

Python For Data Science Cheat Sheet

Keras

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



Keras

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

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.randint(1000,10000)
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
>                  activation='relu',
>                  input_dim=1000))
>>> model.add(Dense(1,activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
>                 loss='binary_crossentropy',
>                 metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

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

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
>>> minst,
>>> cifar10,
>>> imbd
>>> (x_train,y_train), (x_test,y_test) = minst.load_data()
>>> (x_train,y_train2), (x_test2,y_test2) = boston_housing.load_data()
>>> (x_train,y_train3), (x_test3,y_test3) = cifar10.load_data()
>>> (x_train,y_train4), (x_test4,y_test4) = imbd.load_data(nm_words=20000)
>>> num_classes = 10
```

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:-1]
>>> y = data[:, -1]
```

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_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model1 = 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),padding='same',input_shape=x_train.shape[1:]))
>>> 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'))
```

Also see NumPy & Scikit-Learn

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,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)
```

Inspect Model

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

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

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'])
```

Model Training

```
>>> model3.fit(x_train4,
>>> y_train4,
>>> batch_size=32,
>>> epochs=10,
>>> validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
>>> y_test,
>>> batch_size=32)
```

Prediction

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

Save/Reload Models

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

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
>                  optimizer=opt,
>                  metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
>>> y_train4,
>>> batch_size=32,
>>> epochs=10,
>>> validation_data=(x_test4,y_test4),
>>> callbacks=[early_stopping_monitor])
```

DataCamp

Learn Python for Data Science interactively



Python For Data Science Cheat Sheet

NumPy Basics

Learn Python for Data Science interactively at www.DataCamp.com



NumPy

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

Use the following import convention:

```
>>> import numpy as np
```



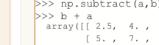
NumPy Arrays

1D array

axis 0

2D array

axis 0 axis 1

3D array

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), [(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
>>> np.empty((2,3,4))
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
```

Inspecting Your Array

```
>>> a.shape
Array dimensions
>>> len(a)
Length of array
>>> b.ndim
Number of array dimensions
>>> b.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
```

Asking For Help

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

Array Mathematics

Arithmetic Operations

>>> a = b	Subtraction
array([[-0.5, 0., 0., 0.], [-3., -3., -3.]])	Addition
>>> np.subtract(a,b)	Division
>>> b + a	Multiplication
array([[0.66666667, 1. , 1. , 1.], [0.25 , 0.4 , 0.5 , 0.5]])	Exponentiation
>>> np.divide(a,b)	Square root
>>> a * b	Print sines of an array
array([[1.5, 4., 9., 1.], [10., 18., 1.]]))	Element-wise cosine
>>> np.multiply(a,b)	Element-wise natural logarithm
>>> np.exp(b)	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
>>> a.corrcoef()
Correlation coefficient
>>> np.std(b)
Standard deviation
```

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

Also see Lists

Select the element at the 2nd index

Select the element at row 0 column 2 (equivalent to b[1][2])

Select items at index 0 and 1

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

Same as [1,:,:]

Reversed array

Select elements from a less than 2

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

Select a subset of the matrix's rows and columns

Subsetting, Slicing, Indexing

Subsetting

```
>>> a[2]
Select the element at the 2nd index
>>> b[1,2]
Select the element at row 0 column 2 (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]
Select items at index 0 and 1
>>> b[0:2,1]
Select items at rows 0 and 1 in column 1
```

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

Select elements from a less than 2

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

Select a subset of the matrix's rows and columns

Also see Lists

Permute array dimensions

Permute array dimensions

Flatten the array

Reshape, but don't change data

Return a new array with shape (2,6)

Append items to an array

Insert items in an array

Delete items from an array

Concatenate arrays

Stack arrays vertically (row-wise)

Stack arrays horizontally (column-wise)

Create stacked column-wise arrays

Create stacked column-wise arrays

Split the array horizontally at the 3rd index

Split the array vertically at the 2nd index

DataCamp

Learn Python for Data Science interactively



Data Wrangling

with pandas

Cheat Sheet

<http://pandas.pydata.org>

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



Each variable is saved in its own column



Each observation is saved in its own row

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



$M * A$

Syntax – Creating DataFrames

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

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

Specify values for each column.

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

Specify values for each row.

	n	a	b	c
d	1	4	7	10
e	2	5	8	11
f	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

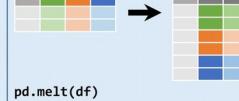
Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

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

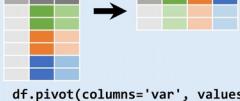
Method Chaining Examples

Reshaping Data

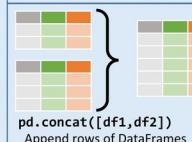
Reshaping Data – Change the layout of a data set



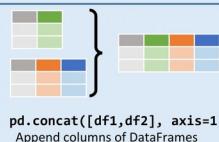
pd.melt(df)
Gather columns into rows.



df.pivot(columns='var', values='val')
Spread rows into columns.



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



pd.concat([df1, df2], axis=0)
Append columns of DataFrames

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

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

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

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(['Length','Height'], axis=1)
Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]
Extract rows that meet logical criteria.
df.drop_duplicates()
Remove duplicate rows (only considers columns).
df.head(n)
Select first n rows.
df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.
df.sample(n=10)
Randomly select n rows.
df.iloc[10:20]
Select rows by position.
df.nlargest(n, 'value')
Select and order top n entries.
df.nsmallest(n, 'value')
Select and order bottom n entries.

Subset Variables (Columns)



df[['width', 'length', 'species']]
Select multiple columns with specific names.
df['width'] or df.width
Select single column with specific name.

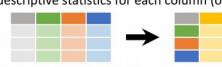
df.filter(regex='regex')
Select columns whose name matches regular expression regex.

regex (Regular Expressions) Examples
'\.'
'Length\$'
'Sepal'
'x[1-5]\$'
'[^Species]'."

df.loc[:, 'x2':'x4']
Select all columns between x2 and x4 (inclusive).
df.iloc[:, [1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).
df.loc[df['a'] > 10, ['a', 'c']]
Select rows meeting logical condition, and only the specific columns.

Summarize Data

```
df['w'].value_counts()
Count number of rows with each unique value of variable
len(df)
# of rows in DataFrame.
df['w'].nunique()
# of distinct values in a column.
df.describe()
Basic descriptive statistics for each column (or GroupBy)
```



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

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

Group Data

```
df.groupby(by='col')
Return a GroupBy object, grouped by values in column named "col".

df.groupby(level='ind')
Return a GroupBy object, grouped by values in index level named "ind".
```

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

size() Size of each group.

agg(function) Aggregate group using function.

Windows

```
df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.
```

Handling Missing Data

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

df.fillna(value)
Replace all NA/null data with value.

Make New Columns

df.assign(Area=lambda df: df.Length*df.Height)
Compute and append one or more new columns.

df['Volume'] = df.Length*df.Height*df.Depth
Add single column.

pd.qcut(df.col, n, labels=False)
Bin column into n buckets.

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

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

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

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

Combine Data Sets

adf + bdf

Standard Joins

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

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

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

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

Filtering Joins

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

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

ydf + zdf

Set-like Operations

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

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

pd.merge(ydf, zdf, how='outer', indicator=True)
.query('_merge == "left_only")
.drop(['_merge'], axis=1)

Rows that appear in ydf but not zdf (Setdiff).

Plotting

df.plot.hist()
Histogram for each column

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

Python For Data Science Cheat Sheet

Pandas Basics

Learn Python for Data Science [Interactively](#) at [www.DataCamp.com](#)



Pandas

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



Use the following import convention:

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

Pandas Data Structures

Series

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



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

DataFrame

Columns

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

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'])
```

I/O

Read and Write to CSV

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

```
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
```

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

Read multiple sheets from the same file

```
>>> xls = pd.ExcelFile('file.xls')
```

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

Asking For Help

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

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
```

Get one element

```
>>> df[1:]
```

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0], [0]]
```

Select single value by row & column

```
'Belgium'
```

```
>>> df.iat[[0], [0]]
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
```

Select single row of subset of rows

```
Country Brazil
```

```
Capital Brasilia
```

```
Population 207847528
```

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

Select a single column of subset of columns

```
0 Brussels
```

```
1 New Delhi
```

```
2 Brasilia
```

```
>>> df.ix[1, 'Capital']
```

Select rows and columns

```
'New Delhi'
```

Boolean Indexing

```
>>> s[s > 1]
```

Series s where value is not > 1

```
>>> s[s < -1] | (s > 2)
```

s where value is <-1 or > 2

```
>>> df[df['Population'] > 1200000000]
```

Use filter to adjust DataFrame

Setting

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

Set index a of Series s to 6

Dropping

```
>>> s.drop(['a', 'c'])
```

Drop values from rows (axis=0)

```
>>> df.drop('Country', axis=1)
```

Drop values from columns (axis=1)

Sort & Rank

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

Sort by row or column index

```
>>> s.order()
```

Sort a series by its values

```
>>> df.rank()
```

Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
```

(rows,columns)

```
>>> df.index
```

Describe index

```
>>> df.columns
```

Describe DataFrame columns

```
>>> df.info()
```

Info on DataFrame

```
>>> df.count()
```

Number of non-NA values

Summary

```
>>> df.sum()
```

Sum of values

```
>>> df.cumsum()
```

Cumulative sum of values

```
>>> df.min() / df.max()
```

Minimum/maximum values

```
>>> df.idmin() / df.idmax()
```

Minimum/Maximum index value

```
>>> df.describe()
```

Summary statistics

```
>>> df.mean()
```

Mean of values

```
>>> df.median()
```

Median of values

Applying Functions

```
>>> f = lambda x: x*x
```

Apply function

```
>>> df.apply(f)
```

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'])
```

```
>>> s3 + s
```

```
a 10.0
```

```
b NaN
```

```
c 5.0
```

```
d 7.0
```

Arithmetic Operations with Fill Methods

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

```
>>> s.add(s3, fill_value=0)
```

```
a 10.0
```

```
b -5.0
```

```
c 5.0
```

```
d 7.0
```

```
>>> s.sub(s3, fill_value=2)
```

```
>>> s.div(s3, fill_value=4)
```

```
>>> s.mul(s3, fill_value=3)
```

DataCamp

Learn Python for Data Science [Interactively](#)



Python For Data Science Cheat Sheet

SciPy - Linear Algebra

Learn More Python for Data Science [Interactively](#) at [www.datacamp.com](#)



SciPy

The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.



Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
```

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

```
>>> b = np.array([(1+5),2,3])
```

```
>>> c = np.array([(1,2,3), (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]
```

Stack arrays horizontally (column-wise)

```
>>> np.vsplit(d,2)
```

Split the array horizontally at the 2nd index

```
>>> np.hsplit(c,2)
```

Split the array vertically at the 2nd index

```
>>> np.dsplit(d,2)
```

Split the array vertically at the 2nd index

```
>>> np.transpose(b)
```

Permute array dimensions

```
>>> b.flatten()
```

Flatten the array

```
>>> np.vstack((a,b))
```

Stack arrays vertically (column-wise)

```
>>> np.vstack((a,b))
```

Stack arrays vertically (row-wise)

```
>>> np.hsplit(c,2)
```

Split the array vertically (row-wise)

```
>>> np.vsplit(c,2)
```

Split the array horizontally (column-wise)

```
>>> np.hsplit(c,2)
```

Split the array horizontally (row-wise)

```
>>> np.concatenate((a,b))
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=1)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=0)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-1)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-2)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-3)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-4)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-5)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-6)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-7)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-8)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-9)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-10)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-11)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-12)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-13)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-14)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-15)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-16)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-17)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-18)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-19)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-20)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-21)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-22)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-23)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-24)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-25)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-26)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-27)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-28)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-29)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-30)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-31)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-32)
```

Concatenate arrays along a new axis

```
>>> np.concatenate((a,b),axis=-33)
```

Concatenate arrays along a new axis

Python For Data Science Cheat Sheet

Matplotlib

Learn Python Interactively at www.DataCamp.com



Matplotlib

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



1 Prepare The Data

Also see [Lists & NumPy](#)

1D Data

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

2D Data or Images

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

2 Create Plot

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

Figure

```
>>> fig = plt.figure()  
>>> fig2 = plt.figure(figsize=plt.rcParams['figure.figsize'])
```

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()  
>>> ax = fig.add_subplot(221) # row-col-num  
>>> ax2 = fig.add_subplot(212)  
>>> fig3, axes = plt.subplots(nrows=2, ncols=2)  
>>> fig4, axes2 = plt.subplots(ncols=3)
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()  
>>> lines = ax.plot(x,y)  
>>> ax.scatter(x,y)  
>>> ax.vlines([0,1,2], [3,4,5])  
>>> ax.patches([0,1,2], [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 horizontal rectangles (constant width)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and o

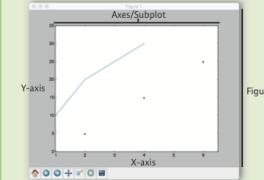
2D Data or Images

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

Colormapped or RGB arrays

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

- 1 Prepare data
- 2 Create plot
- 3 Plot
- 4 Customize plot
- 5 Save plot
- 6 Show plot

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

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, y, alpha=0.4)  
>>> ax.set_alpha(0.5)  
>>> 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,lw='solid')  
>>> plt.plot(x,y,'--',x**2,y**2,'-.')  
>>> plt.setp(lines,color='r', linewidth=4.0)
```

Text & Annotations

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

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.bxpplot(y)  
>>> ax3.violinplot(z)
```

Add an arrow to the axes
Plot a 2D field of arrows
Plot a 2D field of arrows

Plot a histogram
Make a box and whisker plot
Make a violin plot

Subplot Spacing

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

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)  
>>> ax1.spines['bottom'].set_position(('outward', 10))
```

Adjust the spacing between subplots

Fit subplot(s) to the figure area

Make the top axis line for a plot invisible

Move the bottom axis line outward

5 Save Plot

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

6 Show Plot

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

Close & Clear

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

Clear an axis

Clear the entire figure

Close a window

DataCamp

Learn Python for Data Science interactively



Python For Data Science Cheat Sheet

Scikit-Learn

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



Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.



A Basic Example

```
>>> from sklearn.neighbors import NeighborsClassifier  
>>> from sklearn.cross_validation import train_test_split  
>>> from sklearn.metrics import accuracy_score  
>>> iris = datasets.load_iris()  
>>> X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size=0.4, random_state=42)  
>>> knn = KNeighborsClassifier(n_neighbors=5)  
>>> knn.fit(X_train, y_train)  
>>> y_pred = knn.predict(X_test)  
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see [NumPy & Pandas](#)

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrames, are also acceptable.

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

Training And Test Data

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

Preprocessing The Data

Standardization

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

Normalization

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

Binarization

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

Create Your Model

Supervised Learning Estimators

Linear Regression

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

Support Vector Machines (SVM)

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

Naive Bayes

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

KNN

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

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
```

```
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
```

```
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

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

```
>>> svc.fit(X_train, y_train)
```

Unsupervised Learning

```
>>> k_means.fit(X_train)
```

```
>>> pca.fit_transform(X_train)
```

Fit the model to the data

Fit the model to the data

Fit the model to data, then transform it

Prediction

Supervised Estimators

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

```
>>> y_pred = knn.predict_proba(X_test)
```

Unsupervised Estimators

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

Predict labels

Predict labels

Estimate probability of a label

Predict labels in clustering algs

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
```

```
>>> enc = LabelEncoder()
```

```
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
```

```
>>> imp = Imputer(missing_values=np.nan, 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, f-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 Score

```
>>> 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))
```

cross_val_score(estimator=knn, cv=cv)

cross_val_score(estimator=lr, cv=cv)

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": np.arange(1,5),  
    "weights": ["uniform", "distance"]}
```

```
>>> research = RandomizedSearchCV(estimator=knn, param_distributions=params,
```

```
>>> n_iter=8, random_state=5)
```

```
>>> research.fit(X_train, y_train)
```

print(research.best_score_)

DataCamp

Learn Python for Data Science interactively



A mostly complete chart of
Neural Networks

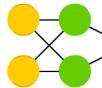
©2016 Fjodor van Veen - asimovinstitute.org

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

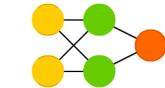
Perceptron (P)



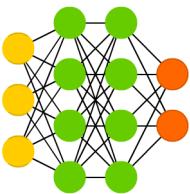
Feed Forward (FF)



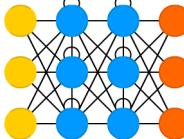
Radial Basis Network (RBF)



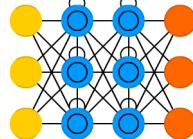
Deep Feed Forward (DFF)



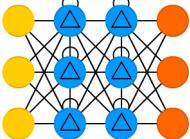
Recurrent Neural Network (RNN)



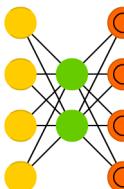
Long / Short Term Memory (LSTM)



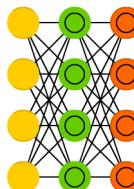
Gated Recurrent Unit (GRU)



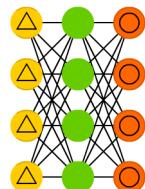
Auto Encoder (AE)



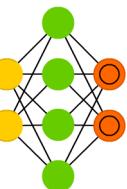
Variational AE (VAE)



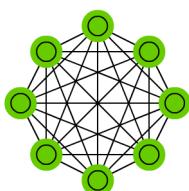
Denoising AE (DAE)



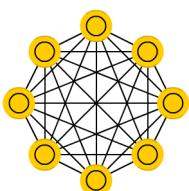
Sparse AE (SAE)



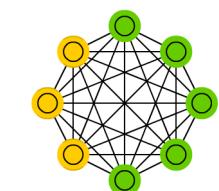
Markov Chain (MC)



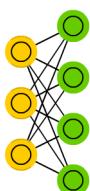
Hopfield Network (HN)



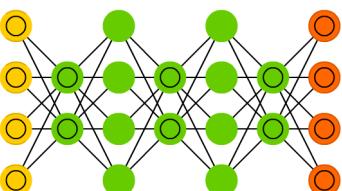
Boltzmann Machine (BM)



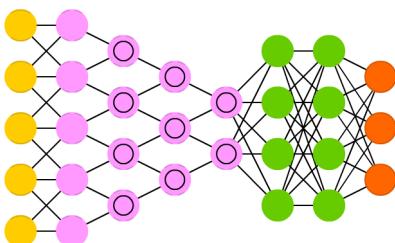
Restricted BM (RBM)



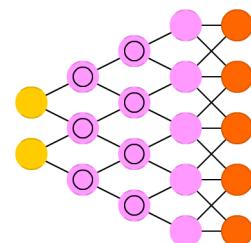
Deep Belief Network (DBN)



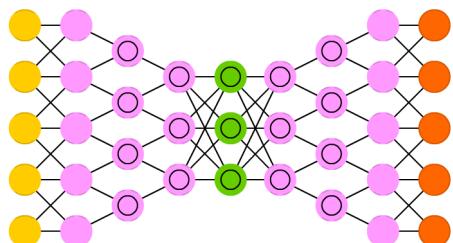
Deep Convolutional Network (DCN)



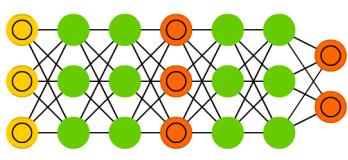
Deconvolutional Network (DN)



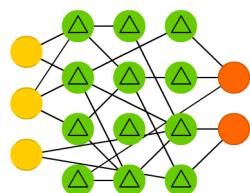
Deep Convolutional Inverse Graphics Network (DCIGN)



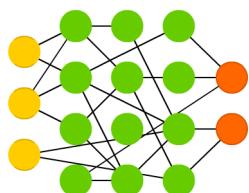
Generative Adversarial Network (GAN)



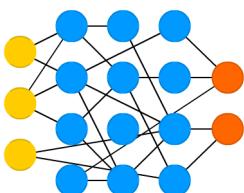
Liquid State Machine (LSM)



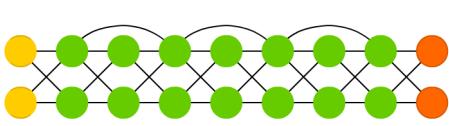
Extreme Learning Machine (ELM)



Echo State Network (ESN)



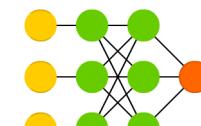
Deep Residual Network (DRN)



Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)

