A mostly complete chart of Neural Networks

- **Perceptron (P)**
- **Feed Forward (FF)**
- **Radial Basis Network (RBF)**
- **Recurrent Neural Network (RNN)**
- **Long / Short Term Memory (LSTM)**
- **Gated Recurrent Unit (GRU)**
- **Auto Encoder (AE)**
- **Variational AE (VAE)**
- **Denoising AE (DAE)**
- **Sparse AE (SAE)**
- **Markov Chain (MC)**
- **Hopfield Network (HN)**
- **Boltzmann Machine (BM)**
- **Restricted BM (RBM)**
- **Deep Belief Network (DBN)**
- **Deep Convolutional Network (DCN)**
- **Deconvolutional Network (DN)**
- **Deep Convolutional Inverse Graphics Network (DCIGN)**
- **Generative Adversarial Network (GAN)**
- **Liquid State Machine (LSM)**
- **Extreme Learning Machine (ELM)**
- **Echo State Network (ESN)**
- **Deep Residual Network (ORN)**
- **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 Recurrent Example

Deep GRU Example (previous iteration)

Deep GRU Example

Deep LSTM Example (previous iteration)

Deep LSTM Example
**Linear Vector Spaces:**

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

1. \( x + y \in X \) for all \( x, y \in X \)
2. \( a \cdot x \in X \) for all \( a \in F \) and \( x \in X \)
3. \( 0 \) and \( -x \) exist for all \( x \in X \)
4. \( 1 \cdot x = x \) for all \( x \in X \)
5. \( x + 0 = x \) for all \( x \in X \)

**Linear Independence:** Consider vectors \( \{x_1, x_2, ..., x_n\} \). If there exists a scalar \( a_1, a_2, ..., a_n \), not all zero, such that:

\[ a_1 x_1 + a_2 x_2 + ... + a_n x_n = 0 \]

then \( \{x_1, x_2, ..., x_n\} \) are linearly dependent.

**Spanning a Space:** Let \( X \) be a linear vector space and let \( \{u_1, u_2, ..., u_n\} \) be a subset of vectors in \( X \). This subset spans \( X \) if and only if for every vector \( x \in X \) there exist scalars \( c_1, c_2, ..., c_n \) such that:

\[ x = c_1 u_1 + c_2 u_2 + ... + c_n u_n \]

**Inner Product:** For an inner product function \( \langle x, y \rangle \) and \( \langle x, y \rangle \) for any scalar function of \( x \) and \( y \):

1. \( \langle x, y \rangle = \langle y, x \rangle \)
2. \( \langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle \)
3. \( \langle a x, y \rangle = a \langle x, y \rangle \)
4. \( \langle x, y \rangle = 0 \) if and only if \( x = 0 \)

**Norm:** As a scalar function of \( \| x \| \) is defined as a norm if it satisfies:

1. \( \| x \| \geq 0 \)
2. \( \| x \| = 0 \) if and only if \( x = 0 \)
3. \( \| x + y \| \leq \| x \| + \| y \| \)

**Angle:** The angle \( \theta \) between vectors \( x \) and \( y \) is defined by:

\[ \cos \theta = \frac{\langle x, y \rangle}{\| x \| \| y \|} \]

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

**Gram Schmidt Orthogonalization:** Assume that we have \( m \) independent vectors \( y_1, y_2, ..., y_m \). From these vectors we will obtain \( m \) orthogonal vectors \( v_1, v_2, ..., v_m \).

\[ v_k = y_k - \sum_{j=1}^{k-1} \left( \frac{\langle y_j, v_k \rangle}{\langle v_j, v_j \rangle} \right) v_j \]

where \( \langle v_j, v_k \rangle \) is the projection of \( y_k \) on \( v_i \).

**Vector Expansions:**

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

**Reciprocal Basis Vectors:**

\[ \left( \frac{1}{v_i, v_j} \right) \neq \delta \quad \delta = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \]

**Transformations:**

A transformation consists of three parts:

- Domain \( X \), range \( Y \), and a rule relating each \( x \in X \) to an element \( y \in Y \).

**Linear Transformations:** transformation \( A \) is linear if:

1. \( A(x + y) = A(x) + A(y) \)
2. \( A(ax) = a(A(x)) \)

**Matrix Representations:**

Let \( \{v_1, v_2, ..., v_n\} \) be a basis for vector space \( X \), and let \( \{u_1, u_2, ..., u_n\} \) 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:

\[ A(y_j) = \sum_{i=1}^{n} a_{ij} v_i \]

**Change of Basis:**

\[ B_A = [v_1, v_2, ..., v_n] \quad B_B = [w_1, w_2, ..., w_n] \]

**Eigenvectors & Eigenvalues:**

\[ Ax = \lambda x, \quad |A - \lambda I| = 0 \]

**Diagonalization:**

\[ B_A = [x_1, x_2, ..., x_n] \]

where \( x_1, x_2, ..., x_n \) are the eigenvectors of a square matrix \( A \).

**Perceptron Architecture:**

\[ a = \text{hardlim}(Wp + b), \quad W = [w_1, w_2, ..., w_F] \]

\[ u_i = \text{hardlim}(u_i) \]

**Decision Boundary:**

\[ w^T p + b = 0 \]

**Hebb’s Postulate:** When an input cell is active enough to excite a cell \( i \) and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in it or both cells such that its efficiency as one of the cells firing it, is increased.

**Linear Association:**

\[ a = \text{purlin}(Wp) \]

**The Hebb Rule:**

Supervised Form:

\[ W_{F+1} = W_F + \eta e \]

**Unsupervised Hebb:**

\[ W_{F+1} = W_F + \alpha r_i q_i \]

**Taylor’s Expansion:**

\[ F(X) = F(X_0) + \sum \frac{1}{i!} \left( \frac{\partial^i F(X)}{\partial x_1^{i_1} ... \partial x_n^{i_n}} \right) (X - X_0)^{i_1 ... i_n} + \frac{1}{2} \sum \frac{1}{i_1! ... i_n!} \left( \frac{\partial^2 F(X)}{\partial x_1^{i_1} ... \partial x_n^{i_n}} \right) (X - X_0)^{i_1 ... i_n} \]

**Hessian:**

\[ \nabla^2 F(x) \]

**Directional Derivatives:**

1st Dir. Der.: \( \frac{\partial F(x)}{\partial x_i} \)

2nd Dir. Der.: \( \frac{\partial^2 F(x)}{\partial x_i \partial x_j} \)

**Minima:**

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

**Global Minimum:** if \( F(x) < F(x + \delta x) \) for all \( x \neq 0 \).

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

**Optional Conditions:**

1st Condition:

2nd Condition:

**Quadratic fn.:**

\[ F(x) = \frac{1}{2} x^T A x + c \]

**Unit:**
General Minimization Algorithm:
\[ x_{k+1} = x_k + \alpha_k p_k \] or
\[ \Delta x_k = (x_{k+1} - x_k) = \alpha_k p_k \]

Steepest Descent Algorithm:
\[ x_{k+1} = x_k - \alpha_k g_k \quad \text{where,} \quad g_k = \nabla f(x)_{|x=x_k} \]

Stable Learning Rate (\(\alpha_k = \alpha, \text{constant} \)): \(\alpha < \frac{1}{\lambda_{\text{max}}}\)

Eigenvalues of Hessian matrix A
Learning Rate to Minimize Along the Line:
\[ x_{k+1} = x_k + \alpha_k p_k \quad \text{or} \quad \Delta x_k = -\frac{g_k}{\nabla^2 f(x_{|x=x_k})} p_k \] (for quadratic fn.)

After Minimization Along the Line:
\[ x_{k+1} = x_k + \alpha_k g_k \]

ADALINE: \(a = \text{purelin}(Wp + b)\)

Mean Square Error: \(E = E[e^2] = E[(t - a)^2] \quad \text{where} \quad c = E[t^2], \quad h = E[t] \quad \text{and} \quad R = E[z^2] \Rightarrow \lambda = 2, \quad d = -2h\)

Minimum, if it exists, is \(x^* = R^{-1}h\),
where \(x = \left[ \begin{array}{c} w_1 \\ \vdots \\ w_n \\ 1 \end{array} \right] \)

LMS Algorithm: \(W(k + 1) = W(k) + 2\alpha e(k)p^T(k)
\)

Convergence Point: \(x^* = R^{-1}h\)

Stable Learning Rate: \(\alpha < \frac{1}{\lambda_{\text{max}}} \quad \text{where} \quad \lambda_{\text{max}}\)

Adaptive Filter ADALINE:
\[ a(k) = \text{purelin}(Wp(k) + b) = \sum_{i=1}^{k} w_i y(k - i + 1) + b \]

Backpropagation Algorithm:

- Performance Index:
  Mean Square Error: \(E[e^2] = E[(t - a)^2] \quad \text{where} \quad c = E[t^2], \quad h = E[t] \quad \text{and} \quad R = E[z^2] \Rightarrow \lambda = 2, \quad d = -2h\)
  Approximate Performance Index: \(E[(t - a)^2] = \sum_{i=1}^{n} w_i^2 y(k - i + 1) + b \)

- Sensitivity: \(s^m = \frac{\partial f}{\partial p} \quad \text{where} \quad s^m \quad \text{is the maximum eigenvalue of} \quad R \quad \text{Adaptive Filter ADALINE:} \quad a(k) = \text{purelin}(Wp(k) + b) = \sum_{i=1}^{k} w_i y(k - i + 1) + b \)

Backpropagation Algorithm:

- Forward Propagation: \(a^0 = p\)
  \(a^{m+1} = f^{m+1}(W^{m+1}a^m + b^{m+1}) \quad \text{for} \quad m = 0, 1, ..., M - 1 \)
  \(a = a^M \)

- Backward Propagation: \(s^m = -2f^m R^{m+1}(t - a) \quad \text{for} \quad m = M - 1, ..., 2, 1, 0 \quad \text{where} \quad f^m = \text{diag}([f^m(\eta^m_{1}), ..., f^m(\eta^m_{n})]) \)
  \(j^m = \frac{\partial f^m(\eta^m)}{\partial \eta^m} \quad \text{Weight Update (Approximate Steepest Descent)}: \)

  \(W^{m+1}(k + 1) = W^m(k) - \alpha_k \eta^m(a^m(\eta^m)^T) \)
  \(b^{m+1}(k + 1) = b^m(k) - \alpha \eta^m \)

Heuristic Variations of Backpropagation:

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

- Variable Learning Rate Backpropagation (VLRP):
  \(1. \) If the squared error (over the entire training set) increases by more than some set percentage \(\xi \) (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor \(\rho < 1\), and the momentum coefficient \(\gamma\) (if it is used) is set to zero.
  \(2. \) If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor \(\eta > 1\).
  \(3. \) If the squared error increases by less than \(\xi\), then the weight update is accepted for the learning rate but the momentum coefficient is unchanged.

Association: \(a = \text{hardlim}(Wp(b) + b)\)

Associative Learning Rules:

- Unsupervised Hebb Rule:
  \(W(q) = W(q - 1) + \alpha a(q)p^T(q)\)

- Hebb with Decay:
  \(W(q) = (1 - \gamma)W(q - 1) + \alpha a(q)p^T(q)\)

- Instar: \(a = \text{hardlim}(Wp + b)\)

- Self-Organiizing with the Kohonen Rule:
  \(W(q) = \gamma W(q - 1) + \alpha (p(q) - w(q - 1)) \quad \text{for} \quad q \in X(q)\)

- Outstar Rule: \(a = \text{satlin}(Wp)\)
  \(w(q) = \gamma W(q - 1) + \alpha (a(q) - w(q - 1)) p(q) \quad \text{Competitive Layer:} \quad a = \text{compn}(Wp) = \text{compn}(n)\)

Competitive Learning with the Kohonen Rule:

- \(W(q) = \gamma W(q - 1) + \alpha (p(q) - w(q - 1)) \quad \text{where} \quad \gamma = \text{the winning neuron}\)

LVQ Network:

- \(w^{k+1}(i) = \text{subclass} \quad \text{of} \quad a \quad \text{or} \quad p \quad \text{or} \quad q \quad \text{for} \quad q \in X(q)\)

- \(N_i(d) = \text{class} \quad \text{for} \quad q \quad \text{or} \quad p \quad \text{or} \quad q \quad \text{for} \quad q \in X(q)\)

- \(\text{LVQ Network Learning with the Kohonen Rule:} \quad \text{i} \quad \text{w}^{k+1}(i) = \gamma \text{i} \quad \text{w}^{k}(i - 1) + \alpha (p(q) - \text{i} \quad \text{w}^{k}(i - 1)) \quad \text{if} \quad a^{k+1}_i = k_{i+1} \)

- \(\text{i} \quad \text{w}^{k+1}(i) = \gamma \text{i} \quad \text{w}^{k}(i - 1) + \alpha (p(q) - \text{i} \quad \text{w}^{k}(i - 1)) \quad \text{if} \quad a^{k+1}_i = k_{i+1} \)

- \(\text{i} \quad \text{w}^{k+1}(i) = \gamma \text{i} \quad \text{w}^{k}(i - 1) + \alpha (p(q) - \text{i} \quad \text{w}^{k}(i - 1)) \quad \text{if} \quad a^{k+1}_i = k_{i+1} \)

- \(\text{i} \quad \text{w}^{k+1}(i) = \gamma \text{i} \quad \text{w}^{k}(i - 1) + \alpha (p(q) - \text{i} \quad \text{w}^{k}(i - 1)) \quad \text{if} \quad a^{k+1}_i = k_{i+1} \)

- \(\text{i} \quad \text{w}^{k+1}(i) = \gamma \text{i} \quad \text{w}^{k}(i - 1) + \alpha (p(q) - \text{i} \quad \text{w}^{k}(i - 1)) \quad \text{if} \quad a^{k+1}_i = k_{i+1} \)
**MACHINE LEARNING IN EMOJI**

**SUPERVISED**
- **LINEAR REGRESSION**
  - linear_model.LinearRegression()
  - Lots of numerical data
  - Target variable is numerical

**UNSUPERVISED**
- **K-MEANS**
  - cluster.KMeans()
  - Similar datum into groups based on centroids
- **ANOMALY DETECTION**
  - covariance.EllipticalEnvelope()
  - Finding outliers through grouping

**REINFORCEMENT**
- **NEURAL NET**
  - neural_network.MLPClassifier()
  - Complex relationships. Prone to overfitting

**CLASSIFICATION**
- **K-NN**
  - neighbors.KNeighborsClassifier()
  - Group membership based on proximity
- **DECISION TREE**
  - tree.DecisionTreeClassifier()
  - If/then/else. Non-contiguous data
- **RANDOM FOREST**
  - ensemble.RandomForestClassifier()
  - Find best split randomly
- **SVM**
  - svm.SVC()
  - 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-DISTRIBUTED STochastic NEar Embedding (T-SNE)**
  - manifold.TSNE()
  - Visualize high dimensional data. Convert similarity to joint probabilities
- **PRINCIPAL COMPONENT ANALYSIS (PCA)**
  - decomposition.PCA()
  - Distill feature space into components that describe greatest variance
- **CANONICAL CORRELATION ANALYSIS (CCA)**
  - decomposition.CCA()
  - Making sense of cross-correlation matrices
- **LINEAR DISCRIMINANT ANALYSIS (LDA)**
  - Lda.LDA()
  - Linear combination of features that separates classes

**OTHER IMPORTANT CONCEPTS**
- **BIAS VARIANCE TRADEOFF**
- **UNDERFITTING / OVERFITTING**
- **INERTIA**
- **ACCURACY FUNCTION**
  - \( \frac{TP + TN}{P + N} \)
- **PRECISION FUNCTION**
  - \( \frac{TP}{TP + FP} \)
- **SPECIFICITY FUNCTION**
  - \( \frac{TN}{FP + TN} \)
- **SENSITIVITY FUNCTION**
  - \( \frac{TP}{TP + FN} \)

@emilyramilion made this
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/x86_64/tensorflow-0.8.0-cp27-cp27mu-linux_x86_64.whl
How to install Skflow
pip install skflow
How to install Keras
pip install keras
update ~/.keras/keras.json - replace "theano" by "tensorflow"

Helpers
Python helper
Important functions
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

Some useful functions
get_default_session()
get_default_graph()
reset_default_graph()
device("/cpu:0")
name_scope(value)
convert_to_tensor(value)

TensorFlow Optimizers
GradientDescentOptimizer
AdamOptimizer
AdagradOptimizer
MomentumOptimizer
AdamOptimizer
FtrlOptimizer
RMSPropOptimizer
Reduction
reduce_sum
reduce_mean
reduce_all
reduce_any
accumulate_n

Activation functions
relu
softplus
softmax
dropout
bias_add
sigmoid
tanh
sigmoid_cross_entropy_with_logits
softmax
log_softmax
softmax_cross_entropy_with_logits
weighted_cross_entropy_with_logits

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,
NumPy Basics

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

Use the following import convention:
```python
>>> import numpy as np
```

**1D Arrays**
```
| 1D array |
|          |
| 0 1 2 3 |
```

**3D Arrays**
```
| 3D array |
|          |
| a b c d |
| e f g h |
```

**Creating Arrays**

```python
>>> a = np.array([1, 2, 3])
>>> b = np.array([4, 5, 6])
>>> c = np.array([7, 8, 9])
```

**Initial Placeholders**

```python
>>> np.zeros(3)
array([0., 0., 0.])
>>> np.ones(3)
array([1., 1., 1.])
>>> np.arange(3)
array([0, 1, 2])
>>> np.linspace(0, 1, 5)
array([0. , 0.2, 0.4, 0.6, 1. ])
>>> np.empty(3)
array([0. , 0. , 0. ])
```

**Arithmetic Operations**

```python
>>> a + b
array([5., 7., 9.])
>>> a - b
array([-3., -2., -1.])
>>> a * b
array([ 4., 10., 18.])
>>> a / b
array([0.25, 0.2 , 0.17])
```

**Comparisons**

```python
>>> np.array_equal(a, b)
False
>>> np.any(a > b)
False
>>> np.all(a < b)
False
```

**Array Functions**

```python
>>> np.argmax(a)
2
>>> np.argmin(a)
0
>>> np.mean(a)
2.5
>>> np.std(a)
1.2909944487358056
```

**Copying Arrays**

```python
>>> np.copy(a)
array([1, 2, 3])
```

**Sorting Arrays**

```python
>>> np.sort(a)
array([1, 2, 3])
```

**Subsetting, Slicing, Indexing**

```python
>>> a[1]
2
>>> a[1:]
array([2, 3])
```

**Array Manipulation**

```python
>>> a.shape
(3,)
>>> a.ndim
1
>>> a.dtype
dtype('int32')
>>> a.base
< no base array
>>> a.itemsize
4
```

**Getting Help**

```python
>>> np.random.seed(1)
```

**Data Types**

```python
>>> np.int32
<class 'numpy.int32'>
>>> np.float64
<class 'numpy.float64'>
>>> np.complex128
<class 'numpy.complex128'>
```

**I/O**

```python
>>> np.save('my_array.npy', a)
>>> np.load('my_array.npy')
```

**Saving & Loading Text Files**

```python
>>> np.savetxt('my_array.txt', a, delimiter=',')
```

**DataCamp**

Learn Python for Data Science at www.DataCamp.com
**12**

**Data Wrangling with pandas Cheat Sheet**

http://pandas.pydata.org

---

### Syntax — Creating DataFrames

```python
import pandas as pd

# Create a DataFrame
df = pd.DataFrame([
    [1, 2, 3, 10],
    [2, 3, 4, 11],
    [3, 4, 5, 12],
    [4, 5, 6, 13],
    [5, 6, 7, 14],
    [6, 7, 8, 15],
    [7, 8, 9, 16],
    [8, 9, 10, 17],
    [9, 10, 11, 18],
    [10, 11, 12, 19]
], columns=['A', 'B', 'C', 'D'])
```

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

- **pd.melt(df)**
  Gather columns into rows.

- **pd.pivot(columns='var', values='val')**
  Spread rows into columns.

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

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

- **df.sort_values('col')**
  Order rows by values of a column (low to high).

- **df.sort_values('col', ascending=False)**
  Order rows by values of a column (high to low).

- **df.rename(columns = {('a': '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'] > 10]**
  Extract rows that meet logical criteria.

- **df.drop_duplicates()**
  Remove duplicate rows (only considers columns).

- **df.head()**
  Select first n rows.

- **df.tail()**
  Select last n rows.

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

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

- **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']**
  Select single column with specific name.

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

---

### Logical in Python (and pandas)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less Than</td>
<td>Not equal to</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>Greater Than</td>
<td>x is not less than</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>Equal To</td>
<td>x is not less than and not greater than</td>
<td>x == y</td>
</tr>
<tr>
<td>Not Equal To</td>
<td>x is not equal to</td>
<td>x != y</td>
</tr>
</tbody>
</table>

---

pandas provides a large set of \textit{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:

- \texttt{sum()}: Sum values of each object.
- \texttt{count()}: Count non-NA/null values of each object.
- \texttt{median()}: Median value of each object.
- \texttt{quantile([0.25, 0.75])}: Quantiles of each object.
- \texttt{apply(function)}: Apply function to each object.

\texttt{df.} \texttt{assign(area=lambda df: df.Length*df.Height)}

Compute and append one or more new columns.

\texttt{df[‘Vol’] = df[‘Length’]*df[‘Height’]*df[‘Depth’]}  

Bin column into \texttt{n} buckets.

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.

- \texttt{df.groupby(by=“col”).sum()}: Return a GroupBy object, grouped by values in column named “col”.
- \texttt{df.groupby(\texttt{level=\text{“ind”}}).sum()}: 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:

- \texttt{size()}: Size of each group.
- \texttt{agg(function)}: Aggregate group using function.

df.plot.hist()  
Histogram for each column

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

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

\texttt{df.fillna(value)}: Replace NA/Null data with value.

\texttt{df.assign(area=lambda df: df[\texttt{‘Length’}]*df[\texttt{‘Height’}])}  
Compute and append one or more new columns.

\texttt{df[\texttt{‘Vol’}]} \texttt{=} \texttt{df[\texttt{‘Length’}]*df[\texttt{‘Height’}]*df[\texttt{‘Depth’}]}  

Bin column into \texttt{n} buckets.

\texttt{df.} \texttt{assign(area=lambda df: df.Length*df.Height)}

Compute and append one or more new columns.

\texttt{df[\texttt{‘Vol’}]} \texttt{=} \texttt{df[\texttt{‘Length’}]*df[\texttt{‘Height’}]*df[\texttt{‘Depth’}]}  

Bin column into \texttt{n} buckets.
Data Wrangling with dplyr and tidyr Cheat Sheet

Syntax - Helpful conventions for wrangling

dplyr::tbl_dfiris()
Converts data to tbl class. tbls are easier to examine than
data frames. R displays only the data that fits onscreen.


dplyr::glimpse(iris)
Information dense summary of tbl data.

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


dplyr::%>%%
Passes object on left hand side as first argument (or
argument) of function on right hand side.

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

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

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

Logic in R: ?Comparison, ?base:Logics

Tidy Data - A foundation for wrangling in R

In a tidy data set:

Each variable is saved in its own column

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

Reshaping Data - Change the layout of a data set

tidyr::gather(cases, "year", "n", 2:4)
Gather columns into rows.

tidyr::spread(pollution, size, amount)
Spread rows into columns.

tidyr::separate(storms, date, c("y", "m", "d"))
Separate one column into several.

tidyr::unite(data, col..., sep)
Unite several columns into one.

dplyr::data_frame(a = 1:3, b = 4:6)
Combine vectors into data frame

(dplyr::arrange(mtcars, mpg)
Order rows by values of a column

(dplyr::arrange(mtcars, desc(mpg))
Order rows by values of a column

(dplyr::rename(tb, y = year)
Rename the columns of a data frame.

Subset Observations (Rows)

dplyr::filter(iris, Sepal.Length > 7)
Extract rows that meet logical criteria.

dplyr::distinct(iris)
Remove duplicate rows.

dplyr::sample_frac(iris, 0.5, replace = TRUE)
Randomly select fraction of rows.

dplyr::sample_n(iris, 10, replace = TRUE)
Randomly select n rows.

dplyr::slice(iris, 10:15)
Select rows by position.

dplyr::top_n(storms, 2, date)
Select and order top n entities (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("S"))
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("S"))
Select columns whose name matches a regular expression.

select(iris, num_range(1, 15))
Select columns numbered 1:2, 2:15, 15:15.

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

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

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

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

Learn more with browseRpackage("dplyr") • dplyr 0.8.4 • dplyr 0.8.5 • dplyr 0.8.6 • updated 1/20

RStudio is a trademark of RStudio, Inc. • © 2013 RStudio - info@rstudio.com - 844-446-1313 - rstudio.com

devtools::install_github("rstudio/DEVELOPERS") to install
**Data Visualization with ggplot2**

**Basics**

- **ggplot()** is based on the grammar of graphics, the idea that you can build any 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 shape.

**Graphical Primitives**

- **ggplot()** creates a theme for the plot, which defines the background and other graphical elements.
- **geom()** adds layers to the plot, each layer representing a different aspect of the data.
- **stat()** generates a statistical transform of the data, such as calculating summary statistics.

**Add a new layer to a plot with a geom()**

- **last_plot()** returns the last plot.
- **ggsave(plot.png, width = 5, height = 5)** saves the plot as a file named "plot.png" in the working directory.
- **matches file type to file extension**

**Learn more at**

- ggplot2.org
- ggplotry.com
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time Complexity</th>
<th>Space Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Average</td>
</tr>
<tr>
<td>Quicksort</td>
<td>Ω(n log(n))</td>
<td>Θ(n log(n))</td>
</tr>
<tr>
<td>Mergesort</td>
<td>Ω(n log(n))</td>
<td>Θ(n log(n))</td>
</tr>
<tr>
<td>Timsort</td>
<td>Ω(1)</td>
<td>Θ(n log(n))</td>
</tr>
<tr>
<td>Heapsort</td>
<td>Ω(n log(n))</td>
<td>Θ(n log(n))</td>
</tr>
<tr>
<td>Bubble Sort</td>
<td>Ω(n)</td>
<td>Θ(n^2)</td>
</tr>
<tr>
<td>Insertion Sort</td>
<td>Ω(1)</td>
<td>Θ(n^2)</td>
</tr>
<tr>
<td>Selection Sort</td>
<td>Ω(n^2)</td>
<td>Θ(n^2)</td>
</tr>
<tr>
<td>Tree Sort</td>
<td>Ω(n log(n))</td>
<td>Θ(n log(n))</td>
</tr>
<tr>
<td>Shell Sort</td>
<td>Ω(n log(n))</td>
<td>Θ(n log(n))^2</td>
</tr>
<tr>
<td>Bucket Sort</td>
<td>Ω(n+k)</td>
<td>Θ(n+k)</td>
</tr>
<tr>
<td>Radix Sort</td>
<td>Ω(nk)</td>
<td>Θ(nk)</td>
</tr>
<tr>
<td>Counting Sort</td>
<td>Ω(n+k)</td>
<td>Θ(n+k)</td>
</tr>
<tr>
<td>Cubesort</td>
<td>Ω(n)</td>
<td>Θ(n log(n))</td>
</tr>
</tbody>
</table>