Python for Finance

Data Analysis with Pandas

Andras Niedermayer

DAUPHINE
UNIVERSITÉ PARIS

Outline

Writing objects to disk: pickle

First, let us generate a list of 10,000 numbers from the standard normal distribution.

```
1 import numpy as np
2 from random import gauss
3 a=[gauss(0,1) for i in range(10000)]
```

Then, we save it to a pickle file:

```
1 import pickle
2 # open file with write permission
3 file=open('data.pkl','w')
4 # dump variable into file
5 pickle.dump(a, file)
6 # close file
7 file.close()
```

Loading objects from disk: pickle

To load the pickle file from the disk:

```
1  # open file
2  file=open('data.pkl','r')
3  # dump variable into Python
4  b=pickle.load(file)
```

1. You can store more than one object in a pickle.
2. However, they are retrieved sequentially (FIFO):
   First object you load is the first object you saved, etc.
3. Creating a dictionary (dict) of all objects helps indexing all objects in the pickle.

Writing objects to disk: csv

We generate a 5 by 3 matrix of numbers from the standard normal distribution. Row names are from A to E.

```
1  m=np.random.standard_normal((5,3))
2  rownames=['A', 'B', 'C', 'D', 'E']
```

We create a header and save both the data and row names to csv:

```
1  # open file
2  csv_file=open('data.csv','w')
3  # generate header
4  header="row name, v1, v2, v3\n"
5  # add data row-by-row
6  for r_, (v1, v2, v3) in zip(rownames,m):
7      s="%s, %f, %f, %f\n" %(r_,v1,v2,v3)
8      # 1 string, 3 floats
9      csv_file.write(s) # write the row
10 csv_file.close()      # close file
```

Loading objects from disk: csv

Read the csv line by line:

```
1  # open file
2  csv_file=open('data.csv','r')
3  for i in range(5):
4      print csv_file.readline()
```

Read the csv all at once:

```
1  csv_file=open('data.csv','r')
2  content=csv_file.readlines()
```

Outline

I/O in pandas

pandas is the Python Data Analysis library:
http://pandas.pydata.org/
Advantage: Can read and write multiple formats natively.

Format	Input	Output	Type
CSV	from_csv	to_csv	CSV file
XLS(X)	from_excel	to_excel	Excel file
DTA	from_stata	to_stata	Stata file
Pickle	from_pickle	to_pickle	Pickles
HTML	from_html	to_html	HTML code

Also works with: HDF, SQL databases, JSON (JavaScript)...

`DataFrame` objects

`DataFrame` objects are indexed tables (panels) – similar to an Excel spreadsheet.

1. Each "cell" can contain any Python data type (lists, dictionaries...);
2. Data is organized on columns: each column is a variable;
3. Data can be indexed (for instance, by date or time).
4. A `Series` object is a single column of a `DataFrame`. Generally, all `DataFrame` methods also work for series.

Loading web-based data

Next, we gather data from Yahoo! on the three financial indices. The data are provided for convenience on my website.

```
1  from pandas import *
2  # load data on the CAC40
3
4  CAC40=read_csv('http://andras.niedermayer.ch'
5        + '/wp-content/uploads/2018/02/CAC40.csv')
6  CAC40['Index']='CAC40' # index name
7
8  # do the same for DAX.csv and FTSE.csv, name
9  # the variables DAX and FTSE
10 ....
11
12 IndicesA=DataFrame() # the overall dataframe
13 for db in [CAC40, DAX, FTSE]:
14     # append all the individual indices
1      IndicesA=IndicesA.append(db,ignore_index=True)
```

Basic operations

1. See all variables in the dataset

```
IndicesA.columns.tolist()
```

2. Getting back the index list (application of drop_duplicates)

```
IndicesA['Index'].drop_duplicates().tolist()
```

3. Selection via index

```
Indices.ix[[1856,3733,5591]]
```

4. Appending databases

```
CACDAXApp=CAC40.append(DAX, ignore_index=True)
```

5. Merging databases

```
CACDAXMerge=CAC40.merge(DAX, on="Date")
```

More on merging databases

The *how* argument to merge specifies how to determine which keys are to be included in the resulting table.

Merge method	Description
left	Use keys from left frame only
right	Use keys from right frame only
outer	Use union of keys from both frames
inner	Use intersection of keys from both frames

The join method

The `.join` method works just as `.merge`, only that it automatically uses the index, and not a specific column.

Summary statistics

1. General summary statistics: count, min, max, mean, quantiles...

```
IndicesA.describe()
```

2. Summary statistics by column (all variables)

```
IndicesA.mean()
IndicesA.std()
IndicesA.median()
```

3. Summary statistics of specific variables

```
IndicesA['Open'].mean()
IndicesA['Close'].std()
```

Dealing with missing values

1. Filling missing values with the previous available value:

```
IndicesA.fillna(method='bfill')
```

2. Filling missing values with the next available value:

```
IndicesA.fillna(method='ffill')
```

3. Filling missing values with a specific value (e.g., zero):

```
IndicesA.fillna(0)
```

Applying functions on data

1. New columns as a function of old ones:

```
IndicesA['HML']=
  IndicesA['High']/IndicesA['Low']
```

2. Applying functions using `map` method.

```
IndicesA['LogReturn']=
  IndicesA['Return'].map(np.log)
```

3. More complex functions (no control flow): `apply` method.

```
IndicesA['Green']=
  IndicesA['Return'].apply(lambda x: x>1)
```

What does 'Green' mean?

Conditional analysis

1. Select only rows that correspond to a criteria:

```
DAX=IndicesA[IndicesA['Index']=='DAX']
IndicesPlus=IndicesA[IndicesA['Return']>=1]
```

2. Multiple criteria (and, or, not):

```
1) IndicesPlusDAX=
  IndicesA[(IndicesA['Index']=='DAX')
      & (IndicesA['Return']>=1)]
2) IndicesDAXCACa=
  IndicesA[(IndicesA['Index']=='DAX')
   | (IndicesA['Index']=='CAC40')]
3) IndicesDAXCACb=
  IndicesA[~(IndicesA['Index']=='FTSE')]
```

GroupBy analysis

Pandas can provide summary statistics for each group.

```
Indices.groupby('Index').count()['Return']
```

Index	Count
CAC40	1785
DAX	1777
FTSE	1785

```
IndicesA.groupby('Index').mean()['Return']
```

Index	Mean
CAC40	0.9998
DAX	0.9998
FTSE	0.9984

Application: close-to-close returns

Problem

Until now, we computed return as open-to-close. However, usually returns are computed on a close-to-close basis. To do that, we need to use data from different rows of the database (different days). The goal is to redefine the 'Return' column as close-to-close returns.

Hints:

1. To sort a DataFrame by a column, one can use the `sort` method:

```
DF_Sort=DF.sort(column, ascending=True)
```

2. The one-row lag of a variable can be found with the `shift` method:

```
DF['Variable_Lag1']=DF['Variable'].shift(1)
```

Application: (one) solution

```
index_list=['CAC40', 'DAX', 'AEX']
for index in index_list:
    T=Indices[Indices['Index']==index]
    T['Return']
      =np.log(T['Close']/T['Close'].shift(1))
    if index==index_list[0]:
        IxNew=T
    else:
        IxNew=IxNew.append(T, ignore_index=True)
```

Outline

Dates and times

The relevant module for working with date-time objects is datetime. Seven fields in a datetime object : year, month, day, hour, minute, second, microseconds.

```
1 import datetime as dt
```

Some simple methods:

1. Current time: dt.datetime.now() or dt.datetime.today()
2. Weekday (Monday=0): dt.datetime.today().weekday()
3. Select fields:

```
1 d=dt.datetime.now()
2 .year .month .day .hour ... d.microsecond
```

Dates and times (c'td)

More simple methods:

1. GMT time: `dt.datetime.utcnow()`.
2. `timedelta` objects: subtracting two time objects creates a new type.
   How long did it take me to present this slide?
3. Convert a string to date or time

```
1  dt.datetime.strptime('2015-10-1', '%Y-%m-%d')
```

Dates in pandas

Data objects in pandas have the `Timestamp` type. Conversion:

1. From `datetime` to Timestamp: `Timestamp(d)`
2. From `Timestamp` to datetime: `ts.to_datetime()`

Data in numpy

Data objects in numpy have the datetime64 type.

Conversion:

1. From datetime to datetime64: `d64=np.datetime64(d)`
2. From datetime64 to datetime: `d=d64.astype(dt.datetime)`

Plotting support with datetime64 is limited in matplotlib - it is better to convert the timestamps before!

Outline

Today's application

We will work with high-frequency trading data, downloaded from Thomson Reuters Tick History. Concretely, all the trade and quotes from July 2009 on NASDAQ OMX Stockholm for three stocks:

1. H&M (`NORDIC_TS_2009-07-HMb.ST`)
2. Nokia (`NORDIC_TS_2009-07-NOKI.ST`)
3. Volvo (`NORDIC_TS_2009-07-VOLVb.ST`)

Download the zip file from `https://www.dropbox.com/s/zfn9wixwdhdrahu/data_L5.zip?dl=0` and unzip the content into your working directory (or go to `http://andras.niedermayer.ch` → Teaching → "Data for Lecture 5 – zip file")

Merging multiple files

Task #1

- Merge all three stock files into a single `DataFrame object`.
- How many observations are there in total?
- What variables does the dataset contain?

Task #1 - Solution

```
1  list_stocks =[ 'HMb.ST', 'NOKI.ST', 'VOLVb.ST']
2  fname ='NORDIC_TS_20090712-2009-07-'
3
4  Data = DataFrame()
5
6  for stock in list_stocks:
7  Temp =
8    DataFrame.from_csv("\%s%s.csv" % (fname,stock))
9    .reset_index()
10 Data = Data.append(Temp, ignore_index=True
11
12 Data.columns.tolist()
13 Data.count()
```

Variables

Variable	Notes
Date[G]	GMT date in format yyyymmdd (int).
Time[G]	GMT time in format hh : mm : ss.micro (string).
GMT Offset	Offset from GMT time.
Type	Type of message.
What are the types of messages? How many of each type?	
Price	Transaction price.
Volume	Transaction volume.
Buyer (Seller) ID	Trader identificator
Bid (Ask) Price	
Bid (Ask) Size	Size of bid (ask) order

Managing timestamps

Task #2

- Create a column "DT" with the date and time of the message in the `datetime` format.
  Hint: use a user defined function.
- Create a column "TS" with the date and time of the message in the `timestamp` format.

Task #2 - Solution

```
1  import datetime as dt
2
3  def mergedate(x):
4      return dt.datetime.strptime(x,
5        "%Y%m%d%H:%M:%S.%f")
6  Data ['DateS']= Data['Date[G]'].map(str)
7      + Data ['Time[G]']
8  Data ['TS']= Data['DateS'].map(mergedate)
9  Data ['DT']= Data['TS'].map(to_datetime)
```

Outline

Two-dimensional plotting

First, we import the pyplot library:

```
1 import matplotlib.pyplot as plt
```

1. Plot function: `plt.plot(y,x, options)`
2. Add a grid: `plt.grid()`
3. Custom axis limits: `plt.xlim(xmin, xmax)`
4. "Tight" axes: `plt.axis('tight')`
5. Axes labels: `plt.xlabel()`, `plt.ylabel()`
6. Add a title: `plt.title()`
7. Show the plot: `plt.show()`
8. Save the plot: `plt.save(path)`

Options: customizing your plot

- Color: `c=` or `color=`
- Line width: `lw=` or `linewidth=`
- Line style: `ls=` or `linestyle=`

Colors	Styles
b blue	- solid
g green	– dashed
r red	-. dash-dot
k black	: dotted
c cyan	. point marker
m magenta	o <> + other markers

A first 2D plot

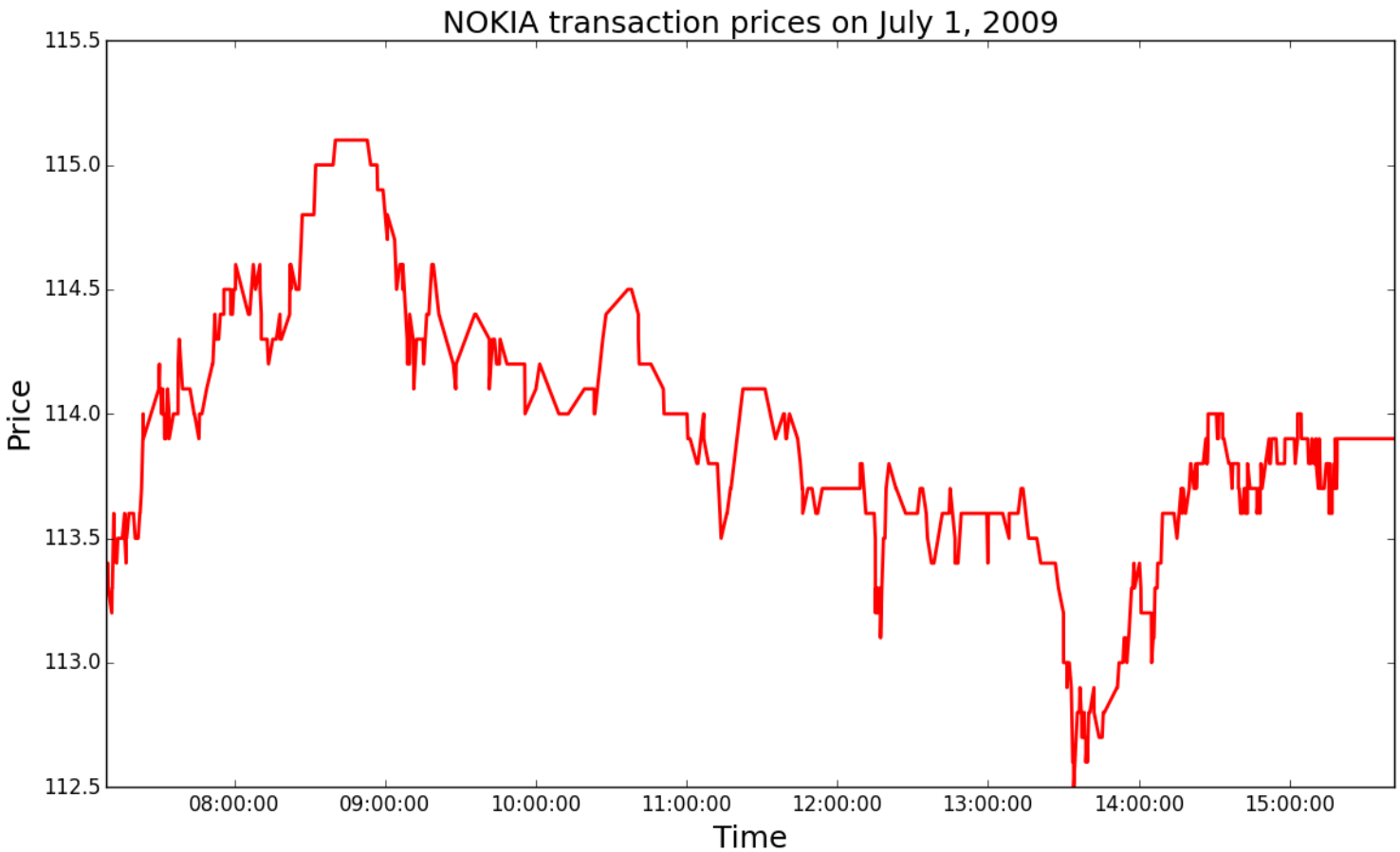
Task #3

Plot the NOKIA trade prices time series for July 1, 2009.
Label the axes and give the figure an appropriate title.

Task #3 - Solution

```
1  Noki =
2       Data[(Data["#RIC"]=="NOKI␣.ST") &
3       (Data["Date[G]"]==20090701)
4       & (Data ["Type"]=="Trade")]
5  plt.plot(Noki['DT'],Noki['Price'])
6  plt.xlabel('Time', fontsize=18)
7  plt.ylabel('Price', fontsize=18)
8  plt.title('NOKIA␣transaction␣prices
9  ␣␣␣on␣July␣1,␣2009', fontsize=18)
10 plt.show()
```

Task #3 - Solution



Figures in `pyplot`

A figure contains one or more plotted series. We define the figure size as follows:

```
1  plt.figure(figsize=(7,4) # add figure size
```

Legend

```
1  plt.legend (loc='best', fontsize=18)
```

Legend location(`loc`) can take multiple values:
'best', 'upper right', 'center left', 'lower right', 'center'...
Plotted series labels given by option label=.

Figures in `pyplot`

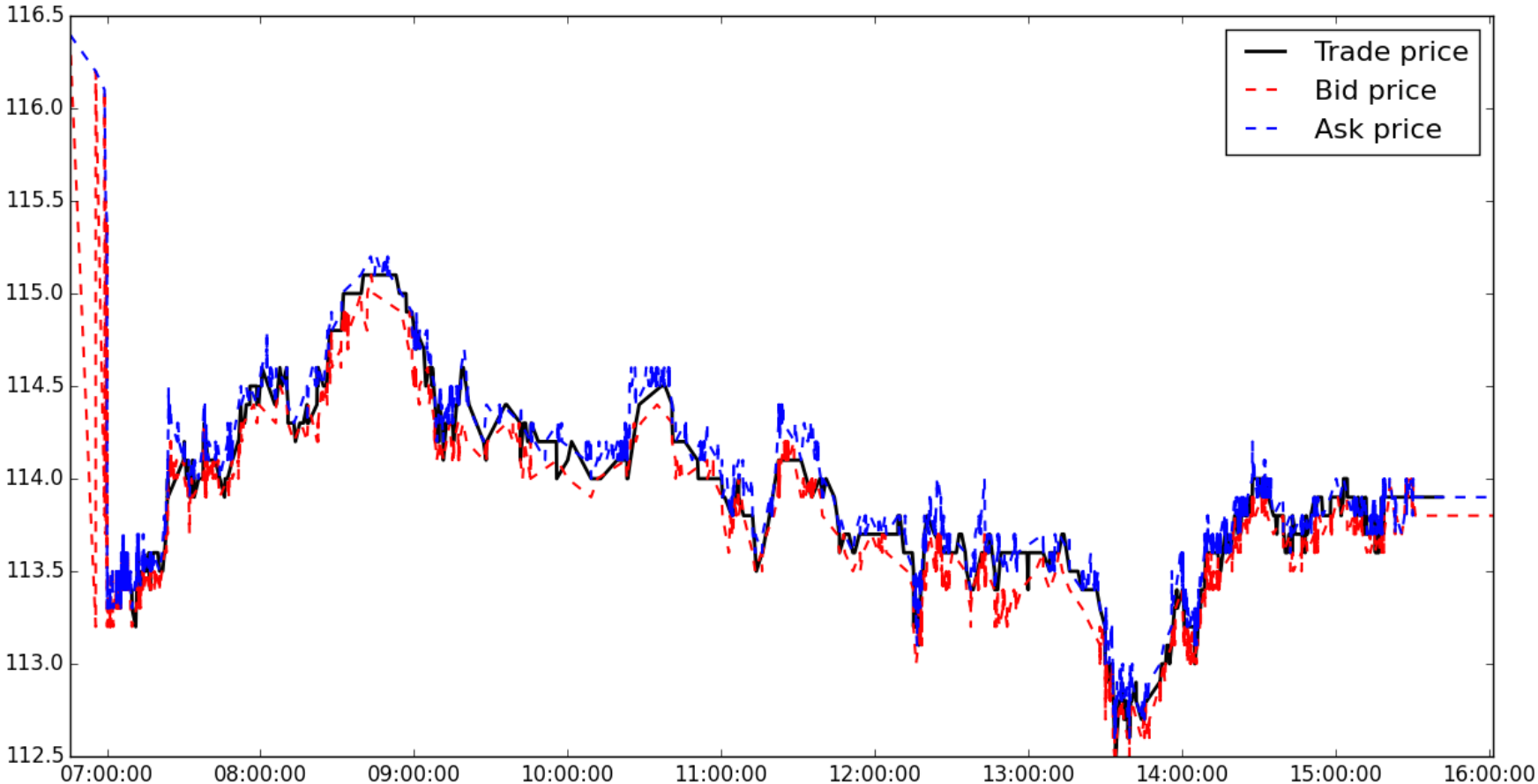
Task #4

Plot the NOKIA trade prices, bid and ask quotes time series for July 1, 2009. *Hint: clean quote data of zeros and* `NaN`.
Create an appropriate legend in the lower right corner.
Label the axes and give the figure an appropriate title.

Task #4 - Solution

```
plt.figure(figsize =(7 ,4))
NQ=Data[(Data["#RIC"]=="NOKI␣.ST")
     & (Data["Date[G]"]==20090701)
     & (Data["Type"]=="Quote")]
plt.plot(Noki["DT"],Noki["Price"],
     label='Trade␣price',c='k',lw=2)
plt.plot(NQ[NQ["Bid␣Price␣"]>0]["DT"],
     NQ[NQ["Bid␣␣Price"]>0]["Bid␣Price␣"].dropna(),
     label='Bid␣price',c='r',lw =1.5,ls='--')
plt.plot(NQ[NQ["Ask␣Price"] >0]["DT"],
     NQ[NQ["Ask␣Price"]>0]["Ask␣Price"].dropna(),
     label='Ask␣price',c='b',lw =1.5,ls='--')
plt.legend(loc='best', fontsize=16)
plt.show()
```

Task #4 - Solution

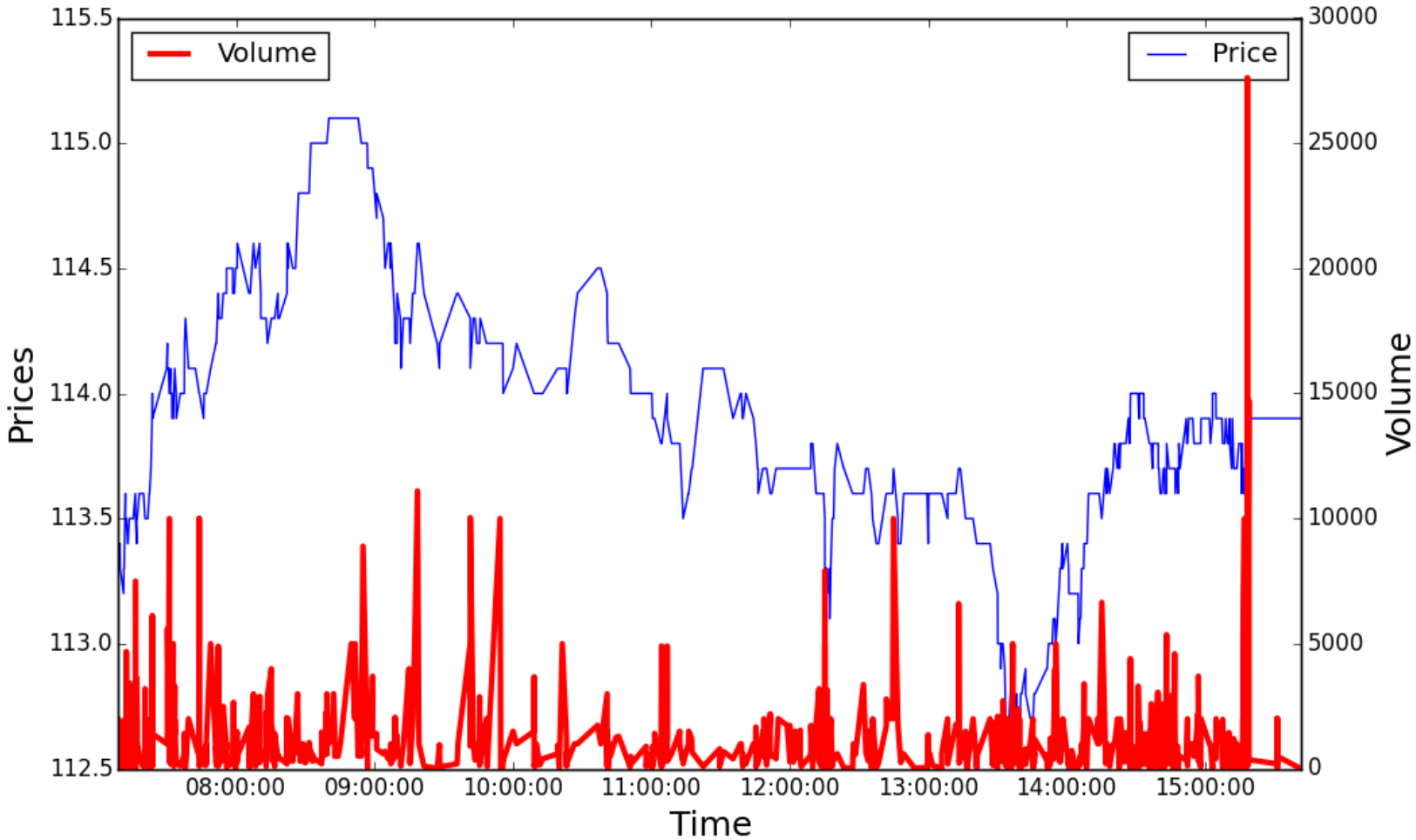


Using two y-axes

Let us plot both volumes and transaction prices for Nokia.
Of course, they have different scales.
We need to duplicate the y-axis, while keeping the x-axis the same (twinx).

```
1  fig , ax1=plt.subplots ()
2  plt.plot(Noki["DT"], Noki["Price"],
3       label="Price")
4  plt.xlabel('Time', fontsize =18)
5  plt.ylabel('Prices', fontsize =18)
6  plt.legend(loc='best')
7  ax2=ax1.twinx()
8  plt.plot(Noki["DT"], Noki["Volume"],
9       label="Volume",c='r',lw=3)
10 plt.ylabel('Volume', fontsize=18)
11 plt.legend(loc='best')
```

Using two `y-axes`

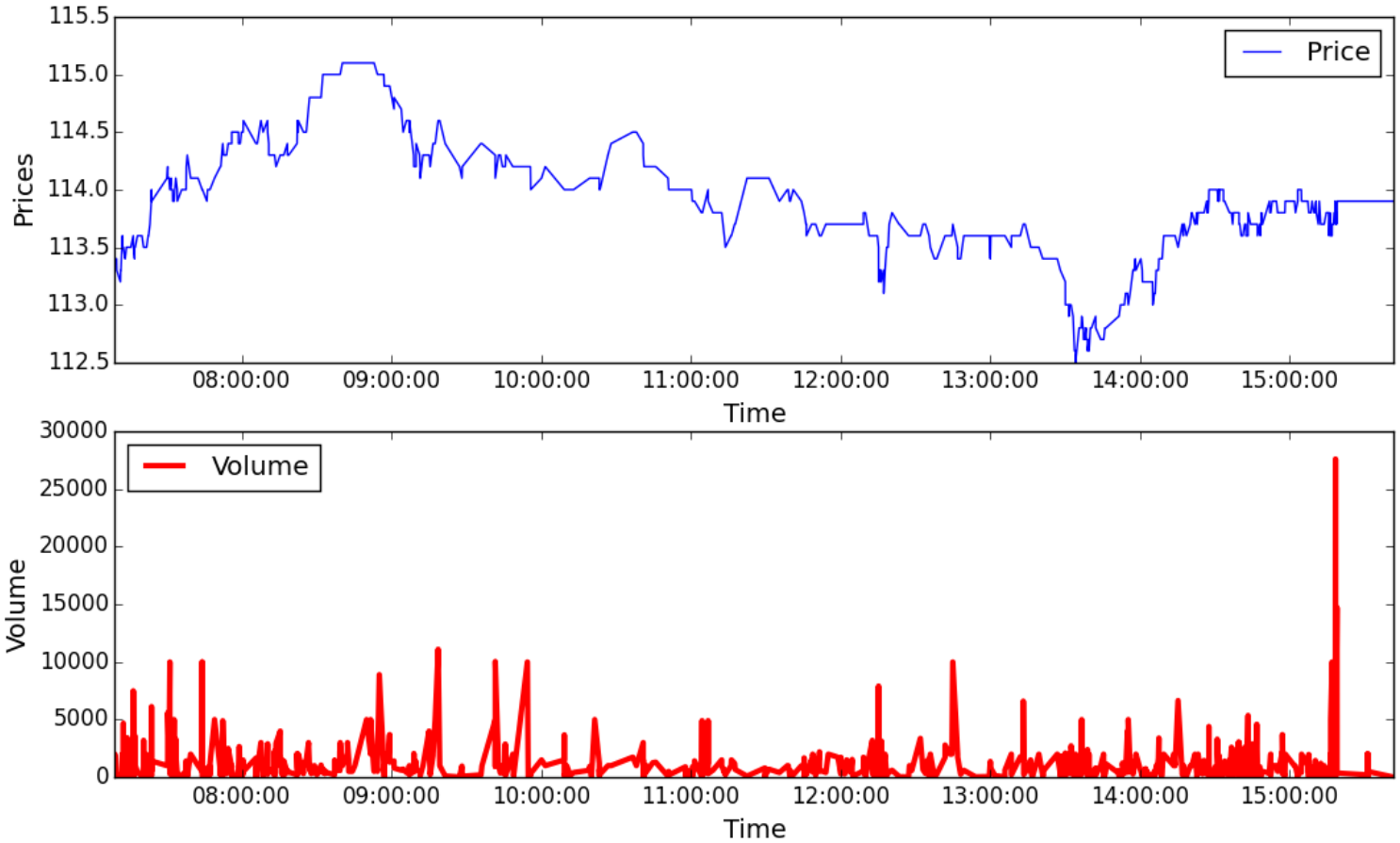


Using two y-axes

Let us plot both volumes and transaction prices for Nokia, now in two different subplots.
We use `plt.subplot(NCR)` where $N = C \times R$. The `subplot` method generates a collection of N plots of C columns and R rows.

```
1  plt.subplot(211)
2  plt.plot(Noki["DT"],Noki ["Price"],
3       label="Price")
4  plt.xlabel('Time', fontsize=18)
5  plt.ylabel('Prices', fontsize=18)
6  plt.legend(loc='best')
7  plt.subplot(212)
8  plt.plot(Noki["DT"], Noki["Volume"],
9       label="Volume",c='r',lw=3)
10 plt.xlabel('Time', fontsize=18)
11 plt.ylabel('Volume', fontsize=18)
12 plt.legend(loc='best')
```

Two subplots

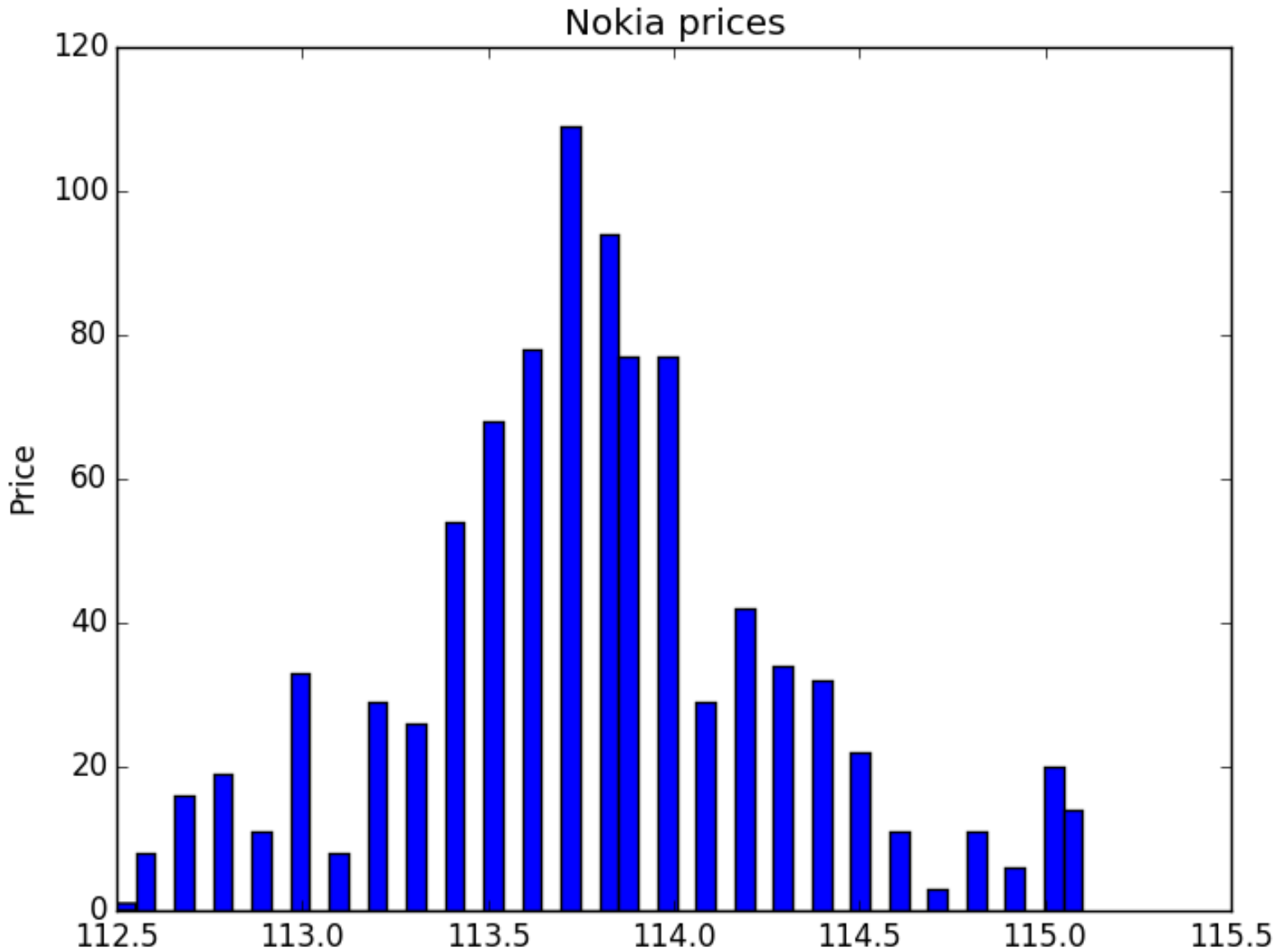


Histogram in `pyplot`

- Histograms in `pyplot` are generated by the function `plt.hist`.
- `Pandas` series' index need to be reset in order to work.
- The fineness of the grid can be fine-tuned through the option `bins=`.

```
1  plt.hist(
2      Noki["Price"].reset_index()["Price"],
3      label="Prices", color ="b", bins=50)
```

Histogram in `pyplot`

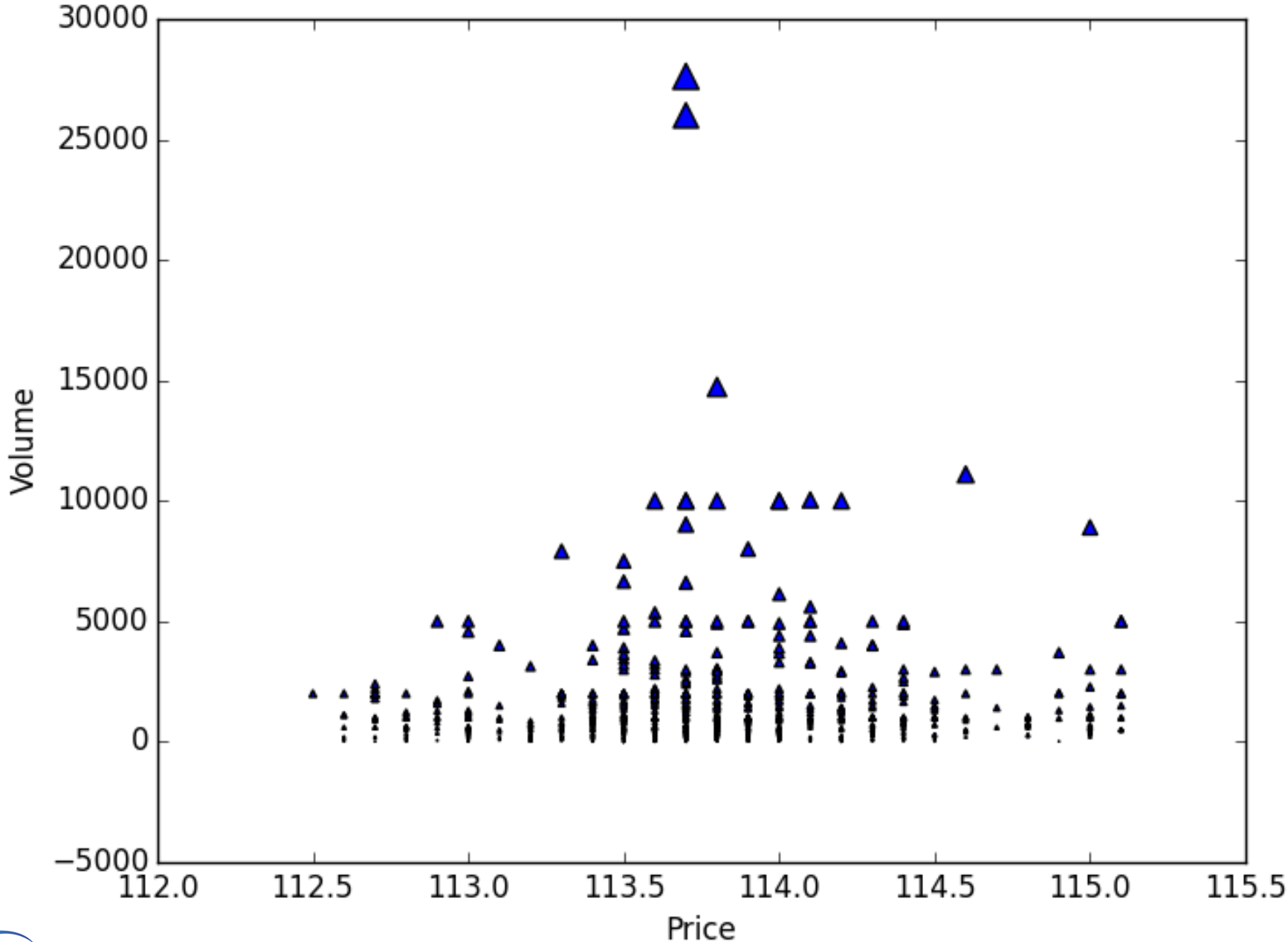


Scatterplot in `pyplot`

- Scatterplots in `pyplot` are generated by the function `plt.scatter(x,y)`.
- `Pandas` series' index do not need to be reset in order to work.
- Markers can be changed through the option `marker=`.
- The size of the marker can be made proportional to another variable, via option `s=`

```
1  plt.scatter(Noki['Price'],
2       Noki['Volume'], marker="^",
3       s=Noki["Volume"]/250)
```

Scatterplot in `pyplot`



Financial plots via `matplotlib.finance`

We first import the financial plots library:

```
1 import matplotlib.finance as mpf
```

Next, we load CAC40 data from Yahoo Finance between 1/09 - 11/09:

```
1 start=(2015,8,11)
2 end=(2015,9,11)
3 quotes=mpf.quotes_historical_yahoo_ohlc("^FCHI ",
4     start,end)
```

Data is in the format:
(Date, Open, High, Low, Close, Volume)

candlestick plots

First, we initialize the figure and axis objects:

```
1 fig, ax=plt.subplots()
```

We generate the plot:

```
1 mpf.candlestick_ohlc(ax, quotes, width=0.5,
2     colorup='g', colordown='r')
```

We format the x-axis as dates (currently integers):

```
1 ax.xaxis_date()
```

Next, we rotate the x-axis to fit better and then we save the plot:

```
1 plt.setp(plt.gca().get_xticklabels(), rotation=30)
2 plt.savefig('/../candlestick_%s.pdf'%"CAC40",
3     bbox_inches='tight')
```