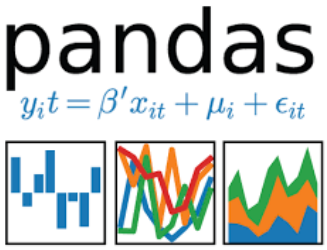


Data Sources for Equities and Factors

Dr Richard Diamond, PhD, CQF, ARPM

Agenda

1. Data Sources (US and International Equities, ETFs, Commodity Futures)

Remember that in addition to large-cap equities (eg, AMZN, GOOG) there are interesting industries (eg, Healthcare, Biotech) and other asset classes. You are more likely to find a sensible prediction scheme for commodity futures rather than a popular large cap.

2. Fixing Yahoo!Finance access

Yahoo!Finance was the easy choice for decates but began to limit the queries that are not coming from its website. You need a special session cookie. (NB You don't have to use Yahoo!Finance if you have more professional sources of equities/prices data).

_pandas_datareader_ and its versions. Packages installation in Python.

3. Fama-French Factors data retrieval and essential interpretation

In portfolio management 'Factor' is always a time series of returns from the special, long/short portfolio. Asset pricing research and practice rely on commonly agreed factors (5_Factors_2x3).

Because factor is a column of returns, we can regress on it to compute Factor Beta (exposure).

If you standardise a column of returns (that represent a Factor) and choose 99th percentile as your confidence, then the column of historical returns collapses into one Factor Value Z in $Z * \sigma = 2.33\sigma$.

4. Quick regression on market vs. factors

sklearn regression coeffients are found by optimisation.

statsmodels.api provides the regression output and computes using the extact analytical formulae

(**5. Statistical Plotting with Pandas: 3D Scatter, Lag Plots, Autocorrelation, Bootstrap Plot**)

This extra material is moving to separate content. Section given as is, without further development.

In [2]:
```
%matplotlib inline

import warnings
import numpy as np
import pandas as pd

import datetime as dt
```

Part 1. Data Sources: Equities, Indicies, ETFs, Commodity Futures

Depending on the version, pandas-datareader offered the following data sources:

- **Yahoo!Finance**. The functionality comes and gos but its is very important. Fixes (additional packages) might be required.

- **IEX API** is a semi-professional solution for Nasdaq/NYSE/AMEX tickers, within 5-year historic period. It supports websockets (to push data to you). Inlcuded since

pandas_datareader 0.8.0 and requires an API key.

- **Alpha Vantage** is part of pandas_datareader as of 2023-May. Historically, we used a dedicated package, more.
  `pip install alpha_vantage`

- **Quandl** has some free datasets, but reqistration for your own API key required.

- **Tiingo** provides historical end-of-day data for a large set of equities, ETFs and mutual funds. Free registration required for get an API key.

Those sources support different kinds of data, not always interoperable with one another particularly when it comes to the indexing. However, you should be able to handle the common datetime stamp index and conversion between time zones.

Deprecated

- Near real-time quotes were available from **Robinhood** but only up to 1-year period. Deprecated functionality. Downgrade to older version might not work due to change in Robinhood API.

- **Google Finance** was present in pandas-datareader v0.5.0, but deprecated since. Some indexes are **Stooq**

- **Quantopian** offered a strategy-testing platform -- there was need not need to download data for market index and factors (to compute the rolling beta against).

Equities/instruments data for the strategy testing has to be (a) cleane and (b) available for the longer periods. The main limitation, however, was the dynamic nature of Quantopian backtesting: you needed to set up calendar rules and strategy rules (allocation, trading) using Python-like coding language, which was derived from the package *zipline*.

Aside: Debugging in Python

```
conda install -c anaconda pandas-datareader #Preferred way for anaconda (neater)

pip install pandas-datareader  #Simple way

pip install git+https://github.com/pydata/pandas-datareader  #To install the newest dev version from Github
```

Python is an evolvling ecosystem. Sometimes, updates to the 'core packages' (eg, *pandas*) create problems for the off-spring packages (eg, *pandas-datareader*) and vice versa.

Please do not give up and check if a problem can be easily fixed because it relates to naming/changed headings. Be prepared to work out simple solutions yourself or look for workarounds online. Eg, Stackoverflow might have the answer.

Below is an example when development version (usually the newest) *pandas-datareader 0.6.0+30.g3c17058* pulled from Github did not work with the updated *pandas*. The solution was as follows:

Locate *fred.py* file inside /anaconda3/lib/python3.6/site-packages/pandas_datareader and replace the import call inside file.

```
from pandas.core.common import is_list_like #REMOVE OR COMMENT OUT

from pandas.api.types import is_list_like #INSERT
```

NOTE When debugging in Python, read the final error message.

1, 2

Yahoo!Finance: Issue and Solution

For a long time, common sources for the time series data (historic prices) were Yahoo!Finance and Google Finance. A history of disruption with Yahoo!Finance API began in 2017.

There were reports Yahoo! IP-blocks if a connection makes too many requests in rapid succession, but most likely the connection is rejected for all but web display purposes. Python script *yqd* listed below explains the authentication issue and retrieves the right cookie.

If we try the link quiery1.finance.yahoo.com without yfinance fix, we will get a message alongside the following:

```
"code": "Unauthorized",
"description": "Invalid cookie"
```

yfinance The package (2019 update) offers **one-stop solution** to downloading equities data. This is your first point of call in travails with Yahoo!Finance. It also shows an interesting overrride -- if you have extisting code that relies on pandas-datareader syntax:

```
pip install yfinance --upgrade --no-cache-dir
```

Yahoo-finance There is an older package *yahoo-finance* which explains the fields of data that were possible to pull from Yahoo!Finance.

IF YAHOO!FINANCE FIX DOES NOT WORK (MIGHT HAPPEN ANY TIME) -- USE ALTERNATIVE SOURCES SUCH AS QUANDL, IEX API

If you have own datasources, from work or other subscription it might be easier to use those.

In [75]:
```
import yfinance as yf
```

In [76]:
```
prices1 = yf.download("^GSPC ^VIX ^FTSE", start="2010-10-01", end="2017-09-30")

prices1.to_excel('data/EquitiesDataIndicies_x3.xlsx') # SAVE DATA INTO LOCAL FI
```

```
[**********************100%**********************]  3 of 3 downloaded
```

In [40]:
```
prices1.head(10)
```

Out[40]:

	Adj Close			Close			High				
	^FTSE	^GSPC	^VIX	^FTSE	^GSPC	^VIX	^FTSE	^GSPC	^VIX	^FTSE	^GSPC
Date											
2010-09-30	NaN	1141.20	23.70	NaN	1141.20	23.70	NaN	1157.16	24.52	NaN	1136.08
2010-10-01	5592.9	1146.24	22.50	5592.9	1146.24	22.50	5615.1	1150.30	23.67	5547.6	1139.42
2010-10-04	5556.0	1137.03	23.53	5556.0	1137.03	23.53	5601.2	1148.16	24.34	5550.8	1131.87
2010-10-05	5635.8	1160.75	21.76	5635.8	1160.75	21.76	5646.1	1162.76	23.08	5550.6	1140.68
2010-10-06	5681.4	1159.97	21.49	5681.4	1159.97	21.49	5695.5	1162.33	22.13	5635.8	1154.85
2010-10-07	5662.1	1158.06	21.56	5662.1	1158.06	21.56	5707.3	1163.87	22.16	5650.8	1151.41
2010-10-08	5657.6	1165.15	20.71	5657.6	1165.15	20.71	5663.7	1167.73	21.64	5606.6	1155.58
2010-10-11	5672.4	1165.32	18.96	5672.4	1165.32	18.96	5686.0	1168.68	19.51	5655.7	1162.02
2010-10-12	5661.6	1169.77	18.93	5661.6	1169.77	18.93	5677.0	1172.58	20.10	5597.5	1155.71
2010-10-13	5747.4	1178.10	19.07	5747.4	1178.10	19.07	5760.5	1184.38	19.16	5661.6	1171.32

In [ ]:
```
# Extra download of GOOG prices -- for ML "market direction prediction"
```

```
prices2 = yf.download("GOOG", start="2010-10-01", end="2017-09-30")
prices2.to_excel('data/EquitiesDataGOOG.xlsx')
```

In [ ]:

Pandas_Datareader (alternative)

This is a convenient library because it encompasses many data sources, not just Yahoo!Finance.

```
conda uninstall pandas-datareader
```

```
conda install -c anaconda pandas-datareader==0.5.0
```

WARNING WARNING installing this older version of *pandas-datareader* package will trigger DOWNGRADE of other packages, particularly *pandas* in your working environment (most likely 'anaconda'). This older version of *pandas* likely TO CONFLICT with your curent/updated *matplotlib* -- and so simply conda install/uninstall matplotlib will not work. **SO** after you download the data you might want to REUPDATE *pandas*.

- Use HOMEBREW to manage your packages and different working environments.
- Python Environments simple management with *conda* and Anaconda installation also available.

After you run yfinance, you can load price data as usual using 0.5.0 pandas-datareader.

Optionaly you can override the pointer to pandas_datareader.

```
from pandas_datareader import data as pdr

yf.pdr_override() # <== overrides the method from pandas-datareader
OPTIONAL
data = pdr.get_data_yahoo("SPY", start="2017-01-01", end="2017-04-
30") # download dataframe
```

However, your older code based the datareader should run just as well. Below is code example,

```
from pandas_datareader import data, wb

data.DataReader(ticker, data_source='yahoo', start=start, end=end)

# routine to be used in the loop -- to retrive several tickers'
prices
def get_ticker(ticker, start, end):

    ticker = data.DataReader(ticker, data_source='yahoo',
start=start, end=end)
```

```
    return ticker
```

In [68]:
```
from pandas_datareader import data as pdr

df_temp = pdr.get_data_yahoo("SPY", start="2017-01-01", end="2017-04-30")
```

```
[**********************100%**********************]  1 of 1 downloaded
```

In [69]:
```
df_temp.head(10)
```

Out[69]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	225.04	225.83	223.88	225.24	213.84	91366500
2017-01-04	225.62	226.75	225.61	226.58	215.11	78744400
2017-01-05	226.27	226.58	225.48	226.40	214.94	78379000
2017-01-06	226.53	227.75	225.90	227.21	215.71	71559900
2017-01-09	226.91	227.07	226.42	226.46	215.00	46939700
2017-01-10	226.48	227.45	226.01	226.46	215.00	63771900
2017-01-11	226.36	227.10	225.59	227.10	215.61	74650000
2017-01-12	226.50	226.75	224.96	226.53	215.07	72113200
2017-01-13	226.73	227.40	226.69	227.05	215.56	62717900
2017-01-17	226.31	226.78	225.80	226.25	214.80	61240800

SUMMARY

1. AFTER you have run the lines below from yfinance -- cookies and authorisation for Yahoo!Finance will be working temporarily -- so you can run get_ticker() routine (and any existing code you might have) to obtain the data from *pandas-datareader*. Result!

1. RETRIEVE the data for each equity/index individually. Example below shows mismatch -- FTSE index value gets shifted down on every 5-7 day point. This is easy to spot when reviewing the full dataframe in Excel *before* proceeding to run statistical and machine learning analysis on it.

1. Good habit will be TO DOWNLOAD the data localy and next time read from the file. Use **pandas.DataFrame.to_csv** or **to_excel** methods to store the data.

1. Be mindful about timestamp/datestamp of your imported data and if Python import recognised any frequency to it. Common issues are: POSIX format vs. other formats, daily prices should have particular timestamp (UTC midnight).

Aside: Scrapping

In the current environment, as data providers close up and other sources exercises limits on downloads, one might need to master a flexible web data scrapping tool.

Scrapping Yahoo!Finance 2019 Examples at this link are excellent for scrapping 'today' data snapshots.

Yahoo quote download reveal that Yahoo!Finance still retrieves the data for own pages via a query -- with a bit of authentication from a "crumb" of a cookie "B". Python script at this link retrieves the matching cookie and crumb. The following package can also be used to get prices data:

```
import yqd
yf_data = yqd.load_yahoo_quote('AAPL', '20170722', '20170725')
```

Financial statements data from Yahoo!Finance such as balance sheets and income statement can also be obtained and the link provides very goood way of doing it (code snippets).

yahoofinancials a powerful script to get price and fundamentals data (version 01/27/2019) and save it in organised JSON format, written by someone for their semi-professional purpose.

Retrieving SEVERAL Tickers

In [77]:
```
from pandas_datareader import data, wb

def get_ticker(ticker, start, end):

    ticker = data.DataReader(ticker, data_source='yahoo', start=start, end=end)

    return ticker

# Futures Data: remember that ticker changes at Expiration/Rollover  https://fi
#print(ticker[0]) for ticker in tickers
#for ticker in tickers: print(ticker[1])
```

In [78]:
```
tickers = [['^GSPC', 'SP500'], ['^VIX', 'VIX'], ['^FTSE', 'FTSE100']]

prices = pd.DataFrame({ ticker[1] : get_ticker(ticker[0], '01-10-2011', '30-09-
#prices.DataFrame.to_csv  TO SAVE INTO FILE
```

In [79]:
```
prices.head(10)
```

Out[79]:

Date	SP500	VIX	FTSE100
2011-01-10	1269.750000	17.540001	5956.299805
2011-01-11	1274.479980	16.889999	6014.000000
2011-01-12	1285.959961	16.240000	6050.700195
2011-01-13	1283.760010	16.389999	6023.899902
2011-01-14	1293.239990	15.460000	6002.100098
2011-01-18	1295.020020	15.870000	6056.399902
2011-01-19	1281.920044	17.309999	5976.700195
2011-01-20	1280.260010	17.990000	5867.899902
2011-01-21	1283.349976	18.469999	5896.299805
2011-01-24	1290.839966	17.650000	5943.899902

In [80]:
```
prices.index
```

Out[80]:
```
DatetimeIndex(['2011-01-10', '2011-01-11', '2011-01-12', '2011-01-13',
               '2011-01-14', '2011-01-18', '2011-01-19', '2011-01-20',
               '2011-01-21', '2011-01-24',
               ...
               '2019-09-17', '2019-09-18', '2019-09-19', '2019-09-20',
               '2019-09-23', '2019-09-24', '2019-09-25', '2019-09-26',
               '2019-09-27', '2019-09-30'],
              dtype='datetime64[ns]', name='Date', length=2157, freq=None)
```

In [81]:
```
start = prices.index.min()
end = prices.index.max()
monthly_dates = pd.date_range(start, end, freq='M') #Create a new index of mont

monthly = prices.reindex(monthly_dates, method='ffill') #Re-indexing


returns = 100 * monthly.pct_change().dropna()
#This can be adjusted into excess returns by subtracting risk-free rate from Fa
returns.head()
```

Out[81]:

	SP500	VIX	FTSE100
2011-02-28	3.195656	-6.041991	2.236096
2011-03-31	-0.104731	-3.324251	-1.421425
2011-04-30	2.613459	-17.587373	2.726444
2011-05-31	-1.123135	5.677155	-1.316330
2011-06-30	-1.825746	6.925566	-0.739563

In [82]:
```
tickers = [['AAPL', 'AAPL']]

prices3 = pd.DataFrame({ ticker[1] : get_ticker(ticker[0], '01-10-2010', '30-09

prices3.to_excel('data/EquitiesDataAPPL.xlsx') #prices.DataFrame.to_csv  TO SAV
```

In [48]:
```
prices3.head(10)
```

Out[48]:

Date	AAPL
2010-01-11	26.195114
2010-01-12	25.897146
2010-01-13	26.262434
2010-01-14	26.110340
2010-01-15	25.673977
2010-01-19	26.809748
2010-01-20	26.397085
2010-01-21	25.940775
2010-01-22	24.654148
2010-01-25	25.317411

In [21]:
```
prices3.pct_change().head(10)
```

Out[21]:

Date	AAPL
2010-01-11	NaN
2010-01-12	-0.011375
2010-01-13	0.014105
2010-01-14	-0.005791
2010-01-15	-0.016712
2010-01-19	0.044238
2010-01-20	-0.015392
2010-01-21	-0.017286
2010-01-22	-0.049599
2010-01-25	0.026903

Part 2. Fama-French Factors Data Retrieval

1. FACTOR IS A TIME SERIES OF RETURNS

2. RETURNS ARE FROM THE SPECIFIC LONG/SHORT STRATEGY

"famafrench" dataset has 262 sets of numerious portfolio combinations, from factor portfolios to regional variations (North America/Europe/Japan/Asia Pacific. The main dataset names to backtest against (rolling beta calculation) are as follows:

- 'F-F_Research_Data_Factors' -- monthly data is **the default choice** for the Fama-French dataset
- 'F-F_Research_Data_Factors_weekly'
- 'F-F_Research_Data_Factors_daily'

Three Factors (Classic Model)

1) Mkt-RF are the market index (S&P500) excess returns.

2) HML (VALUE FACTOR) is the difference between the returns on diversified portfolios of high and low B/M ratio stocks -- the lower value of M itself means higher Book-to-Market ratio and therefore, higher expectation of future return.

3) SMB is the difference between returns on diversified portfolios of small-sized equities minus big-sized (large cap) equties.

Five Factors (Updated Model)

- 'F-F_Research_Data_5_Factors_2x3',
- 'F-F_Research_Data_5_Factors_2x3_daily'

4) RMW is the difference between the returns on diversified portfolios of equities with Robust vs. Weak profitability

5) CMA is the difference between returns on diversified portfolios of low-investment equities (Conservative) minus high-investment equities (Aggressive).

The range of portfolios, for which factors are backtested, is better explained at source http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html

Momentum Factor

There is no formal factor to reflect momentum phenomenon in the research framework and its backtesting. That was a major drawback (rendering Fama-French factors less useable for risk management), and since that **UMD** factor was developed and can be downloaded directly from http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/F-F_Momentum_Factor_daily_CSV.zip

References:

A Five-Factor Asset Pricing Model Fama & French (2014)

Quantopian Risk Model Whitepaper (2018) https://www.quantopian.com/posts/risk-model-white-paper-released-and-available-for-your-reading

In [83]:
```
from pandas_datareader import data #for implementation of new direct calls ==>

from pandas_datareader.famafrench import get_available_datasets
```

In [84]:
```
#Loading Factor Data
FFDataTest = data.DataReader("F-F_Research_Data_5_Factors_2x3_daily", "famafrer
FFDataTest.index = pd.to_datetime(FFDataTest.index, format="%Y%m%d", utc=True)
FFDataTest.head()
```

Out[84]:

Date	Mkt-RF	SMB	HML	RMW	CMA	RF
2010-01-04	1.69	0.75	1.12	-0.25	0.21	0.0
2010-01-05	0.31	-0.37	1.21	-0.10	0.17	0.0
2010-01-06	0.13	-0.16	0.52	-0.03	0.19	0.0
2010-01-07	0.40	0.24	0.94	-0.63	0.23	0.0
2010-01-08	0.33	0.34	0.01	0.25	-0.37	0.0

In [85]:
```
#Loading Factor Data
FF_3Factor = data.DataReader("F-F_Research_Data_Factors_daily", "famafrench")[0
FF_3Factor.index = pd.to_datetime(FF_3Factor.index, format="%Y%m%d", utc=True)
```

In [86]:
```
FF_3Factor.head()
```

Out[86]:

Date	Mkt-RF	SMB	HML	RF
2010-01-04	1.69	0.58	1.12	0.0
2010-01-05	0.31	-0.59	1.21	0.0
2010-01-06	0.13	-0.24	0.52	0.0
2010-01-07	0.40	0.09	0.94	0.0
2010-01-08	0.33	0.40	0.01	0.0

Industry Portfolios

These are not factors because they are not constructed as long/short strategies.

In [25]:
```
FFdata = data.DataReader("5_Industry_Portfolios", "famafrench")
print(FFdata['DESCR'])
```

```
5 Industry Portfolios
---------------------

This file was created by CMPT_IND_RETS using the 201803 CRSP database. It cont
ains value- and equal-weighted returns for 5 industry portfolios. The portfoli
os 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 2018 Kenneth
R. French

  0 : Average Value Weighted Returns -- Monthly (99 rows x 5 cols)
  1 : Average Equal Weighted Returns -- Monthly (99 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 (99 rows x 5 cols)
  5 : Average Firm Size (99 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 [10]:
```
FFdata[1].head()
```

Out[10]:

Date	Cnsmr	Manuf	HiTec	Hlth	Other
2010-01	-0.77	-2.56	-3.16	0.07	2.05
2010-02	5.76	4.35	5.02	1.97	2.46
2010-03	10.01	6.92	8.46	8.30	7.14
2010-04	7.31	8.10	5.91	7.05	9.22
2010-05	-7.37	-9.31	-7.01	-9.05	-8.24

PART 3 Markets and Factors

Back to exploration of relationship between the market (indicies) and the factors.

In [87]:
```
#One more example ['Mkt-RF', 'SMB', 'HML', 'RF']
FFdata_Classic = pd.DataFrame(data.DataReader("F-F_Research_Data_Factors", "fam
FFdata_Classic.head()

#ff.columns = ['Mkt_rf', 'SMB', 'HML', 'rf']
#ff.index = [dt.datetime(d/100, d%100, 1) for d in ff.index] #issue running thi
```

Out[87]:

Date	Mkt-RF	SMB	HML	RF
2010-01	-3.36	0.38	0.30	0.00
2010-02	3.40	1.21	3.16	0.00
2010-03	6.31	1.43	2.11	0.01
2010-04	2.00	4.97	2.81	0.01
2010-05	-7.89	0.05	-2.38	0.01

In [88]:
```
# here was a code to append the data to factors data (into one dataframe)
```

Below we prepare market returns data to be compared to factors.

ASIDE: Dataframe Index

Code below explores the type of index for our downloaded datasets. We need to ensure the index is compatible before merging two dataframes -- usually for a purpose of running a regression.

It makes sense to keep track of relationship and factor beta on monthly rather than daily basis.

pd.concat merges two dataframes, granted they have the same index. Otherwise it appends the second dataframe TO THE BOTTOM -- not the outcome we look for.

freq attribute None, because the index is UTC date+time stamp rather than day of month.

to_period('M') explained in https://stackoverflow.com/questions/23840797/convert-a-column-of-timestamps-into-periods-in-pandas

to_timestamp explained in https://stackoverflow.com/questions/29394730/converting-periodindex-to-datetimeindex

In [89]:
```
returns.head(10)
```

Out[89]:

	SP500	VIX	FTSE100
2011-02-28	3.195656	-6.041991	2.236096
2011-03-31	-0.104731	-3.324251	-1.421425
2011-04-30	2.613459	-17.587373	2.726444
2011-05-31	-1.123135	5.677155	-1.316330
2011-06-30	-1.825746	6.925566	-0.739563
2011-07-31	-2.147443	52.845036	-2.194863
2011-08-31	-5.679111	25.227727	-7.234492
2011-09-30	-7.176199	35.863370	-4.930948
2011-10-31	10.772304	-30.260708	8.105688
2011-11-30	-0.505872	-7.209613	-0.699836

In [90]:
```
returns.index # DatetimeIndex(dtype='datetime64[ns]', freq='M')
```

Out[90]:
```
DatetimeIndex(['2011-02-28', '2011-03-31', '2011-04-30', '2011-05-31',
               '2011-06-30', '2011-07-31', '2011-08-31', '2011-09-30',
               '2011-10-31', '2011-11-30',
               ...
               '2018-12-31', '2019-01-31', '2019-02-28', '2019-03-31',
               '2019-04-30', '2019-05-31', '2019-06-30', '2019-07-31',
               '2019-08-31', '2019-09-30'],
              dtype='datetime64[ns]', length=104, freq='M')
```

In [91]:
```
FFdata_Classic.head(10)
```

Out[91]:

Date	Mkt-RF	SMB	HML	RF
2010-01	-3.36	0.38	0.30	0.00
2010-02	3.40	1.21	3.16	0.00
2010-03	6.31	1.43	2.11	0.01
2010-04	2.00	4.97	2.81	0.01
2010-05	-7.89	0.05	-2.38	0.01
2010-06	-5.56	-1.97	-4.50	0.01
2010-07	6.93	0.17	-0.26	0.01
2010-08	-4.77	-3.00	-1.95	0.01
2010-09	9.54	3.92	-3.13	0.01
2010-10	3.88	1.13	-2.60	0.01

In [92]:
```
FFdata_Classic.index # PeriodIndex(dtype='period[M]', name='Date', freq='M')
```

Out[92]:
```
PeriodIndex(['2010-01', '2010-02', '2010-03', '2010-04', '2010-05', '2010-06',
             '2010-07', '2010-08', '2010-09', '2010-10',
             ...
             '2018-11', '2018-12', '2019-01', '2019-02', '2019-03', '2019-04',
             '2019-05', '2019-06', '2019-07', '2019-08'],
            dtype='period[M]', name='Date', length=116, freq='M')
```

pd.concat()

`FFdata_Classic.to_timestamp()`

will create index with FIRST DAY of the month. Notice that our returns dataframe has index as LAST DAY of month. Therefore, *pd.concat()* will not work after the line above. Instead we have to fix the index type for returns with _to*period('M')* first.

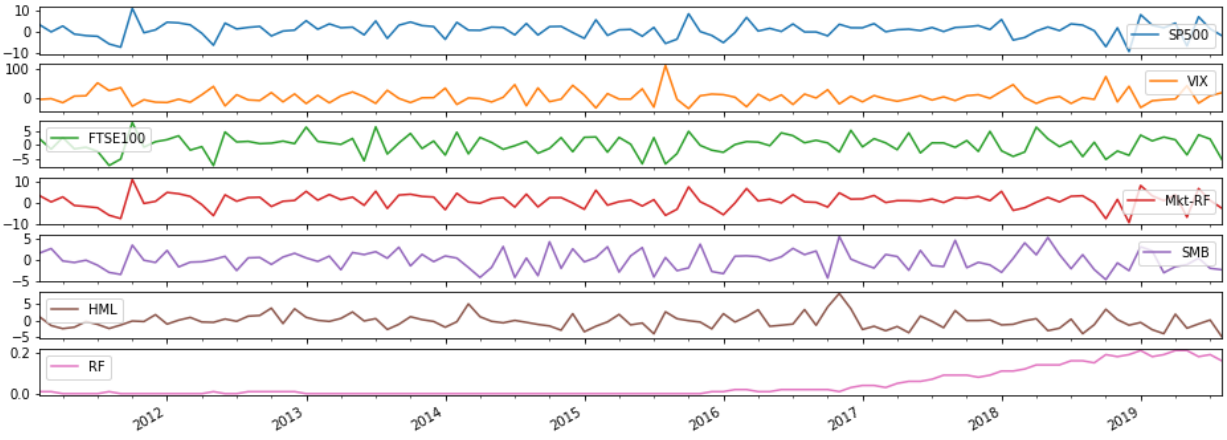
In [93]:
```
returnsM = returns.to_period('M') #converting to PeriodIndex

datamain = pd.concat([returnsM, FFdata_Classic], axis=1).dropna()
datamain = datamain.to_timestamp() #converting back to DatetimeIndex for regres

datamain.head()
```

Out[93]:

	SP500	VIX	FTSE100	Mkt-RF	SMB	HML	RF
2011-02-01	3.195656	-6.041991	2.236096	3.49	1.53	1.10	0.01
2011-03-01	-0.104731	-3.324251	-1.421425	0.45	2.60	-1.58	0.01
2011-04-01	2.613459	-17.587373	2.726444	2.90	-0.34	-2.52	0.00
2011-05-01	-1.123135	5.677155	-1.316330	-1.27	-0.70	-2.08	0.00
2011-06-01	-1.825746	6.925566	-0.739563	-1.75	-0.16	-0.32	0.00

In [94]:
```
datamain.plot(figsize=(16,6), subplots=True)
```

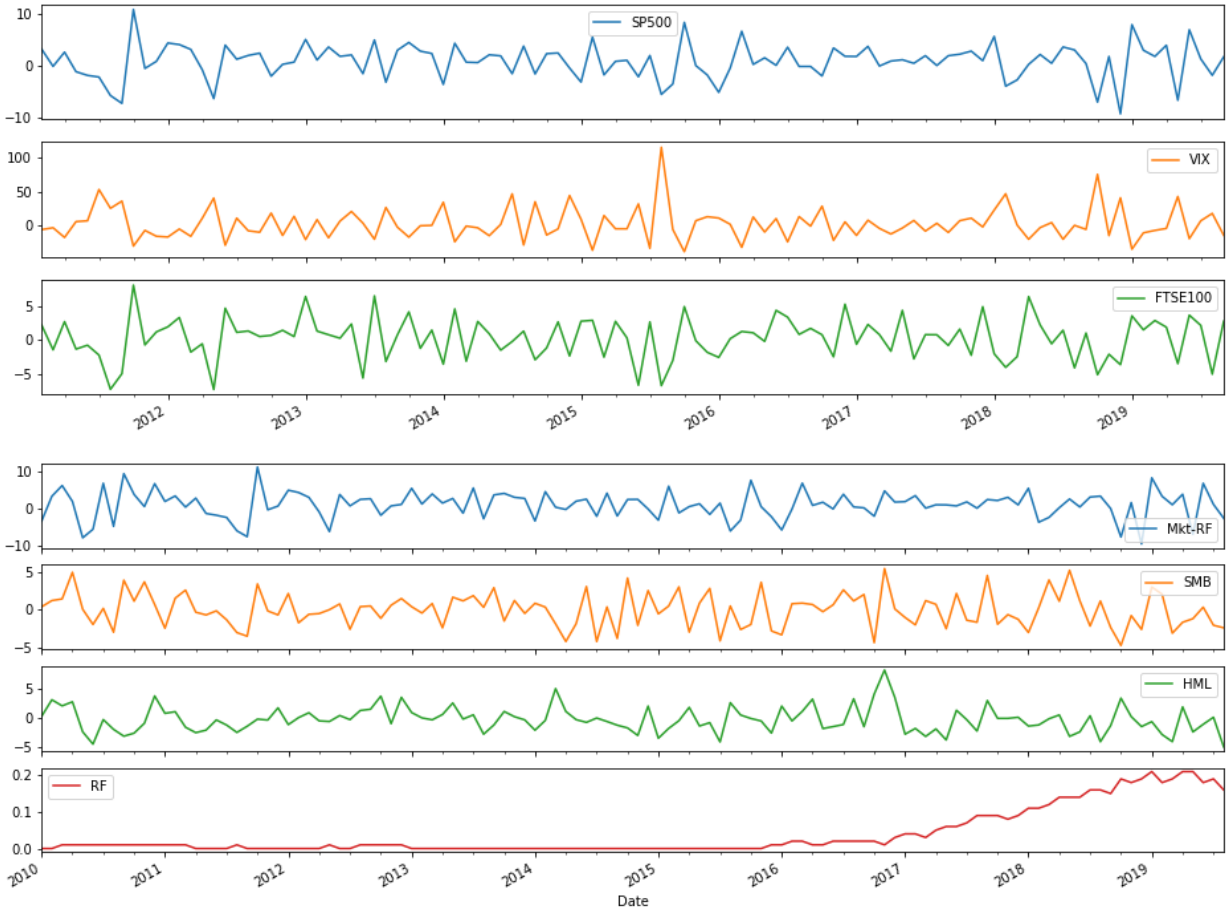
Out[94]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fc390abacc0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3807d9b70>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a2c7d860>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a2c28e48>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc390c9eb00>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a2c7a320>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a307d780>],
      dtype=object)
```



CAUTION Different start points of data, if plotted separately

In [95]:
```
returns.plot(figsize=(16,6), subplots=True)  #'01-10-2011', '30-09-2019'
FFdata_Classic.plot(figsize=(16,6), subplots=True) # 01-01-2010
```

Out[95]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fc390eb74a8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc380ba0080>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a331a748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fc390eef160>],
      dtype=object)
```



EXERCISE: Define position weights (allocations) -- this is your strategy. For trading the mean-reversion, strategy weights can be obtained by applying Cointegration Analysis (for two variables, this is known as ECM model and Engle-Granger procedure)

Quick Regression for Factor Backtesting

There are two choices of packages to run statistical routines in Python (on top of *numpy* core package):

- *sklearn* linear model will be useful for our Machine Learning endeavours, particularly running of the classifiers.

Its LinearRegression implementation though done as a classifier and beta coefficients are found by optimisation, not by exact OLS formulae that stem from analytical optimisation (eg, derivatives equated to zero) of the joint Normal pdf of regression residuals.

- *statsmodels* is a routine choice for running statistics in Python, It is less friendly, and documentation is dry style. There are problems with its *ts.coint()* routine. *statsmodels* have no features to facilitate **Rolling Estimation** of betas or do crossvalidation-type estimation with various sample selection from the dataset.

In [96]:
```
from sklearn import linear_model
```

In [97]:
```
Y_Strategy = datamain.loc[:,['SP500']]
X_Factors  = datamain.loc[:,['Mkt-RF', 'SMB', 'HML']] #df2.loc[startrow:endrow,

# Create linear regression object
OLS = linear_model.LinearRegression(fit_intercept=True)

# Train the model using the training sets
OLS.fit(X_Factors, Y_Strategy)
```

Out[97]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Below are our **Factor Betas** (theory explained in Project Workshop)

In [98]:
```
# Regression coefficients are our betas wrt factors
print('Coefficients: \n', OLS.coef_)
```

```
Coefficients:
 [[ 0.97815059 -0.1304103  -0.01976672]]
```

In [99]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [100…
```
X_Factors = sm.add_constant(X_Factors)

backtest_OLS = sm.OLS(Y_Strategy, X_Factors).fit()
print(backtest_OLS.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  SP500   R-squared:                       0.995
Model:                            OLS   Adj. R-squared:                  0.995
Method:                 Least Squares   F-statistic:                     6954.
Date:                Sat, 26 Oct 2019   Prob (F-statistic):          5.99e-115
Time:                        05:30:14   Log-Likelihood:                 4.1913
No. Observations:                 103   AIC:                            -0.3826
Df Residuals:                      99   BIC:                             10.16
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1230      0.025     -4.970      0.000      -0.172      -0.074
Mkt-RF         0.9782      0.007    138.542      0.000       0.964       0.992
SMB           -0.1304      0.011    -11.710      0.000      -0.153      -0.108
HML           -0.0198      0.011     -1.818      0.072      -0.041       0.002
==============================================================================
Omnibus:                        1.855   Durbin-Watson:                   2.168
Prob(Omnibus):                  0.396   Jarque-Bera (JB):                1.464
Skew:                           0.286   Prob(JB):                        0.481
Kurtosis:                       3.114   Cond. No.                         4.06
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
```

```
/anaconda3/lib/python3.6/site-packages/numpy/core/fromnumeric.py:52: FutureWar
ning: Method .ptp is deprecated and will be removed in a future version. Use n
umpy.ptp instead.
  return getattr(obj, method)(*args, **kwds)
```

EXERCISE: Rolling Parameter Estimation

1) The proper backtesting would require estimation of betas *wrt* factors on the rolling basis, rather than producing one-off beta figures.

2) Altegratively, one can utilise train_test_split functionality and crossvalidate the betas.

REMINDER The advantage of integrated Quantopian backtesting is that you would not need to reinvent computation of rolling beta, SR (and the factor data is held by Quantopian library for you).

Limitations (1) you will be within Quantopian Risk Model as set in their whitepaper amd (2) the need to learn their package and `backtesting command language'.

Naturally, with the full-scale industry implementation you will have own data feeds and own systems. Quantopian library is a mid-way solution.

Part 3. Pandas Plotting Capabilities

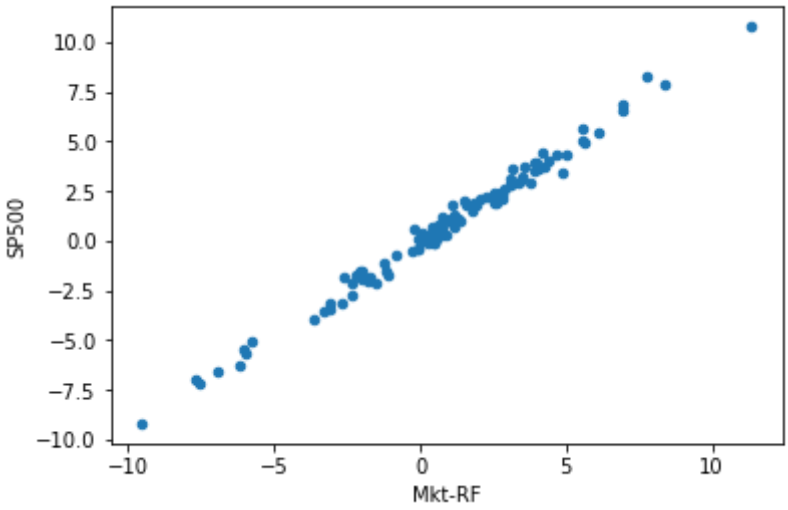
2D and 3D Scatterplots

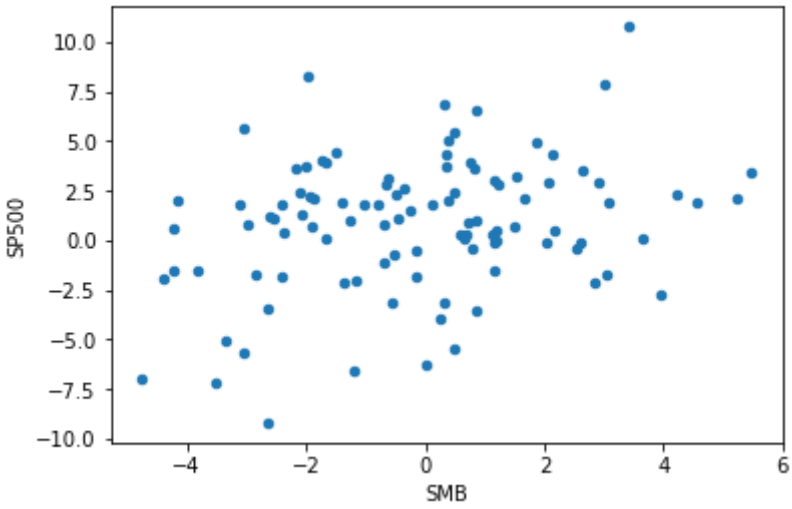
Scatterplots between returns give idea of correlation. An attempt of 3D plot between factor time series gives spatial idea of the relationships between factors themselves. If only we could rotate!

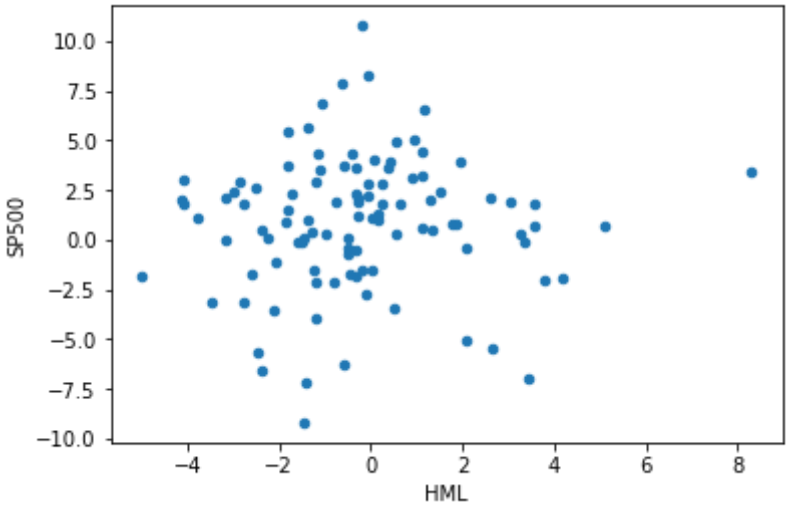
Explanation: since our implied strategy was to invest 100% in S&P500 (from *Yahoo!Finance*) then its logical to observe neat correlation with the Market Factor which is S&P500 excess returns (from *Fama-French dataset*).

In [101…

```
#Scatterplots directly from dataframe objects
#http://pandas.pydata.org/pandas-docs/version/0.15.0/visualization.html#scatter
datamain.plot(kind='scatter', x='Mkt-RF', y='SP500');

datamain.plot(kind='scatter', x='SMB', y='SP500');

datamain.plot(kind='scatter', x='HML', y='SP500');
```

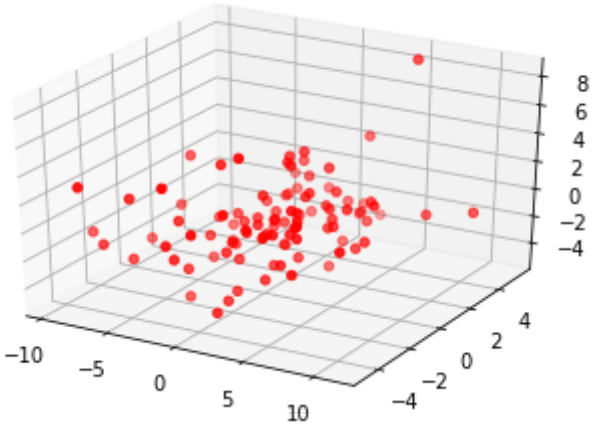






In [102…

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

figure = plt.figure()
ax=figure.add_subplot(111, projection='3d')
ax.scatter(X_Factors['Mkt-RF'], X_Factors['SMB'], X_Factors['HML'], c='r', mark
plt.show()
```



Statistical Analysis via plots: Lag Plots, Autocorrelation, Bootstrap Plots

When plots are not simply plots but long-established tools of statical analysis. For instance, **lag plots** are simple idea of plotting return at time t vs $t-1$, the more cirle-shaped the cloud is, the less autocorrelation and more 'Normality' there is to such data.

Pandas plotting offers interesting features that parallel Matlab,

- matrix of scatterplots (for correlation/copula exploration)
- Andrews curves
- lag plot to check for *iid*-ness
- autocorrelation plot
- bootstrap plots are used to visually assess the uncertainty of a statistic, such as mean

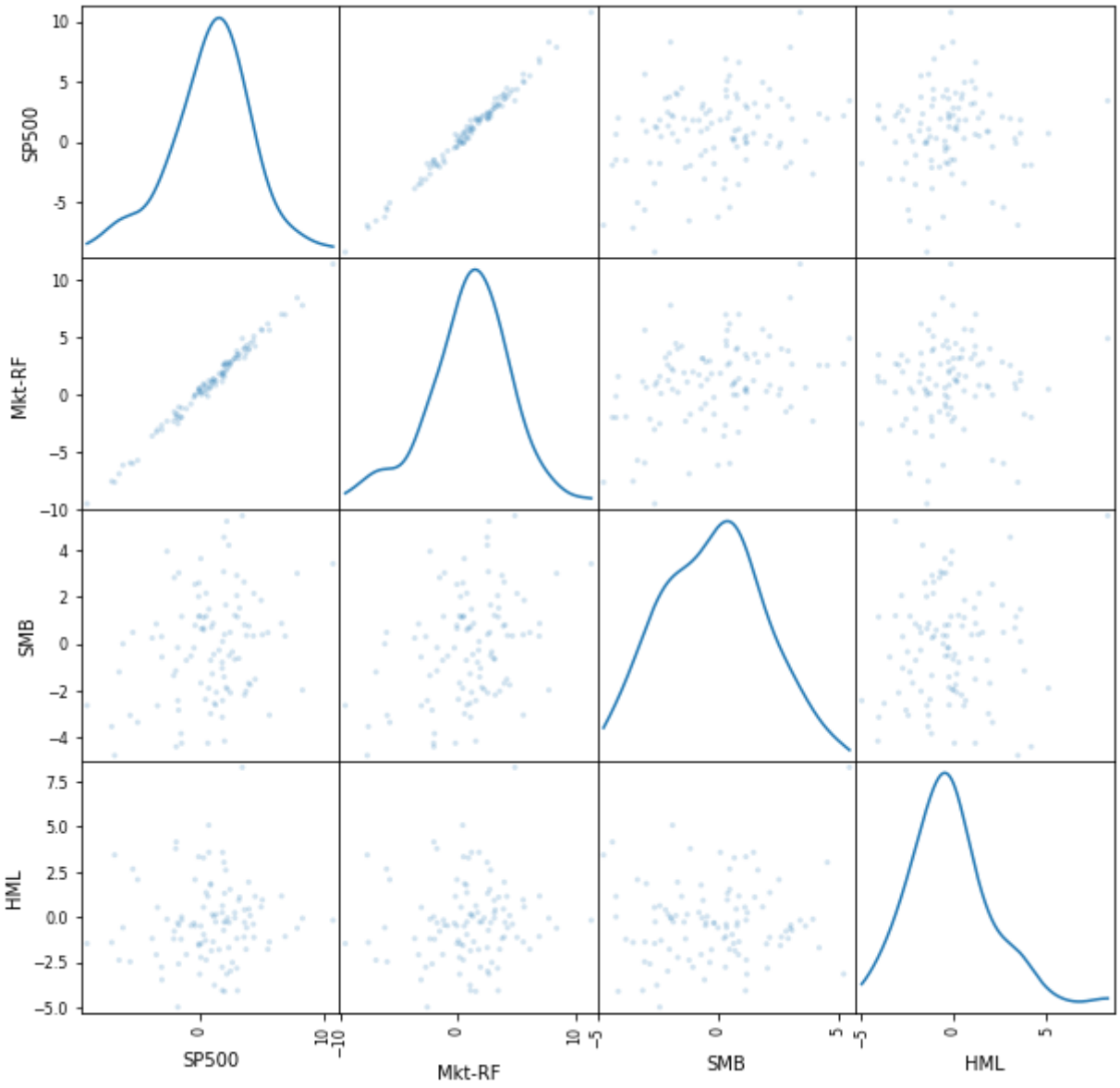
All you need to know about plotting with pandas is available at

MORE EXAMPLES: http://pandas.pydata.org/pandas-docs/version/0.21.0/visualization.html and it appears Pandas implements Graphical Techniques from *Engineering Statistics Textbook* published by NIST.

CODING IN R: in R with similar functionality from *ggplot2* library.
https://opendatascience.com/blog/data-visualization-part-2/

In [105…
```
from pandas.plotting import scatter_matrix

scatter_matrix(datamain.loc[:,['SP500', 'Mkt-RF', 'SMB', 'HML']], alpha=0.2, fi
```

Out[105]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b12972e8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b10f55f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b111ea90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b114e048>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3914b45c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a42dbb00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b104a3c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b109da90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b109db38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a4202f98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a33443c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc380c49320>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3b0a1a278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc390e22ba8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a41df198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3a43094a8>]],
      dtype=object)
```



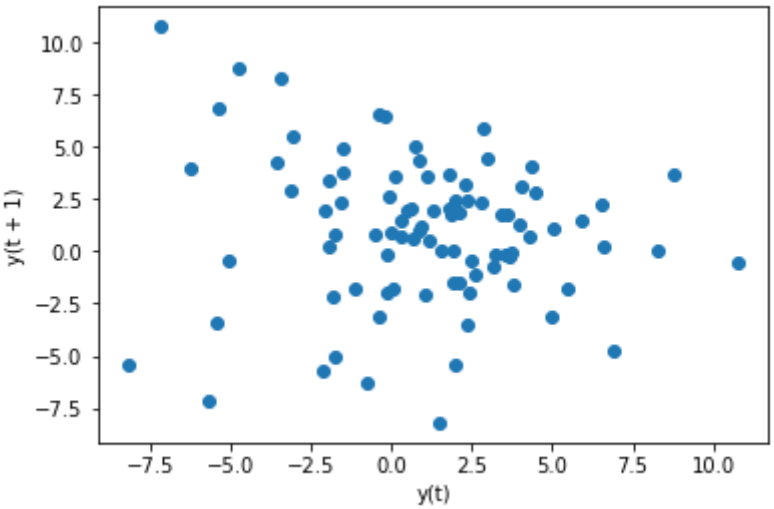
When interpreting autocorrelation, remember that we used MONTHLY returns to be compatible with Fama-French monthly frequency.

Lag Plot

In [19]:
```
from pandas.tools.plotting import lag_plot

lag_plot(datamain.loc[:,['SP500']])
```

```
/Users/diamond/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3:
FutureWarning: 'pandas.tools.plotting.lag_plot' is deprecated, import 'pandas.
plotting.lag_plot' instead.
  This is separate from the ipykernel package so we can avoid doing imports un
til
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10f5dff28>
```

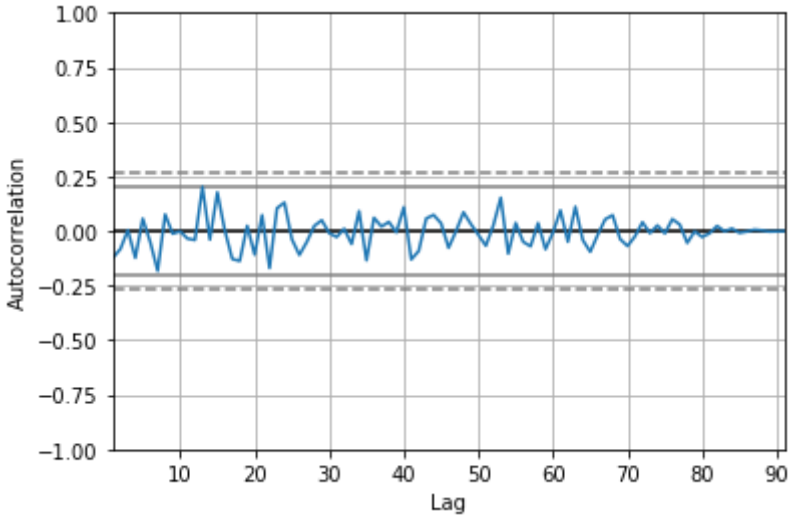


Autocorrelation Plot - ACF

In [21]:
```
from pandas.tools.plotting import autocorrelation_plot

autocorrelation_plot(datamain.loc[:,['SP500']])
autocorrelation_plot(datamain.loc[:,['HML']])
autocorrelation_plot(datamain.loc[:,['SMB']])
```

```
/Users/diamond/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3:
FutureWarning: 'pandas.tools.plotting.autocorrelation_plot' is deprecated, imp
ort 'pandas.plotting.autocorrelation_plot' instead.
  This is separate from the ipykernel package so we can avoid doing imports un
til
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a21e4b080>
```

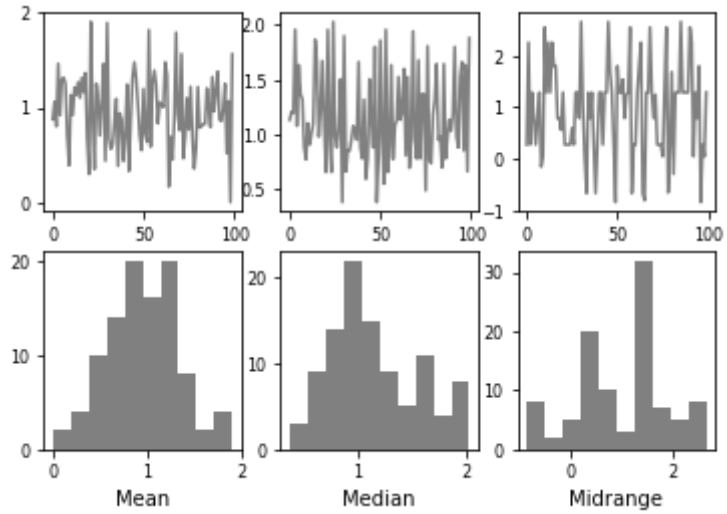


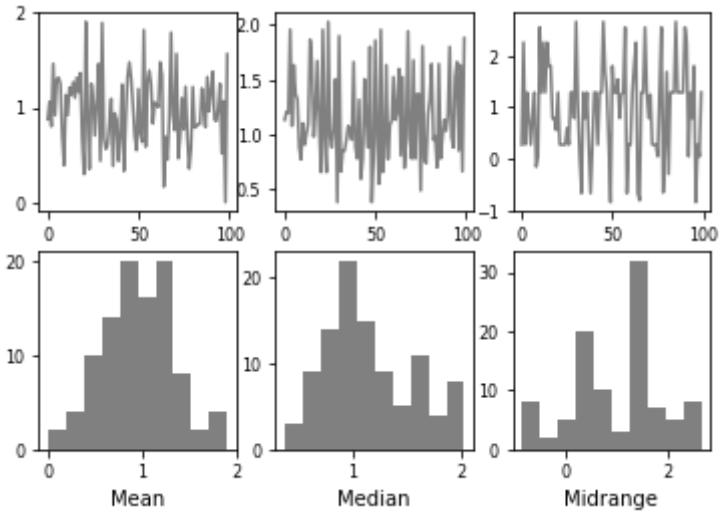
In [37]:
```
# Time series datasets -- indexed by dates, there are no name labels for observ
# One can group monthly returns, or returns clusters for increased vol periods.

#import matplotlib.pyplot as plt

#from pandas.tools.plotting import radviz

#plt.figure()

#radviz(datamain, 'Date')
```

In [35]:
```
from pandas.tools.plotting import bootstrap_plot

bootstrap_plot(datamain['SP500'], size=50, samples=100, color='grey')
```

```
/Users/diamond/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:3:
FutureWarning: 'pandas.tools.plotting.bootstrap_plot' is deprecated, import 'p
andas.plotting.bootstrap_plot' instead.
  This is separate from the ipykernel package so we can avoid doing imports un
til
```

Out[35]:





Visually assess the uncertainty of a statistic, such as mean, median, midrange. A random subset of a specified size is selected, the statistic in question is computed -- the process is repeated a specified number of times *samples=100*. Resulting plots and histograms are what constitutes the **bootstrap plot**.

END OF DEMONSTRATION

In [ ]: