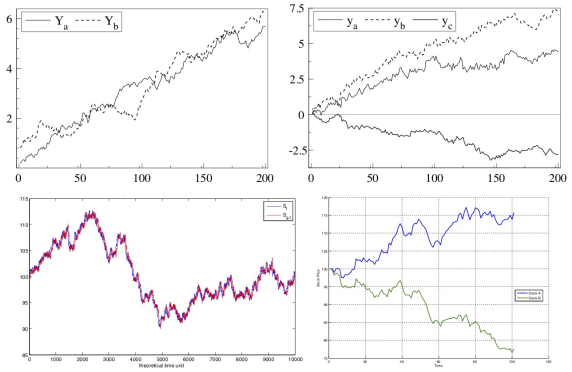


Pairs Trading with Reversion Evaluation

Dr Richard Diamond

PLEASE NOTE CERTAIN CODE NOT RELEASED

Resources for TS Topic

- Project Brief (TS Topic)
- Project Workshop (TS Topic)
- Cointegration Lecture
- Cointegrartion Case B and R code (10Y vs 25Y spot rates)
- Pairs Trading Python Lab (this session)
- Additional Material - curated readings 1) E. Zivot chapter, 2) P. McSharry example, 3) R. Diamond WILMOTT paper

---

While developing the strategy we rely on the frameworks of Vector Autoregression (multivariate linear regression) and Cointegration Analysis (Vector Error Correction Model).

Supporting numerical techniques -- which is what you will implement/compute as well:

- AIC/BIC for optimal lag selection, optionally
- Dickey-Fuller Test for stationarity ("unit root")
- Engle-Granger to confirm cointegration between a pair
- OU process for evaluating the quality of mean-reversion

In [10]:
```
%%html
<style> .container { width: 100%} </style>
```

In [2]:
```
%pylab inline

import warnings
import numpy as np
import pandas as pd
import datetime as dt

#The package now includes various advances, such as Value at Risk, Vecto
r Autoregression, Vector Error Correction
import statsmodels.tsa.api as sm #or as ts.

from scipy import optimize
from IPython.display import Image
```

```
Populating the interactive namespace from numpy and matplotlib
```

Foreword on Backtesting with Quantopian

How do we know that a strategy/portfolio/single stock beats the market? Below is a simple backtest example on the directional 100% investment in Marriott International stock.

Start 2013-01-02, end 2018-06-25.

```
cmd
pip install pyfolio
```

CAUTION Pyfolio package likely require OLD VERSIONS of matplotlib and numpy (possibly entire pre Python 3.7 core) to run correct.

If things not working for you, turn to Quantopian IDE (website (https://www.quantopian.com/algorithms)) for full backtesting capabilities and examples. You will need to register.

In [14]:
```
import pyfolio as pf

tickers_hotel = [['MAR', 'Marriott International'], ['IHG', 'InterContin
ental Hotels']]

cointprices = pd.read_excel('data_coint/prices_coint_hotel.xlsx', index_
col=0)
cointprices.columns = [tickers_hotel[0][0],tickers_hotel[1][0]]

returns = cointprices['MAR'].pct_change()


pf.create_full_tear_sheet(returns)
```

Start date	2013-01-02
End date	2018-06-25
Total months	65

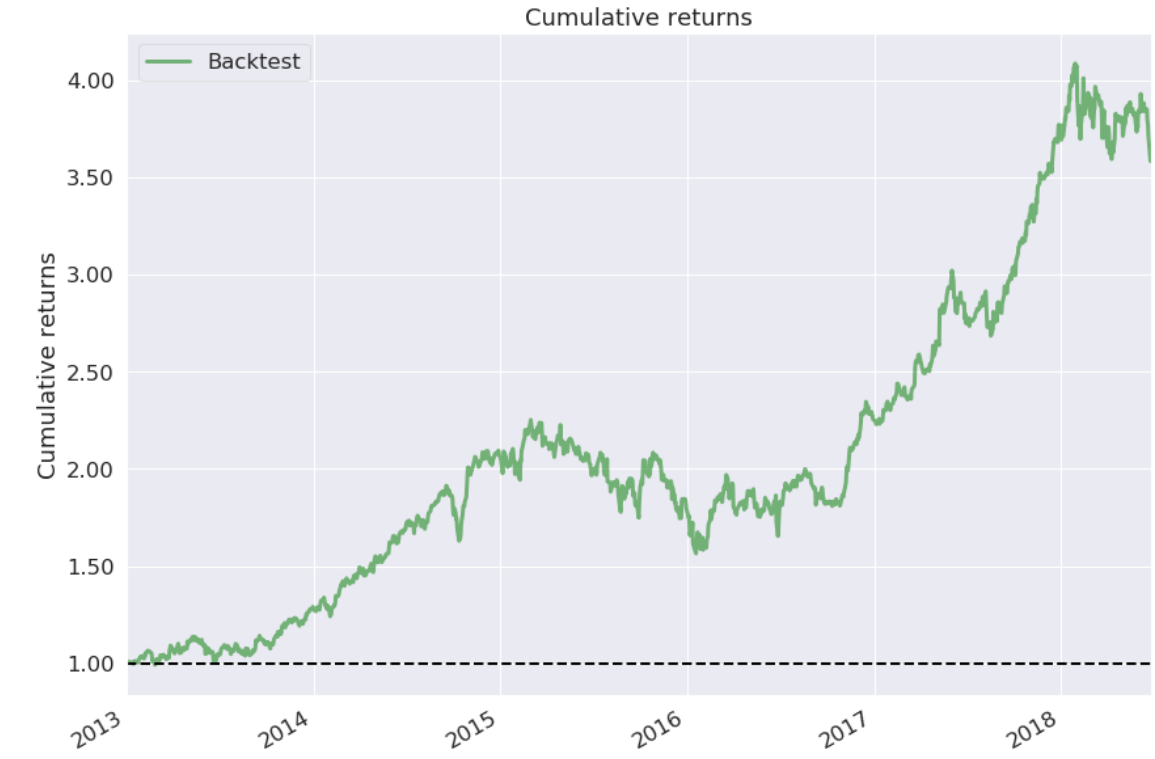
	Backtest
Annual return	26.2%
Cumulative returns	258.3%
Annual volatility	21.3%
Sharpe ratio	1.20
Calmar ratio	0.86
Stability	0.86
Max drawdown	-30.5%
Omega ratio	1.23
Sortino ratio	1.76
Skew	NaN
Kurtosis	NaN
Tail ratio	1.09
Daily value at risk	-2.6%

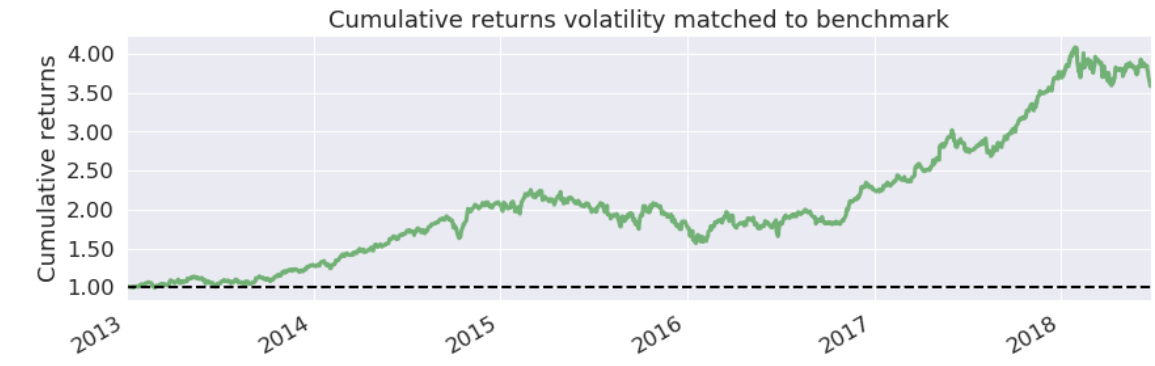
```
/anaconda3/lib/python3.6/site-packages/numpy/core/fromnumeric.py:52: Fu
tureWarning:
The current behaviour of 'Series.argmin' is deprecated, use 'idxmin'
instead.
The behavior of 'argmin' will be corrected to return the positional
minimum in the future. For now, use 'series.values.argmin' or
'np.argmin(np.array(values))' to get the position of the minimum
row.
  return getattr(obj, method)(*args, **kwds)
```

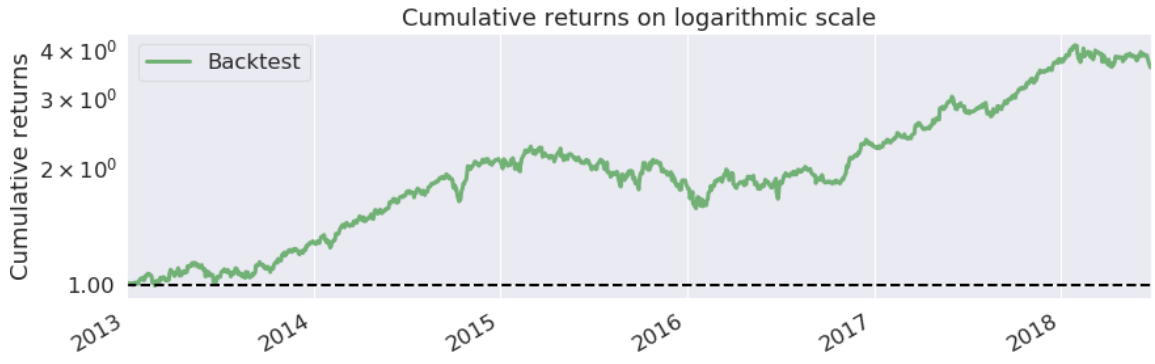
Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	30.49	2015-03-02	2016-01-19	2016-12-07	463
1	14.82	2014-09-18	2014-10-13	2014-10-29	30
2	12.31	2018-01-29	2018-06-25	NaT	NaN
3	11.62	2013-05-15	2013-06-24	2013-09-18	91
4	11.10	2017-06-02	2017-08-17	2017-09-28	85

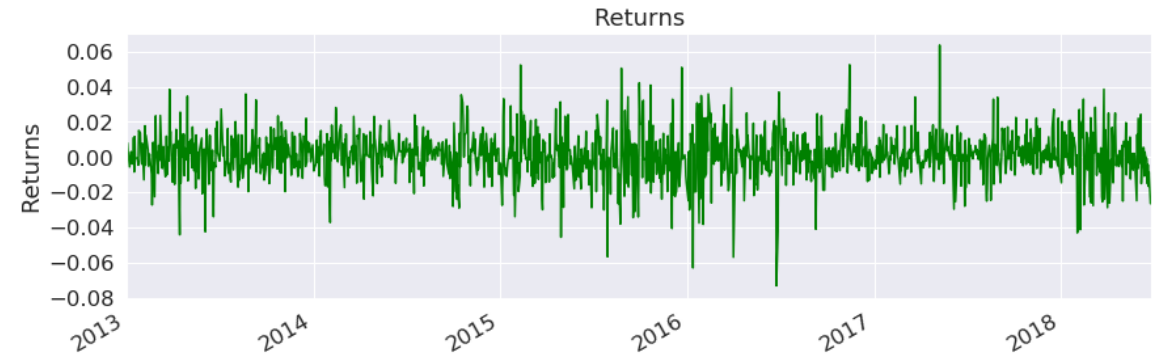
```
/anaconda3/lib/python3.6/site-packages/numpy/core/fromnumeric.py:52: Fu
tureWarning:
The current behaviour of 'Series.argmin' is deprecated, use 'idxmin'
instead.
The behavior of 'argmin' will be corrected to return the positional
minimum in the future. For now, use 'series.values.argmin' or
'np.argmin(np.array(values))' to get the position of the minimum
row.
  return getattr(obj, method)(*args, **kwds)
```

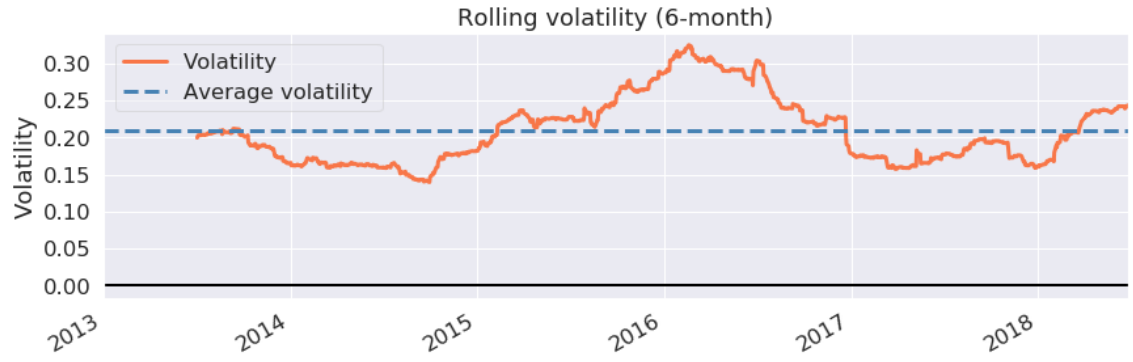
Stress Events	mean	min	max
Apr14	0.21%	-2.38%	2.31%
Oct14	0.37%	-2.90%	3.54%
Fall2015	-0.13%	-3.81%	5.05%
New Normal	0.10%	-7.31%	6.36%

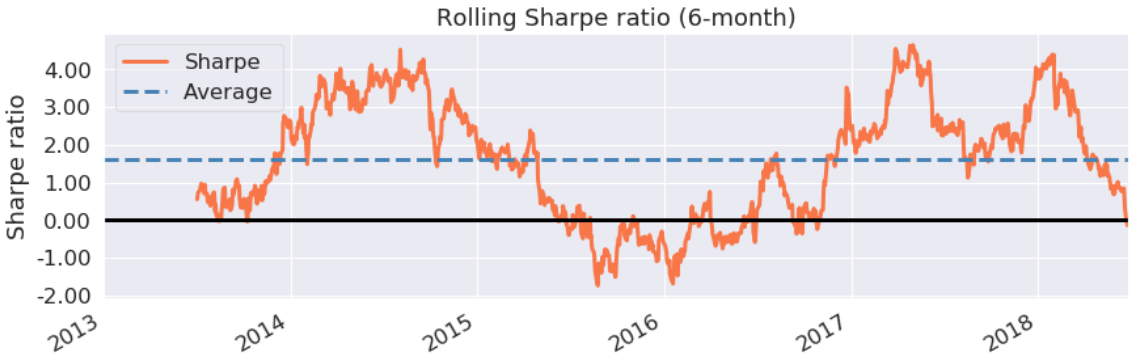

Cumulative returns
Backtest
Cumulative returns
4.00
3.50
3.00
2.50
2.00
1.50
1.00
2013
2014
2015
2016
2017
2018

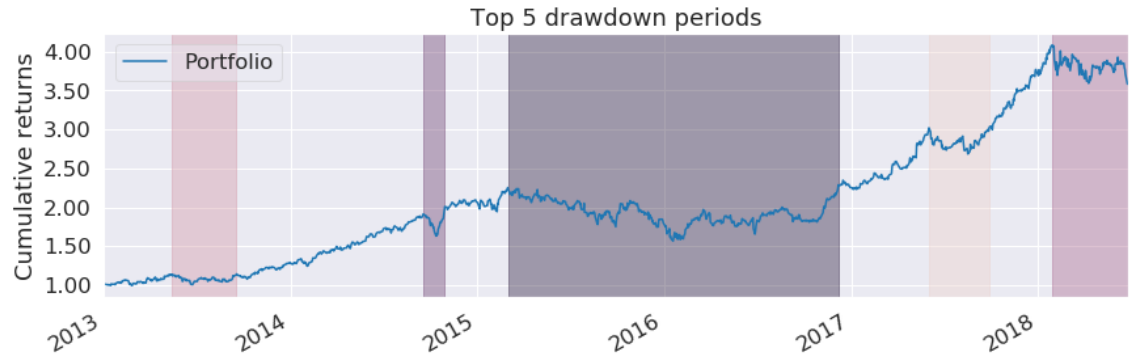

Cumulative returns volatility matched to benchmark
Cumulative returns
4.00
3.50
3.00
2.50
2.00
1.50
1.00
2013
2014
2015
2016
2017
2018

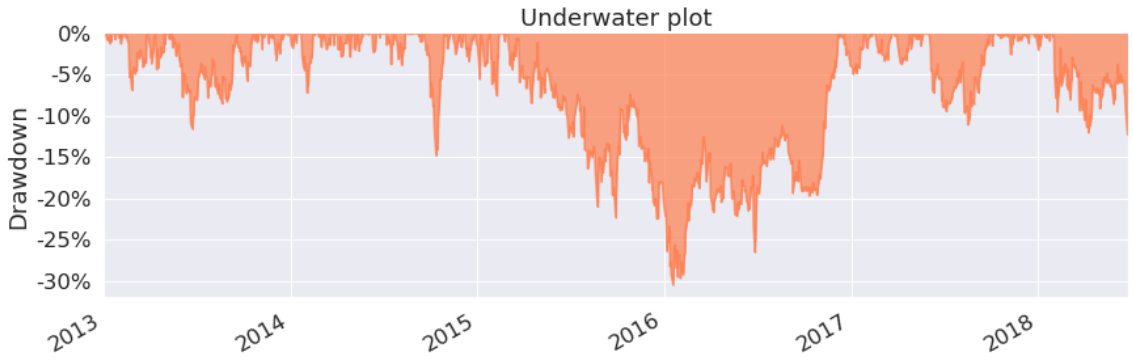

Cumulative returns on logarithmic scale
Backtest
Cumulative returns
4 × 10^0
3 × 10^0
2 × 10^0
1.00
2013
2014
2015
2016
2017
2018

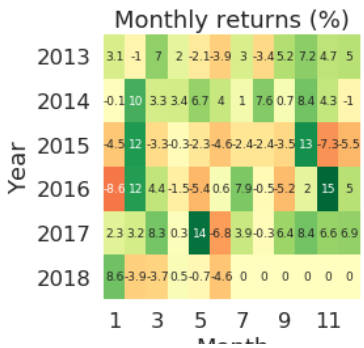

Returns
Returns
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
2013
2014
2015
2016
2017
2018

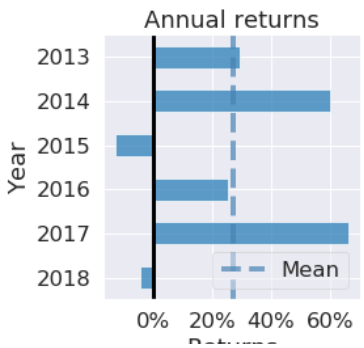

Rolling volatility (6-month)
Volatility
Average volatility
Volatility
0.30
0.25
0.20
0.15
0.10
0.05
0.00
2013
2014
2015
2016
2017
2018

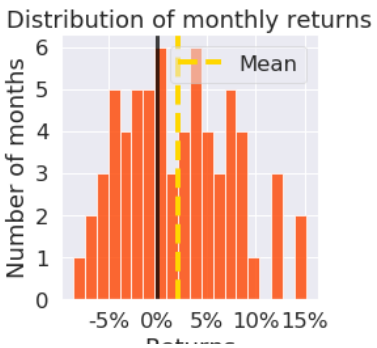

Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio
4.00
3.00
2.00
1.00
0.00
-1.00
-2.00
2013
2014
2015
2016
2017
2018


Top 5 drawdown periods
Portfolio
Cumulative returns
4.00
3.50
3.00
2.50
2.00
1.50
1.00
2013
2014
2015
2016
2017
2018

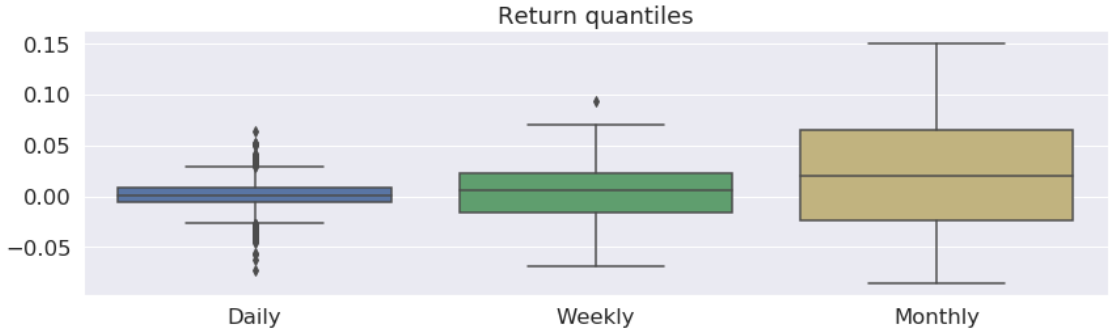

Underwater plot
Drawdown
0%
-5%
-10%
-15%
-20%
-25%
-30%
2013
2014
2015
2016
2017
2018

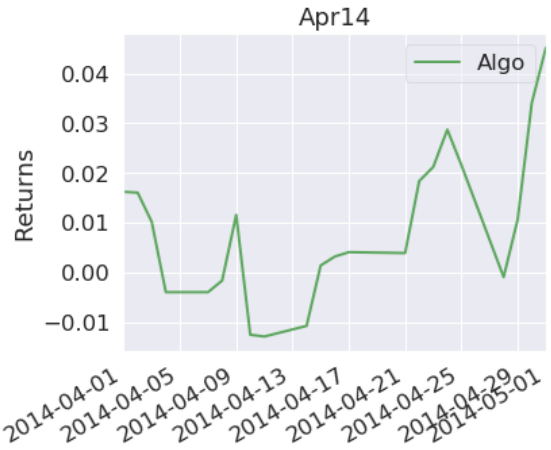

Monthly returns (%)
Year
2013
2014
2015
2016
2017
2018
1
3
5
7
9
11

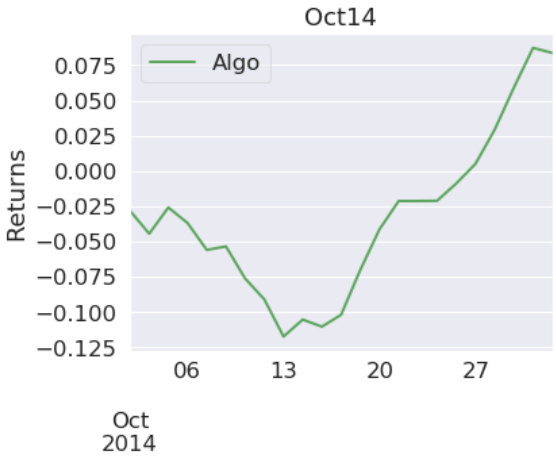

Annual returns
Year
2013
2014
2015
2016
2017
2018
Mean
0%
20%
40%
60%

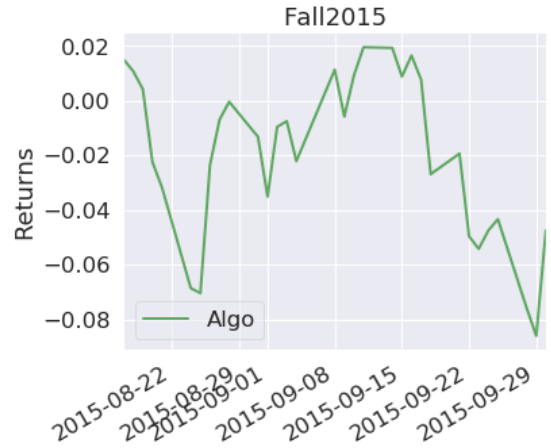

Distribution of monthly returns
Number of months
Mean
6
5
4
3
2
1
0
-5%
0%
5%
10%
15%

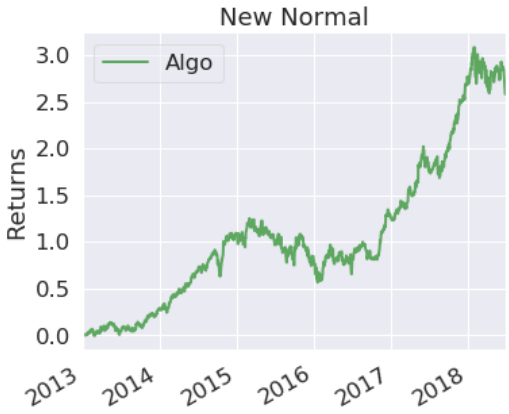
Month Returns Returns











In [ ]:

Vector Auto Regression

This is a model-free set up in which we regress ''everything on everything". The system is endogenous.

Asset returns are are made dependent on their own lagged values (past values).

You might also heard about dynamic structural equations modelling (DSEM). Well, Vector Autoregression is case of DSEM when there are no latent variables (all variables observable) and we rely on residuals. p is the number of lagged values as determined by AIC, BIC tests.

$$y_t = B_{\text{Constant}} + y_{t-1} B_1 + \cdots + y_{t-p} B_p + \varepsilon_t,$$

The following function *VAR(data, p)* estimates using Vector Autoregression in matrix form, calling on general multivariate linear regression model.

The function also checks stability of the system: whether all roots of the characteristic polynomials $\det(\lambda I - \widehat{B}_j)$ are within the unit circle for all $j = 1, \ldots, p$. If not, a warning will be issued.

The result is the estimated block matrix:

$$\widehat{B} = \begin{bmatrix} \widehat{B}_{\text{Constant}} \\ \widehat{B}_1 \\ \vdots \\ \widehat{B}_p \end{bmatrix}.$$

In [3]:

Multivariate Linear Regression

Consider a multivariate linear regression model in which we have $i = 1, \ldots, m$ observations $y_i = (y_{i1}, \ldots, y_{in})^T$ that we are regressing on $x_i = (x_{i1}, \ldots, x_{i\ell})^T$:

$$\begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & & \vdots \\ y_{m1} & \cdots & y_{mn} \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1\ell} \\ \vdots & & \vdots \\ x_{m1} & \cdots & x_{m\ell} \end{bmatrix} \begin{bmatrix} \beta_{11} & \cdots & \beta_{1n} \\ \vdots & & \vdots \\ \beta_{\ell 1} & \cdots & \beta_{\ell n} \end{bmatrix} + \begin{bmatrix} \varepsilon_{11} & \cdots & \varepsilon_{1n} \\ \vdots & & \vdots \\ \varepsilon_{m1} & \cdots & \varepsilon_{mn} \end{bmatrix}$$

This forms the basis for both, Vector Autoregression and Augmented Dickey-Fuller (ADF) Test. The model may also be written in short form as:

$$y_t = x_t B + \varepsilon_t,$$

where $\varepsilon_t = (\varepsilon_{t1}, \ldots, \varepsilon_{tn})^T$ and we have substituted t for i. In matrix notation the model may actually be written even shorter as:

$$Y = XB + \varepsilon$$

linear_regression_core() computes regression coefficients (estimates the model), given pre-formed matricies.

Number of observations:	m		
Number of regressands:	n		
Number of regressors:	ℓ		
Estimate of B:	$\widehat{B}$	$=$	$(X^T X)^{-1} X^T Y$
Residuals:	$\widehat{\varepsilon}$	$=$	$Y - X\widehat{B}$

The following output is useful for **inference**: decisions on specification of regression model and significance of coeffiecients hypothesis testing (if the true value of coeffient is zero, we infer that corresponding variable does not have explanatory power).

Residual covariance matrix:	$\widehat{\Sigma}$	$=$	$\widehat{\varepsilon}^T \widehat{\varepsilon} / m$
Covariance matrix of regression coefficients:	$\widehat{CoV}$	$=$	$\widehat{\Sigma} \otimes (X^T X)^{-1}$
Standard error of regression coefficient:	$\left(\mathrm{Var}\left(\widehat{\beta}_{kj}\right)\right)^{1/2}$	$=$	$\left(\frac{m}{m-\ell}\mathrm{diag}(\widehat{Co}\right.$
t-test statistic for whether regression coefficient is zero:	tstat_{kj}	$=$	$\widehat{\beta}_{kj} / \left(\mathrm{Var}\left(\widehat{\beta}_{kj}\right.\right.$
p-value for the corresponding t-test statistic:	pvalue_{kj}		
Akaike information criterion:	AIC	$=$	$\log\left\|\widehat{\Sigma}\right\| + 2\ell n /$
Bayesian information criterion:	BIC	$=$	$\log\left\|\widehat{\Sigma}\right\| + \log(n$

In [4]:

Cointegration modelling adds the constant (drift) and time-dependent variable (trend) to cointegrated relationship (cointegrated residual equation). It is the drift because y_t participates in correction equation Δy_t.

The above general multivariate linear regression model is extended to:

$$y_t = B_{\text{Constant}} + x_t B + \varepsilon_t, \quad \text{and}$$
$$y_t = B_{\text{Constant}} + t B_{\text{Trend}} + x_t B + \varepsilon_t.$$

Implemented simply by adding a line (or column) to dependent matrix X!

$$X = \left[\begin{array}{c|ccc} 1 & x_{11} & \cdots & x_{1\ell} \\ \vdots & \vdots & & \vdots \\ 1 & x_{m1} & \cdots & x_{m\ell} \end{array}\right], \quad \text{and}$$

$$X = \left[\begin{array}{cc|ccc} 1 & 1 & x_{11} & \cdots & x_{1\ell} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & m & x_{m1} & \cdots & x_{m\ell} \end{array}\right],$$

In [ ]:

Case Study 1: Ford, General Motors, and the Market

We start to explore with the matrix form regression (Vector Autoregression) it on 'a system' of returns series of two major US car manufactors and S&P 500 index ETF.

Please note that VAR exploration on returns does not relate to cointegration analysis directly. It is matrix form regression computation itself used instead of *statsmodels.api*

PLEASE NOTE EXCEL DATA WILL NOT BE DISTRIBUTED. PLEASE USE OWN EQUITIES DATA

In [7]:
```
tickers_auto = [['F', 'Ford'], ['GM', 'General Motors'], ['SPY', 'S&P 500']]

cointprices = pd.read_excel('data_coint/prices_coint_auto.xlsx', index_col=0)
# cointprices.columns = [tickers_auto[0][0], tickers_auto[1][0], tickers_auto[2][0]] #rename columns to tickers
cointprices.head()
```

Out[7]:

Date	Ford	General Motors	S&P 500
2010-11-18	12.541677	28.181452	102.562271
2010-11-19	12.666161	28.239155	102.844398
2010-11-22	12.603919	28.090796	102.758911
2010-11-23	12.214911	27.406654	101.271263
2010-11-24	12.409413	27.596235	102.767479

In [8]:
```
cointprices = cointprices[(cointprices.index > '2010-10-1') & (cointprices.index <= '2016-10-1')]

cointprices
```

Out[8]:

Date	Ford	General Motors	S&P 500
2010-11-18	12.541677	28.181452	102.562271
2010-11-19	12.666161	28.239155	102.844398
2010-11-22	12.603919	28.090796	102.758911
2010-11-23	12.214911	27.406654	101.271263
2010-11-24	12.409413	27.596235	102.767479
...	...	...	...
2016-09-26	10.861752	29.345434	207.159378
2016-09-27	10.834618	29.160870	208.445404
2016-09-28	10.934101	29.437716	209.480042
2016-09-29	10.825577	29.050135	207.584824
2016-09-30	10.916015	29.317751	209.151276

1477 rows × 3 columns

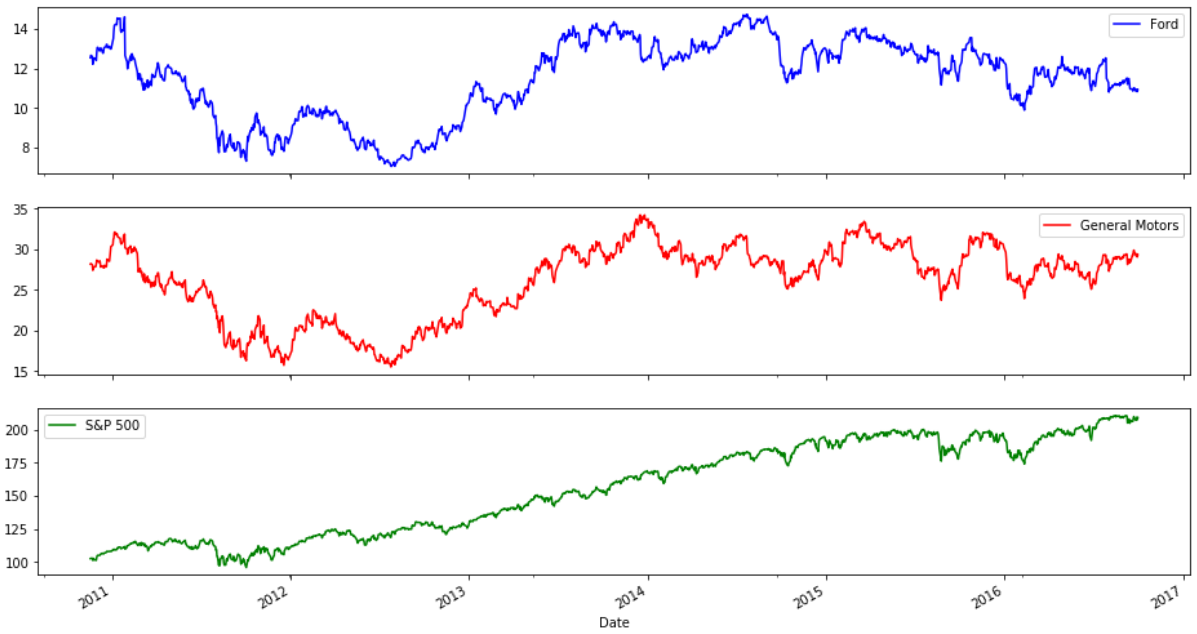
In [9]:
```
cointprices.dtypes
```

Out[9]:
```
Ford              float64
General Motors    float64
S&P 500           float64
dtype: object
```

The (adjusted) closing price for each times series looks like this:

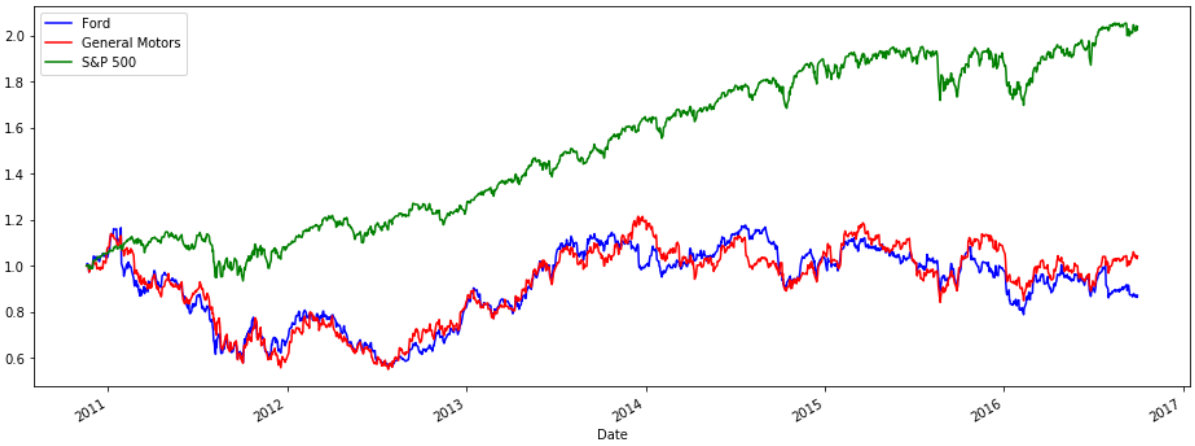
In [10]:
```
cointprices.plot(figsize=(16,9), color=('b', 'r', 'g'), subplots=True) #
https://matplotlib.org/3.1.1/gallery/color/named_colors.html
```

Out[10]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fb33baa9950
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb33baadb10
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb32885b1d0
>],
      dtype=object)
```



In [11]:
```
cointprices_1 = cointprices/cointprices.iloc[0]
cointprices_1.plot(figsize=(16,6), color=('b', 'r', 'g'))
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb33ba30390>



Cointegration testing is sensitive to **the initial value**

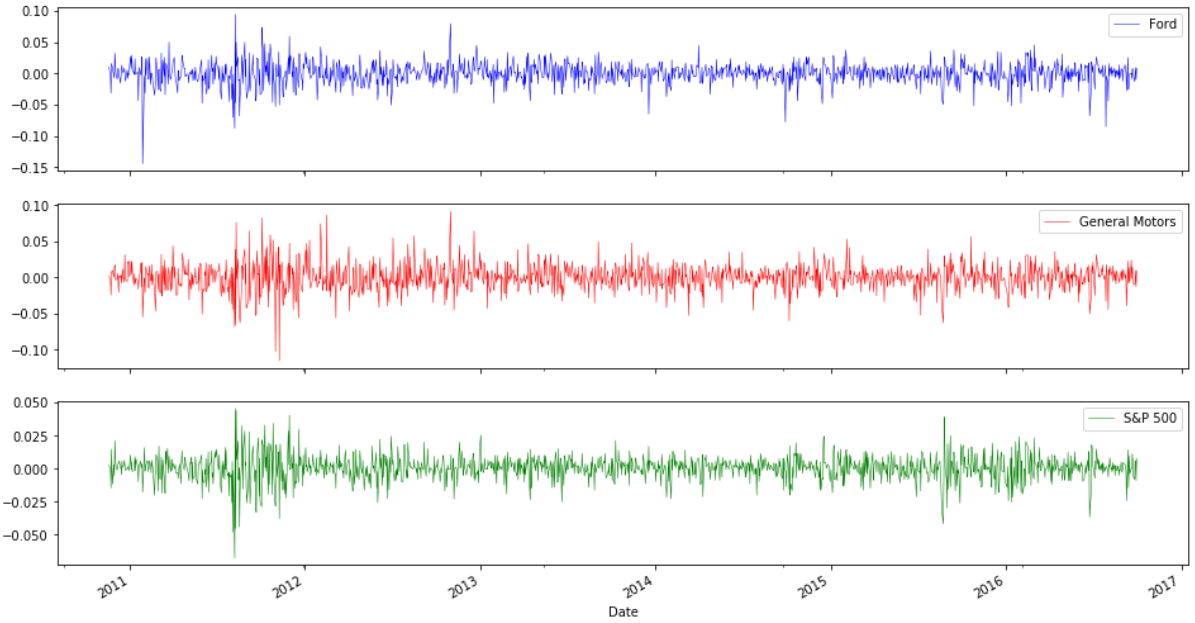
Normalising to start at one -- also allows 1:1 comparison of P&L:

- Both, Ford and General Motors delivered about 0% average annual return -- if we look at the performance over these five-year period.
- S&P 500 index, on the other hand, delived an average annual return > 12%

Daily Returns

In [12]:
```
returns = np.log(cointprices).diff().dropna()
returns.plot(figsize=(16,9), color=('b', 'r', 'g'), subplots=True, linewidth=0.5)
```

Out[12]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fb32889bf50>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb3288c2d50>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb3307f33d0>],
      dtype=object)
```



In [13]:
```
VAR_Demo = CODE_MATRIX_FORM(returns, 1)
VAR_Demo.fit
```

```
/Users/diamond/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py:618: UserWarning: merging between different levels can give an unintended result (1 levels on the left, 2 on the right)
  warnings.warn(msg, UserWarning)
```

Out[13]:

	Estimate			SD of Estimate			t-Statistic		
	Ford	General Motors	S&P 500	Ford	General Motors	S&P 500	Ford	General Motors	S&
Constant	-0.000049	0.000031	0.000508	0.000448	0.000481	0.000250	-0.108667	0.065476	2
(Lag 1, Ford)	0.102431	0.059577	0.004979	0.041445	0.044499	0.023086	2.471511	1.338832	0
(Lag 1, General Motors)	-0.005268	-0.018783	0.004345	0.036682	0.039385	0.020433	-0.143620	-0.476908	0
(Lag 1, S&P 500)	-0.087447	0.000511	-0.056109	0.068361	0.073398	0.038079	-1.279192	0.006964	-1

To investigate whether the daily returns depend on their past values, we fit the VAR(1) model to DAILY RETURNS (above, to be read **column-wise**)

The question now is whether the estimated coefficients are significantly different from zero. We inspect the p-values. We will reject the hypothesis that a regression coefficient is zero if the corresponding p-value is less than 5%.

In the table below TRUE indicates a regression coefficient that is SIGINIFICANTLY DIFFERENT from zero:

In [54]:
```
VAR_Demo.pvalue<0.05
```

Out[54]:

	Ford	General Motors	S&P 500
Constant	False	False	True
(Lag 1, Ford)	True	False	False
(Lag 1, General Motors)	False	False	False
(Lag 1, S&P 500)	False	False	False

The outcome is not very promising.

To investigate and re-check if we should have included more past values in the model, we check Information Criteria (AIC, BIC) for the range $p = 1, \dots, 10$:

In [14]:
```
IC = pd.DataFrame([CODE_MATRIX_FORM(returns, p+1).IC for p in range(10
)], index=[p+1 for p in range(10)])
IC
```

```
/Users/diamond/opt/anaconda3/lib/python3.7/site-packages/pandas/core/re
shape/merge.py:618: UserWarning: merging between different levels can g
ive an unintended result (1 levels on the left, 2 on the right)
  warnings.warn(msg, UserWarning)
```

Out[14]:

	AIC	BIC
1	-26.908843	-26.865753
2	-26.899725	-26.824277
3	-26.902051	-26.794209
4	-26.897263	-26.756991
5	-26.904856	-26.732119
6	-26.893282	-26.688043
7	-26.884298	-26.646522
8	-26.876298	-26.605948
9	-26.872130	-26.569171
10	-26.860620	-26.525015

We select by the largest modulus (absolute value), in this case, optimal lag $p = 1$ according to the both, AIC and BIC.

In [56]:
```
IC.idxmin()
```

Out[56]:
```
AIC    1
BIC    1
dtype: int64
```

The above vignette analysis of returns can be concluded that past lagged values cannot be used to predict the future returns for these two equties.

Things can be different for market indicies and for over long-term, 15-30 years of daily data or even more, but that has limited applicability.

Cointegration Analysis

The idea is that a linear combination of two non-stationary time series (say, each is Brownian Motion) results in the residual, which is not stationary. That is not guaranteed. More formally, we say that Prices are I(1) time series (nonstaionary), whereas their combination -- a residual from naive regression is I(0) time series, or stationary.

In this section we implement the Engle-Granger procedure to confirm (or not) if the pair is cointegrated and therefore, suitable for trade design.

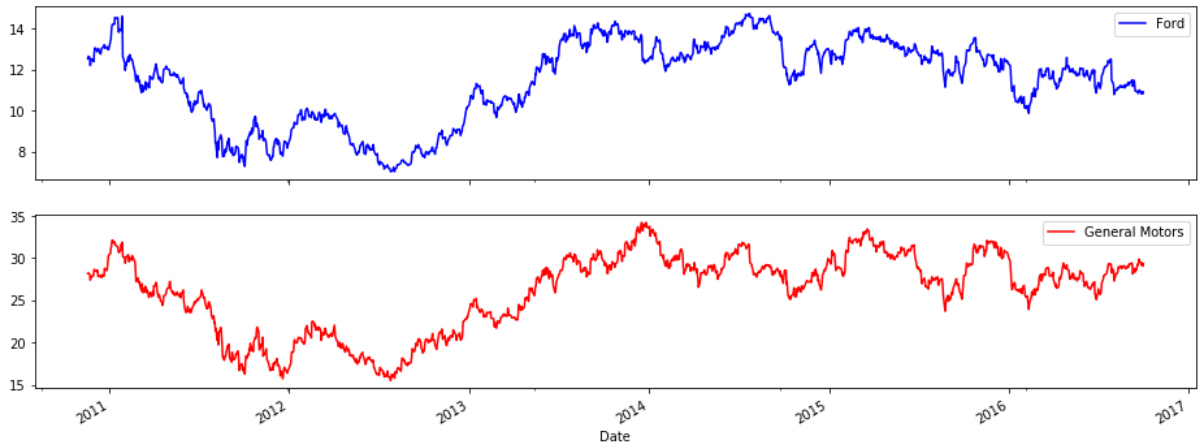
We begin by recapping what the prices for Ford and General Motors from before look like:

Engle-Granger Procedure

In [20]:
```
Price_B = cointprices.iloc[:, [1]] #GM stock price -- variable X
Price_A = cointprices.iloc[:, [0]] #Ford stock price -- variable Y

cointprices.iloc[:, [0,1]].plot(figsize=(16,6), color=('b', 'r'), subplots=True)
```

Out[20]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7fb33bb6ced0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb33c282110>],
      dtype=object)
```



- Improvement 0. Appropriate to remind that we **HAVE NOT NORMALISED** F and GM time series by the initial value -- to begin at 1 each.

ENGLE-GRANGER STEP 1. Cointegrated Residual (spread e_t)

We are now ready to perform the 1st step of the Engle-Granger procedure which is to estimate the model:

$$\text{Ford}_t = \beta\,\text{GM}_t + \mu_e + \varepsilon_t$$

and so, our **NAIVE** cointegrated residual is

$$e_t = Ford_t - 0.4066\,GM_t - 0.8682$$

Vector cointegrating weights can be fine-tuned by running the ready multivariate cointegration (VECM routines, such as *cajorls* in R package *urca*)

$$\boldsymbol{\beta}_{Coint} = [1, -0.4066] \qquad \mu_e = -0.8682$$

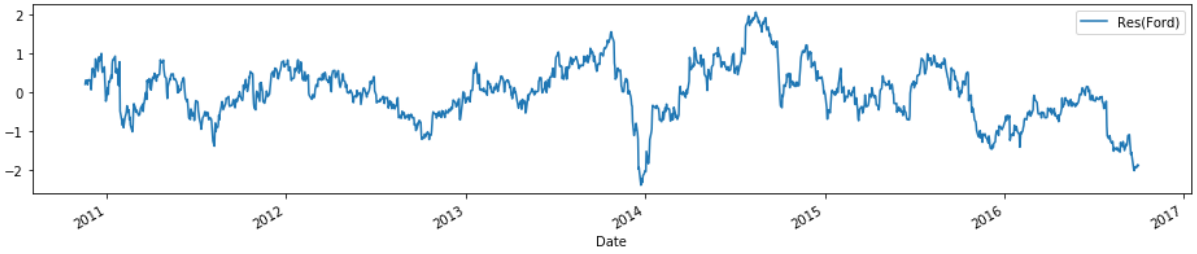
In [21]:
```
EG_1 = linear_regression(Price_A, Price_B, 'c')
EG_1.fit
```

Out[21]:

	Estimate Ford	SD of Estimate Ford	t-Statistic Ford
Constant	0.868166	0.106511	8.150924
General Motors	0.406757	0.004021	101.159333

In [22]:
```
e = EG_1.residuals
e.plot(figsize=(16,3))
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb34877c3d0>
```



The question now is whether the residual series is stationary -- has no unit root (randomness). We perform Augmented Dickey-Fuller Test on this Cointegrated residual (therefore, referred to as CADF).

NAIVE COINTEGRATING RELATIONSHIP Though the stationarity of residual is important, here it only transpires over > 4-year period. Regressing one equity price on another arbitrarily does not create cointegration. This is a first step in the analysis of a potential long-run relationship between the two time series P_t^A and P_t^B.

In [26]:
```
CODE_ADF(e).fit
```

```
/Users/diamond/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py:618: UserWarning: merging between different levels can give an unintended result (1 levels on the left, 2 on the right)
  warnings.warn(msg, UserWarning)

Number of lags used: 1
```

Out[26]:

	Estimate ΔRes(Ford)	SD of Estimate ΔRes(Ford)	t-Statistic ΔRes(Ford)
Constant	0.005364	0.007632	0.702889
Trend	-0.000009	0.000009	-1.030575
(Lag 1, Res(Ford))	-0.019780	0.005417	-3.651647
(Lag 1, ΔRes(Ford))	0.024925	0.018565	1.342579

ADF TEST EQUATION

$$\Delta e_t = \varphi e_{t-1} + \varphi_{aug1} \Delta e_{t-1} + const + \varphi_t t + \varepsilon_t$$

Look at the value of test statistic for the parameter φ. In this case, it is -3.65 which is lower than the 5% Dickey-Fuller critical value of -3.45 (but not much lower). We reject the H_0 hypothesis of unit root (see Cointegration Lecture). The residuals are stationary.

- Improvement 1. Test equation above includes time dependence $\varphi_t t$, referred to as 'trend'.

DO NOT INCLUDE trend in your ADF tests and cointegrating residual -- it will make you think cointegration is present when it is very weak. In fact, without $\varphi_t t$ term, we might not even get stationarity result.

ENGLE-GRANGER STEP 2. Error correction equations

In general form,

$$\Delta P_t^A = \varphi \Delta P_t^B - (1-\alpha)\widehat{e}_{t-1} + \varepsilon_t$$

We are now in position to perform **the 2nd step of Engle-Granger** procedure which is to estimate the Equilibrium Correction Model, or error correction equations:

$$\Delta \text{Ford\_Price}_t = \varphi \Delta \text{GM\_Price}_t - (1-\alpha)\text{Coint\_FordGM}_{t-1} + \varepsilon_t$$

In [29]:
```
#Price_B = cointprices.iloc[:, [1]] #GM stock price -- variable X
#Price_A = cointprices.iloc[:, [0]] #Ford stock price -- variable Y

ΔPrice_A = pd.DataFrame(Price_A).diff().dropna().add_prefix('Δ')

ΔPrice_B = pd.DataFrame(Price_B).diff().dropna().add_prefix('Δ')
rhs = ΔPrice_B.join(e.shift(1).dropna().add_prefix('(Lag 1, ').add_suffi
x(')'))
#where e = EG_1.residuals
```

In [36]:
```
EG_2 = linear_regression(ΔPrice_A, rhs, '')
EG_2.fit
```

Out[36]:

	Estimate ΔFord	SD of Estimate ΔFord	t-Statistic ΔFord
ΔGeneral Motors	0.294164	0.007947	37.017757
(Lag 1, Res(Ford))	-0.014555	0.005032	-2.892389

Here we check significance of $-(1-\alpha)$ term -- that ensures the correction the long run equilibrium.

In error correction equation the p-value is evaluated agaist t-distribution as is common for regression. We will reject H_0 that a regression coefficient is zero if the corresponding p-value is less than 5%. In the table below "True" indicates a regression coefficient that is siginificantly different from zero:

In [94]:
```
EG_2.pvalue<0.05
```

Out[94]:

	ΔFord
ΔGeneral Motors	True
(Lag 1, Res(Ford))	True

EG Correction Equation "other way around" (GM on Ford)

To fully complete checks, we should estimate **EG STEP 1** the other way around: General Motors on Ford.

$$\Delta \text{GM\_Price}_t = \varphi \Delta \text{Ford\_Price}_t - (1-\alpha)\text{Coint\_FordGM}_{t-1} + \varepsilon_t$$

In [37]:
```
rhs_ow = ΔPrice_A.join(e.shift(1).dropna().add_prefix('(Lag 1, ').add_su
ffix(')')) # pluggin in 'wrong way' residuals
EG_2_ow = linear_regression(ΔPrice_B, rhs_ow, '')
EG_2_ow.fit
```

Out[37]:

	Estimate	SD of Estimate	t-Statistic
	ΔGeneral Motors	ΔGeneral Motors	ΔGeneral Motors
ΔFord	1.637771	0.044243	37.017757
(Lag 1, Res(Ford))	0.044028	0.011852	3.714754

Correcting the correction

- Improvement 3. The residual also needs to be re-stimated for GM on Ford, and preferrably also tested by Augmended Dickey-Fuller test before proceeding.

$$\Delta \text{GM\_Price}_t = \varphi \Delta \text{Ford\_Price}_t - (1-\alpha)\text{Coint\_GMFord}_{t-1} + \varepsilon_t$$

The proper cointegrated residual to use now is

$$e_t = GM_t - 2.1488\ Ford_t - 1.4213$$

The second candidate to be a cointegrated relationship (weights) vector.

$$\beta_{Coint} = [1, -2.1488] \qquad \mu_e = -1.4213$$

In [38]:
```
EG_1_ow = linear_regression(Price_B, Price_A, 'c') #linear_regression(x,
y, 'c')
EG_1_ow.fit
```

Out[38]:

	Estimate	SD of Estimate	t-Statistic
	General Motors	General Motors	General Motors
Constant	1.421283	0.247507	5.742398
Ford	2.148750	0.021241	101.159333

In [39]:
```
CODE_ADF(EG_1_ow.residuals).fit
```

```
Number of lags used: 1

/Users/diamond/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py:618: UserWarning: merging between different levels can give an unintended result (1 levels on the left, 2 on the right)
  warnings.warn(msg, UserWarning)
```

Out[39]:

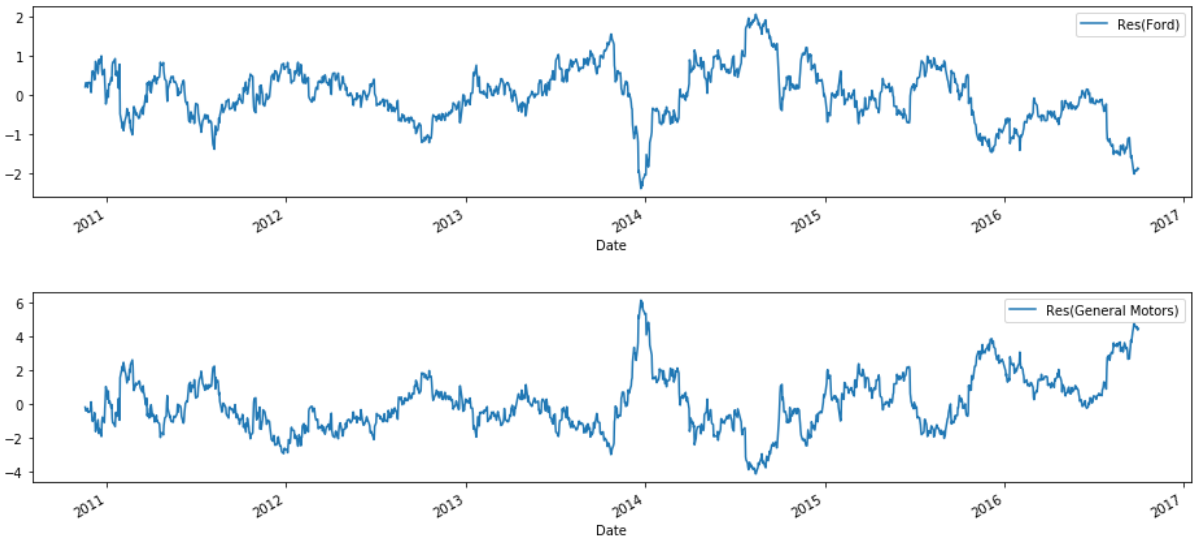
	Estimate	SD of Estimate	t-Statistic
	ΔRes(General Motors)	ΔRes(General Motors)	ΔRes(General Motors)
Constant	-0.025940	0.018285	-1.418671
Trend	0.000040	0.000022	1.817531
(Lag 1, Res(General Motors))	-0.022476	0.005709	-3.936824
(Lag 1, ΔRes(General Motors))	0.019613	0.018487	1.060908

ADF on the residual form that regression gives stronger test statistic -3.94 for the parameter φ. We should have estimated model that way from the start, eg GM on Ford.

In [41]:
```
e_ow = EG_1_ow.residuals
```

In [42]:
```
e.plot(figsize=(16,3))
e_ow.plot(figsize=(16,3))
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb328b6f310>
```



We have estimated

$$\Delta\text{Ford\_Price}_t = \varphi\Delta\text{GM\_Price}_t - (1-\alpha)\text{Coint\_FordGM}_{t-1} + \varepsilon_t$$

In [44]:
```
EG_2 = linear_regression(ΔPrice_A, rhs, '')
EG_2.fit
```

Out[44]:

	Estimate ΔFord	SD of Estimate ΔFord	t-Statistic ΔFord
ΔGeneral Motors	0.294164	0.007947	37.017757
(Lag 1, Res(Ford))	-0.014555	0.005032	-2.892389

$$\Delta\text{GM\_Price}_t = \varphi\Delta\text{Ford\_Price}_t - (1-\alpha)\text{Coint\_GMFord}_{t-1} + \varepsilon_t$$

In [46]:
```
rhs_ow = ΔPrice_A.join(e_ow.shift(1).dropna().add_prefix('(Lag 1, ').add
_suffix(')'))
EG_2_ow = linear_regression(ΔPrice_B, rhs_ow, '')
EG_2_ow.fit
```

Out[46]:

	Estimate ΔGeneral Motors	SD of Estimate ΔGeneral Motors	t-Statistic ΔGeneral Motors
ΔFord	1.634879	0.044235	36.958582
(Lag 1, Res(General Motors))	-0.019399	0.005157	-3.761997

QUALITY OF REVERSION -- "STEP 3" to ENGLE-GRANGER PROCEDURE

The strategy trades the spread from the long/short portfolio -- weights given $\boldsymbol{\beta}_{Coint}$. Profitability and trade-offs (frequency of trades vs. profit per trade) will become clearer from after we fit the cointegrated residual, spread e_t, to the Ornstein-Uhlenbeck process.

$$\mathrm{d}e_{t+\tau} = -\theta(e_t - \mu)\,\mathrm{dt} + \sigma_{OU}\mathrm{dW}_t$$

AR(1) on the spread itself is guaranteed -- see *Learning and Trusting Cointegration in Statistical Arbitrage* by Richard Diamond, **WILMOTT Magazine**, November 2013, Issue 68. SSRN link (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2220092)

$$\widehat{e}_t = C + B\widehat{e}_{t-1} + \varepsilon_t$$

In [95]:
```
OU = VAR(e, 1)
OU.fit
```

```
/Users/diamond/opt/anaconda3/lib/python3.7/site-packages/pandas/core/reshape/merge.py:618: UserWarning: merging between different levels can give an unintended result (1 levels on the left, 2 on the right)
  warnings.warn(msg, UserWarning)
```

Out[95]:

	Estimate Res(Ford)	SD of Estimate Res(Ford)	t-Statistic Res(Ford)
Constant	-0.001390	0.003782	-0.367687
(Lag 1, Res(Ford))	0.981062	0.005354	183.251409

Since we are analysing daily data we have...

In [96]:
```
tau = 1/252
```

and so

In [97]:
```
theta = -np.log(OU.fit.iloc[1, 0])/tau
theta
```

Out[97]: 4.818244729386412

This indicates a half-life of...

In [98]:
```
H = np.log(2)/theta
H
```

Out[98]: 0.14385885721670585

year fraction -- for 1 month of time, $\tau = 0.0833$ (year fraction).

H corresponds to the following number of *working days*:

In [99]:
```
H/tau
```

Out[99]: 36.25243201860988

36 working days.

The speed of mean-reversion, in this case, is not particularly fast so we might not expect that many trades when implementing the strategy that trades the spread by constructing a long/short portfolio.

As mentioned earlier we could also have done the regression in step 1 the other way around, and we saw that the resulting residuals were also stationary. Had we done the entire Engle-Granger procedure the other way around, the resulting half-life would have been 28.4 days. Since this is similar to the above, we will stick with 'Ford' as being our lead variable.

In [ ]:
```
# DEMO STOPS
# DELEGATES TO IMPLEMENT SIGMA_EQ COMPUTATION AND VALIDATION, Z*, and PA
IR TRADE P&L
```

In [ ]:
```

```