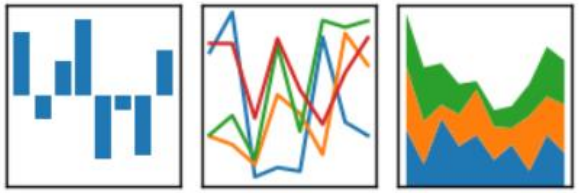
Data analysis with Pandas

Probability, Statistics, and Discrete Mathematics, Spring 2017

Helsinki Metropolia University of Applied Sciences

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



http://pandas.pydata.org/

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NUMFocus sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project.

A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

0.19.2 Final (December 24, 2016)

This is a minor bug-fix release in the 0.19.x series and includes some small regression fixes, bug fixes and performance improvements.

VERSIONS

Release
0.19.2 - December 2016
download // docs // pdf

Development
0.20.0 - 2017
github // docs

Previous Releases
0.19.1 - download // docs // pdf
0.19.0 - download // docs // pdf
0.18.1 - download // docs // pdf
0.18.0 - download // docs // pdf
0.17.1 - download // docs // pdf
0.17.0 - download // docs // pdf
0.16.2 - download // docs // pdf
0.16.1 - download // docs // pdf
0.16.0 - download // docs // pdf
0.15.2 - download // docs // pdf

Table Of Contents

pandas: powerful Python data analysis toolkit

PDF Version

Zipped HTML

Date: Dec 24, 2016 **Version**: 0.19.2

Binary Installers: http://pypi.python.org/pypi/pandas

Source Repository: http://github.com/pydata/pandas

Issues & Ideas: https://github.com/pydata/pandas/issues

Q&A Support: http://stackoverflow.com/questions/tagged/pandas

Developer Mailing List: http://groups.google.com/group/pydata

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. It is already well on its way toward this goal.

pandas is well suited for many different kinds of data:

- Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- Ordered and unordered (not necessarily fixed-frequency) time series data.
- Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels
- Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

From Notebook menu select: Help -> pandas

Table Of Contents

10 Minutes to pandas

This is a short introduction to pandas, geared mainly for new users. You can see more complex recipes in the Cookbook

Customarily, we import as follows:

```
In [1]: import pandas as pd

In [2]: import numpy as np

In [3]: import matplotlib.pyplot as plt
```

Object Creation

See the Data Structure Intro section

Creating a `Series` by passing a list of values, letting pandas create a default integer index:

```
In [4]: s = pd.Series([1,3,5,np.nan,6,8])

In [5]: s
Out[5]:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

http://pandas.pydata.org/pandas-docs/stable/10min.html

JUPYTER • PANDAS • PYTHON • DATASCIENCE

Pandas in Jupyter - Quickstart and Useful Snippets

BY: NIKOLAY GROZEV

DECEMBER 27, 2015

2 COMMENTS

Table of Contents

http://nikgrozev.com/2015/12/27/pandas-in-jupyter-quickstart-and-useful-snippets/

Reading data files

Loading CSV files

Loading a CSV file as a data frame is pretty easy:

```
1 data_frame = pandas.read_csv('file.csv', sep=';')
```

Sometimes the CSV file contains padding spaces in front of the values. To ignore them use the *skipinitialspaces* parameter:

```
1 pandas.read_csv('file.csv', sep=';', skipinitialspace=True)
```

Previewing Data

To preview the data and the metadata of a dataframe you can use the following functions:

```
# Displays the top 5 rows. Accepts an optional int parameter - num. of rows to show
df.head()

# Similar to head, but displays the last rows
df.tail()

# The dimensions of the dataframe as a (rows, cols) tuple
df.shape

# The number of columns. Equal to df.shape[0]
len(df)

# An array of the column names
df.columns

# Columns and their types
df.dtypes

# Converts the frame to a two-dimensional table
df.values

# Displays descriptive stats for all columns
df.describe()
```

Sorting

The *sort_index* method is used to sort the frame by one of its axis indices. The axis is either 0 or 1 - row/column axis respectively:

```
# Sort rows descendingly by the index
df.sort_index(axis=0, ascending=False)
```

We can also sort by one or multiple columns:

```
df.sort_values(by=['col2', 'col1'], ascending=False)
```

Selecting/Querying

Individual columns can be selected with the [] operator or directly as attributes:

```
# Selects only the column named 'col1';
df.col1

# Same as previous
df['col1']

# Select two columns
df[['col1', 'col2']]
```

You can also select by absolute coordinates/position in the frame. Indices are zero based:

```
# Selects second row
df.iloc[1]
# Selects rows 1-to-3
df.iloc[1:3]
# First row, first column
df.iloc[0,0]
# First 4 rows and first 2 columns
df.iloc[0:4, 0:2]
```

```
# Produces and array, not a single value!
df.col3 > 0
```

This allows us to write queries like these:

```
# Query by a single column value
df[df.col3 > 0]

# Query by a single column, if it is in a list of predefined values
df[df['col2'].isin(['Gold', 'Silver'])]

# A conjunction query using two columns
df[(df['col3'] > 0) & (df['col2'] == 'Silver')]

# A disjunction query using two columns
df[(df['col3'] > 0) | (df['col2'] == 'Silver')]

# A query checking the textual content of the cells
df[df.col2.str.contains('ilver')]
```

```
# Will allow us to embed images in the notebook
%matplotlib inline
```

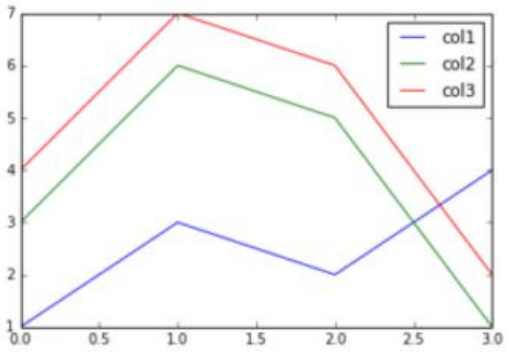
Basic Plotting

In the rest of this section we'll use the following data frame:

```
plot_df = pandas.DataFrame({
    'col1': [1, 3, 2, 4],
    'col2': [3, 6, 5, 1],
    'col3': [4, 7, 6, 2],
})
```

Data frames have a method called *plot*. By default, it plots a line chart with al numerical columns. The x-axis is the row index of the data frame. In other words, you're plotting :

```
plot_df.plot()
```



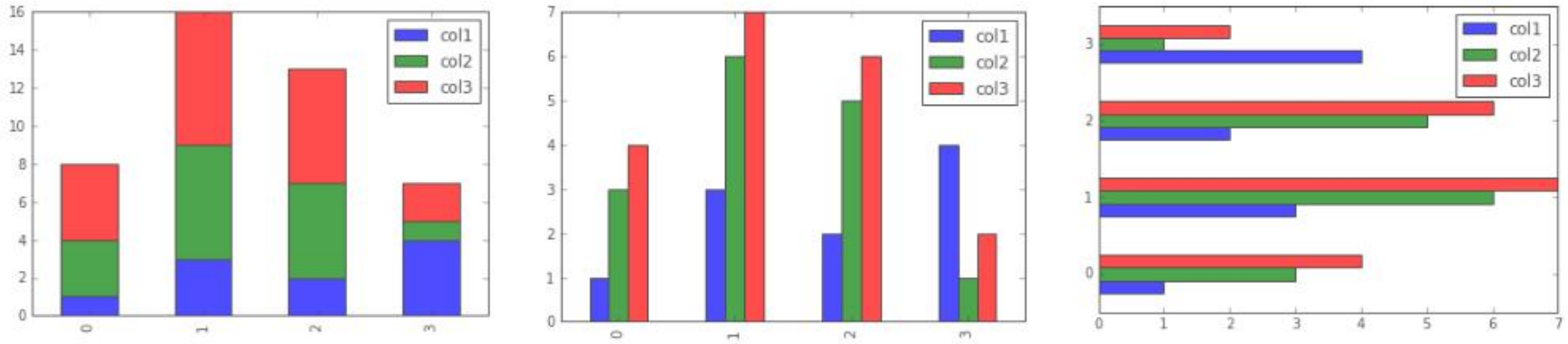
Line chart.

Using *kind='bar'* produces multiple plots - one for each row. In each plot, there's a bar for each cell.

```
# Use kind='hbar' for horizontal bars, and stacked=True to stack the groups
plot_df.plot(kind='bar')
```

Boxplots are displayed with the *kind='box'* options. Each box represents a numeric column.

```
plot_df.plot(kind='box')
```



Various plots.

Basic descriptive statistics

- count (N)
- mean
- standard deviation (std)
- 0 % (min)
- 25%
- 50% (Median)
- 75%
- 100% (max)

```
In [19]: df.describe()
Out[19]:
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.073711 -0.431125 -0.687758 -0.233103
std    0.843157  0.922818  0.779887  0.973118
min   -0.861849 -2.104569 -1.509059 -1.135632
25%   -0.611510 -0.600794 -1.368714 -1.076610
50%    0.022070 -0.228039 -0.767252 -0.386188
75%    0.658444  0.041933 -0.034326  0.461706
max    1.212112  0.567020  0.276232  1.071804
```

http://pandas.pydata.org/pandas-docs/stable/basics.html#descriptive-statistics

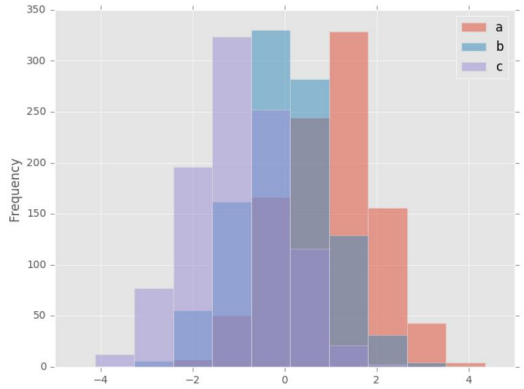
Histograms (Visualization)

Histogram can be drawn by using the `DataFrame.plot.hist()` and `Series.plot.hist()` methods.

```
In [21]: df4 = pd.DataFrame({'a': np.random.randn(1000) + 1, 'b': np.random.randn(1000),
   ....:                     'c': np.random.randn(1000) - 1}, columns=['a', 'b', 'c'])
   ....:

In [22]: plt.figure();

In [23]: df4.plot.hist(alpha=0.5)
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff26779c3d0>
```

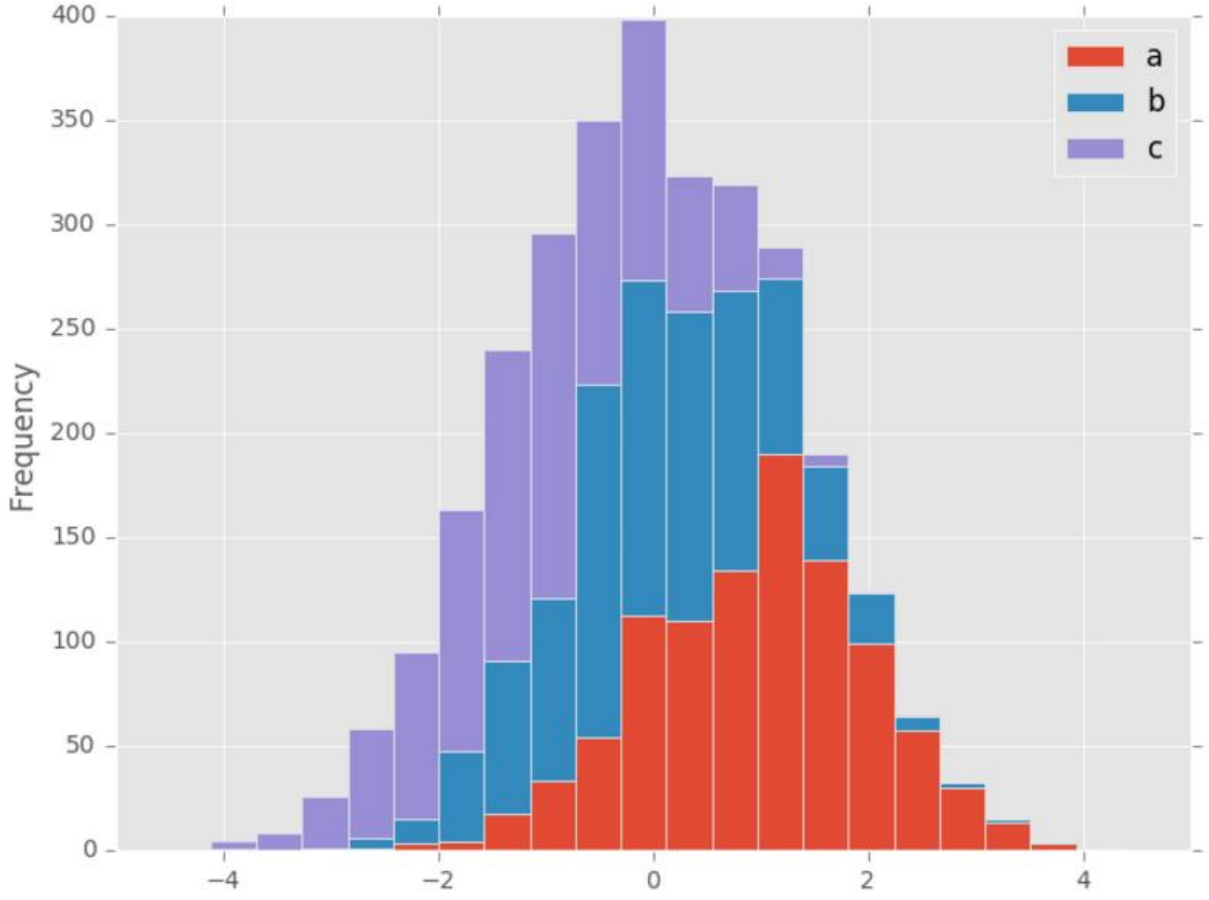


http://pandas.pydata.org/pandas-docs/stable/visualization.html

Histogram can be stacked by `stacked=True`. Bin size can be changed by `bins` keyword.

```
In [24]: plt.figure();

In [25]: df4.plot.hist(stacked=True, bins=20)
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff26caf76d0>
```

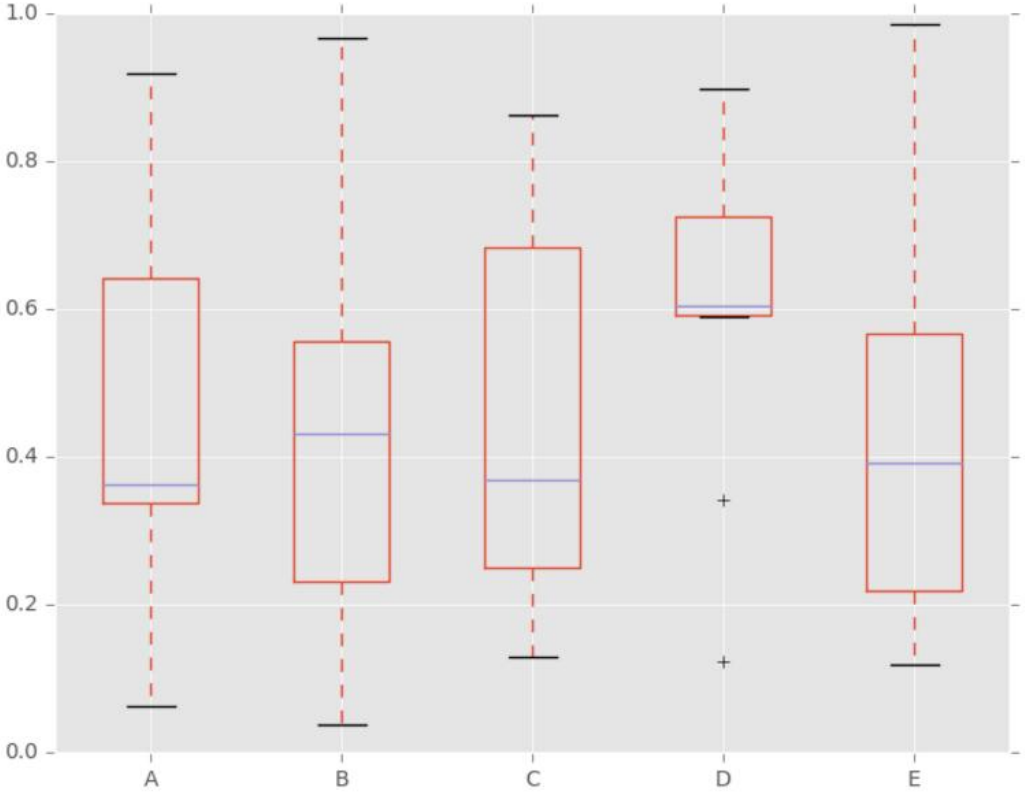


http://pandas.pydata.org/pandas-docs/stable/visualization.html

Boxplot

```
In [34]: df = pd.DataFrame(np.random.rand(10, 5), columns=['A', 'B', 'C', 'D', 'E'])

In [35]: df.plot.box()
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff27132a050>
```



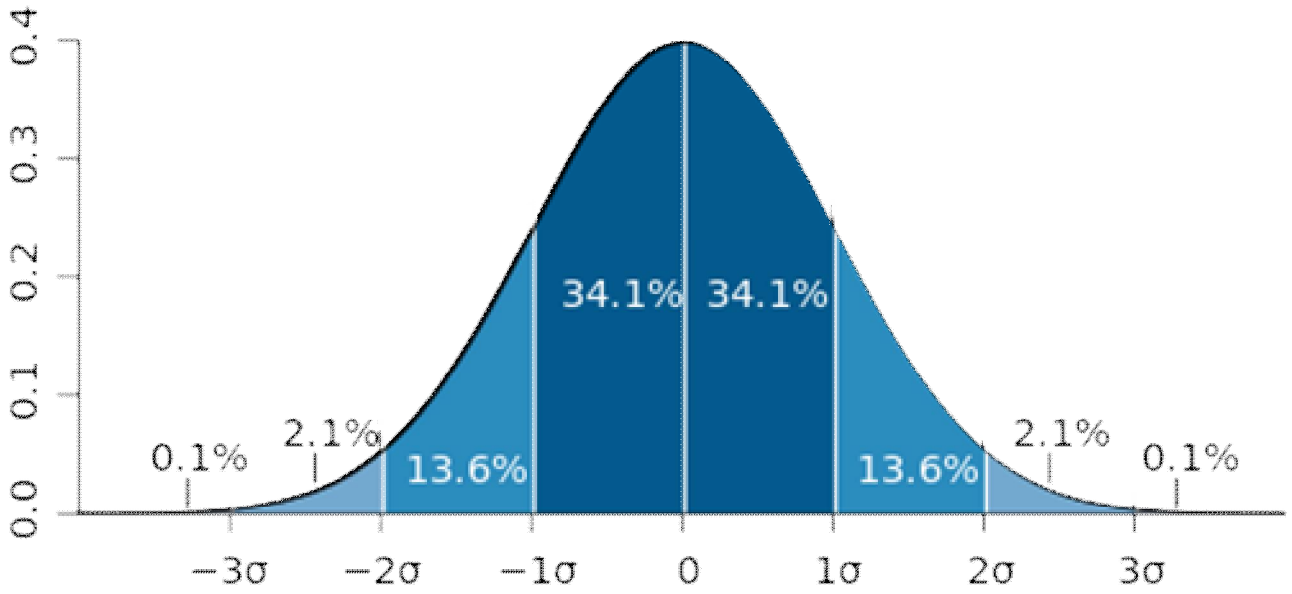
In descriptive statistics, a **box plot** or **boxplot** is a convenient way of graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (*whiskers*) indicating variability outside the upper and lower quartiles, hence the terms **box-and-whisker plot** and **box-and-whisker diagram**. Outliers may be plotted as individual points.

Source: Wikipedia, box plot

What is sample mean and standard deviation?

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

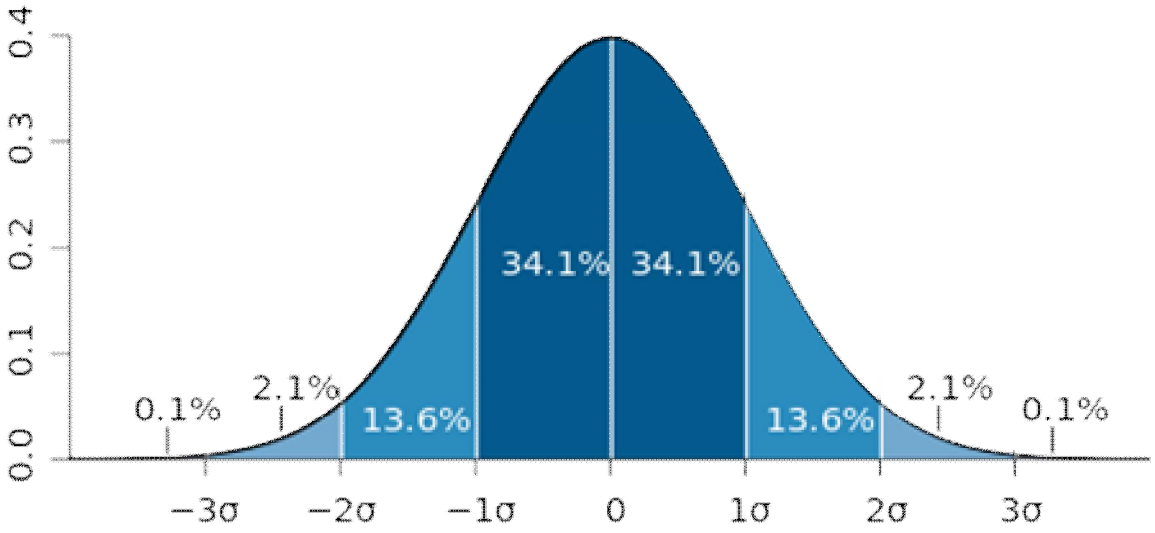


Standard error of the mean and confidence interval

$$\mathrm{SE}_{\bar{x}} = \frac{s}{\sqrt{n}}$$

Upper 95% limit $= \bar{x} + (\mathrm{SE} \times 1.96)$, and

Lower 95% limit $= \bar{x} - (\mathrm{SE} \times 1.96)$.



Mean +- 1*sigma è 68.2 %
Mean +- 2*sigma è 95.4 %
Mean +- 3*sigma è 99.8 %

Mean +- 1.96*SE è 95.0 %