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.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'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

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

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_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[:,0:8]
>>> 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.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

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

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

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=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

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

Prediction

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

Save/ Reload Models

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

Model Fine-tuning

Optimization Parameters

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

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

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)
```


Python For Data Science *Cheat Sheet*

NumPy Basics

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

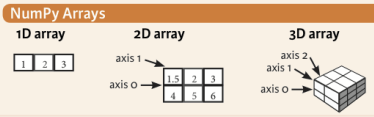
NumPy

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



Creating Arrays

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

Initial Placeholders

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

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.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=" ")
```

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 values
>>> np.object     Python object type
>>> np.string_    Fixed-length string type
>>> np.unicode_   Fixed-length unicode type
```

Inspecting Your Array

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

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

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]
 3
>>> b[1,2]
 6.0
```

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

Slicing

```
>>> a[0:2]
 array([1, 2])
>>> b[0:2,1]
 array([ 2.,  5.])
>>> b[:1]
 array([[1.5, 2., 3.]])
>>> c[1,...]
 array([[[ 3.,  2.,  1.],
         [ 4.,  5.,  6.]]])
>>> a[ : :-1]
 array([3, 2, 1])
```

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1
Select all items at row 0 (equivalent to `b[0:1, :]`)
Same as `[1,:,:]`
Reversed array `a`

Boolean Indexing

```
>>> a[a<2]
 array([1])
```

Select elements from `a` less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]
 array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]
 array([[ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5],
        [ 4. ,  5. ,  6. ,  4. ],
        [ 1.5,  2. ,  3. ,  1.5]])
```

Select elements `(1,0),(0,1),(1,2)` and `(0,0)`
Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))
>>> np.append(h,g)
>>> np.insert(a, 1, 5)
>>> np.delete(a,[1])
```

Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)
 array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))
 array([[ 1. ,  2. ,  3. ],
        [ 1.5,  2. ,  3. ],
        [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]
>>> np.hstack((e,f))
 array([[ 7.,  7.,  1.,  0.],
        [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))
 array([[ 1, 10],
        [ 2, 15],
        [ 3, 20]])
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays

Splitting Arrays

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

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index

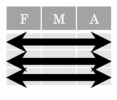
Data Wrangling with pandas Cheat Sheet

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.

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

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

Create DataFrame with a MultiIndex

Method Chaining

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

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

Reshaping Data – Change the layout of a data set

pd.melt(df)
Gather columns into rows.

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

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

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

df.sort_values('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).

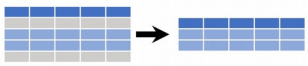
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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.

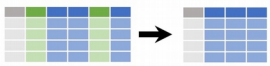
df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Subset Variables (Columns)



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

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

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

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

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

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

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

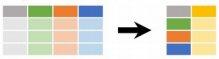
Summarize Data

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

len(df)
# of rows in DataFrame.

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

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



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

sum()
Sum values of each object.

count()
Count non-NA/null values of each object.

median()
Median value of each object.

quantile([0.25,0.75])
Quantiles of each object.

apply(*function*)
Apply function to each object.

min()
Minimum value in each object.

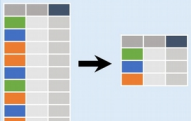
max()
Maximum value in each object.

mean()
Mean value of each object.

var()
Variance of each object.

std()
Standard deviation of each object.

Group Data



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

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

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

size()
Size of each group.

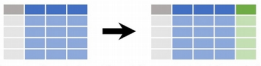
agg(*function*)
Aggregate group using function.

Handling Missing Data

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

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

Make New Columns



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

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

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

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

max(axis=1)
Element-wise max.

clip(lower=-10,upper=10)
Trim values at input thresholds

min(axis=1)
Element-wise min.

abs()
Absolute value.

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

shift(1)
Copy with values shifted by 1.

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.

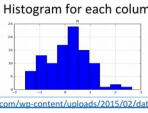
Windows

df.expanding()
Return an Expanding object allowing summary functions to be applied cumulatively.

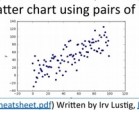
df.rolling(n)
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

df.plot.hist()
Histogram for each column



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



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

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

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

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

x1	x2	x3
A	1	T
B	2	F

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

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

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

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

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

x1	x2
A	1
B	2
C	3
D	4

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

x1	x2
A	1

```
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).

Python For Data Science *Cheat Sheet*

Pandas Basics

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

Pandas

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

Use the following import convention:

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

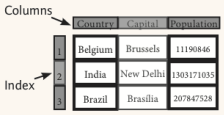
Pandas Data Structures

Series

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


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

DataFrame



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

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasilia'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

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

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
 -5

>>> df[1:]
   Country   Capital  Population
1   India  New Delhi  1303171035
2  Brazil  Brasilia   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

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

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
Capital     Brasilia
Population  207847528
```

Select single row of subset of rows

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

Select a single column of subset of columns

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

Select rows and columns

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

Dropping

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

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

Sort & Rank

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

Sort by row or column index
Sort a series by its values
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

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

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idmin()/df.idmax()
>>> 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

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

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

```
>>> 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)
```


Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

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

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]
>>> np.ogrid[0:2,0:2]
>>> np.r_[3,[0]*5,-1:1:10j]
>>> np.c_[b,c]
```

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

Shape Manipulation

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

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

Polynomials

```
>>> from numpy import polyld
>>> p = polyld([3,4,5])
```

Create a polynomial object

Vectorizing Functions

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

Vectorize functions

Type Handling

```
>>> np.real(b)
>>> np.imag(b)
>>> np.real_if_close(c,tol=1000)
>>> np.cast['f'](np.pi)
```

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

Other Useful Functions

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

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

Inverse
Inverse

Transposition

```
>>> A.T
>>> A.H
```

Tranpose matrix
Conjugate transposition

Trace

```
>>> np.trace(A)
```

Trace

Norm

```
>>> linalg.norm(A)
>>> linalg.norm(A,1)
>>> linalg.norm(A,np.inf)
```

Frobenius norm
L1 norm (max column sum)
L inf norm (max row sum)

Rank

```
>>> np.linalg.matrix_rank(C)
```

Matrix rank

Determinant

```
>>> linalg.det(A)
```

Determinant

Solving linear problems

```
>>> linalg.solve(A,b)
>>> E = np.mat(a).T
>>> linalg.lstsq(F,E)
```

Solver for dense matrices
Solver for dense matrices
Least-squares solution to linear matrix equation

Generalized inverse

```
>>> linalg.pinv(C)
>>> linalg.pinv2(C)
```

Compute the pseudo-inverse of a matrix (least-squares solver)
Compute the pseudo-inverse of a matrix (SVD)

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)
>>> G = np.mat(np.identity(2))
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)
>>> I = sparse.csc_matrix(D)
>>> J = sparse.dok_matrix(A)
>>> E.todense()
>>> sparse.isspmatrix_csc(A)
```

Create a 2X2 identity matrix
Create a 2x2 identity matrix
Compressed Sparse Row matrix
Compressed Sparse Column matrix
Dictionary Of Keys matrix
Sparse matrix to full matrix
Identify sparse matrix

Sparse Matrix Routines

Inverse

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

Inverse

Norm

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

Norm

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I)
```

Solver for sparse matrices

Sparse Matrix Functions

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

Sparse matrix exponential

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)
>>> np.dot(A,D)
>>> np.vdot(A,D)
>>> np.inner(A,D)
>>> np.outer(A,D)
>>> np.tensordot(A,D)
>>> np.kron(A,D)
```

Multiplication
Dot product
Vector dot product
Inner product
Outer product
Tensor dot product
Kronecker product

Exponential Functions

```
>>> linalg.expm(A)
>>> linalg.expm2(A)
>>> linalg.expm3(D)
```

Matrix exponential
Matrix exponential (Taylor Series)
Matrix exponential (eigenvalue decomposition)

Logarithm Function

```
>>> linalg.logm(A)
```

Matrix logarithm

Trigonometric Functions

```
>>> linalg.sinm(D)
>>> linalg.cosm(D)
>>> linalg.tanm(A)
```

Matrix sine
Matrix cosine
Matrix tangent

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
>>> linalg.coshm(D)
>>> linalg.tanhm(A)
```

Hyperbolic matrix sine
Hyperbolic matrix cosine
Hyperbolic matrix tangent

Matrix Sign Function

```
>>> np.signm(A)
```

Matrix sign function

Matrix Square Root

```
>>> linalg.sqrtm(A)
```

Matrix square root

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)
```

Evaluate matrix function

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)
>>> l1, l2 = la
>>> v[:,0]
>>> v[:,1]
>>> linalg.eigvals(A)
```

Solve ordinary or generalized eigenvalue problem for square matrix
Unpack eigenvalues
First eigenvector
Second eigenvector
Unpack eigenvalues

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)
```

Singular Value Decomposition (SVD)
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)
>>> sparse.linalg.svds(H, 2)
```

Eigenvalues and eigenvectors
SVD

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

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.

matplotlib

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]
>>> 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'))
```

2 Create Plot

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

Figure

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

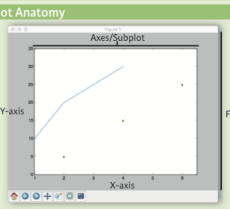
Axes

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)
```

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

4 Customize Plot

Colors, Color Bars & Color Maps

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

Markers

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

Linestyles

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

Text & Annotations

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

Mathtext

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

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)                 Add padding to a plot
>>> ax.axis('equal')                        Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])   Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)                     Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',         Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')                   No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),          Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,        Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()                      Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)    Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))  Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)
>>> ax.scatter(x,y)
>>> axes[0,0].bar([1,2,3],[3,4,5])
>>> axes[1,0].barh([0.5,1,2.5],[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 horiontal rectangles (constant height)
Draw a horizontal line across axes
Draw a vertical line across axes
Draw filled polygons
Fill between y-values and o

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 a 2D field of arrows

Data Distributions

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

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

2D Data or Images

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

Colormapped or RGB arrays

```
>>> axes2[0].pcolor(data2)
>>> axes2[0].pcolormesh(data)
>>> CS = plt.contour(Y,X,U)
>>> axes2[2].contourf(data1)
>>> axes2[2]= ax.clabel(CS)
```

Pseudocolor plot of 2D array
Pseudocolor plot of 2D array
Plot contours
Plot filled contours
Label a contour plot

5 Save Plot

Save figures

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

Save transparent figures

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

6 Show Plot

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

Close & Clear

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

Clear an axis
Clear the entire figure
Close a window

DataCamp
Learn Python for Data Science Interactively

Python For Data Science *Cheat Sheet*

Scikit-Learn

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

Scikit-learn

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

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= 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(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 DataFrame, are also acceptable.

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

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)
```

Fit the model to the data

Unsupervised Learning

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

Fit the model to the data
Fit to data, then transform it

Prediction

Supervised Estimators

```
>>> y_pred = svc.predict(np.random.random((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 = k_means.predict(X_test)
```

Predict labels in clustering algos

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

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

Estimator score method
Metric scoring functions

Classification Report

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

Precision, recall, f1-score and support

Confusion Matrix

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

Regression Metrics

Mean Absolute Error

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

Mean Squared Error

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

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "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_)
```

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

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)
```

DataCamp
Learn Python for Data Science Interactively

A mostly complete chart of

Neural Networks

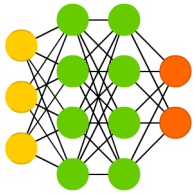
- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

Perceptron (P)

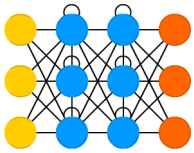
Feed Forward (FF)

Radial Basis Network (RBF)

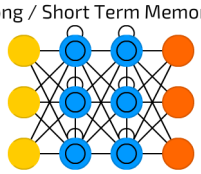
Deep Feed Forward (DFF)



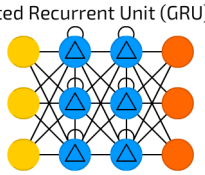
Recurrent Neural Network (RNN)



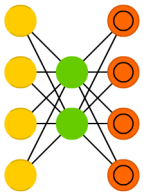
Long / Short Term Memory (LSTM)



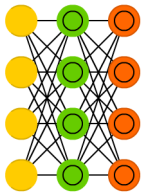
Gated Recurrent Unit (GRU)



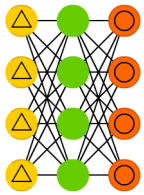
Auto Encoder (AE)



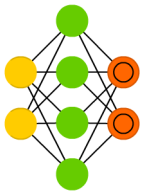
Variational AE (VAE)



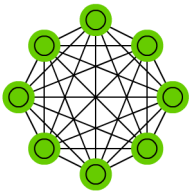
Denoising AE (DAE)



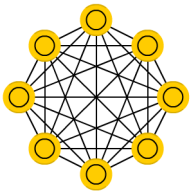
Sparse AE (SAE)



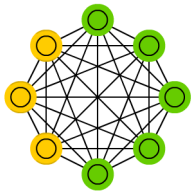
Markov Chain (MC)



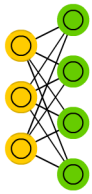
Hopfield Network (HN)



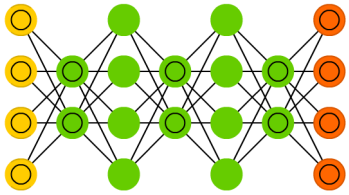
Boltzmann Machine (BM)



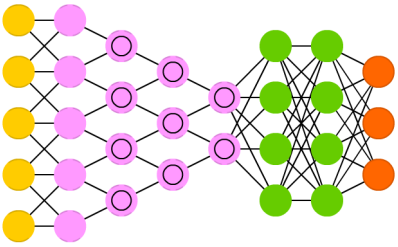
Restricted BM (RBM)



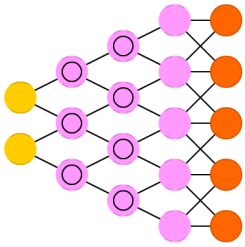
Deep Belief Network (DBN)



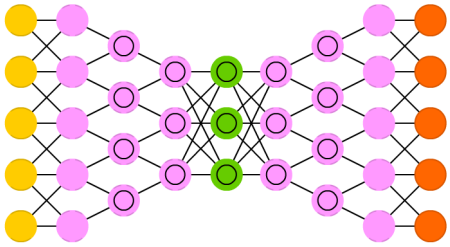
Deep Convolutional Network (DCN)



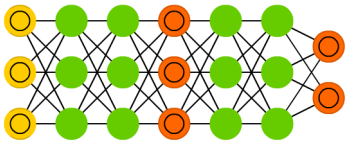
Deconvolutional Network (DN)



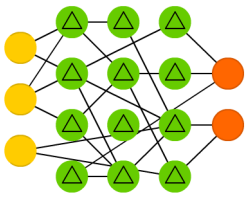
Deep Convolutional Inverse Graphics Network (DCIGN)



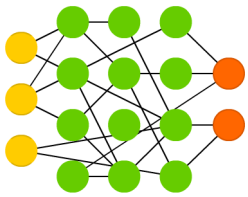
Generative Adversarial Network (GAN)



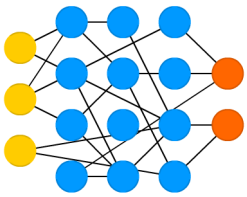
Liquid State Machine (LSM)



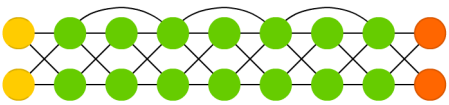
Extreme Learning Machine (ELM)



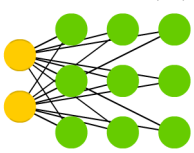
Echo State Network (ESN)



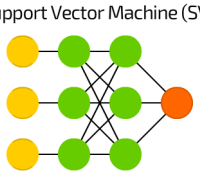
Deep Residual Network (DRN)



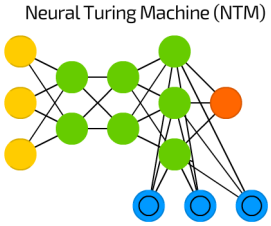
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)

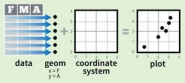


Data Visualization with ggplot2 Cheat Sheet

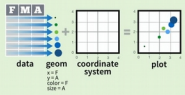
RStudio

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. Supplies many useful defaults.

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

data · add layers, elements with + · layer = geom + default stat + layer specific mappings · additional elements

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

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

last_plot()
Returns the last plot

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

Geoms - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

One Variable

Continuous

a <- ggplot(mpg, aes(hwy))

a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")

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

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

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

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

Discrete

b <- ggplot(mpg, aes(fl))

b + geom_bar()
x, alpha, color, fill, linetype, size, weight

Graphical Primitives

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

c + geom_polygon(aes(group = group))
x, y, alpha, color, fill, linetype, size

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

d + geom_path(lineend="butt", linejoin='round', linemitre=1)
x, y, alpha, color, linetype, size

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

e <- ggplot(seals, aes(x = long, y = lat))

e + geom_segment(aes(
xend = long + delta_long,
yend = lat + delta_lat))
x, xend, y, yend, alpha, color, linetype, size

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

Two Variables

Continuous X, Continuous Y

f <- ggplot(mpg, aes(cty, hwy))

f + geom_blank()

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

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

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

f + geom_rug(sides = "bl")
alpha, color, linetype, size

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

f + geom_text(aes(label = cty))
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

Continuous Bivariate Distribution

i <- ggplot(movies, aes(year, rating))

i + geom_bin2d(binwidth = c(5, 0.5))
xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight

i + geom_density2d()
x, y, alpha, colour, linetype, size

i + geom_hex()
x, y, alpha, colour, fill size

Continuous Function

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

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

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

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

Visualizing error

df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
k <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

k + geom_crossbar(fatten = 2)
x, y, ymax, ymin, alpha, color, fill, linetype, size

k + geom_errorbar()
x, ymax, ymin, alpha, color, linetype, size, width (also **geom_errorbarh()**)

k + geom_linerange()
x, ymin, ymax, alpha, color, linetype, size

k + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, linetype, shape, size

Discrete X, Continuous Y

g <- ggplot(mpg, aes(class, hwy))

g + geom_bar(stat = "identity")
x, y, alpha, color, fill, linetype, size, weight

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

g + geom_dotplot(binaxis = "y", stackdir = "center")
x, y, alpha, color, fill

g + geom_violin(scale = "area")
x, y, alpha, color, fill, linetype, size, weight

Maps

data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
map <- map_data("state")
l <- ggplot(data, aes(fill = murder))

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

Discrete X, Discrete Y

h <- ggplot(diamonds, aes(cut, color))

h + geom_jitter()
x, y, alpha, color, fill, shape, size

Three Variables

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

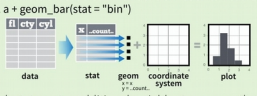
m + geom_contour(aes(z = z))
x, y, z, alpha, colour, linetype, size, weight

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

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

Stats - An alternative way to build a layer

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



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

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

stat function · layer specific mappings · variable created by transformation

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

geom for layer · parameters for stat

1D distributions

a + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..
a + stat_bindot(binwidth = 1, binaxis = "x")
x, y, | ..count.., ..ncount..
a + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

2D distributions

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

3 Variables

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

Comparisons

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

Functions

f + stat_ecdf(n = 40)
x, y | ..x.., ..y..
f + stat_quantile(quantiles = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")
x, y | ..quantile.., ..x.., ..y..
f + stat_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)
x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

General Purpose

ggplot() + stat_function(aes(x = -3:3), fun = dnorm, n = 101, args = list(sd=0.5))
x | ..y..
f + stat_identity()
ggplot() + stat_qq(aes(sample=1:100), distribution = qt, dparams = list(df=5))
sample, x, y | ..x.., ..y..
f + stat_sum()
x, y, size | ..size..
f + stat_summary(fun.data = "mean_cl_boot")
f + stat_unique()

Scales

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

n <- b + **geom_bar**(aes(fill = fl))

n

scale_ · aesthetic to adjust · prepackaged scale to use · scale specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping · title to use in legend/axis · labels to use in legend/axis · breaks to use in legend/axis

General Purpose scales

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

scale_*_continuous() - map cont' values to visual values
scale_*_discrete() - map discrete values to visual values
scale_*_identity() - use data values **as** visual values
scale_*_manual(values = c()) - map discrete values to manually chosen visual values

X and Y location scales

Use with x or y aesthetics (x shown here)

scale_x_date(labels = date_format("%m/%d"), breaks = date_breaks("2 weeks")) - treat x values as dates. See ?strptime for label formats.
scale_x_datetime() - treat x values as date times. Use same arguments as scale_x_date().
scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

Color and fill scales

Discrete	Continuous
n <- b + **geom_bar**(aes(fill = fl))	o <- a + **geom_dotplot**(aes(fill = ..x..))
n + **scale_fill_brewer**(palette = "Blues") For palette choices: library(RColorBrewer) display.brewer.all()	o + **scale_fill_gradient**(low = "red", high = "yellow")
	o + **scale_fill_gradient2**(low = "red", hight = "blue", mid = "white", midpoint = 25)
n + **scale_fill_grey**(start = 0.2, end = 0.8, na.value = "red")	o + **scale_fill_gradientn**(colours = terrain.colors(6)) Also: rainbow(), heat.colors(), topo.colors(), cm.colors(), RColorBrewer::brewer.pal()

Shape scales

p <- f + **geom_point**(aes(shape = fl))

p + **scale_shape**(
solid = FALSE)

p + **scale_shape_manual**(
values = c(3:7))
Shape values shown in chart on right

Manual shape values

Size scales

q <- f + **geom_point**(aes(size = cyl))

q + **scale_size_area**(max = 6)
Value mapped to area of circle (not radius)

Coordinate Systems

r <- b + geom_bar()

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

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

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

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

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set extras and strains to the name of a window function.

z + coord_map(projection = "ortho", orientation=c(41, -74, 0))
projection, orientation, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(. ~ fl)
facet into columns based on fl

t + facet_grid(year ~ .)
facet into rows based on year

t + facet_grid(year ~ fl)
facet into both rows and columns

t + facet_wrap(~ fl)
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

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

Set **labeller** to adjust facet labels

t + facet_grid(. ~ fl, labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(. ~ fl, labeller = label_bquote(alpha ^ .(x)**))**

α^{c}	α^{d}	α^{e}	α^{p}	α^{r}

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

c	d	e	p	r

Position Adjustments

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

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

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

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

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

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

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

Labels

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

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

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

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

Use scale functions to update legend labels

Legends

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

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

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

Themes

r + theme_bw()
White background with grid lines

r + theme_classic()
White background no gridlines

r + theme_grey()
Grey background (default theme)

r + theme_minimal()
Minimal theme

ggthemes - Package with additional ggplot2 themes

Zooming

Without clipping (preferred)

t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + xlim(0, 100) + **ylim**(10, 20)
t + scale_x_continuous(limits = c(0, 100)) + **scale_y_continuous**(limits = c(0, 100))