# **Python For Data Science** Cheat Sheet

**Pandas Basics** 

### **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**

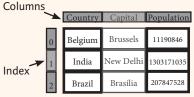
#### Series

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

Get one element

Get subset of a DataFrame

```
>>> s['b']
>>> df[1:]
   Country
             Capital Population
 1 India New Delhi 1303171035
 2 Brazil
            Brasília 207847528
```

### Selecting, Boolean Indexing & Setting

#### **By Position**

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

#### **By Label**

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

#### By Label/Position

```
>>> df.ix[2]
 Country
              Brazil
 Capital
            Brasília
 Population 207847528
>>> df.ix[:,'Capital']
       Brussels
      New Delhi
       Brasília
>>> df.ix[1,'Capital']
 'New Delhi'
```

# **Boolean Indexing**

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

# 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

Use filter to adjust DataFrame

Set index a of Series s to 6

# Read and Write to SQL Query or Database Table

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

#### 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')
```

```
>>> from sqlalchemy import create engine
```

```
>>> engine = create engine('sglite:///: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

>>>	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()
                                        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()
                             Number of non-NA values
>>> df.count()
```

#### Summary

```
Sum of values
>>> df.sum()
>>> df.cumsum()
                                Cummulative sum of values
                                Minimum/maximum values
>>> df.min()/df.max()
                               Minimum/Maximum index value
>>> df.idxmin()/df.idxmax()
>>> 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
                            Apply function element-wise
>>> df.applymap(f)
```

# **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
```

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