Remote Data Access

> **Warning**
>
> Yahoo! Finance has been immediately deprecated. Yahoo! substantially altered their API in late 2017 and the csv endpoint was retired.

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:

- Google Finance
- Morningstar
- IEX
- Robinhood
- Enigma
- Quandl
- St.Louis FED (FRED)
- Kenneth French's data library
- World Bank
- OECD
- Eurostat
- Thrift Savings Plan
- Nasdaq Trader symbol definitions
- Stooq
- MOEX

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.

Google Finance

> **Warning**
>
> Google'a API has become less reliable during 2017. While the google datareader often works as expected, it is not uncommon to experience a range of errors when attempting to read data, especially in bulk.

```
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', 'google', start, end)

In [6]: f.ix['2010-01-04']
Out[6]:
Open            10.17
High            10.28
Low             10.05
Close           10.28
Volume    60855796.00
Name: 2010-01-04 00:00:00, dtype: float64
```

Tiingo

Tiingo is a tracing platform that provides a data api with historical end-of-day prices on equities, mutual funds and ETFs. Free registration is required to get an API key. Free accounts are rate limited and can access a limited number of symbols (500 at the time of writing).

```
In [7]: import os

In [8]: import pandas_datareader as pdr

In [9]: df = pdr.get_data_tiingo('GOOG', api_key=os.getenv('TIINGO_API_KEY'))
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-9-c390cc1bcafe> in <module>()
----> 1 df = pdr.get_data_tiingo('GOOG', api_key=os.getenv('TIINGO_API_KEY'))

AttributeError: module 'pandas_datareader' has no attribute 'get_data_tiingo'

In [10]: df.head()
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-c42a15b2c7cf> in <module>()
----> 1 df.head()

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

Morningstar

OHLC and Volume data is available from Morningstar using the same API which powers their charts.

```
In [11]: import pandas_datareader.data as web

In [12]: from datetime import datetime

In [13]: start = datetime(2015, 2, 9)

In [14]: end = datetime(2017, 5, 24)

In [15]: f = web.DataReader('F', 'morningstar', start, end)

In [16]: f.head()
Out[16]:
                  Close   High    Low   Open    Volume
Symbol Date
F      2015-02-09  15.92  16.03  15.72  15.76  20286720
       2015-02-10  16.09  16.14  15.91  16.05  27928530
       2015-02-11  16.25  16.31  16.01  16.08  34285331
       2015-02-12  16.36  16.45  16.30  16.34  23738806
       2015-02-13  16.30  16.36  16.19  16.33  19954568
```

IEX

The Investors Exchange (IEX) provides a wide range of data through an API. Historical stock prices are available for up to 5 years:

```
In [17]: import pandas_datareader.data as web

In [18]: from datetime import datetime

In [19]: start = datetime(2015, 2, 9)

In [20]: end = datetime(2017, 5, 24)

In [21]: f = web.DataReader('F', 'iex', start, end)
5y

In [22]: f.loc['2015-02-09']
Out[22]:
open             15.76
high             16.03
low              15.72
close            15.92
volume    20286720.00
Name: 2015-02-09, dtype: float64
```

There are additional interfaces to this API that are directly exposed: tops (*'iex-tops'*) and last (*'iex-lasts'*). A third interface to the deep API is exposed through *Deep* class or the *get_iex_book* function.

```
In [23]: import pandas_datareader.data as web

In [24]: f = web.DataReader('gs', 'iex-tops')

In [25]: f[:10]
Out[25]:
                                  0
askPrice                          0
askSize                           0
bidPrice                          0
bidSize                           0
lastSalePrice                272.48
lastSaleSize                      3
lastSaleTime           1517518796044
lastUpdated            1517518800000
marketPercent                0.02056
sector         diversifiedfinancials
```

Robinhood

Robinhood is a stock trading platform with an API that provides a limited set of data. Historical daily data is limited to 1 year relative to today.

```
In [26]: import pandas_datareader.data as web

In [27]: from datetime import datetime

In [28]: f = web.DataReader('F', 'robinhood')

In [29]: f.head()
Out[29]:
                  close_price high_price  interpolated low_price open_price  \
symbol begins_at
F      2017-02-02     11.5323    11.6169         False   11.4854    11.5511
       2017-02-03     11.7953    11.8516         False   11.6356    11.6638
       2017-02-06     11.7577    11.8469         False   11.7014    11.7859
       2017-02-07     11.5887    11.7577         False   11.5605    11.7389
       2017-02-08     11.6263    11.6920         False   11.5230    11.5887

                  session    volume
symbol begins_at
F      2017-02-02     reg  29035383
       2017-02-03     reg  38245251
       2017-02-06     reg  26916768
       2017-02-07     reg  32914413
       2017-02-08     reg  26411417
```

Enigma

Access datasets from Enigma, the world's largest repository of structured public data. Note that the Enigma URL has changed from app.enigma.io as of release `0.6.0`, as the old API deprecated.

Datasets are unique identified by the `uuid4` at the end of a dataset's web address. For example, the following code downloads from USDA Food Recalls 1996 Data.

```
In [30]: import os

In [31]: import pandas_datareader as pdr

In [32]: df = pdr.get_data_enigma('292129b0-1275-44c8-a6a3-2a0881f24fe1',
os.getenv('ENIGMA_API_KEY'))
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-32-f46ac2b42095> in <module>()
----> 1 df = pdr.get_data_enigma('292129b0-1275-44c8-a6a3-2a0881f24fe1', os.getenv('ENIGMA_API_KEY'))

~/checkouts/readthedocs.org/user_builds/pandas-datareader/envs/latest/lib/python3.5/site-
packages/pandas_datareader-0.6.0-py3.5.egg/pandas_datareader/data.py in get_data_enigma(*args,
**kwargs)
     66
     67 def get_data_enigma(*args, **kwargs):
---> 68     return EnigmaReader(*args, **kwargs).read()
     69
     70

~/checkouts/readthedocs.org/user_builds/pandas-datareader/envs/latest/lib/python3.5/site-
packages/pandas_datareader-0.6.0-py3.5.egg/pandas_datareader/enigma.py in __init__(self, dataset_id,
api_key, retry_count, pause, session)
     41             self._api_key = os.getenv('ENIGMA_API_KEY')
     42             if self._api_key is None:
---> 43                 raise ValueError("Please provide an Enigma API key or set "
     44                                  "the ENIGMA_API_KEY environment variable\n"
     45                                  "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: http://public.enigma.com/signup

In [33]: df.columns
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-33-b666bf274d0a> in <module>()
----> 1 df.columns

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

Quandl

Daily financial data (prices of stocks, ETFs etc.) from Quandl. The symbol names consist of two parts: DB name and symbol name. DB names can be all the free ones listed on the Quandl website. Symbol names vary with DB name; for WIKI (US stocks), they are the common ticker symbols, in

some other cases (such as FSE) they can be a bit strange. Some sources are also mapped to suitable ISO country codes in the dot suffix style shown above, currently available for BE, CN, DE, FR, IN, JP, NL, PT, UK, US.

As of June 2017, each DB has a different data schema, the coverage in terms of time range is sometimes surprisingly small, and the data quality is not always good.

```
In [34]: import pandas_datareader.data as web

In [35]: symbol = 'WIKI/AAPL'  # or 'AAPL.US'

In [36]: df = web.DataReader(symbol, 'quandl', '2015-01-01', '2015-01-05')

In [37]: df.loc['2015-01-02']
Out[37]:
              Open    High     Low   Close      Volume  ExDividend  \
Date
2015-01-02  111.39  111.44  107.35  109.33  53204626.0         0.0

            SplitRatio     AdjOpen     AdjHigh      AdjLow    AdjClose  \
Date
2015-01-02         1.0  105.820966  105.868466  101.982949  103.863957

             AdjVolume
Date
2015-01-02  53204626.0
```

FRED

```
In [38]: import pandas_datareader.data as web

In [39]: import datetime

In [40]: start = datetime.datetime(2010, 1, 1)

In [41]: end = datetime.datetime(2013, 1, 27)

In [42]: gdp = web.DataReader('GDP', 'fred', start, end)

In [43]: gdp.ix['2013-01-01']
Out[43]:
GDP    16475.44
Name: 2013-01-01 00:00:00, dtype: float64

# Multiple series:
In [44]: inflation = web.DataReader(['CPIAUCSL', 'CPILFESL'], 'fred', start, end)

In [45]: inflation.head()
Out[45]:
            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 [46]: from pandas_datareader.famafrench import get_available_datasets

In [47]: import pandas_datareader.data as web

In [48]: len(get_available_datasets())
Out[48]: 262

In [49]: ds = web.DataReader('5_Industry_Portfolios', 'famafrench')

In [50]: print(ds['DESCR'])
5 Industry Portfolios
---------------------

This file was created by CMPT_IND_RETS using the 201712 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 (96 rows x 5 cols)
  1 : Average Equal Weighted Returns -- Monthly (96 rows x 5 cols)
  2 : Average Value Weighted Returns -- Annual (8 rows x 5 cols)
  3 : Average Equal Weighted Returns -- Annual (8 rows x 5 cols)
  4 : Number of Firms in Portfolios (96 rows x 5 cols)
  5 : Average Firm Size (96 rows x 5 cols)
  6 : Sum of BE / Sum of ME (8 rows x 5 cols)
  7 : Value-Weighted Average of BE/ME (8 rows x 5 cols)

In [51]: ds[4].head()
Out[51]:
            Cnsmr  Manuf  HiTec  Hlth   Other
Date
2010-01     622    737    830    467    1232
2010-02     620    734    821    464    1221
2010-03     614    729    818    458    1215
2010-04     614    726    807    458    1203
2010-05     611    723    804    457    1195
```

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]: mathces = wb.search('gdp.*capita.*const')
```

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 available 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 'TUD'.

```
In [52]: import pandas_datareader.data as web

In [53]: import datetime

In [54]: df = web.DataReader('TUD', 'oecd', end=datetime.datetime(2012, 1, 1))

In [55]: df.columns
Out[55]:
MultiIndex(levels=[['Australia', 'Austria', 'Belgium', 'Canada', 'Chile', 'Czech Republic',
'Denmark', 'Estonia', 'Finland', 'France', 'Germany', 'Greece', 'Hungary', 'Iceland', 'Ireland',
'Israel', 'Italy', 'Japan', 'Korea', 'Latvia', 'Lithuania', 'Luxembourg', 'Mexico', 'Netherlands',
'New Zealand', 'Norway', 'Poland', 'Portugal', 'Slovak Republic', 'Slovenia', 'Spain', 'Sweden',
'Switzerland', 'Turkey', 'United Kingdom', 'United States'], ['Annual'], ['Administrative data',
'Survey data'], ['Employees', 'Trade union  density', 'Union members'], ['Percentage', 'Thousands']],
           labels=[[13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18,
19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 19, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 24, 24, 24, 24,
24, 24, 24, 24, 24, 24, 24, 24, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 16, 16, 16, 16, 16,
16, 16, 16, 16, 16, 16, 16, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 28, 9, 9, 9, 9, 9, 9, 9, 9,
9, 9, 9, 9, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 22, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 35,
35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 35, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, 14, 14,
14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 23, 23, 23, 23, 23, 23, 23,
23, 23, 23, 23, 23, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 25, 29, 29, 29, 29,
29, 29, 29, 29, 29, 29, 29, 29, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 33, 33, 33, 33, 33, 33, 33, 33,
33, 33, 33, 33, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 27, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 30, 26,
26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 20, 20, 20, 20, 20,
20, 20, 20, 20, 20, 20, 20], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0], [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1,
```

```
1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0], [2, 2, 1, 1, 0, 0, 2,
2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0,
0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1,
1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2,
2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0,
0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1,
1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2,
2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0,
0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1,
1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2,
2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0,
0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1,
1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2,
2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0, 2, 2, 1, 1, 0, 0], [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
1, 0, 1, 0, 1, 0, 1, 0]],
           names=['Country', 'Frequency', 'Source', 'Series', 'Measure'])

In [56]: df[['Japan', 'United States']]
Out[56]:
Country            Japan                                                       \
Frequency         Annual
Source       Survey data
Series     Union members            Trade union  density            Employees
Measure        Thousands Percentage           Thousands Percentage Thousands
Year
2010-01-01           NaN        NaN                 NaN        NaN       NaN
2011-01-01           NaN        NaN                 NaN        NaN       NaN
2012-01-01           NaN        NaN                 NaN        NaN       NaN

Country                                                                    \
Frequency
Source                    Administrative data
Series                          Union members           Trade union  density
Measure    Percentage               Thousands Percentage            Thousands
Year
2010-01-01        NaN                 12417.5        NaN                  NaN
2011-01-01        NaN                 12271.9        NaN                  NaN
2012-01-01        NaN                 12227.1        NaN                  NaN

Country                  ...            United States                     \
```

```
Frequency                ...                  Annual
Source                   ...             Survey data
Series                   ...     Trade union  density            Employees
Measure    Percentage    ...               Thousands Percentage Thousands
Year                     ...
2010-01-01       28.9    ...                     NaN       17.4   97406.0
2011-01-01       27.6    ...                     NaN       16.5  102403.0
2012-01-01       25.9    ...                     NaN       15.9  106924.0

Country                                                                  \
Frequency
Source                Administrative data
Series                      Union members            Trade union  density
Measure    Percentage           Thousands Percentage             Thousands
Year
2010-01-01        NaN                 NaN        NaN                   NaN
2011-01-01        NaN                 NaN        NaN                   NaN
2012-01-01        NaN                 NaN        NaN                   NaN

Country
Frequency
Source
Series                Employees
Measure    Percentage Thousands Percentage
Year
2010-01-01        NaN   97406.0        NaN
2011-01-01        NaN  102403.0        NaN
2012-01-01        NaN  106924.0        NaN

[3 rows x 24 columns]
```

Eurostat

Eurostat are available via `DataReader`.

Get Rail accidents by type of accident (ERA data) 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 [57]: import pandas_datareader.data as web

In [58]: df = web.DataReader('tran_sf_railac', 'eurostat')

In [59]: df
Out[59]:
ACCIDENT    Collisions of trains, including collisions with obstacles within the clearance gauge  \
UNIT                                                                                      Number
GEO                                                                                      Austria
FREQ                                                                                      Annual
TIME_PERIOD
2010-01-01                                                                                    3.0
2011-01-01                                                                                    2.0
2012-01-01                                                                                    1.0
2013-01-01                                                                                    4.0
2014-01-01                                                                                    1.0
2015-01-01                                                                                    7.0
2016-01-01                                                                                    7.0

ACCIDENT                                                                  \
UNIT
GEO         Belgium Bulgaria Switzerland Channel Tunnel Czech Republic
FREQ         Annual   Annual      Annual          Annual          Annual
TIME_PERIOD
2010-01-01      5.0      2.0         5.0             0.0             3.0
2011-01-01      0.0      0.0         4.0             0.0             6.0
2012-01-01      3.0      3.0         4.0             0.0             6.0
2013-01-01      1.0      2.0         6.0             0.0             5.0
2014-01-01      3.0      4.0         0.0             0.0            13.0
2015-01-01      0.0      3.0         3.0             0.0            14.0
2016-01-01      2.0      3.0         2.0             0.0             6.0

ACCIDENT                                                                      \
UNIT
GEO         Germany (until 1990 former territory of the FRG) Denmark Estonia
FREQ                                                  Annual  Annual  Annual
TIME_PERIOD
2010-01-01                                              13.0     0.0     1.0
2011-01-01                                              18.0     1.0     0.0
2012-01-01                                              23.0     1.0     3.0
2013-01-01                                              29.0     0.0     0.0
2014-01-01                                              32.0     0.0     0.0
2015-01-01                                              40.0     3.0     0.0
2016-01-01                                              29.0     0.0     3.0

ACCIDENT               ...           Unknown                                  \
UNIT                   ...            Number
GEO         Greece     ...       Netherlands Norway Poland Portugal Romania
FREQ        Annual     ...            Annual Annual Annual   Annual  Annual
TIME_PERIOD            ...
2010-01-01     4.0     ...               NaN    NaN    NaN      NaN     NaN
2011-01-01     1.0     ...               NaN    NaN    NaN      NaN     NaN
2012-01-01     2.0     ...               NaN    NaN    NaN      NaN     NaN
```

```
2013-01-01     2.0      ...              NaN    NaN    NaN      NaN     NaN
2014-01-01     1.0      ...              NaN    NaN    NaN      NaN     NaN
2015-01-01     2.0      ...              NaN    NaN    NaN      NaN     NaN
2016-01-01     1.0      ...              NaN    NaN    NaN      NaN     NaN

ACCIDENT
UNIT
GEO         Sweden Slovenia Slovakia Turkey United Kingdom
FREQ        Annual   Annual   Annual Annual         Annual
TIME_PERIOD
2010-01-01     NaN      NaN      NaN    0.0            NaN
2011-01-01     NaN      NaN      NaN    0.0            NaN
2012-01-01     NaN      NaN      NaN    0.0            NaN
2013-01-01     NaN      NaN      NaN    0.0            NaN
2014-01-01     NaN      NaN      NaN    0.0            NaN
2015-01-01     NaN      NaN      NaN    0.0            NaN
2016-01-01     NaN      NaN      NaN    0.0            NaN

[7 rows x 264 columns]
```

TSP Fund Data

Download mutual fund index prices for the TSP.

```
In [60]: import pandas_datareader.tsp as tsp

In [61]: tspreader = tsp.TSPReader(start='2015-10-1', end='2015-12-31')

In [62]: tspreader.read()
Out[62]:
            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]
```

Nasdaq Trader Symbol Definitions

Download the latest symbols from Nasdaq.

Note that Nasdaq updates this file daily, and historical versions are not available. More information on the field 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
```

Stooq Index Data

Google finance doesn't provide common index data download. The Stooq site has the data for download.

```
In [63]: import pandas_datareader.data as web

In [64]: f = web.DataReader('^DJI', 'stooq')

In [65]: f[:10]
Out[65]:
                Open      High       Low     Close       Volume
Date
2018-02-01  26083.04  26306.70  26014.44  26186.71          NaN
2018-01-31  26268.17  26338.03  26050.98  26149.39  140120144.0
2018-01-30  26198.45  26256.99  26028.42  26076.89  111840144.0
2018-01-29  26584.28  26608.90  26435.34  26439.48  110919888.0
2018-01-26  26466.74  26616.71  26425.35  26616.71  123610888.0
2018-01-25  26313.06  26458.25  26259.72  26392.79   95732448.0
2018-01-24  26282.07  26392.80  26106.94  26252.12  123271104.0
2018-01-23  26214.87  26246.19  26143.90  26210.81  109272288.0
2018-01-22  26025.32  26215.23  25974.65  26214.60  126357768.0
2018-01-19  25987.35  26071.72  25942.83  26071.72  171541424.0
```

MOEX Data

The Moscow Exchange (MOEX) provides historical data.

```
In [66]: import pandas_datareader.data as web

In [67]: f = web.DataReader('USD000UTSTOM', 'moex', start='2017-07-01', end='2017-07-31')

In [68]: f.head()
Out[68]:
            BOARDID   SHORTNAME         SECID   OPEN     LOW     HIGH    CLOSE  \
TRADEDATE
2017-07-03    CETS  USDRUB_TOM  USD000UTSTOM  58.95  58.790  59.4825  59.2650
2017-07-04    CETS  USDRUB_TOM  USD000UTSTOM  59.30  59.135  59.4575  59.4125
2017-07-04    CNGD  USDRUB_TOM  USD000UTSTOM  59.36  58.930  59.3600  59.3575
2017-07-05    CETS  USDRUB_TOM  USD000UTSTOM  59.30  59.300  60.2600  59.9825
2017-07-05    CNGD  USDRUB_TOM  USD000UTSTOM  59.34  59.265  60.1800  60.1800

             NUMTRADES        VOLRUR  WAPRICE
TRADEDATE
2017-07-03       29108  1.472424e+11  59.1903
2017-07-04       21053  1.090265e+11  59.2700
2017-07-04          37  1.046416e+09      NaN
2017-07-05       50108  2.874226e+11  59.9234
2017-07-05          35  6.339036e+09      NaN
```