Remote Data Access

Functions from `pandas_datareader.data` and `pandas_datareader.wb` extract data from various Internet sources into a pandas DataFrame. Currently the following sources are supported:

- Yahoo! Finance
- Google Finance
- Enigma
- St.Louis FED (FRED)
- Kenneth French's data library
- World Bank
- OECD
- Eurostat
- Thrift Savings Plan
- Oanda currency historical rate
- Nasdaq Trader symbol definitions<remote_data.nasdaq_symbols

It should be noted, that various sources support different kinds of data, so not all sources implement the same methods and the data elements returned might also differ.

Yahoo! Finance

Historical stock prices from Yahoo! Finance.

```
In [1]: import pandas_datareader.data as web

In [2]: import datetime

In [3]: start = datetime.datetime(2010, 1, 1)

In [4]: end = datetime.datetime(2013, 1, 27)

In [5]: f = web.DataReader("F", 'yahoo', start, end)

RemoteDataErrorTraceback (most recent call last)
<ipython-input-5-679b50ab962b> in <module>()
----> 1 f = web.DataReader("F", 'yahoo', start, end)

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/data.pyc in DataReader(name, data_source, start, end, retry_count,
pause, session, access_key)
    115                                 adjust_price=False, chunksize=25,
    116                                 retry_count=retry_count, pause=pause,
--> 117                                 session=session).read()
    118
    119     elif data_source == "yahoo-actions":

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/yahoo/daily.pyc in read(self)
     75     def read(self):
     76         """ read one data from specified URL """
---> 77         df = super(YahooDailyReader, self).read()
     78         if self.ret_index:
     79             df['Ret_Index'] = _calc_return_index(df['Adj Close'])

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in read(self)
    155         if isinstance(self.symbols, (compat.string_types, int)):
    156             df = self._read_one_data(self.url,
--> 157                                     params=self._get_params(self.symbols))
    158         # Or multiple symbols, (e.g., ['GOOG', 'AAPL', 'MSFT'])
    159         elif isinstance(self.symbols, DataFrame):

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _read_one_data(self, url, params)
     72         """ read one data from specified URL """
     73         if self._format == 'string':
---> 74             out = self._read_url_as_StringIO(url, params=params)
     75         elif self._format == 'json':
     76             out = self._get_response(url, params=params).json()

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _read_url_as_StringIO(self, url, params)
     83         Open url (and retry)
     84         """
---> 85         response = self._get_response(url, params=params)
     86         text = self._sanitize_response(response)
     87         out = StringIO()

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _get_response(self, url, params)
    118         if params is not None and len(params) > 0:
    119             url = url + "?" + urlencode(params)
--> 120         raise RemoteDataError('Unable to read URL: {0}'.format(url))
    121
    122     def _read_lines(self, out):

RemoteDataError: Unable to read URL: http://ichart.finance.yahoo.com/table.csv?
a=0&ignore=.csv&s=F&b=1&e=27&d=0&g=d&f=2013&c=2010
```

```
In [6]: f.ix['2010-01-04']

NameErrorTraceback (most recent call last)
<ipython-input-6-a881e6f45410> in <module>()
----> 1 f.ix['2010-01-04']

NameError: name 'f' is not defined
```

Historical corporate actions (Dividends and Stock Splits) with ex-dates from Yahoo! Finance.

```
In [7]: import pandas_datareader.data as web

In [8]: import datetime

In [9]: start = datetime.datetime(2010, 1, 1)

In [10]: end = datetime.datetime(2015, 5, 9)

In [11]: web.DataReader('AAPL', 'yahoo-actions', start, end)

RemoteDataErrorTraceback (most recent call last)
<ipython-input-11-d102632855e1> in <module>()
----> 1 web.DataReader('AAPL', 'yahoo-actions', start, end)

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/data.pyc in DataReader(name, data_source, start, end, retry_count,
pause, session, access_key)
    120            return YahooActionReader(symbols=name, start=start, end=end,
    121                                      retry_count=retry_count, pause=pause,
--> 122                                      session=session).read()
    123     elif data_source == "yahoo-dividends":
    124            return YahooDailyReader(symbols=name, start=start, end=end,

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in read(self)
    155            if isinstance(self.symbols, (compat.string_types, int)):
    156                df = self._read_one_data(self.url,
--> 157                                          params=self._get_params(self.symbols))
    158            # Or multiple symbols, (e.g., ['GOOG', 'AAPL', 'MSFT'])
    159            elif isinstance(self.symbols, DataFrame):

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _read_one_data(self, url, params)
     72            """ read one data from specified URL """
     73            if self._format == 'string':
---> 74                out = self._read_url_as_StringIO(url, params=params)
     75            elif self._format == 'json':
     76                out = self._get_response(url, params=params).json()

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _read_url_as_StringIO(self, url, params)
     83            Open url (and retry)
     84            """
---> 85            response = self._get_response(url, params=params)
     86            text = self._sanitize_response(response)
     87            out = StringIO()

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _get_response(self, url, params)
    118            if params is not None and len(params) > 0:
    119                url = url + "?" + urlencode(params)
--> 120            raise RemoteDataError('Unable to read URL: {0}'.format(url))
    121
    122     def _read_lines(self, out):

RemoteDataError: Unable to read URL: http://ichart.finance.yahoo.com/x?
a=0&s=AAPL&b=1&e=9&d=4&g=v&f=2015&c=2010
```

Historical dividends from Yahoo! Finance.

```
In [12]: import pandas_datareader.data as web

In [13]: import datetime

In [14]: start = datetime.datetime(2010, 1, 1)

In [15]: end = datetime.datetime(2013, 1, 27)

In [16]: f = web.DataReader("F", 'yahoo-dividends', start, end)

RemoteDataErrorTraceback (most recent call last)
<ipython-input-16-42b2f1e0124d> in <module>()
----> 1 f = web.DataReader("F", 'yahoo-dividends', start, end)

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/data.pyc in DataReader(name, data_source, start, end, retry_count,
pause, session, access_key)
    125                                        adjust_price=False, chunksize=25,
    126                                        retry_count=retry_count, pause=pause,
--> 127                                        session=session, interval='v').read()
    128
    129     elif data_source == "google":

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/yahoo/daily.pyc in read(self)
     75     def read(self):
     76         """ read one data from specified URL """
---> 77         df = super(YahooDailyReader, self).read()
     78         if self.ret_index:
     79             df['Ret_Index'] = _calc_return_index(df['Adj Close'])

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in read(self)
    155         if isinstance(self.symbols, (compat.string_types, int)):
    156             df = self._read_one_data(self.url,
--> 157                                      params=self._get_params(self.symbols))
    158         # Or multiple symbols, (e.g., ['GOOG', 'AAPL', 'MSFT'])
    159         elif isinstance(self.symbols, DataFrame):

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _read_one_data(self, url, params)
     72         """ read one data from specified URL """
     73         if self._format == 'string':
---> 74             out = self._read_url_as_StringIO(url, params=params)
     75         elif self._format == 'json':
     76             out = self._get_response(url, params=params).json()

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _read_url_as_StringIO(self, url, params)
     83         Open url (and retry)
     84         """
---> 85         response = self._get_response(url, params=params)
     86         text = self._sanitize_response(response)
     87         out = StringIO()

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/base.pyc in _get_response(self, url, params)
    118         if params is not None and len(params) > 0:
    119             url = url + "?" + urlencode(params)
--> 120         raise RemoteDataError('Unable to read URL: {0}'.format(url))
    121
    122     def _read_lines(self, out):

RemoteDataError: Unable to read URL: http://ichart.finance.yahoo.com/table.csv?
a=0&ignore=.csv&s=F&b=1&e=27&d=0&g=v&f=2013&c=2010
```

```
In [17]: f

NameErrorTraceback (most recent call last)
<ipython-input-17-9a8ad92c50ca> in <module>()
----> 1 f

NameError: name 'f' is not defined
```

Yahoo! Finance Quotes

Experimental

The YahooQuotesReader class allows to get quotes data from Yahoo! Finance.

```
In [18]: import pandas_datareader.data as web

In [19]: amzn = web.get_quote_yahoo('AMZN')

In [20]: amzn
Out[20]:
          PE change_pct    last  short_ratio    time
AMZN  181.93     +0.85%  966.07         2.09  4:00pm
```

Yahoo! Finance Options

Experimental

The Options class allows the download of options data from Yahoo! Finance.

The `get_all_data` method downloads and caches option data for all expiry months and provides a formatted `DataFrame` with a hierarchical index, so its easy to get to the specific option you want.

```
In [21]: from pandas_datareader.data import Options

In [22]: aapl = Options('aapl', 'yahoo')

In [23]: data = aapl.get_all_data()

In [24]: data.iloc[0:5, 0:5]
Out[24]:
                                               Last      Bid      Ask       Chg  \
Strike Expiry     Type Symbol
2.5    2017-05-19 call AAPL170519C00002500  153.15   151.95   154.40   8.449997
                  put  AAPL170519P00002500    0.02     0.00     0.02   0.000000
       2017-06-16 put  AAPL170616P00002500    0.01     0.00     0.01  -0.010000
       2017-08-18 call AAPL170818C00002500  153.30   153.15   154.10   0.000000
       2018-01-19 call AAPL180119C00002500  153.13   151.60   154.55   0.000000

                                                PctChg
Strike Expiry     Type Symbol
2.5    2017-05-19 call AAPL170519C00002500    5.839666
                  put  AAPL170519P00002500    0.000000
       2017-06-16 put  AAPL170616P00002500  -50.000000
       2017-08-18 call AAPL170818C00002500    0.000000
       2018-01-19 call AAPL180119C00002500    0.000000

#Show the $100 strike puts at all expiry dates:
In [25]: data.loc[(100, slice(None), 'put'),:].iloc[0:5, 0:5]
Out[25]:
                                             Last   Bid   Ask   Chg  PctChg
Strike Expiry     Type Symbol
100    2017-05-19 put  AAPL170519P00100000   0.01  0.00  0.02 -0.01     -50
       2017-05-26 put  AAPL170526P00100000   0.04  0.01  0.05  0.00       0
       2017-06-16 put  AAPL170616P00100000   0.01  0.00  0.01  0.00       0
       2017-06-30 put  AAPL170630P00100000   0.01  0.00  0.03  0.00       0
       2017-07-21 put  AAPL170721P00100000   0.01  0.00  0.01 -0.01     -50

#Show the volume traded of $100 strike puts at all expiry dates:
In [26]: data.loc[(100, slice(None), 'put'),'Vol'].head()
Out[26]:
Strike  Expiry       Type   Symbol
100     2017-05-19   put    AAPL170519P00100000      25
        2017-05-26   put    AAPL170526P00100000       2
        2017-06-16   put    AAPL170616P00100000      52
        2017-06-30   put    AAPL170630P00100000      10
        2017-07-21   put    AAPL170721P00100000      15
Name: Vol, dtype: float64
```

If you don't want to download all the data, more specific requests can be made.

```
In [27]: import datetime

In [28]: expiry = datetime.date(2016, 1, 1)

In [29]: data = aapl.get_call_data(expiry=expiry)

In [30]: data.iloc[0:5:, 0:5]
Out[30]:
                                                 Last      Bid      Ask      Chg  \
Strike Expiry       Type Symbol
2.5    2017-05-19 call AAPL170519C00002500   153.15   151.95   154.40   8.449997
5.0    2017-05-19 call AAPL170519C00005000   150.65   149.45   151.65   0.000000
7.5    2017-05-19 call AAPL170519C00007500   148.05   146.95   149.30   0.000000
10.0   2017-05-19 call AAPL170519C00010000   143.53   145.65   146.60   0.000000
20.0   2017-05-19 call AAPL170519C00020000   133.52   135.65   136.60   0.000000

                                                 PctChg
Strike Expiry       Type Symbol
2.5    2017-05-19 call AAPL170519C00002500   5.839666
5.0    2017-05-19 call AAPL170519C00005000   0.000000
7.5    2017-05-19 call AAPL170519C00007500   0.000000
10.0   2017-05-19 call AAPL170519C00010000   0.000000
20.0   2017-05-19 call AAPL170519C00020000   0.000000
```

Note that if you call `get_all_data` first, this second call will happen much faster, as the data is cached.

If a given expiry date is not available, data for the next available expiry will be returned (January 15, 2015 in the above example).

Available expiry dates can be accessed from the `expiry_dates` property.

```
In [31]: aapl.expiry_dates
Out[31]:
[datetime.date(2017, 5, 19),
 datetime.date(2017, 5, 26),
 datetime.date(2017, 6, 2),
 datetime.date(2017, 6, 9),
 datetime.date(2017, 6, 16),
 datetime.date(2017, 6, 23),
 datetime.date(2017, 6, 30),
 datetime.date(2017, 7, 21),
 datetime.date(2017, 8, 18),
 datetime.date(2017, 9, 15),
 datetime.date(2017, 10, 20),
 datetime.date(2017, 11, 17),
 datetime.date(2017, 12, 15),
 datetime.date(2018, 1, 19),
 datetime.date(2018, 2, 16),
 datetime.date(2018, 6, 15),
 datetime.date(2019, 1, 18)]

In [32]: data = aapl.get_call_data(expiry=aapl.expiry_dates[0])

In [33]: data.iloc[0:5:, 0:5]
Out[33]:
                                                 Last     Bid     Ask      Chg  \
Strike Expiry     Type Symbol
2.5    2017-05-19 call AAPL170519C00002500  153.15  151.95  154.40  8.449997
5.0    2017-05-19 call AAPL170519C00005000  150.65  149.45  151.65  0.000000
7.5    2017-05-19 call AAPL170519C00007500  148.05  146.95  149.30  0.000000
10.0   2017-05-19 call AAPL170519C00010000  143.53  145.65  146.60  0.000000
20.0   2017-05-19 call AAPL170519C00020000  133.52  135.65  136.60  0.000000

                                               PctChg
Strike Expiry     Type Symbol
2.5    2017-05-19 call AAPL170519C00002500  5.839666
5.0    2017-05-19 call AAPL170519C00005000  0.000000
7.5    2017-05-19 call AAPL170519C00007500  0.000000
10.0   2017-05-19 call AAPL170519C00010000  0.000000
20.0   2017-05-19 call AAPL170519C00020000  0.000000
```

A list-like object containing dates can also be passed to the expiry parameter, returning options data for all expiry dates in the list.

```
In [34]: data = aapl.get_near_stock_price(expiry=aapl.expiry_dates[0:3])

In [35]: data.iloc[0:5:, 0:5]
Out[35]:
                                              Last   Bid   Ask    Chg    PctChg
Strike Expiry     Type Symbol
152.5  2017-05-26 call AAPL170526C00152500  4.05  4.00  4.10  -0.45   -9.999995
       2017-06-02 call AAPL170602C00152500  4.09  4.10  4.15  -0.16   -3.764702
155.0  2017-05-19 call AAPL170519C00155000  1.11  1.11  1.12  -0.48  -30.188679
       2017-05-26 call AAPL170526C00155000  2.00  2.00  2.02  -0.35  -14.893614
       2017-06-02 call AAPL170602C00155000  2.50  2.52  2.55  -0.35  -12.280699
```

The `month` and `year` parameters can be used to get all options data for a given month.

Google Finance

```
In [36]: import pandas_datareader.data as web

In [37]: import datetime

In [38]: start = datetime.datetime(2010, 1, 1)

In [39]: end = datetime.datetime(2013, 1, 27)

In [40]: f = web.DataReader("F", 'google', start, end)

In [41]: f.ix['2010-01-04']
Out[41]:
Open            10.17
High            10.28
Low             10.05
Close           10.28
Volume      60855796.00
Name: 2010-01-04 00:00:00, dtype: float64
```

Google Finance Quotes

Experimental

The GoogleQuotesReader class allows to get quotes data from Google Finance.

```
In [42]: import pandas_datareader.data as web

In [43]: q = web.get_quote_google(['AMZN', 'GOOG'])

In [44]: q
Out[44]:
      change_pct    last                 time
AMZN        0.85  966.07  2017-05-16 16:00:00
GOOG        0.63  943.00  2017-05-16 16:00:00
```

Google Finance Options

Experimental

The Options class allows the download of options data from Google Finance.

The `get_options_data` method downloads options data for specified expiry date and provides a formatted `DataFrame` with a hierarchical index, so its easy to get to the specific option you want.

Available expiry dates can be accessed from the `expiry_dates` property.

```
In [45]: from pandas_datareader.data import Options

In [46]: goog = Options('goog', 'google')

In [47]: data = goog.get_options_data(expiry=goog.expiry_dates[0])

In [48]: data.iloc[0:5, 0:5]
Out[48]:
                                              Last    Bid    Ask  Chg  PctChg
Strike Expiry      Type Symbol
350    2017-06-16 call GOOG170616C00350000  580.50  590.5  593.80    0       0
                  put  GOOG170616P00350000    0.03    NaN    0.45    0       0
360    2017-06-16 call GOOG170616C00360000  482.80  580.5  585.00    0       0
                  put  GOOG170616P00360000    0.05    NaN    0.15    0       0
370    2017-06-16 call GOOG170616C00370000  474.20  570.5  575.00    0       0
```

Enigma

Access datasets from Enigma, the world's largest repository of structured public data.

```
In [49]: import os

In [50]: import pandas_datareader as pdr

In [51]: df = pdr.get_data_enigma('enigma.trade.ams.toxic.2015', os.getenv('ENIGMA_API_KEY'))

ValueErrorTraceback (most recent call last)
<ipython-input-51-8b19d4dc1932> in <module>()
----> 1 df = pdr.get_data_enigma('enigma.trade.ams.toxic.2015', os.getenv('ENIGMA_API_KEY'))

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/data.pyc in get_data_enigma(*args, **kwargs)
     42
     43 def get_data_enigma(*args, **kwargs):
---> 44     return EnigmaReader(*args, **kwargs).read()
     45
     46

/home/docs/checkouts/readthedocs.org/user_builds/pandas-
datareader/envs/latest/local/lib/python2.7/site-packages/pandas_datareader-0.4.0-
py2.7.egg/pandas_datareader/enigma.pyc in __init__(self, datapath, api_key, retry_count, pause,
session)
     47             self._api_key = os.getenv('ENIGMA_API_KEY')
     48             if self._api_key is None:
---> 49                 raise ValueError("Please provide an Enigma API key or set "
     50                                   "the ENIGMA_API_KEY environment variable\n"
     51                                   "If you do not have an API key, you can get "

ValueError: Please provide an Enigma API key or set the ENIGMA_API_KEY environment variable
If you do not have an API key, you can get one here: https://app.enigma.io/signup

In [52]: df.columns

NameErrorTraceback (most recent call last)
<ipython-input-52-6a4642092433> in <module>()
----> 1 df.columns

NameError: name 'df' is not defined
```

FRED

```
In [53]: import pandas_datareader.data as web

In [54]: import datetime

In [55]: start = datetime.datetime(2010, 1, 1)

In [56]: end = datetime.datetime(2013, 1, 27)

In [57]: gdp = web.DataReader("GDP", "fred", start, end)

In [58]: gdp.ix['2013-01-01']
Out[58]:
GDP     16475.4
Name: 2013-01-01 00:00:00, dtype: float64

# Multiple series:
In [59]: inflation = web.DataReader(["CPIAUCSL", "CPILFESL"], "fred", start, end)

In [60]: inflation.head()
Out[60]:
            CPIAUCSL   CPILFESL
DATE
2010-01-01   217.488    220.633
2010-02-01   217.281    220.731
2010-03-01   217.353    220.783
2010-04-01   217.403    220.822
2010-05-01   217.290    220.962
```

Fama/French

Access datasets from the Fama/French Data Library. The `get_available_datasets` function returns a list of all available datasets.

```
In [61]: from pandas_datareader.famafrench import get_available_datasets

In [62]: import pandas_datareader.data as web

In [63]: len(get_available_datasets())
Out[63]: 262

In [64]: ds = web.DataReader("5_Industry_Portfolios", "famafrench")

In [65]: print(ds['DESCR'])
5 Industry Portfolios
---------------------

This file was created by CMPT_IND_RETS using the 201703 CRSP database. It contains value- and
equal-weighted returns for 5 industry portfolios. The portfolios are constructed at the end of
June. The annual returns are from January to December. Missing data are indicated by -99.99 or
-999. Copyright 2017 Kenneth R. French

  0 : Average Value Weighted Returns -- Monthly (87 rows x 5 cols)
  1 : Average Equal Weighted Returns -- Monthly (87 rows x 5 cols)
  2 : Average Value Weighted Returns -- Annual (7 rows x 5 cols)
  3 : Average Equal Weighted Returns -- Annual (7 rows x 5 cols)
  4 : Number of Firms in Portfolios (87 rows x 5 cols)
  5 : Average Firm Size (87 rows x 5 cols)
  6 : Sum of BE / Sum of ME (7 rows x 5 cols)
  7 : Value-Weighted Average of BE/ME (7 rows x 5 cols)

In [66]: ds[4].ix['1926-07']

KeyErrorTraceback (most recent call last)
<ipython-input-66-79093f940e41> in <module>()
----> 1 ds[4].ix['1926-07']

/usr/lib/python2.7/dist-packages/pandas/core/indexing.pyc in __getitem__(self, key)
     68                 return self._getitem_tuple(key)
     69             else:
---> 70                 return self._getitem_axis(key, axis=0)
     71
     72     def _get_label(self, label, axis=0):

/usr/lib/python2.7/dist-packages/pandas/core/indexing.pyc in _getitem_axis(self, key, axis)
    965                     return self._get_loc(key, axis=axis)
    966
--> 967                 return self._get_label(key, axis=axis)
    968
    969     def _getitem_iterable(self, key, axis=0):

/usr/lib/python2.7/dist-packages/pandas/core/indexing.pyc in _get_label(self, label, axis)
     84                 raise IndexingError('no slices here, handle elsewhere')
     85
---> 86             return self.obj._xs(label, axis=axis)
     87
     88     def _get_loc(self, key, axis=0):

/usr/lib/python2.7/dist-packages/pandas/core/generic.pyc in xs(self, key, axis, level, copy, drop_level)
   1484                                                                        drop_level=drop_level)
   1485             else:
-> 1486                 loc = self.index.get_loc(key)
   1487
   1488                 if isinstance(loc, np.ndarray):

/usr/lib/python2.7/dist-packages/pandas/tseries/period.pyc in get_loc(self, key, method, tolerance)
    667                 return Index.get_loc(self, key.ordinal, method, tolerance)
    668             except KeyError:
--> 669                 raise KeyError(key)
    670
    671     def _maybe_cast_slice_bound(self, label, side, kind):

KeyError: Period('1926-07', 'M')
```

World Bank

`pandas` users can easily access thousands of panel data series from the World Bank's World Development Indicators by using the `wb` I/O functions.

Indicators

Either from exploring the World Bank site, or using the search function included, every world bank indicator is accessible.

For example, if you wanted to compare the Gross Domestic Products per capita in constant dollars in North America, you would use the `search` function:

```
In [1]: from pandas_datareader import wb

In [2]: wb.search('gdp.*capita.*const').iloc[:,:2]
Out[2]:
                   id                                      name
3242          GDPPCKD          GDP per Capita, constant US$, millions
5143      NY.GDP.PCAP.KD              GDP per capita (constant 2005 US$)
5145      NY.GDP.PCAP.KN                   GDP per capita (constant LCU)
5147  NY.GDP.PCAP.PP.KD  GDP per capita, PPP (constant 2005 internation...
```

Then you would use the `download` function to acquire the data from the World Bank's servers:

```
In [3]: dat = wb.download(indicator='NY.GDP.PCAP.KD', country=['US', 'CA', 'MX'], start=2005, end=2008)

In [4]: print(dat)
                          NY.GDP.PCAP.KD
country       year
Canada        2008   36005.5004978584
              2007   36182.9138439757
              2006   35785.9698172849
              2005   35087.8925933298
Mexico        2008   8113.10219480083
              2007   8119.21298908649
              2006   7961.96818458178
              2005   7666.69796097264
United States 2008   43069.5819857208
              2007   43635.5852068142
              2006    43228.111147107
              2005   42516.3934699993
```

The resulting dataset is a properly formatted `DataFrame` with a hierarchical index, so it is easy to apply `.groupby` transformations to it:

```
In [6]: dat['NY.GDP.PCAP.KD'].groupby(level=0).mean()
Out[6]:
country
Canada             35765.569188
Mexico              7965.245332
United States      43112.417952
dtype: float64
```

Now imagine you want to compare GDP to the share of people with cellphone contracts around the world.

```
In [7]: wb.search('cell.*%').iloc[:,:2]
Out[7]:
                   id                                                name
3990  IT.CEL.SETS.FE.ZS  Mobile cellular telephone users, female (% of ...
3991  IT.CEL.SETS.MA.ZS  Mobile cellular telephone users, male (% of po...
4027       IT.MOB.COV.ZS  Population coverage of mobile cellular telepho...
```

Notice that this second search was much faster than the first one because `pandas` now has a cached list of available data series.

```
In [13]: ind = ['NY.GDP.PCAP.KD', 'IT.MOB.COV.ZS']
In [14]: dat = wb.download(indicator=ind, country='all', start=2011, end=2011).dropna()
In [15]: dat.columns = ['gdp', 'cellphone']
In [16]: print(dat.tail())
                        gdp  cellphone
country    year
Swaziland  2011  2413.952853      94.9
Tunisia    2011  3687.340170     100.0
Uganda     2011   405.332501     100.0
Zambia     2011   767.911290      62.0
Zimbabwe   2011   419.236086      72.4
```

Finally, we use the `statsmodels` package to assess the relationship between our two variables using ordinary least squares regression. Unsurprisingly, populations in rich countries tend to use cellphones at a higher rate:

```
In [17]: import numpy as np
In [18]: import statsmodels.formula.api as smf
In [19]: mod = smf.ols("cellphone ~ np.log(gdp)", dat).fit()
In [20]: print(mod.summary())
                        OLS Regression Results
==============================================================================
Dep. Variable:              cellphone   R-squared:                       0.297
Model:                            OLS   Adj. R-squared:                  0.274
Method:                 Least Squares   F-statistic:                     13.08
Date:                Thu, 25 Jul 2013   Prob (F-statistic):            0.00105
Time:                        15:24:42   Log-Likelihood:                -139.16
No. Observations:                  33   AIC:                             282.3
Df Residuals:                      31   BIC:                             285.3
Df Model:                           1
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept      16.5110     19.071      0.866      0.393     -22.384     55.406
np.log(gdp)     9.9333      2.747      3.616      0.001       4.331     15.535
==============================================================================
Omnibus:                       36.054   Durbin-Watson:                   2.071
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              119.133
Skew:                          -2.314   Prob(JB):                     1.35e-26
Kurtosis:                      11.077   Cond. No.                         45.8
==============================================================================
```

Country Codes

The `country` argument accepts a string or list of mixed two or three character ISO country codes, as well as dynamic World Bank exceptions to the ISO standards.

For a list of the the hard-coded country codes (used solely for error handling logic) see `pandas_datareader.wb.country_codes`.

Problematic Country Codes & Indicators

🛈 Note

The World Bank's country list and indicators are dynamic. As of 0.15.1, `wb.download()` is more flexible. To achieve this, the warning and exception logic changed.

The world bank converts some country codes, in their response, which makes error checking by pandas difficult. Retired indicators still persist in the search.

Given the new flexibility of 0.15.1, improved error handling by the user may be necessary for fringe cases.

To help identify issues:

There are at least 4 kinds of country codes:

1. Standard (2/3 digit ISO) - returns data, will warn and error properly.
2. Non-standard (WB Exceptions) - returns data, but will falsely warn.
3. Blank - silently missing from the response.
4. Bad - causes the entire response from WB to fail, always exception inducing.

There are at least 3 kinds of indicators:

1. Current - Returns data.
2. Retired - Appears in search results, yet won't return data.
3. Bad - Will not return data.

Use the `errors` argument to control warnings and exceptions. Setting errors to ignore or warn, won't stop failed responses. (ie, 100% bad indicators, or a single "bad" (#4 above) country code).

See docstrings for more info.

OECD

OECD Statistics are avaliable via `DataReader`. You have to specify OECD's data set code.

To confirm data set code, access to `each data -> Export -> SDMX Query`. Following example is to download "Trade Union Density" data which set code is "UN_DEN".

```
In [67]: import pandas_datareader.data as web

In [68]: import datetime

In [69]: df = web.DataReader('UN_DEN', 'oecd', end=datetime.datetime(2012, 1, 1))

In [70]: df.columns
Out[70]:
Index([u'Australia', u'Austria', u'Belgium', u'Canada', u'Czech Republic',
       u'Denmark', u'Finland', u'France', u'Germany', u'Greece', u'Hungary',
       u'Iceland', u'Ireland', u'Italy', u'Japan', u'Korea', u'Luxembourg',
       u'Mexico', u'Netherlands', u'New Zealand', u'Norway', u'Poland',
       u'Portugal', u'Slovak Republic', u'Spain', u'Sweden', u'Switzerland',
       u'Turkey', u'United Kingdom', u'United States', u'OECD countries',
       u'Chile', u'Slovenia', u'Estonia', u'Israel'],
      dtype='object', name=u'Country')

In [71]: df[['Japan', 'United States']]
Out[71]:
Country         Japan  United States
Time
2010-01-01  18.403807      11.383460
2011-01-01  18.995042      11.329488
2012-01-01  17.972384      10.815352
```

Eurostat

[Eurostat](#) are avaliable via `DataReader`.

Get \` Rail accidents by type of accident (ERA data) <[http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=tran_sf_railac&lang=en](#)>\`_ data. The result will be a `DataFrame` which has `DatetimeIndex` as index and `MultiIndex` of attributes or countries as column. The target URL is:

- [http://appsso.eurostat.ec.europa.eu/nui/show.do?dataset=tran_sf_railac&lang=en](#)

You can specify dataset ID "tran_sf_railac" to get corresponding data via `DataReader`.

```
In [72]: import pandas_datareader.data as web

In [73]: df = web.DataReader("tran_sf_railac", 'eurostat')

In [74]: df
Out[74]:
ACCIDENT     Collisions of trains, including collisions with obstacles within the clearance gauge
\
UNIT                                                                                       Number
GEO                                                                                       Austria
FREQ                                                                                       Annual
TIME_PERIOD
2010-01-01                                                    3
2011-01-01                                                    2
2012-01-01                                                    1
2013-01-01                                                    4
2014-01-01                                                    1
2015-01-01                                                    7

ACCIDENT                                                                           \
UNIT
GEO          Belgium Bulgaria Switzerland Channel Tunnel Czech Republic
FREQ          Annual   Annual      Annual         Annual          Annual
TIME_PERIOD
2010-01-01         5        2           5              0               3
2011-01-01         0        0           4              0               6
2012-01-01         3        3           4              0               6
2013-01-01         1        2           6              0               5
2014-01-01         3        4           0              0              13
2015-01-01         0        3           3              0              14

ACCIDENT                                                                         \
UNIT
GEO          Germany (until 1990 former territory of the FRG) Denmark Estonia
FREQ                                                   Annual   Annual  Annual
TIME_PERIOD
2010-01-01                                                 13        0       1
2011-01-01                                                 18        1       0
2012-01-01                                                 23        1       3
2013-01-01                                                 29        0       0
2014-01-01                                                 32        0       0
2015-01-01                                                 40        3       0

ACCIDENT                  ...      Total                                 \
UNIT                      ...     Number
GEO          Greece       ...      Latvia Netherlands Norway Poland Portugal
FREQ         Annual       ...      Annual      Annual Annual Annual   Annual
TIME_PERIOD               ...
2010-01-01        4       ...          41          24     20    449       42
2011-01-01        1       ...          35          29     36    488       27
2012-01-01        2       ...          25          30     19    379       36
2013-01-01        2       ...          26          36     30    328       48
2014-01-01        1       ...          22          20     28    313       50
2015-01-01        1       ...          25          31     19    307       23

ACCIDENT
UNIT
GEO          Romania Sweden Slovenia Slovakia United Kingdom
FREQ          Annual Annual   Annual   Annual         Annual
TIME_PERIOD
2010-01-01        271     69       21       85             62
2011-01-01        217     54       11       84             78
2012-01-01        215     47       14       96             75
2013-01-01        180     43       13       94             84
2014-01-01        185     53       15      113             54
2015-01-01        141     40       14       87             40

[6 rows x 210 columns]
```

EDGAR Index

** As of December 31st, the SEC disabled access via FTP. EDGAR support currently broken until re-write to use HTTPS. **

Company filing index from EDGAR (SEC).

The daily indices get large quickly (i.e. the set of daily indices from 1994 to 2015 is 1.5GB), and the FTP server will close the connection past some downloading threshold . In testing, pulling one year at a time works well. If the FTP server starts refusing your connections, you should be able to reconnect after waiting a few minutes.

TSP Fund Data

Download mutual fund index prices for the TSP.

```
In [75]: import pandas_datareader.tsp as tsp

In [76]: tspreader = tsp.TSPReader(start='2015-10-1', end='2015-12-31')

In [77]: tspreader.read()
Out[77]:
            L Income    L 2020    L 2030    L 2040    L 2050    G Fund    F Fund  \
date
2015-10-01   17.5164   22.5789   24.2159   25.5690   14.4009   14.8380   17.0467
2015-10-02   17.5707   22.7413   24.4472   25.8518   14.5805   14.8388   17.0924
2015-10-05   17.6395   22.9582   24.7571   26.2306   14.8233   14.8413   17.0531
2015-10-06   17.6338   22.9390   24.7268   26.1898   14.7979   14.8421   17.0790
2015-10-07   17.6639   23.0324   24.8629   26.3598   14.9063   14.8429   17.0725
2015-10-08   17.6957   23.1364   25.0122   26.5422   15.0240   14.8437   17.0363
2015-10-09   17.7048   23.1646   25.0521   26.5903   15.0554   14.8445   17.0511
...              ...       ...       ...       ...       ...       ...       ...
2015-12-22   17.7493   23.1452   24.9775   26.4695   14.9611   14.9076   16.9607
2015-12-23   17.8015   23.3149   25.2208   26.7663   15.1527   14.9084   16.9421
2015-12-24   17.7991   23.3039   25.2052   26.7481   15.1407   14.9093   16.9596
2015-12-28   17.7950   23.2811   25.1691   26.7015   15.1101   14.9128   16.9799
2015-12-29   17.8270   23.3871   25.3226   26.8905   15.2319   14.9137   16.9150
2015-12-30   17.8066   23.3216   25.2267   26.7707   15.1556   14.9146   16.9249
2015-12-31   17.7733   23.2085   25.0635   26.5715   15.0263   14.9154   16.9549

             C Fund    S Fund    I Fund
date
2015-10-01   25.7953   34.0993   23.3202    NaN
2015-10-02   26.1669   34.6504   23.6367
2015-10-05   26.6467   35.3565   24.1475
2015-10-06   26.5513   35.1320   24.2294
2015-10-07   26.7751   35.6035   24.3671
2015-10-08   27.0115   35.9016   24.6406
2015-10-09   27.0320   35.9772   24.7723
...              ...       ...       ...   ...
2015-12-22   27.4848   35.0903   23.8679
2015-12-23   27.8272   35.5749   24.3623
2015-12-24   27.7831   35.6084   24.3272
2015-12-28   27.7230   35.4625   24.2816
2015-12-29   28.0236   35.8047   24.4757
2015-12-30   27.8239   35.5126   24.4184
2015-12-31   27.5622   35.2356   24.0952

[62 rows x 11 columns]
```

Oanda currency historical rate

Download currency historical rate from Oanda.

```
In [1]: from pandas_datareader.oanda import get_oanda_currency_historical_rates
In [2]: start, end = "2016-01-01", "2016-06-01"
In [3]: quote_currency = "USD"
In [4]: base_currency = ["EUR", "GBP", "JPY"]
In [5]: df_rates = get_oanda_currency_historical_rates(
            start, end,
            quote_currency=quote_currency,
            base_currency=base_currency
        )
In [6]: print(df_rates)

            EUR/USD    GBP/USD    JPY/USD
  Date
  2016-01-01  1.087090  1.473989  0.008320
  2016-01-02  1.087090  1.473989  0.008320
  2016-01-03  1.087090  1.473989  0.008320
  2016-01-04  1.086730  1.473481  0.008370
  2016-01-05  1.078760  1.469430  0.008388

  ...              ...       ...       ...
  2016-05-28  1.111669  1.462630  0.009072
  2016-05-29  1.111669  1.462630  0.009072
  2016-05-30  1.112479  1.461999  0.009006
  2016-05-31  1.114269  1.461021  0.009010
  2016-06-01  1.115170  1.445410  0.009095

  [153 rows x 3 columns]
```

Nasdaq Trader Symbol Definitions

Download the latest symbols from
`Nasdaq<ftp://ftp.nasdaqtrader.com/SymbolDirectory/nasdaqtraded.txt/>`__.

Note that Nasdaq updates this file daily, and historical versions are not available. More
information on the *field<http://www.nasdaqtrader.com/trader.aspx?id=symboldirdefs/>*
definitions.

```
In [12]: from pandas_datareader.nasdaq_trader import get_nasdaq_symbols
In [13]: symbols = get_nasdaq_symbols()
In [14]: print(symbols.ix['IBM'])
    Nasdaq Traded                                             True
    Security Name      International Business Machines Corporation Co...
    Listing Exchange                                             N
    Market Category
    ETF                                                      False
    Round Lot Size                                             100
    Test Issue                                               False
    Financial Status                                           NaN
    CQS Symbol                                                 IBM
    NASDAQ Symbol                                              IBM
    NextShares                                               False
    Name: IBM, dtype: object
```