

Principles of Programming in Econometrics

Introduction, structure, and advanced programming techniques

Charles S. Bos

Vrije Universiteit Amsterdam
Tinbergen Institute

`c.s.bos@vu.nl`

August 2020 – Version Python

Separate lecture slides

Compilation: August 14, 2020

Overview

Principles of Programming in Econometrics

D0: Syntax, example 2⁸

D1: Structure, scope

D2: Numerics, packages

D3: Optimisation, speed

Day 2: Numerics and flow

9.30 Numbers and representation

- ▶ Steps, flow and structure
- ▶ Floating point numbers
- ▶ Practical Do's and Don'ts
- ▶ Packages
- ▶ Graphics

13.30 Practical

- ▶ Cleaning OLS program
- ▶ Loops
- ▶ Bootstrap OLS estimation
- ▶ Handling data: Inflation

A module: Pandas

Extensive set of data analytics and data handling routines, [Pandas](#).

Goal:

- ▶ Loading/saving
- ▶ Indexing/selecting
- ▶ Manipulating
- ▶ ...

A module: Pandas

Extensive set of data analytics and data handling routines, [Pandas](#).

Goal:

- ▶ Loading/saving
- ▶ Indexing/selecting
- ▶ Manipulating
- ▶ ...
- ▶ Printing nicely
- ▶ Plotting
- ▶ and other?

Initialisation:

```
import pandas as pd
```

Pandas Types

From Pandas we'll use two types:

- ▶ DataFrame: matrix-like format, with row index and columns names
- ▶ Series: vector-like format, with row index and name

```
import pandas as pd

sData= 'shoesize_bk2020'

# Initialisation
df= pd.read_csv('data/%s.csv' % sData)      # DataFrame
sf= df['Gender']                             # Series

print ('Type df: %s\nType sf: %s' % (type(df), type(sf)))
```

NB: Normally, work with the DataFrame itself... Not much use to extract the separate series

Pandas Types II

Instead of reading data into a DataFrame, we can also create one based on data:

```
dfR= pd.DataFrame(np.random.randn(10,4), columns=['a', 'b', 'c', 'd'])  
print (dfR)  
print (dfR.to_latex(float_format='%.4f'))
```

Why?

- ▶ To store a set of results, in a convenient dataframe
- ▶ Also, to print them in a clean format (even as L^AT_EX)

Pandas Input files

Reading files: Use `df= pd.read_...` with

- ▶ `csv`: Clean input, easy to check in editor or excel, but large in size
- ▶ `excel`: Convenient, but a bit dangerous as each version of excel behaves differently
- ▶ `csv.gz`: Gzipped csv, smaller
- ▶ `hdf`, `pickle`, ...: Many formats [available](#)

Extra options (and many others):

- ▶ **CSV**: `skiprows=8`, `sep=';'`, for choosing to skip some input, or indicate the separator
- ▶ **Excel**: `sheet_name='Sheet 2'`, `usecols=[0, 3, 4]`, for choosing specific sheet, or only some columns
- ▶ with both: `index_col=['Year', 'Period']`, to indicate what column(s) will be the index

Pandas elements

Check the contents of the DataFrame and Series, either printing all, or only the `.head()` or `.tail()`:

```
print ('Head of df: \n', df.head(), sep='')  
print ('Tail of sf:\n', sf.tail(), sep='')
```

resulting in

Head of df:				Tail of sf:	
	Shoesize	Length	Gender	114	Male
0	45.0	187.0	Male	115	Male
1	40.0	180.0	Female	116	Male
2	45.0	185.0	Male	117	Male
3	43.0	185.0	Male	118	Male
4	43.0	174.0	Male	Name: Gender, dtype: object	

Notice: index 0, ..., 118, columns Shoesize, Length, Gender, Name: Gender

Pandas: Information

Check out the contents of the data with e.g.

- ▶ `df.head()`, `df.tail()`, `df`: Either show a part, or the full data frame (or a limited number of rows and columns, that is)
- ▶ `df.info()`: More detailed information on the contents
- ▶ `df.mean()`, `df.var()`, `df.min()`, `df.max()`: Find the mean/var/min/max over the columns
- ▶ `df.shape`: What size is it?
- ▶ `df.index`, `df.columns`: What are the row/column indices?
- ▶ ...

and especially:

- ▶ `df.values`: Extract the *values* from the dataframe, as a numpy matrix...!

Pandas: Indexing

Different methods:

<code>asC= ['Shoesize', 'Length']; asR= range(4, 8)</code>	
<code>df[asC]</code>	Select <i>columns</i> by name
<code>vI= df['Gender'] == 'Male'; df[vI]</code>	Select <i>rows</i> by boolean indexing
<code>df.loc[asR,:]</code>	Select rows by index, all columns
<code>df.loc[asR, asC]</code>	Subset of rows and columns
<code>df.iloc[8, 2]</code>	Read out single element, indexed column location
<code>df.iloc[vR, vC]</code>	Subset of rows and columns, in ranges

Remarks:

- ▶ Needs practice...
- ▶ I regularly move to a NumPy matrix/array, leaving DataFrames only for input/output

Pandas: Advanced indexing I

What if I want to find the average length of the males?

- a. Index, find only the males: `vI= df['Gender'] == 'Male';
dfM= df[vI]; dfM['Length'].mean()`
- b. Move to *wide* instead of *long* table...

Definition:

- ▶ Long format: All subjects are placed one below the other, with observations on the necessary variables in a single row
- ▶ Wide format: Observations on several types of subjects may be placed next to each other, for the same *index*

Pandas: Long vs wide

df - DataFrame (Long Table)

Index	Shoesize	Length	Gender
0	45	187	Male
1	40	180	Female
2	45	185	Male
3	43	185	Male
4	43	174	Male
5	43	184	Male
6	40	178	Female
7	44	183	Male
8	44	187	Male
9	42.5	184	Male
10	44	182	Male
11	40	170	Female
12	41	178	Female
13	40	175	Female

df1 - DataFrame (Wide Table)

Index	0	1	2	3
None	Shoesize	Shoesize	Length	Length
Gender	Female	Male	Female	Male
0	nan	45	nan	187
1	40	nan	180	nan
2	nan	45	nan	185
3	nan	43	nan	185
4	nan	43	nan	174
5	nan	43	nan	184
6	40	nan	178	nan
7	nan	44	nan	183
8	nan	44	nan	187
9	nan	42.5	nan	184
10	nan	44	nan	182
11	40	nan	170	nan

Long vs. wide table

```
df1= df.pivot(columns='Gender', values=['Shoesize', 'Length'])
df1[asC].mean()           # Give means of both values, per Gender
```

Here: Not too useful. But what about data with observations for each month/quarter/half year?

Pandas: Advanced indexing II

With pivoted table, one gets to **MultiIndex** tables:

```
In[74]: df1.columns
Out[74]: MultiIndex([( 'Shoesize', 'Female'),
                    ( 'Shoesize', 'Male'),
                    ( 'Length', 'Female'),
                    ( 'Length', 'Male')],
                   names=[None, 'Gender'])
```

Or: Index contains both variable name and pivot value, in a *tuple*.
Hence: Select a single column with a *tuple* etc:

```
df1[( 'Shoesize', 'Male')].mean()    # Single mean
df1[ 'Shoesize'].mean()              # Both Female and Male means
```

Warning: Do try this at home... Options, way to work with MultiIndex, takes *lots* of practice...

Pandas: Saving

With data, you also want to save... Options: **Many...**

Personal preference (with e.g. `sData='shoesize_bk2020'`):

1. `df.to_csv('data/%s_out.csv' % sData)`: Clean csv file, easy to read in editor or excel, robust
2. `df.to_csv('data/%s_out.csv.gz' % sData)`: Clean csv file, but gzipped: Smaller, quite easy to read in editor or excel
3. `df.to_excel('data/%s_out.xlsx' % sData)`: Pure excel file (but with limits on number of columns/rows!)
4. `df.to_excel('data/%s_out.ods' % sData)`: Pure OpenDocument format file (but with limits on number of columns/rows!)

Extra options:

- ▶ `df.to...(sOut, index=False)`: Do not write the index column along (sometimes not informative)



`df.to_excel(sOut, sheet_name='BK2020', sheet_size=14/15)`

Pandas: Plotting

Plotting is a separate chapter, with **too many details** to cover here.
Hence an example:

```
df.plot.area(figsize=(8,4))  
df.plot.area(subplots=True)  
df.plot.density(subplots=True)  
plt.figure(figsize=(8,4)); df.plot.box(); plt.savefig('graphs/shoesize_box'); plt.show()
```

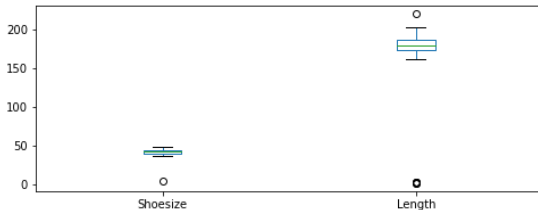


Figure: Shoesize and length of 2020 class of BK Statistics

Q: What is the origin of those dots at the bottom of the figure?