

New Ways to Access Financial Databases

Data Center Tutorial

Errikos Melissinos

November 21, 2019

Outline

In this tutorial I will:

- almost exclusively use Python,
- talk about the WRDS API,
- talk about the Eikon API,
- talk about the Bloomberg API,
- use Pandas but not go into the detail of it,
- do a little bit of plotting,
- provide as many resources as practically possible
- and show a bonus library if I have time!

WRDS API

- WRDS is a platform through which many databases are provided.
- In our case there is CRSP, Amadeus, OptionMetrics, BoardEx (soon) etc.
- So, I will be showing the basics of navigating these databases and downloading data.

WRDS - Setup

Set let's see how to start ([WRDS resources](#)):

- Download the wrds library for Python (from the command prompt):

```
pip install wrds
```

- Establish a connection through Python:

```
import wrds  
db = wrds.Connection()
```

- After this you will be asked for your user name and password. In order to not have to go through this procedure the following times you run a program you should create a file with your credentials. This is quite easy:

```
db.create_pgpass_file()
```

- Let's check if it works:

```
db.close()  
db = wrds.Connection(wrds_username=[your user name])
```

WRDS - Basic Functions

Let's look at the basic functions of the module:

- The main functions are: `db.close()`, `db.list_libraries()`, `db.list_tables([some library])`, `db.describe_table([some library], [some table])`, `db.get_table([library], [table], columns=[], obs=...)`, `db.raw_sql()`.
- In order to figure out exactly what we want, we should use the info in the [WRDS website](#).

Excursion to Pandas

Now, that we already got some data we can look at the form in which they come, Pandas data frames:

- You can find resources [here](#), and [this](#) is a cool cheat sheet.
- There is a lot that you can do with these pandas data frames and the best way to learn is to play around with it.
- Let's look at displaying some data, taking subsets and summarising the data using Pandas.

WRDS - .raw_sql()

- This `get_table` function may still not be that useful, as it does not allow more specific queries.
- For more specific searches we should use the `db.raw_sql()` function ([resources for SQL queries](#)).

```
data = db.raw_sql("""select idnr, closdate, fias, ifas,  
tfas, ofas from bvd.financials_v where country=  
'GERMANY' and fias>=1000000 LIMIT 1000""")
```

WRDS - .raw_sql() (cont'd)

- We can also provide arguments to this function and of course take advantage of the possibilities that SQL offers. For example we may only want data for specific companies and we may want to combine datasets. In this case we provide the tickers in a dictionary as a tuple and then we pass them in the function as follows:

```
parm = {'id_nrs': ('DE8170003036', 'DE6190316434')}  
data = db.raw_sql("""select a.closdate, b.name_nat,  
a.idnr, b.sd_isin, b.sd_ticker, a.fias, a.ifas, a.tfas,  
a.ofas from bvd.financials_v as a left join  
bvd.amadeus_v as b on a.idnr=b.idnr where  
a.idnr in %(id_nrs)s""", params= parm)
```


WRDS - Fama-French Factors

Now I will illustrate what we can do with a more advanced example:

- We will look at ff.py (should be available in all computers). This is a file that is based on [an example offered by WRDS](#).
- We can take this opportunity and see some plotting with Python ([Plotly: Getting Started](#)):

```
import fama as f
import plotly.offline as pyo
pyo.plot([{'x': f._ffcomp70.date, 'y':f._ffcomp70['smb']},
{'x': f._ffcomp70.date, 'y':f._ffcomp70['WSMB']}]])
pyo.plot([{'x': f._ffcomp70.date, 'y':f._ffcomp70['hml']},
{'x': f._ffcomp70.date, 'y':f._ffcomp70['WHML']}]])
```

WRDS - On R

As long as you have created the .pgpass file (as we did using Python) downloading data through R is easy ([WRDS resources](#)):

- Make sure you are running the latest version of R and then install library RPostgres
- Make a connection and download sample data with the following commands:

```
library(RPostgres); wrds <- dbConnect(Postgres(),  
                                       host='wrds-pgdata.wharton.upenn.edu',  
                                       port=9737, sslmode='require',  
                                       dbname='wrds', user='errikos')  
res <- dbSendQuery(wrds, "SELECT date,dji FROM  
                        djones.djdaily LIMIT 1000")  
  
data <- dbFetch(res)  
dbClearResult(res)  
data
```

WRDS - On Stata...

- Unfortunately we do not have Stata on these computers. So, I will not be showing you how to make a connection with Stata.
- [Here](#) you can find some resources if you are interested.
- Keep in mind that you will have to use the SQL syntax to access the data as was the case with R.
- If you are really interested let us know and we could have a tutorial on WRDS in a different room where computers have Stata.

Eikon - Overview

Regarding Eikon I will talk about:

- The Setup of the API
- Symbolology
- Downloading data, including news
- Some plotting

Eikon - Setup

So, let's see how to start using the Eikon API in Python:

- [Here](#) are some very useful resources, which include video tutorials out of which I have taken a lot of material for today while the documentation under the corresponding tab is also helpful.
- I assume that the Eikon Desktop Application is already installed and running (if not you can download it [here](#))
- Use this environment to perform a general search for the "App Library" and in the App Library you can search for "App Key Generator".
- Generate your key and save it in a file named "eikon.cfg" as follows (do not use quotes around the key):

```
[eikon]  
app_key = YOUR_APP_KEY_HERE
```
- This file will need to be available to the code that we will be running later (e.g. you can include it in the same folder as the Python file we are running).

Eikon - Setup (cont'd)

Let's go on with the setup:

- Install the eikon library (from the terminal):

```
pip install eikon
```

- Import the libraries that we will use:

```
import eikon as ek  
import configparser as cp
```

- Make the connection (the Desktop Application should be running):

```
cfg = cp.ConfigParser()  
cfg.read('eikon.cfg')  
ek.set_app_key['eikon']['app_id']
```

- And now we can download data!

Eikon - Downloading the first Data

Let's see a first example of downloading data:

- In the case historical end-of-day data for Apple:

```
data = ek.get_timeseries('AAPL.O', fields='*',  
start_date='2019-01-01', end_date='2019-11-20')
```

- This gives us Pandas data frame.
- We can use plotly as before to plot the data...
- ...or we can use cufflinks ([resources here](#)):

```
import plotly as py; import cufflinks as cf  
cf.set_config_file(offline=True)  
fig = data['close'].iplot(asFigure=True)  
py.offline.plot(fig)
```

Eikon - Downloading News Headlines

Let's see another example of downloading news headlines:

- Downloading news about Apple

```
headlines = ek.get_news_headlines('R:AAPL.O',  
                                   date_from='2019-11-19T20:00:00',  
                                   date_to='2019-11-19T22:00:00',  
                                   count=20)
```

- This is a good place to mention that we can use the help function with the Eikon methods:

```
help(ek.get_news_headlines)
```


Eikon - Symbology

Let's look at some basics:

- A handy way to look up individual identifiers is through the Data Item Browser of the Desktop Application.

- Converting to different kinds of identifiers:

```
ek.get_symbology(['AAPL.O', 'DE10YT=RR'],  
from_symbol_type='RIC',  
to_symbol_type=['ISIN', 'ticker'])
```

- Keep in mind that here multiple RICs are inputted and multiple codes are returned, but a single string could also be provided in both cases.
- If we have some other identifier we just change the `from_symbol_type` and the `to_symbol_type` correspondingly.
- So, this is a way to download data in case we are given an alternative identifier.

Eikon - get_data() and Chain RICs

A generic way to download data is with the `get_data()` function:

- In order to figure out the data fields that we are interested in, it is again handy to use the data item browser.
- In this example we want to look at the companies comprising the DAX, we could provide the individual RICs for each of them but we can also use the *chain RIC*.
- add “0#” in front of the relevant RIC of the DAX (and the same recipe works with other indices)
- The `get_data()` function returns first the data and then any error that arose while downloading:

```
data, err = ek.get_data('0#.GDAXI', fields=[  
    'TR.CommonName', 'TR.TotalReturnYTD'])
```

- Now we can have Python plot these results for us.

Bloomberg

With regard to Bloomberg I will talk about:

- How to set up the Bloomberg API in a Python Environment.
- What are the difficulties associated with the Bloomberg API.
- A library that can be used to make our life easier.
- The mechanics of the examples provided by Bloomberg and how we can take advantage of them.

Bloomberg API - Setup

Overview of the setup:

- You can download the [API](#)
- You would need to download the file under API Windows: C/C++ Experimental Release.
- Find the two .dll files that are included in these files and include them in the same folder as the python script you want to run.
- Download the corresponding Python library, blpapi, by writing the following in the terminal:

```
> python -m pip install --index-url=https://bloomberg.bintray.com/pip/simple blpapi
```

- Use Python to download Bloomberg Data!

Bloomberg API - Examples

We quickly find that there are some difficulties:

- Almost no documentation!
- We only have some working examples.
- These examples print some output when they are run.
- However, they do not provide the data as a Python object and it is not clear how to change them so that this becomes possible.

Bloomberg API - pdblp

Thus, there are additional libraries built on top (not by Bloomberg):

- Out of the ones I tested, pdblp was the best choice.
- It has good [documentation](#), including examples.
- It provides functions that directly return Python objects and specifically data frames.
- So, let's look at how to download historical data.

Bloomberg API - pdblp: bdh

This has a functionality similar to the corresponding bdh function in the MS Excel add-on:

- Import the library and make a connection:

```
import pdblp  
con = pdblp.BCon(timeout=5000)  
con.start()
```

- Use the function to get the data:

```
data = con.bdh('SPY US Equity', 'PX_LAST',  
'20150629', '20150630')
```

Bloomberg API - pdblp: ref

This has a functionality similar to the bdp function in the MS Excel add-on:

- This gives single points of data for each variable.

```
data = con.ref('EURUSD Curncy', 'SETTLE_DT')
```

- But we can ask for multiple securities and multiple variables:

```
data = con.ref(['EURUSD Curncy', 'EURGBP Curncy'],  
               ['SETTLE_DT', 'DAYS_TO_MTY'])
```

- Some variables allow for specific options which are called “overrides”:

```
data = con.ref(['EURUSD Curncy', 'EURGBP Curncy'],  
               'SETTLE_DT', [('REFERENCE_DATE', '20150715')])
```


Bloomberg API - pdblp: ref_hist

- We can also use these overrides with a special function to create time series(`.ref_hist()`):

```
data = con.ref_hist('EUR1M Curncy',  
                    'DAYS_TO_MTY', dates=['20150625', '20150626']))
```

- Beware this can burn the Bloomberg data limit very fast.

The Bloomberg Examples

pdblp possibly deals with all the examples that you are interested in, but there are also some Bloomberg examples which show how to use the library directly.

- You can find the examples [here](#).
- A couple of examples should also appear in the Bloomberg folders in the computers.
- Let's run one of them to see what happens. We see that it runs but it only prints and it gives results for specific securities.
- Let's look at the structure of the example a bit.
- Let's try to get the result for any security that we are interested in.

Bloomberg - Modified Examples

So, I have adapted a couple of examples to make them a bit more useful:

- I have built on the SimpleHistoryExample to create a more usable file, which can be imported as follows:

```
import SHistoryExampleAdapted as hist
data = hist.bdh(security=['AAPL US Equity',
                          'MSFT US Equity'], data_field='PX_LAST',
start_date = '20190629', end_date =
'20190630', frequency='DAILY', printing=False)
```

- Then I have adapted the SimpleIntradayBarExample to be a bit more useful:

```
import SIntradayBarExampleAdapted as bar
data = bar.bdib(security='AAPL US Equity',
event_type='TRADE', interval=60, printing=False)
```

- This is a way of adapting the official Bloomberg examples to get data frames while also programmatically choosing the parameters of the query.

Joblib - a useful Python library

Using Python to download could become a bit frustrating and potentially even dangerous:

- You could be downloading big amounts of data multiple times.
- This could waste too much of your time.
- If you make a mistake in your code and let some for-loop or some while-loop running you could end up downloading some data too many times!
- For Bloomberg this could actually be dangerous as the data limit could be reached just by a mistake in the code.
- * You can find information and examples [here](#)

Joblib - a useful Python library (cont'd)

Let's see how joblib can help us:

- To download and install just use:

```
pip install joblib
```

- Now let's look at an example to see how joblib operates (joblib_example.py)
- Just specify a folder name ".\folder-name", then initialise the memory and finally wrap the function you are interested in.

```
import joblib
temp_dir = "./cache"
mem = joblib.Memory(temp_dir)
wrapped_f = mem.cache(class_name.f, ignore=['self'])
```

- * The ignore part is necessary if the function you are wrapping is a property of a class object.

Reminder

- Go to the user, then follow the path “AppData/Roaming/postgresql” and delete the file with your credentials.

Conclusion

- I hope that this was a good introduction for those of you who are beginners...
- ...while also not too slow for those among you who are more advanced.
- We will be happy to receive your feedback.

Thank You!

Links

- Pandas is a Python library to manage tables of data: [Pandas resources](#), [Pandas Cheat Sheet](#)
- SQL is a programming language used to manage databases: [resources for SQL](#)
- Plotly is a Python library to produce graphs: [Getting Started](#)
- Cufflinks is also a Python library to produce graphs, uses plotly and is supposed to have better integration with Pandas data frames: [Cufflinks resources](#)
- [WRDS datasets](#)
- [Fama-French Factors example based on example offered by WRDS](#)
- [WRDS API for Python](#), [WRDS API for R](#), [WRDS API for STATA](#)
- [To download the Eikon Desktop Application](#)
- [Resources on the Eikon API](#)
- [To download the Bloomberg API](#), [Official Bloomberg examples](#)
- [Resources for the third party library, pdblp, that uses the Bloomberg API](#)