These slides are based on:
Pandas by Maik Röder

https://www.slideshare.net/maikroeder/pandas-16424935
\[ y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it} \]
Pandas

• Powerful and productive Python data analysis and management library

• **Panel Data System**
  
  A.k.a. Excel for Python (with a bit of databases)

• Open Sourced by AQR Capital Management, LLC in late 2009

• 30,000 lines of tested Python/Cython code

• Used in production in many companies
Pandas

• Rich data structures and functions to make working with structured data fast, easy, and expressive

• Built on top of Numpy with its high performance array-computing features

• Flexible data manipulation capabilities of spreadsheets and relational databases

• Sophisticated indexing functionality

• slice, dice, perform aggregations, select subsets of data
The ideal tool for data scientists

• Cleaning data
• Analyzing data
• Modeling data
• Organizing the results of the analysis into a form suitable for plotting or tabular display
Series

• one-dimensional array-like object

```python
>>> s = Series((1,2,3,4,5))
```

• Contains an array of data (of any Numpy data type)

```python
>>> s.values
```

• Has an associated array of data labels, the `index` (Default index from 0 to N - 1)

```python
>>> s.index
```
Series

- Subclass of `numpy.ndarray`
- Data: any type
- Index labels need not be ordered
- Duplicates are possible (but result in reduced functionality)
Series data structure

```python
>>> import numpy
>>> randn = numpy.random.randn
>>> from pandas import *
>>> s = Series(randn(3),('a','b','c'))
>>> s
a   -0.889880
b    1.102135
c   -2.187296
>>> s.mean()
-0.65834710697853194
```
Series to/from dict

- Series to Python dict - No more explicit order

```python
>>> dict(s)
{'a': -0.88988001423312313,
 'c': -2.1872960440695666,
 'b':  1.1021347373670938}
```

- Back to a Series with a new Index from sorted dictionary keys

```python
>>> Series(dict(s))
a  -0.889880
b  1.102135
c  -2.187296
```
Reindexing labels

```python
>>> s
a   -0.496848
b   0.607173
c  -1.570596
>>> s.index
Index([a, b, c], dtype=object)
>>> s.reindex(['c','b','a'])
c   -1.570596
b   0.607173
a   -0.496848
```
Vectorization

```python
>>> s + s
a  -1.779760
b  2.204269
c  -4.374592
• Series work with Numpy

>>> numpy.exp(s)

a  0.410705
b  3.010586
c  0.112220
```
DataFrame

• Like data.frame in the statistical language/package R
• 2-dimensional tabular data structure
• Data manipulation with integrated indexing
• Support heterogeneous columns
• Homogeneous rows
DataFrame

- NumPy array-like
- Each column can have a different type
- Row and column index
- Size mutable: insert and delete columns
```python
>>> d = {'one': s*s, 'two': s+s}
>>> DataFrame(d)
   one     two
a  0.791886 -1.779760
b  1.214701  2.204269
c  4.784264 -4.374592
```
```python
>>> df.index
Index([a, b, c], dtype=object)
```
```python
>>> df.columns
Index([one, two], dtype=object)
```
**Dataframe add column**

- Add a third column
  ```python
  >>> df['three'] = s * 3
  ```
- It will share the existing index

```python
>>> df
```
```
<table>
<thead>
<tr>
<th></th>
<th>one</th>
<th>two</th>
<th>three</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.791886</td>
<td>-1.779760</td>
<td>-2.669640</td>
</tr>
<tr>
<td>b</td>
<td>1.214701</td>
<td>2.204269</td>
<td>3.306404</td>
</tr>
<tr>
<td>c</td>
<td>4.784264</td>
<td>-4.374592</td>
<td>-6.561888</td>
</tr>
</tbody>
</table>
```
Access to columns

- Access by attribute

```python
>>> df.one
one
a 0.791886
b 1.214701
c 4.784264
```

- Access by dict like notation

```python
>>> df['one']
one
a 0.791886
b 1.214701
c 4.784264
```
Reindexing

```py
>>> df.reindex(['c', 'b', 'a'])
>>> df

<table>
<thead>
<tr>
<th></th>
<th>one</th>
<th>two</th>
<th>three</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>4.784264</td>
<td>-4.374592</td>
<td>-6.561888</td>
</tr>
<tr>
<td>b</td>
<td>1.214701</td>
<td>2.204269</td>
<td>3.306404</td>
</tr>
<tr>
<td>a</td>
<td>0.791886</td>
<td>-1.779760</td>
<td>-2.669640</td>
</tr>
</tbody>
</table>
```
Drop entries from an axis

```python
>>> df.drop('c')

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>1.21</td>
<td>2.20</td>
<td>3.30</td>
</tr>
<tr>
<td>a</td>
<td>0.79</td>
<td>-1.78</td>
<td>-2.67</td>
</tr>
</tbody>
</table>
```

```python
dl = df.drop(['b', 'a'])
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>two</td>
<td>three</td>
</tr>
<tr>
<td>c</td>
<td>4.78</td>
<td>-4.37</td>
</tr>
</tbody>
</table>
Descriptive statistics

```python
>>> df.mean()
one   2.263617
two  -1.316694
three -1.975041
```

- Also: count, sum, median, min, max, abs, prod, std, var, skew, kurt, quantile, cumsum, cumprod, cummax, cummin
Computational Tools

- Covariance

```python
>>> s1 = Series(randn(1000))
>>> s2 = Series(randn(1000))
>>> s1.cov(s2)
0.01397370932323221539
```

- Also: pearson, kendall, spearman
I/O Operations

Read from
- csv, json, excel, html, SQL…
Data alignment

- Binary operations are joins!

\[
\begin{array}{c|c}
B & 1 \\
C & 2 \\
D & 3 \\
E & 4 \\
\end{array} + \begin{array}{c|c}
A & 0 \\
B & 1 \\
C & 2 \\
D & 3 \\
\end{array} = \begin{array}{c|c}
A & NA \\
B & 2 \\
C & 4 \\
D & 6 \\
E & NA \\
\end{array}
\]
Data manipulations

- Join / merge – database-like syntax
- Get_dummies (one hot vector from categorical data)
Aggregated Statistics

- Group by
- Describe
Plotting

```
DataFrame.plot.plotType()
- hist, pie, density, box,…
```
This and much more...

- Group by: split-apply-combine
- Merge, join and aggregate
- Reshaping and Pivot Tables
- Time Series / Date functionality
- Plotting with matplotlib
- IO Tools (Text, CSV, HDF5, ...)
- Sparse data structures
Resources

- http://pypi.python.org/pypi/pandas
- http://code.google.com/p/pandas