Python programming — Pandas

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

October 5, 2013
Overview

Pandas?

Reading data

Summary statistics

Indexing

Merging, joining

Group-by and cross-tabulation

Statistical modeling
Pandas?

“Python Data Analysis Library”

Young library for data analysis

Developed from http://pandas.pydata.org/

Why Pandas?

A better Numpy: keep track of variable names, better indexing, easier linear modeling.

A better R: Access to more general programming language.

Why not pandas?

R: Still primary language for statisticians, means most advanced tools are there.

NaN/NA (Not a number/Not available)

Support to third-party algorithms compared to Numpy? Numexpr? (NumExpr in 0.11)
Pandas

Get some data from R

Get a standard dataset, _Pima_, from R:

```r
$ R
> library(MASS)
> write.csv(Pima.te, "pima.csv")
```

*pima.csv* now contains comma-separated values:

```
"","npreg","glu","bp","skin","bmi","ped","age","type"
"1",6,148,72,35,33.6,0.627,50,"Yes"
"2",1,85,66,29,26.6,0.351,31,"No"
"3",1,89,66,23,28.1,0.167,21,"No"
"4",3,78,50,32,31,0.248,26,"Yes"
"5",2,197,70,45,30.5,0.158,53,"Yes"
"6",5,166,72,19,25.8,0.587,51,"Yes"
```
Read data with Pandas

Back in Python:

```python
>>> import pandas as pd
>>> pima = pd.read_csv("pima.csv")
```

“pima” is now what Pandas call a *DataFrame* object. This object keeps track of both data (numerical as well as text), and column and row headers.

Let's use the first columns and the index column:

```python
>>> import pandas as pd
>>> pima = pd.read_csv("pima.csv", index_col=0)
```
Summary statistics

```python
>>> pima.describe()

<table>
<thead>
<tr>
<th></th>
<th>count</th>
<th>npreg</th>
<th>glu</th>
<th>bp</th>
<th>skin</th>
<th>bmi</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>332.000000</td>
<td>332.000000</td>
<td>332.000000</td>
<td>332.000000</td>
<td>332.000000</td>
<td>332.000000</td>
</tr>
<tr>
<td>mean</td>
<td>166.500000</td>
<td>3.484940</td>
<td>119.259036</td>
<td>71.653614</td>
<td>29.162651</td>
<td>33.239759</td>
</tr>
<tr>
<td>std</td>
<td>95.984374</td>
<td>3.283634</td>
<td>30.501138</td>
<td>12.799307</td>
<td>9.748068</td>
<td>7.282901</td>
</tr>
<tr>
<td>min</td>
<td>1.000000</td>
<td>0.000000</td>
<td>65.000000</td>
<td>24.000000</td>
<td>7.000000</td>
<td>19.400000</td>
</tr>
<tr>
<td>25%</td>
<td>83.750000</td>
<td>1.000000</td>
<td>96.000000</td>
<td>64.000000</td>
<td>22.000000</td>
<td>28.175000</td>
</tr>
<tr>
<td>50%</td>
<td>166.500000</td>
<td>2.000000</td>
<td>112.000000</td>
<td>72.000000</td>
<td>29.000000</td>
<td>32.900000</td>
</tr>
<tr>
<td>75%</td>
<td>249.250000</td>
<td>5.000000</td>
<td>136.250000</td>
<td>80.000000</td>
<td>36.000000</td>
<td>37.200000</td>
</tr>
<tr>
<td>max</td>
<td>332.000000</td>
<td>17.000000</td>
<td>197.000000</td>
<td>110.000000</td>
<td>63.000000</td>
<td>67.100000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ped</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>332.000000</td>
<td>332.000000</td>
</tr>
<tr>
<td>mean</td>
<td>0.528389</td>
<td>31.316265</td>
</tr>
<tr>
<td>std</td>
<td>0.363278</td>
<td>10.636225</td>
</tr>
<tr>
<td>min</td>
<td>0.085000</td>
<td>21.000000</td>
</tr>
<tr>
<td>25%</td>
<td>0.266000</td>
<td>23.000000</td>
</tr>
<tr>
<td>50%</td>
<td>0.440000</td>
<td>27.000000</td>
</tr>
<tr>
<td>75%</td>
<td>0.679250</td>
<td>37.000000</td>
</tr>
<tr>
<td>max</td>
<td>2.420000</td>
<td>81.000000</td>
</tr>
</tbody>
</table>
```
...Summary statistics

Other summary statistics (McKinney, 2012, around page 101):

pima.count() Count the number of rows

pima.mean(), pima.median(), pima.quantile()

pima.std(), pima.var()

pima.min(), pima.max()

Operation across columns instead, e.g., with the mean method:

pima.mean(axis=1)
Indexing the rows

For example, you can see the first two rows or the three last rows:

```python
>>> pima[0:2]
npreg  glu  bp  skin  bmi    ped  age  type
1     6    148  72   35  33.6   0.627  50  Yes
2     1     85  66   29  26.6   0.351  31  No
```

```python
>>> pima[-3:]
npreg  glu  bp  skin  bmi    ped  age  type
330    10   101  76   48  32.9   0.171  63  No
331     5   121  72   23  26.2   0.245  30  No
332     1    93  70   31  30.4   0.315  23  No
```

Notice that this is not an ordinary numerical matrix: We also got text (in the “type” column) within the “matrix”!
Indexing the columns

See a specific column, here 'bmi' (body-mass index):

```python
>>> pima['bmi']
1    33.6
2    26.6
3    28.1
4    31.0
[here I cut out several lines]
330   32.9
331   26.2
332   30.4
Name: bmi, Length: 332
```

The returned type is another of Pandas `Series` object, — another of the fundamental objects in the library:

```python
>>> type(pima['bmi'])
<class 'pandas.core.series.Series'>
```
Conditional indexing

Get the fat people (those with BMI above 30):

```python
>>> pima.shape
(332, 9)
>>> pima[pima["bmi"]>30].shape
(210, 9)
```

See histogram (with from pylab import *):

```python
>>> pima["bmi"].hist()
>>> show()
```

Or kernel density estimation plot ([McKinney, 2012, p 239](#))

```python
>>> pima["bmi"].plot(kind="kde")
>>> show()
```
Histogram and kernel density estimate (KDE) of the “bmi” variable (body mass index) of the Pima data set.
Row and column conditional indexing

Example by David Marx in R:

A <- runif(10)
B <- runif(10)
C <- runif(10)
D <- runif(10)
E <- runif(10)

df <- data.frame(A,B,C,D,E)
sliced_df <- df[, df[1,]<.5 ]

That is, select the columns in a dataframe where the values of the first row is below 0.5. Here with a 10-by-5 dataset with uniformly-distributed random numbers and columns indexed by letters.
...Row and column conditional indexing

Equivalent in Python

```python
import pandas as pd
from pylab import *
df = pd.DataFrame(rand(10,5), columns=["A", "B", "C", "D", "E"])
df.ix[:, df.ix[0, :]<0.5]
```

These variations do not work

```python
df[:, df[0]<0.5]
df[:, df[:1]<0.5]
df.ix[:, df[:1]<0.5]
```
Constructing a DataFrame from a dictionary where the keys become the column names

```python
>>> import pandas as pd
>>> import string

>>> spam_corpus = map(string.split, ["buy viagra", "buy antibody"])
>>> unique_words = set([word for doc in spam_corpus for word in doc])
>>> word_counts = [(word, map(lambda doc: doc.count(word), spam_corpus))
                     for word in unique_words]

>>> spam_bag_of_words = pd.DataFrame(dict(word_counts))
>>> print(spam_bag_of_words)

antibody    buy  viagra
0            0    1      1
1            1    1      0
```
**Concatenation**

Another corpus and then \texttt{concatenation} with the previous dataset

```python
>>> other_corpus = map(string.split, ["buy time", "hello"])
>>> unique_words = set([word for doc in other_corpus for word in doc])
>>> word_counts = [(word, map(lambda doc: doc.count(word), other_corpus))
                 for word in unique_words]
>>> other_bag_of_words = pd.DataFrame(dict(word_counts))
>>> print(other_bag_of_words)
   buy  hello  time
0   1      0    1
1   0      1    0

>>> pd.concat([spam_bag_of_words, other_bag_of_words], ignore_index=True)
   antibody  buy  hello  time  viagra
0         NaN   1     NaN   NaN   1
1         NaN   1     NaN   NaN   0
2         NaN   1     0      1   NaN
3         NaN   0     1     NaN   NaN
```

Finn Ärup Nielsen
Filling in missing data

(McKinney, 2012, page 145+)

```python
>>> pd.concat([spam_bag_of_words, other_bag_of_words], ignore_index=True)
  antibody  buy hello time  viagra
0      NaN  0     1  NaN  NaN     1
1       1   1    NaN  NaN   NaN     0
2       NaN  1     0   1   NaN
3       NaN  0     1   0   NaN

>>> pd.concat([spam_bag_of_words, other_bag_of_words], ignore_index=True).fillna(0)
  antibody  buy hello time  viagra
0      NaN  0     1  NaN  NaN     1
1       1   1     0  NaN  NaN
2       0   1     0   1   0
3       0   0     1   0   0
```
Combining datasets

See [http://pandas.pydata.org/pandas-docs/dev/merging.html](http://pandas.pydata.org/pandas-docs/dev/merging.html) for other Pandas operations:

- `concat`
- `join`
- `merge`
- `combine_first`
Pandas

Join example

Two data sets with partially overlapping rows (as not all students answer each questionnaire) where the columns should be concatenated (i.e., scores for individual questionnaires)

```python
import pandas as pd

xl = pd.ExcelFile("E13_1_Resultater-2013-10-02.xlsx")
df1 = xl.parse("Resultater", index_col=[0, 1, 2], header=3)
df1.columns = map(lambda colname: unicode(colname) + "_1", df1.columns)

xl = pd.ExcelFile("E13_2_Resultater-2013-10-02.xlsx")
df2 = xl.parse("Resultater", index_col=[0, 1, 2], header=3)
df2.columns = map(lambda colname: unicode(colname) + "_2", df2.columns)

df = pd.DataFrame().join([df1, df2], how="outer")
df[["Score_1", "Score_2"]].corr()  # Score correlation
```

Finn Årup Nielsen
Processing after join

```python
>>> df.ix[:5,["Score_1", "Score_2"]]

<table>
<thead>
<tr>
<th>Bruger</th>
<th>Fornavn</th>
<th>Efternavn</th>
<th>Score_1</th>
<th>Score_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(faan)</td>
<td>Finn Årup</td>
<td>Nielsen</td>
<td>1.000000</td>
<td>1.000000</td>
</tr>
<tr>
<td>s06...</td>
<td>...</td>
<td></td>
<td>0.409467</td>
<td>NaN</td>
</tr>
<tr>
<td>s07...</td>
<td>...</td>
<td></td>
<td>NaN</td>
<td>0.870900</td>
</tr>
<tr>
<td>s07...</td>
<td>...</td>
<td></td>
<td>0.576568</td>
<td>0.741800</td>
</tr>
<tr>
<td>s07...</td>
<td>...</td>
<td></td>
<td>0.686347</td>
<td>0.569666</td>
</tr>
</tbody>
</table>
```

(edited)

Note that the second user (“s06...”) did not solve the second assignment. The joining operation by default adds a NaN to the missing element, indicating a missing value (not available, NA).
The Groupby

Groupby method (McKinney, 2012, chapter 9): splits the dataset based on a key, e.g., a DataFrame column name.

Think of SQL’s GROUP BY.

Example with Pima Indian data set splitting on the 'type' column (elements are “yes” and “no”) and taking the mean in each of the two groups:

```python
>>> pima.groupby("type").mean()

  npreg    glu    bp    skin    bmi    ped   age
type
No  2.932735 108.188341 70.130045 27.340807 31.639910 0.464565 29.215247
Yes 4.614679 141.908257 74.770642 32.889908 36.512844 0.658963 35.614679
```

The returned object from groupby is a `DataFrameGroupBy` object while the `mean` method on that object/class returns a `DataFrame` object.
The Groupby

More elaborate with two aggregating methods:

```python
>>> grouped_by_type = pima.groupby("type")
>>> grouped_by_type.agg([np.mean, np.std])
```

<table>
<thead>
<tr>
<th></th>
<th>npreg</th>
<th>glu</th>
<th>bp</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>std</td>
<td>mean</td>
<td>std</td>
</tr>
<tr>
<td>type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>2.932735</td>
<td>2.781852</td>
<td>108.188341</td>
</tr>
<tr>
<td>Yes</td>
<td>4.614679</td>
<td>3.901349</td>
<td>141.908257</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>skin</th>
<th>bmi</th>
<th>ped</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>std</td>
<td>mean</td>
<td>std</td>
<td>mean</td>
</tr>
<tr>
<td>type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>27.340807</td>
<td>9.567705</td>
<td>31.639910</td>
<td>6.648015</td>
</tr>
<tr>
<td>Yes</td>
<td>32.889908</td>
<td>9.065951</td>
<td>36.512844</td>
<td>7.457548</td>
</tr>
</tbody>
</table>

```

<table>
<thead>
<tr>
<th></th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>10.131493</td>
</tr>
<tr>
<td>Yes</td>
<td>10.390441</td>
</tr>
</tbody>
</table>

Finn Årup Nielsen
... The Groupby

Without groupby checking mean (32.889908) and std (9.065951) for 'skin'='Yes':

```python
>>> np.mean(pima[pima["type"]="Yes"]["skin"])
32.889908256880737 # Correct
```

```python
>>> np.std(pima[pima["type"]="Yes"]["skin"])
9.0242684519300891 # ???
```

```python
>>> import scipy.stats
>>> scipy.stats.nanstd(pima[pima["type"]="Yes"]["skin"])
9.065951207005341 # Ok
```

```python
>>> np.std(pima[pima["type"]="Yes"]["skin"], ddof=1)
9.065951207005341 # Degrees of freedom!
```

Numpy’s std is the biased estimate while Pandas std is the unbiased estimate.
Cross-tabulation

For categorical variables select two columns and generate a matrix with counts for occurrences (McKinney, 2012, p. 277)

```python
>>> pd.crosstab(pima.type, pima.npreg)

<table>
<thead>
<tr>
<th>npreg</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>15</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>No</td>
<td>34</td>
<td>56</td>
<td>38</td>
<td>23</td>
<td>19</td>
<td>13</td>
<td>14</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>15</td>
<td>15</td>
<td>11</td>
<td>15</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Remember:

```python
>>> pima[1:4]

<table>
<thead>
<tr>
<th>npreg</th>
<th>glu</th>
<th>bp</th>
<th>skin</th>
<th>bmi</th>
<th>ped</th>
<th>age</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>85</td>
<td>66</td>
<td>29</td>
<td>26.6</td>
<td>0.351</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>89</td>
<td>66</td>
<td>23</td>
<td>28.1</td>
<td>0.167</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>78</td>
<td>50</td>
<td>32</td>
<td>31.0</td>
<td>0.248</td>
<td>26</td>
</tr>
</tbody>
</table>
```
# Wrong ordering

```
pd.crosstab(pima.type, pima.npreg).plot(kind="bar")
```
Cross-tabulation plot

# Transpose
pd.crosstab(pima.type, pima.npreg).T.plot(kind="bar")
Cross-tabulation plot

# Or better:
pd.crosstab(pima.npreg, pima.type).plot(kind="bar")
Other Pandas capabilities

Hierarchical indexing (McKinney, 2012, page 147+)

Missing data support (McKinney, 2012, page 142+)

Pivoting (McKinney, 2012, chapter 9)

Time series (McKinney, 2012, chapter 10)
Statistical modeling with statsmodels

Example with Longley dataset.

Ordinary least squares fitting a dependent variable “TOTEMP” (Total Employment) from 6 independent variables:

```python
import statsmodels.api as sm

# For 'load_pandas' you need a recent statsmodels
data = sm.datasets.longley.load_pandas()

# Endogeneous (response/dependent) & exogeneous variables (design matrix)
y, x = data.endog, data.exog

result = sm.OLS(y, x).fit()  # OLS: ordinary least squares
result.summary()            # Print summary
```
OLS Regression Results

<table>
<thead>
<tr>
<th>Dep. Variable:</th>
<th>TOTEMP</th>
<th>R-squared:</th>
<th>0.988</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model:</td>
<td>OLS</td>
<td>Adj. R-squared:</td>
<td>0.982</td>
</tr>
<tr>
<td>Method:</td>
<td>Least Squares</td>
<td>F-statistic:</td>
<td>161.9</td>
</tr>
<tr>
<td>Date:</td>
<td>Mon, 17 Jun 2013</td>
<td>Prob (F-statistic):</td>
<td>3.13e-09</td>
</tr>
<tr>
<td>Time:</td>
<td>13:56:35</td>
<td>Log-Likelihood:</td>
<td>-117.56</td>
</tr>
<tr>
<td>No. Observations:</td>
<td>16</td>
<td>AIC:</td>
<td>247.1</td>
</tr>
<tr>
<td>Df Residuals:</td>
<td>10</td>
<td>BIC:</td>
<td>251.8</td>
</tr>
<tr>
<td>Df Model:</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| coef   | std err | t     | P>|t|   | [95.0% Conf. Int.] |
|--------|---------|-------|-------|-------------------|
| GNPDEFL | -52.9936 | 129.545 | -0.409 | 0.691 | -341.638 235.650 |
| GNP     | 0.0711   | 0.030  | 2.356 | 0.040 | 0.004 0.138 |
| UNEMP   | -0.4235  | 0.418  | -1.014 | 0.335 | -1.354 0.507 |
| ARMED   | -0.5726  | 0.279  | -2.052 | 0.067 | -1.194 0.049 |
| POP     | -0.4142  | 0.321  | -1.289 | 0.226 | -1.130 0.302 |
| YEAR    | 48.4179  | 17.689 | 2.737 | 0.021 | 9.003 87.832 |

Omnibus: 1.443 Durbin-Watson: 1.277
Prob(Omnibus): 0.486 Jarque-Bera (JB): 0.605
Skew: 0.476 Prob(JB): 0.739
Kurtosis: 3.031 Cond. No. 4.56e+05

Finn Årup Nielsen 29 October 5, 2013
Statsmodels > 0.5

“Minimal example” from statsmodels documentation:

```python
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

url = "http://vincentarelbundock.github.io/Rdatasets/csv/HistData/Guerry.csv"
dat = pd.read_csv(url)
results = smf.ols("Lottery ~ Literacy + np.log(Pop1831)", data=dat).fit()
results.summary()
```

Note: 1) Loading of data with URL, 2) import statsmodels.formula.api (possible in statsmodels > 0.5), 3) R-like specification of linear model formula (from patsy).
More information

http://pandas.pydata.org/

The canonical book “Python for data analysis” (McKinney, 2012).

Will it Python?: Porting R projects to Python, exemplified though scripts from Machine Learning for Hackers (MLFH) by Drew Conway and John Miles White.
Pandas helps you represent your data (both numerical and categorical) and helps you keep track of what they refer to (by column and row name).

Pandas makes indexing easy.

Pandas has some basic statistics and plotting facilities.

Pandas may work more or less seamlessly with standard statistical models (e.g., general linear model with OLS-estimation)

Watch out: Pandas is still below version 1 numbering!

Standard packaging not up to date: Newest version of Pandas is 0.11.0, while, e.g., Ubuntu LTS 12.04 is 0.7.0: sudo pip install --upgrade pandas

Latest pip-version of statsmodels is 0.4.3, development version is 0.5 with statsmodels.formula.api that yields more R-like linear modeling.
References