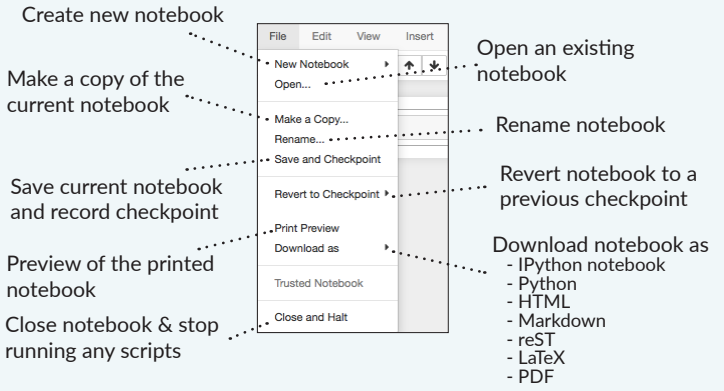
Python For Data Science *Cheat Sheet*

Jupyter Notebook

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

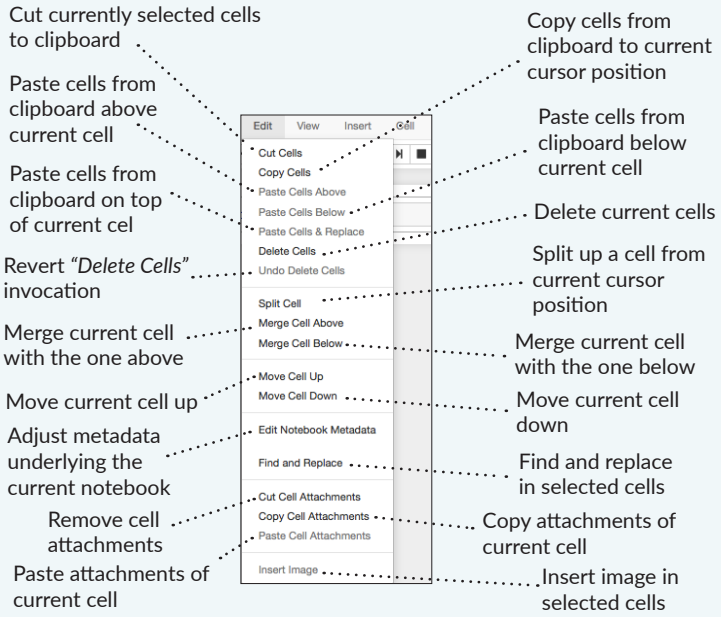
Saving/Loading Notebooks



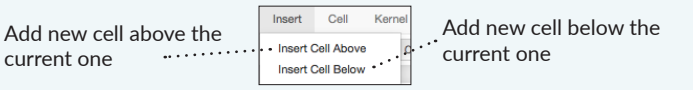
Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Edit Cells



Insert Cells

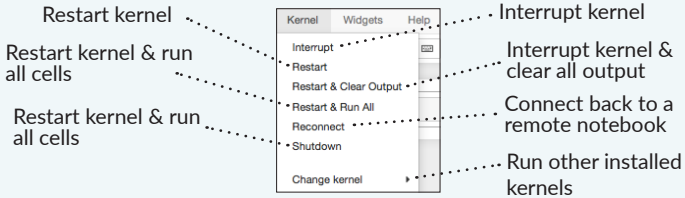


Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython IRkernel IJulia

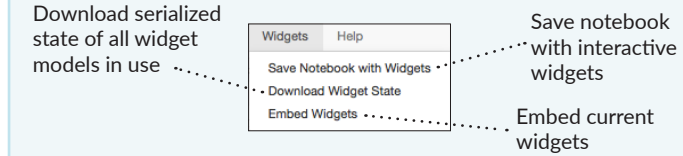
Installing Jupyter Notebook will automatically install the IPython kernel.



Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

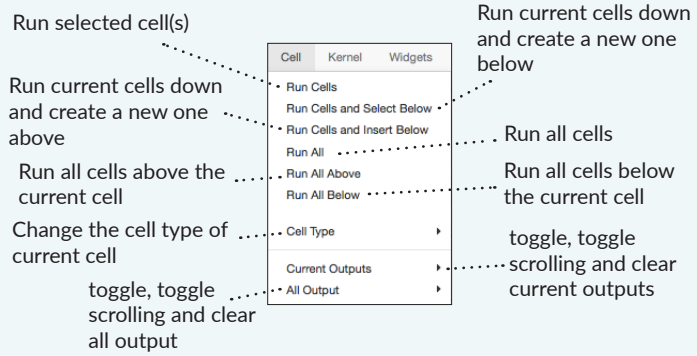


Command Mode:

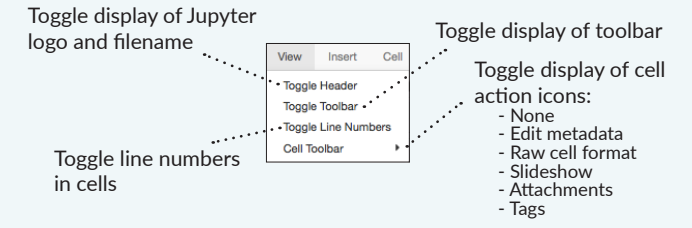
Edit Mode:

1. Save and checkpoint
2. Insert cell below
3. Cut cell
4. Copy cell(s)
5. Paste cell(s) below
6. Move cell up
7. Move cell down
8. Run current cell
9. Interrupt kernel
10. Restart kernel
11. Display characteristics
12. Open command palette
13. Current kernel
14. Kernel status
15. Log out from notebook server

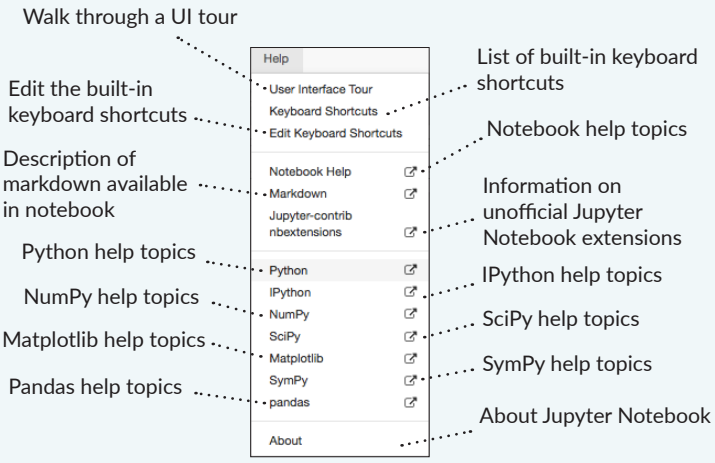
Executing Cells



View Cells



Asking For Help



Python For Data Science *Cheat Sheet*

Python Basics

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

Variables and Data Types

Variable Assignment

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

Calculations With Variables

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

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

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

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	
`>>> my_list2[1][0]`	my_list[list][itemOfList]
`>>> my_list2[1][:2]`	

List Operations

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

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

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

Selective import

```
>>> from math import pi
```

pandas — Data analysis

learn — Machine learning

NumPy — Scientific computing

matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

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

Numpy Arrays

Also see Lists

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

Selecting Numpy Array Elements

Index starts at 0

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

Numpy Array Operations

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

Numpy Array Functions

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

Strings

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

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

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

String Methods

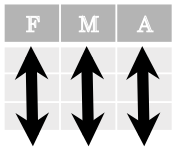
`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Data Wrangling with pandas Cheat Sheet

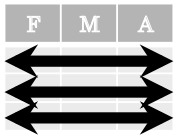
http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



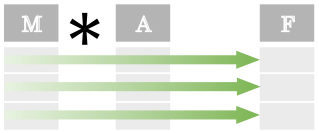
&



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.

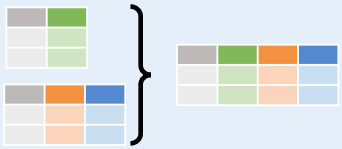
```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

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).

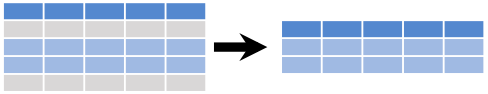
df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

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.

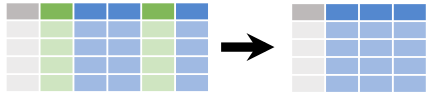
df.iloc[10:20]
Select rows by position.

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.

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.

Handling Missing Data

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

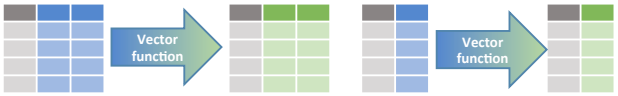
`df.fillna(value)`
Replace all NA/null data with value.

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.

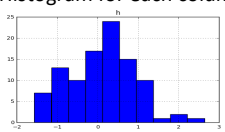
Windows

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

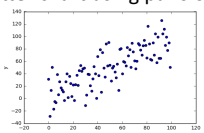
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

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.

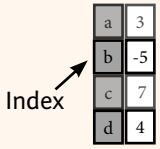
Use the following import convention:

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

Pandas Data Structures

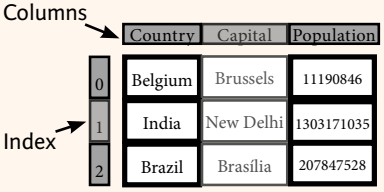
Series

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', 'Brasília'],
            '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   Brasília   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    Brasília
Population  207847528
```

Select single row of subset of rows

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

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

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

Retrieving Series/DataFrame Information

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.idxmin()/df.idxmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

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

Read and Write to Excel

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

Read multiple sheets from the same file

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

Read and Write to SQL Query or Database Table

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

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

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

Python For Data Science *Cheat Sheet*

Pandas

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

Reshaping Data

Pivot

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

Spread rows into columns

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

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

Pivot Table

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

Spread rows into columns

Stack / Unstack

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

Pivot a level of column labels
Pivot a level of index labels

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

Melt

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

Gather columns into rows

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

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

Iteration

```
>>> df.iteritems()
>>> df.iterrows()
```

(Column-index, Series) pairs
(Row-index, Series) pairs

Advanced Indexing

Also see NumPy Arrays

Selecting

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

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

Indexing With isin

```
>>> df[(df.Country.isin(df2.Type))]
>>> df3.filter(items="a","b"])
>>> df.select(lambda x: not x%5)
```

Find same elements
Filter on values
Select specific elements

Where

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

Subset the data

Query

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

Query DataFrame

Setting/Resetting Index

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

Set the index
Reset the index
Rename DataFrame

Reindexing

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

Forward Filling

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

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

Backward Filling

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

0	3
1	3
2	3
3	3
4	3

MultiIndexing

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

Duplicate Data

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

Return unique values
Check duplicates
Drop duplicates
Check index duplicates

Grouping Data

Aggregation

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

Transformation

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

Missing Data

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

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

Combining Data

data1

	X1	X2
a		11.432
b		1.303
c		99.906

data2

	X1	X3
a		20.784
b		NaN
d		20.784

Merge

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

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

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

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

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

	X1	X2	X3
a		11.432	20.784
b		1.303	NaN

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

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

Join

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

Concatenate

Vertical

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

Horizontal/Vertical

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

Dates

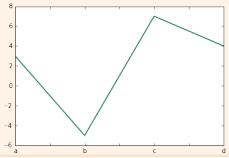
```
>>> df2['Date']= pd.to_datetime(df2['Date'])
>>> df2['Date']= pd.date_range('2000-1-1',
                               periods=6,
                               freq='M')
>>> dates = [datetime(2012,5,1), datetime(2012,5,2)]
>>> index = pd.DatetimeIndex(dates)
>>> index = pd.date_range(datetime(2012,2,1),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

Visualization

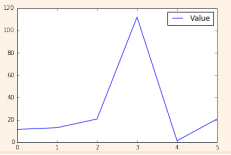
Also see Matplotlib

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

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



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



Python For Data Science *Cheat Sheet*

NumPy Basics

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

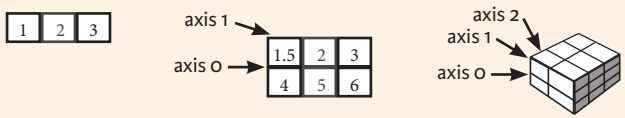
NumPy

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]                             Select the element at the 2nd index
  3
>>> b[1,2]                           Select the element at row 0 column 2
  6.0                                (equivalent to b[1][2])
```

Slicing

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

Boolean Indexing

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

Fancy Indexing

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

Array Manipulation

Transposing Array

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

Changing Array Shape

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

Adding/Removing Elements

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

Combining Arrays

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

Splitting Arrays

```
>>> np.hsplit(a,3)                   Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]  index
>>> np.vsplit(c,2)                   Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

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

Shape Manipulation

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

Polynomials

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

Vectorizing Functions

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

Type Handling

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

Other Useful Functions

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

Linear Algebra

Also see NumPy

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

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

Creating Matrices

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

Basic Matrix Routines

Inverse

```
>>> A.I                        Inverse
>>> linalg.inv(A)              Inverse
>>> A.T                        Tranpose matrix
>>> A.H                        Conjugate transposition
>>> np.trace(A)                Trace
```

Norm

```
>>> linalg.norm(A)             Frobenius norm
>>> linalg.norm(A,1)           L1 norm (max column sum)
>>> linalg.norm(A,np.inf)      L inf norm (max row sum)
```

Rank

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

Determinant

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

Solving linear problems

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

Generalized inverse

```
>>> linalg.pinv(C)             Compute the pseudo-inverse of a matrix
                               (least-squares solver)
>>> linalg.pinv2(C)            Compute the pseudo-inverse of a matrix
                               (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)         Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)   Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)   Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)   Dictionary Of Keys matrix
>>> E.todense()                Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A)   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

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

Exponential Functions

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

Logarithm Function

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

Trigonometric Tunctions

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

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)            Hypberbolic matrix sine
>>> linalg.coshm(D)            Hyperbolic matrix cosine
>>> linalg.tanhm(A)            Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.sigm(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)      Solve ordinary or generalized
                               eigenvalue problem for square matrix
>>> l1, l2 = la                Unpack eigenvalues
>>> v[:,0]                     First eigenvector
>>> v[:,1]                     Second eigenvector
>>> linalg.eigvals(A)          Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)     Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  Construct sigma matrix in SVD
```

LU Decomposition

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

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)  Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)         SVD
```

Asking For Help

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

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.model_selection 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.model_selection 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)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

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)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score — Estimator score method

```
>>> knn.score(X_test, y_test)
```

Metric scoring functions

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

Classification Report — Precision, recall, f1-score and support

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

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_)
```

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*

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))
```

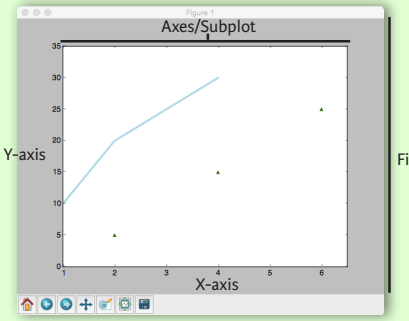
Axes

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)                     Draw points with lines or markers connecting them
>>> ax.scatter(x,y)                          Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])           Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])      Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)                  Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)                  Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')                Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')      Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)             Add an arrow to the axes
>>> axes[1,1].quiver(y,z)                    Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)            Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                              Plot a histogram
>>> ax3.boxplot(y)                           Make a box and whisker plot
>>> ax3.violinplot(z)                        Make a violin plot
```

2D Data or Images

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

```
>>> axes2[0].pcolor(data2)                   Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)                Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)                  Plot contours
>>> axes2[2].contourf(data1)                 Plot filled contours
>>> axes2[2]= ax.clabel(CS)                  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()                                Clear an axis
>>> plt.clf()                                Clear the entire figure
>>> plt.close()                              Close a window
```

Python For Data Science *Cheat Sheet*

Seaborn

Learn Data Science **Interactively** at www.DataCamp.com

Statistical Data Visualization With Seaborn

The Python visualization library **Seaborn** is based on matplotlib and provides a high-level interface for drawing attractive statistical graphics.

Make use of the following aliases to import the libraries:

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

The basic steps to creating plots with Seaborn are:

1. Prepare some data
2. Control figure aesthetics
3. Plot with Seaborn
4. Further customize your plot

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> tips = sns.load_dataset("tips")                    Step 1
>>> sns.set_style("whitegrid")                         Step 2
>>> g = sns.lmplot(x="tip",                            Step 3
                   y="total_bill",
                   data=tips,
                   aspect=2)
>>> g = (g.set_axis_labels("Tip","Total bill(USD)").
set(xlim=(0,10),ylim=(0,100)))                         Step 4
>>> plt.title("title")
>>> plt.show(g)                                        Step 5
```

1 Data

Also see Lists, NumPy & Pandas

```
>>> import pandas as pd
>>> import numpy as np
>>> uniform_data = np.random.rand(10, 12)
>>> data = pd.DataFrame({'x':np.arange(1,101),
                         'y':np.random.normal(0,4,100)})
```

Seaborn also offers built-in data sets:

```
>>> titanic = sns.load_dataset("titanic")
>>> iris = sns.load_dataset("iris")
```

2 Figure Aesthetics

Also see Matplotlib

```
>>> f, ax = plt.subplots(figsize=(5,6))
```
Create a figure and one subplot

Seaborn styles

```
>>> sns.set()
>>> sns.set_style("whitegrid")
>>> sns.set_style("ticks",
                  {"xtick.major.size":8,
                   "ytick.major.size":8})
>>> sns.axes_style("whitegrid")
```
(Re)set the seaborn default
Set the matplotlib parameters
Set the matplotlib parameters

Return a dict of params or use with `with` to temporarily set the style

Context Functions

```
>>> sns.set_context("talk")
>>> sns.set_context("notebook",
                    font_scale=1.5,
                    rc={"lines.linewidth":2.5})
```
Set context to `"talk"`
Set context to `"notebook"`, scale font elements and override param mapping

Color Palette

```
>>> sns.set_palette("husl",3)
>>> sns.color_palette("husl")
>>> flatui = ["#9b59b6","#3498db","#95a5a6","#e74c3c","#34495e","#2ecc71"]
>>> sns.set_palette(flatui)
```
Define the color palette
Use with `with` to temporarily set palette
Set your own color palette

3 Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,
                      col="survived",
                      row="sex")
>>> g = g.map(plt.hist,"age")
```
Subplot grid for plotting conditional relationships

```
>>> sns.factorplot(x="pclass",
                   y="survived",
                   hue="sex",
                   data=titanic)
```
Draw a categorical plot onto a Facetgrid

```
>>> sns.lmplot(x="sepal_width",
               y="sepal_length",
               hue="species",
               data=iris)
```
Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)
>>> h = h.map(plt.scatter)
>>> sns.pairplot(iris)
```
Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions

```
>>> i = sns.JointGrid(x="x",
                      y="y",
                      data=data)
>>> i = i.plot(sns.regplot,
               sns.distplot)
```
Grid for bivariate plot with marginal univariate plots

```
>>> sns.jointplot("sepal_length",
                  "sepal_width",
                  data=iris,
                  kind='kde')
```
Plot bivariate distribution

Categorical Plots

Scatterplot

```
>>> sns.stripplot(x="species",
                  y="petal_length",
                  data=iris)
```
Scatterplot with one categorical variable

```
>>> sns.swarmplot(x="species",
                  y="petal_length",
                  data=iris)
```
Categorical scatterplot with non-overlapping points

Bar Chart

```
>>> sns.barplot(x="sex",
                y="survived",
                hue="class",
                data=titanic)
```
Show point estimates and confidence intervals with scatterplot glyphs

Count Plot

```
>>> sns.countplot(x="deck",
                  data=titanic,
                  palette="Greens_d")
```
Show count of observations

Point Plot

```
>>> sns.pointplot(x="class",
                  y="survived",
                  hue="sex",
                  data=titanic,
                  palette={"male":"g",
                           "female":"m"},
                  markers=["^","o"],
                  linestyles=["-","--"])
```
Show point estimates and confidence intervals as rectangular bars

Boxplot

```
>>> sns.boxplot(x="alive",
                y="age",
                hue="adult_male",
                data=titanic)
```
Boxplot

```
>>> sns.boxplot(data=iris,orient="h")
```
Boxplot with wide-form data

Violinplot

```
>>> sns.violinplot(x="age",
                   y="sex",
                   hue="survived",
                   data=titanic)
```
Violin plot

Regression Plots

```
>>> sns.regplot(x="sepal_width",
                y="sepal_length",
                data=iris,
                ax=ax)
```
Plot data and a linear regression model fit

Distribution Plots

```
>>> plot = sns.distplot(data.y,
                        kde=False,
                        color="b")
```
Plot univariate distribution

Matrix Plots

```
>>> sns.heatmap(uniform_data,vmin=0,vmax=1)
```
Heatmap

4 Further Customizations

Also see Matplotlib

Axisgrid Objects

```
>>> g.despine(left=True)
>>> g.set_ylabels("Survived")
>>> g.set_xticklabels(rotation=45)
>>> g.set_axis_labels("Survived",
                      "Sex")
```
Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

```
>>> h.set(xlim=(0,5),
          ylim=(0,5),
          xticks=[0,2.5,5],
          yticks=[0,2.5,5])
```
Set the limit and ticks of the x-and y-axis

Plot

```
>>> plt.title("A Title")
>>> plt.ylabel("Survived")
>>> plt.xlabel("Sex")
>>> plt.ylim(0,100)
>>> plt.xlim(0,10)
>>> plt.setp(ax,yticks=[0,5])
>>> plt.tight_layout()
```
Add plot title
Adjust the label of the y-axis
Adjust the label of the x-axis
Adjust the limits of the y-axis
Adjust the limits of the x-axis
Adjust a plot property
Adjust subplot params

5 Show or Save Plot

Also see Matplotlib

```
>>> plt.show()
>>> plt.savefig("foo.png")
>>> plt.savefig("foo.png",
                transparent=True)
```
Show the plot
Save the plot as a figure
Save transparent figure

Close & Clear

Also see Matplotlib

```
>>> plt.cla()
>>> plt.clf()
>>> plt.close()
```
Clear an axis
Clear an entire figure
Close a window

Python for Data Science Cheat Sheet spaCy

Learn more Python for data science interactively at www.datacamp.com

DataCamp

About spaCy

spaCy is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It's designed specifically for production use and helps you build applications that process and "understand" large volumes of text. **Documentation:** spacy.io

```
$ pip install spacy
```

```
import spacy
```

Statistical models

Download statistical models

Predict part-of-speech tags, dependency labels, named entities and more. See here for available models: spacy.io/models

```
$ python -m spacy download en_core_web_sm
```

Check that your installed models are up to date

```
$ python -m spacy validate
```

Loading statistical models

```
import spacy
# Load the installed model "en_core_web_sm"
nlp = spacy.load("en_core_web_sm")
```

Documents and tokens

Processing text

Processing text with the `nlp` object returns a `Doc` object that holds all information about the tokens, their linguistic features and their relationships

```
doc = nlp("This is a text")
```

Accessing token attributes

```
doc = nlp("This is a text")
# Token texts
[token.text for token in doc]
# ['This', 'is', 'a', 'text']
```

Spans

Accessing spans

Span indices are **exclusive**. So `doc[2:4]` is a span starting at token 2, up to – but not including! – token 4.

```
doc = nlp("This is a text")
span = doc[2:4]
span.text
# 'a text'
```

Creating a span manually

```
# Import the Span object
from spacy.tokens import Span
# Create a Doc object
doc = nlp("I live in New York")
# Span for "New York" with label GPE (geopolitical)
span = Span(doc, 3, 5, label="GPE")
span.text
# 'New York'
```

Linguistic features

Attributes return label IDs. For string labels, use the attributes with an underscore. For example, `token.pos_` .

Part-of-speech tags

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("This is a text.")
# Coarse-grained part-of-speech tags
[token.pos_ for token in doc]
# ['DET', 'VERB', 'DET', 'NOUN', 'PUNCT']
# Fine-grained part-of-speech tags
[token.tag_ for token in doc]
# ['DT', 'VBZ', 'DT', 'NN', '.']
```

Syntactic dependencies

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("This is a text.")
# Dependency labels
[token.dep_ for token in doc]
# ['nsubj', 'ROOT', 'det', 'attr', 'punct']
# Syntactic head token (governor)
[token.head.text for token in doc]
# ['is', 'is', 'text', 'is', 'is']
```

Named entities

PREDICTED BY STATISTICAL MODEL

```
doc = nlp("Larry Page founded Google")
# Text and label of named entity span
[(ent.text, ent.label_) for ent in doc.ents]
# [('Larry Page', 'PERSON'), ('Google', 'ORG')]
```

Syntax iterators

Sentences

USUALLY NEEDS THE DEPENDENCY PARSER

```
doc = nlp("This a sentence. This is another one.")
# doc.sents is a generator that yields sentence spans
[sent.text for sent in doc.sents]
# ['This is a sentence.', 'This is another one.']
```

Base noun phrases

NEEDS THE TAGGER AND PARSER

```
doc = nlp("I have a red car")
# doc.noun_chunks is a generator that yields spans
[chunk.text for chunk in doc.noun_chunks]
# ['I', 'a red car']
```

Label explanations

```
spacy.explain("RB")
# 'adverb'
spacy.explain("GPE")
# 'Countries, cities, states'
```

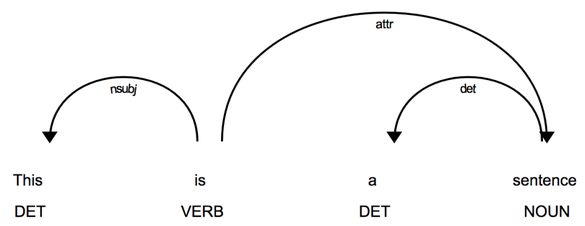
Visualizing

If you're in a Jupyter notebook, use `displacy.render` . Otherwise, use `displacy.serve` to start a web server and show the visualization in your browser.

```
from spacy import displacy
```

Visualize dependencies

```
doc = nlp("This is a sentence")
displacy.render(doc, style="dep")
```



Visualize named entities

```
doc = nlp("Larry Page founded Google")
displacy.render(doc, style="ent")
```

Larry Page PERSON founded Google ORG

Word vectors and similarity

To use word vectors, you need to install the larger models ending in `md` or `lg`, for example `en_core_web_lg`.

Comparing similarity

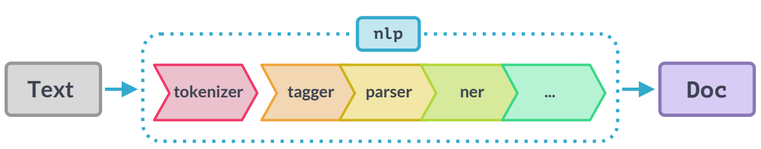
```
doc1 = nlp("I like cats")
doc2 = nlp("I like dogs")
# Compare 2 documents
doc1.similarity(doc2)
# Compare 2 tokens
doc1[2].similarity(doc2[2])
# Compare tokens and spans
doc1[0].similarity(doc2[1:3])
```

Accessing word vectors

```
# Vector as a numpy array
doc = nlp("I like cats")
# The L2 norm of the token's vector
doc[2].vector
doc[2].vector_norm
```

Pipeline components

Functions that take a `Doc` object, modify it and return it.



Pipeline information

```
nlp = spacy.load("en_core_web_sm")
nlp.pipe_names
# ['tagger', 'parser', 'ner']
nlp.pipeline
# [('tagger', <spacy.pipeline.Tagger>),
#  ('parser', <spacy.pipeline.DependencyParser>),
#  ('ner', <spacy.pipeline.EntityRecognizer>)]
```

Custom components

```
# Function that modifies the doc and returns it
def custom_component(doc):
    print("Do something to the doc here!")
    return doc

# Add the component first in the pipeline
nlp.add_pipe(custom_component, first=True)
```

Components can be added `first`, `last` (default), or `before` or `after` an existing component.

Extension attributes

Custom attributes that are registered on the global `Doc`, `Token` and `Span` classes and become available as `._`.

```
from spacy.tokens import Doc, Token, Span
doc = nlp("The sky over New York is blue")
```

Attribute extensions

WITH DEFAULT VALUE

```
# Register custom attribute on Token class
Token.set_extension("is_color", default=False)
# Overwrite extension attribute with default value
doc[6]._.is_color = True
```

Property extensions

WITH GETTER & SETTER

```
# Register custom attribute on Doc class
get_reversed = lambda doc: doc.text[::-1]
Doc.set_extension("reversed", getter=get_reversed)
# Compute value of extension attribute with getter
doc._.reversed
# 'eulb si kroY weN revo yks ehT'
```

Method extensions

CALLABLE METHOD

```
# Register custom attribute on Span class
has_label = lambda span, label: span.label_ == label
Span.set_extension("has_label", method=has_label)
# Compute value of extension attribute with method
doc[3:5].has_label("GPE")
# True
```

Rule-based matching

Using the matcher

```
# Matcher is initialized with the shared vocab
from spacy.matcher import Matcher
# Each dict represents one token and its attributes
matcher = Matcher(nlp.vocab)
# Add with ID, optional callback and pattern(s)
pattern = [{"LOWER": "new"}, {"LOWER": "york"}]
matcher.add("CITIES", None, pattern)
# Match by calling the matcher on a Doc object
doc = nlp("I live in New York")
matches = matcher(doc)
# Matches are (match_id, start, end) tuples
for match_id, start, end in matches:
    # Get the matched span by slicing the Doc
    span = doc[start:end]
    print(span.text)
# 'New York'
```

Rule-based matching

Token patterns

```
# "love cats", "loving cats", "loved cats"
pattern1 = [{"LEMMA": "love"}, {"LOWER": "cats"}]
# "10 people", "twenty people"
pattern2 = [{"LIKE_NUM": True}, {"TEXT": "people"}]
# "book", "a cat", "the sea" (noun + optional article)
pattern3 = [{"POS": "DET", "OP": "?"}, {"POS": "NOUN"}]
```

Operators and quantifiers

Can be added to a token dict as the `"OP"` key.

`!`	Negate pattern and match **exactly 0 times**.
`?`	Make pattern optional and match **0 or 1 times**.
`+`	Require pattern to match **1 or more times**.
`*`	Allow pattern to match **0 or more times**.

Glossary

Tokenization	Segmenting text into words, punctuation etc.
Lemmatization	Assigning the base forms of words, for example: "was" → "be" or "rats" → "rat".
Sentence Boundary Detection	Finding and segmenting individual sentences.
Part-of-speech (POS) Tagging	Assigning word types to tokens like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Named Entity Recognition (NER)	Labeling named "real-world" objects, like persons, companies or locations.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Statistical model	Process for making predictions based on examples.
Training	Updating a statistical model with new examples.