

A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

○ Backfed Input Cell

● Input Cell

△ Noisy Input Cell

● Hidden Cell

○ Probabilistic Hidden Cell

△ Spiking Hidden Cell

● Output Cell

○ Match Input Output Cell

● Recurrent Cell

○ Memory Cell

△ Different Memory Cell

● Kernel

○ Convolution or Pool

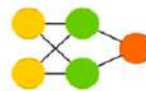
Perceptron (P)



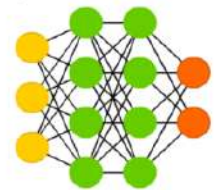
Feed Forward (FF)



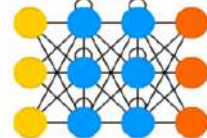
Radial Basis Network (RBF)



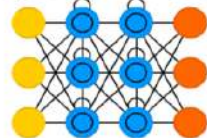
Deep Feed Forward (DFF)



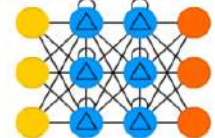
Recurrent Neural Network (RNN)



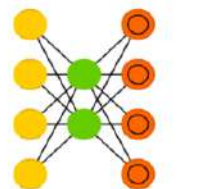
Long / Short Term Memory (LSTM)



Gated Recurrent Unit (GRU)



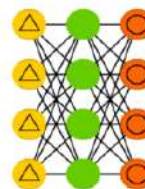
Auto Encoder (AE)



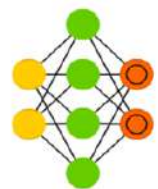
Variational AE (VAE)



Denoising AE (DAE)



Sparse AE (SAE)



Markov Chain (MC)



Hopfield Network (HN)



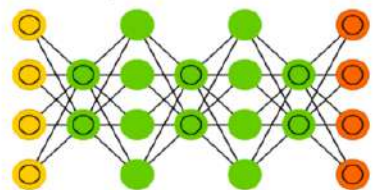
Boltzmann Machine (BM)



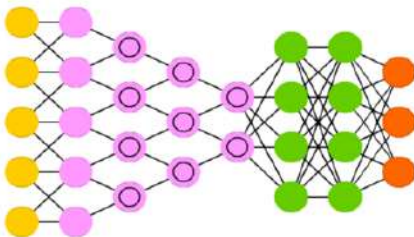
Restricted BM (RBM)



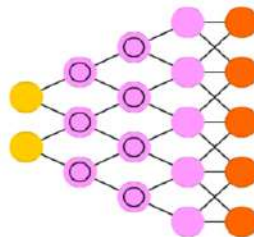
Deep Belief Network (DBN)



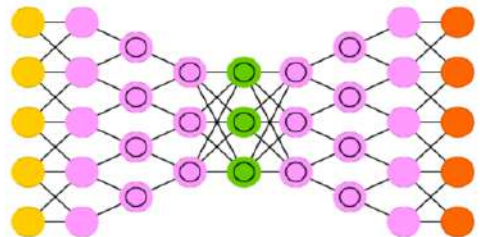
Deep Convolutional Network (DCN)



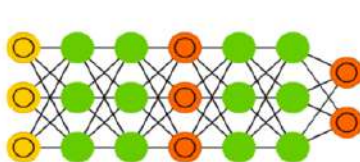
Deconvolutional Network (DN)



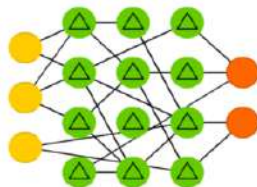
Deep Convolutional Inverse Graphics Network (DCIGN)



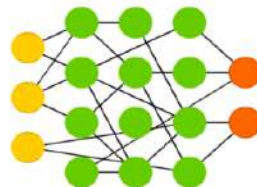
Generative Adversarial Network (GAN)



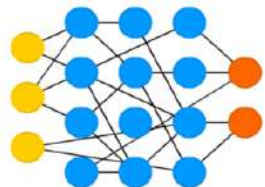
Liquid State Machine (LSM)



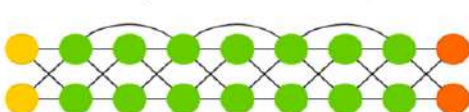
Extreme Learning Machine (ELM)



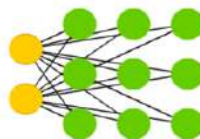
Echo State Network (ESN)



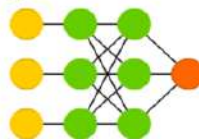
Deep Residual Network (DRN)



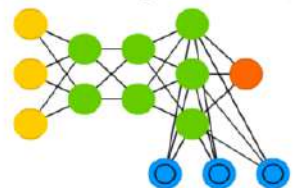
Kohonen Network (KN)



Support Vector Machine (SVM)

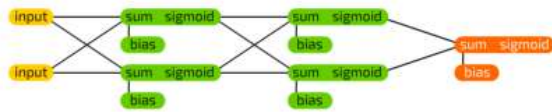


Neural Turing Machine (NTM)

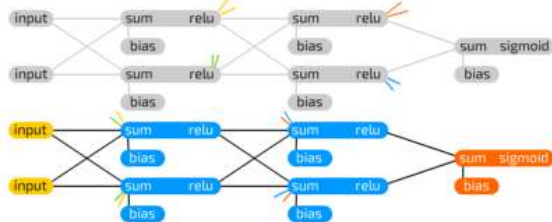
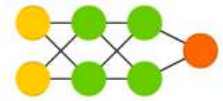


An informative chart to build Neural Network Graphs

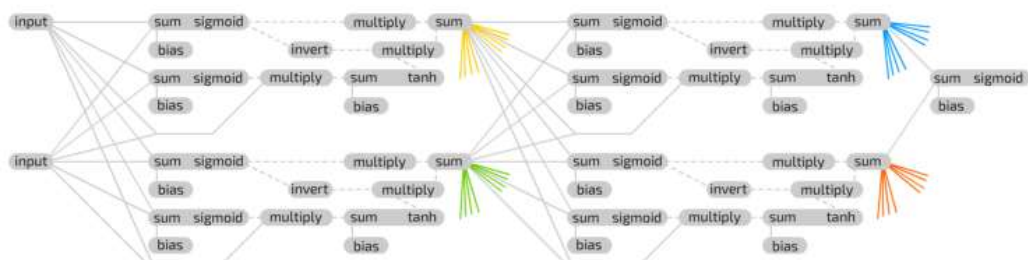
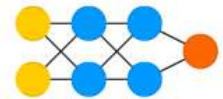
©2016 Fjodor van Veen - asimovinstitute.org



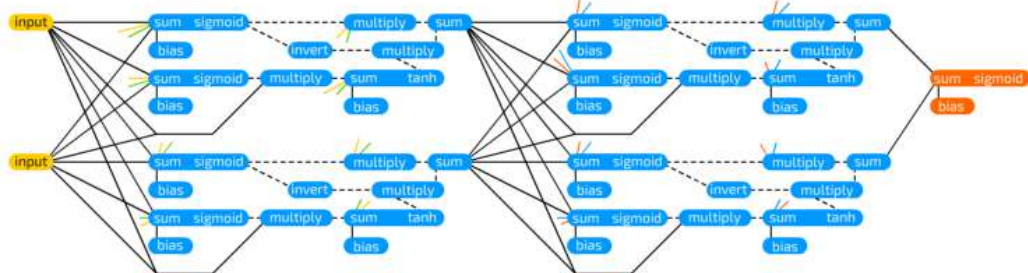
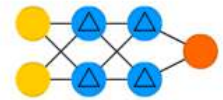
Deep Feed Forward Example



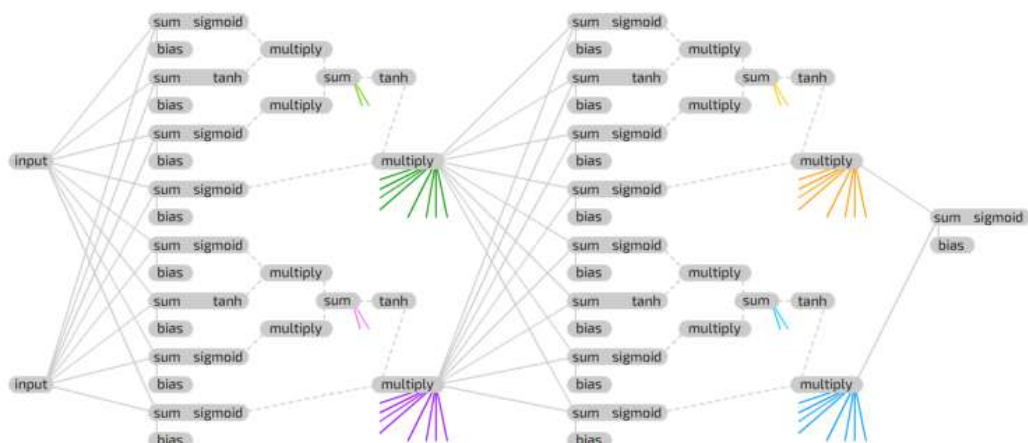
Deep Recurrent Example
(previous iteration)



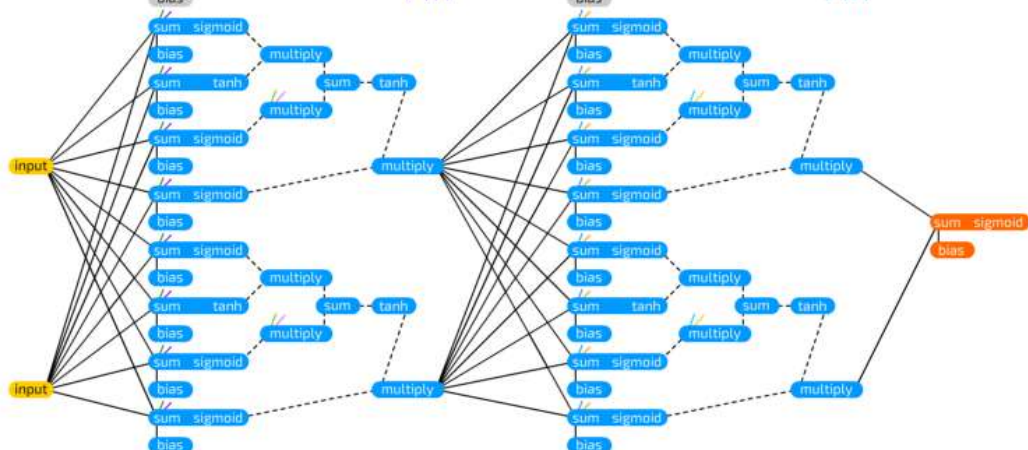
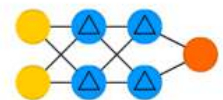
Deep GRU Example
(previous iteration)



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

Spanning a Space:

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

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

1. $(x, x) = (x, x)$
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 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 $y_1, y_2, \dots, y_n$. From these vectors we will obtain n orthogonal vectors $v_1, v_2, \dots, v_n$.

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

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

Vector Expansions:

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

for orthogonal vectors, $x_i = \frac{(v_i, x)}{(v_i, v_i)}$

Reciprocal Basis Vectors:

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

To compute the reciprocal basis vectors: set $B = [v_1 \ v_2 \ \dots \ v_n]$,

$R = [r_1 \ r_2 \ \dots \ r_n]$, $R^T = B^{-1}$ In matrix form: $x^v = B^{-1} x^s$

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 $\{v_1, v_2, \dots, v_n\}$ be a basis for vector space X , and let $\{u_1, u_2, \dots, u_n\}$ be a basis for vector space Y . Let A be a linear transformation with domain X and range Y : $A(x) = y$

The coefficients of the matrix representation are obtained from

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

Change of Basis: $B_t = [t_1 \ t_2 \ \dots \ t_n]$, $B_w = [w_1 \ w_2 \ \dots \ w_n]$
 $A' = [B_w^{-1} A B_t]$

Eigenvalues & Eigenvectors: $Az = \lambda z$, $[[A - \lambda I]] = 0$

Diagonalization: $B = [z_1 \ z_2 \ \dots \ z_n]$,

where $\{z_1, z_2, \dots, z_n\}$ are the eigenvectors of a square matrix A ,
 $[B^{-1} A B] = \text{diag}([\lambda_1 \ \lambda_2 \ \dots \ \lambda_n])$

Perceptron Architecture:

$$a = \text{hardlim}(Wp + b), \quad W = [{}_1w^T \ {}_2w^T \ \dots \ {}_sw^T]^T, \\ a_i = \text{hardlim}(n_i) = \text{hardlim}({}_i w^T p + b_i)$$

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

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

Perceptron Learning Rule

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

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: $a = \text{purelin}(Wp)$

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

$$W = t_1 p_1^T + t_2 p_2^T + \dots + t_q p_q^T \\ W = [t_1 \ t_2 \ \dots \ t_q] \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_q^T \end{bmatrix} = TP^T$$

Pseudoinverse Rule: $W = TP^+$

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

Variations of Hebbian Learning:

Filtered Learning (Ch.14): $W^{new} = (1 - \gamma)W^{old} + \alpha t_q p_q^T$

Delta Rule (Ch.10): $W^{new} = W^{old} + \alpha(t_q - a_q)p_q^T$

Unsupervised Hebb (Ch.13): $W^{new} = W^{old} + \alpha a_q p_q^T$

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

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

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

$$\begin{bmatrix} \frac{\partial}{\partial x_1} \frac{\partial}{\partial x_1} F(x) & \frac{\partial}{\partial x_1} \frac{\partial}{\partial x_2} F(x) & \dots & \frac{\partial}{\partial x_1} \frac{\partial}{\partial x_n} F(x) \\ \frac{\partial}{\partial x_2} \frac{\partial}{\partial x_1} F(x) & \frac{\partial}{\partial x_2} \frac{\partial}{\partial x_2} F(x) & \dots & \frac{\partial}{\partial x_2} \frac{\partial}{\partial x_n} F(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_n} \frac{\partial}{\partial x_1} F(x) & \frac{\partial}{\partial x_n} \frac{\partial}{\partial x_2} F(x) & \dots & \frac{\partial}{\partial x_n} \frac{\partial}{\partial x_n} F(x) \end{bmatrix}$$

Directional Derivatives:

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

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: $\nabla F(x)|_{x=x^*} = 0$ (Stationary Points)

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

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

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

General Minimization Algorithm: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ 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) $\alpha < \frac{2}{\lambda_{\max}}$ $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ Eigenvalues of Hessian matrix A Learning Rate to Minimize Along the Line: $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ (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: $\mathbf{a} = \text{purelin}(\mathbf{W}\mathbf{p} + \mathbf{b})$ 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 $\mathbf{x} = \begin{bmatrix} 1 \\ \mathbf{w} \end{bmatrix}$ and $\mathbf{z} = \begin{bmatrix} 1 \\ \mathbf{p} \end{bmatrix}$ 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 R Adaptive Filter ADALINE: $a(k) = \text{purelin}(\mathbf{W}\mathbf{p}(k) + \mathbf{b}) = \sum_{i=1}^R \mathbf{w}_{i,1} y(k-i+1) + b$	*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: $\mathbf{a} = \text{hardlim}(\mathbf{W}^0 \mathbf{p}^0 + \mathbf{W} \mathbf{p} + \mathbf{b})$ 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: $\mathbf{a} = \text{hardlim}(\mathbf{W} \mathbf{p} + \mathbf{b})$, $\mathbf{a} = \text{hardlim}(\mathbf{I} \mathbf{w}^T \mathbf{p} + \mathbf{b})$ The instar is activated for $\mathbf{I} \mathbf{w}^T \mathbf{p} = \ \mathbf{I} \mathbf{w}\ \ \mathbf{p}\ \cos \theta \geq -b$ where θ is the angle between $\mathbf{p}$ and $\mathbf{I} \mathbf{w}$. Instar Rule: $\mathbf{I} \mathbf{w}(q) = \mathbf{I} \mathbf{w}(q-1) + \alpha a_i(q) (\mathbf{p}(q) - \mathbf{I} \mathbf{w}(q-1))$ $\mathbf{I} \mathbf{w}(q) = (1-\alpha) \mathbf{I} \mathbf{w}(q-1) + \alpha \mathbf{p}(q), \text{ if } (a_i(q) = 1)$ Kohonen Rule: $\mathbf{I} \mathbf{w}(q) = \mathbf{I} \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - \mathbf{I} \mathbf{w}(q-1)) \text{ for } i \in X(q)$ Outstar Rule: $\mathbf{a} = \text{satlins}(\mathbf{W} \mathbf{p})$ $\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha (\mathbf{a}(q) - \mathbf{w}_j(q-1)) p_j(q)$ Competitive Layer: $\mathbf{a} = \text{compet}(\mathbf{W} \mathbf{p}) = \text{compet}(\mathbf{n})$ Competitive Learning with the Kohonen Rule: $\mathbf{I} \mathbf{w}(q) = \mathbf{I} \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - \mathbf{I} \mathbf{w}(q-1))$ $= (1-\alpha) \mathbf{I} \mathbf{w}(q-1) + \alpha \mathbf{p}(q)$ $\mathbf{I} \mathbf{w}(q) = \mathbf{I} \mathbf{w}(q-1), \text{ if } i \neq i^* \text{ where } i^* \text{ is the winning neuron.}$ Self-Organizing with the Kohonen Rule: $\mathbf{I} \mathbf{w}(q) = \mathbf{I} \mathbf{w}(q-1) + \alpha (\mathbf{p}(q) - \mathbf{I} \mathbf{w}(q-1))$ $= (1-\alpha) \mathbf{I} \mathbf{w}(q-1) + \alpha \mathbf{p}(q), \text{ if } i \in N_i(d)$ $N_i(d) = \{j, d_{i,j} \leq d\}$ LVO Network: ($w_{k,i}^2 = 1$) $\Rightarrow$ subclass i is a part of class k $\mathbf{n}_i^1 = -\ \mathbf{I} \mathbf{w}^1 - \mathbf{p}\ , \mathbf{a}^1 = \text{compet}(\mathbf{n}^1), \mathbf{a}^2 = \mathbf{W}^2 \mathbf{a}^1$ LVO Network Learning with the Kohonen Rule: $\mathbf{I} \mathbf{w}^1(q) = \mathbf{I} \mathbf{w}^1(q-1) + \alpha (\mathbf{p}(q) - \mathbf{I} \mathbf{w}^1(q-1)),$ $\text{if } a_{k^*}^2 = t_{k^*} = 1$ $\mathbf{I} \mathbf{w}^1(q) = \mathbf{I} \mathbf{w}^1(q-1) - \alpha (\mathbf{p}(q) - \mathbf{I} \mathbf{w}^1(q-1)),$ $\text{if } a_{k^*}^2 = 1 \neq t_{k^*} = 0$
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: $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$, postlin: $a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$ 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: $a = \begin{cases} -1 & n < 0 \\ -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$ Delay: $a(t) = u(t-1)$, Integrator: $a(t) = \int_0^t u(\tau) d\tau + a(0)$	HINT: $\text{diag}([1 \ 2 \ 3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

MACHINE LEARNING IN EMOJI

SUPERVISED

UNSUPERVISED

REINFORCEMENT

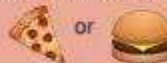
SUPERVISED human builds model based on input / output
UNSUPERVISED human input, machine output
 human utilizes if satisfactory
REINFORCEMENT human input, machine output
 human reward/punish, cycle continues

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-DISTRIBUTION STOCHASTIC NEIGH 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 $TN / (FP + TN)$

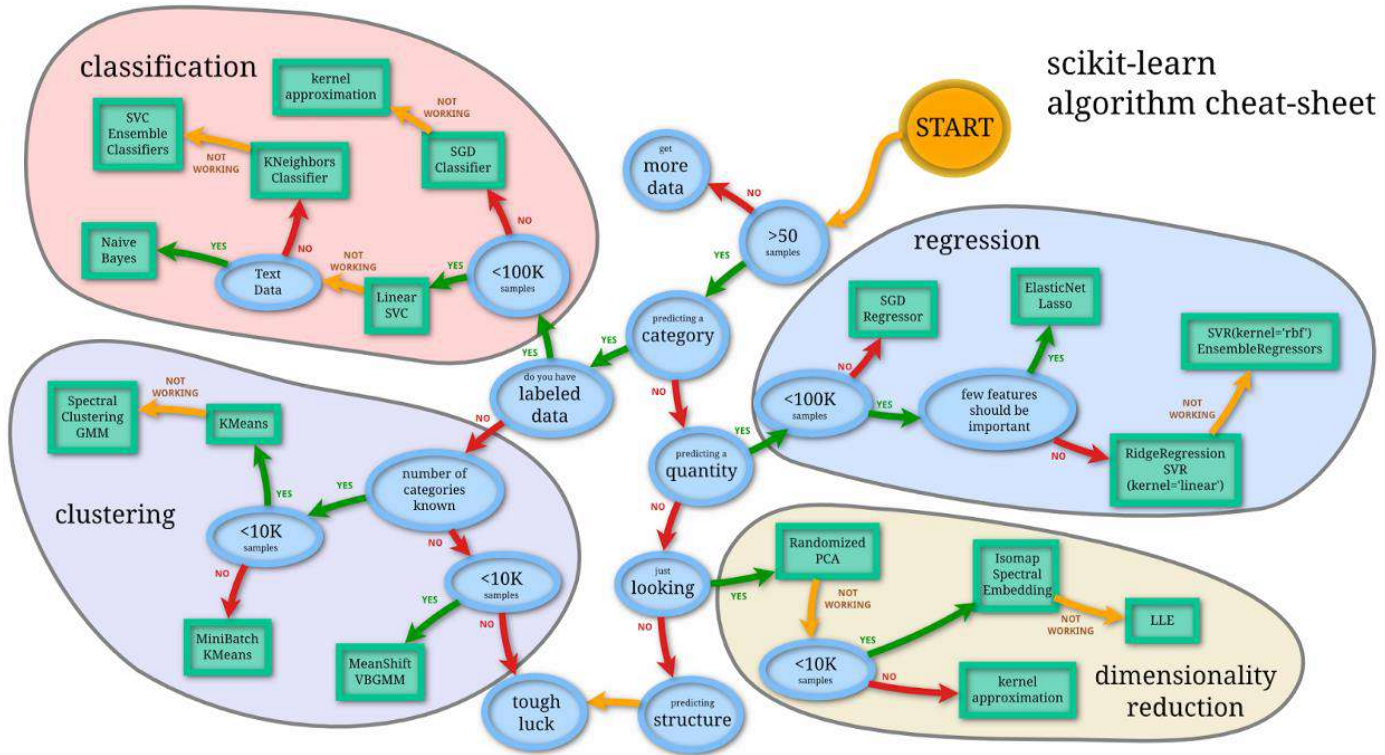


SENSITIVITY FUNCTION $TP / (TP + FN)$



@emiliyamillion made this

scikit-learn algorithm cheat-sheet



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[:, :4], iris.target
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
>>> 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 DataFrames, are also acceptable.

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

Training And Test Data

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

Preprocessing The Data

Standardization

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

Normalization

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

Binartization

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

Create Your Model

Supervised Learning Estimators

Linear Regression

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

Support Vector Machines (SVM)

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

Naive Bayes

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

KNN

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

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

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

K Means

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

Model Fitting

Supervised learning	Fit the model to the data
<code>lr.fit(X, y)</code>	
<code>knn.fit(X_train, y_train)</code>	
<code>svc.fit(X_train, y_train)</code>	
Unsupervised Learning	Fit the model to the data
<code>k_means.fit(X_train)</code>	
<code>pca_model = pca.fit_transform(X_train)</code>	Fit to data, then transform it

Prediction

Supervised Estimators	Predict labels
<code>y_pred = svc.predict(np.random.random((2,5)))</code>	
<code>y_pred = lr.predict(X_test)</code>	
<code>y_pred = knn.predict_proba(X_test)</code>	Estimate probability of a label
Unsupervised Estimators	Predict labels in clustering algos
<code>y_pred = k_means.predict(X_test)</code>	

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)
```

Classification Report

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

Confusion Matrix

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

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

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_true, y_pred)
```

R² 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(1,9),
             "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(1,9),
             "metric": ["uniform", "distance"]}
>>> research = RandomizedSearchCV(estimator=knn,
                                  param_distributions=params,
                                  cv=5,
                                  n_iter=5,
                                  random_state=5)
>>> research.fit(X_train, y_train)
>>> print(research.best_score_)
```

DataCamp

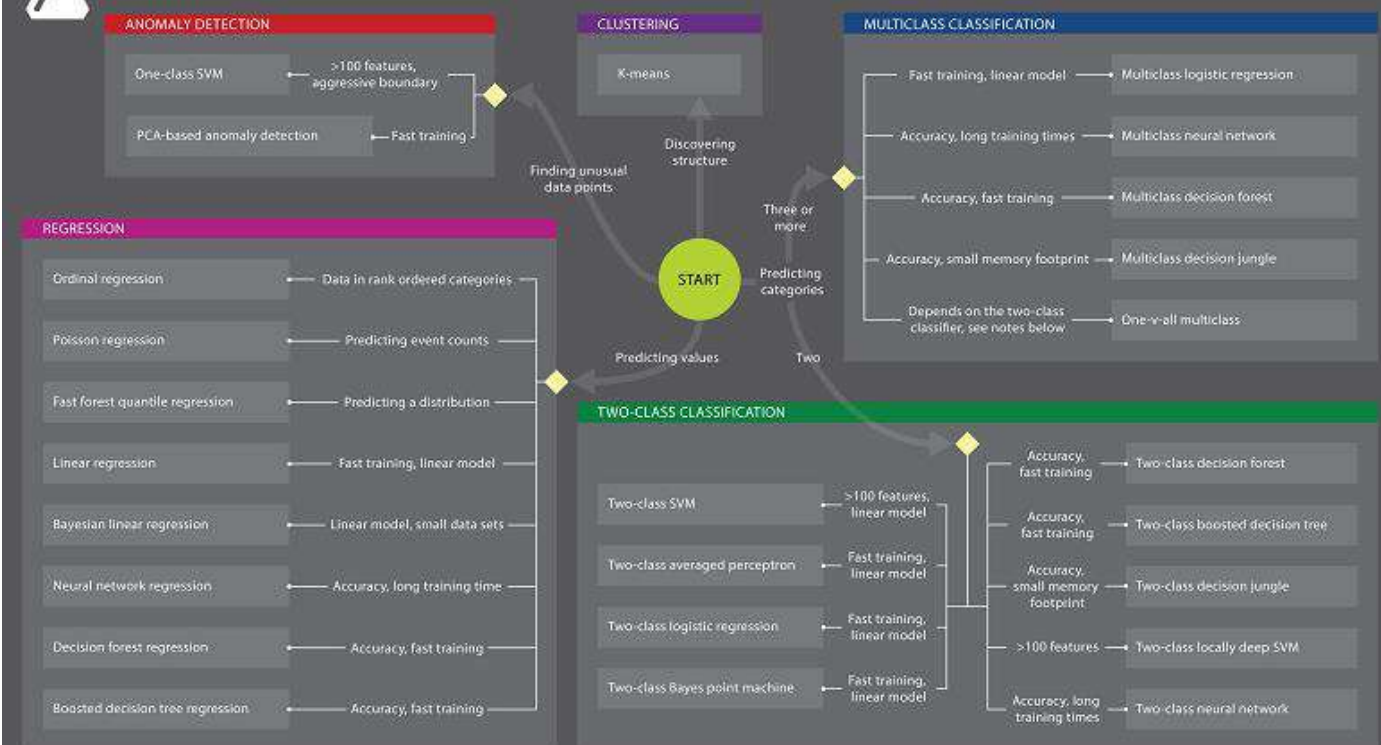
Learn Python For Data Science interactively





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.



© 2015 Microsoft Corporation. All rights reserved.

Created by the Azure Machine Learning Team

Email: AzurePoster@microsoft.com

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

Microsoft

Python For Data Science Cheat Sheet

Python Basics

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



Variables and Data Types

Variable Assignment

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

Calculations With Variables

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

Types and Type Conversion

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

Asking For Help

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

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> 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_list2[listOfList]
>>> 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('u')
Count String elements
>>> my_string.replace('u', '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]
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])
>>> 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(my_array, 5)
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.corrcoeff()
Correlation coefficient
>>> np.std(my_array)
Standard deviation
```

DataCamp

Learn Python for Data Science [interactively](https://www.datacamp.com)



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]
>>> y = [6, 7, 2, 4, 3]
>>> p = figure(title="simple line example",
>>>             x_axis_label='x',
>>>             y_axis_label='y')
>>> p.line(x, y, legend="Temp.", line_width=2)
>>> output_file("lines.html")
>>> show(p)
```

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([[13.9, 4.65, 'US'],
>>>                             [12.4, 4.65, '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, toolbar="pan,box_zoom")
>>> p2 = figure(plot_width=300, plot_height=300,
>>>             x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

3 Renderers & Visual Customizations

Glyphs

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

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

Rows & Columns Layout

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

Columns
>>> from bokeh.layouts import column
>>> 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, Tab
>>> tab1 = Panel(child=p1, title="tab1")
>>> tab2 = Panel(child=p2, title="tab2")
>>> layout = TabPanel(tabs=[tab1, tab2])
```

Legends

```
Legend Location
>>> p.legend.location = 'bottom left'

Inside Plot Area
>>> r1 = p2.circle(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')
```

Customized Glyphs

Also see [Data](#)

```
Selection and Non-Selection Glyphs
>>> p.circle('mpg', 'cyl', source=cds_df,
>>>          selection_color='red',
>>>          nonselection_alpha=0.1)

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

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

Also see [Data](#)

Linked Plots

```
Linked Axes
>>> p2.x_range = p1.x_range
>>> p2.y_range = p1.y_range

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

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')
```

DataCamp
Learn Python for Data Science interactively

About

TensorFlow

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

Skflow

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

Keras

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

Installation

How to install new package in Python:

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

Example: `pip install requests`

How to install tensorflow?

```
device = cpu/gpu
```

```
python_version = cp27/cp34
```

```
sudo pip install
```

```
https://storage.googleapis.com/
tensorflow/linux/$device/tensorflow-
0.8.0-$python_version-none-linux_x86_
_64.whl
```

How to install Skflow

```
pip install sklearn
```

How to install Keras

```
pip install keras
```

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

Helpers

Python helper

Important functions

```
type(object)
```

Get object type

```
help(object)
```

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

```
dir(object)
```

Get list of object attributes
(fields, functions)

```
str(object)
```

Transform an object to string

```
object?
```

Shows documentations about the object

```
globals()
```

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

```
locals()
```

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

```
id(object)
```

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

```
import __builtin__
```

```
dir(__builtin__)
```

Other built-in functions

TensorFlow

Main classes

```
tf.Graph()
```

```
tf.Operation()
```

```
tf.Tensor()
```

```
tf.Session()
```

Some useful functions

```
tf.get_default_session()
```

```
tf.get_default_graph()
```

```
tf.reset_default_graph()
```

```
ops.reset_default_graph()
```

```
tf.device("/cpu:0")
```

```
tf.name_scope(value)
```

```
tf.convert_to_tensor(value)
```

TensorFlow Optimizers

```
GradientDescentOptimizer
```

```
AdadeltaOptimizer
```

```
AdagradOptimizer
```

```
MomentumOptimizer
```

```
AdamOptimizer
```

```
FtrlOptimizer
```

```
RMSPropOptimizer
```

Reduction

```
reduce_sum
```

```
reduce_prod
```

```
reduce_min
```

```
reduce_max
```

```
reduce_mean
```

```
reduce_all
```

```
reduce_any
```

```
accumulate_n
```

Activation functions

```
tf.nn?
```

```
relu
```

```
relu6
```

```
elu
```

```
softplus
```

```
softsign
```

```
dropout
```

```
bias_add
```

```
sigmoid
```

```
tanh
```

```
sigmoid_cross_entropy_with_logits
```

```
softmax
```

```
log_softmax
```

```
softmax_cross_entropy_with_logits
```

```
sparse_softmax_cross_entropy_with_logits
```

```
weighted_cross_entropy_with_logits
```

etc.

Skflow

Main classes

```
TensorFlowClassifier
```

```
TensorFlowRegressor
```

```
TensorFlowDNNClassifier
```

```
TensorFlowDNNRegressor
```

```
TensorFlowLinearClassifier
```

```
TensorFlowLinearRegressor
```

```
TensorFlowRNNClassifier
```

```
TensorFlowRNNRegressor
```

TensorFlowEstimator

Each classifier and regressor have following fields

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

```
batch_size=32,
```

```
steps=200, // except
```

```
TensorFlowRNNClassifier - there is 50
```

```
optimizer='Adagrad',
```

```
learning_rate=0.1,
```

Python For Data Science Cheat Sheet

Keras

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



Keras

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

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.randn(1000,100)
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(12,
>>>                 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_train,y_train),(x_test,y_test) = imdb.load_data(unlabeled=0.0001)
>>> num_classes = 10
```

Other

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

Preprocessing

Sequence Padding

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

One-Hot Encoding

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

Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

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

Multi-Class Classification

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

Regression

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

Convolutional Neural Network (CNN)

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

Recurrent Neural Network (RNN)

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

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> x_train,x_test,y_train,y_test = train_test_split(
>>>     x, y, test_size=0.15,
>>>     random_state=42)
```

Standardization/Normalization

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

Inspect Model

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

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

```
>>> model.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.



Use the following import convention:

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

Pandas Data Structures

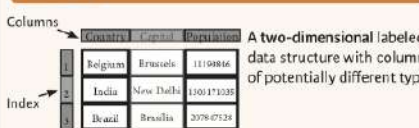
Series

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



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

DataFrame



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

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
>>>          'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
>>>          'Population': [119886, 130371035, 207847528]}
>>> df = pd.DataFrame(data,
>>>                    columns=['Country', 'Capital', 'Population'])
```

I/O

Read and Write to CSV

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

Read and Write to Excel

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

Asking For Help

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

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
>>> -5
>>> df[1:]
>>> Country Capital Population
>>> 1 India New Delhi 130371035
>>> 2 Brazil Brasilia 207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[10, 10]
>>> 'Belgium'
>>> df.iat[10, 10]
>>> 'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[10, ['Country']]
>>> 'Belgium'
>>> df.at[10, ['Country']]
>>> 'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
>>> Country Brazil
>>> Capital Brasilia
>>> Population 207847528
>>> df.ix[:, 'Capital']
>>> 0 Brussels
>>> 1 New Delhi
>>> 2 Brasilia
```

Select single row of subset of rows

Select a single column of subset of columns

Select rows and columns

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

Boolean Indexing

```
>>> a[(a > 1)]
>>> a[(a < -1) | (a > 2)]
>>> df[df['Population'] > 200000000]
```

Series a where value is not >1

Series a where value is <-1 or >2

Use filter to adjust DataFrame

Setting

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

Set index a of Series s to 6

Dropping

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

Drop values from rows (axis=0)

Drop values from columns (axis=1)

Sort & Rank

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

Sort by row or column index

Sort a series by its values

Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

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

(rows, columns)

Describe index

Describe DataFrame columns

Info on DataFrame

Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values

Cumulative sum of values

Minimum/maximum values

Minimum/maximum index value

Summary statistics

Mean of values

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:

```
>>> s1 = 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:

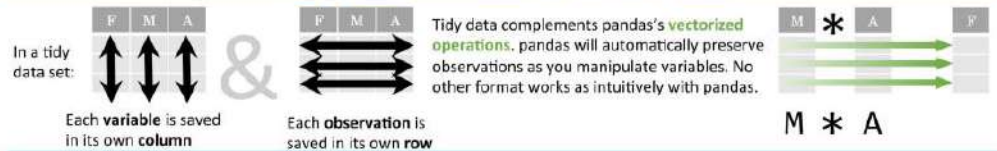
```
>>> s.add(s3, fill_value=0)
>>> a 10.0
>>> b -5.0
>>> c 5.0
>>> d 7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

DataCamp

Learn Python for Data Science interactively



Tidy Data – A foundation for wrangling in 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.

n	v	a	b	c
d	1	4	7	10
e	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



```
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).

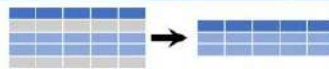
df.rename(columns = {'y': 'year'})
Rename the columns of a DataFrame

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.

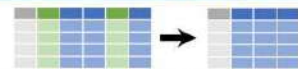
df.sample(n=10)
Randomly select n rows.

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.
```

Subset Variables (Columns)



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

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

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

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

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

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

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

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

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

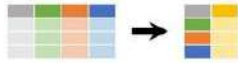
Summarize Data

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

len(df)
of rows in DataFrame.

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

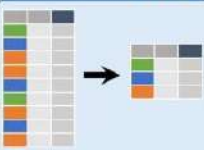
df.describe()
Basic descriptive statistics for each column (or GroupBy)



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

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

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.

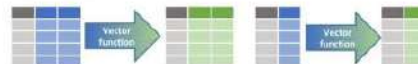
Make New Columns



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.



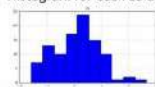
pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

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

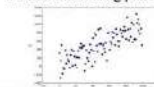
The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

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

df.plot.hist()
Histogram for each column



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



Combine Data Sets



Standard Joins

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

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

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

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

Filtering Joins

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

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



Set-like Operations

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

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

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://founder.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 Luttig, Princeton Consultants

Data Wrangling with dplyr and tidyr

Cheat Sheet

RStudio

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	1.3	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.8	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)

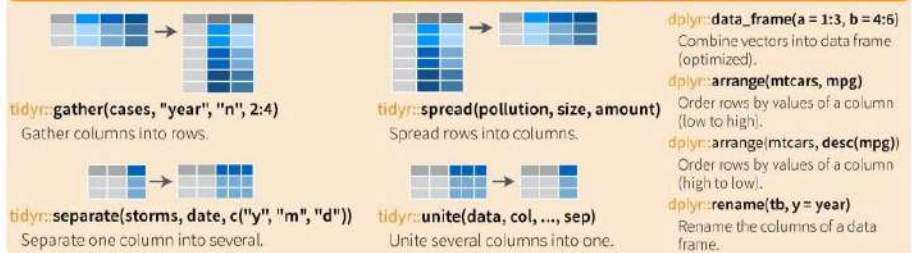
"Piping" with %>% makes code more readable, e.g.

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

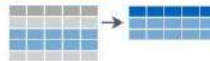
Tidy Data - A foundation for wrangling in R



Reshaping Data - Change the layout of a data set



Subset Observations (Rows)



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

Extract rows that meet logical criteria.

dplyr::distinct(iris)

Remove duplicate rows.

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

Randomly select fraction of rows.

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

Randomly select n rows.

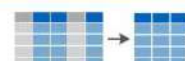
dplyr::slice(iris, 10:15)

Select rows by position.

dplyr::top_n(storms, 2, date)

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

Subset Variables (Columns)



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

Select columns by name or helper function.

Helper functions for select - ?select

```
select(iris, contains("x"))
  Select columns whose name contains a character string.

select(iris, ends_with("Length"))
  Select columns whose name ends with a character string.

select(iris, everything())
  Select every column.

select(iris, matches("x.*"))
  Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))
  Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))
  Select columns whose names are in a group of names.

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

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

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

Logic in R - ?Comparison, ?base::Logic			
<	Less than	!=	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

RStudio is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 344-448-1212 • rstudio.com

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

Learn more with [browse vignettes](#) (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)

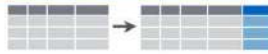
Remove grouping information from data frame.

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 get to first value.

dplyr::ntile

Bin vector into n buckets.

dplyr::between

Are values between a and b?

dplyr::cume_dist

Cumulative distribution.

dplyr::cumall

Cumulative all

dplyr::cumany

Cumulative any

dplyr::cummean

Cumulative mean

cumsum

Cumulative sum

cummax

Cumulative max

cummin

Cumulative min

cumprod

Cumulative prod

pmax

Element-wise max

pmin

Element-wise min

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.

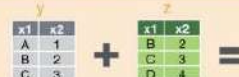
Filtering Joins

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.

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

Index Tricks

```
>>> np.mgrid[0:5,0:5]
>>> np.ogrid[0:5,0:5]
>>> no_1_1_1[0]+1:-1:1:1
>>> np.c_[b,c]
```

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

Shape Manipulation

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

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

Polynomials

```
>>> from numpy import polyval
>>> p = polyval([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)
>>> np.imag(b)
>>> np.real_if_close(c,tol=10000)
>>> np.c_[c['f']]
```

Return the real part of the array elements
Return the imaginary part of the array elements
Return a real array if complex parts close to 0
Cast object to a data type

Other Useful Functions

```
>>> np.angle(b,deg=True)
>>> g = np.linspace(0,np.pi,num=5)
>>> g[3:] = np.pi
>>> np.unwrap(g)
>>> np.linspace(0,10,3)
>>> np.select([c<4],[c+2])
>>> mskr = factorial(x)
>>> nlsr = comb(10,x,ascent=True)
>>> nlsr = central_diff_weights(x)
>>> nlsr = derivative(myfunc,1.0)
```

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

Linear Algebra

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

>>> from scipy import linalg, sparse

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(B)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[1,4], [5,6]])
```

Basic Matrix Routines

Inverse	Inverse
>>> A.I	Inverse
>>> linalg.inv(A)	Inverse
Transpose	Transpose matrix
>>> A.T	Conjugate transposition
>>> A.H	
Trace	Trace
>>> np.trace(A)	
Norm	Frobenius norm
>>> linalg.norm(A)	Li norm (max column sum)
>>> linalg.norm(A,1)	Linf norm (max row sum)
>>> linalg.norm(A,np.inf)	
Rank	Matrix rank
>>> np.linalg.matrix_rank(C)	
Determinant	Determinant
>>> linalg.det(A)	
Solving linear problems	Solver for dense matrices
>>> linalg.solve(A,b)	Solver for dense matrices
>>> B = np.mat(a).T	Least-squares solution to linear matrix equation
>>> linalg.lstsq(F,E)	

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)
>>> G = np.mat(np.identity(2))
>>> C[0,0] = 0.5
>>> H = sparse.csr_matrix(C)
>>> T = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.linalg.spmatrix_csc(A)
```

Create a 3x2 identity matrix
Create a 2x2 identity matrix
Compressed Sparse Row matrix
Compressed Sparse Column matrix
Dictionary of Keys matrix
Sparse matrix to full matrix
Identify sparse matrix

Sparse Matrix Routines

Inverse	Inverse
>>> sparse.linalg.inv(I)	Inverse
Norm	Norm
>>> sparse.linalg.norm(I)	
Solving linear problems	Solver for sparse matrices
>>> sparse.linalg.solve(H,I)	

Sparse Matrix Functions

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

Sparse matrix exponential

Asking For Help

```
>>> help(sparse.linalg.diagsvd)
>>> np.info(np.linalg)
```

Matrix Functions

Addition	Addition
>>> np.add(A,D)	Addition
Subtraction	Subtraction
>>> np.subtract(A,D)	Subtraction
Division	Division
>>> np.divide(A,D)	Division
Multiplication	Multiplication operator
>>> A @ D	(Python3)
>>> np.multiply(B,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	Matrix exponential
>>> linalg.exp(A)	Matrix exponential (Taylor Series decomposition)
>>> linalg.exp2(A)	Matrix exponential (Taylor Series decomposition)
>>> linalg.expn(D)	Matrix exponential (Taylor Series decomposition)
Logarithm Function	Matrix logarithm
>>> linalg.logn(A)	Matrix logarithm
Trigonometric Functions	Matrix sine
>>> linalg.sin(A)	Matrix sine
>>> linalg.cos(A)	Matrix cosine
>>> linalg.tan(A)	Matrix tangent
Hyperbolic Trigonometric Functions	Hyperbolic matrix sine
>>> linalg.sinh(A)	Hyperbolic matrix sine
>>> linalg.cosh(A)	Hyperbolic matrix cosine
>>> linalg.tanh(A)	Hyperbolic matrix tangent
Matrix Sign Function	Matrix sign function
>>> np.sign(A)	Matrix sign function
Matrix Square Root	Matrix square root
>>> linalg.sqrtm(A)	Matrix square root
Arbitrary Functions	Evaluate matrix function
>>> 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
Unpack eigenvalues
Unpack eigenvectors
>>> l1, l2 = la
>>> v1, v2 = v
>>> linalg.eigvals(A)
Singular Value Decomposition
>>> U, s, Vh = linalg.svd(B)
Singular Value Decomposition (SVD)
>>> M, N = B.shape
Construct sigma matrix in SVD
>>> Sig = linalg.diagsvd(s,M,N)
LU Decomposition
>>> P, L, U = linalg.lu(C)
LU Decomposition

Sparse Matrix Decompositions

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

DataCamp

Learn Python for Data Science Interactively



Python For Data Science Cheat Sheet

Matplotlib

Learn Python Interactively at www.datacamp.com



Matplotlib

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



1 Prepare The Data

Also see Lists & NumPy

1D Data

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

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 1 * np.random.random((10, 10))
>>> Y, X = np.mgrid[0:10,0:10], -3:3:100j
>>> T = -1 - X**2 + Y
>>> from matplotlib.colors import get_sample_data
>>> img = np.load(get_sample_data('mook_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

Figure

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.

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([1,2,3],[6,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(y,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

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

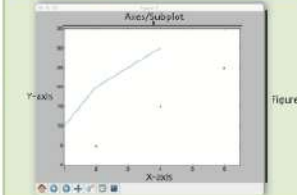
2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img, cmap='gray', interpolation='nearest', vmin=0, vmax=2)
```

Colormapped or RGB arrays

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

4 Customize Plot

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

Markers

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

Linestyles

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

Text & Annotations

```
>>> ax.text(1, -2, 'Example Graph', style='italic')
>>> ax.annotate('PSI=1', xy=(8, 0), xycoords='data', xytext=(10, 5, 0), textcoords='data', arrowprops=dict(arrowstyle='->', connectionstyle='arc3'))
```

MathText

```
>>> plt.title('Psi=1', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling
>>> ax.margins(x=0.0, y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[10, 50], ylim=[-1.5, 1.5])
>>> ax.set_xlim(0, 10.5)
Add padding to a plot. Set the aspect ratio of the plot to 1. Set limits for x and y-axis.
Legends
>>> ax.set(title='An Example Axes', ylabel='Y-Axis', xlabel='X-Axis')
Set a title and x and y-axis labels
No overlapping plot elements
Ticks
>>> ax.xaxis.set(ticks=range(1, 5))
>>> ax.ticks[0].label[0] = 'foo'
>>> ax.tick_params(axis='y', direction='inout', length=10)
Manually set x-ticks
Make y-ticks longer and go in and out
Subplot Spacing
>>> fig3.subplots_adjust(hspace=0.5, wspace=0.5, left=0.125, right=0.8, top=0.9, bottom=0.1)
Adjust the spacing between subplots
Axis Spines
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position('outward', 10)
Make the top axis line for a plot invisible
Move the bottom axis line outward

5 Save Plot

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

Save figures
Save transparent figures

6 Show Plot

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

Close & Clear

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

Clear an axis
Clear the entire figure
Close a window

DataCamp

Learn Python for Data Science Interactively



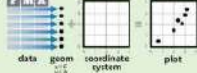
Data Visualization with ggplot2

Cheat Sheet

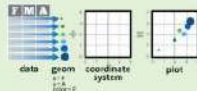


Basics

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



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



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

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

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

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

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

last_plot()

Returns the last plot

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(fill))

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 = "b")
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

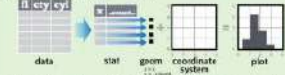
sealsSz ~ with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

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

**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

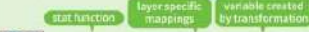
Stats - An alternative way to build a layer

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")`



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

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



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

`geom` for layer `parameters` for stat

`a + stat_bin(bwidth = 1, origin = 10)` 1D distributions
`a + stat_bin_dens(bwidth = 1, bnaes = "y")`
`x, y, count, incout`
`a + stat_density(adjust = 1, kernel = "gaussian")`
`x, y, count, density, scaled`

`f + stat_bin2d(bins = 30, drop = TRUE)` 2D distributions
`x, y, fill, count, density`
`f + stat_binhex(bins = 30)`
`x, y, fill, count, density`
`f + stat_density2d(contour = TRUE, n = 100)`
`x, y, color, size, level`

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

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

`f + stat_ecdf(n = 40)` Functions
`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, ..x.., y.., ymin, ymax`

`ggplot() + stat_function(aes(x = 3:5, fun = geom, n = 10, args = list(sd = 0.5)))` General Purpose
`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, ..x.., y..`
`f + stat_summary(fun.data = "mean_of_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 + 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` `style to use in legend/axis` `labels to use in legend/axis` `looks 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 shows 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

`n + scale_fill_brewer(aes(fill = f))` `o + scale_fill_brewer(aes(fill = f))`

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

`n + scale_fill_brewer(palette = "blues")` For palette choices, library(RColorBrewer) display.brewer.all()

Coordinate Systems

`r + 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, xlim, ylim`
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), azequarea, lgrange, etc.)

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

`z + coord_map(projection = "ortho", orientation = c(41, 74, 0))`

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)`

`t + facet_grid(~ fl, labeller = label_quote(alpha ~ .))`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

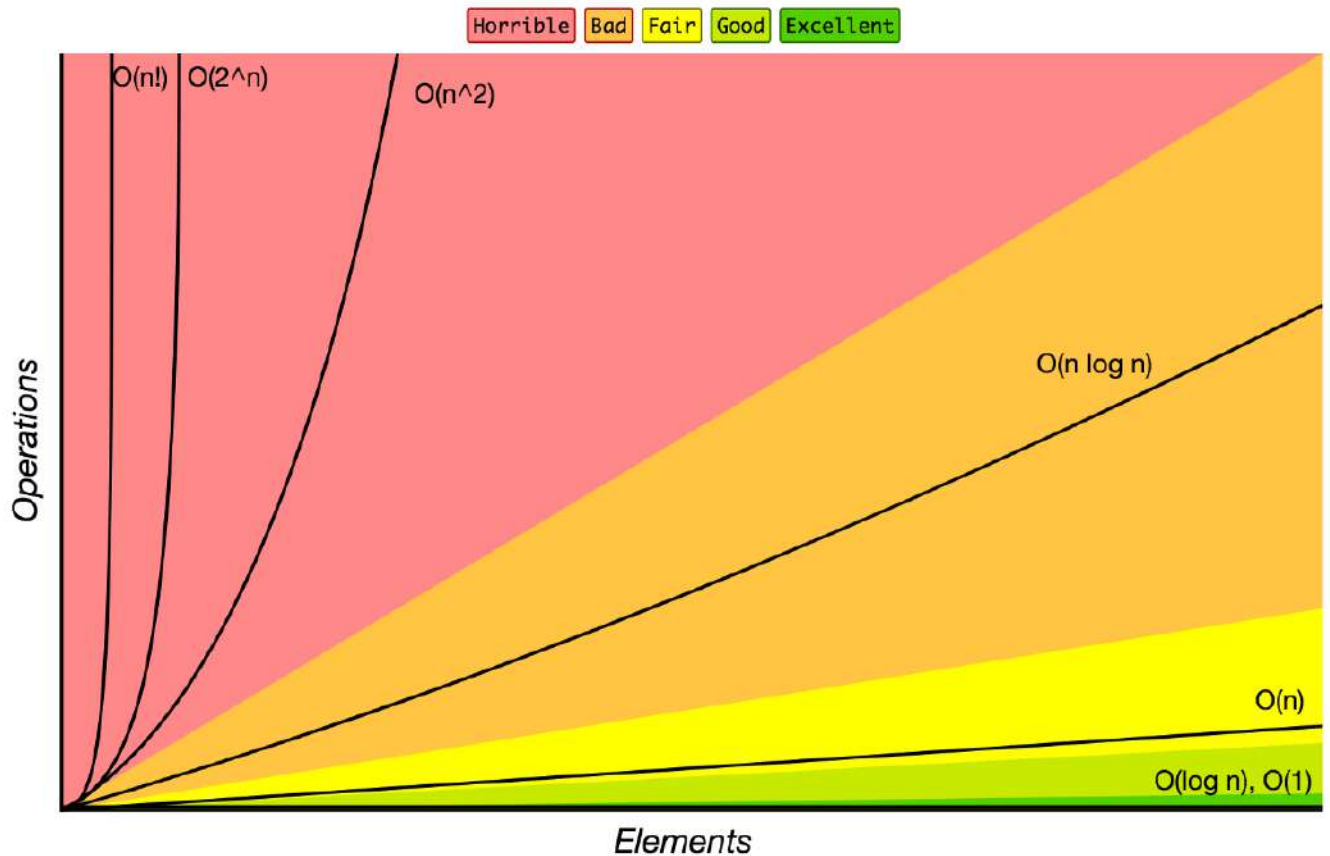
`t + facet_grid(~ fl, labeller = label_parsed)`

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

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

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

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	$O(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Stack	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Queue	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Singly-Linked List	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Doubly-Linked List	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
Skip List	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n \log(n))$
Hash Table	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Binary Search Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Cartesian Tree	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$O(n)$	$O(n)$	$O(n)$	$O(n)$
B-Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$
Red-Black Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$
Splay Tree	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$
AVL Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$
KD Tree	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$

Array Sorting Algorithms

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