



# Python For Data Science

## Pandas Basics Cheat Sheet

Learn Pandas Basics online at [www.DataCamp.com](http://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'])
```

|  |   |    |
|--|---|----|
|  | a | 3  |
|  | b | -5 |
|  | c | 7  |
|  | d | 4  |

Index →

#### Dataframe

A **two-dimensional** labeled data structure with columns of potentially different types

|           |   |         |           |            |
|-----------|---|---------|-----------|------------|
| Columns → |   | Country | Capital   | Population |
| Index →   | 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'])
```

### > Dropping

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

### > Asking For Help

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

### > 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
```

### > 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')
>>> df.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()
>>> df.to_sql('myDf', engine)
```

### > Selection

Also see NumPy Arrays

#### Getting

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

#### Selecting, Boolean Indexing & Setting

By Position

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

By Label

```
>>> df.loc[0, ['Country']] #Select single value by row & column labels
'Belgium'
>>> df.at[0, ['Country']]
'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
Country Brazil
Capital Brasilia
Population 207847528
>>> df.ix[:, 'Capital'] #Select a single column of subset of columns
0 Brussels
1 New Delhi
2 Brasilia
>>> df.ix[1, 'Capital'] #Select rows and columns
'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>12000000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

### > Retrieving Series/DataFrame Information

#### Basic Information

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

#### Summary

```
>>> df.sum() #Sum of values
>>> df.cumsum() #Cumulative sum of values
>>> df.min()/df.max() #Minimum/maximum values
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value
>>> df.describe() #Summary statistics
>>> df.mean() #Mean of values
>>> df.median() #Median of values
```

### > 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
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)
```

Learn Data Skills Online at  
[www.DataCamp.com](http://www.DataCamp.com)







# Python For Data Science

## Data Wrangling in Pandas Cheat Sheet

Learn Data Wrangling online at [www.DataCamp.com](https://www.DataCamp.com)

### > Reshaping Data

#### Pivot

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

|   | 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      |
|---|------------|--------|--------|--------|
| 0 | 2016-03-01 | 11.432 | NaN    | 20.784 |
| 1 | 2016-03-02 | 1.303  | 13.031 | NaN    |
| 2 | 2016-03-03 | 99.906 | NaN    | 20.784 |

#### Pivot Table

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

#### Stack / Unstack

```
>>> stacked = df5.stack() #Pivot a level of column labels
>>> stacked.unstack() #Pivot a level of index labels
```

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

Unstacked

|   | 5        | 4        | 3        |
|---|----------|----------|----------|
| 1 | 0.233482 | 0.390959 | 0.433522 |
| 2 | 0.184713 | 0.237102 | 0.429401 |

Stacked

#### Melt

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

|   | 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       | Type  | Value  |
|----|------------|-------|--------|
| 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  | c      |
| 5  | 2016-03-03 | Value | 11.432 |
| 6  | 2016-03-02 | Value | 13.031 |
| 7  | 2016-03-02 | Value | 99.906 |
| 8  | 2016-03-01 | Value | 20.784 |
| 9  | 2016-03-03 | Value | 1.303  |
| 10 | 2016-03-02 | Value | 1.303  |
| 11 | 2016-03-03 | Value | 20.784 |

### > Iteration

```
>>> df.iteritems() #(Column-index, Series) pairs
>>> df.iterrows() #(Row-index, Series) pairs
```

### > Missing Data

```
>>> df.dropna() #Drop NaN values
>>> df3.fillna(df3.mean()) #Fill NaN values with a predetermined value
>>> df2.replace("a", "f") #Replace values with others
```

### > Advanced Indexing

Also see NumPy Arrays

#### Selecting

```
>>> df3.loc[:,(df3>1).any()] #Select cols with any vals >1
>>> df3.loc[:,(df3>1).all()] #Select cols with vals > 1
>>> df3.loc[:,df3.isnull().any()] #Select cols with NaN
>>> df3.loc[:,df3.notnull().all()] #Select cols without NaN
```

#### Indexing With isin()

```
>>> df[(df.Country.isin(df2.Type))] #Find same elements
>>> df3.filter(items="a","b") #Filter on values
>>> df.select(lambda x: not x%5) #Select specific elements
```

#### Where

```
>>> s.where(s > 0) #Subset the data
```

#### Query

```
>>> df6.query('second > first') #Query DataFrame
```

#### Setting/Resetting Index

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

#### Reindexing

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

#### Forward Filling

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

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

#### Backward Filling

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

|   | 0 | 3 |
|---|---|---|
| 1 | 1 | 3 |
| 2 | 2 | 3 |
| 3 | 3 | 3 |
| 4 | 4 | 3 |

#### MultiIndexing

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

### > Duplicate Data

```
>>> s3.unique() #Return unique values
>>> df2.duplicated('Type') #Check duplicates
>>> df2.drop_duplicates('Type', keep='last') #Drop duplicates
>>> df.index.duplicated() #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)
```

### > Combining Data

| data1 |        | data2 |        |
|-------|--------|-------|--------|
| X1    | X2     | X1    | X3     |
| a     | 11.432 | a     | 20.784 |
| b     | 1.303  | b     | NaN    |
| c     | 99.906 | d     | 20.784 |

#### Merge

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

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

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

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

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

| X1 | X2     | X3     |
|----|--------|--------|
| a  | 11.432 | 20.784 |
| b  | 1.303  | NaN    |

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

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

#### Join

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

#### Concatenate

##### Vertical

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

##### Horizontal/Vertical

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

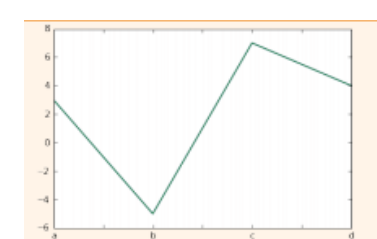
### > Dates

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

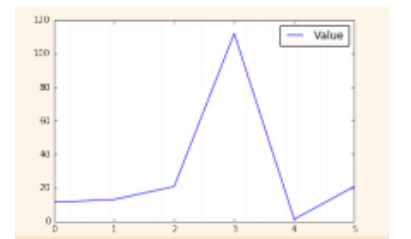
### > Visualization

Also see Matplotlib

```
>>> import matplotlib.pyplot as plt
>>> s.plot()
>>> plt.show()
```



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



Learn Data Skills Online at [www.DataCamp.com](https://www.DataCamp.com)