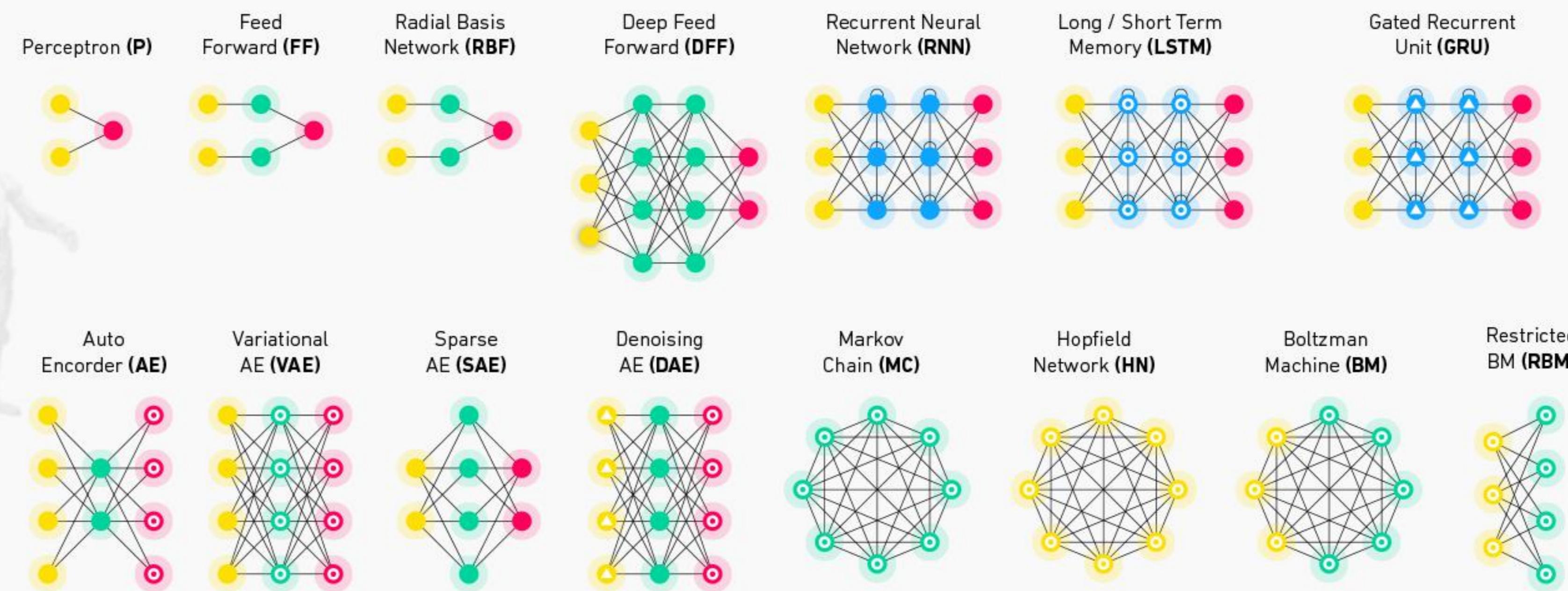


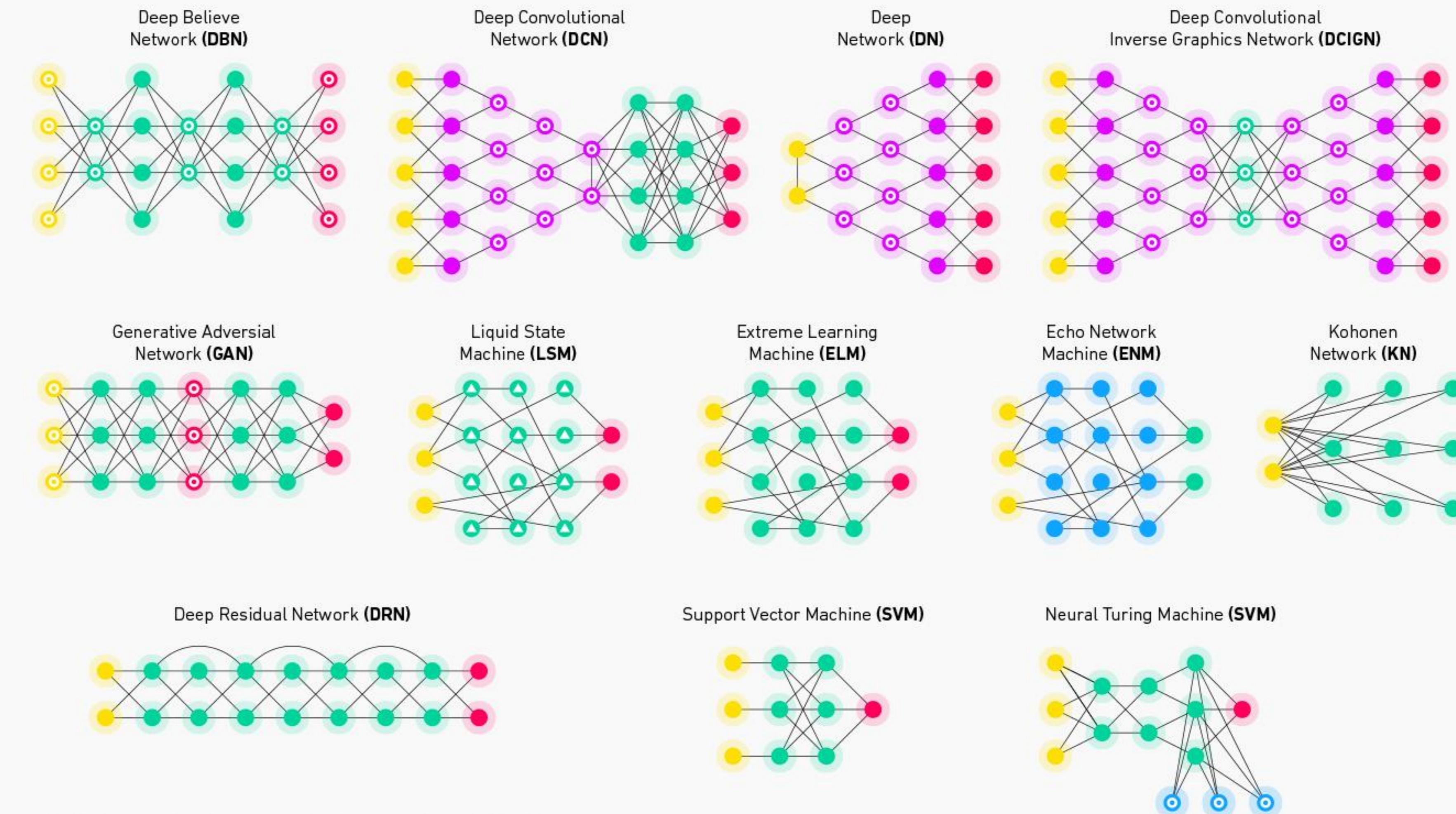
Neural Networks Basic Cheat Sheet

BecomingHuman.AI



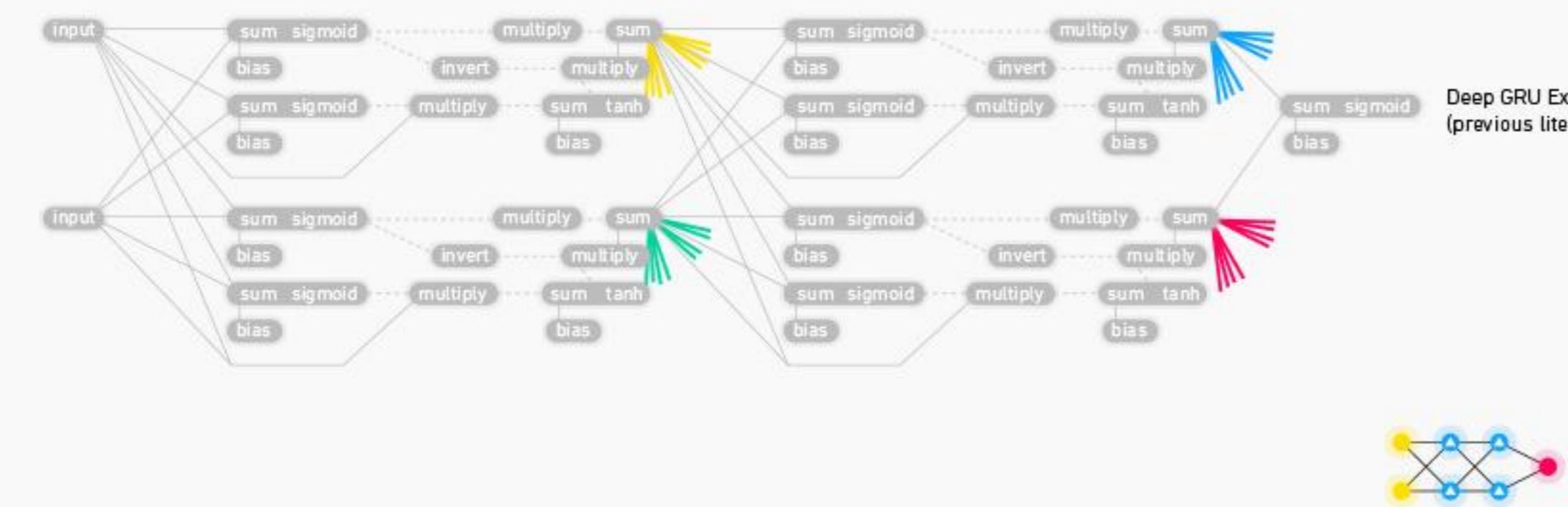
Index

- (○) 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
- (○) Convolutional or Pool

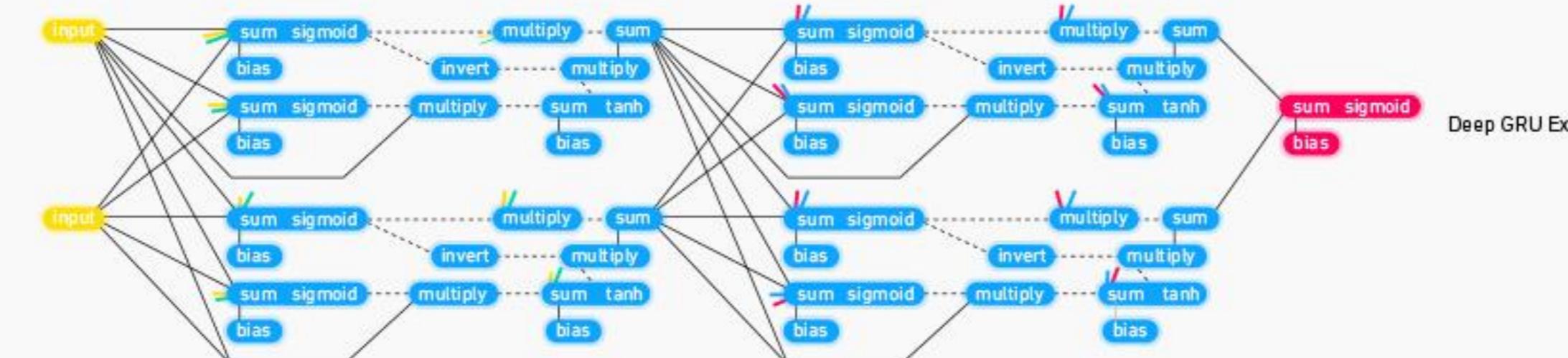


Neural Networks Graphs Cheat Sheet

BecomingHuman.AI



Deep GRU Example
(previous iteration)



Deep GRU Example



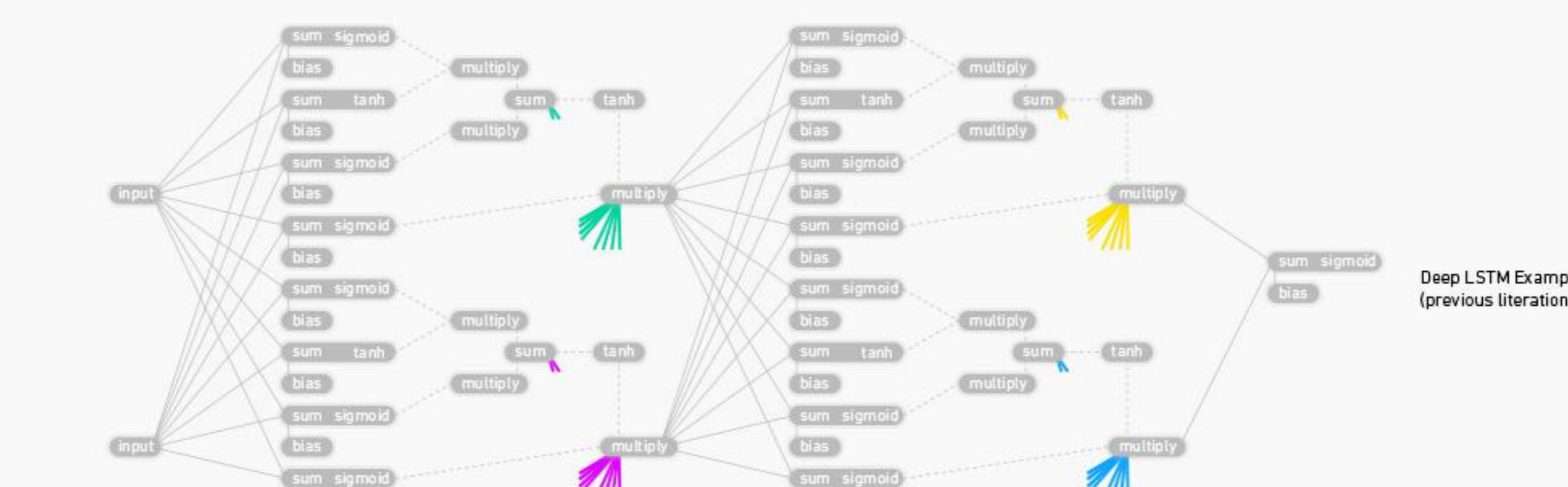
Deep Feed
Forward Example



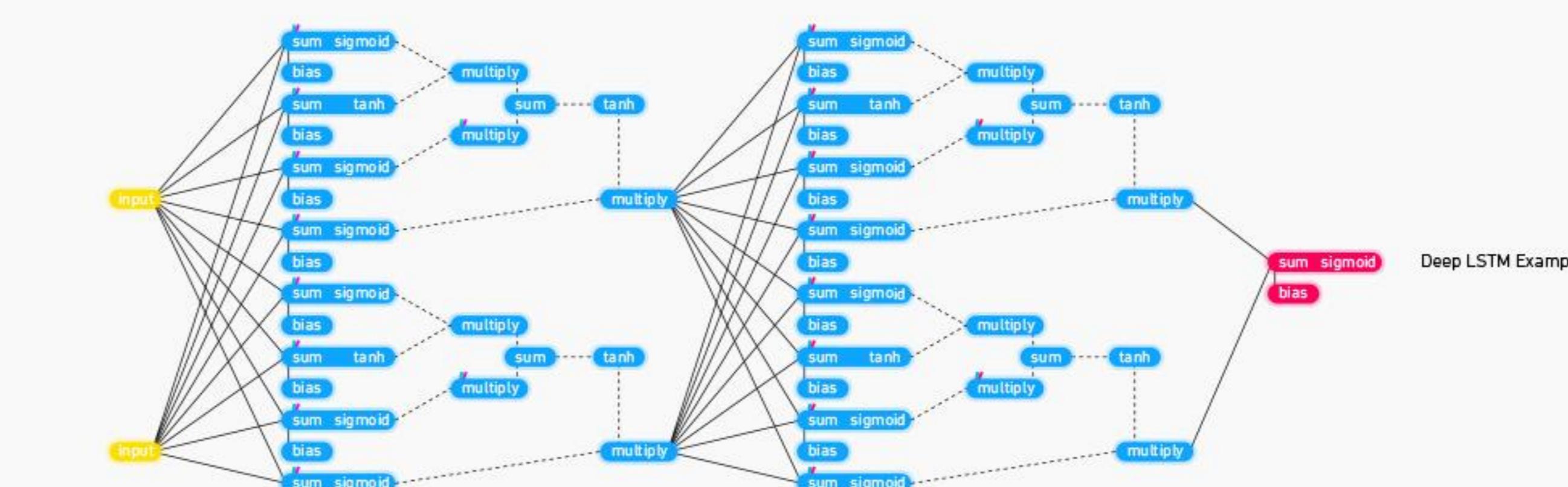
Deep Recurrent
Example
(previous iteration)



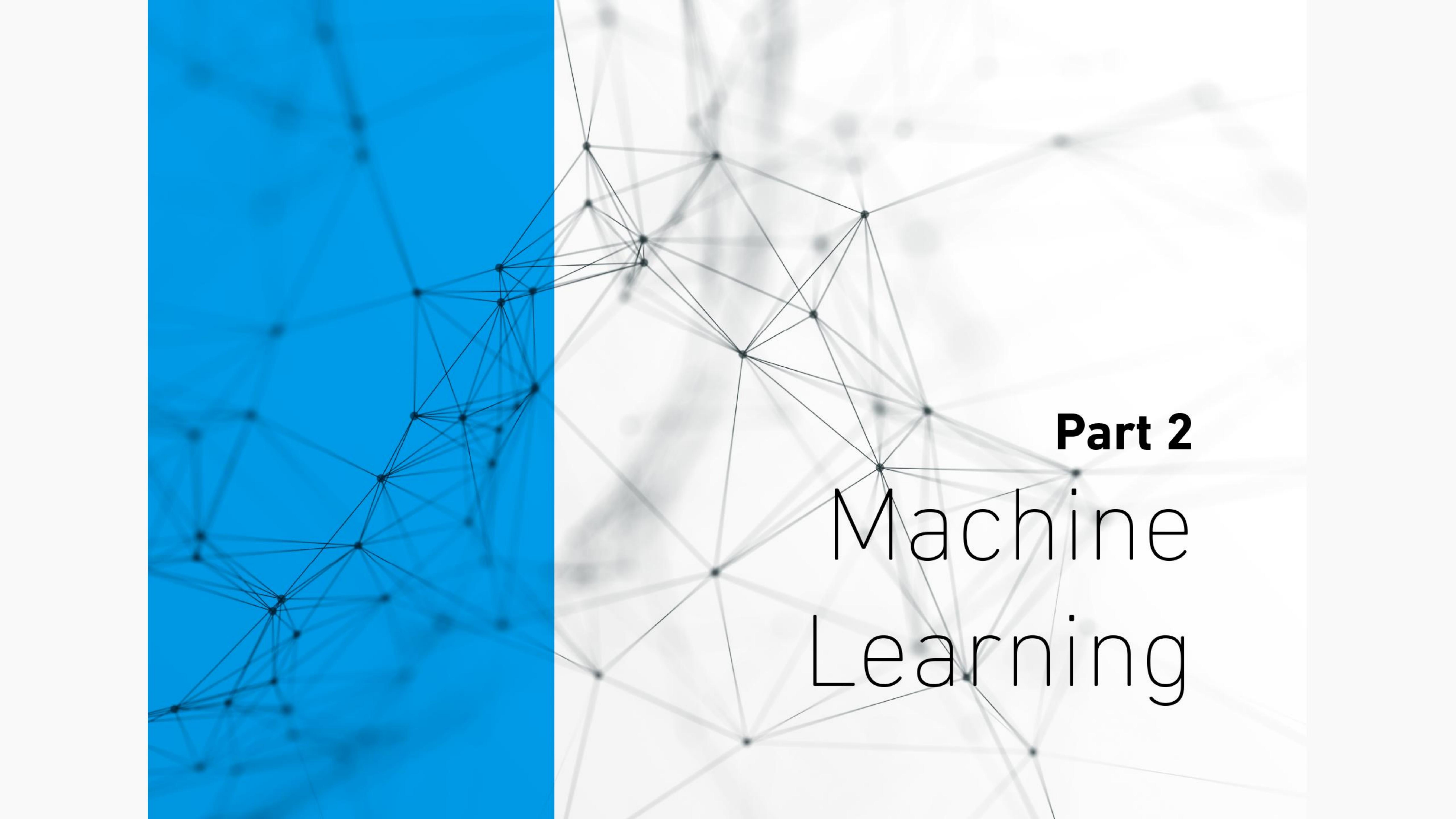
Deep Recurrent
Example



Deep LSTM Example
(previous iteration)



Deep LSTM Example



Part 2

Machine Learning

MachineLearning Overview

MACHINE LEARNING IN EMOJI

BecomingHuman.AI

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

CLUSTER ANALYSIS

K-MEANS



`cluster.KMeans()`
Similar datum into groups based on centroids

ANOMALY DETECTION



`covariance.EllipticalEnvelope()`
Finding outliers through grouping

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

FEATURE REDUCTION

T-DISTRIB STOCHASTIC

NEIB EMBEDDING



`manifold.TSNE()`

Visual 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



`lida.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 `manifold.TSNE()`

SPECIFICITY FUNCTION $TN / (FP+TN)$

SENSITIVITY FUNCTION $TP / (TP+FN)$

Cheat-Sheet Skicit learn

Phyton For Data Science

BecomingHuman.AI



Skicit Learn

Skicit Learn is an open source Phyton library that implements a range if machine learning, processing, cross validation and visualization algorithm using a unified

A basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris() >>> X, y = iris.data[:, :2], iris.target
>>> Xtrain, X test, y_train, y test = train_test_split(X, y, random_state=3)
>>> 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)
```

Prediction

Supervised Estimators
>>> y_pred = svc.predict(np.random.rand(2,5))

Predict labels
Estimate probability of a label

Unsupervised Estimators
>>> y_pred = k_means.predict(X test)

Predict labels in clustering algos

Loading the Data

Your data beeds to be nmueric and stored as NumPy arrays or SciPy sparse matrix, other types that they are convertible to numeric arrays, such as Pandas Dataframe, are also acceptable

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

Preprocessing The Data

Standardization

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

Normalization

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

Binarization

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

Encoding Categorical Features

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

Imputing Missing Values

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

Generating Polynomial Features

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

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

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

Estimator score method
Metric scoring functions

Classification Report

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

Precision, recall, f1-score and support

Confusion Matrix

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

Regression Metrics

Mean Absolute Error

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

Mean Squared Error

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

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))
```

Model Fitting

Supervised learning

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

Fit the model to the data

Unsupervised Learning

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

Fit the model to the data

Fit to data, then transform it

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)
```

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)
```

Tune Your Model

Grid Search

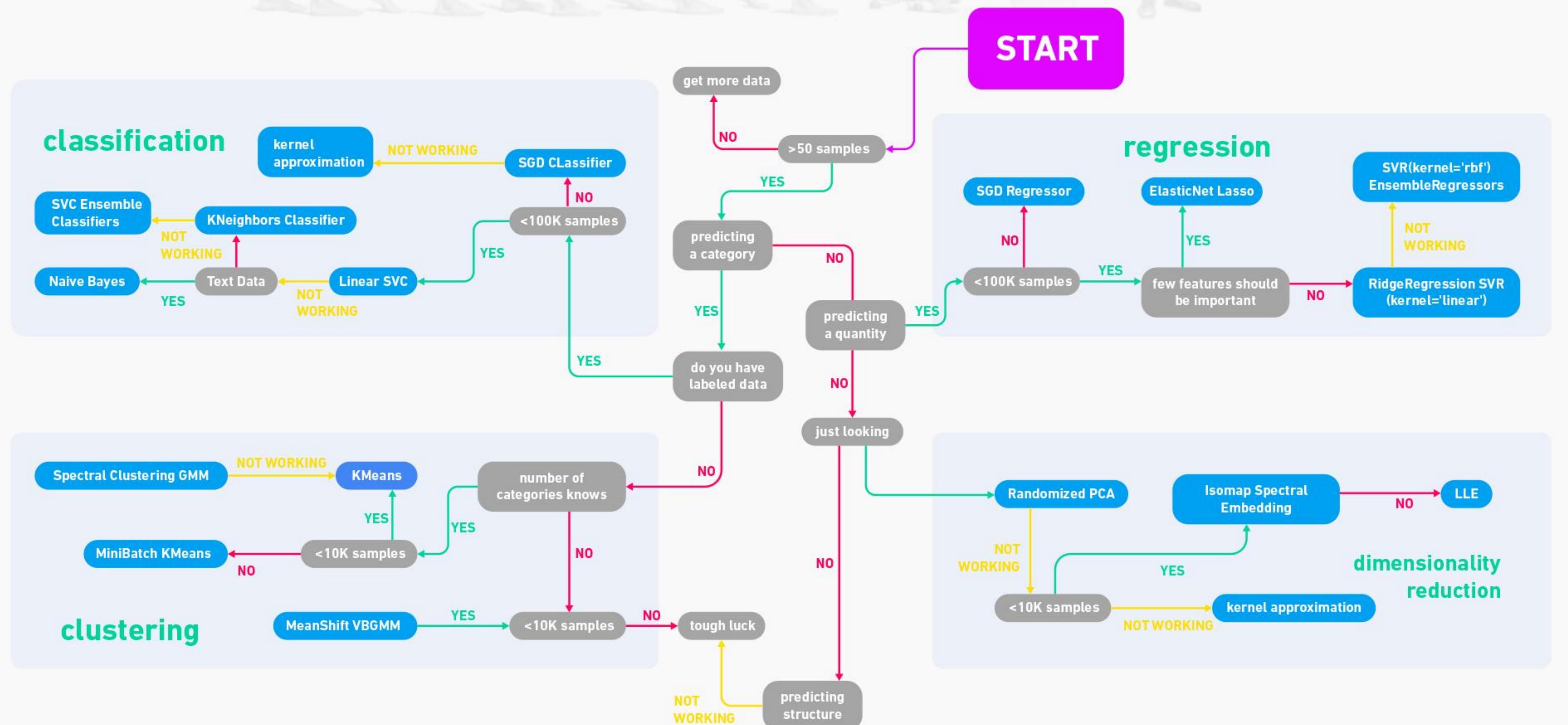
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {'n_neighbors': np.arange(1,3),
'metric': ['euclidean', 'cityblock']}
>>> grid = GridSearchCV(estimator=knn,
param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

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

Skicit-learn Algorithm

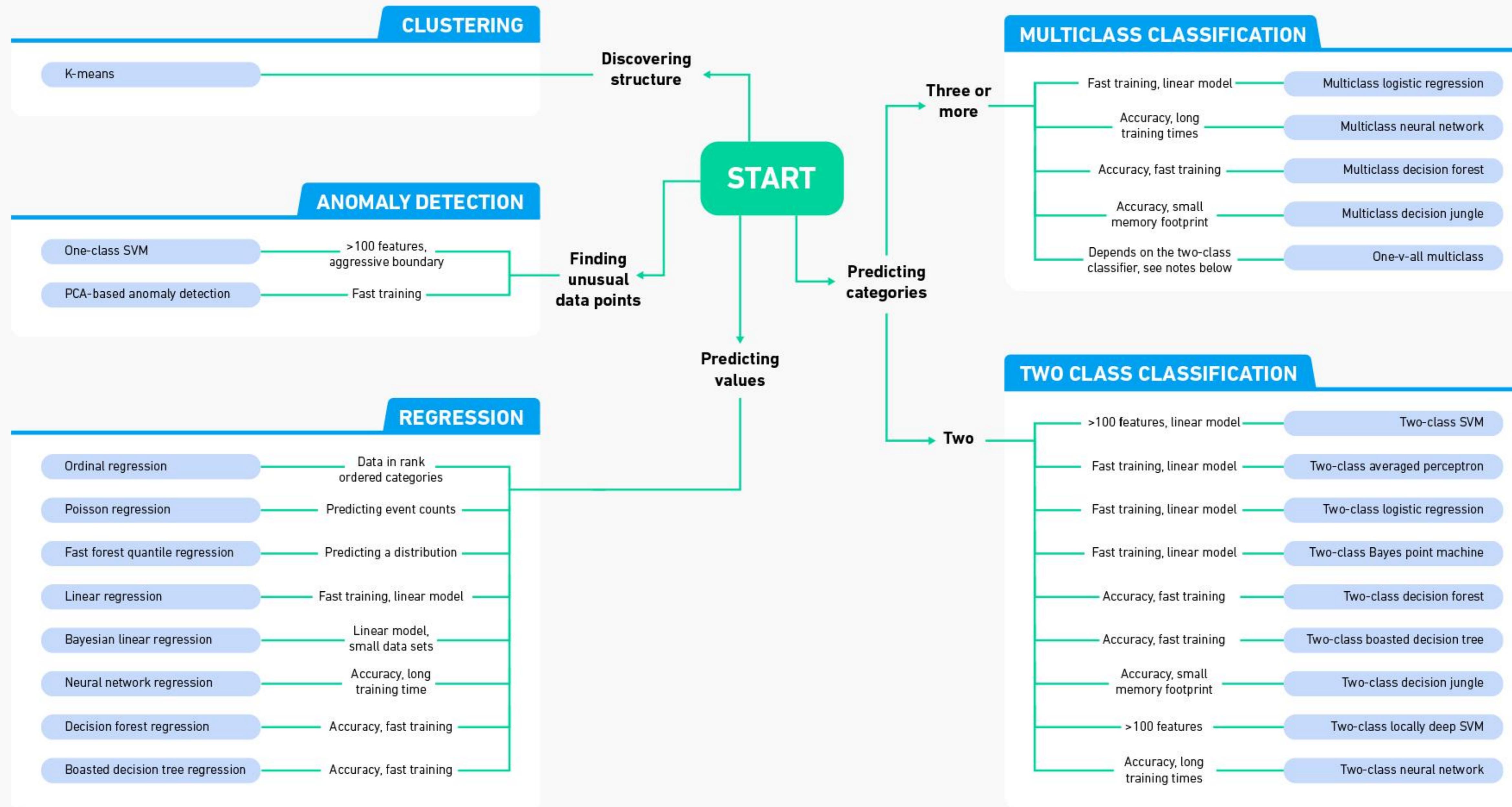
BecomingHuman.AI



Algorithm Cheat Sheet

BecomingHuman.AI

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.



The background of the slide features a complex, abstract network graph. It consists of numerous small, semi-transparent grey nodes connected by thin grey lines, forming a dense web of triangles. On the left side, there is a large, solid orange rectangle that overlaps the network. This orange area also contains a smaller, darker orange network graph, which is more concentrated in the center-left of the slide.

Part 3

Data Science with Python

Tensor Flow Cheat Sheet

BecomingHuman.AI



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

Info

TensorFlow

TensorFlow™ is an open source software library created by Google for numerical computation and large scale computation. Tensorflow bundles together Machine Learning, Deep learning models and frameworks and makes them useful by way of common metaphor.

Keras

Keras is an open sourced neural networks library, written in Python and is built for fast experimentation via deep neural networks and modular design. It is capable of running on top of TensorFlow, Theano, Microsoft Cognitive Toolkit, or PlaidML.

Skflow

Scikit Flow is a high level interface base on tensorflow which can be used like sklearn. You can build your own model on your own data quickly without rewriting extra code.provides a set of high level model classes that you can use to easily integrate with your existing Scikit-learn pipeline code.

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  
sudo pip install
```

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
```

Tensor Flow

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 (Classifier)

```
batch_size=32,  
steps=200, // except  
TensorFlowRNNClassifier - there is 50  
optimizer='Adagrad',  
learning_rate=0.1,
```

Each class has a method fit

```
fit(X, y, monitor=None, logdir=None)
```

X: matrix or tensor of shape [n_samples, n_features...]. Can be iterator that returns arrays of features. The training input samples for fitting the model.

Y: vector or matrix [n_samples] or [n_samples, n_outputs]. Can be iterator that returns array of targets. The training target values (class labels in classification, real numbers in regression).

monitor: Monitor object to print training progress and invoke early stopping

logdir: the directory to save the log file that can be used for optional visualization.

```
predict (X, axis=1, batch_size=None)
```

Args:

X: array-like matrix, [n_samples, n_features...] or iterator.
axis: Which axis to argmax for classification.
By default axis 1 (next after batch) is used. Use 2 for sequence predictions.

batch_size: If test set is too big, use batch size to split it into mini batches. By default the batch_size member variable is used.

Returns:

y: array of shape [n_samples]. The predicted classes or predicted value.

Phyton For Data Science

Cheat-Sheet Phyton Basic

BecomingHuman.AI



Variables and Data Types

Variable Assignment

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

Calculations With Variables

>>> x+2	Sum of two variables
7	
>>> x-2	Subtraction of two variables
3	
>>> x*2	Multiplication of two variables
10	
>>> x**2	Exponentiation of a variable
25	
>>> x%2	Remainder of a variable
1	
>>> x/float(2)	Division of a variable
2.5	

Calculations With Variables

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)
```

Lists

Also see NumPy Arrays

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

Selecting List Elements

Index starts at 0

Subset
>>> my_list[1] Select item at index 1
>>> my_list[-3] Select 3rd last item
Slice
>>> my_list[1:3] Select items at index 1 and 2
>>> my_list[1:] Select items after index 0
>>> my_list[:3] Select items before index 3
>>> my_list[::] Copy my_list
Subset Lists of Lists
>>> my_list2[1][0] my_list[list][itemOfList]
>>> my_list2[1][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('l')	Append an item at a time
>>> my_list.remove('l')	Remove an item
>>> del(my_list[0:1])	Remove an item
>>> my_list.reverse()	Reverse the list
>>> my_list.extend('l')	Append an item
>>> my_list.pop(-1)	Remove an item
>>> my_list.insert(0,'l')	Insert an item
>>> my_list.sort()	Sort the list

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

Numpy Array Operations

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

Numpy Array Operations

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

Strings

Also see NumPy Arrays

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

String Operations

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

String Operations

Index starts at 0

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

String Methods

>>> my_string.upper()	String to uppercase
>>> my_string.lower()	String to lowercase
>>> my_string.count('w')	Count String elements
>>> my_string.replace('e', 'i')	Replace String elements
>>> my_string.strip()	Strip whitespaces

Libraries

Import libraries
>>> import numpy
>>> import numpy as np
Selective import
>>> from math import pi

Install Python



Leading open data science platform
powered by Python



Free IDE that is included
with Anaconda



Create and share
documents with live code,
visualizations, text, ...

Python For Data Science Cheat Sheet

PySpark - RDD Basics

BecomingHuman.AI



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

Initializing Spark

SparkContext

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

Calculations With Variables

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

Configuration

```
>>> from pyspark import SparkConf, SparkContext  
>>> conf = (SparkConf()  
          .setMaster('local')  
          .setAppName("My app")  
          .set("spark.executor.memory", "1g"))  
>>> sc = SparkContext(conf = conf)
```

Configuration

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]  
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python `.zip`, `.egg` or `.py` files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])  
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])  
>>> rdd3 = sc.parallelize(range(100))  
>>> rdd4 = sc.parallelize([('a','x','y','z'),  
                         ('b','p','r')])
```

External Data

Read either one text file from HDFS, a local file system or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile('/my/directory/*.txt')  
>>> textFile2 = sc.wholeTextFiles('/my/directory/')
```

Selecting Data

Getting

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

Sampling

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

Filtering

```
>>> rdd.filter(lambda x: "a" in x).collect()  
[(a,7),(a,2)]  
>>> rdd5.distinct().collect()  
[(a,2),(b,7)]  
>>> rdd.keys().collect()  
[(a, a),(b, b)]
```

Iterating

Getting

```
>>> def g(x): print(x)  
>>> rdd.foreach(g)  
(a, 7)  
(b, 2)  
(a, 2)
```

Retrieving RDD Information

Basic Information

```
>>> rdd.getNumPartitions()  
3  
>>> rdd.count()  
3  
>>> rdd.countByKey()  
defaultdict(<type 'int'>,[(a,2),(b,1)])  
>>> rdd.countByValue()  
defaultdict(<type 'int'>,[(b,2):1,(a,2):1,(a,7):1])  
>>> rdd.collectAsMap()  
{(a, 2): b, (b, 2): a}  
>>> rdd3.sum()  
Sum of RDD elements  
4950  
>>> sc.parallelize([]).isEmpty()  
True
```

List the number of partitions
Count RDD instances

Count RDD instances by key

Count RDD instances by value

Return (key,value) pairs as a dictionary

Sum of RDD elements

Check whether RDD is empty

Summary

```
>>> rdd3.max()  
99  
>>> rdd3.min()  
0  
>>> rdd3.mean()  
49.5  
>>> rdd3.stdev()  
28.86607004772218  
>>> rdd3.variance()  
833.25  
>>> rdd3.histogram(3)  
([(0,33,66,99),(33,33,34)])  
>>> rdd3.stats()  
Summary statistics (count, mean, stdev, max & min)
```

Maximum value of RDD elements

Minimum value of RDD elements

Mean value of RDD elements

Standard deviation of RDD elements

Compute variance of RDD elements

Compute histogram by bins

Summary statistics (count, mean, stdev, max & min)

Applying Functions

```
>>> rdd.map(lambda x: x+(x[1],x[0])).collect()  
[(a,7,a),(a,2,2),(b,2,2,b)]  
>>> rdd5 = rdd.flatMap(lambda x:  
x+[x[1],x[0]])  
>>> rdd5.collect()  
[(a,7),(a,2,2),(b,2,2,b)]  
>>> rdd4.flatMapValues(lambda x: x).collect()  
[(a,x),(a,y),(a,z),(b,p),(b,r)]
```

Apply a function to each RDD element

Apply a function to each RDD element and flatten the result

Apply a flatMap function to each (key,value) pair of rdd4 without changing the keys

Mathematical Operations

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

Return each rdd value not contained

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

Return the Cartesian product of rdd and rdd2

Sort

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

Sort RDD by given function

RDD by key

Reshaping Data

Reducing

```
>>> rdd.reduceByKey(lambda x,y: x+y).collect()  
[(a,9),(b,2)]  
>>> rdd.reduce(lambda a, b: a + b).collect()  
(a,7,a,2,b,2)
```

Merge the rdd values for

Merge the rdd values

Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2).mapValues(list).collect()  
[(a,[7,2]),(b,[2])]  
>>> rdd3.groupByKey().mapValues(list).collect()  
[(a,[7,2]),(b,[2])]
```

Return RDD of grouped values

Group rdd by key

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y[1]))  
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))  
>>> rdd3.aggregate((0,0),seqOp,combOp).collect()  
(4950,100)  
>>> rdd3.aggregateByKey((0,0),seqOp,combOp).collect()  
[(a,(9,2)),(b,(2,1))]  
>>> rdd3.fold(0,add).collect()  
(4950)  
>>> rdd3.foldByKey(0, add).collect()  
[(a,9),(b,2)]  
>>> rdd3.keyBy(lambda x: x+x).collect()  
Create tuples of RDD elements by applying a function
```

Aggregate RDD elements of each partition and then the results

Aggregate values of each RDD key

Aggregate the elements of each 4950 partition, and then the results

Merge the values for each key

Reshaping Data

Reshaping Data

```
>>> rdd.repartition(4)  
New RDD with 4 partitions  
>>> rdd.coalesce(1)  
Decrease the number of partitions in the RDD to 1
```

Saving

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

Stopping SparkContext

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

Execution

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

NumPy Basics Cheat Sheet

BecomingHuman.AI



1D array

1	2	3
---	---	---

2D array

1.5	2	3
4	5	6

3D array

axis 1	axis 0
axis 2	axis 1

axis 0

axis 1

axis 2

Creating Arrays

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

Initial Placeholders

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

I/O

Saving & Loading On Disk

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

Saving & Loading Text Files

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

Inspecting Your Array

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

Data Types

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

Asking For Help

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

Array Mathematics

Arithmetic Operations

```
>>> g = a - b               Subtraction
>>> np.subtract(a,b)        Subtraction
>>> b + a                  Addition
>>> np.add(b,a)            Addition
>>> a / b                  Division
>>> np.divide(a,b)          Division
>>> a * b                  Multiplication
>>> np.multiply(a,b)        Multiplication
>>> np.exp(b)               Exponentiation
>>> np.sqrt(b)              Square root
>>> np.sin(a)               Print sines of an array
>>> np.cos(b)               Element-wise cosine
>>> np.log(a)               Element-wise natural logarithm
>>> e.dot(f)                Dot product
```

Comparison

```
>>> a == b                 Element-wise comparison
>>> a < 2                  Element-wise comparison
>>> np.array_equal(a, b)    Array-wise comparison
```

Aggregate Functions

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

Copying Arrays

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

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2]                   Select the element at the 2nd index
3
>>> b[1,2]                 Select the element at row 1 column 2
6.0

```

Slicing

```
>>> a[0:2]                 Select items at index 0 and 1
array([1, 2])
>>> b[0:2,1]               Select items at rows 0 and 1 in column 1
array([2., 5.])
>>> b[:,1]
array([1.5, 2., 3.])
>>> c[1,...]
array([[3, 2, 1],
       [4, 5, 6]])
>>> a[::1]
array([3, 2, 1])
```

Select the element at the 2nd index

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

Select items at index 0 and 1

Select items at rows 0 and 1 in column 1

Select all items at row 0 (equivalent to b[0,:,1])

Same as [1,:,:]

Reversed array a

Boolean Indexing

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

Select elements from less than 2

Fancy Indexing

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

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

Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

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

Changing Array Shape

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

Adding/Removing Elements

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

Combining Arrays

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

Splitting Arrays

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

Split the array horizontally at the 3rd

Split the array vertically at the 2nd index



Bokeh Cheat Sheet

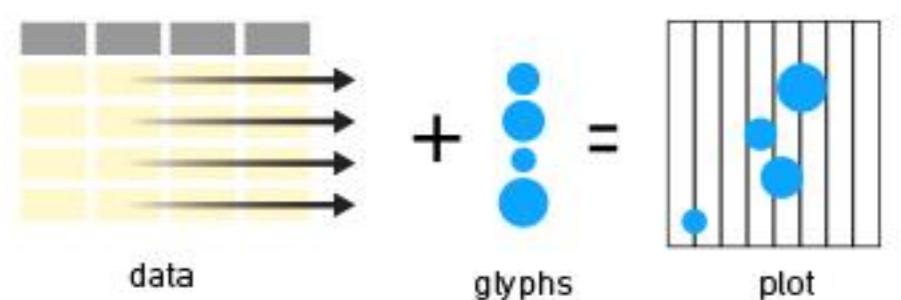
BecomingHuman.Ai



Data Types

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

Bokeh's mid-level general purpose bokeh.plotting interface is centered around two main components: data and glyphs.



The basic steps to creating plots with the bokeh.plotting interface are:

1. Prepare some data:
Python lists, NumPy arrays, Pandas DataFrames and other sequences of values
2. Create a new plot
3. Add renderers for your data, with visual customizations
4. Specify where to generate the output
5. Show or save the results

```
>>> from bokeh.plotting import figure
>>> from bokeh.io import output_file, show
>>> x = [1, 2, 3, 4, 5] step 1
>>> y = [6, 7, 2, 4, 5]
>>> p = figure(title='simple line example', step 2
              x_axis_label='x',
              y_axis_label='y')
>>> p.line(x,y,legend='Temp', line_width=2) step 3
>>> output_file('lines.html') step 4
>>> show(p) step 5
```

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([[32.9,4.65, 'US'],
                                [32.4,4.66, 'Asia'],
                                [21.4,4.10, 'Europe']]),
                                columns=['mpg','cyl','hp','origin'],
                                index=['Toyota', 'Fiat', 'Volve'])
```

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

Plotting

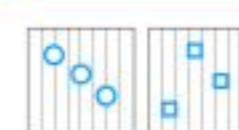
```
>>> from bokeh.plotting import figure
>>> p1 = figure(plot_width=300, tools='pan,box_zoom')
>>> p2 = figure(plot_width=300, plot_height=300,
               x_range=(0, 8), y_range=(0, 8))
>>> p3 = figure()
```

Show or Save Your Plots

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

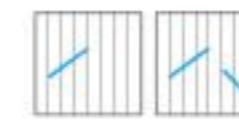
Renderers & Visual Customizations

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]])
```

Legends

Legend Location

Inside Plot Area

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

Outside Plot Area

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

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()
```

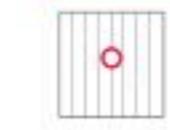
Standalone HTML

```
>>> from bokeh.embed import file_html
>>> html = file_html(p, CDN, 'my_plot')
```

Components

```
>>> from bokeh.embed import components
>>> script, div = components(p)
```

Customized Glyphs



Selection and Non-Selection Glyphs

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



Hover Glyphs

```
>>> hover = HoverTool(tooltips=None, mode='vline')
>>> p3.add_tools(hover)
```



Colormapping

```
>>> color_mapper = CategoricalColorMapper(
             factors=['US', 'Asia', 'Europe'],
             palette=['blue', 'red', 'green'])
>>> p3.circle('mpg', 'cyl', source=cds_df,
             color=dict(field='origin',
                        transform=color_mapper),
             legend='Origin')
```

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')
```

Tabbed Layout

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

Legend Orientation

```
>>> p.legend.orientation = 'horizontal'
>>> p.legend.orientation = 'vertical'
```

Legend Background & Border

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

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',
```

Keras Cheat Sheet

BecomingHuman.AI



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

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
>                  activation='relu',
>                  input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
>                 loss='binary_crossentropy',
>                 metrics=[accuracy])
```

Data

Also see NumPy, Pandas & Scikit-Learn

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

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
>>>                      mnist,
>>>                      cifar10,
>>>                      imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Other

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

Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

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

Multi-Class Classification

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

Regression

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

Convolutional Neural Network (CNN)

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

Inspect Model

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

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

Prediction

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

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)
```

Compile Model

MLP: Binary Classification

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

MLP: Multi-Class Classification

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

MLP: Regression

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

Recurrent Neural Network

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

Save/ Reload Models

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

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)
```

Train and Test Sets

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

Standardization/ Normalization

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

Pandas Basics

Cheat Sheet

BecomingHuman.AI



Use the following import convention: >>> import pandas as pd

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

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'])
```

Data Frame

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

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
   ...:         'Capital': ['Brussels', 'New Delhi', 'Brasília'],
   ...:         'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
   ...:                  columns=['Country', 'Capital', 'Population'])
```

Dropping

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

Drop values from rows (axis=0)
Drop values from columns (axis=1)

Sort & Rank

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

Sort by labels along an axis
Sort by the values along an axis
Assign ranks to entries

Retrieving Series/ DataFrame 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
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
-> 5
>>> df[1:]
   Country Capital Population
1 India New Delhi 1303171035
2 Brazil Brasilia 207847528
```

Get one element
Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position
>>> df.iloc[[0], [0]]
'Belgium'
>>> df.iat[[0], [0]]
'Belgium'

By Label
>>> df.loc[[0], ['Country']]
'Belgium'
>>> df.at[0, 'Country'] 'Belgium'

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

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

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

Select single value by row & column
Select single value by row & column labels

Select single row of subset of rows

Select a single column of subset of columns

Select rows and columns

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

Set index a of Series s to 6

Asking For Help

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

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> 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)

I/O

Read and Write to CSV

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

Read and Write to Excel

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

Read multiple sheets from the same file

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

Read and Write to SQL Query or Database Table

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

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

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

Pandas

Cheat Sheet

BecomingHuman.AI

Pandas Data Structures

Pivot

```
>>> df3 = df2.pivot(index='Date',
                   columns='Type',
                   values='Value')
```

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Type	a	b	c
0	Date			
1	2016-03-01	11.432	NaN	20.784
2	2016-03-02	1.303	13.031	NaN
3	2016-03-03	99.906	NaN	20.784

Pivot Table

```
>>> df4 = pd.pivot_table(df2,
                        values='Value',
                        index='Date',
                        columns='Type')
```

	0	1
1	5	0.233482
2	4	0.184713
3	3	0.433522

Unstacked

	0	1	2	3	4	5				
1	5	0.233482	0.390959							
2	4	0.184713	0.237102							
3	3	0.433522	0.429401							
4				1	0.184713					
5					1	0.237102				
6						0	0.233482			
7							1	0.390959		
8								2	0.433522	
9									3	0.429401

Stacked

Melt

```
>>> pd.melt(df2,
            id_vars=['Date'],
            value_vars=['Type', 'Value'],
            value_name='Observations')
```

	Date	Type	Value
0	2016-03-01	a	11.432
1	2016-03-02	b	13.031
2	2016-03-01	c	20.784
3	2016-03-03	a	99.906
4	2016-03-02	a	1.303
5	2016-03-03	c	20.784

	Date	Variable	Observations
0	2016-03-01	Type	a
1	2016-03-02	Type	b
2	2016-03-01	Type	c
3	2016-03-03	Type	a
4	2016-03-02	Type	a
5	2016-03-03	Type	c
6	2016-03-01	Value	11.432
7	2016-03-02	Value	13.031
8	2016-03-01	Value	20.784
9	2016-03-03	Value	99.906
10	2016-03-02	Value	1.303
11	2016-03-03	Value	20.784

Advanced Indexing

Also see NumPy Arrays

Selecting

```
>>> df3.loc[:,(df3>1).any()]
>>> df3.loc[:,(df3>1).all()]
>>> df3.loc[:,df3.isnull().any()]
>>> df3.loc[:,df3.notnull().all()]
```

Select cols with any vals > 1
Select cols with vals > 1
Select cols with NaN
Select cols without NaN

Indexing With isin

```
>>> df[(df.Country.isin(df.Type))]
>>> df.filter(items=["a","b"])
>>> df.select(lambda x: not x%5)
```

Find same elements
Filter on values
Select specific elements

Where

```
>>> s.where(s > 0)
```

Subset the data

Query

```
>>> df6.query('second > first')
```

Query DataFrame

Setting/Resetting Index

```
>>> df.set_index('Country')
>>> df4 = df.reset_index()
>>> df = df.rename(index=str,
                  columns={"Country":"cntry",
                           "Capital":"cptl",
                           "Population":"ppln"})
```

Set the index
Reset the index
Rename DataFrame

Reindexing

```
>>> s2 = s.reindex(['a','c','d','e','b'])
```

Forward Filling

```
>>> df.reindex(range(4),
               method='ffill')
```

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

Forward Filling
method='bfill'

0 3
1 3
2 3
3 3
4 3

Multilindexing

```
>>> arrays = [np.array([1,2,3]),
             np.array([5,4,3])]
>>> df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
>>> tuples = list(zip(*arrays))
>>> index = pd.MultiIndex.from_tuples(tuples,
                                      names=['first', 'second'])
>>> df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
>>> df2.set_index(['Date', 'Type'])
```

Duplicate Data

```
>>> s3.unique()
>>> df2.duplicated('Type')
>>> df2.drop_duplicates('Type', keep='last')
>>> df.index.duplicated()
```

Return unique values
Check duplicates
Drop duplicates
Drop duplicates

Grouping Data

Aggregation

```
>>> df2.groupby(by=['Date','Type']).mean()
>>> df4.groupby(level=0).sum()
>>> df4.groupby(level=0).agg({'a':lambda x:sum(x)/len(x), 'b': np.sum})
```

Transformation

```
>>> customSum = lambda x: (x+x%2)
>>> df4.groupby(level=0).transform(customSum)
```

Missing Data

```
>>> df.dropna()
>>> df3.fillna(df3.mean())
>>> df2.replace("a", "f")
```

Drop NaN value
Fill NaN values with a predetermined value
Replace values with others

Combining Data

data1

X1	X2
a	11.432
b	1.303
c	99.906

X1	X3
a	20.784
b	NaN
d	20.784

Pivot

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
d	NaN	20.784

X1	X2	X3
a	11.432	20.784
b	1.303	NaN
c	99.906	NaN

Join

```
>>> data1.join(data2, how='right')
```

Concatenate

Vertical

<pre

Data Wrangling with pandas Cheat Sheet

BecomingHuman.AI

Syntax Creating DataFrames

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

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

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

	a	b	c
n	v		
d	1	4	7
e	2	5	8
	6	9	11
	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')
     )
```

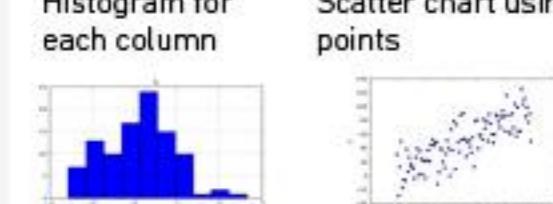
Windows

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.

Windows

df.plot.hist()
Histogram for each column

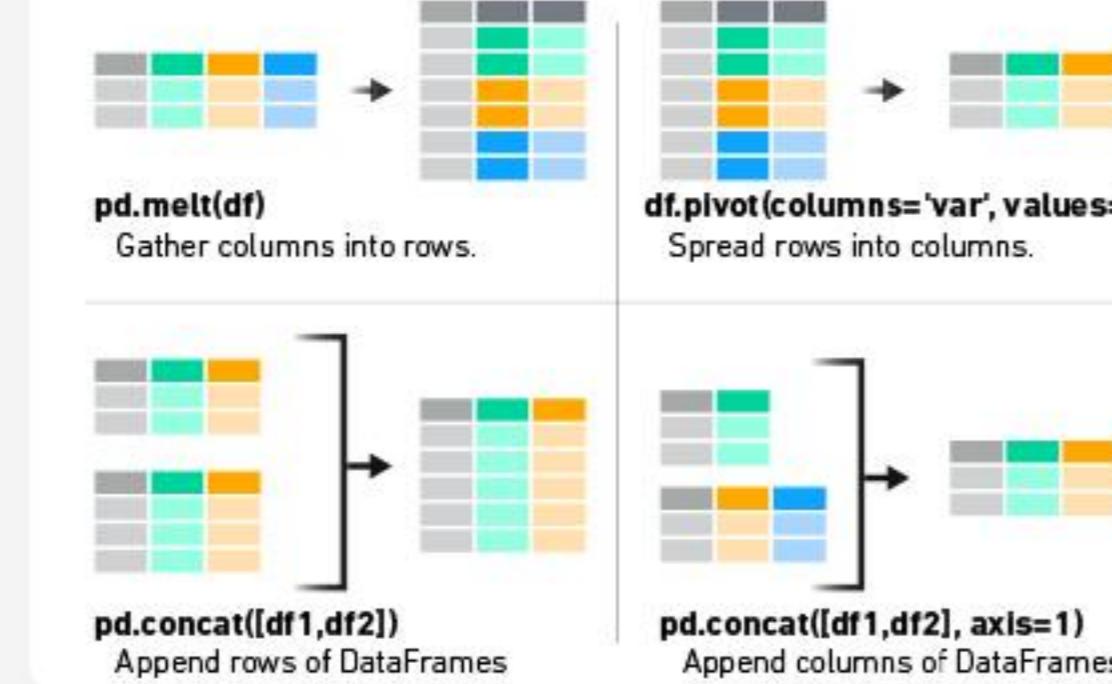


df.plot.scatter(x='w',y='h')
Scatter chart using pairs of points

Tidy Data A foundation for wrangling in pandas



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(columns=['Length','Height'])
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.
```

Logic in Python (and pandas)

<	Less than	df.column.isin(values)	Not equal to
>	Greater than		Group membership
==	Equal to	pd.isnull(obj)	Is NaN
<=	Less than or equal to	pd.notnull(obj)	Is not NaN
>=	Greater than or equal to	&, , ~, df.any(), df.all()	Logical and, or, not, xor, any, all

Logic in Python (and pandas)

<	Less than	df.column.isin(values)	Not equal to
>	Greater than		Group membership
==	Equal to	pd.isnull(obj)	Is NaN
<=	Less than or equal to	pd.notnull(obj)	Is not NaN
>=	Greater than or equal to	&, , ~, df.any(), df.all()	Logical and, or, not, xor, any, all

Windows



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.

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'.

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

```
shift(1)
Copy with values shifted by 1.

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

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

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

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

Summarise 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:

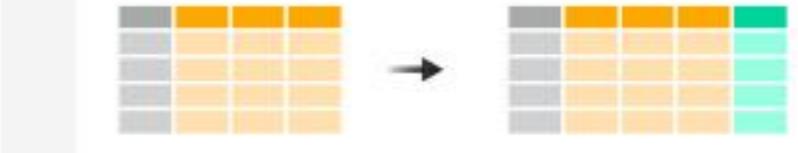
sum()	Sum values of each object.	min()	Minimum value in each object.
count()	Count non-NA/null values of each object.	max()	Maximum value in each object.
median()	Median value of each object.	mean()	Mean value of each object.
quantile([0.25,0.75])	Quantiles of each object.	var()	Variance of each object.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

Handling Missing Data

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

df.fillna(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.

Combine Data Sets



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(columns=['_merge'])

Rows that appear in ydf but not zdf (Setdiff)



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

Data Wrangling with dplyr and tidyr

Cheat Sheet

BecomingHuman.AI

Syntax Helpful conventions for wrangling

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

Source: local data frame [150 x 5]

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

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

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

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa

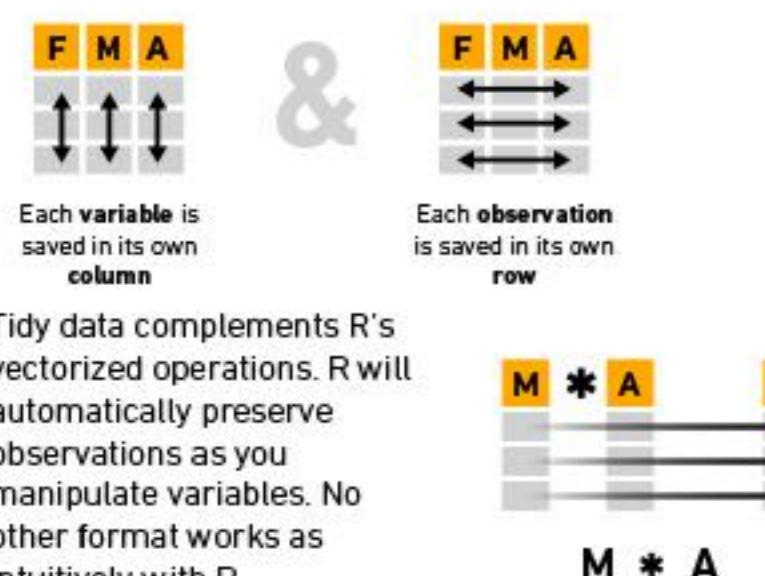
dplyr::%>%
Passes object on left hand side as first argument (or argument) of function on righthand side.
 $x \%>\% f(y)$ is the same as $f(x, y)$
 $y \%>\% f(x, ., z)$ is the same as $f(x, y, z)$

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

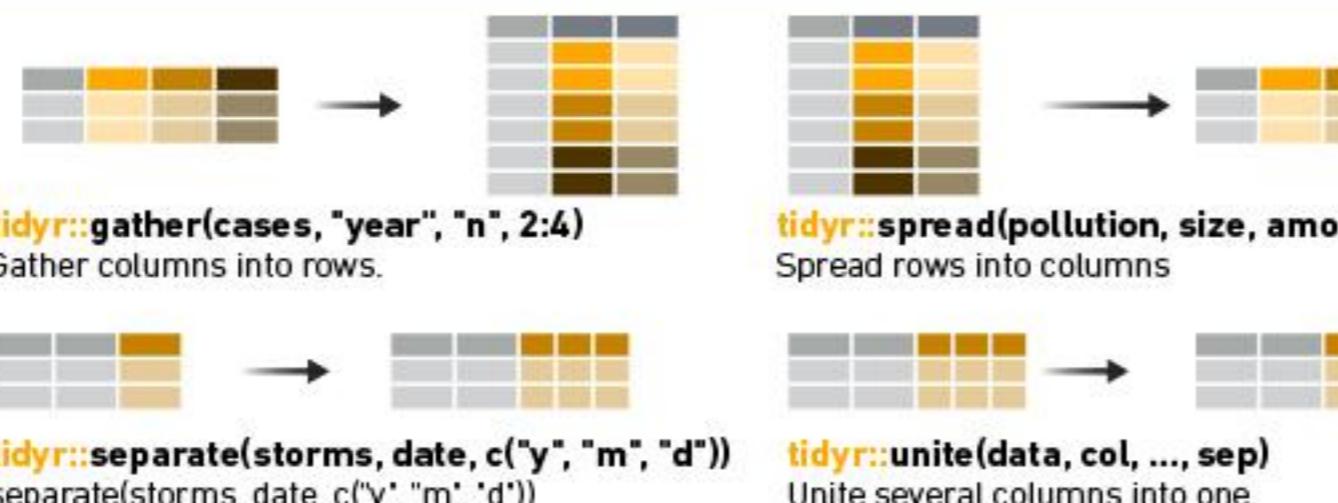
```
iris %>%
  group_by(Species) %>%
  summarise(avg = mean(Sepal.Width)) %>%
  arrange(avg)
```

Tidy Data A foundation for wrangling in R

In a tidy data set:



Reshaping Data Change the layout of a data set



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

dplyr::arrange(mtcars, mpg)
Order rows by values of a column (low to high).

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

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

Subset Observations (Rows)



dplyr::distinct(iris)
Remove duplicate rows.

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

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

dplyr::slice(iris, 10:15)

Select rows by position.

dplyr::top_n(storms, 2, date)
Select and order top n entries (by group if grouped data).

Logic in R - ?	Comparison, %base	:=Logic
<	Less than	!=
>	Greater than	%in%
==	Equal to	is.na
<=	Less than or equal to	is.na
>=	Greater than or equal to	&, l, xor, any.all

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("A.*"))
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).

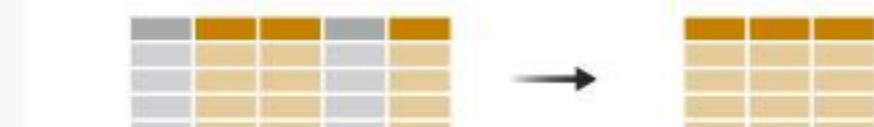
select(iris, -Species)
Select all columns except Species.

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.

Subset Variables (Columns)



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("A.*"))
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.

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).

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

dplyr::first	First value of a vector.	min	Minimum value in a vector.
dplyr::last	Last value of a vector.	max	Maximum value in a vector.
dplyr::nth	Nth value of a vector.	mean	Mean value of a vector.
dplyr::n	# of values in a vector.	median	Median value of a vector.
dplyr::n_distinct	# of distinct values in a vector.	var	Variance of a vector.
dplyr::IQR	IQR of a vector	sd	Standard deviation of a vector.

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

window function
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.	cumall
dplyr::lag	Copy with values lagged by 1.	cumany
dplyr::dense_rank	Ranks with no gaps.	cummean
dplyr::min_rank	Ranks. Ties get min rank.	cumsum
dplyr::percent_rank	Ranks rescaled to [0, 1].	cummax
dplyr::row_number	Ranks. Ties get to first value.	cummin
dplyr::ntile	Bin vector into n buckets.	cumprod
dplyr::between	Are values between a and b?	pmax
dplyr::cume_dist	Cumulative distribution.	pmin

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.

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.

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

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.

Scipy Linear Algebra Cheat Sheet

BecomingHuman.AI



Interacting With NumPy

Also see NumPy

```
>>> import numpy as np  
>>> a = np.array([1,2,3])  
>>> b = np.array([(1+5j,2,3j), (4j,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] Create a dense meshgrid  
>>> np.ogrid[0:2,0:2] Create an open meshgrid  
>>> np.r_[3,0]*5,-1:1:10j Stack arrays vertically (row-wise)  
>>> np.c_[b,c] Create stacked column-wise arrays
```

Shape Manipulation

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

Polynomials

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

Vectorizing Functions

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

Type Handling

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

Other Useful Functions

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

Linear Algebra

Also see NumPy

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

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

Creating Matrices

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

Basic Matrix Routines

	Inverse
>>> A.I	Inverse
>>> linalg.inv(A)	Inverse

	Transposition
>>> A.T	Transpose matrix
>>> A.H	Conjugate transposition

	Trace
>>> np.trace(A)	Trace

	Norm
>>> linalg.norm(A)	Frobenius norm
>>> linalg.norm	L1 norm (max column sum)
>>> linalg.norm(A,np.inf)	L inf norm (max row sum)

	Rank
>>> np.linalg.matrix_rank(C)	Matrix rank

	Determinant
>>> linalg.det(A)	Determinant

	Solving linear problems
>>> linalg.solve(A,b)	Solver for dense matrices
>>> E = np.mat(a).T	Solver for dense matrices
>>> linalg.lstsq(F,E)	Least-squares solution to linear matrix

	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 Matrices
>>> F = np.eye(3, k=1)	Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2))	Create a 2x2 identity matrix

	Compressed Sparse Row matrix
>>> C[C > 0.5] = 0	Compressed Sparse Column matrix
>>> H = sparse.csr_matrix(C)	Dictionary Of Keys matrix
>>> I = sparse.csc_matrix(D)	Sparse matrix to full matrix
>>> J = sparse.dok_matrix(A)	Identify sparse matrix
>>> E.todense()	
>>> sparse.isspmatrix_csc(A)	

Matrix Functions

	Addition
>>> np.add(A,D)	Addition

	Subtraction
>>> np.subtract(A,D)	Subtraction

	Division
>>> np.divide(A,D)	Division

	Multiplication
>>> A @ D	Multiplication operator (Python 3)
>>> np.multiply(D,A)	Multiplication
>>> np.dot(A,D)	Dot product
>>> np.vdot(A,D)	Vector dot product
>>> np.inner(A,D)	Inner product
>>> np.outer(A,D)	Outer product
>>> np.tensordot(A,D)	Tensor dot product
>>> np.kron(A,D)	Kronecker product

	Exponential Functions
>>> linalg.expm(A)	Matrix exponential
>>> linalg.expm2(A)	Matrix exponential (Taylor Series)
>>> linalg.expm3(D)	Matrix exponential (eigenvalue decomposition)

	Logarithm Function
>>> linalg.logm(A)	Matrix logarithm

	Trigonometric Functions
>>> linalg.sinm(D)	Matrix sine
>>> linalg.cosm(D)	Matrix cosine
>>> linalg.tanm(A)	Matrix tangent

	Hyperbolic Trigonometric Functions
>>> linalg.sinhm(D)	Hyperbolic matrix sine
>>> linalg.coshm(D)	Hyperbolic matrix cosine
>>> linalg.tanhm(A)	Hyperbolic matrix tangent

	Matrix Sign Function
>>> np.signm(A)	Matrix sign function

	Matrix Square Root
>>> linalg.sqrtm(A)	Matrix square root

	Arbitrary Functions
>>> linalg.funm(A, lambda x: x*x)	Evaluate matrix function

Sparse Matrix Routines

	Inverse
>>> sparse.linalg.inv(I)	Inverse

	Norm
>>> sparse.linalg.norm(I)	Norm

	Solving linear problems
>>> sparse.linalg.spesolve(H,I)	Solver for sparse matrices

	Sparse Matrix Functions
>>> sparse.linalg.expm(I)	Sparse matrix exponential

	Decompositions
>>> la, v = linalg.eig(A)	Eigenvalues and Eigenvectors

>>> l1, l2 = la	Solve ordinary or generalized eigenvalue problem for square matrix
-----------------	--

>>> v[0]	First eigenvector
>>> v[1]	Second eigenvector
>>> linalg.eigvals(A)	Unpack eigenvalues

	Singular Value Decomposition
>>> U,S,Vh = linalg.svd(B)	Singular Value Decomposition (SVD)

>>> M,N = B.shape	
>>> Sig = linalg.diagsvd(S,M,N)	Construct sigma matrix in SVD

	LU Decomposition
>>> P,L,U = linalg.lu(C)	LU Decomposition

	Sparse Matrix Decompositions
>>> la, v = sparse.linalg.eigs(F,1)	Eigenvalues and eigenvectors

>>> sparse.linalg.svds(H, 2)	SVD
------------------------------	-----

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
```

```
>>> np.info(np.matrix)
```

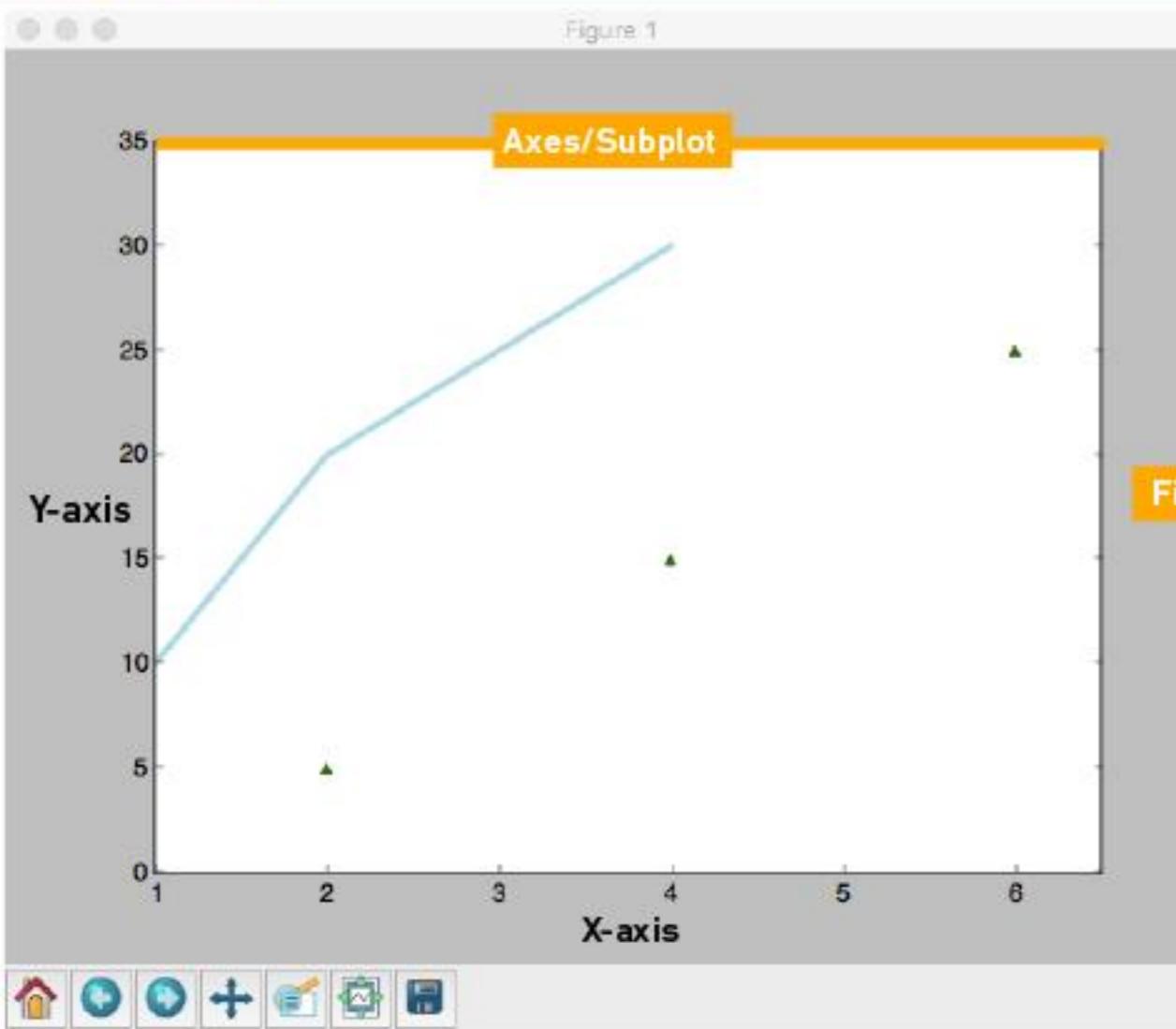
Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

Matplotlib Cheat Sheet

BecomingHuman.AI

Anatomy & Workflow

Plot Anatomy



Workflow

- 01 Prepare data
- 02 Create plot
- 03 Plot
- 04 Customize plot
- 05 Save plot
- 06 Show plot

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

Prepare The Data

Also see [Lists & NumPy](#)

Index Tricks

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

2D Data or Images

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

Create Plot

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

Figure

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

Axes

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

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

Plotting Routines

1D Data

```
>>> lines = ax.plot(x,y)  
>>> ax.scatter(x,y)  
>>> axes[0,0].bar([1,2,3],[3,4,5])  
>>> axes[0,1].barh([0.5,1.2,2.5],[0,1,2])  
>>> axes[1,0].axhline(0.45)  
>>> axes[0,1].axvline(0.65)  
>>> ax.fill(x,y,color='blue')  
>>> ax.fill_between(x,y,color='yellow')
```

2D Data

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

Customize Plot

Colors, Color Bars & Color Maps

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

Markers

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

Linestyles

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

Text & Annotations

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

MathText

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

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)  
>>> axes[1,1].quiver(y,z)  
>>> axes[0,1].streamplot(X,Y,U,V)
```

Data Distributions

```
>>> ax1.hist(y)  
>>> ax3.boxplot(y)  
>>> ax3.violinplot(z)  
  
>>> axes2[0].pcolor(data2)  
>>> axes2[0].pcolormesh(data)  
>>> CS = plt.contour(Y,X,U)  
>>> axes2[2].contourf(data1)  
>>> axes2[2]= ax.clabel(CS)
```

Limits, Legends & Layouts

Limits & Autoscaling
>>> ax.margins(x=0.0,y=0.1)
>>> ax.axis('equal')
>>> ax.set(xlim=[0,10.5],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
Set limits for x-axis

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

Set a title and x-and y-axis labels
No overlapping plot elements

Linestyles
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,ls='-.')
>>> plt.set(lines,color='r',linewidth=4.0)

Manually set x-ticks
Make y-ticks longer and go in and out

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

fig.tight_layout()

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

Save Plot

Save figures

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

Save transparent figures

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

Show Plot

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

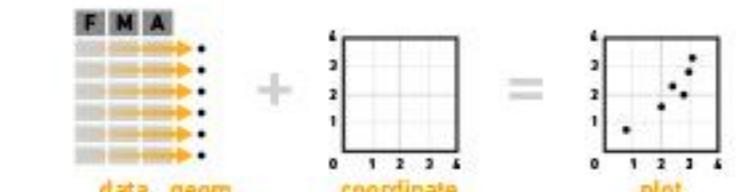
Close & Clear

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

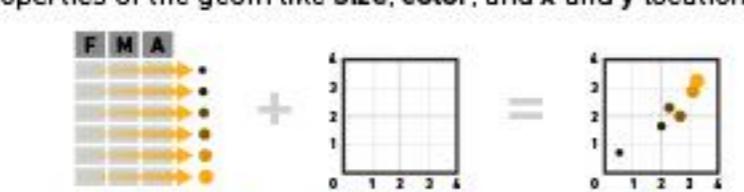
Data Visualisation with ggplot2 Cheat Sheet

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 `plot()` or `ggplot()`

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

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

data
ggplot(mpg, aes(hwy, cyl)) + add layers, elements with +
  geom_point(aes(color = cyl)) + layer = geom + default stat +
  geom_smooth(method = "lm") + layer = specific mappings +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw() + additional elements
```

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

`last_plot()`
Returns the last plot

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

Coordinate Systems

```
r <- b + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
Transformed cartesian coordinates. Set extras and strings 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), aequalares, lagrange, etc.)
```

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(x = density), stat = "bin")

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

a + geom_dotplot()
x, y, alpha, color, fill, shape, size

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

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

a + geom_text(label = cyl)
x, y, alpha, color, fill, linetype, size, weight
```

Discrete

```
b <- ggplot(mpg, aes(fct))
b + geom_bar()
x, alpha, color, fill, linetype, size, weight
```

Graphical Primitives

```
c <- ggplot(mpg, aes(long, lat))
c + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

d <- ggplot(economics, aes(date, unemploy))
d + geom_path(lineend = "butt",
  linejoin = "round", linemtire = 1)
x, y, alpha, color, linetype, size

d + geom_ribbon(aes(ymin = unemploy - 900,
  ymax = unemploy + 900))
x, y, alpha, color, fill, linetype, size

d <- ggplot(economics, aes(date, unemploy))
d + geom_segment(aes(
  xend = long + delta_long,
  yend = lat + delta_lat))
x, y, alpha, color, linetype, size

d + geom_rect(aes(xmin = long, ymin = lat,
  xmax = long + delta_long,
  ymax = lat + delta_lat))
x, y, alpha, color, fill, linetype, size
```

Three Variables

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2))
m <- ggplot(seals, aes(long, lat))

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

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

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

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 of one or more discrete variables.

Set scales to let axis limits vary across facets
t + facet_grid(~ x, scales = "free")
x and y axis limits adjust to individual facets
  * "free_x" - x axis limits adjust
  * "free_y" - y axis limits adjust
```

```
t + theme(legend.position = "bottom")
Place legend at bottom, top, left, or right

t + guides(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.
```

Two Variables

Continuous X, Continuous Y

```
f <- ggplot(mpg, aes(cty, hwy))
f + geom_blank()
i + geom_litter()
x, y, alpha, color, fill, shape, size
f + geom_hex()
x, y, alpha, color, fill, size, weight

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

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

f + geom_rug(sides = "bl")
x, y, alpha, color, fill, linetype, size, weight

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

Discrete X, Continuous Y

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

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

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

Continuous Function

```
i <- ggplot(economics, aes(date, unemploy))
i + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
```

Visualizing error

```
di <- 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, y, ymax, ymin, alpha, color, fill, linetype, size, weight

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

k + geom_dotplot(binaxis = "y",
  stackdir = "center")
x, y, alpha, color, fill, linetype, size

k + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, shape, size
```

Discrete X, Discrete Y

```
h <- ggplot(diamonds, aes(cut, color))
h + geom_jitter()
x, y, alpha, color, fill, linetype, size, weight

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

Position Adjustments

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

```
s <- ggplot(mpg, aes(fl, fill = drv))
s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

s + geom_bar(position = "stack")
Stack elements on top of one another

f + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting
```

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

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

Labels

```
t + ggtitle("New Plot Title")
Add a main title above the plot

t + xlabel("New X label")
Change the label on the X axis

t + ylabel("New Y label")
Change the label on the Y axis

t + labs(title = "New title", x = "New x", y = "New y")
All of the above
```

Legends

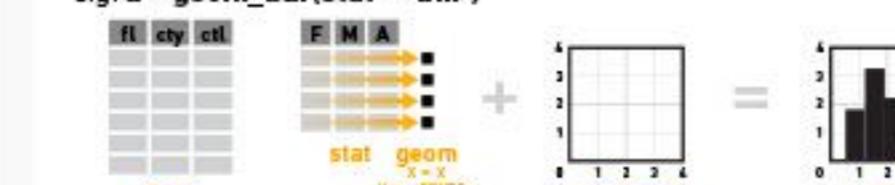
```
t + theme(legend.position = "bottom")
Place legend at bottom, top, left, or right

t + guides(color = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

t + scale_fill_discrete(name = "Title", labels = c("A", "B", "C"))
Set legend title and labels with a scale function.
```

Stats An alternative way to build a layer

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



Each stat creates additional variables to map aesthetics to. These variables use a common `.name` syntax. stat functions and geom functions both combine a stat with a geom to make a layer, i.e. `stat_bin(geom="bar")` does the same as `geom_bar(stat="bin")`

stat function layer specific mappings variable created by transformation
geom for layer parameters for stat

```
a + geom_bar(stat = "bin")
x, y, alpha, color, fill, linetype, size, weight
```

```
a + stat_bin(binwidth = 1, origin = 10)
x, y, ..count, ..density, ..density..
```

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

```
a + stat_density(adjust = 1, kernel = "gaussian")
x, y, ..count, ..density.., ..scaled..
```

```
i + stat_bin2d(bins = 30, drop = TRUE)
x, y, fill, ..count..
```

```
i + stat_hexbin(bins = 30)
x, y, fill, ..count..
```

```
i + stat_density2d(contour = TRUE, n = 100)
x, y, color, size, ..level..
```

```
m + stat_contour(z = z)
x, y, z, order, ..level..
```

```
m + stat_speckle(radius = z, angle = z)
angle, radius, x, xend, y, yend, x, xend, y, yend
```

```
m + stat_summary_hex(z = z, bins = 30, fun = mean)
x, y, z, fill, ..value..
```

```
g + stat_boxplot(coef = 1.5)
x, y, lower, ..middle, ..upper, ..outliers..
```

```
g + stat_idensity(adjust = 1, kernel = "gaussian", scale = "area")
x, y, ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..
```

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

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

```
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80,
  fullrange = FALSE, level = 0.95)
x, y, ..se.., ..x.., ..y.., ..ymin.., ..ymax..
```

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

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

```
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80,
  fullrange = FALSE, level = 0.95)
x, y, ..se.., ..x.., ..y.., ..ymin.., ..ymax..
```

```
ggplot() + stat_function(aes(x = -3:3),
  fun = dnorm, n = 101, args = list(sd=0.5))
x, y, ..x.., ..y..
```

```
f + stat_identity()
ggplot() + stat_qq(aes(sample = 1:100), distribution = qt,
  dparams = list(df=5))
sample, x, y, ..x.., ..y..
```

```
f + stat_sum()
x, y, size, ..size..
```

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

```
f + stat_unique()

```

Themes

geom point **geom** shape **geom** summary
size area **size** shape **size** unique

0 □ 6 ▽ 12 ▨ 18 ◆ 24 ▲
1 ○ 7 □ 13 ▨ 19 ● 25 ▲
2 △ 8 □ 14 ▨ 20 ● *
3 + 9 △ 15 ■ 21 ○
4 × 10 □ 16 ● 22 □ 0 ○
5 ◇ 11 □ 17 ▲ 23 ◇ 0 ○

scale shape manual
values = (3:7)
Value mapped to shape
Value mapped to shape in chart on right

q + f + geom_point(aes(size = cyl))
Value mapped to area of circle (not radius)

geom point **geom** shape manual
Value mapped to area of circle (not radius)

ggthemes - Package with additional ggplot2 themes

Scales

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

n <- b + geom_bar(aes(fill = fl))
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 aesthetic to adjust packaged scale to use
range of values to include in mapping scale to use in legend/axis labels to use in legend/axis

range of values to include in mapping scale to use in legend/axis labels to use in legend/axis

General Purpose scales

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

scale_*continuous() - map cont values to visual values

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

scale_*identity() - use data values as visual values

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

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"),
 breaks = date_breaks("2 weeks"))
 - treat 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

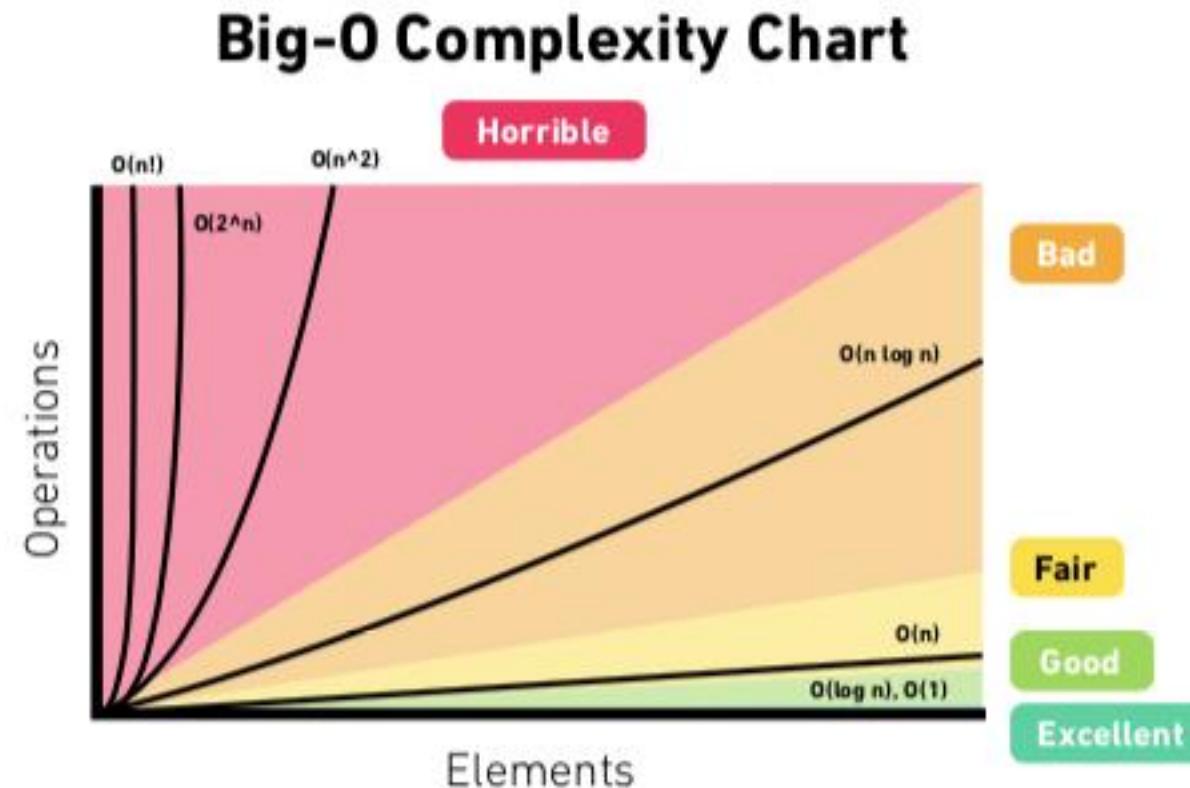
scale_x_reverse() - Reverse direction of x axis

scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Use with x or y aesthetics (x shown here)

n <- b + geom_bar(aes(fill = fl))
n + a + geom_dotplot(aes(fill = ..x..))



Data Structure Operation										Array Sorting Algorithms					
Time Complexity										Space Complexity					
Average					Worst					Worst					
Access	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	Access	Quicksort	Ω($n \log(n)$)	Θ($n \log(n)$)	O(n^2)	O($n \log(n)$)	
Stack	Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Mergesort	Ω($n \log(n)$)	Θ($n \log(n)$)	O($n \log(n)$)	O($n \log(n)$)	
Queue	Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Timsort	Ω(n)	Θ($n \log(n)$)	O($n \log(n)$)	Θ(n)	
Singly-Linked List	Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Heapsort	Ω($n \log(n)$)	Θ($n \log(n)$)	O($n \log(n)$)	O($n \log(n)$)	
Doubly-Linked List	Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Bubble Sort	X	Ω(n)	Θ(n^2)	O(n^2)	Θ(n)
Skip List	Skip List	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ(n)	Θ(n)	Θ(n)	Θ($n \log(n)$)	Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	Θ(n)	
Hash Table	Hash Table	X	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	Θ(n)	Θ(n)	Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	Ω(n^2)	
Binary Search Tree	Binary Search Tree	X	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ(n)	Θ(n)	Θ(n)	Tree Sort	Ω($n \log(n)$)	Θ($n \log(n)$)	O(n^2)	O($n \log(n)$)	
Cartesian Tree	Cartesian Tree	X	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	N/A	Θ(n)	Θ(n)	Shell Sort	Ω($n \log(n)$)	Θ($n(\log(n))^2$)	O($n(\log(n))^2$)	O($n \log(n)$)	
B-Tree	B-Tree	XX	N/A	Θ($\log(n)$)	Bucket Sort	Ω($n+k$)	Θ($n+k$)	O(n^2)	Ω($n+k$)						
Red-Black Tree	Red-Black Tree	X	Θ($\log(n)$)	Radix Sort	Ω($n+k$)	Θ($n+k$)	Ω($n+k$)	Ω($n+k$)							
Splay Tree	Splay Tree	X	N/A	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	N/A	Θ($\log(n)$)	Θ($\log(n)$)	Counting Sort	Ω($n+k$)	Θ($n+k$)	Ω($n+k$)	Ω($n+k$)	
AVL Tree	AVL Tree	X	Θ($\log(n)$)	Cubesort	Ω(n)	Θ($n \log(n)$)	O($n \log(n)$)	O($n \log(n)$)							
KD Tree	KD Tree	X	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ($\log(n)$)	Θ(n)	Θ(n)	Θ(n)						