Fundamental Analysis of Securities Trading  
(II) Data B 

Kuan-Lun Wang

National Taiwan University

Preliminary Draft: September 12, 2019

1E-mail: polyphonicared@gmail.com
2Home Page: https://www.csie.ntu.edu.tw/~d06922002/

Short Biodata

Kuan-Lun Wang is a doctoral student majoring in generalized pairs trading. The main goal of his research is to develop an algorithmic trading mechanism based on statistical arbitrage. His areas of expertise include automatic search procedures for model selection, multivariate co-integration approach, and structural change test.
Kuan-Lun Wang’s research interests comprise time series models, simulation modeling, and portfolio choice. The central themes of his application are the study of multivariate pairs trading in real time, search for assets with a long-run equilibrium, and building of riskless portfolios. Much of his current work involves conducting structural change analysis and co-integration test of the finite order vector autoregressive process and estimating the probability of mean reversion. Such methods are important in a variety of applications, including economic indicators and hedging. One such application is index funds being tied to indexes with very low costs and risks.

Clearly, we need to read a file and return user requirement data:

- getFilename
- fileRead
- infoSearch
str: filename=getFilename(int: yyyymmdd)
list: rows=fileRead(str: filename)
infoSearch
Since user maybe wants to search a single ticker or multiple tickers (single info or multiple info), the input of function is defined as

```
str: tickers[1+] (str: infoNames[1+]).
```

- str: filename=getFilename(int: yyyymmd)
- list: rows=fileRead(str: filename)
- str: vlist[1+][0+]=infoSearch(str: tickers[1+], str: infoNames[1+], list: rows)

**Example:**

- `infoSearch('0050', 'Opening Price')` returns `'77.7'`
- `infoSearch('0050', ['Opening Price', 'Closing Price'])` returns `['77.7', '77.65']`
- `infoSearch(['0050', '0051'], 'Opening Price')` returns `[['77.70'], ['30.74']]`
- `infoSearch(['0050', '0051'], ['Opening Price', 'Closing Price'])` returns `[['77.70', '77.65'], ['30.74', '30.75']]

**Note:**

- The function `infoSearch` takes a list of tickers and a list of info names as input.
- It returns a list of info values for each ticker.
- For single ticker, it returns a single info value.
- For multiple tickers, it returns a list of info values for each ticker.
Since user maybe want to search a single day or multiple days, the download API is defined as:

```plaintext
str: info[1+][0+]=read(str: tickerIDs[1+], int: infoNames[1+], int: yyyymmdd[1+])
```

```plaintext
str: filename=getFilename(int: yyyymmdd)
use
list: rows=fileRead(str: filename)
use
str: vlist[1+][0+]=infoSearch(str: tickers[1+], str: infoNames[1+], list: rows)
return
```
### Listing 1: getFilename

```python
def getFilename(yyyymmdd):
    filename='{filename}.{extension}'.format(filename=yyyymmdd,extension='csv')
    return filename
```

### Listing 2: Help of csv.reader

```python
Help on built-in function reader in module _csv:

reader(...)  
    csv_reader = reader(iterable [, dialect='excel']  
                       [optional keyword args])  
    for row in csv_reader:  
        process(row)  

The "iterable" argument can be any object that returns a line of input for each iteration, such as a file object or a list. The optional "dialect" parameter is discussed below. The function also accepts optional keyword arguments which override -- More --
```
Listing 3: fileRead

```python
from csv import reader as _csvRead

def fileRead(filename):
    with open(filename) as csvFile:
        table = [rowText for rowText in _csvRead(csvFile, delimiter=',',)]
    return table
```

- Search row includ tickerID
  - tickerIDsRowSearch
- Search value in corresponding row
  - rowInfosValueSearch
Since tickerID must in first column, we collect all elements in the column.

Listing 4: fileRead

```python
def _getFirstColumn(rows):
    if len(rows) == 0:
        return None
    column = []
    for row in rows:
        if len(row) == 0:
            c0 = None
        elif len(row) > 0:
            c0 = row[0]
        column.append(c0)
    return column
```

If the first word in tickerID is '0', then the tickerID must be convert to '="tickerID"'.

Listing 5: tickerIDsRowSearch

```python
def tickerIDsRowSearch(tickerIDs, table):
    if type(tickerIDs) == list:
        ...
    else:
        column = _getFirstColumn(table)
        if column == None:
            return None
        if tickerIDs[0] == '0':
            tickerIDs = '="{}"'.format(s=tickerIDs)
        index = column.index(tickerIDs)
        rows = table[index]
    return rows
```
Listing 6: tickerIDsRowSearch

```python
def tickerIDsRowSearch(tickerIDs, table):
    if type(tickerIDs) == list:
        rows = []
        for tickerID in tickerIDs:
            row0 = tickerIDsRowSearch(tickerID, table)
            rows.append(row0)
    else:
        ...
    return rows
```

Listing 7: rowInfosValueSearch

```python
def rowInfosValueSearch(infoNames, row):
    if type(infoNames) == list:
        ...
    else:
        infoNames = infoNames.lower()
        indexInfoName = _referenceList().index(infoNames)
        if row != None:
            vlist = row[indexInfoName]
        else:
            vlist = None
    return vlist
```
Listing 8: rowInfosValueSearch/referenceList

```python
def _referenceList():
    vlist=['Security Code', 'Trade Volume', 'Transaction', 'Trade Value',
           'Opening Price', 'Highest Price', 'Lowest Price', 'Closing Price',
           'Dir(+/-)', 'Change', 'Last Best Bid Price', 'Last Best Bid Volume',
           'Last Best Ask Price', 'Last Best Ask Volume',
           'Price-Earning ratio']
    vlist=[v.lower() for v in vlist]
    return vlist
```

Listing 9: rowInfosValueSearch

```python
def rowInfosValueSearch(infoNames, row):
    if type(infoNames)==list:
        vlist=[]
        for n in range(len(infoNames)):
            vlist0=rowInfosValueSearch(infoNames[n], row)
            vlist.append(vlist0)
    else:
        ...
    return vlist
```
Listing 10: infoSearch

```python
def infoSearch(tickerIDs, infoNames, table):
    rows = _tickerIDsRowSearch(tickerIDs, table)
    if type(tickerIDs) == list:
        vlist = []
        for n in range(len(tickerIDs)):
            vlist0 = _rowInfosValueSearch(infoNames, rows[n])
            if type(vlist0) == list:
                vlist0 = [vlist0]
            vlist.append(vlist0)
    else:
        vlist = _rowInfosValueSearch(infoNames, rows)
    return vlist
```

Listing 11: read

```python
def read(tickerIDs, infoNames, yyyymmdd):
    filename = _getFilename(yyyymmdd)
    table = _fileRead(filename)
    info = _infoSearch(tickerIDs, infoNames, table)
    return info
```
We rename read as _singleRead.
We use a while-loop to read from yyyymmddStart to yyyymmddEnd.
In this while-loop, _singleRead is used multiple times.
However, we maybe have some miss since this function has multiple inputs.

**Listing 12: functools.partial**

```
Help on class partial in module functools:

class partial(builtins.object)
    partial(func, *args, **keywords) - new function
    with partial application of the given arguments
    and keywords.

    Methods defined here:

    __call__(self, /, *args, **kwargs)
        Call self as a function.

    __delattr__(self, name, /)
        Implement delattr(self, name).

    __getattr__(self, name, /)
        Implement getattr.

    __getattribute__(self, name, /)
        Implement getattr.

    __get__(*args, **kwargs)
        Implement get.

    __new__(cls, func, *args, **keywords)
        New-style class constructor.

    __reduce__(self, /)
        Implement _reduced.

    __repr__(self, /)
        Implement repr.

    __setattr__(self, name, value, /)
        Implement setattr.

    __setstate__(self, state, /)
        Implement _state.

    __setitem__(self, key, value, /)
        Implement __setitem__.

    __delitem__(self, key, /)
        Implement __delitem__.
```
Read Multiple Daily Quotes (3/5)

Listing 13: read

```python
def read(yyyymmdd, tickerIDs, infoNames):
    filename=_getFilename(yyyymmdd)
    table=_fileRead(filename)
    info=_infoSearch(tickerIDs, infoNames, table)
    return info
```

We rename read as _singleRead.

Listing 14: singleRead

```python
def singleRead(yyyymmdd, tickerIDs, infoNames):
    filename=_getFilename(yyyymmdd)
    table=_fileRead(filename)
    info=_infoSearch(tickerIDs, infoNames, table)
    return info
```

Read Multiple Daily Quotes (4/5)

Listing 15: read

```python
def read(tickerIDs, infoNames, yyyymmddInterval):
    singleRead = partial(_singleRead, \
                          tickerIDs=tickerIDs, \
                          infoNames=infoNames \n                        )
```
Listing 16: read

```python
def read(tickerIDs, infoNames, yyyymmddInterval):
    ...
    if type(yyyymmddInterval) != list:
        yyyymmdd = yyyymmddInterval
        vlist = singleRead(yyyymmdd)
    else:
        yyyymmdd = yyyymmddInterval[0]
        yyyymmddEnd = yyyymmddInterval[1]
        vlist = []
        while yyyymmdd <= yyyymmddEnd:
            vlist0 = singleRead(yyyymmdd)
            yyyymmdd = getNextDay(yyyymmdd)
            vlist.append(vlist0)
        return vlist
```

We want the download file save in a nice folder (not anywhere). Moreover, the user can select the folder path.
Listing 17: Help of os.path.join

```python
Help on function join in module ntpath:

join(path, *paths)
    # Join two (or more) paths.
```

Listing 18: Help of os.path.isdir

```python
Help on built-in function _isdir in module nt:

_isdir(path, /)
    Return true if the pathname refers to an existing directory.
```

Listing 19: Help of os.path.makedirs

```python
Help on function makedirs in module os:

makedirs(name, mode=511, exist_ok=False)
    makedirs(name [, mode=0o777][, exist_ok=False])

Super-mkdir; create a leaf directory and all intermediate ones. Works like mkdir, except that any intermediate path segment (not just the rightmost) will be created if it does not exist. If the target directory already exists, raise an OSError if exist_ok is False. Otherwise no exception is raised. This is recursive.
```
Listing 20: getFolderPath

```python
from os.path import join as _pathJoin
from os.path import isdir as _isdir
from os import makedirs as _mkdir

def getFolderPath(databasePath):
    folderpath=_pathJoin(databasePath,'TWSE')
    if not _isdir(folderpath):
        _mkdir(folderpath)
    folderpath=_pathJoin(folderpath,'Daily_Quotes')
    if not _isdir(folderpath):
        _mkdir(folderpath)
    return folderpath
```

Listing 21: download/_singleDownload

```python
def _singleDownload(yyyymmdd,databasePath,pauseTime,
                         tryAgainTime):
    yyyymmdd=str(yyyymmdd)
    filename=_getFilename(yyyymmdd)
    folderpath=_getFolderPath(databasePath)
    filename=_pathJoin(folderpath,filename)
    ...
```

Listing 22: download

```python
def download(yyyymmddInterval,databasePath,\
             pauseTime,tryAgainTime \)
    :
    ...
```
Listing 23: read/_singleRead

```python
1 def _singleRead(yyyymmdd, tickerIDs, infoNames):
2     filename = _getFilename(yyyymmdd)
3     folderpath = _getFolderPath(databasePath)
4     filename = _pathJoin(folderpath, filename)
5     table = _fileRead(filename)
6     info = _infoSearch(tickerIDs, infoNames, table)
7     return info
```

Listing 24: read

```python
1 def read(tickerIDs, infoNames, yyyymmddInterval, \ 
2         databasePath = \
3         ): 
4         ...
```

Listing 25: dailyQuotes

```python
class dailyQuotes():
    def __init__(self, databasePath=":."): 
        self.databasePath = databasePath 
    def download(self, 
        yyyymmddInterval, 
        pauseTime=5, tryAgainTime=60\ 
    ): 
        _download(yyyymmddInterval, 
        self.databasePath, 
        pauseTime, 
        tryAgainTime 
    )
    ...
```
Listing 26: dailyQuotes

```python
class dailyQuotes:
    def __init__(self, databasePath="."):
        self.databasePath = databasePath

    def read(self, tickerIDs, infoNames, yyyymmddInterval):
        vlist = _read(tickerIDs, infoNames, yyyymmddInterval, self.databasePath)
        return vlist
```

---

**Definition: Public, Private, and Protected**

- **Public**: Any class can refer to the field or call the method.
- **Protected**: Only the current class and sub-classes (and sometimes also same-package classes) of this class will have access to the field or method.
- **Private**: Only the current class will have access to the field or method.

Listing 27: dailyQuotes

```java
1. public class dailyQuotes
2.     |- public attribute databasePath
3.     |- public operation download
4.     |- public operation read

- dailyQuotes
- str: databasePath
- NoneType: download()
- str: read()

- dailyQuotes
- str: databasePath
- NoneType: download(int: yyyymmdd[1+])
- str[1+][0+]: read(str: tickerIDs[1+], int: infoNames[1+], int: yyyymmdd[1+])
```