Berlin School of Economics and Law
Prof. Dr. Natalie Packham

Summer term 2020
LV-Nr. 425093

Computational Finance and FinTech

File I/O

Contents

5 Input/Output (IO)

Input / Output

- Further reading: **Py4Fi, Chapter 9**
- This chapter is about loading data from and storing data to a persistent data source (e.g. hard drive).
- Often financial data comes as `.csv`-files or `xlsx-` / `xls`-files.
- Before working with the data it must be loaded into Python first.
- Data can be loaded, written and stored using Python's built-in functions or `csv` or `pandas`.
- The book contains additional information on reading from and writing to SQL databases.

Input / Output

- The usual initialisation:

[1]:
```
import numpy as np
import pandas as pd
from pylab import mpl, plt
plt.style.use('seaborn')
%matplotlib inline
```

5.1 Basic IO with Python

CSV files

- A popular data format for working with spreadsheet-like data is **comma-separated values**, abbreviated as `.csv`.
- In a csv file, data entries are stored in rows, with column separated by comma.
- The top row may contain the column names.

CSV files

- Create a dummy data set to demonstrate how to work with csv files:

[2]:
```
rows = 10000
a = np.random.standard_normal((rows, 5)).round(4)
a[:2]
```

[2]:
```
array([[ 0.3811,  0.5971, -0.849 , -0.1402, -0.0285],
       [-0.7632,  0.713 , -1.0156,  0.6465, -0.4061]])
```

[3]:
```
t = pd.date_range(start='2019/1/1', periods=rows, freq='H') # creates a␣
↪DateTimeIndex
```

[4]:
```
t[:4]
```

[4]:
```
DatetimeIndex(['2019-01-01 00:00:00', '2019-01-01 01:00:00',
               '2019-01-01 02:00:00', '2019-01-01 03:00:00'],
              dtype='datetime64[ns]', freq='H')
```

Note: 'H' is used for an hourly frequency, see here for more information about the `freq` settings.

Creating a .csv-file in Python

- `open()` with the `w` flag creates an empty file.

[5]:
```
import os
path = os.getcwd() + '/' # the current working directory
path
```

[5]:
```
'/Users/nat/Documents/GitHub/compfin_dev/'
```

[6]:
```
csv_file = open(path + 'RandomNumbers.csv', 'w') # opens a file for writing
header = 'date,no1,no2,no3,no4,no5\n'
csv_file.write(header) # defines the header row and writes it as the first line
```

[6]:
```
25
```

Creating a .csv-file in Python

- The data is written to the `.csv`-file using Python's built-in function `write()`:

[7]:
```
for t_, (no1, no2, no3, no4, no5) in zip(t, a): # combines the data row-wise ...
    s = '{},{},{},{},{},{}\n'.format(t_, no1, no2, no3, no4, no5) # ... into string␣
↪objects ...
    csv_file.write(s) # ... and writes it to the file line-by-line
```

- The changes are saved by calling `close()`.

[8]:
```
csv_file.close()
```

[9]:
```
ls -l RandomN* # Here it is!
```

```
-rw-r--r--@ 1 nat  staff  569440  3 Apr 15:54 RandomNumbers.csv
-rw-r--r--  1 nat  staff  445150  3 Apr 15:53 RandomNumbers.xlsx
-rw-r--r--  1 nat  staff  579560  3 Apr 15:53 RandomNumbers_2.csv
-rw-r--r--  1 nat  staff  603528  3 Apr 15:53 RandomNumbers_3.csv
```

Reading a .csv-file in Python

[10]:
```
csv_file = open(path + 'RandomNumbers.csv', 'r') # open file for reading ('r')
RandomNumbers = csv_file.readlines() # read the file contents in one step
RandomNumbers[:5]
```

[10]:
```
['date,no1,no2,no3,no4,no5\n',
 '2019-01-01 00:00:00,0.3811,0.5971,-0.849,-0.1402,-0.0285\n',
 '2019-01-01 01:00:00,-0.7632,0.713,-1.0156,0.6465,-0.4061\n',
 '2019-01-01 02:00:00,-0.0329,0.355,2.6329,-0.1466,0.8553\n',
 '2019-01-01 03:00:00,1.0199,-0.6185,0.7659,-1.1717,-0.6345\n']
```

[11]:
```
csv_file.close() # close the file
```

csv reader

- CSV files are so common that a dedicated package for reading (and writing) CSV files exists.

[12]:
```
import csv

with open(path + 'RandomNumbers.csv', 'r') as f:
    numbers = csv.reader(f) # returns every line as a list object
    lines = [line for line in numbers]

lines[:5]
```

[12]:
```
[['date', 'no1', 'no2', 'no3', 'no4', 'no5'],
 ['2019-01-01 00:00:00', '0.3811', '0.5971', '-0.849', '-0.1402', '-0.0285'],
 ['2019-01-01 01:00:00', '-0.7632', '0.713', '-1.0156', '0.6465', '-0.4061'],
 ['2019-01-01 02:00:00', '-0.0329', '0.355', '2.6329', '-0.1466', '0.8553'],
 ['2019-01-01 03:00:00', '1.0199', '-0.6185', '0.7659', '-1.1717', '-0.6345']]
```

[13]:
```
type(lines)
```

[13]:
```
list
```

Note: Aside of a `list`, the `csv` module can also return an `OrderedDict`, by using `DictReader()`. To learn more about the object `OrderedDict`, see here.

csv writer

- CSV files are easily written to a file using the `csv` package.
- Write the `.csv`-file, using the functions `writer()` and `writerow()`.

[14]:
```
with open(path + 'RandomNumbers_2.csv', 'w') as f:
    numbers = csv.writer(f, delimiter = ',')
    for line in lines:
        numbers.writerow(line)
```

5.2 CSV files with `pandas`

- Some of the data formats that `pandas` can read/write:
  - CSV (Comma-separated values)
  - SQL (Structured query language)
  - XLS / XLSX (Microsoft Excel files)
  - JSON (Javascript object notation)
  - HTML (Hyptertext markup language)

CSV files with `pandas`

- Reading a CSV file with pandas loads the data into a `DataFrame` object.
- The first line of of the file is assumed to be the header.

[15]:
```
df = pd.read_csv('RandomNumbers.csv')
```

[16]:
```
df.head()
```

[16]:
```
                  date     no1     no2     no3     no4     no5
0  2019-01-01 00:00:00  0.3811  0.5971 -0.8490 -0.1402 -0.0285
1  2019-01-01 01:00:00 -0.7632  0.7130 -1.0156  0.6465 -0.4061
2  2019-01-01 02:00:00 -0.0329  0.3550  2.6329 -0.1466  0.8553
3  2019-01-01 03:00:00  1.0199 -0.6185  0.7659 -1.1717 -0.6345
4  2019-01-01 04:00:00 -1.7502  0.6594  0.0146  0.0576 -1.0475
```

CSV files with pandas

- `pd.read_csv()` adds an index by default.
- To use a different index, add `set_index(columnName)`.

[17]:
```
df.set_index('date', inplace=True)
df.head()
```

[17]:
```
                        no1     no2     no3     no4     no5
date
2019-01-01 00:00:00  0.3811  0.5971 -0.8490 -0.1402 -0.0285
2019-01-01 01:00:00 -0.7632  0.7130 -1.0156  0.6465 -0.4061
2019-01-01 02:00:00 -0.0329  0.3550  2.6329 -0.1466  0.8553
2019-01-01 03:00:00  1.0199 -0.6185  0.7659 -1.1717 -0.6345
2019-01-01 04:00:00 -1.7502  0.6594  0.0146  0.0576 -1.0475
```

Writing a `.csv`-file using pandas

[18]:
```
df['no1'] = df['no1'] + 1000
df.to_csv('RandomNumbers_3.csv') # couldn't be easier...
```

5.3 Excel-files with pandas

- Microsoft Excel files can be read and created easily using `pandas`.
- This is quite similar to reading / writing CSV files.

[19]:
```
df.to_excel('RandomNumbers.xlsx')
```

[20]:
```
ls -l RandomNumbers*
```

```
-rw-r--r--@ 1 nat  staff  569440  3 Apr 15:54 RandomNumbers.csv
-rw-r--r--  1 nat  staff  445640  3 Apr 15:54 RandomNumbers.xlsx
-rw-r--r--  1 nat  staff  579441  3 Apr 15:54 RandomNumbers_2.csv
-rw-r--r--  1 nat  staff  603359  3 Apr 15:54 RandomNumbers_3.csv
```

Reading a `.xlsx`-file using pandas

[21]:
```
df = pd.read_excel('RandomNumbers.xlsx', index_col=0)
```

[22]:
```
df.head()
```

[22]:
```
                           no1     no2     no3     no4     no5
date
```

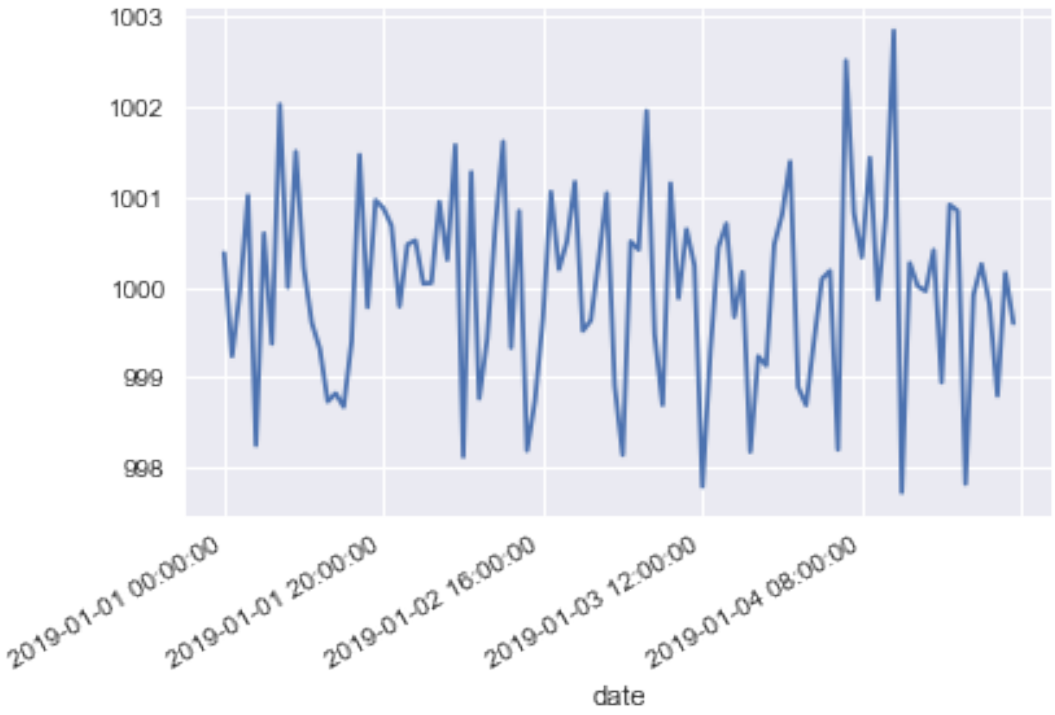
```
2019-01-01 00:00:00  1000.3811  0.5971 -0.8490 -0.1402 -0.0285
2019-01-01 01:00:00   999.2368  0.7130 -1.0156  0.6465 -0.4061
2019-01-01 02:00:00   999.9671  0.3550  2.6329 -0.1466  0.8553
2019-01-01 03:00:00  1001.0199 -0.6185  0.7659 -1.1717 -0.6345
2019-01-01 04:00:00   998.2498  0.6594  0.0146  0.0576 -1.0475
```

5.4 Saving plots

- This section demonstrates how to make plots persistent for further processing.
- The file format is typically detected by the file name extensions.

[23]:
```
from matplotlib import pyplot as plt
import matplotlib.dates as mdates
```

[24]:
```
fig, ax = plt.subplots()
df['no1'][:100].plot()
fig.autofmt_xdate()
plt.savefig('LineChart.png')
```



Generating a `pdf` plot

[25]:
```
fig, ax = plt.subplots()
df['no1'][:100].plot()
fig.autofmt_xdate()
plt.savefig('LineChart.pdf')
```

Generating plots

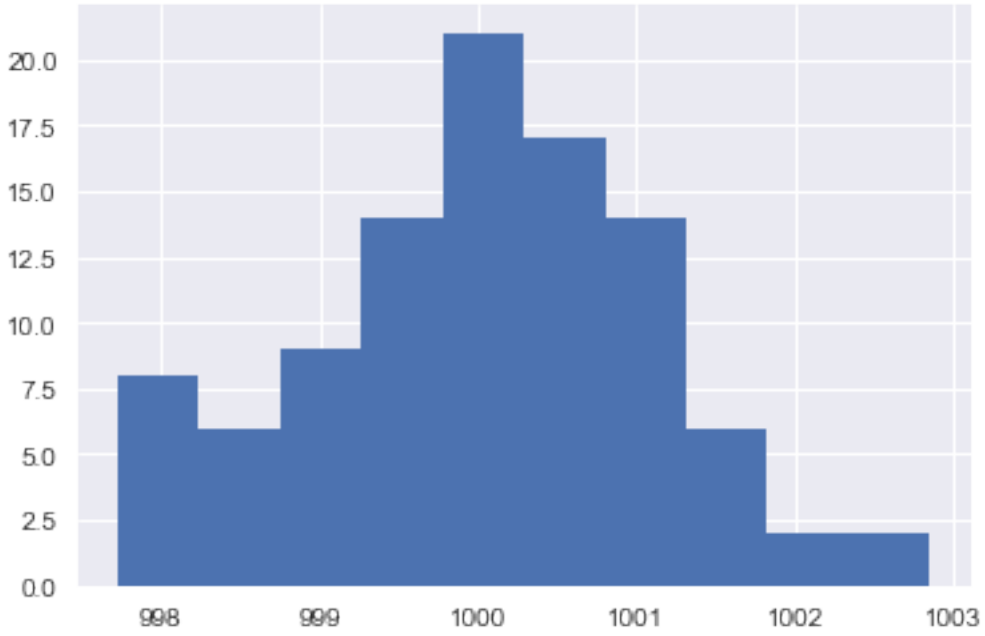
- There are various options for fine-tuning the appearance and quality of the plot.
- Let's check that both plots have been written to the hard disk:

[26]:
```
ls -l LineChart*
```

```
-rw-r--r--  1 nat  staff   9891  3 Apr 15:54 LineChart.pdf
-rw-r--r--@ 1 nat  staff  31715  3 Apr 15:54 LineChart.png
```

Another pdf plot

[27]:
```
df['no1'][1:100].hist()
plt.savefig('Hist.pdf')
```



5.5 Direct download from the internet: `pandas-datareader`

- Many data providers offer direct downloads through an API (application programming interface), an programming interface that controls data access and the data format.
- In other words, instead of manually downloading and subsequently importing a file, the data download can directly be incorporated into the program.
- The package `pandas-datareader` provides a convenient unified way of extracting data from various Internet sources into a pandas `DataFrame`.

Installing `pandas-datareader`

- Installation via anaconda: go to environments in the left menubar, enter `datareader` in `search packages` and install.
- Alternatively, use a terminal programme (command line) and type:
  - `pip install pandas-datareader`
- This is a frequently changing package... I had to use:
  - `pip install git+https://github.com/pydata/pandas-datareader.git`
- For more information, see here and here.

Using `pandas-datareader`

Using TIINGO requires that you create a free account and obtain an API key.

[28]:
```
import os
import pandas_datareader as pdr
```

```
/Users/natalie/anaconda3/lib/python3.7/site-
packages/pandas_datareader/compat/__init__.py:9: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at
pandas.testing instead.
  from pandas.util.testing import assert_frame_equal
```

[29]:
```
df = pdr.get_data_tiingo('GOOG', api_key=os.getenv('TIINGO_API_KEY'))
df.head()
```

[29]:

symbol	date	close	high	low	open	volume
GOOG	2015-04-06 00:00:00+00:00	536.765	538.41	529.57	532.22	1320767
	2015-04-07 00:00:00+00:00	537.020	542.69	536.00	538.08	1299298
	2015-04-08 00:00:00+00:00	541.610	543.85	538.38	538.38	1175332
	2015-04-09 00:00:00+00:00	540.780	541.95	535.49	541.03	1553586
	2015-04-10 00:00:00+00:00	540.010	542.29	537.31	542.29	1405574

symbol	date	adjClose	adjHigh	adjLow	adjOpen
GOOG	2015-04-06 00:00:00+00:00	536.765	538.41	529.57	532.22
	2015-04-07 00:00:00+00:00	537.020	542.69	536.00	538.08
	2015-04-08 00:00:00+00:00	541.610	543.85	538.38	538.38
	2015-04-09 00:00:00+00:00	540.780	541.95	535.49	541.03
	2015-04-10 00:00:00+00:00	540.010	542.29	537.31	542.29

symbol	date	adjVolume	divCash	splitFactor
GOOG	2015-04-06 00:00:00+00:00	1320767	0.0	1.0
	2015-04-07 00:00:00+00:00	1299298	0.0	1.0
	2015-04-08 00:00:00+00:00	1175332	0.0	1.0
	2015-04-09 00:00:00+00:00	1553586	0.0	1.0
	2015-04-10 00:00:00+00:00	1405574	0.0	1.0