A mostly complete chart of

Neural Networks

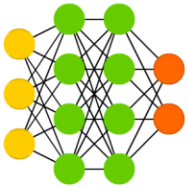
- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

Perceptron (P)

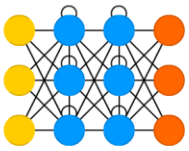
Feed Forward (FF)

Radial Basis Network (RBF)

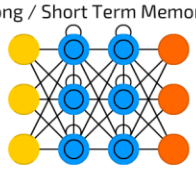
Deep Feed Forward (DFF)



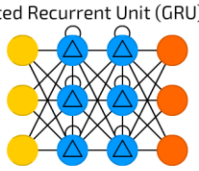
Recurrent Neural Network (RNN)



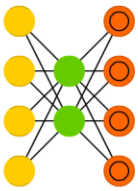
Long / Short Term Memory (LSTM)



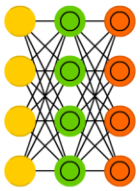
Gated Recurrent Unit (GRU)



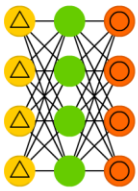
Auto Encoder (AE)



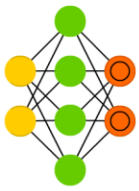
Variational AE (VAE)



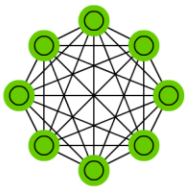
Denoising AE (DAE)



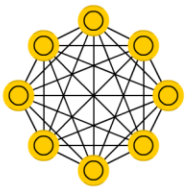
Sparse AE (SAE)



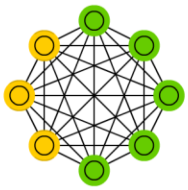
Markov Chain (MC)



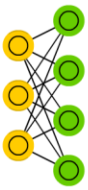
Hopfield Network (HN)



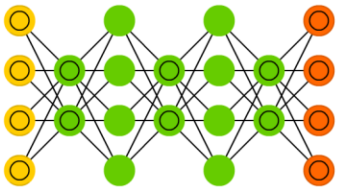
Boltzmann Machine (BM)



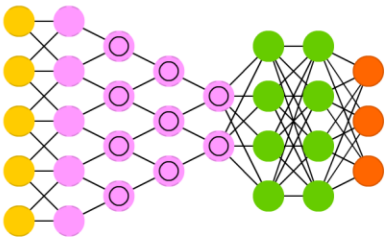
Restricted BM (RBM)



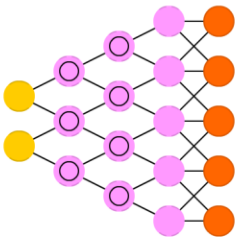
Deep Belief Network (DBN)



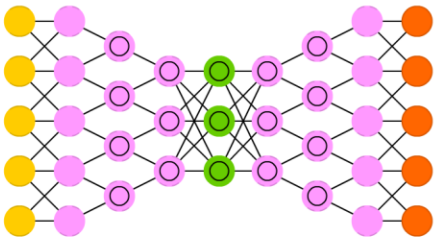
Deep Convolutional Network (DCN)



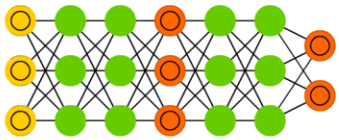
Deconvolutional Network (DN)



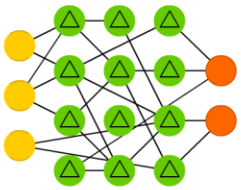
Deep Convolutional Inverse Graphics Network (DCIGN)



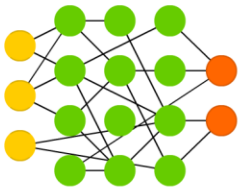
Generative Adversarial Network (GAN)



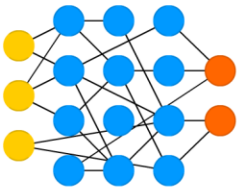
Liquid State Machine (LSM)



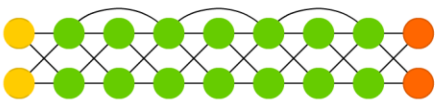
Extreme Learning Machine (ELM)



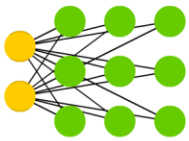
Echo State Network (ESN)



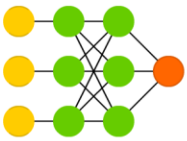
Deep Residual Network (DRN)



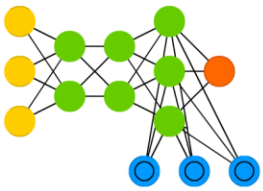
Kohonen Network (KN)



Support Vector Machine (SVM)

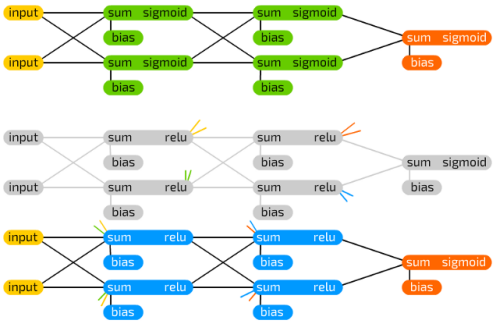


Neural Turing Machine (NTM)

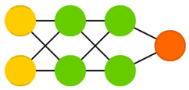


An informative chart to build

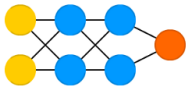
Neural Network Graphs



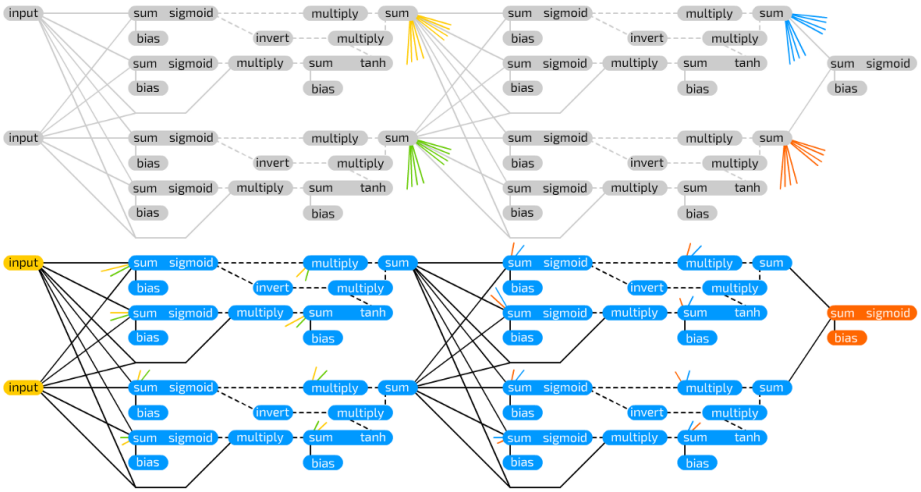
Deep Feed Forward Example



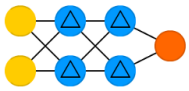
Deep Recurrent Example
(previous iteration)



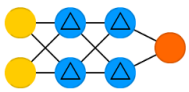
Deep Recurrent Example

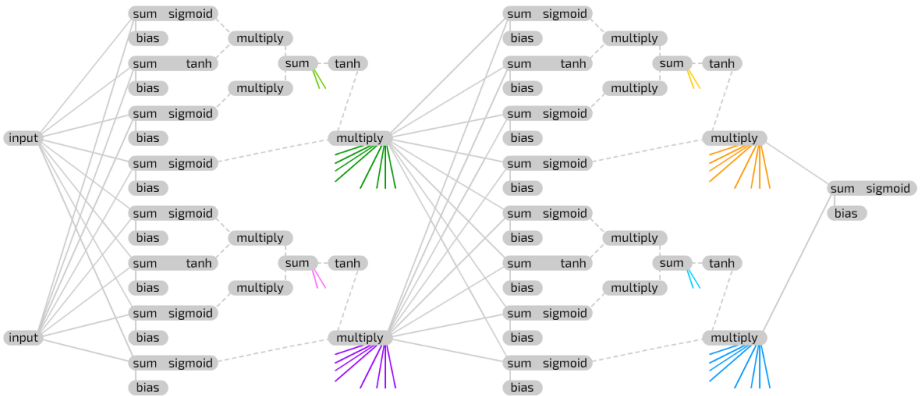


Deep GRU Example
(previous iteration)

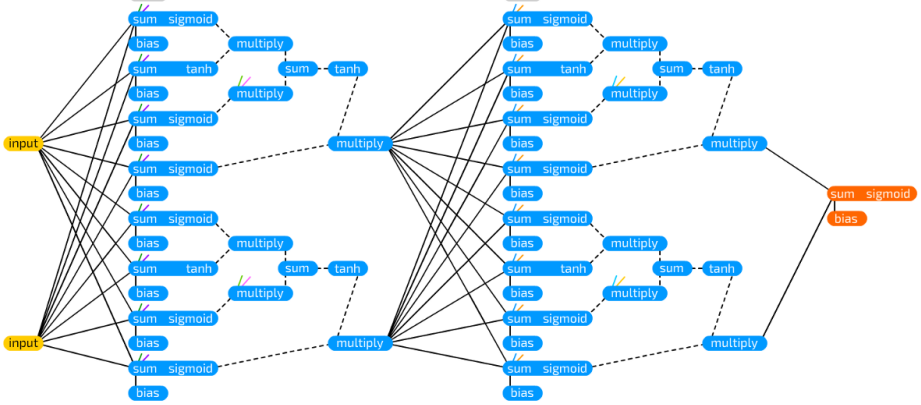


Deep GRU Example





Deep LSTM Example
(previous iteration)



Deep LSTM Example

Linear Vector Spaces:

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

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

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

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

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

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

9) $(a+b)x = ax + bx$. **10)** $a(x+y) = ax + ay$.

Linear Independence: Consider n vectors $\{x_1, x_2, .., x_n\}$. 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: Ascalar function $\|x\|$ is called a norm if it satisfies:

1. $\|x\| \geq 0$ 2. $\|x\| = 0$ if and only if $x = 0$.

3. $\|ax\| = |a|\|x\|$ 4. $\|x + y\| \leq \|x\| + \|y\|$

Angle: The angle θ bet. 2 vectors x and y is defined by $\cos\theta = \frac{(x,y)}{\|x\|\,\|y\|}$

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:

$$x = \sum_{i=1}^{n} x_i v_i = x_1v_1 + x_2v_2 + \cdots + x_nv_n,$$

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

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 $\mathbf{B} = [v_1\ v_2\ \ldots\ v_n]$, $\mathbf{R} = [r_1\ r_2\ \ldots\ r_n]$, $\mathbf{R}^T = \mathbf{B}^{-1}$ 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: $\mathbf{Az} = \lambda\mathbf{z}$, $|[\mathbf{A} - \lambda\mathbf{I}]| = 0$

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

where $\{\mathbf{z}, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ 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:

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

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

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

The decision boundary is always orthogonal to the weight vector.

Single-layer perceptrons can only classify linearly separable vectors.

Perceptron Learning Rule

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

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:* $w_{ij}^{new} = w_{ij}^{old} + t_{qi}P_{qi}$

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

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

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

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: $\nabla^2 F(\mathbf{x}) =$

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

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 Δx such that $\delta > \|\Delta x\| > 0$.

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

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

Necessary Conditions for Optimality:

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

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

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

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

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: $\mathbf{a} = purelin(\mathbf{W}\mathbf{p} + \mathbf{b})$

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 λ_{max} is the maximum eigenvalue of R

Adaptive Filter ADALINE:

$$a(k) = purelin(\mathbf{W}\mathbf{p}(k) + b) = \sum_{i=1}^{R} \mathbf{w}_{1,i} y(k - i + 1) + b$$

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)

$\hat{F}(x) = \mathbf{e}^T(k)\mathbf{e}(k) = (\mathbf{t}(k) - \mathbf{a}(k))^T(\mathbf{t}(k) - \mathbf{a}(k))$

Sensitivity: $\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left[\frac{\partial \hat{F}}{\partial n_1^m} \quad \frac{\partial \hat{F}}{\partial n_2^m} \quad \ldots \quad \frac{\partial \hat{F}}{\partial n_{s^m}^m}\right]^T$

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 ζ (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $\rho < 1$, and the momentum coefficient γ (if it is used) is set to zero.

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 γ has been previously set to zero, it is reset to its original value.

3. If the squared error increases by less than ζ, then the weight update is accepted but the learning rate and the momentum coefficient are unchanged.

Association: $\mathbf{a} = hardlim(\mathrm{W}^0\mathrm{p}^0 + \mathbf{W}\mathbf{p} + b)$

An association is a link between the inputs and outputs of a network so that when a stimulus A is presented to the network, it will output a response B.

Associative Learning Rules:

Unsupervised Hebb Rule:

$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha\,\mathbf{a}(q)\mathbf{p}^T(q)$

Hebb with Decay:

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

Instar: $\mathbf{a} = hardlim(\mathbf{W}\mathbf{p} + b)$, $\mathbf{a} = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$

The instar is activated for ${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\| cos\theta \geq -b$

where θ 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: $\mathbf{a} = satlins(\mathbf{W}p)$

$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha\,\left(\mathbf{a}(q) - \mathbf{w}_j(q-1)\right)\mathrm{p}_j(q)$

Competitive Layer: $\mathbf{a} = compet(\mathbf{W}\mathbf{p}) = compet(\mathbf{n})$

Competitive Learning with the Kohonen Rule:

${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha\,\left(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)\right)$

$= (\mathbf{1} - \alpha)\,{}_{i^*}\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$

${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1)$, $i \neq i^*$ where i^* is the winning neuron.

Self-Organizing with the Kohonen Rule:

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\,\left(\mathbf{p}(q) - {}_i\mathbf{w}(q-1)\right)$

$= (\mathbf{1} - \alpha)\,{}_i\mathbf{w}(q-1) + \alpha\,\mathbf{p}(q)$, $i \in N_{i^*}(d)$

$N_i(d) = \{j, d_{i,j} \leq d\}$

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

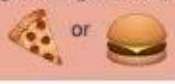
BASIC REGRESSION

LINEAR — linear_model.LinearRegression()

Lots of numerical data

LOGISTIC — linear_model.LogisticRegression()

Target variable is categorical



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 NEIH 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

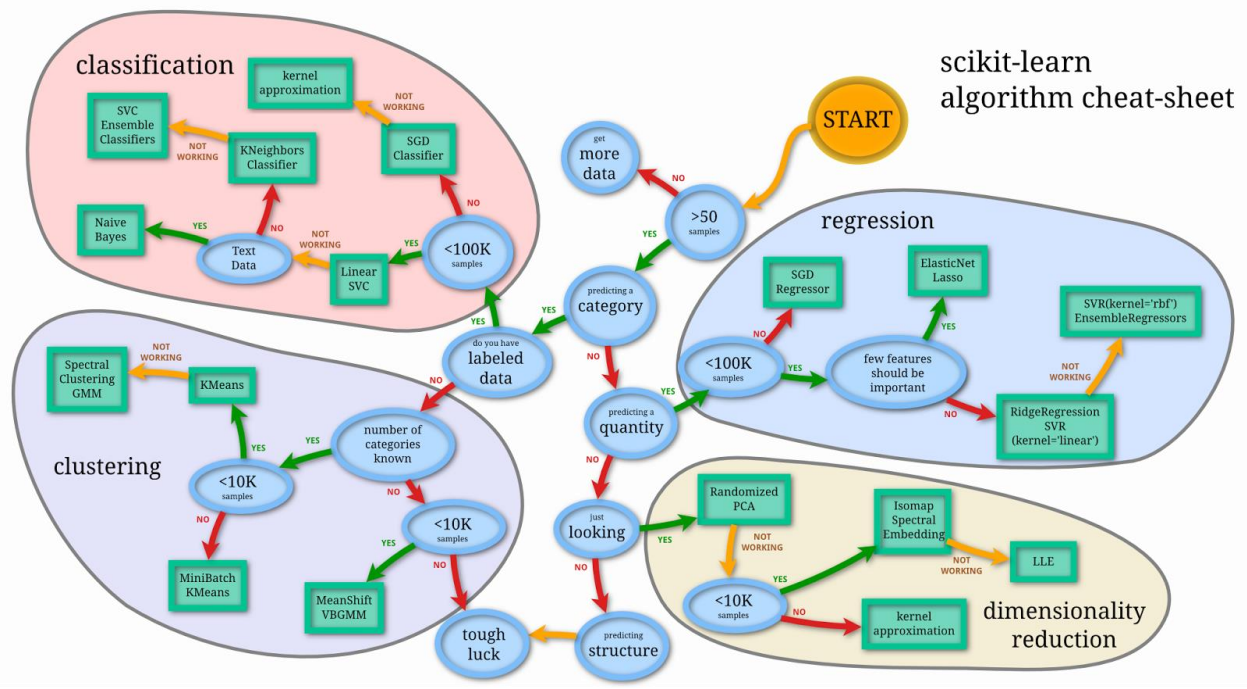
ACCURACY FUNCTION $(TP + TN) / (P + N)$

PRECISION FUNCTION $TP / (TP + FP)$

SPECIFICITY FUNCTION $TN / (FP + TN)$

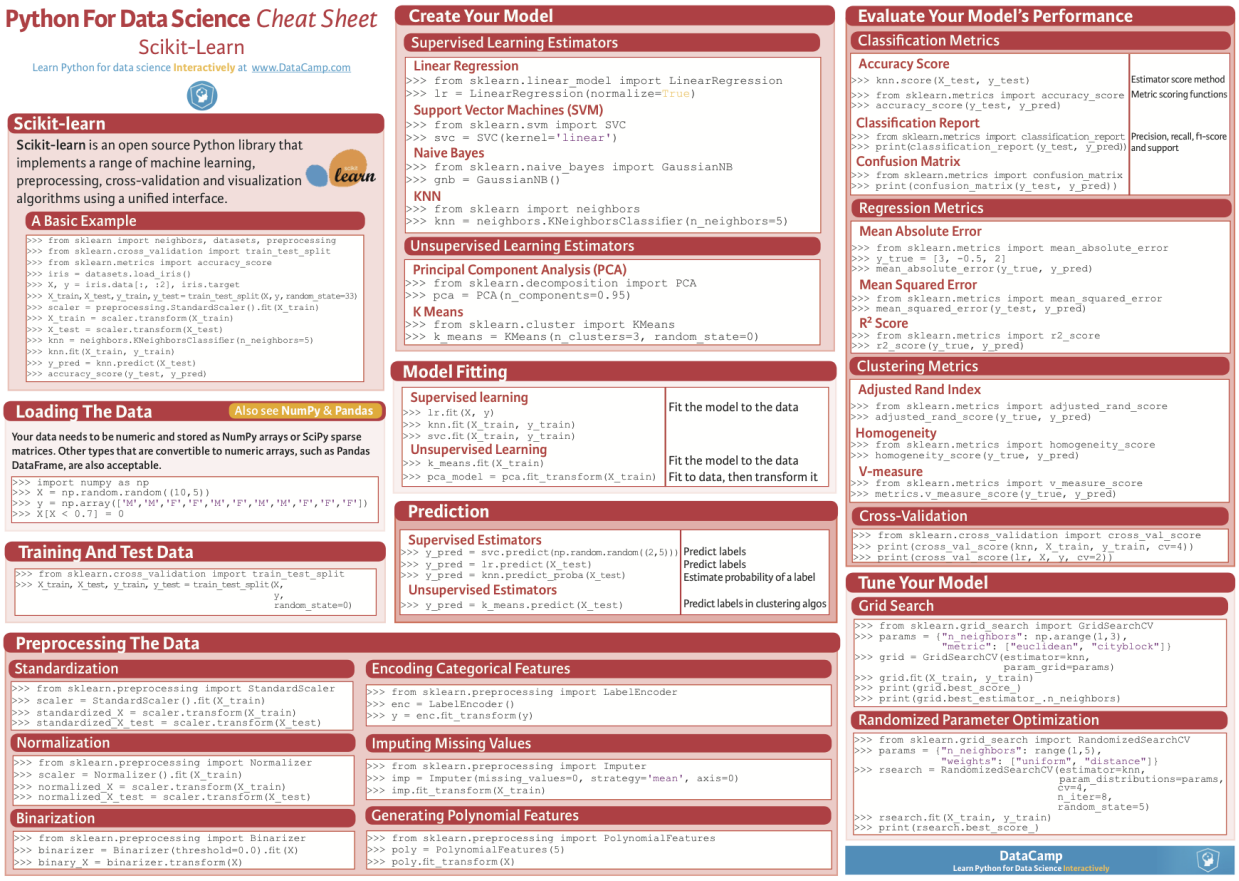
SENSITIVITY FUNCTION $TP / (TP + FN)$

@emilyinamillion made this



Scikit–Learn

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



Python for Data Science

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
>>> x-2
3
>>> x*2
10
>>> x**2
25
>>> x%2
1
>>> x/float(2)
2.5
```

Sum of two variables
Subtraction of two variables
Multiplication of two variables
Exponentiation of a variable
Remainder of a variable
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]
>>> my_list[-3]
```

Select item at index 1
Select 3rd last item

Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list

Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my_list[list][itemOfList]

List Operations

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

List Methods

```
>>> my_list.index(a)
>>> my_list.count(a)
>>> my_list.append('!')
>>> my_list.remove('!')
>>> del(my_list[0:1])
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> my_list.sort()
```

Get the index of an item
Count an item
Append an item at a time
Remove an item
Remove an item
Reverse the list
Append an item
Remove an item
Insert an item
Sort the list

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
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

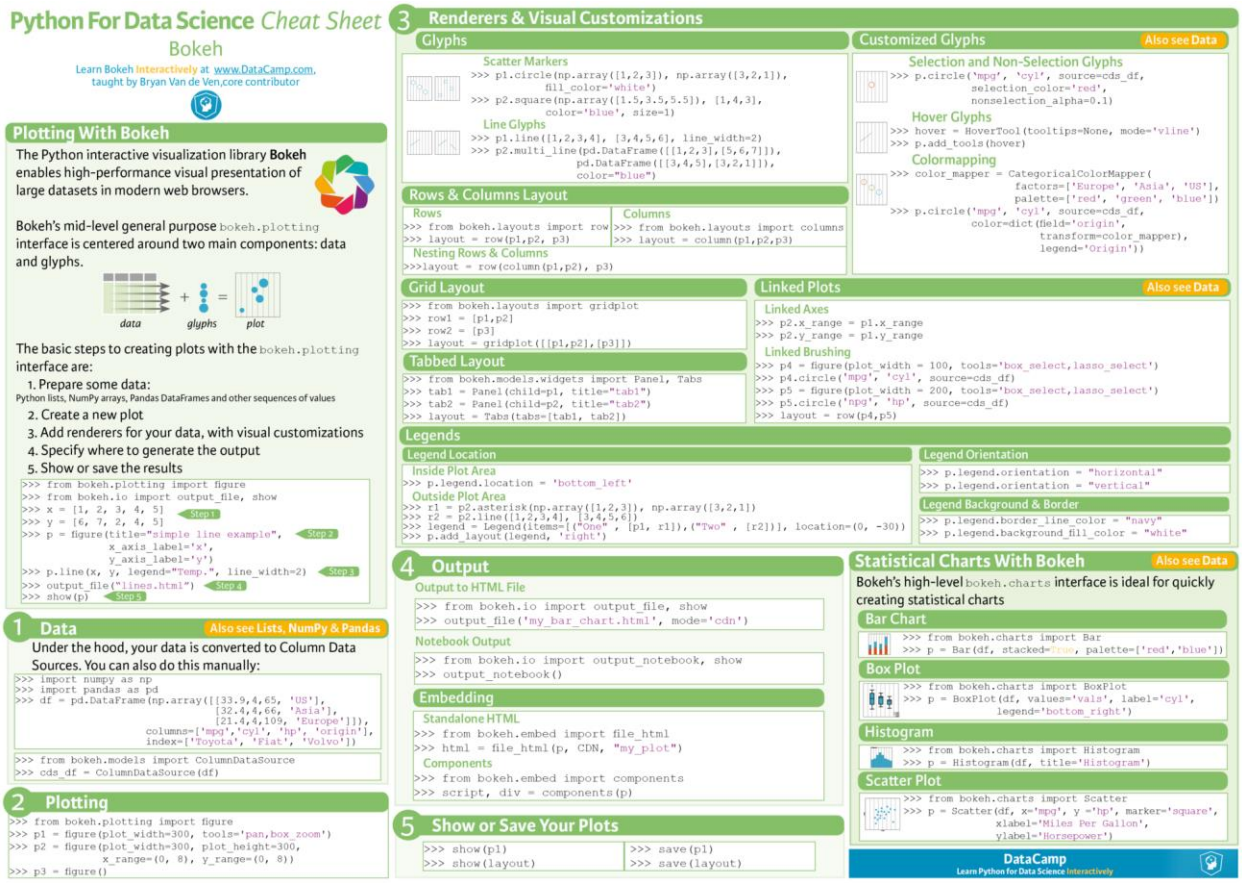
```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape
>>> np.append(other_array)
>>> np.insert(my_array, 1, 5)
>>> np.delete(my_array,[1])
>>> np.mean(my_array)
>>> np.median(my_array)
>>> my_array.corrcoef()
>>> np.std(my_array)
```

Get the dimensions of the array
Append items to an array
Insert items in an array
Delete items in an array
Mean of the array
Median of the array
Correlation coefficient
Standard deviation

DataCamp
Learn Python For Data Science Interactively



TensorFlow

In May 2017 Google announced the second-generation of the TPU, as well as the availability of the TPUs in Google Compute Engine.[12] The second-generation TPUs deliver up to 180 teraflops of performance, and when organized into clusters of 64 TPUs provide up to 11.5 petaflops.

ALTOROS

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

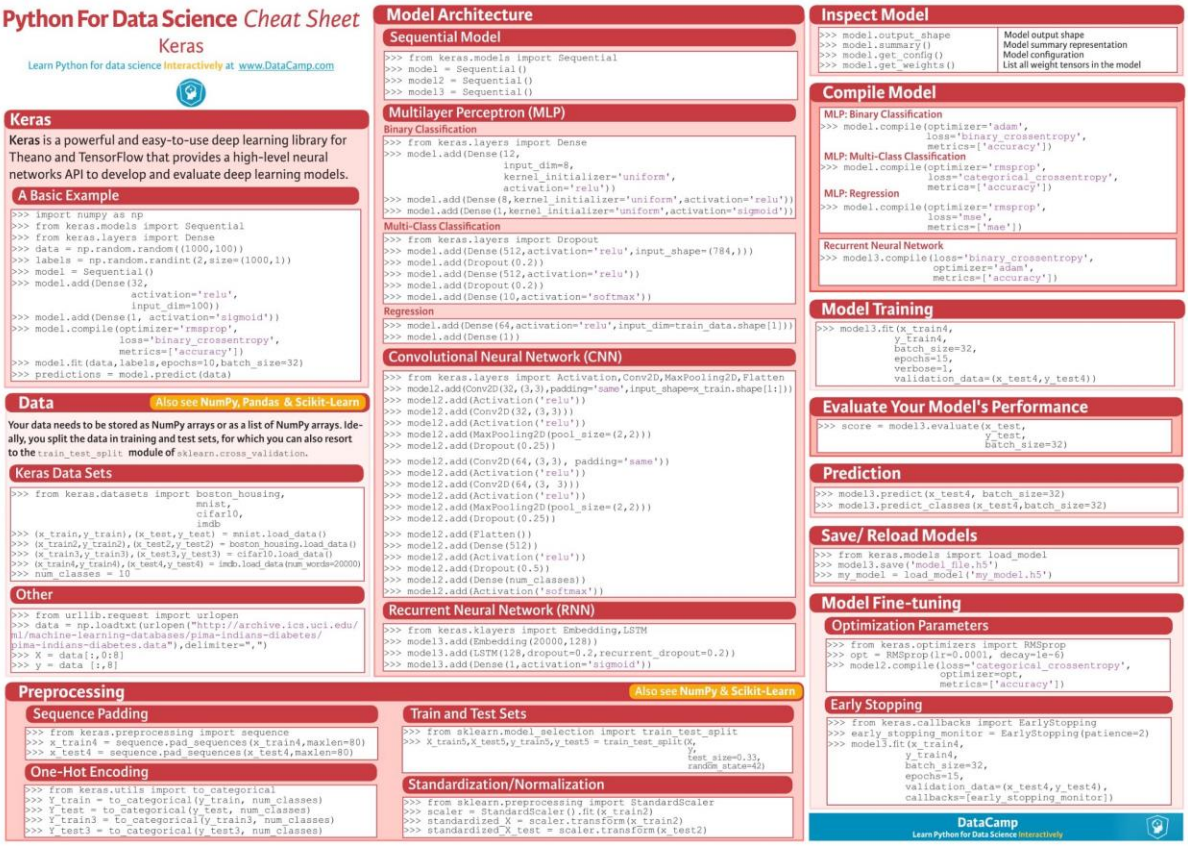
Each classifier and regressor have following fields

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,

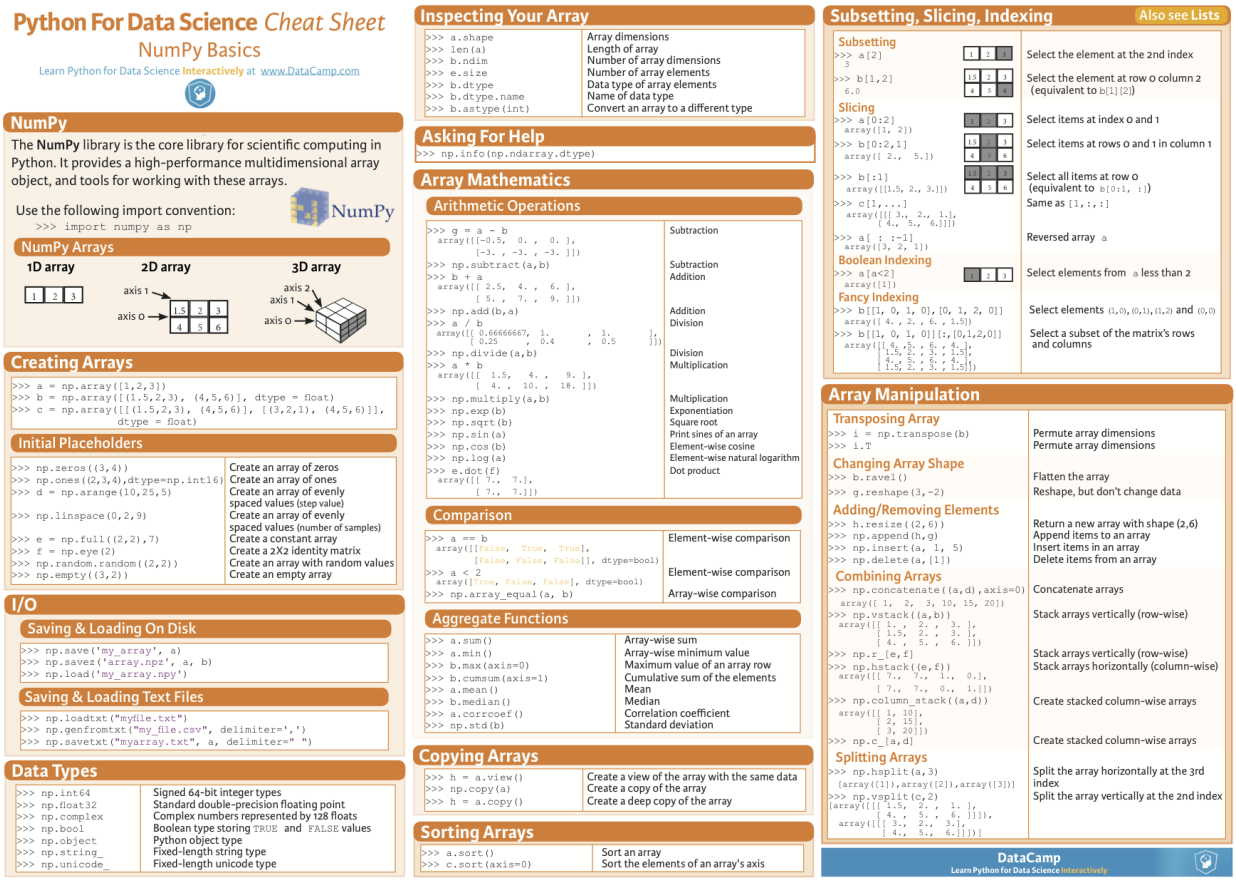
Keras

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



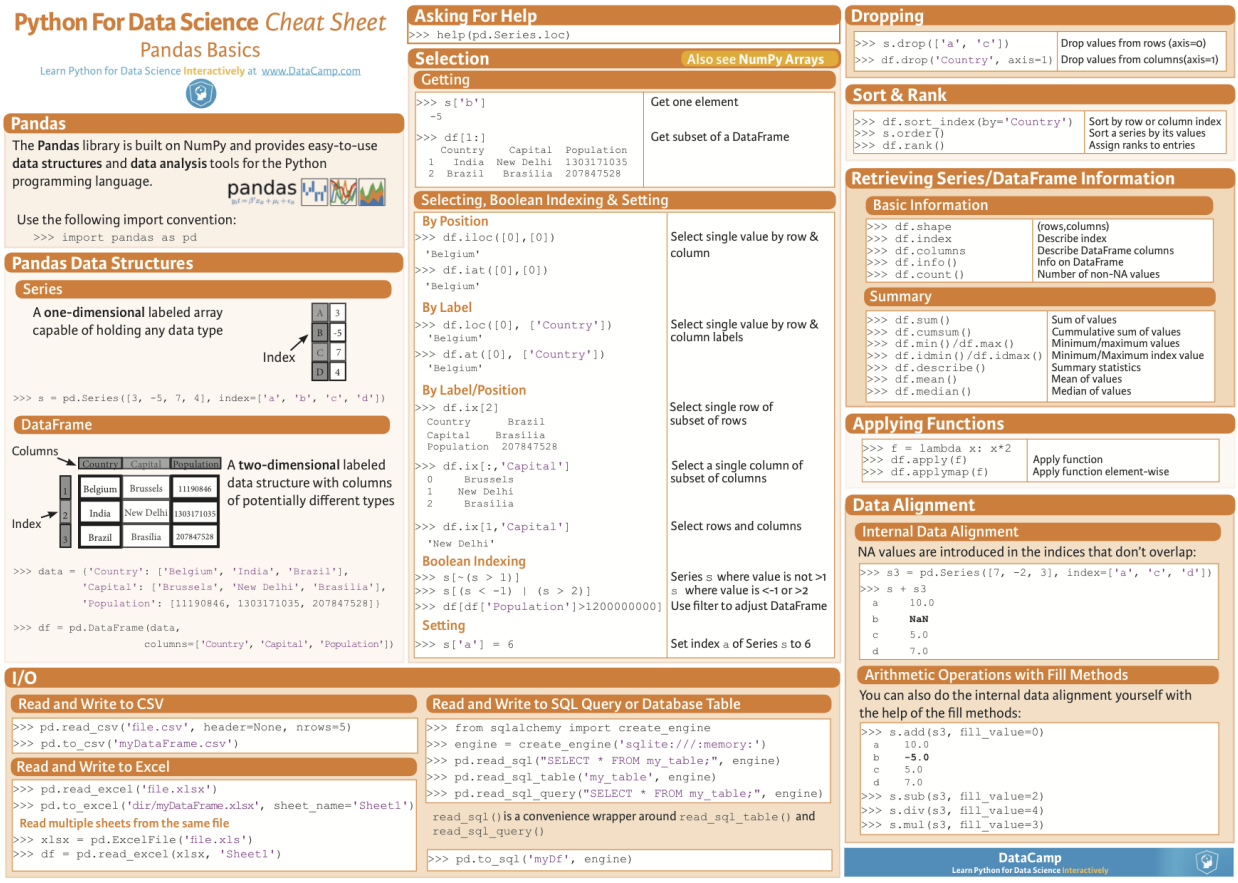
Numpy

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



Pandas

The name ‘Pandas’ is derived from the term “panel data”, an econometricsterm for multidimensional structured data sets.



Data Wrangling

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

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

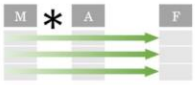
In a tidy data set:

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.



M * A

Syntax – Creating DataFrames

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

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

Specify values for each column.

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

Specify values for each row.

		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

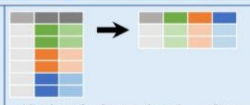
Method Chaining

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

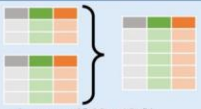
```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

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).

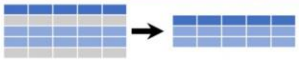
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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*.

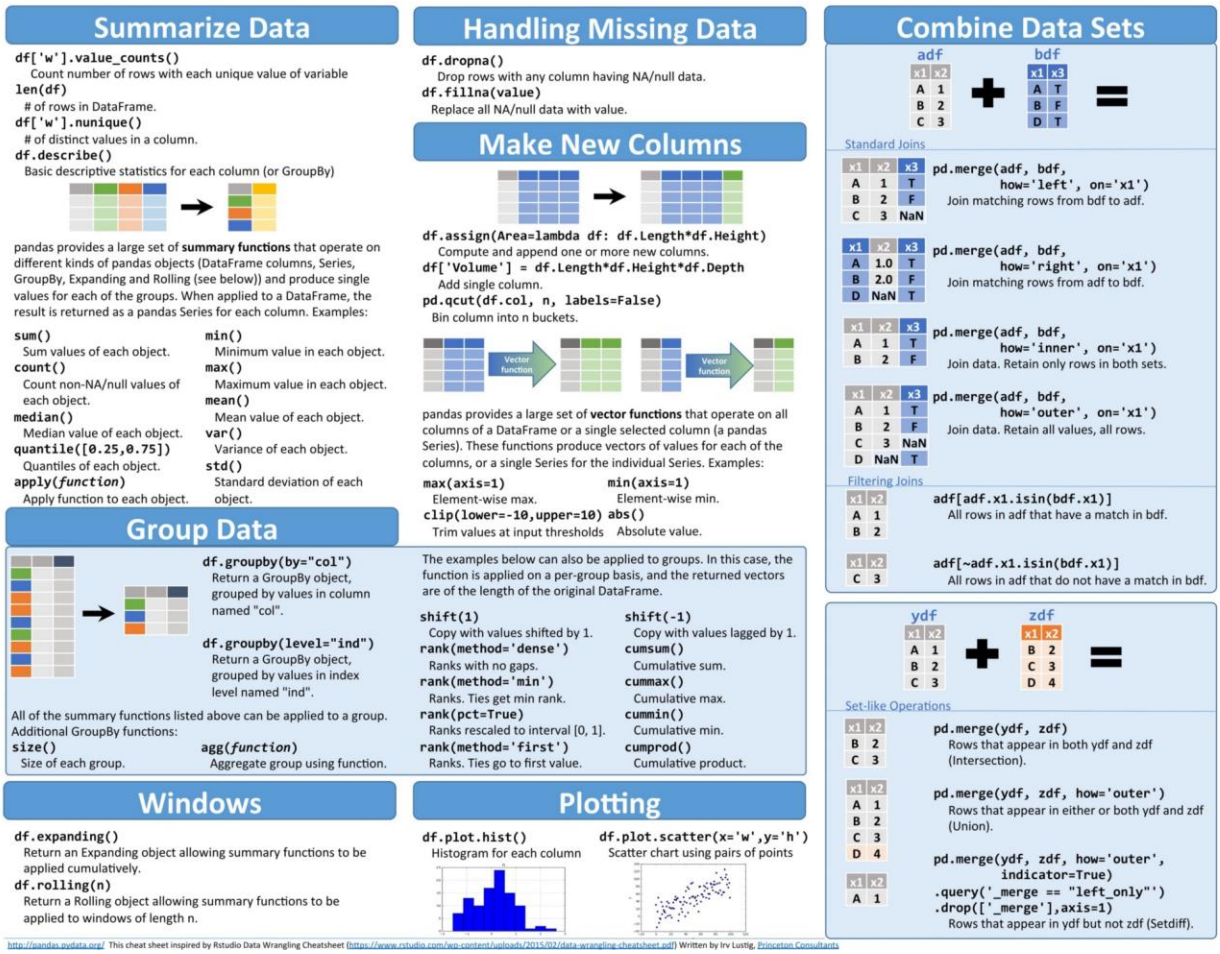
regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except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

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

Data Wrangling
with dplyr and tidyr
Cheat Sheet

RStudio

Tidy Data - A foundation for wrangling in R

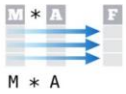
In a tidy data set:

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:

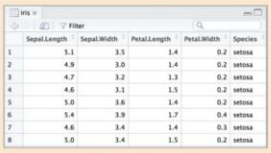
```
Source: local data frame [150 x 5]

   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).

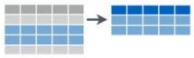
dplyr::arrange(mtcars, **desc(mpg)**)

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.

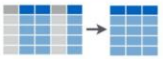
dplyr::**top_n(storms, 2, date)**

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.

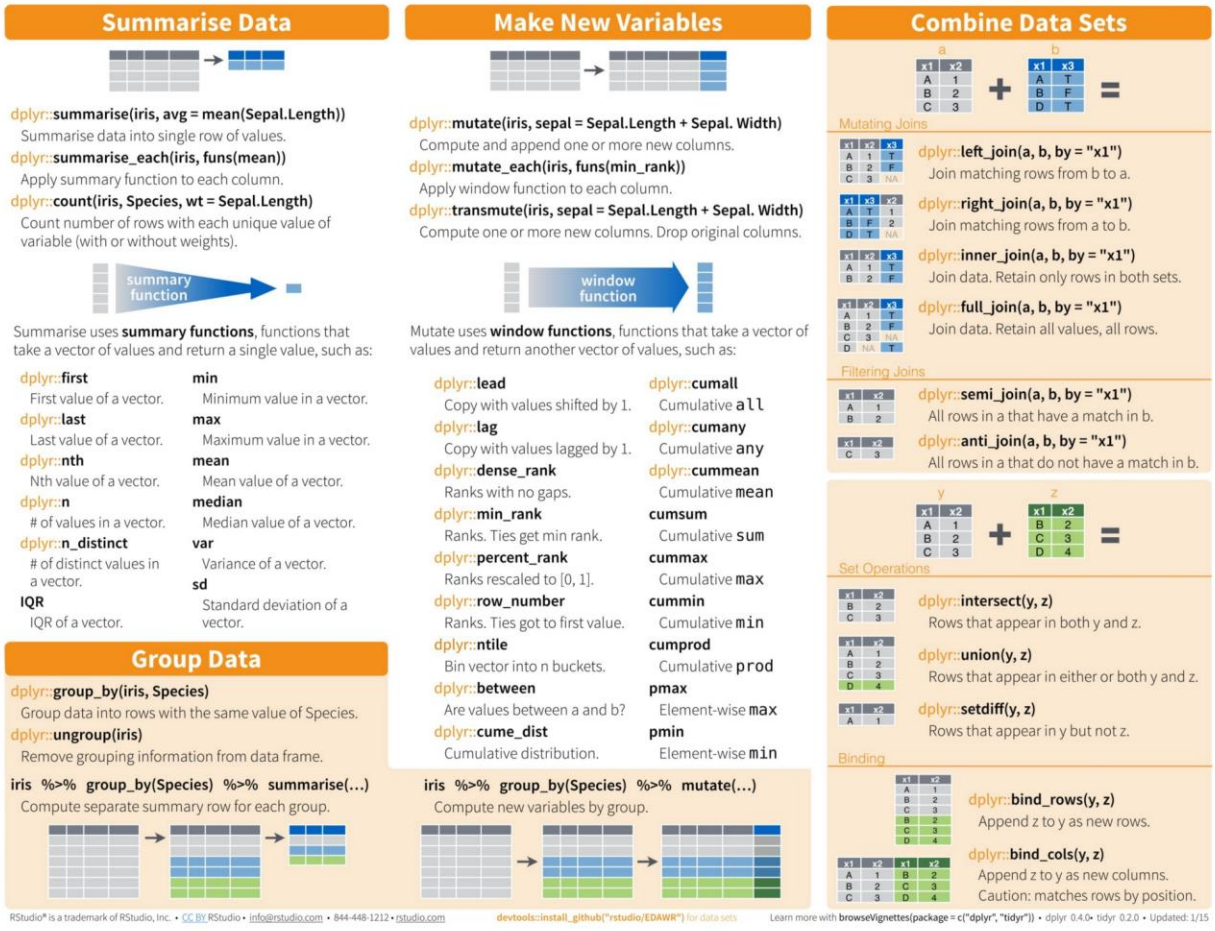
select(iris, starts_with("Sepal"))
Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)
Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)
Select all columns except Species.

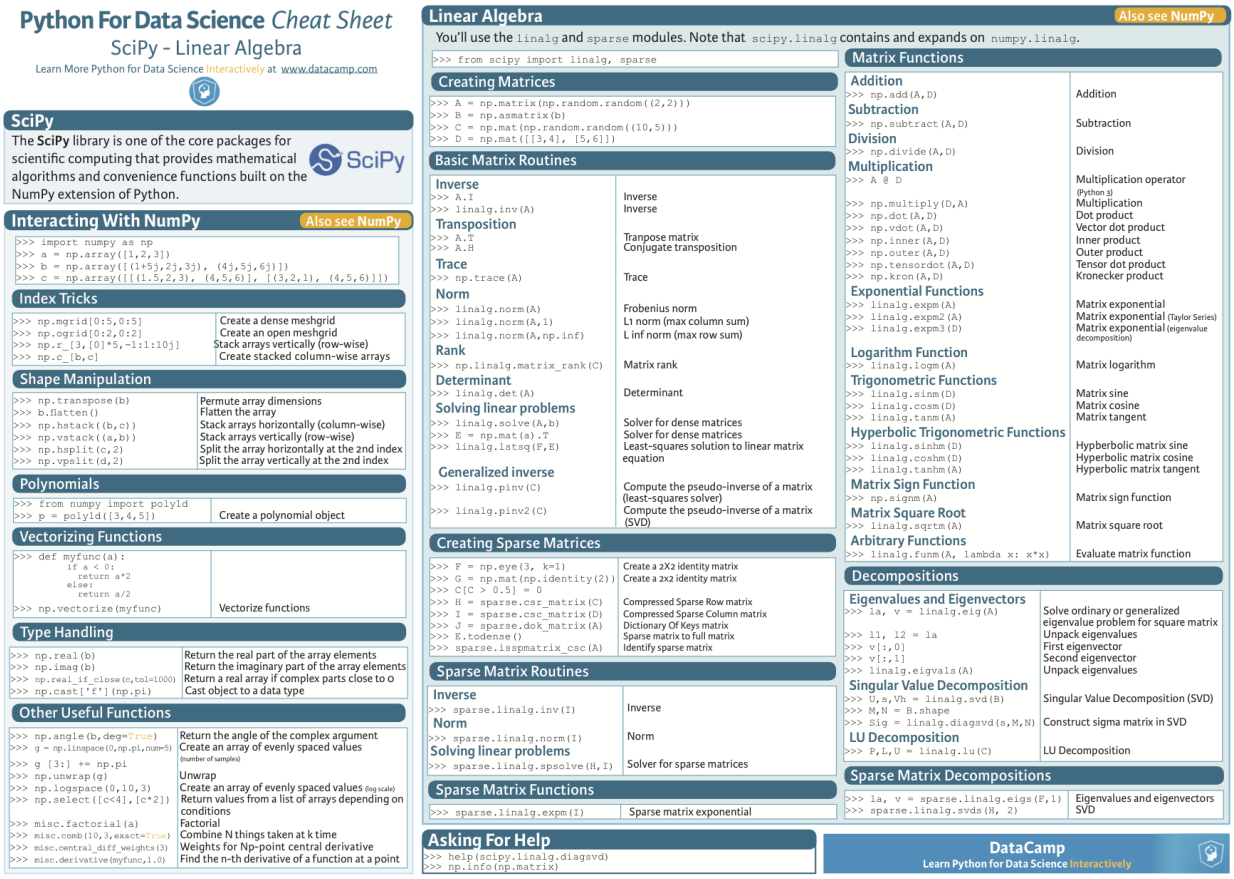
devtools::**install_github("rstudio/EDAWR")** for data sets

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



Scipy

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



Matplotlib

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

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

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

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

matplotlib

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))
```

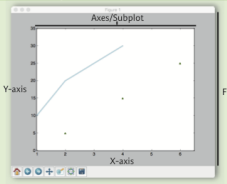
Axes

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
```

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',         Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                   No overlapping plot elements
```

Ticks

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

Subplot Spacing

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

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 o
```

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
```

2D Data or Images

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

```
>>> 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
```

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

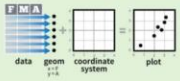
Data Visualization

Data Visualization with ggplot2 Cheat Sheet

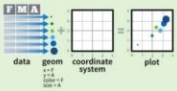
RStudio

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

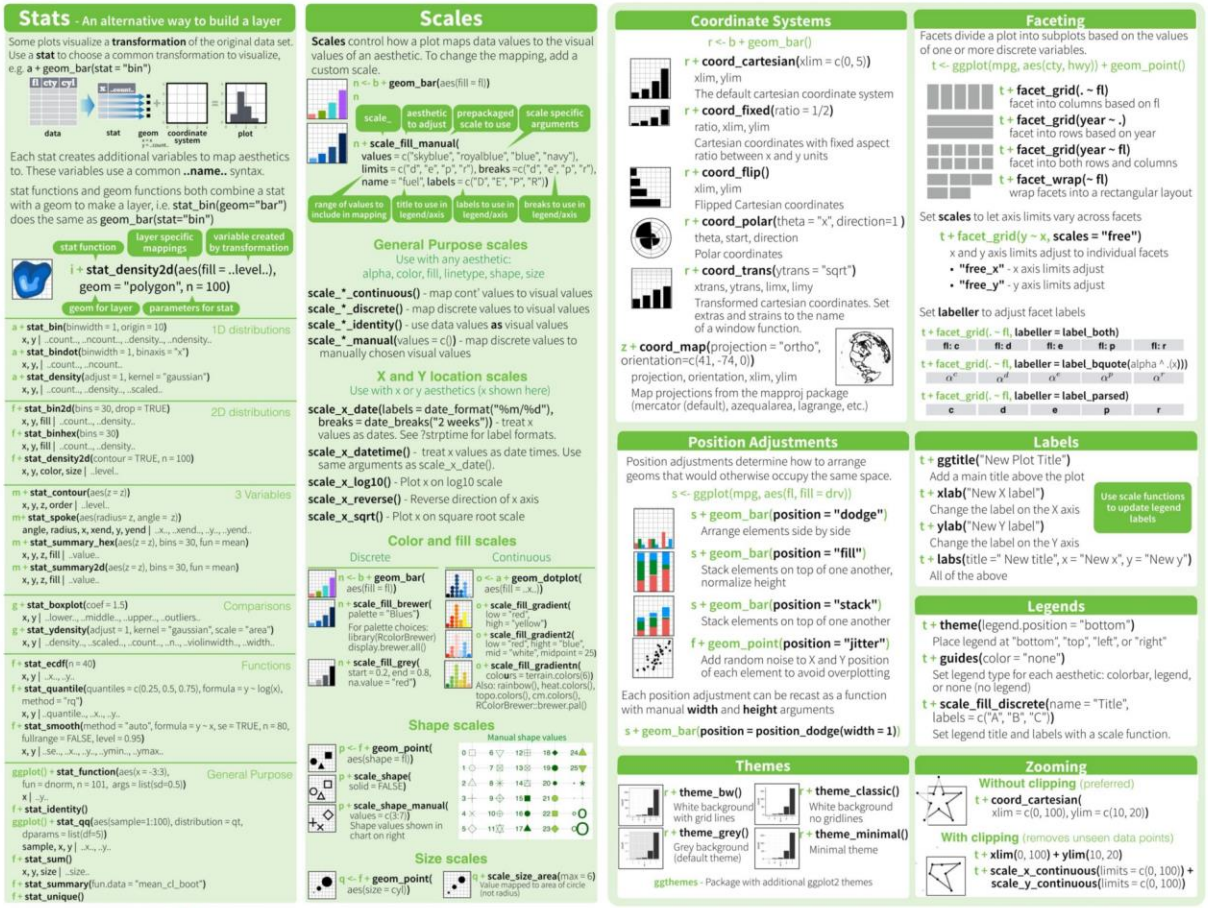
Three Variables

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


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")
data
stat
geom
coordinate system
plot
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
2D distributions
3 Variables
Comparisons
Functions
General Purpose
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))
scale_
aesthetic to adjust
prepackaged scale to use
scale specific arguments
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
Continuous
Shape scales
Manual shape values
Size scales
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_grey()
Grey background (default theme)
r + theme_classic()
White background no gridlines
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)))
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))
RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com
Learn more at docs.ggplot2.org • ggplot2 0.9.3.1 • Updated: 3/15

PySpark

Python For Data Science *Cheat Sheet*

PySpark Basics

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

Spark

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

Initializing Spark

SparkContext

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

Inspect SparkContext

```
>>> 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 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()                     Count RDD instances
 3
>>> rdd.countByKey()                Count RDD instances by key
 defaultdict(<type 'int'>,{'a':2,'b':1})
>>> rdd.countByValue()              Count RDD instances by value
 defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})
>>> rdd.collectAsMap()              Return (key,value) pairs as a dictionary
 {'a': 2,'b': 2}
>>> rdd3.sum()                      Sum of RDD elements
 4950
>>> sc.parallelize([]).isEmpty()    Check whether RDD is empty
 True
```

Summary

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

Applying Functions

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

Selecting Data

Getting

```
>>> rdd.collect()                   Return a list with all RDD elements
  [('a', 7), ('a', 2), ('b', 2)]
>>> rdd.take(2)                     Take first 2 RDD elements
  [('a', 7), ('a', 2)]
>>> rdd.first()                     Take first RDD element
  ('a', 7)
>>> rdd.top(2)                      Take top 2 RDD elements
  [('b', 2), ('a', 7)]
```

Sampling

```
>>> rdd3.sample(False, 0.15, 81).collect()   Return sampled subset of rdd3
  [3,4,27,31,40,41,42,43,60,76,79,80,86,97]
```

Filtering

```
>>> rdd.filter(lambda x: "a" in x)  Filter the RDD
       .collect()
  [('a',7),('a',2)]
>>> rdd5.distinct().collect()       Return distinct RDD values
  ['a',2,'b',7]
>>> rdd.keys().collect()            Return (key,value) RDD's keys
  ['a', 'a', 'b']
```

Iterating

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

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)     Merge the rdd values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd.reduce(lambda a, b: a + b)        Merge the rdd values
  ('a',7,'a',2,'b',2)
```

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)         Return RDD of grouped values
        .mapValues(list)
        .collect()
>>> rdd.groupByKey()                      Group rdd by key
       .mapValues(list)
       .collect()
  [('a',[7,2]),('b',[2])]
```

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))
>>> rdd3.aggregate((0,0),seqOp,combOp)    Aggregate RDD elements of each partition and then the results
  (4950,100)
>>> rdd.aggregateByKey((0,0),seqop,combop)   Aggregate values of each RDD key
       .collect()
  [('a',(9,2)), ('b',(2,1))]
>>> rdd3.fold(0,add)                      Aggregate the elements of each partition, and then the results
  4950
>>> rdd.foldByKey(0, add)                 Merge the values for each key
       .collect()
  [('a',9),('b',2)]
>>> rdd3.keyBy(lambda x: x+x)             Create tuples of RDD elements by applying a function
       .collect()
```

Mathematical Operations

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

Sort

```
>>> rdd2.sortBy(lambda x: x[1])           Sort RDD by given function
       .collect()
  [('d',1),('b',1),('a',2)]
>>> rdd2.sortByKey()                      Sort (key, value) RDD by key
       .collect()
  [('a',2),('b',1),('d',1)]
```

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
```