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. pandas 🗔

Use the following import convention:

>>> import pandas as pd

Pandas Data Structures

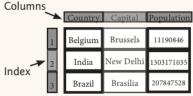
Series

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']
>>> df[1:]
             Capital Population
   Country
 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

Bv Label

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

Select single value by row & column labels

Bv Label/Position

. . . 16 ! [0]

>>>	ar.ıx[2]
Co	untry	Brazil
		Brasília
Po	pulatio:	n 207847528
>>>	df.ix[:,'Capital']
0	Bru	ssels
1	New	Delhi
2	Bra	sília

Select single row of subset of rows

Select a single column of subset of columns

Select rows and columns

Boolean Indexing

'New Delhi'

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

>>>	s[~(s > 1)]
>>>	s[(s < -1) (s > 2)]
>>>	df[df['Population']>12000000

Series s where value is not >1 s where value is <-1 or >2

Setting

>>> s['a'] = 6

001 Use filter to adjust DataFrame

Set index a of Series s to 6

1/0

Read and Write to CSV

```
>>> pd.read csv('file.csv', header=None, nrows=5)
>>> pd.to csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read excel('file.xlsx')
>>> pd.to excel('dir/myDataFrame.xlsx', sheet name='Sheet1')
 Read multiple sheets from the same file
```

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

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create engine
>>> engine = create engine('sqlite:///:memory:')
>>> pd.read sql("SELECT * FROM my table;", engine)
>>> pd.read sql_table('my_table', engine)
>>> pd.read sql query("SELECT * FROM my table;", engine)
read sql() is a convenience wrapper around read sql table() and
```

read sql query() >>> pd.to sql('myDf', engine)

Dropping

>>> df.drop('Country', axis=1) Drop values from columns(axis=	

Sort & Rank

```
>>> df.sort index(by='Country')
                                          Sort by row or column index
>>> s.order()
                                          Sort a series by its values
>>> df.rank()
                                          Assign ranks to entries
```

Retrieving Series/DataFrame Information

Basic Information

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

Summary

```
>>> df.sum()
                              Sum of values
                              Cummulative sum of values
>>> df.cumsum()
>>> df.min()/df.max()
                              Minimum/maximum values
                              Minimum/Maximum index value
>>> df.idmin()/df.idmax()
>>> df.describe()
                              Summary statistics
                              Mean of values
>>> df.mean()
                              Median of values
>>> df.median()
```

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
                            Apply function
>>> 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
 b
       NaN
       5.0
```

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)
    10.0
 b
     -5.0
     5.0
 C
     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 Int

