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

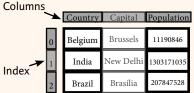
Series

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[1:1
   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

By Label

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

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

By Label/Position

>>>	df.ix[2]	
Co	untry Brazil	
Ca	pital Brasília	
Po	pulation 207847528	
>>>	<pre>df.ix[:,'Capital']</pre>	
0	Brussels	
1	New Delhi	
2	Brasília	

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

Boolean Indexing

>>>	s[~(s > 1)]
>>>	s[(s < -1) (s > 2)]
>>>	df[df['Population']>1200000

Setting

>>> s['a'] = 6

Select single value by row & column

Select single value by row & column labels

Select single row of subset of rows

Select a single column of subset of columns

Select rows and columns

Series s where value is not >1

s where value is <-1 or >2 000001 Use filter to adjust DataFrame

Set index a of Series s to 6

Read and Write to SQL Query or Database Table

>>>	<pre>pd.read_csv('file.csv', header=None, nrows=5)</pre>
>>>	<pre>df.to csv('myDataFrame.csv')</pre>

Read and Write to Excel

Read and Write to CSV

```
>>> 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')
```

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

read sql() is a convenience wrapper around read sql table() and read sql query()

```
>>> pd.to sql('myDf', engine)
```

Dropping

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

Sort & Rank

```
>>> df.sort index()
                                           Sort by labels along an axis
>>> df.sort values(by='Country')
                                           Sort by the values along an axis
>>> df.rank(\overline{1})
                                           Assign ranks to entries
```

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
                             (rows,columns)
>>> df.index
                             Describe index
>>> df.columns
                             Describe DataFrame columns
                             Info on DataFrame
>>> df.info()
>>> df.count()
                             Number of non-NA values
```

Summary

>>> df.cumsum() >>> df.min()/df.max() >>> df.idxmin()/df.idxmax() >>> df.describe() >>> df.mean()	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
                            Apply function
>>> df.apply(f)
>>> df.applymap(f)
                            Apply function element-wise
```

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
       10.0
       NaN
       5.0
 С
       7.0
 d
```

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)
    10.0
      -5.0
     5.0
 С
 d
     7.0
>>> s.sub(s3, fill value=2)
>>> s.div(s3, fill value=4)
>>> s.mul(s3, fill value=3)
```

DataCamp Learn Python for Data Science Interactively