

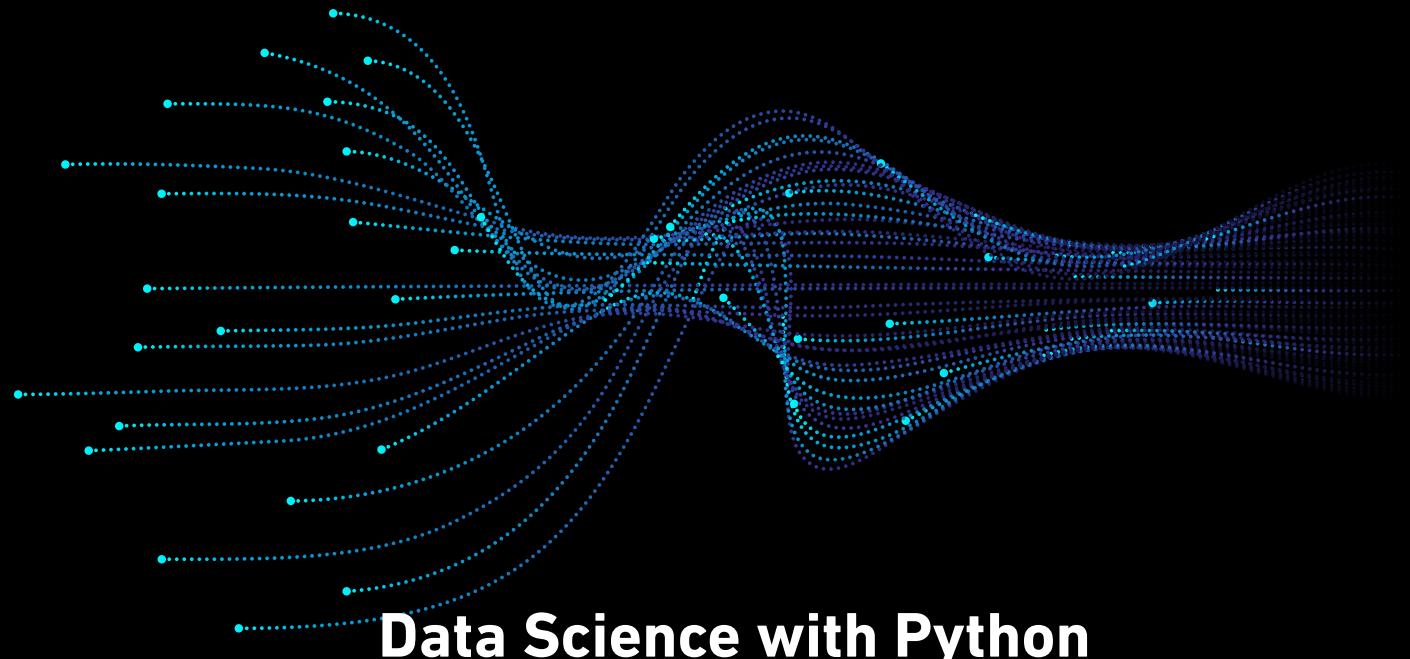
# Cheat Sheets for AI

## Neural Networks, Machine Learning, DeepLearning & Big Data

**The Most Complete List  
of Best AI Cheat Sheets**

BecomingHuman.AI

# Table of Content



## Data Science with Python

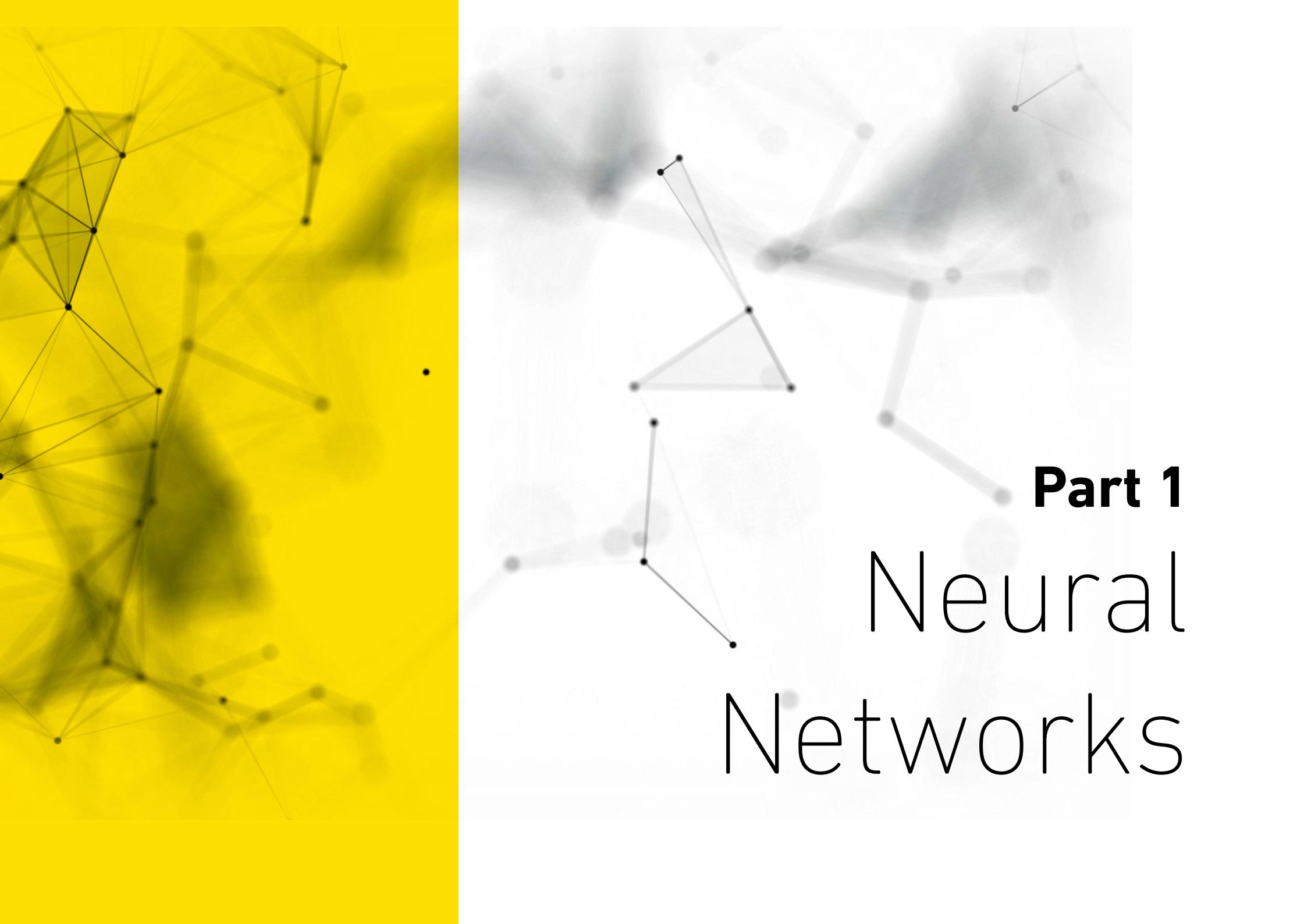
### Neural Networks

- 03 Neural Networks Basics
- 04 Neural Network Graphs

### Machine Learning

- 06 Machine Learning Basics
- 07 Scikit Learn with Python
- 08 Scikit Learn Algorithm
- 09 Choosing ML Algorithm

- |    |                |    |                                   |
|----|----------------|----|-----------------------------------|
| 11 | Tensor Flow    | 17 | Pandas                            |
| 12 | Python Basics  | 18 | Data Wrangling with Pandas        |
| 13 | PySpark Basics | 19 | Data Wrangling with dplyr & tidyr |
| 14 | Numpy Basics   | 20 | SciPi                             |
| 15 | Bokeh          | 21 | MatPlotLib                        |
| 16 | Karas          | 22 | Data Visualization with ggplot    |
| 23 | Big-O          |    |                                   |

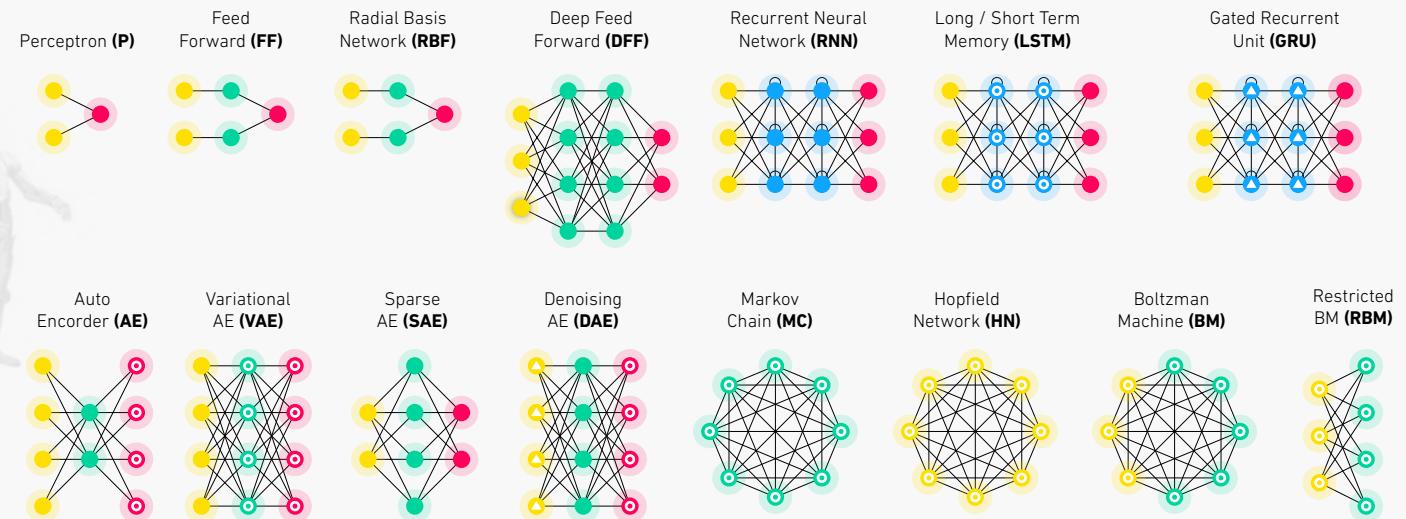


**Part 1**

Neural  
Networks

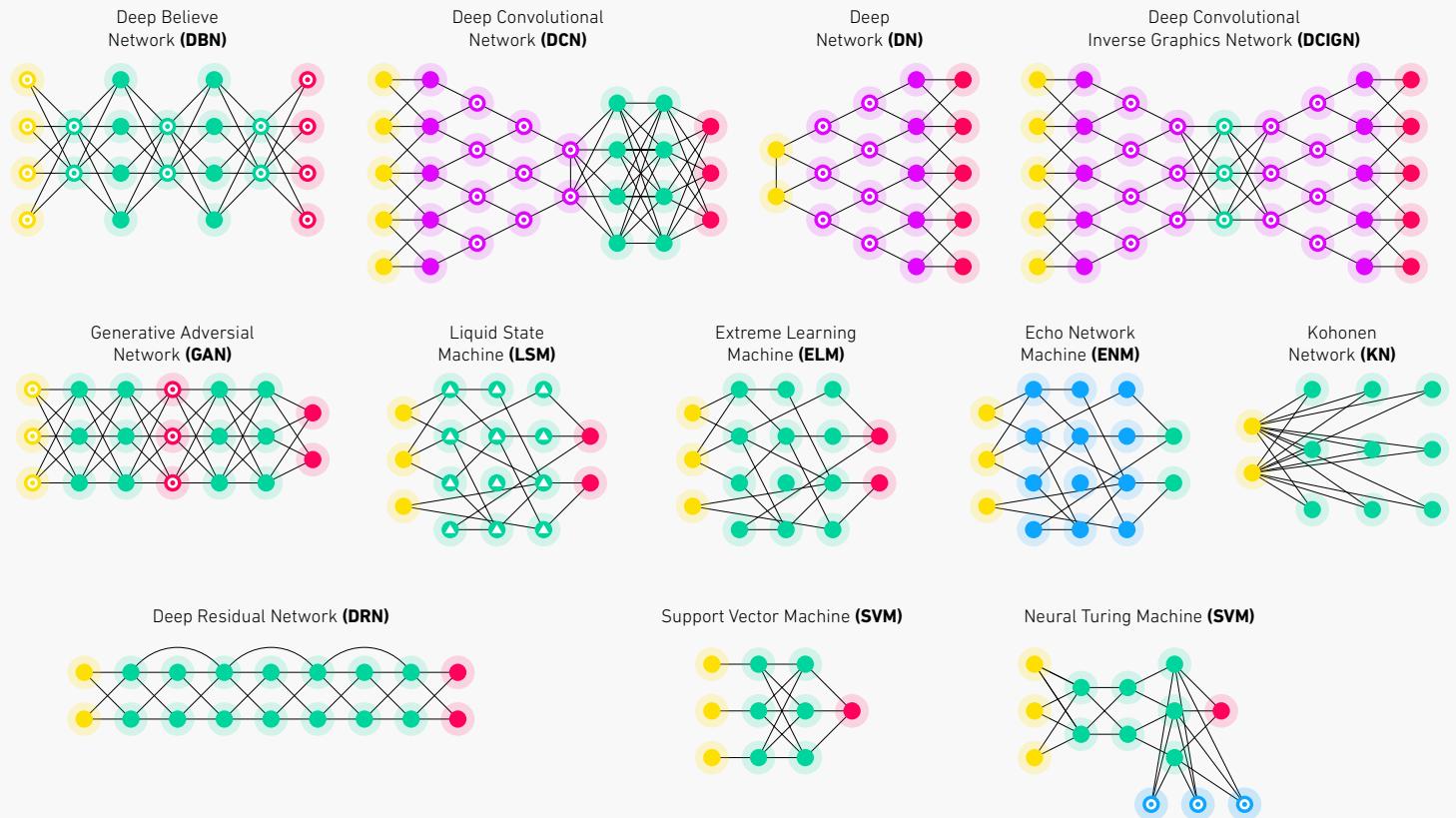
# Neural Networks Basic Cheat Sheet

BecomingHuman.AI



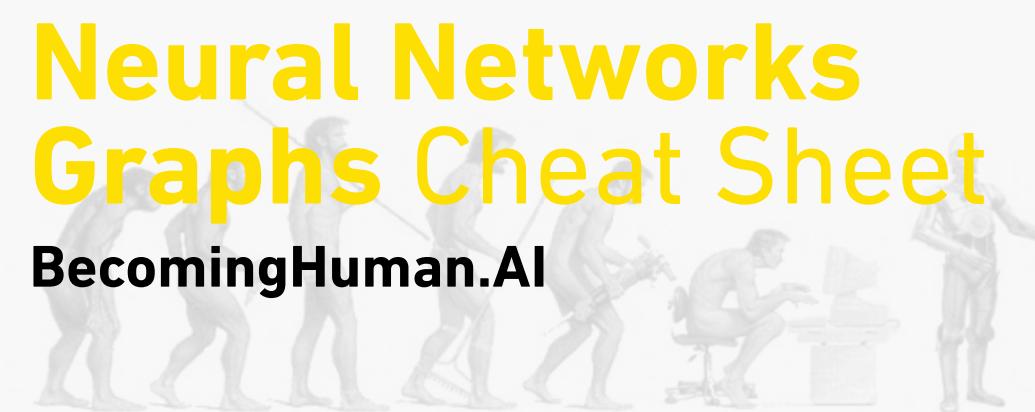
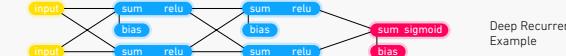
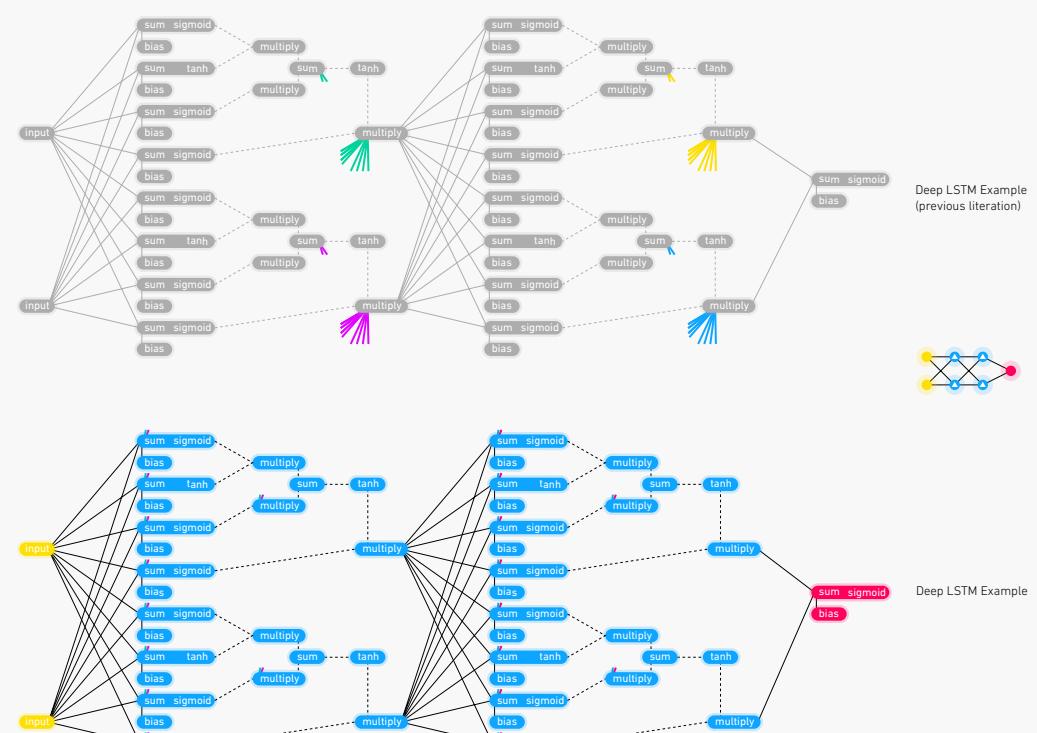
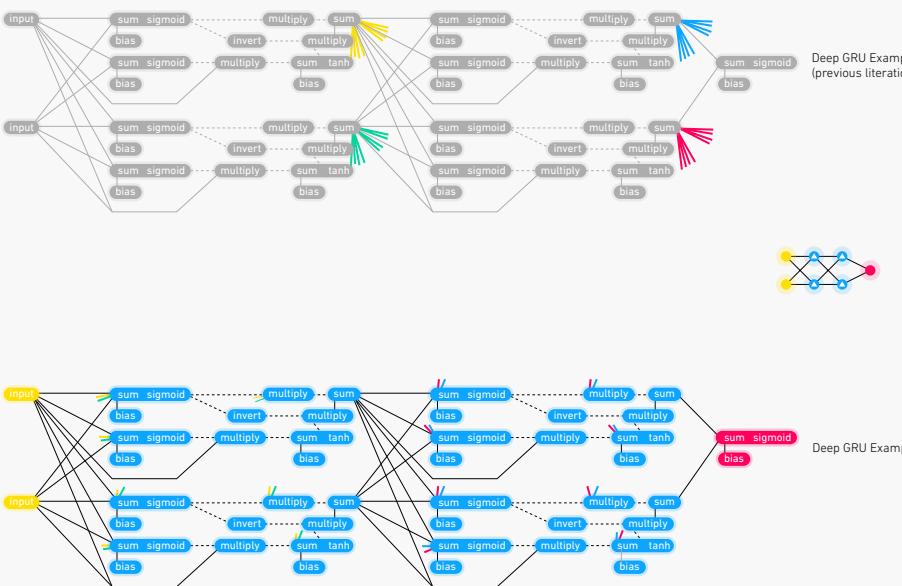
## Index

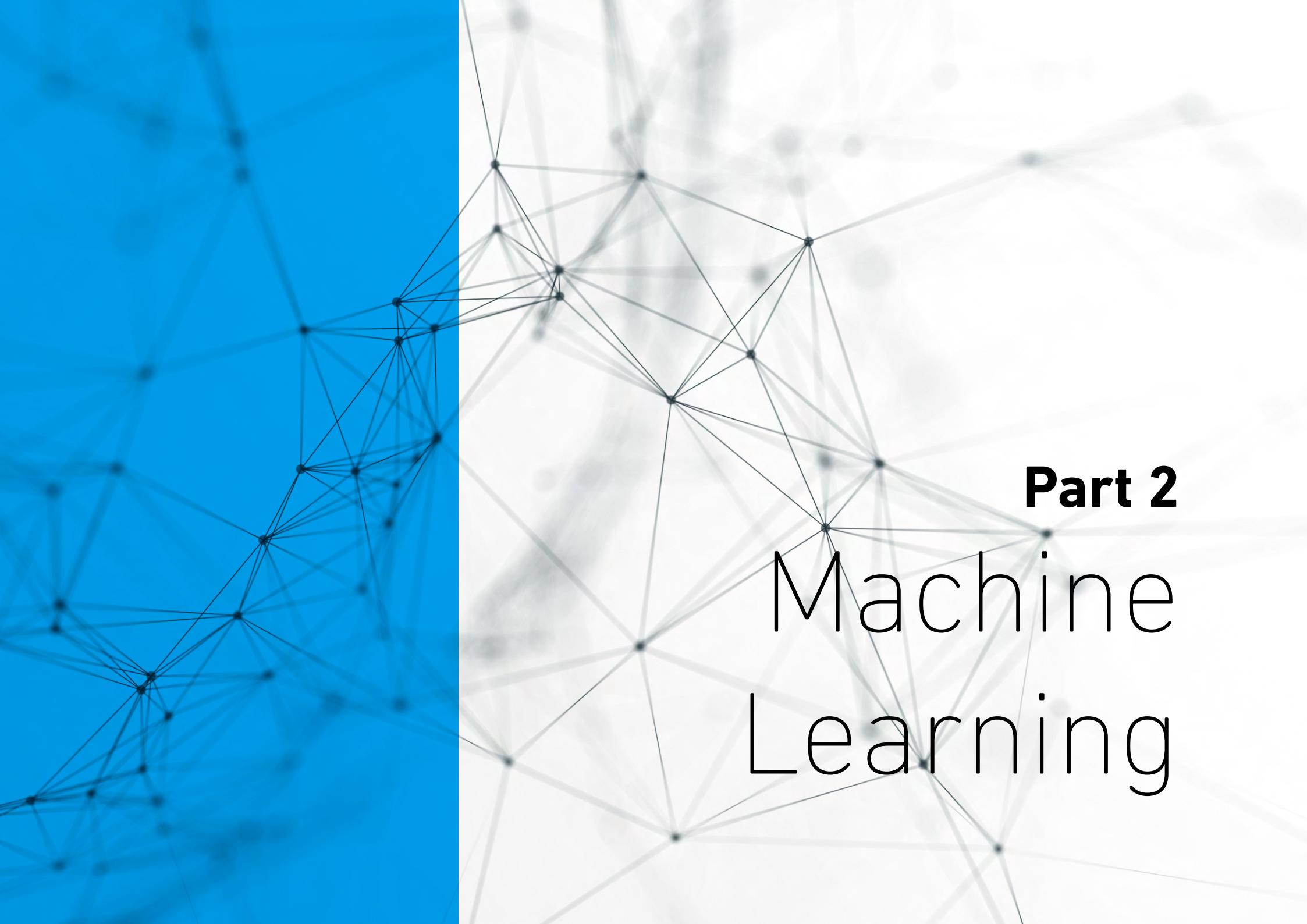
- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolutional or Pool



# Neural Networks Graphs Cheat Sheet

BecomingHuman.AI



The background of the slide features a complex network graph composed of numerous small, semi-transparent black dots connected by thin gray lines, creating a sense of data connectivity and structure.

# Part 2

# Machine Learning

# MachineLearning Overview

## MACHINE LEARNING IN EMOJI

### BecomingHuman.AI

#### SUPERVISED

#### UNSUPERVISED

#### REINFORCEMENT

#### BASIC REGRESSION

**LINEAR**  
`linear_model.LinearRegression()`  
Lots of numerical data

**LOGISTIC**  
`linear_model.LogisticRegression()`  
Target variable is categorical

human builds model based on input / output

human input, machine output  
human utilizes if satisfactory

human input, machine output  
human reward/punish, cycle continues

#### 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**  
`lda.LDA()`

Linear combination of features that separates classes

#### OTHER IMPORTANT CONCEPTS

**BIAS VARIANCE TRADEOFF**

**UNDERFITTING / OVERFITTING**

**INERTIA**

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

**Precision Function**  
`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, Xtest, ytrain, ytest = train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(ytest, y_pred)
```

## Prediction

### Supervised Estimators

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

Predict labels  
Predict labels  
Estimate probability of a label

### Unsupervised Estimators

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

Predict labels in clustering algos

## Loading the Data

Your data needs to be nmueric and stored as NumPy arrays or SciPy sparse matric. 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', 'IM', 'F', 'F', 'M', 'F', 'NI', 'tv1', '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(ytest, y_pred)
```

Estimator score method  
Metric scoring functions

#### Classification Report

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

Precision, recall, f1-score and support

#### Confusion Matrix

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

#### R<sup>2</sup> Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(ytrue, ypred)
```

### Clustering Metrics

#### Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(ytrue, ypred)
```

#### Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(ytrue, ypred)
```

#### V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(ytrue, ypred)
```

### 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

```
>>> kmeans.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
>>> kmeans = KMeans(n_clusters=3, random_state=0)
```

## Training And Test Data

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

## Tune Your Model

### Grid Search

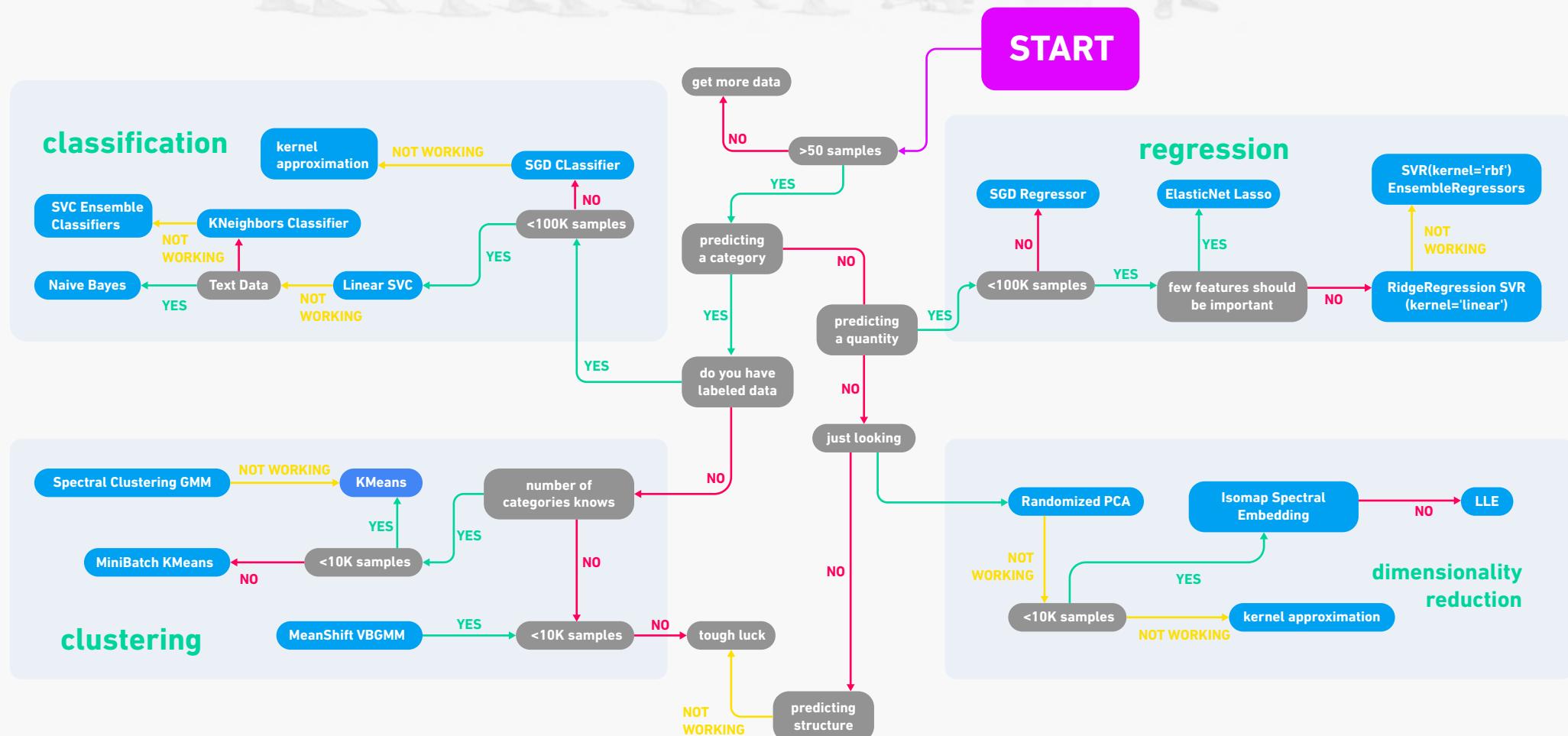
```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {'n_neighbors': np.arange(1,3),
...            'metric': ['euclidean','cityblock']}
>>> grid = GridSearchCV(estimator=knn,
...                      param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)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

### 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

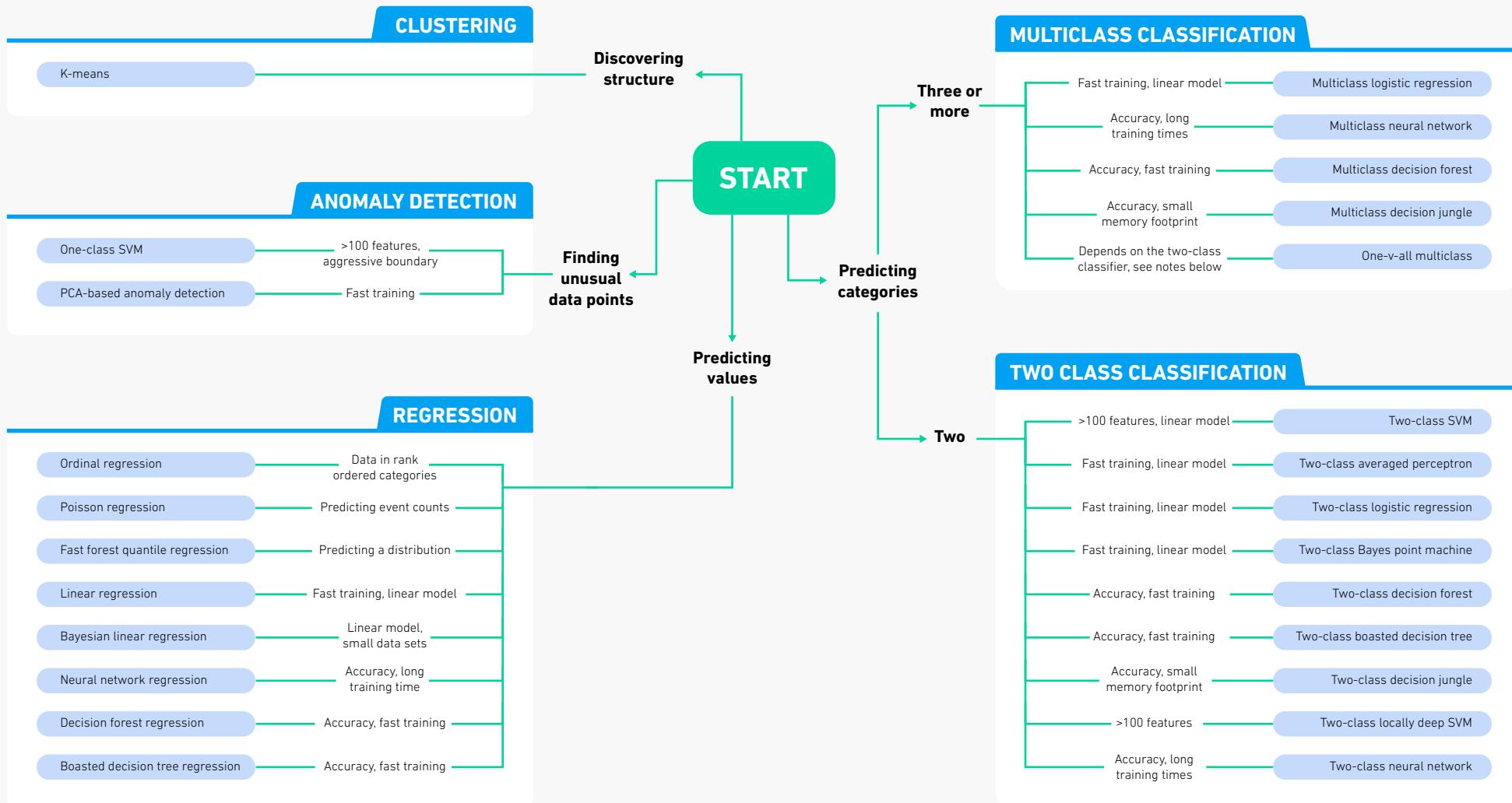
BecomingHuman.AI

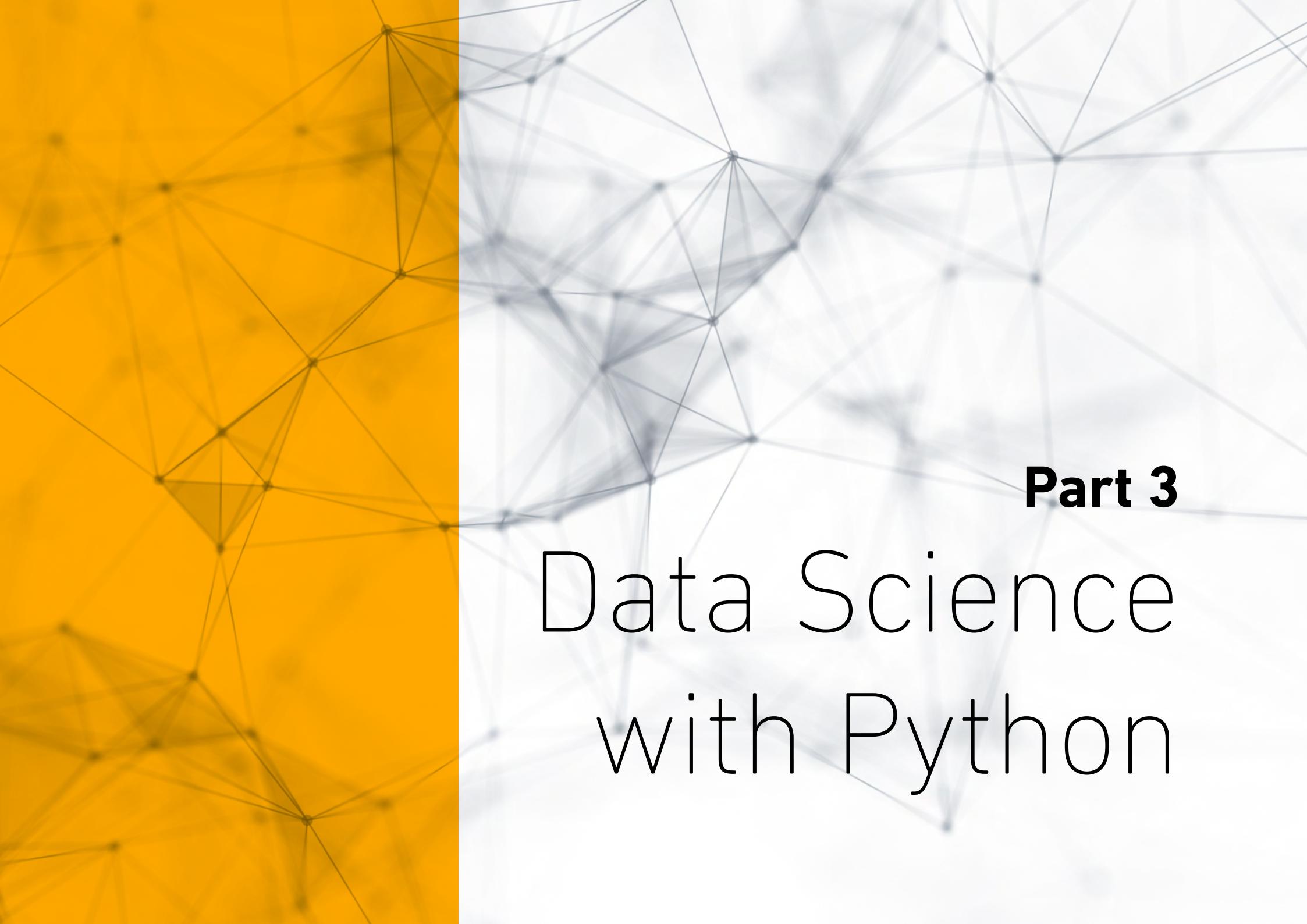


# Algorithm Cheat Sheet

## 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 network graph composed of numerous small, semi-transparent grey dots connected by thin grey lines, creating a sense of data connectivity and structure. The left side of the graph is set against a solid yellow background, while the right side is on a white background.

**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]
```

#### Slice

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

```
>>> my_list[-3:]
```

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

```
>>> my_list[:]
```

#### Subset Lists of Lists

```
>>> my_list2[1][0]
```

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

Select item at index 1  
Select 3rd last item

Select items at index 1 and 2  
Select items after index 0

Select items before index 3

Copy my\_list

my\_list[[list][itemOfList]]

### List Operations

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

### List Methods

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

Get the index of an item

Count an item

Append an item at a time

Remove an item

Remove an item

Reverse the list

Append an item

Remove an item

Insert an item

Sort the list

## Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]  
>>> my_array = np.array(my_list)  
>>> my_2darray =  
np.array([[1,2,3],[4,5,6]])
```

### Selecting Numpy Array Elements

Index starts at 0

#### Subset

```
>>> my_array[1]
```

```
2
```

#### Slice

```
>>> my_array[0:2]
```

```
array([1, 2])
```

#### Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
```

```
array([1, 4])
```

Select item at index 1

Select items at index 0 and 1

Select items at index 0 and 1

### 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 + 'Inninit'  
'thisStringIsAwesomeInninit'  
>>> 'm' in my_string  
True
```

### String Operations

```
>>> 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] --pyfiles 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 `--pyfiles`.

### 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()	Return a list with all RDD elements
>>> rdd.take(2)	Take first 2 RDD elements
>>> rdd.first()	Take first RDD element
>>> rdd.top(2)	Take top 2 RDD elements

#### Sampling

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

Return sampled subset of rdd3

#### Filtering

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

### Iterating

#### Getting

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

### Retrieving RDD Information

#### Basic Information

>>> rdd.getNumPartitions()	List the number of partitions
>>> rdd.count()	Count RDD instances
>>> rdd.countByKey()	Count RDD instances by key
>>> rdd.countByValue()	Count RDD instances by value
>>> rdd.collectAsMap()	Return (key,value) pairs as a dictionary
>>> rdd3.sum()	Sum of RDD elements
>>> 4950	Sum of RDD elements
>>> sc.parallelize().isEmpty()	Check whether RDD is empty
True	

#### Summary

>>> rdd3.max()	Maximum value of RDD elements
99	
>>> rdd3.min()	Minimum value of RDD elements
0	
>>> rdd3.mean()	Mean value of RDD elements
49.5	
>>> rdd3.stdev()	Standard deviation of RDD elements
28.86670047722118	
>>> rdd3.variance()	Compute variance of RDD elements
833.25	
>>> rdd3.histogram(3)	Compute histogram by bins
([0,33.66,99],[33,33,34])	
>>> rdd3.stats()	Summary statistics (count, mean, stdev, max & min)

### Applying Functions

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

### Mathematical Operations

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

### Sort

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

### Reshaping Data

#### Reducing

```
>>> rdd.reduceByKey(lambda x,y : x+y)  
.collect()
```

Merge the rdd values for each key

#### Grouping by

```
>>> rdd3.groupBy(lambda x: x % 2)  
.mapValues(list)  
.collect()
```

Return RDD of grouped values

Group rdd by key

#### Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))  
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))  
>>> rdd3.aggregate((0,0),seqOp,combOp)  
.collect()
```

Aggregate RDD elements of each partition and then the results

Aggregate values of each RDD key

```
>>> rdd3.aggregateByKey((0,0),seqOp,combOp)  
.collect()
```

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

Merge the values for each key

Create tuples of RDD elements by applying a function

### Reshaping Data

#### Repartition

```
>>> 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

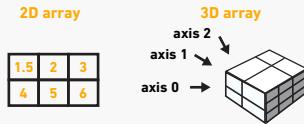
BecomingHuman.AI



1D array



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



## Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1,2,3), (4,5,6)], dtype = float)
>>> c = np.array([(1,2,3), (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),dtype=np.int16) Create an array of ones
>>> d = np.arange(10,25,5)      Create an array of evenly spaced values (step value)
>>> np.linspace(0,2,9)         Create an array of evenly spaced values (number of samples)
>>> e = np.full((2,2),7)        Create a constant array
>>> f = np.eye(2)              Create a 2X2 identity matrix
>>> np.random.random((2,2))    Create an array with random values
>>> np.empty((3,2))           Create an empty array
```

## I/O

### Saving & Loading On Disk

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

### Saving & Loading Text Files

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

## Inspecting Your Array

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

### Comparison

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

### Aggregate Functions

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

## Copying Arrays

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

```
>>> np.copy(a)
Create a copy of the array
```

```
>>> h = a.copy()
Create a deep copy of the array
```

## Sorting Arrays

```
>>> a.sort()
Sort an array
```

```
>>> c.argsort(axis=0)
Sort the elements of an array's axis
```

## Subsetting, Slicing, Indexing

### Subsetting

1 2 3	Select the element at the 2nd index
1.5 2 3 4 5 6	Select the element at row 1 column 2 (equivalent to b[1][2])

### Slicing

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

### Boolean Indexing

1 2 3	Select elements from a less than 2
-------	------------------------------------

### Fancy Indexing

1 2 3	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
```

### Combining Arrays

>>> np.concatenate((a,d),axis=0)	Concatenate arrays
>>> np.vstack((a,b))	Stack arrays vertically (row-wise)
>>> np.column_stack((a,d))	Create stacked column-wise arrays
>>> np.r_[e,f]	Stack arrays vertically (row-wise)
>>> np.hsplit(a,3)	Stack arrays horizontally (column-wise)
>>> np.vsplit(a,2)	Create stacked column-wise arrays
>>> np.c_[a,d]	Create stacked column-wise arrays



# Bokeh Cheat Sheet

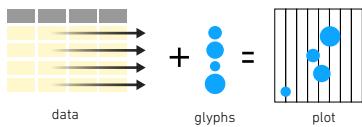
BecomingHuman.AI



## 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([[33,9,4,65, 'US'],
                               [32,4,4,66, 'Asia'],
                               [21,4,4,109, 'Europe']]),
                     columns=['mpg','cyl','hp','origin'],
                     index=['Toyota','Fiat','Volvo'])

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

## 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 columns
>>> layout = column(p1,p2,p3)
```

#### Nesting Rows & Columns

```
>>> layout = row(column(p1,p2),p3)
```

### Grid Layout

```
>>> from bokeh.layouts import gridplot
>>> row1 = [p1,p2]
>>> row2 = [p3]
>>> layout = gridplot([[p1,p2],[p3]])
```

### Legends

#### Legend Location

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

#### Inside 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

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



**K** 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(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

### Recurrent Neural Network (RNN)

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

## Inspect Model

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

## 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.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', 'Brasilia'],
   'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
   columns=['Country', 'Capital', 'Population'])
```

### Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=0)
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']
Get one element
-5
>>> df[1:]
Get subset of a DataFrame
   Country Capital    Population
1  India    New Delhi  1303171035
2  Brazil   Brasilia  207847528
```

#### Selecting, Boolean Indexing & Setting

##### By Position

```
>>> df.iloc[[0],[0]]
'Belgium'
>>> df.at[[0],[0]]
'Belgium'
```

Select single value by row & column

##### By Label

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

Select single value by row & column labels

##### By Label/Position

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

Select single row of subset of rows

##### Boolean Indexing

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

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

##### Setting

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

Set index a of Series s to 6

### 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'])
>>> s = s3
a 10.0
b NaN
c 5.0
d 7.0
```

#### Arithmetic Operations with Fill Methods

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

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

### 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

Spread rows into columns



Type	a	b	c
Date			
2016-03-01	11.432		
2016-03-02		13.031	
2016-03-03			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

Spread rows into columns



1	5	0	0.233482
2	4	0	0.184713
3	3	0	0.433522
4	3	1	0.429401

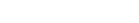
Stacked

### Melt

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

	Date	Type	Value	Observations
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	

Gather columns into rows



	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(df2.Type))]
>>> df3.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':'ctry',
                           'Capital':'cptl',
                           'Population':'pltn'})
```

Set the index  
Reset the index  
Rename DataFrame

## Reindexing

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

### Forward Filling

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

Forward Filling

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

0 3
1 3
2 3
3 3
4 3

## Multilabeling

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

### Forward Filling

```
>>> s3 = s.reindex(range(5),
                   method='bfill')
```

## 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

### data2

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

### Pivot

```
>>> pd.merge(data1,
             data2,
             how='left',
             on=X1)
```

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
d	NaN	NaN

### Join

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

### Concatenate

#### Vertical

```
>>> s.append(s2)
```

#### Horizontal/Vertical

```
>>> pd.concat([s,s2],axis=1,keys=[One,Two])
>>> pd.concat([data1,data2],axis=1,join='inner')
```

### Dates

```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1', periods=6,
                                freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1), end, freq='BM')
```

### Visualization

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

```
>>> s.plot()
>>> plt.show()
```

```
>>> df2.plot()
>>> plt.show()
```

# 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(
    [[1, 10],
     [5, 11],
     [6, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

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

```
df = pd.DataFrame(
    {"a": [4, 5, 6],
     "b": [7, 8, 9],
     "c": [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
        names=['n','v']))
```

Create DataFrame with a MultiIndex

## Method Chaining

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

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

## 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

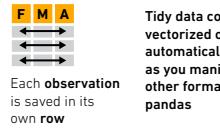
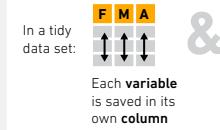
**df.plot.hist()**  
Histogram for each column



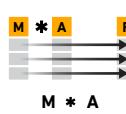
**df.plot.scatter(x='w',y='h')**  
Scatter chart using pairs of points



## Tidy Data A foundation for wrangling in pandas

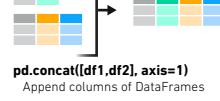
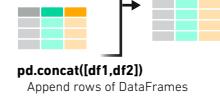
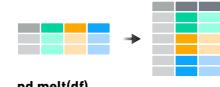


Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas



M \* A

## 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

## 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.

quantile(q=0.25, q=0.75)

Quantiles 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)

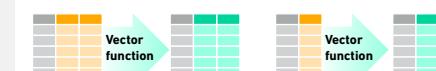
Compute and append one or more new columns.

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 altcolumns 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



### Set Operations

pd.merge(ydf, zdf)

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

pd.merge(ydf, zdf, how='outer')

Rows that appear in either or both ydf and zdf (Union).

pd.merge(ydf, zdf, how='outer', indicator=True)

.query('\_merge == "left\_only")

.drop(columns=['\_merge'])

Rows that appear in ydf but not zdf (Setdiff)



### Standard Joins

pd.merge(adf, bdf, how='left', on='x1')

Join matching rows from bdf to adf.

pd.merge(adf, bdf, how='right', on='x1')

Join matching rows from adf to bdf.

pd.merge(adf, bdf, how='inner', on='x1')

Join data. Retain only rows in both sets.

pd.merge(adf, bdf, how='outer', on='x1')

Join data. Retain all values, all rows.

### Filtering Joins

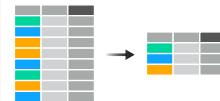
adf[x1.isin(bdf.x1)]

All rows in adf that have a match in bdf.

adf[~adfx1.isin(bdf.x1)]

All rows in adf that do not have a match in bdf

## Windows



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='dense')

Ranks with no gaps.

rank(method='min')

Ranks. Ties get min rank.

rank(pct=True)

Ranks rescaled to interval [0, 1].

rank(method='first')

Ranks. Ties go to first value.

shift(-1)

Copy with values lagged by 1.

cumsum()

Cumulative sum.

cummax()

Cumulative max.

cummin()

Cumulative min.

cumprod()

Cumulative product

# 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	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2
..	...	...	...	...
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)

Iris x					
	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(y, x, z)`

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

`iris %>%`

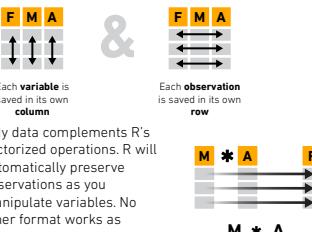
`group_by(Species) %>%`

`summarise(avg = mean(Sepal.Width)) %>%`

`arrange(avg)`

### Tidy Data A foundation for wrangling in R

#### In a tidy data set:



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

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



`tidy::gather(cases, "year", "n", 2:4)`

Gather columns into rows.



`tidy::spread(pollution, size, amount)`

Spread rows into columns.



`tidy::separate(storms, date, c("y", "m", "d"))`

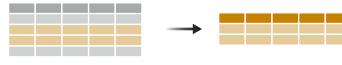
separate(storms, date, c("y", "m", "d"))



`tidy::unite(data, col, ..., sep)`

Unite several columns into one.

### Subset Observations (Rows)



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

Extract rows that meet logical criteria.

`dplyr::distinct(iris)`

Remove duplicate rows.

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

Randomly select fraction of rows.

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

Randomly select n rows.

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

Select rows by position.

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

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

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

### Subset Variables (Columns)



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

Select columns by name or helper function.

#### Helper functions for select - ?select

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

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches("t.*"))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of("Species", "Genus"))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepat"))`

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).



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

`dplyr::first`

First value of a vector.

`dplyr::last`

Last value of a vector.

`dplyr::nth`

Nth value of a vector.

`dplyr::n`

# of values in a vector.

`dplyr::n_distinct`

# of distinct values in a vector.

`dplyr::IQR`

IQR of a vector.

`dplyr::min`

Minimum value in a vector.

`dplyr::max`

Maximum value in a vector.

`dplyr::mean`

Mean value of a vector.

`dplyr::median`

Median value of a vector.

`dplyr::var`

Variance of a vector.

`dplyr::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



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

`dplyr::lead`

Copy with values shifted by 1.

`dplyr::cumall`

Cumulative all

`dplyr::cumany`

Cumulative any

`dplyr::dense_rank`

Ranks with no gaps.

`dplyr::min_rank`

Ranks. Ties get min rank.

`dplyr::percent_rank`

Ranks rescaled to [0, 1].

`dplyr::row_number`

Ranks. Ties get to first value.

`dplyr::ntile`

Bin vector into n buckets.

`dplyr::between`

Are values between a and b?

`dplyr::cume_dist`

Cumulative distribution.

`dplyr::max`

Element-wise max

`dplyr::min`

Element-wise min

### Combine Data Sets



`dplyr::left_join(a, b, by = "x1")`

Join matching rows from b to a.



`dplyr::right_join(a, b, by = "x1")`

Join matching rows from a to b.



`dplyr::inner_join(a, b, by = "x1")`

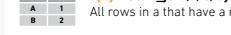
Join data. Retain only rows in both sets.



`dplyr::full_join(a, b, by = "x1")`

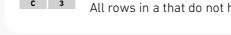
Join data. Retain all values, all rows.

### Filtering Joins



`dplyr::semi_join(a, b, by = "x1")`

All rows in a that have a match in b.



`dplyr::anti_join(a, b, by = "x1")`

All rows in a that do not have a match in b

### Group Data

`dplyr::group_by(iris, Species)`

Group data into rows with the same value of Species.

`dplyr::ungroup(iris)`

Remove grouping information from data frame.

`iris %>% group_by(Species) %>% summarise(...)`

Compute separate summary row for each group.

`iris %>% group_by(Species) %>% mutate(...)`

Compute new variables by group.



Creative Commons Data Wrangling with dplyr and tidyr by RStudio is licensed under CC BY SA 4.0

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,2,3), (4j,5j,6j)])
>>> c = np.array([(1,2,3), (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)>>>	Return the imaginary part of the array elements
np.real_if_close(c,tol=1000)	Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)	Cast object to a data type

#### Other Useful Functions

>>> np.angle(b,deg=True)	Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)	Create an array of evenly spaced values (number of samples)
>>> g[3:] += np.pi	
>>> np.unwrap(g)	Unwrap
>>> np.logspace(0,1,0,3)	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
>>> linalg.inv(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.fmm(A, lambda x: x*x) Evaluate matrix function
```

#### Sparse Matrix Routines

##### Inverse

```
>>> sparse.linalg.inv(l) Inverse
```

##### Norm

```
>>> sparse.linalg.norm(l) Norm
```

##### Solving linear problems

```
>>> sparse.linalg.spsolve(H,l) Solver for sparse matrices
```

#### Sparse Matrix Functions

```
>>> sparse.linalg.expm(l) Sparse matrix exponential
```

#### Decompositions

##### Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A) Solve ordinary or generalized eigenvalue problem for square matrix
```

```
>>> l, l2 = la 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)
```

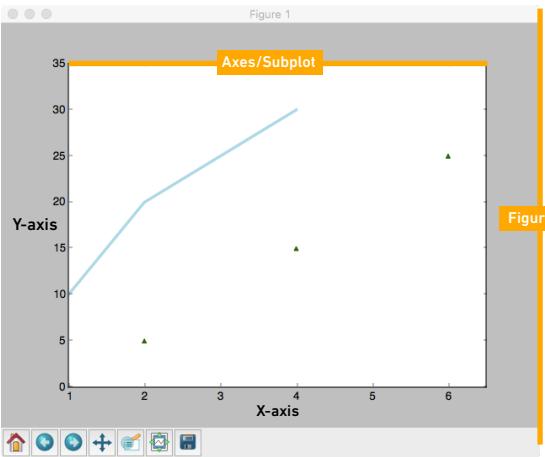
Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

# Matplotlib Cheat Sheet

BecomingHuman.AI

## Anatomy & Workflow

### Plot Anatomy



### Workflow

- |                        |                          |
|------------------------|--------------------------|
| <b>01</b> Prepare data | <b>04</b> Customize plot |
| <b>02</b> Create plot  | <b>05</b> Save plot      |
| <b>03</b> Plot         | <b>06</b> Show plot      |

```
>>> import matplotlib.pyplot as plt
step 1 >>> x = [1,2,3,4]
step 2 >>> y = [10,20,25,30]
step 3 >>> fig = plt.figure()
step 3,4 >>> 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='^')
step 5 >>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> 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[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

## 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[1,0].barh([0.5,1.2,2.5],[0,1,2])
>>> axes[1,1].axhline(0.45)
>>> axes[0,1].axvline(0.65)
>>> ax.fill(x,y,color='blue')
>>> ax.fill_between(x,y,color='yellow')
```

Draw points with lines or markers connecting them

Draw unconnected points, scaled or colored

Plot vertical rectangles (constant width)

Plot horizontal rectangles (constant height)

Draw a horizontal line across axes

Draw a vertical line across axes

Draw filled polygons

Fill between y-values and 0

Colormapped or RGB

### 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)
```

Add an arrow to the axes

Plot a 2D field of arrows

Plot 2D vector fields

### Data Distributions

```
>>> ax1.hist(y)
>>> ax3.boxplot(y)
>>> ax3.violinplot(z)
```

Plot a histogram

Make a box and whisker plot

Make a violin plot

Pseudocolor plot of 2D array

Pseudocolor plot of 2D array

Plot contours

Plot filled contours

Label a contour plot

## 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')
```

### Limits, Legends & Layouts

#### Limits & Autoscaling

```
>>> ax.margins(x=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

#### Legends

```
>>> 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

#### Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),
                ticklabels=[3,100,-12,'foo'],
                direction='inout',
                length=10)
```

Manually set x-ticks

Make y-ticks longer and go in and out

#### Subplot Spacing

```
>>> fig.subplots_adjust(wspace=0.5,
                        hspace=0.3,
                        left=0.25,
                        right=0.9,
                        top=0.9,
                        bottom=0.1)
```

Set subplot spacing

#### Axis Spines

```
>>> ax1.spines['top'].set_visible(False)
>>> ax1.spines['bottom'].set_position((0,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.clf()
```

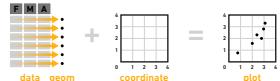
```
>>> plt.cla()
```

```
>>> plt.close()
```

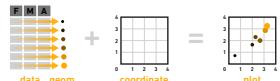
# Data Visualisation with ggplot2 Cheat Sheet

## 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 **qplot()** or **ggplot()**

**aesthetic mappings    data    geom**

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

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

**data**  
`ggplot(mpg, aes(hwy, 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)**  
ratio, xlim, ylim  
Equal aspect ratio coordinates with fixed aspect ratio between x and y units

**r + coord\_flip()**  
xmin, xmax, ymin, ymax  
Flipped Cartesian coordinates

**r + coord\_polar(theta = "x", direction = 1)**  
theta, start, direction  
Polar coordinates

**r + coord\_trans(xtrans = "sqrt")**  
xtrans, ytrans, xlim, ylim  
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

**r + coord\_map(projection = "orthographic", orientation = c(-74, 0))**  
projection, orientation, xlim, ylim  
Map projections from the maptools package (mercator (default), azequalarea, lagrange, etc.)

## Geoms

### One Variable

#### Continuous

```
a <- ggplot(mpg, aes(hwy))
a + geom_area(stat = "bin")
a + geom_density(kernel = "gaussian")
a + geom_dotplot()
a + geom_hex()
a + geom_histogram(binwidth = 5)
a + geom_rug(sides = "bl")
a + geom_smooth(method = lm)
a + geom_text(aes(label = cyl))
a + geom_text(stat = "direction" = "hv")
```

#### Discrete

```
b <- ggplot(mpg, aes(fct))
b + geom_bar()
```

## Graphical Primitives

c <- ggplot(mpg, aes(long, lat))

c + geom\_polygon(aes(group = group))

d <- ggplot(economics, aes(date, unemploy))

d + geom\_path(lineend = "butt",

linejoin = "round", lineinde = 1)

d + geom\_ribbon(ymin = unemploy - 900,

ymax = unemploy + 900)

d <- ggplot(economics, aes(date, unemploy))

e + geom\_segment(aes(x =

xend = long + delta\_long, yend = lat + delta\_lat))

e + geom\_rect(xmin = long, ymin = lat,

xmax = long + delta\_long, ymax = lat + delta\_lat))

geom\_text(aes(x = long, y = lat, label = date))

## Three Variables

sealsSz <- with(seals, sqrt(diamonds\$long^2 + diamonds\$lat^2))

m <- ggplot(seals, aes(long, lat))

m + geom\_contour(aes(z = z))

m + geom\_raster(aes(fill = z), hijust = 0.5,

viridis, interpolate = FALSE)

m + geom\_tile(aes(fill = z))

## 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, ~ fl)

facet into columns based on fl

t + facet\_grid(~ year ~ .)

facet into rows based on year

t + facet\_grid(~ fl \* year ~ .)

facet into both rows and columns

t + facet\_wrap(~ fl)

wrap facets into a rectangular layout of one column per discrete variable

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

Set labeller to adjust facet labels

t + facet\_grid(~ fl, labeller = label\_both)

t + facet

