create and share documents with live code, visualizations, text, ...
Numpy Arrays
Also see Lists
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
Selecting Numpy Array Elements
Index starts at 0
Subset
>>> my_array[1]
2
Select item at index 1
Slice
>>> my_array[0:2]
array([1, 2])
Select items at index 0 and 1
Subset 2D Numpy arrays
>>> my_2darray[:,0]
array([1, 4])
my_2darray[rows, columns]
Numpy Array Operations
>>> my_array > 3
array([False, False, False, True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
Numpy Array Functions
>>> my_array.shape
Get the dimensions of the array
>>> np.append(other_array)
Append items to an array
>>> np.insert(my_array, 1, 5)
Insert items in an array
>>> np.delete(my_array,[1])
Delete items in an array
>>> np.mean(my_array)
Mean of the array
>>> np.median(my_array)
Median of the array
>>> my_array.corrcoef()
Correlation coefficient
>>> np.std(my_array)
Standard deviation
DataCamp
Learn Python for Data Science Interactively

# 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",      Step 2
               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'])
```

```
>>> from bokeh.models import ColumnDataSource
>>> cds_df = ColumnDataSource(df)
```

## 2 Plotting

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

## 3 Renderers & Visual Customizations

### Glyphs

#### Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
              fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
              color='blue', size=1)
```

#### Line Glyphs

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

### Customized Glyphs

Also see Data

#### Selection and Non-Selection Glyphs

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

#### Hover Glyphs

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

#### Colormapping

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

### Rows & Columns Layout

#### Rows

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

#### Columns

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

### Linked Plots

Also see Data

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

### Legends

#### Legend Location

Inside Plot Area

```
>>> p.legend.location = 'bottom_left'
```

Outside Plot Area

```
>>> r1 = p2.asterisk(np.array([1,2,3]), np.array([3,2,1])
>>> r2 = p2.line([1,2,3,4], [3,4,5,6])
>>> legend = Legend(items=[("One" , [p1, r1]),("Two" , [r2])], location=(0, -30))
>>> p.add_layout(legend, 'right')
```

#### 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"
```

## 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)
>>> show(layout)
```

```
>>> save(p1)
>>> save(layout)
```

## Statistical Charts With Bokeh

Also see Data

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

### Bar Chart

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

### Box Plot

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

### Histogram

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

### Scatter Plot

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

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

## Data

Also see NumPy, Pandas & Scikit-Learn

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

### Keras Data Sets

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

### Other

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

## Model Architecture

### Sequential Model

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

### Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

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

Regression

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

### Convolutional Neural Network (CNN)

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

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

### Recurrent Neural Network (RNN)

```
>>> from keras.klayers 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'))
```

## Preprocessing

Also see NumPy & Scikit-Learn

### 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)
```

### Train and Test Sets

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

### Standardization/Normalization

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

## Inspect Model

| | |
|---|---|
| `>>> model.output_shape` | Model output shape |
| `>>> model.summary()` | Model summary representation |
| `>>> model.get_config()` | Model configuration |
| `>>> model.get_weights()` | List all weight tensors in the model |

## Compile Model

MLP: Binary Classification

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

MLP: Multi-Class Classification

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

MLP: Regression

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

Recurrent Neural Network

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

## Model Training

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

## Evaluate Your Model's Performance

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

## Prediction

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

## Save/ Reload Models

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

## Model Fine-tuning

### Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

### Early Stopping

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

DataCamp
Learn Python for Data Science Interactively

# 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.

pandas

Use the following import convention:

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

## Pandas Data Structures

### Series

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

Index

| | |
|---|---|
| A | 3 |
| B | -5 |
| C | 7 |
| D | 4 |

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

### DataFrame

Columns

Index

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

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

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

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

## Asking For Help

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

## Selection

Also see NumPy Arrays

### Getting

```
>>> s['b']
 -5
```

Get one element

```
>>> df[1:]
   Country     Capital  Population
 1   India   New Delhi  1303171035
 2  Brazil    Brasilia   207847528
```

Get subset of a DataFrame

### Selecting, Boolean Indexing & Setting

**By Position**

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

Select single value by row & column

**By Label**

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

Select single value by row & column labels

**By Label/Position**

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

Select single row of subset of rows

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

**Boolean Indexing**

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

Series `s` where value is not >1
`s` where value is <-1 or >2
Use filter to adjust DataFrame

**Setting**

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

Set index `a` of Series `s` to 6

## I/O

### Read and Write to CSV

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

### Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

### Read and Write to SQL Query or Database Table

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

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

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

## Dropping

| | |
|---|---|
| `>>> s.drop(['a', 'c'])` | Drop values from rows (axis=0) |
| `>>> df.drop('Country', axis=1)` | Drop values from columns(axis=1) |

## Sort & Rank

| | |
|---|---|
| `>>> df.sort_index(by='Country')` | Sort by row or column index |
| `>>> s.order()` | Sort a series by its values |
| `>>> df.rank()` | Assign ranks to entries |

## Retrieving Series/DataFrame Information

### Basic Information

| | |
|---|---|
| `>>> df.shape` | (rows,columns) |
| `>>> df.index` | Describe index |
| `>>> df.columns` | Describe DataFrame columns |
| `>>> df.info()` | Info on DataFrame |
| `>>> df.count()` | Number of non-NA values |

### Summary

| | |
|---|---|
| `>>> df.sum()` | Sum of values |
| `>>> df.cumsum()` | Cummulative sum of values |
| `>>> df.min()/df.max()` | Minimum/maximum values |
| `>>> df.idmin()/df.idmax()` | Minimum/Maximum index value |
| `>>> df.describe()` | Summary statistics |
| `>>> df.mean()` | Mean of values |
| `>>> df.median()` | Median of values |

## Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> 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'])
>>> s + s3
 a    10.0
 b     NaN
 c     5.0
 d     7.0
```

### Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
 a   10.0
 b   -5.0
 c    5.0
 d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

DataCamp
Learn Python for Data Science Interactively

# Python For Data Science *Cheat Sheet*

## NumPy Basics

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

## NumPy

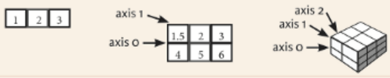
The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

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

### NumPy Arrays

1D array 2D array 3D array




## Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

### Initial Placeholders

| | |
|---|---|
| `>>> np.zeros((3,4))` | Create an array of zeros |
| `>>> np.ones((2,3,4),dtype=np.int16)` | Create an array of ones |
| `>>> d = np.arange(10,25,5)` | Create an array of evenly spaced values (step value) |
| `>>> np.linspace(0,2,9)` | Create an array of evenly spaced values (number of samples) |
| `>>> e = np.full((2,2),7)` | Create a constant array |
| `>>> f = np.eye(2)` | Create a 2X2 identity matrix |
| `>>> np.random.random((2,2))` | Create an array with random values |
| `>>> np.empty((3,2))` | Create an empty array |

## I/O

### Saving & Loading On Disk

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

### Saving & Loading Text Files

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

## Data Types

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

## Inspecting Your Array

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

## Asking For Help

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

## Array Mathematics

### Arithmetic Operations

```
>>> g = a - b                                    Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                             Subtraction
>>> b + a                                        Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                                  Addition
>>> a / b                                        Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                               Division
>>> a * b                                        Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                             Multiplication
>>> np.exp(b)                                    Exponentiation
>>> np.sqrt(b)                                   Square root
>>> np.sin(a)                                    Print sines of an array
>>> np.cos(b)                                    Element-wise cosine
>>> np.log(a)                                    Element-wise natural logarithm
>>> e.dot(f)                                     Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

### Comparison

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

### Aggregate Functions

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

## Copying Arrays

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

## Sorting Arrays

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

## Subsetting, Slicing, Indexing

Also see Lists

### Subsetting

```
>>> a[2]                                         Select the element at the 2nd index
  3
>>> b[1,2]                                       Select the element at row 0 column 2
  6.0                                            (equivalent to b[1][2])
```

### Slicing

```
>>> a[0:2]                                       Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]                                     Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]                                        Select all items at row 0
  array([[1.5, 2., 3.]])                         (equivalent to b[0:1, :])
>>> c[1,...]                                     Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]                                    Reversed array a
  array([3, 2, 1])
```

### Boolean Indexing

```
>>> a[a<2]                                       Select elements from a less than 2
  array([1])
```

### Fancy Indexing

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

## Array Manipulation

### Transposing Array

```
>>> i = np.transpose(b)                          Permute array dimensions
>>> i.T                                          Permute array dimensions
```

### Changing Array Shape

```
>>> b.ravel()                                    Flatten the array
>>> g.reshape(3,-2)                              Reshape, but don't change data
```

### Adding/Removing Elements

```
>>> h.resize((2,6))                              Return a new array with shape (2,6)
>>> np.append(h,g)                               Append items to an array
>>> np.insert(a, 1, 5)                           Insert items in an array
>>> np.delete(a,[1])                             Delete items from an array
```

### Combining Arrays

```
>>> np.concatenate((a,d),axis=0)                 Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))                             Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                                   Stack arrays vertically (row-wise)
>>> np.hstack((e,f))                             Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))                       Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                                   Create stacked column-wise arrays
```

### Splitting Arrays

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

**DataCamp**
Learn Python for Data Science Interactively

# Data Wrangling
## with pandas
## Cheat Sheet
## http://pandas.pydata.org

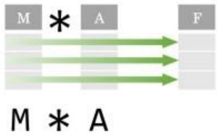
## Tidy Data – A foundation for wrangling in pandas

In a tidy data set:


Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

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




## Syntax – Creating DataFrames

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

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

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

| | | a | b | c |
|---|---|---|---|---|
| n | v | | | |
| d | 1 | 4 | 7 | 10 |
| | 2 | 5 | 8 | 11 |
| e | 2 | 6 | 9 | 12 |

```
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

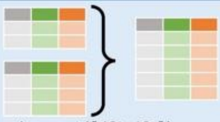
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

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

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

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.



`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`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).

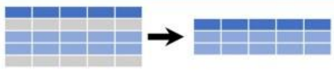
`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`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.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

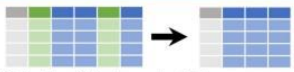
`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

| 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 |

## Subset Variables (Columns)



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

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

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

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

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

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

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

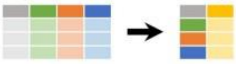
# Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
\# of rows in DataFrame.

`df['w'].nunique()`
\# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)



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

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

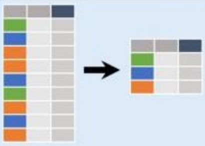
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
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.

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)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

# Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

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

# Handling Missing Data

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

`df.fillna(value)`
Replace all NA/null data with value.

# Make New Columns

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

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.


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

`max(axis=1)`
Element-wise max.

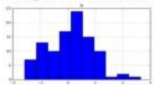
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

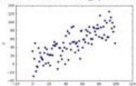
`abs()`
Absolute value.

# 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 |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |

bdf

| x1 | x3 |
|---|---|
| A | T |
| B | F |
| D | T |

Standard Joins

| x1 | x2 | x3 |
|---|---|---|
| A | 1 | T |
| B | 2 | F |
| C | 3 | NaN |

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

| x1 | x2 | x3 |
|---|---|---|
| A | 1.0 | T |
| B | 2.0 | F |
| D | NaN | T |

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

| x1 | x2 | x3 |
|---|---|---|
| A | 1 | T |
| B | 2 | F |

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

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

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

Filtering Joins

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

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

| x1 | x2 |
|---|---|
| C | 3 |

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

ydf

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

zdf

| x1 | x2 |
|---|---|
| B | 2 |
| C | 3 |
| D | 4 |

Set-like Operations

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

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

| x1 | x2 |
|---|---|
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |

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

| x1 | x2 |
|---|---|
| A | 1 |

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

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

# Data Wrangling
## with dplyr and tidyr
### Cheat Sheet

RStudio

## Tidy Data - A foundation for wrangling in R

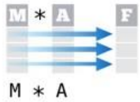
In a tidy data set:


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.




## Syntax - Helpful conventions for wrangling

dplyr::**tbl_df(iris)**

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

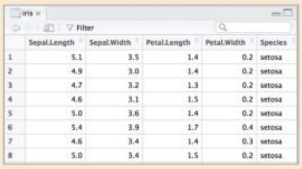
```
Source: local data frame [150 x 5]

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

dplyr::**glimpse(iris)**

Information dense summary of tbl data.

utils::**View(iris)**

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




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

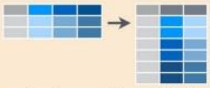
**x %>% f(y)** *is the same as* **f(x, y)**

**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)
```

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



tidyr::**gather(cases, "year", "n", 2:4)**

Gather columns into rows.

tidyr::**spread(pollution, size, amount)**

Spread rows into columns.

tidyr::**separate(storms, date, c("y", "m", "d"))**

Separate one column into several.

tidyr::**unite(data, col, ..., sep)**

Unite several columns into one.

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

Combine vectors into data frame (optimized).

dplyr::**arrange(mtcars, mpg)**

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

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

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

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

Rename the columns of a data frame.

## Subset Observations (Rows)

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

Extract rows that meet logical criteria.

dplyr::**distinct(iris)**

Remove duplicate rows.

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

Randomly select fraction of rows.

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

Randomly select n rows.

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

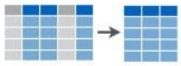
Select rows by position.

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

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

| Logic in R - ?Comparison, ?base::Logic | | | |
|---|---|---|---|
| < | Less than | != | Not equal to |
| > | Greater than | %in% | Group membership |
| == | Equal to | is.na | Is NA |
| <= | Less than or equal to | !is.na | Is not NA |
| >= | Greater than or equal to | &,\|,!,xor,any,all | Boolean operators |

## Subset Variables (Columns)



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

Select columns by name or helper function.

**Helper functions for select - ?select**

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



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

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

## Summarise Data

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

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

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


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

dplyr::**first**
First value of a vector.

dplyr::**last**
Last value of a vector.

dplyr::**nth**
Nth value of a vector.

dplyr::**n**
# of values in a vector.

dplyr::**n_distinct**
# of distinct values in a vector.

**IQR**
IQR of a vector.

**min**
Minimum value in a vector.

**max**
Maximum value in a vector.

**mean**
Mean value of a vector.

**median**
Median value of a vector.

**var**
Variance of a vector.

**sd**
Standard deviation of a vector.

## Group Data

dplyr::**group_by(iris, Species)**
Group data into rows with the same value of Species.

dplyr::**ungroup(iris)**
Remove grouping information from data frame.

**iris %>% group_by(Species) %>% summarise(...)**
Compute separate summary row for each group.

## Make New Variables

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

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

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


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

dplyr::**lead**
Copy with values shifted by 1.

dplyr::**lag**
Copy with values lagged by 1.

dplyr::**dense_rank**
Ranks with no gaps.

dplyr::**min_rank**
Ranks. Ties get min rank.

dplyr::**percent_rank**
Ranks rescaled to [0, 1].

dplyr::**row_number**
Ranks. Ties got to first value.

dplyr::**ntile**
Bin vector into n buckets.

dplyr::**between**
Are values between a and b?

dplyr::**cume_dist**
Cumulative distribution.

dplyr::**cumall**
Cumulative `all`

dplyr::**cumany**
Cumulative `any`

dplyr::**cummean**
Cumulative `mean`

**cumsum**
Cumulative `sum`

**cummax**
Cumulative `max`

**cummin**
Cumulative `min`

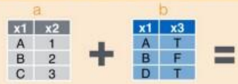
**cumprod**
Cumulative `prod`

**pmax**
Element-wise `max`

**pmin**
Element-wise `min`

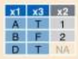
**iris %>% group_by(Species) %>% mutate(...)**
Compute new variables by group.

## Combine Data Sets

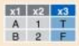



### Mutating Joins

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



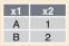
dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.



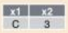
dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

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

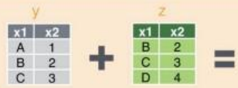
### Filtering Joins



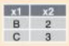
dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.



dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.




### Set Operations



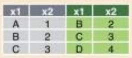
dplyr::**intersect(y, z)**
Rows that appear in both y and z.

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

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

### Binding

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



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



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

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

# 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+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

### Index Tricks

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

### Shape Manipulation

| | |
|---|---|
| `>>> np.transpose(b)` | Permute array dimensions |
| `>>> b.flatten()` | Flatten the array |
| `>>> np.hstack((b,c))` | Stack arrays horizontally (column-wise) |
| `>>> np.vstack((a,b))` | Stack arrays vertically (row-wise) |
| `>>> np.hsplit(c,2)` | Split the array horizontally at the 2nd index |
| `>>> np.vpslit(d,2)` | Split the array vertically at the 2nd index |

### Polynomials

| | |
|---|---|
| `>>> from numpy import poly1d`<br>`>>> p = poly1d([3,4,5])` | Create a polynomial object |

### Vectorizing Functions

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

Vectorize functions

### Type Handling

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

### Other Useful Functions

| | |
|---|---|
| `>>> np.angle(b,deg=True)` | Return the angle of the complex argument |
| `>>> g = np.linspace(0,np.pi,num=5)` | Create an array of evenly spaced values (number of samples) |
| `>>> g [3:] += np.pi` | |
| `>>> np.unwrap(g)` | Unwrap |
| `>>> np.logspace(0,10,3)` | Create an array of evenly spaced values (log scale) |
| `>>> np.select([c<4],[c*2])` | Return values from a list of arrays depending on conditions |
| `>>> misc.factorial(a)` | Factorial |
| `>>> misc.comb(10,3,exact=True)` | Combine N things taken at k time |
| `>>> misc.central_diff_weights(3)` | Weights for Np-point central derivative |
| `>>> misc.derivative(myfunc,1.0)` | Find the n-th derivative of a function at a point |

## Linear Algebra

Also see NumPy

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

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

### Creating Matrices

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

### Basic Matrix Routines

| | |
|---|---|
| **Inverse** | |
| `>>> A.I` | Inverse |
| `>>> linalg.inv(A)` | Inverse |
| **Transposition** | |
| `>>> A.T` | Tranpose matrix |
| `>>> A.H` | Conjugate transposition |
| **Trace** | |
| `>>> np.trace(A)` | Trace |
| **Norm** | |
| `>>> linalg.norm(A)` | Frobenius norm |
| `>>> linalg.norm(A,1)` | L1 norm (max column sum) |
| `>>> linalg.norm(A,np.inf)` | L inf norm (max row sum) |
| **Rank** | |
| `>>> np.linalg.matrix_rank(C)` | Matrix rank |
| **Determinant** | |
| `>>> linalg.det(A)` | Determinant |
| **Solving linear problems** | |
| `>>> linalg.solve(A,b)` | Solver for dense matrices |
| `>>> E = np.mat(a).T` | Solver for dense matrices |
| `>>> linalg.lstsq(F,E)` | Least-squares solution to linear matrix equation |
| **Generalized inverse** | |
| `>>> linalg.pinv(C)` | Compute the pseudo-inverse of a matrix (least-squares solver) |
| `>>> linalg.pinv2(C)` | Compute the pseudo-inverse of a matrix (SVD) |

### Creating Sparse Matrices

| | |
|---|---|
| `>>> F = np.eye(3, k=1)` | Create a 2X2 identity matrix |
| `>>> G = np.mat(np.identity(2))` | Create a 2x2 identity matrix |
| `>>> C[C > 0.5] = 0` | |
| `>>> H = sparse.csr_matrix(C)` | Compressed Sparse Row matrix |
| `>>> I = sparse.csc_matrix(D)` | Compressed Sparse Column matrix |
| `>>> J = sparse.dok_matrix(A)` | Dictionary Of Keys matrix |
| `>>> E.todense()` | Sparse matrix to full matrix |
| `>>> sparse.isspmatrix_csc(A)` | Identify sparse matrix |

### Sparse Matrix Routines

| | |
|---|---|
| **Inverse** | |
| `>>> sparse.linalg.inv(I)` | Inverse |
| **Norm** | |
| `>>> sparse.linalg.norm(I)` | Norm |
| **Solving linear problems** | |
| `>>> sparse.linalg.spsolve(H,I)` | Solver for sparse matrices |

### Sparse Matrix Functions

| | |
|---|---|
| `>>> sparse.linalg.expm(I)` | Sparse matrix exponential |

### Matrix Functions

| | |
|---|---|
| **Addition** | |
| `>>> np.add(A,D)` | Addition |
| **Subtraction** | |
| `>>> np.subtract(A,D)` | Subtraction |
| **Division** | |
| `>>> np.divide(A,D)` | Division |
| **Multiplication** | |
| `>>> A @ D` | Multiplication operator (Python 3) |
| `>>> np.multiply(D,A)` | Multiplication |
| `>>> np.dot(A,D)` | Dot product |
| `>>> np.vdot(A,D)` | Vector dot product |
| `>>> np.inner(A,D)` | Inner product |
| `>>> np.outer(A,D)` | Outer product |
| `>>> np.tensordot(A,D)` | Tensor dot product |
| `>>> np.kron(A,D)` | Kronecker product |
| **Exponential Functions** | |
| `>>> linalg.expm(A)` | Matrix exponential |
| `>>> linalg.expm2(A)` | Matrix exponential (Taylor Series) |
| `>>> linalg.expm3(D)` | Matrix exponential (eigenvalue decomposition) |
| **Logarithm Function** | |
| `>>> linalg.logm(A)` | Matrix logarithm |
| **Trigonometric Functions** | |
| `>>> linalg.sinm(D)` | Matrix sine |
| `>>> linalg.cosm(D)` | Matrix cosine |
| `>>> linalg.tanm(A)` | Matrix tangent |
| **Hyperbolic Trigonometric Functions** | |
| `>>> linalg.sinhm(D)` | Hypberbolic matrix sine |
| `>>> linalg.coshm(D)` | Hyperbolic matrix cosine |
| `>>> linalg.tanhm(A)` | Hyperbolic matrix tangent |
| **Matrix Sign Function** | |
| `>>> np.signm(A)` | Matrix sign function |
| **Matrix Square Root** | |
| `>>> linalg.sqrtm(A)` | Matrix square root |
| **Arbitrary Functions** | |
| `>>> linalg.funm(A, lambda x: x*x)` | Evaluate matrix function |

### Decompositions

| | |
|---|---|
| **Eigenvalues and Eigenvectors** | |
| `>>> la, v = linalg.eig(A)` | Solve ordinary or generalized eigenvalue problem for square matrix |
| `>>> l1, l2 = la` | Unpack eigenvalues |
| `>>> v[:,0]` | First eigenvector |
| `>>> v[:,1]` | Second eigenvector |
| `>>> linalg.eigvals(A)` | Unpack eigenvalues |
| **Singular Value Decomposition** | |
| `>>> U,s,Vh = linalg.svd(B)` | Singular Value Decomposition (SVD) |
| `>>> M,N = B.shape` | |
| `>>> Sig = linalg.diagsvd(s,M,N)` | Construct sigma matrix in SVD |
| **LU Decomposition** | |
| `>>> P,L,U = linalg.lu(C)` | LU Decomposition |

### Sparse Matrix Decompositions

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

## Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

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

## 2 Create Plot

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

### Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

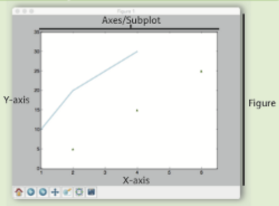
### Axes

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

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

## Plot Anatomy & Workflow

### Plot Anatomy




### Workflow

The basic steps to creating plots with matplotlib are:

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

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

## 3 Plotting Routines

### 1D Data

| | |
|---|---|
| `>>> fig, ax = plt.subplots()` | |
| `>>> lines = ax.plot(x,y)` | Draw points with lines or markers connecting them |
| `>>> ax.scatter(x,y)` | Draw unconnected points, scaled or colored |
| `>>> axes[0,0].bar([1,2,3],[3,4,5])` | Plot vertical rectangles (constant width) |
| `>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])` | Plot horiontal rectangles (constant height) |
| `>>> axes[1,1].axhline(0.45)` | Draw a horizontal line across axes |
| `>>> axes[0,1].axvline(0.65)` | Draw a vertical line across axes |
| `>>> ax.fill(x,y,color='blue')` | Draw filled polygons |
| `>>> ax.fill_between(x,y,color='yellow')` | Fill between y-values and 0 |

### 2D Data or Images

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

Colormapped or RGB arrays

| | |
|---|---|
| `>>> axes2[0].pcolor(data2)` | Pseudocolor plot of 2D array |
| `>>> axes2[0].pcolormesh(data)` | Pseudocolor plot of 2D array |
| `>>> CS = plt.contour(Y,X,U)` | Plot contours |
| `>>> axes2[2].contourf(data1)` | Plot filled contours |
| `>>> axes2[2]= ax.clabel(CS)` | Label a contour plot |

### Vector Fields

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

### Data Distributions

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

## 4 Customize Plot

### Colors, Color Bars & Color Maps

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

### Markers

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

### Linestyles

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

### Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

### Mathtext

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

### Limits, Legends & Layouts

**Limits & Autoscaling**

| | |
|---|---|
| `>>> ax.margins(x=0.0,y=0.1)` | Add padding to a plot |
| `>>> ax.axis('equal')` | Set the aspect ratio of the plot to 1 |
| `>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])` | Set limits for x-and y-axis |
| `>>> ax.set_xlim(0,10.5)` | Set limits for x-axis |

**Legends**

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

Set a title and x-and y-axis labels

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

No overlapping plot elements

**Ticks**

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

Manually set x-ticks

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

Make y-ticks longer and go in and out

**Subplot Spacing**

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

Adjust the spacing between subplots

```
>>> fig.tight_layout()
```

Fit subplot(s) in to the figure area

**Axis Spines**

| | |
|---|---|
| `>>> ax1.spines['top'].set_visible(False)` | Make the top axis line for a plot invisible |
| `>>> ax1.spines['bottom'].set_position(('outward',10))` | Move the bottom axis line outward |

## 5 Save Plot

**Save figures**

```
>>> plt.savefig('foo.png')
```

**Save transparent figures**

```
>>> plt.savefig('foo.png', transparent=True)
```

## 6 Show Plot

```
>>> plt.show()
```

## Close & Clear

| | |
|---|---|
| `>>> plt.cla()` | Clear an axis |
| `>>> plt.clf()` | Clear the entire figure |
| `>>> plt.close()` | Close a window |

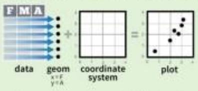
**DataCamp**
Learn Python for Data Science Interactively

# Data Visualization
## with ggplot2
### Cheat Sheet

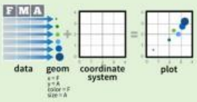
RStudio

## Basics

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




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




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

aesthetic mappings | data | geom

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

**ggplot**(data = mpg, **aes**(x = cty, y = hwy))
Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

data

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

add layers, elements with +

layer = geom + default stat + layer specific mappings

additional elements

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

**last_plot()**
Returns the last plot

**ggsave("plot.png", width = 5, height = 5)**
Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

## 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
b + geom_area(aes(y = ..density..), stat = "bin")

a + **geom_density(**kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

a + **geom_dotplot()**
x, y, alpha, color, fill

a + **geom_freqpoly()**
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

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

#### 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

### 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

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

#### Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + **geom_bar(stat = "identity")**
x, y, alpha, color, fill, linetype, size, weight

g + **geom_boxplot()**
lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

g + **geom_dotplot(**binaxis = "y", stackdir = "center"**)**
x, y, alpha, color, fill

g + **geom_violin(**scale = "area"**)**
x, y, alpha, color, fill, linetype, size, weight

#### Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

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

#### Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + **geom_bin2d(**binwidth = c(5, 0.5)**)**
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + **geom_density2d()**
x, y, alpha, colour, linetype, size

i + **geom_hex()**
x, y, alpha, colour, fill size

#### 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_contour(**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

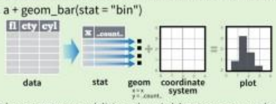
m + **geom_tile(**aes(fill = z)**)**
x, y, alpha, color, fill, linetype, size



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")**

stat function | layer specific mappings | variable created by transformation

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

geom for layer | parameters for stat

**1D distributions**

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

**2D distributions**

f + **stat_bin2d**(bins = 30, drop = TRUE)
 **x, y, fill** | ..count.., ..density..
f + **stat_binhex**(bins = 30)
 **x, y, fill** | ..count.., ..density..
f + **stat_density2d**(contour = TRUE, n = 100)
 **x, y, color, size** | ..level..

**3 Variables**

m + **stat_contour**(aes(z = z))
 **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, z, fill** | ..value..
m + **stat_summary2d**(aes(z = z), bins = 30, fun = mean)
 **x, y, z, fill** | ..value..

**Comparisons**

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

**Functions**

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

**General Purpose**

ggplot() + **stat_function**(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
 **x** | ..y..
f + **stat_identity**()
ggplot() + **stat_qq**(aes(sample=1:100), distribution = qt, dparams = list(df=5))
 **sample, x, y** | ..x.., ..y..
f + **stat_sum**()
 **x, y, size** | ..size..
f + **stat_summary**(fun.data = "mean_cl_boot")
f + **stat_unique**()

## 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**(aes(fill = fl))
n

scale_ | aesthetic to adjust | prepackaged scale to use | scale specific arguments

n + **scale_fill_manual**(
 **values** = c("skyblue", "royalblue", "blue", "navy"),
 **limits** = c("d", "e", "p", "r"), **breaks** =c("d", "e", "p", "r"),
 **name** = "fuel", **labels** = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

### General Purpose scales

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

**scale_*_continuous()** - map cont' values to visual values
**scale_*_discrete()** - map discrete values to visual values
**scale_*_identity()** - use data values **as** visual values
**scale_*_manual**(values = c()) - map discrete values to manually chosen visual values

### X and Y location scales

Use with x or y aesthetics (x shown here)

**scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks"))** - treat x values as dates. 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**

n <- b + **geom_bar**(aes(fill = fl))
n + **scale_fill_brewer**(palette = "Blues")
For palette choices: library(RcolorBrewer) display.brewer.all()
n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")

**Continuous**

o <- a + **geom_dotplot**(aes(fill = ..x..))
o + **scale_fill_gradient**(low = "red", high = "yellow")
o + **scale_fill_gradient2**(low = "red", hight = "blue", mid = "white", midpoint = 25)
o + **scale_fill_gradientn**(colours = terrain.colors(6))
Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

### Shape scales

p <- f + **geom_point**(aes(shape = fl))
p + **scale_shape**(solid = FALSE)
p + **scale_shape_manual**(values = c(3:7))
Shape values shown in chart on right

Manual shape values

### Size scales

q <- f + **geom_point**(aes(size = cyl))
q + **scale_size_area**(max = 6)
Value mapped to area of circle (not radius)

## Coordinate Systems

r <- b + geom_bar()

r + **coord_cartesian**(xlim = c(0, 5))
 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**(ytrans = "sqrt")
 xtrans, ytrans, limx, limy
 Transformed cartesian coordinates. Set extras and strains to the name of a window function.
z + **coord_map**(projection = "ortho", orientation=c(41, -74, 0))
 projection, orientation, xlim, ylim
 Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

## Position Adjustments

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

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

s + geom_bar(**position = "dodge"**)
 Arrange elements side by side
s + geom_bar(**position = "fill"**)
 Stack elements on top of one another, normalize height
s + geom_bar(**position = "stack"**)
 Stack elements on top of one another
f + geom_point(**position = "jitter"**)
 Add random noise to X and Y position of each element to avoid overplotting

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

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

## Themes

r + **theme_bw()**
 White background with grid lines
r + **theme_classic()**
 White background no gridlines
r + **theme_grey()**
 Grey background (default theme)
r + **theme_minimal()**
 Minimal theme

ggthemes - Package with additional ggplot2 themes

## Faceting

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

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

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

Set **scales** to let axis limits vary across facets

t + facet_grid(y ~ 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 |
|---|---|---|---|---|

## 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

Use scale functions to update legend labels

## Legends

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

## Zooming

**Without clipping** (preferred)

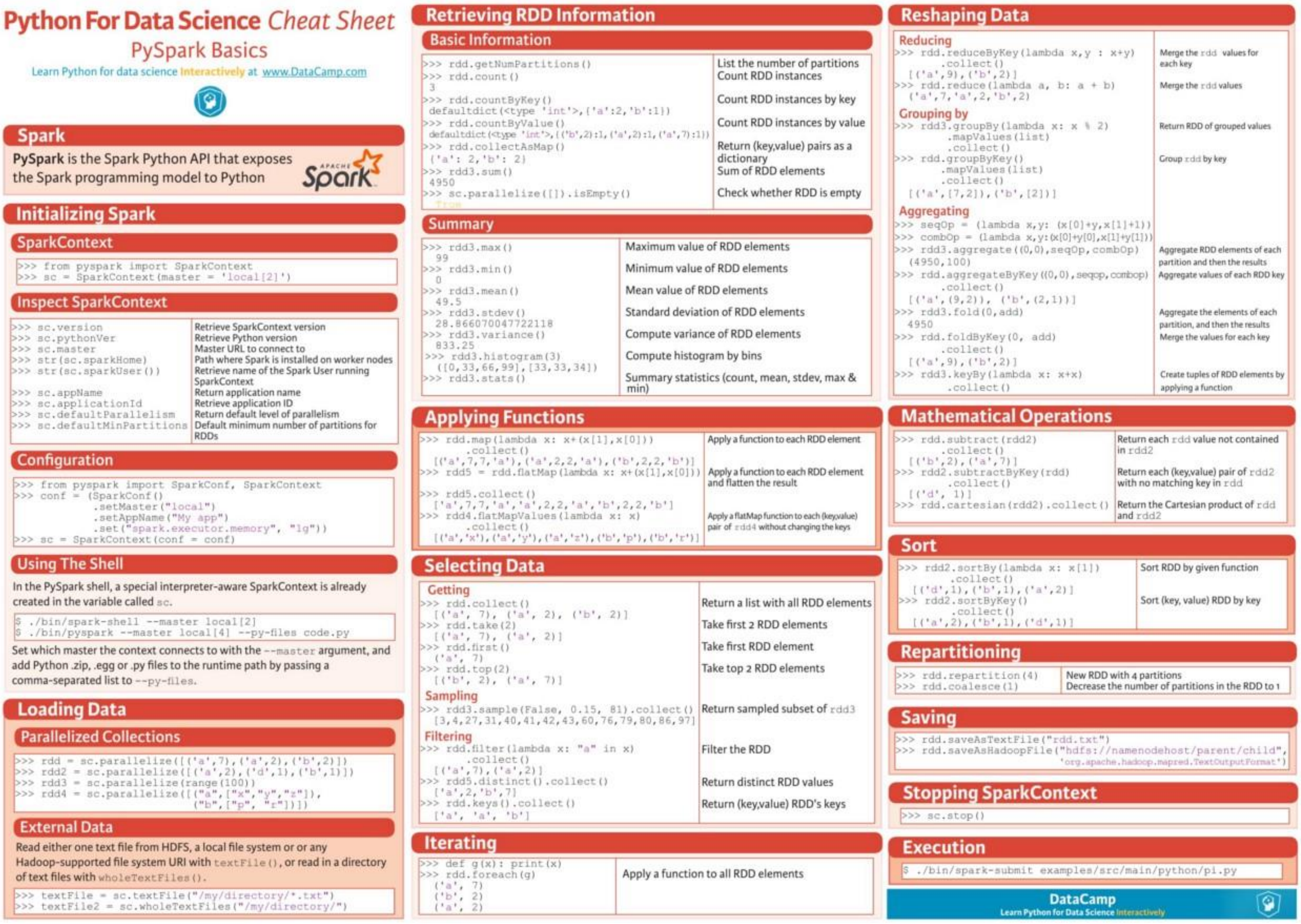
t + **coord_cartesian**(
 xlim = c(0, 100), ylim = c(10, 20))

**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` | Return application name |
| `>>> sc.applicationId` | Retrieve application ID |
| `>>> sc.defaultParallelism` | Return default level of parallelism |
| `>>> sc.defaultMinPartitions` | Default minimum number of partitions for RDDs |

#### Configuration
```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
            .setMaster("local")
            .setAppName("My app")
            .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```
#### Using The Shell
In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.
```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```
Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

### Loading Data
#### Parallelized Collections
```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```
#### External Data
Read either one text file from HDFS, a local file system or 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()` | List the number of partitions |
| `>>> rdd.count()` `3` | Count RDD instances |
| `>>> rdd.countByKey()` `defaultdict(<type 'int'>,{'a':2,'b':1})` | Count RDD instances by key |
| `>>> rdd.countByValue()` `defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})` | Count RDD instances by value |
| `>>> rdd.collectAsMap()` `{'a': 2,'b': 2}` | Return (key,value) pairs as a dictionary |
| `>>> rdd3.sum()` `4950` | Sum of RDD elements |
| `>>> sc.parallelize([]).isEmpty()` `True` | Check whether RDD is empty |

#### Summary
| | |
|---|---|
| `>>> rdd3.max()` `99` | Maximum value of RDD elements |
| `>>> rdd3.min()` `0` | Minimum value of RDD elements |
| `>>> rdd3.mean()` `49.5` | Mean value of RDD elements |
| `>>> rdd3.stdev()` `28.866070047722118` | Standard deviation of RDD elements |
| `>>> rdd3.variance()` `833.25` | Compute variance of RDD elements |
| `>>> rdd3.histogram(3)` `([0,33,66,99],[33,33,34])` | Compute histogram by bins |
| `>>> rdd3.stats()` | Summary statistics (count, mean, stdev, max & min) |

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

### Selecting Data
| | |
|---|---|
| **Getting** | |
| `>>> rdd.collect()` `[('a', 7), ('a', 2), ('b', 2)]` | Return a list with all RDD elements |
| `>>> rdd.take(2)` `[('a', 7), ('a', 2)]` | Take first 2 RDD elements |
| `>>> rdd.first()` `('a', 7)` | Take first RDD element |
| `>>> rdd.top(2)` `[('b', 2), ('a', 7)]` | Take top 2 RDD elements |
| **Sampling** | |
| `>>> rdd3.sample(False, 0.15, 81).collect()` `[3,4,27,31,40,41,42,43,60,76,79,80,86,97]` | Return sampled subset of `rdd3` |
| **Filtering** | |
| `>>> rdd.filter(lambda x: "a" in x).collect()` `[('a',7),('a',2)]` | Filter the RDD |
| `>>> rdd5.distinct().collect()` `['a',2,'b',7]` | Return distinct RDD values |
| `>>> rdd.keys().collect()` `['a', 'a', 'b']` | Return (key,value) RDD's keys |

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

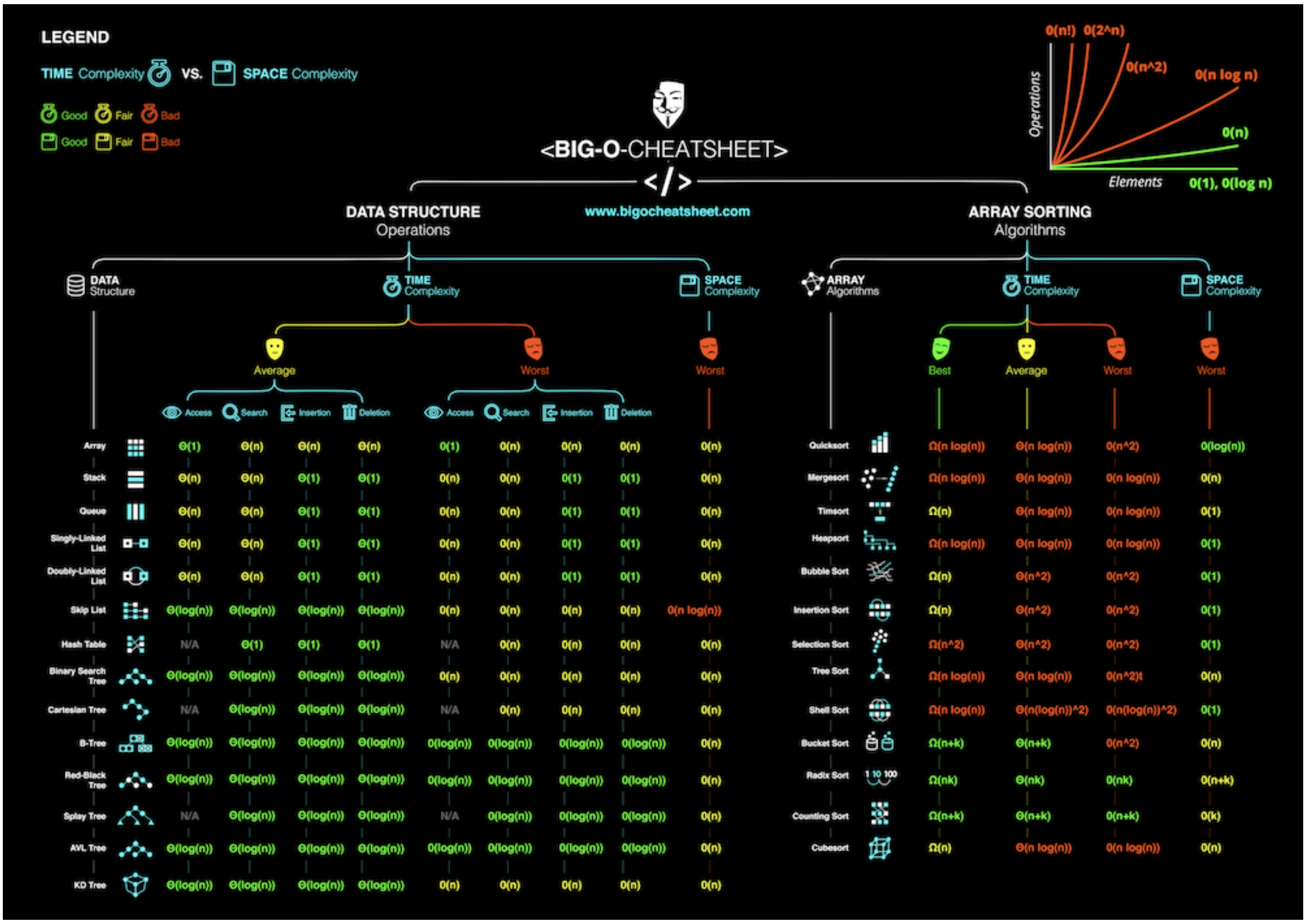
### Reshaping Data
| | |
|---|---|
| **Reducing** | |
| `>>> rdd.reduceByKey(lambda x,y : x+y).collect()` `[('a',9),('b',2)]` | Merge the `rdd` values for each key |
| `>>> rdd.reduce(lambda a, b: a + b)` `('a',7,'a',2,'b',2)` | Merge the `rdd` values |
| **Grouping by** | |
| `>>> rdd3.groupBy(lambda x: x % 2).mapValues(list).collect()` | Return RDD of grouped values |
| `>>> rdd.groupByKey().mapValues(list).collect()` `[('a',[7,2]),('b',[2])]` | Group `rdd` by key |
| **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)` | Aggregate RDD elements of each partition and then the results |
| `>>> rdd.aggregateByKey((0,0),seqop,combop).collect()` `[('a',(9,2)), ('b',(2,1))]` | Aggregate values of each RDD key |
| `>>> rdd3.fold(0,add)` `4950` | Aggregate the elements of each partition, and then the results |
| `>>> rdd.foldByKey(0, add).collect()` `[('a',9),('b',2)]` | Merge the values for each key |
| `>>> rdd3.keyBy(lambda x: x+x).collect()` | Create tuples of RDD elements by applying a function |

### Mathematical Operations
| | |
|---|---|
| `>>> rdd.subtract(rdd2).collect()` `[('b',2),('a',7)]` | Return each `rdd` value not contained in `rdd2` |
| `>>> rdd2.subtractByKey(rdd).collect()` `[('d', 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)]` | Sort RDD by given function |
| `>>> rdd2.sortByKey().collect()` `[('a',2),('b',1),('d',1)]` | Sort (key, value) RDD by key |

### Repartitioning
| | |
|---|---|
| `>>> rdd.repartition(4)` | New RDD with 4 partitions |
| `>>> rdd.coalesce(1)` | 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>
</>
www.bigocheatsheet.com



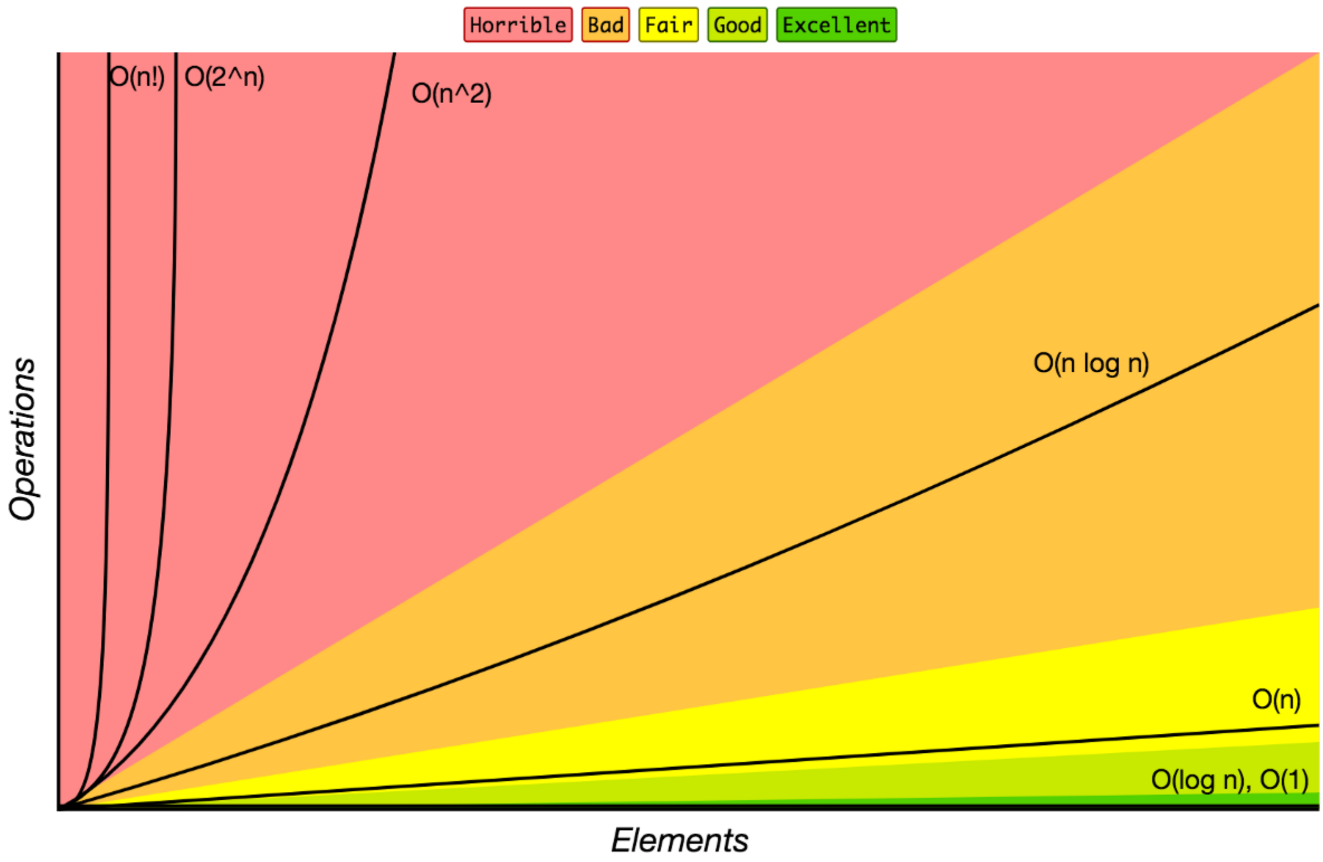
**DATA STRUCTURE** Operations

| Data Structure | Time Complexity: Average Access | Average Search | Average Insertion | Average Deletion | Worst Access | Worst Search | Worst Insertion | Worst Deletion | Space Complexity: Worst |
|---|---|---|---|---|---|---|---|---|---|
| Array | Θ(1) | Θ(n) | Θ(n) | Θ(n) | 0(1) | 0(n) | 0(n) | 0(n) | 0(n) |
| Stack | Θ(n) | Θ(n) | Θ(1) | Θ(1) | 0(n) | 0(n) | 0(1) | 0(1) | 0(n) |
| Queue | Θ(n) | Θ(n) | Θ(1) | Θ(1) | 0(n) | 0(n) | 0(1) | 0(1) | 0(n) |
| Singly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | 0(n) | 0(n) | 0(1) | 0(1) | 0(n) |
| Doubly-Linked List | Θ(n) | Θ(n) | Θ(1) | Θ(1) | 0(n) | 0(n) | 0(1) | 0(1) | 0(n) |
| Skip List | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | 0(n) | 0(n) | 0(n) | 0(n) | 0(n log(n)) |
| Hash Table | N/A | Θ(1) | Θ(1) | Θ(1) | N/A | 0(n) | 0(n) | 0(n) | 0(n) |
| Binary Search Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | 0(n) | 0(n) | 0(n) | 0(n) | 0(n) |
| Cartesian Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | 0(n) | 0(n) | 0(n) | 0(n) |
| B-Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(n) |
| Red-Black Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(n) |
| Splay Tree | N/A | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | N/A | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(n) |
| AVL Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(log(n)) | 0(n) |
| KD Tree | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | Θ(log(n)) | 0(n) | 0(n) | 0(n) | 0(n) | 0(n) |

**ARRAY SORTING** Algorithms

| Array Algorithms | Time Complexity: Best | Average | Worst | Space Complexity: Worst |
|---|---|---|---|---|
| Quicksort | Ω(n log(n)) | Θ(n log(n)) | 0(n^2) | 0(log(n)) |
| Mergesort | Ω(n log(n)) | Θ(n log(n)) | 0(n log(n)) | 0(n) |
| Timsort | Ω(n) | Θ(n log(n)) | 0(n log(n)) | 0(1) |
| Heapsort | Ω(n log(n)) | Θ(n log(n)) | 0(n log(n)) | 0(1) |
| Bubble Sort | Ω(n) | Θ(n^2) | 0(n^2) | 0(1) |
| Insertion Sort | Ω(n) | Θ(n^2) | 0(n^2) | 0(1) |
| Selection Sort | Ω(n^2) | Θ(n^2) | 0(n^2) | 0(1) |
| Tree Sort | Ω(n log(n)) | Θ(n log(n)) | 0(n^2)t | 0(n) |
| Shell Sort | Ω(n log(n)) | Θ(n(log(n))^2) | 0(n(log(n))^2) | 0(1) |
| Bucket Sort | Ω(n+k) | Θ(n+k) | 0(n^2) | 0(n) |
| Radix Sort | Ω(nk) | Θ(nk) | 0(nk) | 0(n+k) |
| Counting Sort | Ω(n+k) | Θ(n+k) | 0(n+k) | 0(k) |
| Cubesort | Ω(n) | Θ(n log(n)) | 0(n log(n)) | 0(n) |

## Big-O Complexity Chart




## Common Data Structure Operations

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

# Array Sorting Algorithms

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