

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

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

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)
```

Strings

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

String Operations

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

Lists

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

Slice

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

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

Subset Lists of Lists

```
>>> my_list2[1][0]  
>>> my_list2[1][:2]
```

Select item at index 1
Select 3rd last item

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

Select items before index 3

Copy my_list

```
my_list[list][itemOfList]
```

List Operations

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

List Methods

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

Get the index of an item
Count an item
Append an item at a time
Remove an item
Remove an item
Reverse the list
Append an item
Remove an item
Insert an item
Sort the list

String Operations

Index starts at 0

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

String Methods

```
>>> my_string.upper()  
>>> my_string.lower()  
>>> my_string.count('w')  
>>> my_string.replace('e', 'i')  
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
Strip whitespaces

Also see NumPy Arrays

Libraries

Import libraries

```
>>> import numpy  
>>> import numpy as np  
Selective import  
>>> from math import pi
```

pandas 
Data analysis

Machine learning

NumPy 
Scientific computing

matplotlib 
2D plotting

Install Python



ANACONDA

Leading open data science platform
powered by Python



Free IDE that is included
with Anaconda



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

Get the dimensions of the array
Append items to an array
Insert items in an array
Delete items in an array
Mean of the array
Median of the array
Correlation coefficient
Standard deviation

DataCamp

Learn Python for Data Science [Interactively](#)



Python For Data Science Cheat Sheet

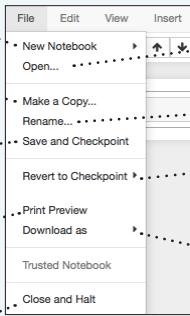
Jupyter Notebook

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



Saving/Loading Notebooks

Create new notebook



Make a copy of the current notebook

Open an existing notebook

Rename notebook

Save current notebook and record checkpoint

Revert notebook to a previous checkpoint

Preview of the printed notebook

Download notebook as - IPython notebook, Python, HTML, Markdown, reST, LaTeX, PDF

Close notebook & stop running any scripts

Trusted Notebook
Close and Halt

Writing Code And Text

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

Edit Cells

Cut currently selected cells to clipboard

Copy cells from clipboard to current cursor position

Paste cells from clipboard above current cell

Paste cells from clipboard below current cell

Paste cells from clipboard on top of current cell

Delete current cells

Revert "Delete Cells" invocation

Split up a cell from current cursor position

Merge current cell with the one above

Merge current cell with the one below

Merge current cell up

Move current cell down

Adjust metadata underlying the current notebook

Find and replace in selected cells

Remove cell attachments

Copy attachments of current cell

Paste attachments of current cell

Insert image in selected cells

Insert Cells

Add new cell above the current one



Add new cell below the current one

Working with Different Programming Languages

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



IPython



IRkernel



Julia

Installing Jupyter Notebook will automatically install the IPython kernel.

Restart kernel

Restart kernel & run all cells

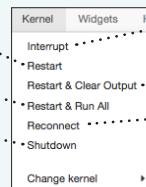
Restart kernel & run all cells

Interrupt kernel

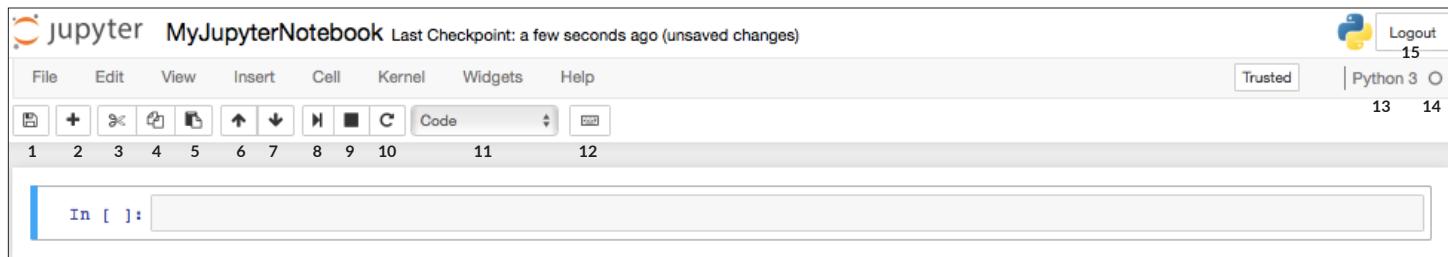
Interrupt kernel & clear all output

Connect back to a remote notebook

Run other installed kernels



Command Mode:



Edit Mode:



Executing Cells

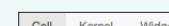
Run selected cell(s)

Run current cells down and create a new one above

Run all cells above the current cell

Change the cell type of current cell

toggle, toggle scrolling and clear all output



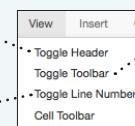
Run current cells down and create a new one below

Run all cells

Run all cells below the current cell
toggle, toggle scrolling and clear current outputs

View Cells

Toggle display of Jupyter logo and filename



Toggle line numbers in cells

Toggle display of toolbar

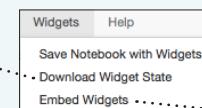
Toggle display of cell action icons:
- None
- Edit metadata
- Raw cell format
- Slideshow
- Attachments
- Tags

Widgets

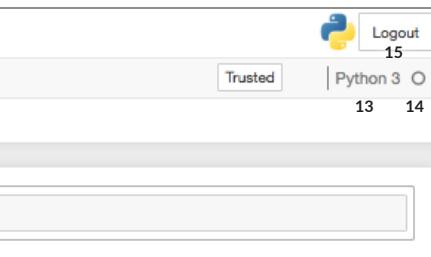
Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

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

Download serialized state of all widget models in use



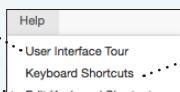
Save notebook with interactive widgets
Embed current widgets



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

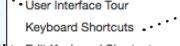
Asking For Help

Walk through a UI tour



List of built-in keyboard shortcuts

Edit the built-in keyboard shortcuts



Notebook help topics

Description of markdown available in notebook



Information on unofficial Jupyter Notebook extensions

Python help topics



IPython help topics

NumPy help topics



SciPy help topics

Matplotlib help topics



Sympy help topics

Pandas help topics



About Jupyter Notebook



DataCamp

Learn Python for Data Science Interactively



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

a	3
b	-5
c	7
d	4

Index

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

DataFrame

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

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

I/O

Read and Write to CSV

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

Asking For Help

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

Selection

Getting

>>> s['b'] -5	Get one element
>>> df[1:] Country Capital Population 1 India New Delhi 1303171035 2 Brazil Brasilia 207847528	Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

>>> df.iloc[[0], [0]] 'Belgium'	Select single value by row & column
>>> df.iat[[0], [0]] 'Belgium'	Select single value by row & column labels

By Label

>>> df.loc[[0], ['Country']] 'Belgium'	Select single row of subset of rows
>>> df.at[[0], ['Country']] 'Belgium'	Select a single column of subset of columns

By Label/Position

>>> df.ix[2] Country Brazil Capital Brasilia Population 207847528	Select rows and columns
>>> df.ix[:, 'Capital'] 0 Brussels 1 New Delhi 2 Brasilia	Boolean indexing

Boolean Indexing

>>> s[~(s > 1)] >>> s[(s < -1) (s > 2)]	Series s where value is not >1 s where value is <-1 or >2
>>> df[df['Population'] > 1200000000]	Use filter to adjust DataFrame
>>> s['a'] = 6	Setting

Set index a of Series s to 6

Dropping

```
>>> s.drop(['a', 'c'])
Drop values from rows (axis=0)
>>> df.drop('Country', axis=1)
Drop values from columns(axis=1)
```

Sort & Rank

```
>>> df.sort_index()
>>> 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)
```



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
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	0.233482	0.390959
2	0.184713	0.237102
3	0.433522	0.429401
Unstacked		

	5	0	2.233482
1	5	1	0.390959
2	4	0	0.184713
3	3	1	0.237102
4	2	0	0.433522
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

Selecting

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

Indexing With isin

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

Where

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

Query

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

Also see NumPy Arrays

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

Find same elements
Filter on values
Select specific elements

Subset the data
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 Brasilia 207847528  
3 Brazil Brasilia 207847528
```

Backward Filling

```
>>> s3 = s.reindex(range(5),  
               method='bfill')  
0 3  
1 3  
2 3  
3 3  
4 3
```

MultIndexing

```
>>> 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
c	99.906

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

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

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

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')
```

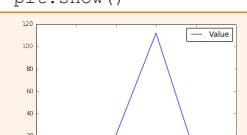
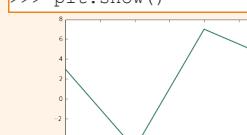
Visualization

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

```
>>> s.plot()
```

```
>>> df2.plot()
```

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



Also see Matplotlib

Learn Python for Data Science Interactively



Data Wrangling

with pandas

Cheat Sheet

<http://pandas.pydata.org>

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
e	2	5	8	11
		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'))
```

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:

F	M	A
4	7	10
5	8	11
6	9	12

Each variable is saved in its own column

Each observation is saved in its own row

F	M	A
4	7	10
5	8	11
6	9	12

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

M	*	A	F
4		7	10
5		8	11
6		9	12

Reshaping Data – Change the layout of a data set

pd.melt(df)

Gather columns into rows.

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

Spread rows into columns.

pd.concat([df1, df2], axis=1)

Append columns of DataFrames

df.sort_values('mpg')

Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)

Order rows by values of a column (high to low).

df.rename(columns = {'y': 'year'})

Rename the columns of a DataFrame

df.sort_index()

Sort the index of a DataFrame

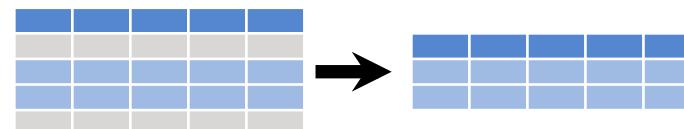
df.reset_index()

Reset index of DataFrame to row numbers, moving index to columns.

df.drop(['Length', 'Height'], axis=1)

Drop columns from DataFrame

Subset Observations (Rows)



df[df.Length > 7]

Extract rows that meet logical criteria.

df.drop_duplicates()

Remove duplicate rows (only considers columns).

df.head(n)

Select first n rows.

df.tail(n)

Select last n rows.

df.sample(frac=0.5)

Randomly select fraction of rows.

df.sample(n=10)

Randomly select n rows.

df.iloc[10:20]

Select rows by position.

df.nlargest(n, 'value')

Select and order top n entries.

df.nsmallest(n, 'value')

Select and order bottom n entries.

Subset Variables (Columns)



df[['width', 'length', 'species']]

Select multiple columns with specific names.

df['width'] or df.width

Select single column with specific name.

df.filter(regex='regex')

Select columns whose name matches regular expression regex.

regex (Regular Expressions) Examples

'.'	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).

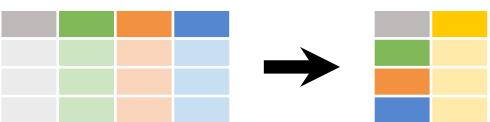
df.loc[df['a'] > 10, ['a', 'c']]

Select rows meeting logical condition, and only the specific columns .

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

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.

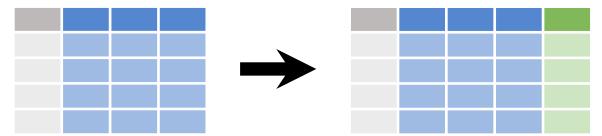
Windows

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

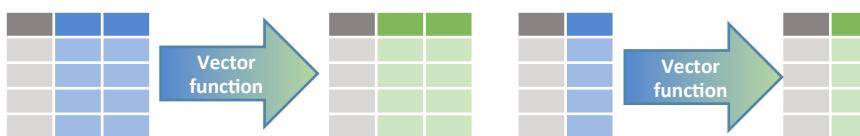
Handling Missing Data

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

Make New Columns



```
df.assign(Area=lambda df: df.Length*df.Height)
Compute and append one or more new columns.
df['Volume'] = df.Length*df.Height*df.Depth
Add single column.
pd.qcut(df.col, n, labels=False)
Bin column into n buckets.
```



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

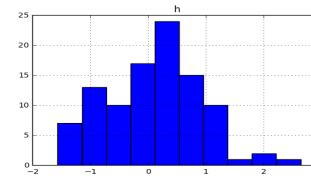
max(axis=1)	Element-wise max.
clip(lower=-10,upper=10)	Trim values at input thresholds
abs()	Absolute value.
min(axis=1)	Element-wise min.

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.

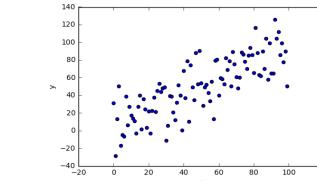
shift(1)	Copy with values shifted by 1.
rank(method='dense')	Ranks with no gaps.
rank(method='min')	Ranks. Ties get min rank.
rank(pct=True)	Ranks rescaled to interval [0, 1].
rank(method='first')	Ranks. Ties go to first value.
shift(-1)	Copy with values lagged by 1.
cumsum()	Cumulative sum.
cummax()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

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

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)
.y.query('_merge == "left_only"')
.y.drop(['_merge'], axis=1)
Rows that appear in ydf but not zdf (Setdiff).
```

Python For Data Science Cheat Sheet

Importing Data

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



Importing Data in Python

Most of the time, you'll use either NumPy or pandas to import your data:

```
>>> import numpy as np  
>>> import pandas as pd
```

Help

```
>>> np.info(np.ndarray.dtype)  
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'  
>>> file = open(filename, mode='r')  
>>> text = file.read()  
>>> print(file.closed)  
>>> file.close()  
>>> print(text)
```

Open the file for reading
Read a file's contents
Check whether file is closed
Close file

Using the context manager with

```
>>> with open('huck_finn.txt', 'r') as file:  
    print(file.readline())  
    print(file.readline())  
    print(file.readline())
```

Read a single line

Table Data: Flat Files

Importing Flat Files with numpy

Files with one data type

```
>>> filename = 'mnist.txt'  
>>> data = np.loadtxt(filename,  
                    delimiter=',',  
                    skiprows=2,  
                    usecols=[0,2],  
                    dtype=str)
```

String used to separate values
Skip the first 2 lines
Read the 1st and 3rd column
The type of the resulting array

Files with mixed data types

```
>>> filename = 'titanic.csv'  
>>> data = np.genfromtxt(filename,  
                    delimiter=',',  
                    names=True,  
                    dtype=None)
```

Look for column header

```
>>> data_array = np.recfromcsv(filename)
```

The default `dtype` of the `np.recfromcsv()` function is `None`.

Importing Flat Files with pandas

```
>>> filename = 'winequality-red.csv'  
>>> data = pd.read_csv(filename,  
                    nrows=5,  
                    header=None,  
                    sep='\t',  
                    comment='#',  
                    na_values=['?'])
```

Number of rows of file to read
Row number to use as col names
Delimiter to use
Character to split comments
String to recognize as NA/NaN

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'  
>>> data = pd.ExcelFile(file)  
>>> df_sheet2 = data.parse('1960-1966',  
                           skiprows=[0],  
                           names=['Country',  
                                  'AAM: War(2002)'])  
  
>>> df_sheet1 = data.parse(0,  
                           parse_cols=[0],  
                           skiprows=[0],  
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

SAS Files

```
>>> from sas7bdat import SAS7BDAT  
>>> with SAS7BDAT('urbanpop.sas/bdat') as file:  
    df_sas = file.to_data_frame()
```

Stata Files

```
>>> data = pd.read_stata('urbanpop.dta')
```

Relational Databases

```
>>> from sqlalchemy import create_engine  
>>> engine = create_engine('sqlite:///Northwind.sqlite')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()  
>>> rs = con.execute("SELECT * FROM Orders")  
>>> df = pd.DataFrame(rs.fetchall())  
>>> df.columns = rs.keys()  
>>> con.close()
```

Using the context manager with

```
>>> with engine.connect() as con:  
    rs = con.execute("SELECT OrderID FROM Orders")  
    df = pd.DataFrame(rs.fetchmany(size=5))  
    df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype  
>>> data_array.shape  
>>> len(data_array)
```

Data type of array elements
Array dimensions
Length of array

pandas DataFrames

```
>>> df.head()  
>>> df.tail()  
>>> df.index  
>>> df.columns  
>>> df.info()  
>>> data_array = data.values
```

Return first DataFrame rows
Return last DataFrame rows
Describe index
Describe DataFrame columns
Info on DataFrame
Convert a DataFrame to a NumPy array

Pickled Files

```
>>> import pickle  
>>> with open('pickled_fruit.pkl', 'rb') as file:  
    pickled_data = pickle.load(file)
```

HDF5 Files

```
>>> import h5py  
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'  
>>> data = h5py.File(filename, 'r')
```

Matlab Files

```
>>> import scipy.io  
>>> filename = 'workspace.mat'  
>>> mat = scipy.io.loadmat(filename)
```

Exploring Dictionaries

Accessing Elements with Functions

```
>>> print(mat.keys())  
>>> for key in mat.keys():  
    print(key)  
meta  
quality  
strain  
>>> pickled_data.values()  
>>> print(mat.items())
```

Print dictionary keys
Print dictionary keys

Return dictionary values
Returns items in list format of (key, value) tuple pairs

Accessing Data Items with Keys

```
>>> for key in data['meta'].keys():  
    print(key)  
Description  
DescriptionURL  
Detector  
Duration  
GRSstart  
Observatory  
Type  
UTCstart  
>>> print(data['meta']['Description'].value)
```

Explore the HDF5 structure

Retrieve the value for a key

Navigating Your FileSystem

Magic Commands

```
!ls  
%cd ..  
%pwd
```

List directory contents of files and directories
Change current working directory
Return the current working directory path

os Library

```
>>> import os  
>>> path = "/usr/tmp"  
>>> wd = os.getcwd()  
>>> os.listdir(wd)  
>>> os.chdir(path)  
>>> os.rename("test1.txt", "test2.txt")  
>>> os.remove("test1.txt")  
>>> os.mkdir("newdir")
```

Store the name of current directory in a string
Output contents of the directory in a list
Change current working directory
Rename a file

Delete an existing file
Create a new directory



Python For Data Science Cheat Sheet

NumPy Basics

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



NumPy

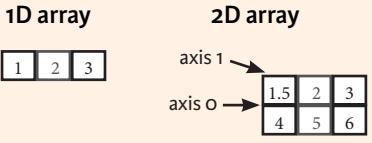
The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

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))
>>> np.ones((2,3,4),dtype=np.int16)
>>> d = np.arange(10,25,5)

>>> np.linspace(0,2,9)

>>> e = np.full((2,2),7)
>>> f = np.eye(2)
>>> np.random.random((2,2))
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2x2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savetxt('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

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

Inspecting Your Array

```
>>> a.shape
>>> len(a)
>>> b.ndim
>>> e.size
>>> b.dtype
>>> b.dtype.name
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

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

Array Mathematics

Arithmetic Operations

```
>>> g = a - b
      array([[-0.5,  0.,  0.],
             [-3., -3., -3.]])
>>> np.subtract(a,b)
>>> b + a
      array([[ 2.5,  4.,  6.],
             [ 5.,  7.,  9.]])
>>> np.add(b,a)
>>> a / b
      array([[ 0.66666667,  1.        ,  1.        ],
             [ 0.25,  0.4,  0.5       ]])
>>> np.divide(a,b)
>>> a * b
      array([[ 1.5,  4.,  9.],
             [ 4., 10., 18.]])
>>> np.multiply(a,b)
>>> np.exp(b)
>>> np.sqrt(b)
>>> np.sin(a)
>>> np.cos(b)
>>> np.log(a)
>>> e.dot(f)
      array([[ 7.,  7.],
             [ 7.,  7.]])
```

Subtraction
Addition
Addition
Division
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b
      array([[False,  True,  True],
             [False, False, False]], dtype=bool)
>>> a < 2
      array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()
>>> a.min()
>>> b.max(axis=0)
>>> b.cumsum(axis=1)
>>> a.mean()
>>> b.median()
>>> a.correlcoef()
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()
>>> np.copy(a)
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Subsetting

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

Select the element at the 2nd index

Slicing

```
>>> a[0:2]
      array([1, 2])
>>> b[0:2,1]
      array([ 2.,  5.])
```

Select items at index 0 and 1

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

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

```
>>> a[ : :-1]
      array([3, 2, 1])
```

Reversed array a

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

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

```
>>> 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 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.,  3.],
              [ 4.,  5.,  6.]]),
       array([[ 3.,  2.,  1.],
              [ 4.,  5.,  6.]])]
```

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



Python For Data Science Cheat Sheet

Also see NumPy

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.vsplit(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 poly1d  
>>> p = poly1d([3,4,5])
```

Create a polynomial object

Vectorizing Functions

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

Vectorize functions

Type Handling

>>> np.real(b) >>> 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 N-point central derivative Find the n-th derivative of a function at a point
--	---

Linear Algebra

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)
```

Transposition

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

Trace

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

Norm

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

Rank

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

Determinant

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

Solving linear problems

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

Generalized inverse

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

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

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)
```

Inverse

Inverse

Transpose matrix
Conjugate transposition

Trace

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

Matrix rank

Determinant

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

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

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)
```

Norm

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

Solving linear problems

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

Inverse

Norm

Solver for sparse matrices

Sparse Matrix Functions

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

Sparse matrix exponential

Matrix Functions

Addition

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

Subtraction

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

Division

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

Multiplication

```
>>> A @ D
```

```
>>> 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)
```

Exponential Functions

```
>>> linalg.expm(A)
```

```
>>> linalg.expm2(A)
```

```
>>> linalg.expm3(D)
```

Logarithm Function

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

Trigonometric Functions

```
>>> linalg.sinm(D)
```

```
>>> linalg.cosm(D)
```

```
>>> linalg.tanm(A)
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)
```

```
>>> linalg.coshm(D)
```

```
>>> linalg.tanhm(A)
```

Matrix Sign Function

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

Matrix Square Root

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

Arbitrary Functions

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

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(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)
```

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

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

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

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', 'F', 'F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

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

Preprocessing The Data

Standardization

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

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> 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)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

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

Support Vector Machines (SVM)

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

Naive Bayes

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

KNN

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

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning

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

Unsupervised Learning

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

Fit the model to the data

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)
```

Unsupervised Estimators

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

Predict labels
Predict labels
Estimate probability of a label
Predict labels in clustering algos

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

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

Estimator score method

Metric scoring functions

Classification Report

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

Precision, recall, f1-score and support

Confusion Matrix

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

Regression Metrics

Mean Absolute Error

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

Mean Squared Error

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

R² Score

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

Clustering Metrics

Adjusted Rand Index

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

Homogeneity

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

V-measure

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

Cross-Validation

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

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=kn,
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]
```

Preprocessing

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> y_train = to_categorical(y_train, num_classes)
>>> y_test = to_categorical(y_test, num_classes)
>>> y_train3 = to_categorical(y_train3, num_classes)
>>> y_test3 = to_categorical(y_test3, num_classes)
```

Model Architecture

Sequential Model

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

Multilayer Perceptron (MLP)

Binary Classification

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

Multi-Class Classification

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

Regression

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

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

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

Also see NumPy & Scikit-Learn

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)
```

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])
```



Python For Data Science Cheat Sheet

Matplotlib

Learn Python Interactively at www.DataCamp.com



Matplotlib

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



1 Prepare The Data

Also see [Lists & NumPy](#)

1D Data

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

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))  
>>> data2 = 3 * np.random.random((10, 10))  
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]  
>>> 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)
```

3 Plotting Routines

1D Data

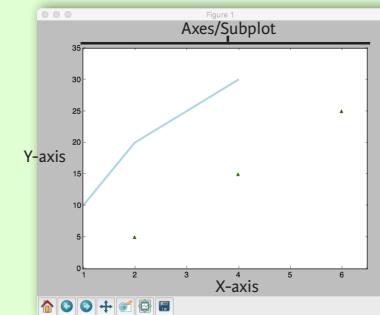
```
>>> lines = ax.plot(x,y)  
>>> ax.scatter(x,y)  
>>> axes[0,0].bar([1,2,3],[3,4,5])  
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])  
>>> axes[1,1].axhline(0.45)  
>>> axes[0,1].axvline(0.65)  
>>> ax.fill(x,y,color='blue')  
>>> ax.fill_between(x,y,color='yellow')
```

2D Data or Images

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

Plot Anatomy & Workflow

Plot Anatomy



Figure

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]  
>>> y = [10,20,25,30] Step 1  
>>> fig = plt.figure() Step 2  
>>> ax = fig.add_subplot(111) Step 3  
>>> ax.plot(x, y, color='lightblue', linewidth=3) Step 3.4  
>>> 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"),)
```

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)
```

Mathtext

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

Limits, Legends & Layouts

```
>>> ax.margins(x=0.0,y=0.1)  
>>> ax.axis('equal')  
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])  
>>> ax.set_xlim(0,10.5)
```

Legends

```
>>> ax.set(title='An Example Axes',  
           ylabel='Y-Axis',  
           xlabel='X-Axis')  
>>> ax.legend(loc='best')
```

Ticks

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

Subplot Spacing

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

Axis Spines

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

Add padding to a plot
Set the aspect ratio of the plot to 1
Set limits for x-and y-axis
Set limits for x-axis

Set a title and x-and y-axis labels

No overlapping plot elements

Manually set x-ticks

Make y-ticks longer and go in and out

Adjust the spacing between subplots

Fit subplot(s) in to the figure area

Make the top axis line for a plot invisible

Move the bottom axis line outward

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



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")  
>>> sns.set_style("whitegrid")  
>>> g = sns.lmplot(x="tip",  
y="total_bill",  
data=tips,  
aspect=2)  
>>> g.set_axis_labels("Tip", "Total bill(USD)")  
set(xlim=(0,10), ylim=(0,100))  
>>> plt.title("title")  
>>> plt.show(g)
```

Step 1
Step 2
Step 3
Step 4
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

```
>>> 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

3) Plotting With Seaborn

Axis Grids

```
>>> g = sns.FacetGrid(titanic,  
col="survived",  
row="sex")  
>>> g.map(plt.hist, "age")  
>>> sns.factorplot(x="pclass",  
y="survived",  
hue="sex",  
data=titanic)  
>>> sns.lmplot(x="sepal_width",  
y="sepal_length",  
hue="species",  
data=iris)
```

Subplot grid for plotting conditional relationships

Draw a categorical plot onto a Facetgrid

Plot data and regression model fits across a FacetGrid

```
>>> h = sns.PairGrid(iris)  
>>> h.map(plt.scatter)  
>>> sns.pairplot(iris)  
>>> i = sns.JointGrid(x="x",  
y="y",  
data=data)  
>>> i.plot(sns.regplot,  
sns.distplot)  
>>> sns.jointplot("sepal_length",  
"sepal_width",  
data=iris,  
kind='kde')
```

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Plot bivariate distribution

Categorical Plots

Scatterplot
`>>> sns.stripplot(x="species",
y="petal_length",
data=iris)
>>> sns.swarmplot(x="species",
y="petal_length",
data=iris)`

Bar Chart

```
>>> sns.barplot(x="sex",  
y="survived",  
hue="class",  
data=titanic)
```

Count Plot

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

Point Plot

```
>>> sns.pointplot(x="class",  
y="survived",  
hue="sex",  
data=titanic,  
palette={"male":"g",  
"female":"m"},  
markers=["^", "o"],  
linestyles=["-", "--"])
```

Boxplot

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

Violinplot

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

Scatterplot with one categorical variable

Categorical scatterplot with non-overlapping points

Show point estimates and confidence intervals with scatterplot glyphs

Show count of observations

Show point estimates and confidence intervals as rectangular bars

Boxplot

Boxplot with wide-form data

Violin plot

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

Subplot grid for plotting pairwise relationships
Plot pairwise bivariate distributions
Grid for bivariate plot with marginal univariate plots

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")  
>>> h.set(xlim=(0,5),  
ylim=(0,5),  
xticks=[0,2.5,5],  
yticks=[0,2.5,5])
```

Remove left spine
Set the labels of the y-axis
Set the tick labels for x
Set the axis labels

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

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

Clear an axis
Clear an entire figure
Close a window



Python For Data Science Cheat Sheet

Bokeh

Learn Bokeh [Interactively](#) at www.DataCamp.com, taught by Bryan Van de Ven, core contributor

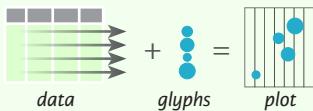


Plotting With Bokeh

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



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



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

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

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

1 Data

Also see Lists, NumPy & Pandas

Under the hood, your data is converted to Column Data Sources. You can also do this manually:

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame(np.array([[33.9, 4, 65, 'US'],
                               [32.4, 4, 66, 'Asia'],
                               [21.4, 4, 109, 'Europe']]),
                     columns=['mpg', 'cyl', 'hp', 'origin'],
                     index=['Toyota', 'Fiat', 'Volvo'])

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

2 Plotting

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

3 Renderers & Visual Customizations

Glyphs

Scatter Markers

```
>>> p1.circle(np.array([1,2,3]), np.array([3,2,1]),
             fill_color='white')
>>> p2.square(np.array([1.5,3.5,5.5]), [1,4,3],
             color='blue', size=1)
```

Line Glyphs

```
>>> p1.line([1,2,3,4], [3,4,5,6], line_width=2)
>>> p2.multi_line(pd.DataFrame([[1,2,3],[5,6,7]]),
                  pd.DataFrame([[3,4,5],[3,2,1]]),
                  color="blue")
```

Rows & Columns Layout

Rows

```
>>> from bokeh.layouts import row
```

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

Columns

```
>>> from bokeh.layouts import column
```

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

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

Grid Layout

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

Tabbed Layout

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

Legends

Legend Location

Inside Plot Area

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

Outside Plot Area

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

Customized Glyphs

Selection and Non-Selection Glyphs



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



Hover Glyphs

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



Colormapping

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

Also see Data

4 Output

Output to HTML File

```
>>> from bokeh.io import output_file, show
>>> output_file('my_bar_chart.html', mode='cdn')
```

Notebook Output

```
>>> from bokeh.io import output_notebook, show
>>> output_notebook()
```

Embedding

Standalone HTML

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

Components

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

5 Show or Save Your Plots

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

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

Also see Data

Statistical Charts With Bokeh

Also see Data

Bokeh's high-level `bokeh.charts` interface is ideal for quickly creating statistical charts

Bar Chart

```
>>> from bokeh.charts import Bar
>>> p = Bar(df, stacked=True, palette=['red','blue'])
```

Box Plot

```
>>> from bokeh.charts import BoxPlot
>>> p = BoxPlot(df, values='vals', label='cyl',
                legend='bottom_right')
```

Histogram

```
>>> from bokeh.charts import Histogram
>>> p = Histogram(df, title='Histogram')
```

Scatter Plot

```
>>> from bokeh.charts import Scatter
>>> p = Scatter(df, x='mpg', y ='hp', marker='square',
                xlabel='Miles Per Gallon',
                ylabel='Horsepower')
```





GETTING STARTED

1. Install

In the terminal
sudo pip install plotly

2. Sign Up & Configure

<http://www.plot.ly/python/getting-started>

3. Boilerplate Imports

```
import plotly.plotly as py
import plotly.graph_objs as go
```

4. A Hello World Figure

```
trace = {'x': [1, 2], 'y': [1, 2]}
data = [trace]
fig = go.Figure()
    data = data, layout = layout )
```

5. Plot the Figure!

In the terminal:
plot_url = py.plot(fig)

Or in the IPython notebook:
py.iplot(fig)

BASIC CHARTS

Line Plots

```
trace1 = go.Scatter(
    x = [1, 2], y = [1, 2])
trace2 = go.Scatter(
    x = [1, 2], y = [2, 1])
py.iplot([trace1, trace2])
```

Bubble Charts

```
trace = go.Scatter(
    x = [1, 2, 3], y = [1, 2, 3],
    marker = dict(
        color = ['red', 'blue',
        'green'],
        size = [30, 80, 200]),
        mode = 'markers')
py.iplot([trace])
```

Scatter Plots

```
trace1 = go.Scatter(
    x = [1, 2, 3], y = [1, 2, 3],
    text = ['A', 'B', 'C'],
    textposition = 'top center',
    mode = 'markers+text')
mode = [trace]
py.iplot(data)
```

Heatmaps

```
trace = go.Heatmap(
    z = [[1, 2, 3, 4],
        [5, 6, 7, 8]])
data = [trace]
py.iplot(data)
```

Bar Charts

```
trace = go.Bar(
    x = [1, 2], y = [1, 2])
data = [trace]
py.iplot(data)
```

Area Plots

```
trace = go.Scatter(
    x = [1, 2], y = [1, 2],
    fill = 'tonexty')
data = [trace]
py.iplot(data)
```

LAYOUT

Legends

```
trace1 = go.Scatter(
    name = 'Calvin',
    x = [1, 2], y = [1, 2])
trace2 = go.Scatter(
    name = 'Hobbes',
    x = [2, 1], y = [2, 1])
```

Axes

```
trace = go.Scatter(
    x = [1, 2, 3, 4],
    y = [1, 2, 3, 6])
```

```
axis_template = dict(
    showgrid = False,
    zeroline = False,
    nticks = 20,
    showline = True,
    title = 'X AXIS',
    mirror = 'all')
layout = go.Layout(
    showlegend = True,
    legend = dict(
        x = 0.2, y = 0.5))
    )
```

```
data = [trace1, trace2]
fig = go.Figure(
    data = data,
    layout = layout)
py.iplot(fig)
```

STATISTICAL CHARTS

Histograms

```
trace = go.Histogram(  
    x = [ 1, 2, 3, 3, 3, 4, 5 ])  
data = [ trace ]  
py.iplot ( data )
```

Box Plots

```
trace = go.Box (  
    x = [ 1, 2, 3, 3, 3, 4, 5 ])  
data = [ trace ]  
py.iplot ( data )
```

2D Histogram

```
trace = go.Histogram2d (  
    x = [ 1, 2, 3, 3, 3, 4, 5 ],  
    y = [ 1, 2, 3, 3, 3, 4, 5 ])  
data = [ trace ]  
py.iplot ( data )
```

MAPS

Bubble Map

```
trace = dict (  
    type = 'scattergeo',  
    lon = [ 100, 400 ], lat = [ 0, 0 ],  
    marker = dict (  
        marker = [ 'red', 'blue' ]  
        size = [ 30, 50 ] ),  
    mode = 'markers' )  
py.iplot ([ trace ])
```

Choropleth Map

```
trc = dict (  
    type = 'choropleth',  
    locations = [ 'AZ', 'CA', 'VT' ],  
    locationmode = 'USA-states',  
    colorscale = [ 'Viridis' ],  
    z = [ 10, 20, 40 ] )  
lyt = dict ( geo = dict ( scope = 'usa' ) )  
map = go.Figure ( data = [ trc ],  
    layout = lyt )  
py.iplot ( map )
```

Scatter Map

```
trace = dict (  
    type = 'scattergeo',  
    lon = [ 42, 39 ], lat = [ 12, 22 ],  
    marker = [ 'Rome', 'Greece' ],  
    mode = 'markers' )  
py.iplot ([ trace ])
```

3D CHARTS

3D Surface Plots

```
trace = go.Surface (  
    colorscale = 'Viridis',  
    z = [ [ 3, 5, 8, 13 ],  
          [ 21, 13, 8, 5 ] ] )  
data = [ trace ]  
py.iplot ( data )
```

3D Line Plots

```
trace = go.Scatter3D (  
    x = [ 9, 8, 5, 1 ], y = [ 1, 2, 4, 8 ],  
    z = [ 11, 8, 15, 3 ],  
    mode = 'lines' )  
data = [ trace ]  
py.iplot ( data )
```

3D Scatter Plots

```
trace = go.Scatter3D (  
    x = [ 9, 8, 5, 1 ], y = [ 1, 2, 4, 8 ],  
    z = [ 11, 8, 15, 3 ],  
    mode = 'markers' )  
data = [ trace ]  
py.iplot ( data )
```

FIGURE HIERARCHY

Figure {}

DATA []
TRACE {}
x, y, z []
color, text, size []
colorscale ABC or []
MARKER {}
color ABC
symbol ABC
LINE {}
color ABC
width 123

LAYOUT {}
title ABC
XAXIS, YAXIS {}
SCENE {}
XAXIS, YAXIS, ZAXIS {}
GEO {}
LEGEND {}
ANNOTATIONS {}

{ } = dictionary
[] = list
ABC = string
123 = number

Python For Data Science Cheat Sheet

PySpark Basics

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



Spark

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



Initializing Spark

SparkContext

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

Inspect SparkContext

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

Configuration

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

Using The Shell

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

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

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

Loading Data

Parallelized Collections

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

External Data

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

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

Retrieving RDD Information

Basic Information

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

List the number of partitions
Count RDD instances
Count RDD instances by key
Count RDD instances by value
Return (key,value) pairs as a dictionary
Sum of RDD elements
Check whether RDD is empty

Summary

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

Maximum value of RDD elements
Minimum value of RDD elements
Mean value of RDD elements
Standard deviation of RDD elements
Compute variance of RDD elements
Compute histogram by bins
Summary statistics (count, mean, stdev, max & min)

Applying Functions

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

Apply a function to each RDD element
Apply a function to each RDD element and flatten the result
Apply a flatMap function to each (key,value) pair of `rdd4` without changing the keys

Selecting Data

Getting

```
>>> rdd.collect()  
[('a', 7), ('a', 2), ('b', 2)]
```

Return a list with all RDD elements

```
>>> rdd.take(2)  
[('a', 7), ('a', 2)]
```

Take first 2 RDD elements

```
>>> rdd.first()  
('a', 7)
```

Take first RDD element

```
>>> rdd.top(2)  
[('b', 2), ('a', 7)]
```

Take top 2 RDD elements

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

Return sampled subset of `rdd3`

Sampling

```
>>> rdd.filter(lambda x: "a" in x)  
     .collect()
```

Filter the RDD

```
>>> rdd5.distinct().collect()
```

Return distinct RDD values

```
[('a',2,'b',7)]
```

```
>>> rdd.keys().collect()  
['a', 'a', 'b']
```

Return (key,value) RDD's keys

Iterating

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

Apply a function to all RDD elements

Reshaping Data

Reducing

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

Merge the rdd values for each key
Merge the rdd values

Grouping by

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

Return RDD of grouped values
Group rdd by key

Aggregating

```
>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))  
>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))  
>>> rdd3.aggregate((0,0),seqOp,combOp)  
(4950,100)  
>>> rdd.aggregateByKey((0,0),seqOp,combOp)  
     .collect()  
[(('a',(9,2)), ('b',(2,1)))]  
>>> rdd3.fold(0,add)  
4950  
>>> rdd.foldByKey(0, add)  
     .collect()  
[(('a',(9,2)))]  
>>> rdd3.keyBy(lambda x: x+x)  
     .collect()
```

Aggregate RDD elements of each partition and then the results
Aggregate values of each RDD key
Aggregate the elements of each partition, and then the results
Merge the values for each key
Create tuples of RDD elements by applying a function

Mathematical Operations

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

Return each rdd value not contained in rdd2
Return each (key,value) pair of rdd2 with no matching key in rdd
Return the Cartesian product of rdd and rdd2

Sort

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

Sort RDD by given function
Sort (key, value) RDD by key

Repartitioning

```
>>> rdd.repartition(4)  
>>> rdd.coalesce(1)
```

New RDD with 4 partitions
Decrease the number of partitions in the RDD to 1

Saving

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

Stopping SparkContext

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

Execution

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

