# PYTHON DATA SCIENCE

# PANDAS

# CHEAT SHEET

### What is Pandas?

It is a library that provides easy to use data structure and data analysis tool for Python Programming Language.

### Import Convention

import pandas as pd – Import pasdas

### Pandas Data Structure

- Series:

s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])

- Data Frame:

data_mobile = {'Mobile':['iPhone', 'Samsung', 'Redmi'], 'Color':['Red', 'White', 'Black'], 'Price':[High, Medium,Low]}

df = pd.DataFrame(data_mobile, columns=['Mobile', 'Color', 'Price'])

## DATA STRUCTURES

### SERIES (1D)

One-dimensional array-like object containing an array of data (of any **NumPy** data type) and an associated array of data labels, called its **"index"**. If index of data is not specified, then a default one consisting of the integers 0 through N-1 is created.

| | |
|---|---|
| Create Series | `series1 = pd.Series ([1, 2], index = ['a', 'b'])`<br>`series1 = pd.Series(dict1)*` |
| Get Series Values | `series1.values` |
| Get Values by Index | `series1['a']`<br>`series1[['b','a']]` |
| Get Series Index | `series1.index` |
| Get Name Attribute<br>(None is default) | `series1.name`<br>`series1.index.name` |
| ** Common Index Values are Added | `series1 + series2` |
| Unique But Unsorted | `series2 = series1.unique()` |

- * Can think of Series as a fixed-length, ordered dict. Series can be substitued into many functions that expect a dict.
- ** Auto-align differently-indexed data in arithmetic operations

### DATAFRAME (2D)

**Tabular** data structure with ordered collections of columns, each of which can be different value type. Data Frame (DF) can be thought of as a dict of Series.

| | |
|---|---|
| Create DF<br>(from a dict of equal-length lists or NumPy arrays) | `dict1 = {'state': ['Ohio', 'CA'], 'year': [2000, 2010]}`<br>`df1 = pd.DataFrame(dict1)`<br># columns are placed in sorted order<br>`df1 = pd.DataFrame(dict1, index = ['row1', 'row2']))`<br># specifying index<br>`df1 = pd.DataFrame(dict1, columns = ['year', 'state'])`<br># columns are placed in your given order |
| * Create DF<br>(from nested dict of dicts)<br>The inner keys as row indices | `dict1 = {'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 3, 'row2': 4}}`<br>`df1 = pd.DataFrame(dict1)` |

| | |
|---|---|
| Get Columns and Row Names | `df1.columns`<br>`df1.index` |
| Get Name Attribute<br>(None is default) | `df1.columns.name`<br>`df1.index.name` |
| Get Values | `df1.values`<br># returns the data as a 2D ndarray, the dtype will be chosen to accomandate all of the columns |
| ** Get Column as Series | `df1['state'] or df1.state` |
| ** Get Row as Series | `df1.ix['row2'] or df1.ix[1]` |
| Assign a column that doesn't exist will create a new column | `df1['eastern'] = df1.state == 'Ohio'` |
| Delete a column | `del df1['eastern']` |
| Switch Columns and Rows | `df1.T` |

- * Dicts of Series are treated the same as Nested dict of dicts.
- ** Data returned is a 'view' on the underlying data, NOT a copy. Thus, any in-place modificatons to the data will be reflected in df1.

### PANEL DATA (3D)

Create Panel Data : (Each item in the Panel is a DF)

```
import pandas_datareader.data as web
panel1 = pd.Panel({stk : web.get_data_
yahoo(stk, '1/1/2000', '1/1/2010')
for stk in ['AAPL', 'IBM']})
# panel1 Dimensions : 2 (item) * 861 (major) * 6 (minor)
```

"Stacked" DF form : (Useful way to represent panel data)

```
panel1 = panel1.swapaxes('item', 'minor')
panel1.ix[:, '6/1/2003', :].to_frame() *
```

=> Stacked DF (with **hierarchical indexing ****) :

```
#                Open High Low Close Volume Adj-Close
# major      minor
# 2003-06-01 AAPL
#            IBM
# 2003-06-02 AAPL
#            IBM
```

# Data Structures continued

- * DF has a "to_panel()" method which is the inverse of "to_frame()".
- ** Hierarchical indexing makes N-dimensional arrays unnecessary in a lot of cases. Aka prefer to use Stacked DF, not Panel data.

## INDEX OBJECTS

Immutable objects that hold the axis labels and other metadata (i.e. axis name)

- i.e. Index, MultiIndex, DatetimeIndex, PeriodIndex
- Any sequence of labels used when constructing Series or DF internally converted to an Index.
- Can functions as fixed-size set in additional to being array-like.

## HIERARCHICAL INDEXING

Multiple index levels on an axis : A way to work with higher dimensional data in a lower dimensional form.

MultiIndex :

```
series1 = Series(np.random.randn(6), index = [['a', 'a', 'a', 'b', 'b', 'b'], [1, 2, 3, 1, 2, 3]])

series1.index.names = ['key1', 'key2']
```

| | |
|---|---|
| Series Partial Indexing | `series1['b']` # Outer Level<br>`series1[:, 2]` # Inner Level |
| DF Partial Indexing | `df1['outerCol3','InnerCol2']`<br>Or<br>`df1['outerCol3']['InnerCol2']` |

### Swaping and Sorting Levels

| | |
|---|---|
| Swap Level (level interchanged) * | `swapSeries1 = series1.swaplevel('key1', 'key2')` |
| Sort Level | `series1.sortlevel(1)`<br># sorts according to first inner level |
| Common Ops : Swap and Sort ** | `series1.swaplevel(0, 1).sortlevel(0)`<br># the order of rows also change |

- * The order of the rows do not change. Only the two levels got swapped.
- ** Data selection performance is much better if the index is sorted starting with the outermost level, as a result of calling `sortlevel(0)` or `sort_index()`.

### Summary Statistics by Level

Most stats functions in DF or Series have a "level" option that you can specify the level you want on an axis.

| | |
|---|---|
| Sum rows (that have same 'key2' value) | `df1.sum(level = 'key2')` |
| Sum columns .. | `df1.sum(level = 'col3', axis = 1)` |

- Under the hood, the functionality provided here utilizes panda's "**groupby**".

### DataFrame's Columns as Indexes

DF's "set_index" will create a new DF using one or more of its columns as the index.

| | |
|---|---|
| New DF using columns as index | `df2 = df1.set_index(['col3', 'col4'])` * ‡<br># col3 becomes the outermost index, col4 becomes inner index. Values of col3, col4 become the index values. |

- * "reset_index" does the opposite of "set_index", the hierarchical index are moved into columns.
- ‡ By default, 'col3' and 'col4' will be removed from the DF, though you can leave them by option : `'drop = False'`.

## 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
```

**Forward Filling**

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

## Combining Data

data1

| X1 | X2 |
|---|---|
| a | 11.432 |
| b | 1.303 |
| c | 99.906 |

data2

| X1 | X3 |
|---|---|
| a | 20.784 |
| b | NaN |
| d | 20.784 |

### Pivot

```
>>> 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 | |

### 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

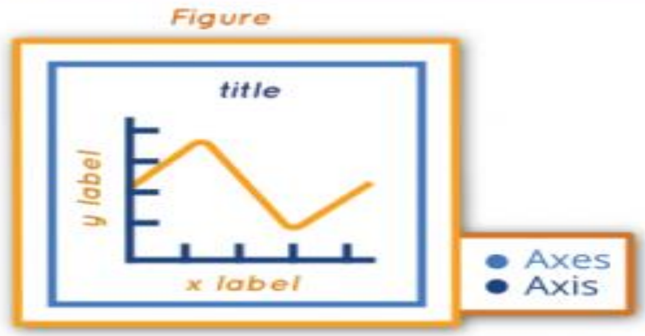
```
>>> import matplotlib.pyplot as plt
```

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

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

## Parts of a Figure

An Axes object is what we think of as a "plot". It has a title and two Axis objects that define data limits. Each Axis can have a label. There can be multiple Axes objects in a Figure.




## Plotting with Pandas Objects

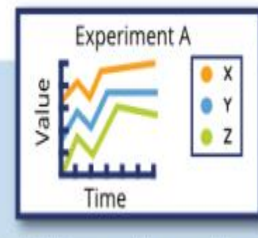
### Series


With a Series, Pandas plots values against the

### Dataframe


With a DataFrame, Pandas creates one line per

### Labels




Use Matplotlib to override or add annotations:

## Setup

Import packages:

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

Execute this at IPython prompt to display figures in new windows:

```
> %matplotlib
```

Use this in Jupyter notebooks to display static images inline:

```
> %matplotlib inline
```

Use this in Jupyter notebooks to display zoom-able images inline:

```
> %matplotlib notebook
```