Mark Holt Data Exploration 03

Based on the dataset provided one might "speculate" that a price difference of 11 dollars a barrel might be associated with a change in the treasury yield of 0.37. If OPEC has to expend money to defend a $100 price floor for a barrel of oil then it's spending on US Treasury Bonds may well fall, thus lowering the yield. This hypothesis is based on the 2012 data from the supplied dataset.

I am confused about how, given the analysis performed, to quote a confidence interval for the prediction.

However, whether one should be making any assumptions that there is a connection between oil price and the US Treasury yield is a point for discussion. The 2 time series are partially correlated, but is this simply due to the stochastic drift in the respective time series. There seems little evidence for the cointegration of the 2 time series.

In [53]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from numpy import nan as NA
```

In [9]:
```
dF = pd.read_csv("/home/vagrant/fall_2014_assignments/dataexplor03/SuperH
appyFunDataSet.csv")
```

There are six variables, all indexed by time - i.e. all are time series. Look to aline the variables onto a common single time index The index will be time

In [10]:
```
dFC = dF.dropna()
```

In [11]:
```
dFC.columns
```

Out[11]:
```
Index([u'LIBOR_date', u'LIBOR_price', u'OIS_date', u'OIS_price', u'Oil_dat
e', u'Oil_price', u'US_equity_index_date', u'US_equity_index', u'US10Y_dat
e', u'US10Y_yield', u'USD_trade_weighted_date', u'USD_trade_weighted_index
'], dtype='object')
```

In [12]:
```
dFC.values
```

Out[12]:
```
array([['1/2/2003', 1.38, '1/1/2003', ..., 3.8175, '1/2/2003', 101.88],
       ['1/3/2003', 1.39, '1/2/2003', ..., 4.0305, '1/3/2003', 101.69],
       ['1/6/2003', 1.38875, '1/3/2003', ..., 4.0169, '1/6/2003', 101.36],
       ...,
       ['9/10/2012', 0.40425, '5/21/2012', ..., 1.7688, '10/24/2012', 80.9
7],
       ['9/11/2012', 0.39875, '5/22/2012', ..., 1.7346, '10/25/2012', 80.9
2],
       ['9/12/2012', 0.39425, '5/23/2012', ..., 1.7774, '10/26/2012', 81.0
1]], dtype=object)
```

In [13]:
```
dFC.head()
```

Out[13]:

	LIBOR_date	LIBOR_price	OIS_date	OIS_price	Oil_date	Oil_price	US_equity_index_date
0	1/2/2003	1.38000	1/1/2003	1.231	1/2/2003	31.85	1/2/2003
1	1/3/2003	1.39000	1/2/2003	1.226	1/3/2003	33.08	1/3/2003
2	1/6/2003	1.38875	1/3/2003	1.234	1/6/2003	32.10	1/6/2003
3	1/7/2003	1.38750	1/6/2003	1.237	1/7/2003	31.08	1/7/2003
4	1/8/2003	1.38000	1/7/2003	1.234	1/8/2003	30.56	1/8/2003

Take each time series separately as a pair of vectors - the dates and the variable itself. Multiply all dollar values by 100 so that we are dealing with cents. Na's have been dropped.

In [14]:
```
lib1=dFC['LIBOR_date']
lib2=dFC['LIBOR_price'].values * 100.0
```

In [15]:
```
oil1 = dFC['Oil_date']
oil2 = dFC['Oil_price'].values * 100.00
```

In [16]:
```
ois1 = dFC['OIS_date']
ois2 = dFC['OIS_price'].values * 100.00
```

In [17]:
```
usei1 = dFC['US_equity_index_date']
usei2 = dFC['US_equity_index'].values
```

In [18]:
```
us10Y1 = dFC['US10Y_date']
us10Y2 = dFC['US10Y_yield'].values
```

In [19]:
```
usTW1 = dFC['USD_trade_weighted_date']
usTW2 = dFC['USD_trade_weighted_index'].values
```

Using the pandas data conversion function intrepret the date strings. Then create 6 separate time series, all formally indexed by date.

In [20]:
```
lib_s = [pd.datetime.strptime(x, '%m/%d/%Y') for x in lib1]
oil_s = [pd.datetime.strptime(x, '%m/%d/%Y') for x in oil1]
ois_s = [pd.datetime.strptime(x, '%m/%d/%Y') for x in ois1]
usei_s =[pd.datetime.strptime(x, '%m/%d/%Y') for x in usei1]
us10Y_s =[pd.datetime.strptime(x, '%m/%d/%Y') for x in us10Y1]
usTW_s =[pd.datetime.strptime(x, '%m/%d/%Y') for x in usTW1]

libS = pd.Series(lib2, index=lib_s)
oisS = pd.Series(ois2, index=ois_s)
oilS = pd.Series(oil2, index=oil_s)
useiS = pd.Series(usei2, index=usei_s)
us10YS = pd.Series(us10Y2, index=us10Y_s)
usTWS = pd.Series(usTW2, index=usTW_s)
```

Using the date indices recombine the variables into a single time series. All variables are now date aligned.

In [21]:
```
a=pd.concat([libS,oisS,oilS,useiS,us10YS,usTWS], axis=1)
a.head()
#Na's have arisen because of the time alignment - some colummns did not h
ave entries for times that other columns did.
```

Out[21]:

	0	1	2	3	4	5
2003-01-01	NaN	123.1	NaN	NaN	3.8175	NaN
2003-01-02	138.000	122.6	3185	909.03	4.0305	101.88
2003-01-03	139.000	123.4	3308	908.59	4.0169	101.69
2003-01-06	138.875	123.7	3210	929.01	4.0518	101.36
2003-01-07	138.750	123.4	3108	922.93	4.0053	101.45

In [22]:
```
dFCFinal = a.dropna()
dFCFinal.columns = ["LIB_Price", "OIS_Price", "Oil_Price", "EIndex", "Yie
ld","TWI"]
cL = ["LIB_Price", "OIS_Price", "Oil_Price", "EIndex", "Yield","TWI"]
dFCFinal.head()
```

Out[22]:

	LIB_Price	OIS_Price	Oil_Price	EIndex	Yield	TWI
2003-01-02	138.000	122.6	3185	909.03	4.0305	101.88
2003-01-03	139.000	123.4	3308	908.59	4.0169	101.69
2003-01-06	138.875	123.7	3210	929.01	4.0518	101.36
2003-01-07	138.750	123.4	3108	922.93	4.0053	101.45
2003-01-08	138.000	122.3	3056	909.93	4.0169	101.34

After dealing with remaining NAs the columns were relabelled

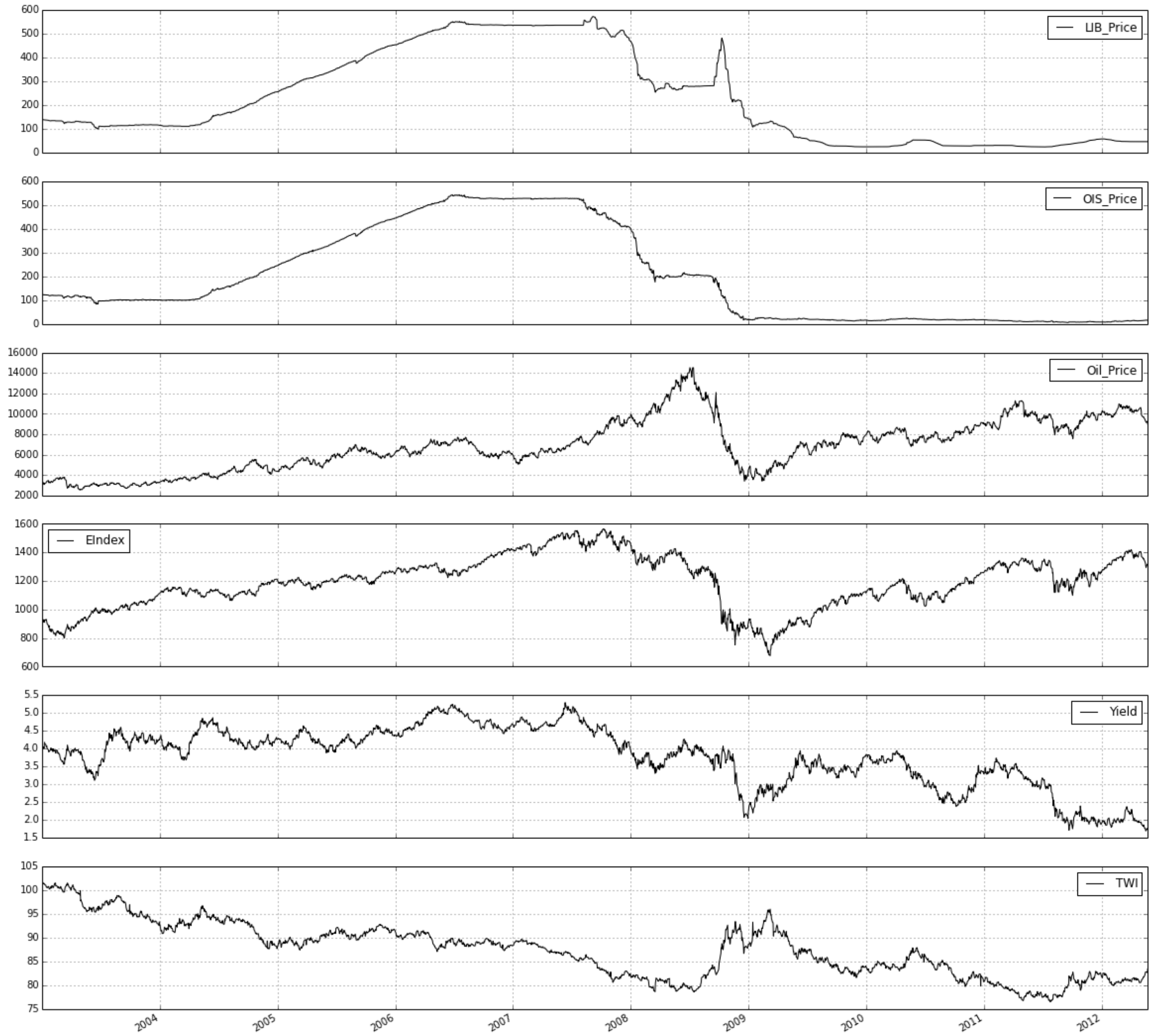
In [23]:
```
dFCFinal.describe()
```

Out[23]:

	LIB_Price	OIS_Price	Oil_Price	EIndex	Yield	TWI
count	2294.000000	2294.000000	2294.000000	2294.000000	2294.000000	2294.000000
mean	228.031763	197.617554	6861.165214	1192.055924	3.785376	87.440510
std	189.867619	192.778965	2518.032535	179.118072	0.834743	5.757146
min	24.500000	6.300000	2524.000000	676.530000	1.697100	76.550000
25%	48.506250	18.725000	4916.000000	1091.517500	3.334525	82.320000
50%	135.000000	116.050000	6761.000000	1199.555000	3.933750	87.975000
75%	404.447000	378.600000	8608.750000	1315.997500	4.427675	91.310000
max	572.500000	543.400000	14529.000000	1565.150000	5.292800	101.880000

Visualize the time aligned data

In [24]:
```
dFCFinal[cL].plot(subplots=True, figsize=(20,20))
plt.show()
```



Many of the variables look highly correlated:

In [25]:
```
dFCFinal.corrwith(dFCFinal['Yield'])
```

Out[25]:
```
LIB_Price    0.728116
OIS_Price    0.773676
Oil_Price   -0.345260
EIndex       0.272471
Yield        1.000000
TWI          0.449098
dtype: float64
```

In [26]:
```
dFCFinal.corrwith(dFCFinal['LIB_Price'])
```

Out[26]:
```
LIB_Price    1.000000
OIS_Price    0.976345
Oil_Price   -0.034957
EIndex       0.550633
Yield        0.728116
TWI          0.156625
dtype: float64
```

In [27]:
```
dFCFinal.corrwith(dFCFinal['Oil_Price'])
```

Out[27]:
```
LIB_Price   -0.034957
OIS_Price   -0.082498
Oil_Price    1.000000
EIndex       0.552027
Yield       -0.345260
TWI         -0.901076
dtype: float64
```

Commentary:

It is difficult to know how to approach this problem. We seek the effect of an change in oil price on Yield. Oil price and yield have a negative correlation of approx -0.35. There are other variables present in this data that are highly correlated. Yield is very correlated with both LIBOR and OIS (overnight index swap). LIBOR and OIS are almost perfectly correlated themselves, as evidenced from the graph above and the correlation index (0.97). Conversely LIBOR and OIS have almost zero correlation with oil price. Re-examine the data looking at just the most recent year - 2012

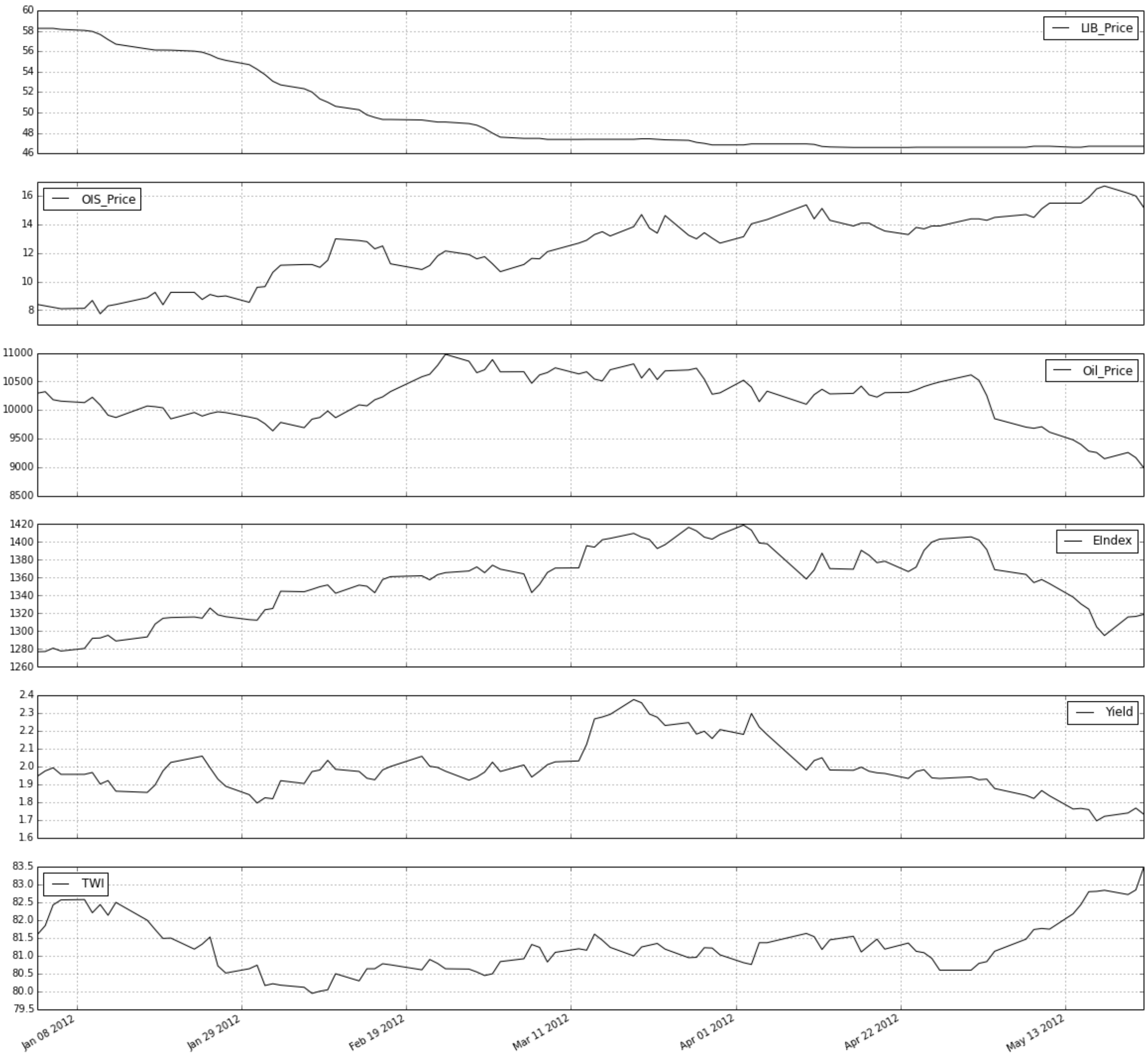
In [28]:
```
dF_after2012 = dFCFinal[pd.datetime(2012,1,1):].copy()
dF_after2012.describe()
```

Out[28]:

	LIB_Price	OIS_Price	Oil_Price	EIndex	Yield	TWI
count	96.000000	96.000000	96.000000	96.000000	96.000000	96.000000
mean	49.658750	12.348750	10194.687500	1355.384375	1.990294	81.265833
std	3.946566	2.376515	439.208554	38.908512	0.148403	0.733423
min	46.565000	7.750000	8990.000000	1277.060000	1.697100	79.950000
25%	46.685000	10.962500	9890.750000	1322.782500	1.922700	80.757500
50%	47.365000	12.890000	10268.500000	1361.720000	1.974700	81.190000
75%	52.081250	14.125000	10541.500000	1390.712500	2.033600	81.562500
max	58.250000	16.700000	10977.000000	1419.040000	2.377200	83.480000

In [29]:
```
dF_after2012[cL].plot(subplots=True, figsize=(20,20))
```

Out[29]:
```
array([<matplotlib.axes.AxesSubplot object at 0x7fad53759a90>,
       <matplotlib.axes.AxesSubplot object at 0x7fad540911d0>,
       <matplotlib.axes.AxesSubplot object at 0x7fad536f5050>,
       <matplotlib.axes.AxesSubplot object at 0x7fad54503690>,
       <matplotlib.axes.AxesSubplot object at 0x7fad536e28d0>,
       <matplotlib.axes.AxesSubplot object at 0x7fad54090510>], dtype=obje
ct)
```

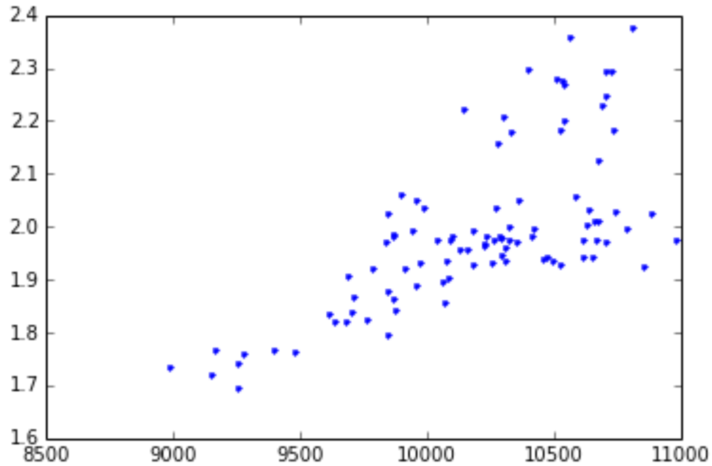


In [30]:
```
dF_after2012.corrwith(dF_after2012['Oil_Price'])
```

Out[30]:
```
LIB_Price   -0.193822
OIS_Price   -0.116396
Oil_Price    1.000000
EIndex       0.592772
Yield        0.685879
TWI         -0.473844
dtype: float64
```

In [31]:
```
X=dF_after2012['Oil_Price']
y=dF_after2012['Yield']
plt.plot(X,y,'.')
```

Out[31]: [<matplotlib.lines.Line2D at 0x7fad5418c790>]



I am not sure that I can develop a good solution to this problem (see below - cointegration). However, I will fit some linear models to this data.

In [32]:
```
from sklearn import cross_validation
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression, Ridge
```

A simple 3 degree poylnomial fitted to the 2012 data

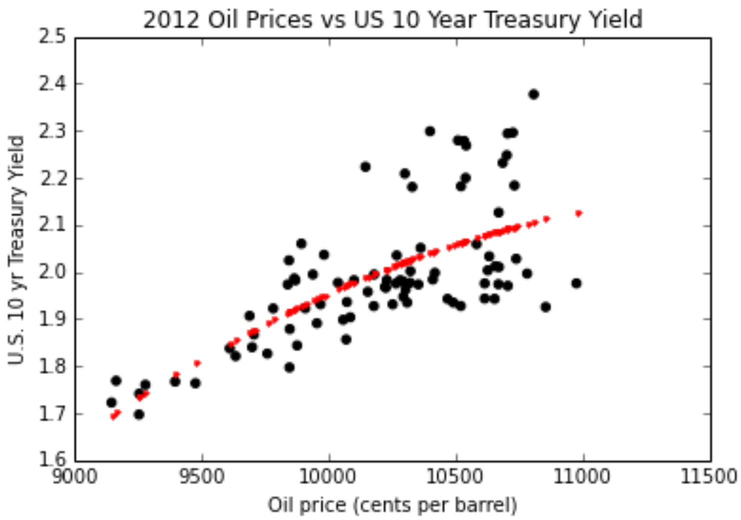
In [33]:
```
theX=X.values
theX.shape
theX = theX[:,np.newaxis]
print theX.shape
the_y=y.values
the_y = the_y[:,np.newaxis]
print the_y.shape
```

```
(96, 1)
(96, 1)
```

In [34]:
```
X_train, X_test, y_train, y_test = train_test_split(theX, the_y, test_siz
e=10)
```

In [35]:
```
est = make_pipeline(PolynomialFeatures(2), LinearRegression())
est.fit(X_train, y_train)
ax=plt.gca()
ax.scatter(X_train, y_train, color='black')
ax.plot(X_train, est.predict(X_train), '.', color='red')
ax.set_ylabel('U.S. 10 yr Treasury Yield')
ax.set_xlabel('Oil price (cents per barrel)')
ax.set_title('2012 Oil Prices vs US 10 Year Treasury Yield')
```

Out[35]:
```
<matplotlib.text.Text at 0x7fad50125d10>
```



Make a prediction:

In [36]:
```
X_val=[8900, 10000.]
X_val=np.array(X_val)
X_val = X_val[:,np.newaxis]
print "x val shape is ", X_val.shape
print "x values are ", X_val
Y_val=est.predict(X_val)
Y_val
```

```
x val shape is  (2, 1)
x values are  [[  8900.]
 [ 10000.]]
```

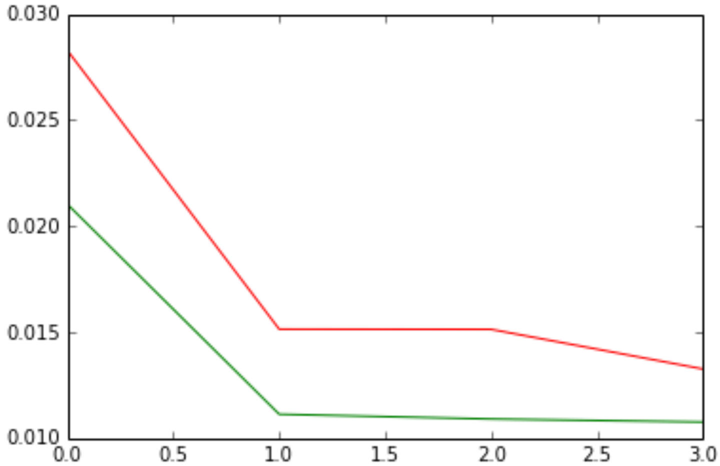
Out[36]:
```
array([[ 1.6028574],
       [ 1.9557705]])
```

Try a Ridge Regression model:

In [37]:
```
from sklearn.metrics import mean_squared_error
theOrder=4
al=1.0e-05
train_error=np.empty(theOrder)
test_error=np.empty(theOrder)
for degree in range(theOrder):
    est = make_pipeline(PolynomialFeatures(degree), Ridge(alpha=al))
    est.fit(X_train,y_train)
    train_error[degree] = mean_squared_error(y_train,est.predict(X_train)
)
    test_error[degree] = mean_squared_error(y_test, est.predict(X_test))
```

In [38]:
```
plt.plot(np.arange(theOrder), train_error, color='green', label='train')
plt.plot(np.arange(theOrder), test_error, color='red', label='test')
```

Out[38]: [<matplotlib.lines.Line2D at 0x7fad500986d0>]

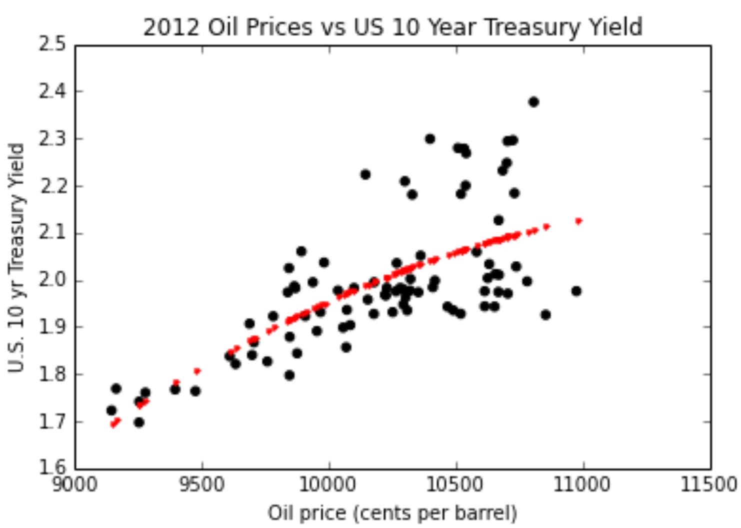


In [39]:
```
est.steps[-1][1].coef_.ravel()
```

Out[39]:
```
array([  0.00000000e+00,  -3.44750579e-02,   3.53340802e-06,
        -1.19650121e-10])
```

In [40]:
```
est = make_pipeline(PolynomialFeatures(2), Ridge(alpha=al))
est.fit(X_train, y_train)
ax=plt.gca()
ax.scatter(X_train, y_train, color='black')
ax.plot(X_train, est.predict(X_train), '.', color='red')
ax.set_ylabel('U.S. 10 yr Treasury Yield')
ax.set_xlabel('Oil price (cents per barrel)')
ax.set_title('2012 Oil Prices vs US 10 Year Treasury Yield')
```

Out[40]: <matplotlib.text.Text at 0x7fad4aa29d90>



In [41]:
```
X_val=[8900, 10000.]
X_val=np.array(X_val)
X_val = X_val[:,np.newaxis]
print "x val shape is ", X_val.shape
print "x values are ", X_val
Y_val=est.predict(X_val)
Y_val
```

```
x val shape is  (2, 1)
x values are  [[  8900.]
 [ 10000.]]
```

Out[41]:
```
array([[ 1.6028574],
       [ 1.9557705]])
```

Let's try out some statsmodels models.

In [45]:
```
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

In [63]:
```
dataf=pd.DataFrame(X_train)
dataf['y']=y_train
dataf.columns=['X','y']
```

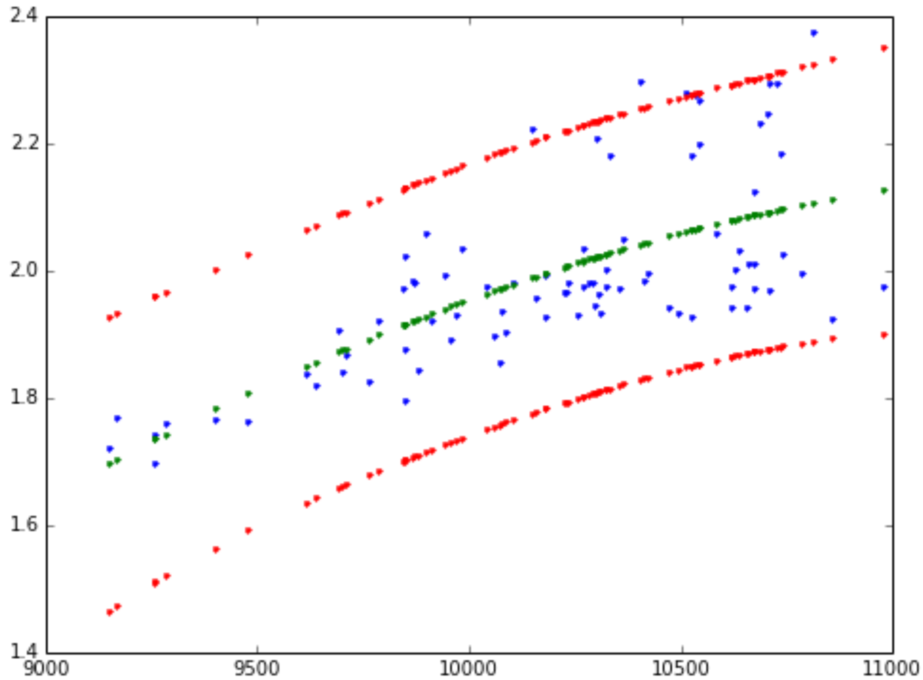
In [64]:
```
res=smf.ols(formula="y ~ 1 + X + I(X ** 2)",data=dataf).fit()
print(res.summary())
```

```
                            OLS Regression Results

==========================================================================
====
Dep. Variable:                      y   R-squared:                       0
.482
Model:                            OLS   Adj. R-squared:                  0
.469
Method:                 Least Squares   F-statistic:                     3
8.55
Date:                Wed, 15 Oct 2014   Prob (F-statistic):           1.44
e-12
Time:                        01:10:53   Log-Likelihood:                 72
.263
No. Observations:                  86   AIC:                            -1
38.5
Df Residuals:                      83   BIC:                            -1
31.2
Df Model:                           2

==========================================================================
====
                 coef    std err          t      P>|t|      [95.0% Conf. I
nt.]
--------------------------------------------------------------------------
----
Intercept     -7.4663      5.425     -1.376      0.172       -18.256     3
.324
X              0.0016      0.001      1.523      0.132        -0.001     0
.004
I(X ** 2)  -6.982e-08   5.34e-08     -1.307      0.195     -1.76e-07  3.64
e-08
==========================================================================
====
Omnibus:                        9.324   Durbin-Watson:                   2
.312
Prob(Omnibus):                  0.009   Jarque-Bera (JB):                9
.975
Skew:                           0.834   Prob(JB):                      0.0
0682
Kurtosis:                       2.960   Cond. No.                     4.97
e+10
==========================================================================
====

Warnings:
[1] The condition number is large, 4.97e+10. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

In [65]:
```
from statsmodels.sandbox.regression.predstd import wls_prediction_std
```

In [66]:
```
prstd, iv_l, iv_u = wls_prediction_std(res)
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(X_train, y_train, '.', label='data')
ax.plot(X_train, res.fittedvalues, '.', label='OLS')
ax.plot(X_train, iv_u, '.', color='red')
ax.plot(X_train, iv_l, '.', color='red')
```

Out[66]:
```
[<matplotlib.lines.Line2D at 0x7fad4939f190>]
```



Consider different data ranges. Also remove outliers from the "Yield" axis.

In [69]:
```
dttm = pd.datetime(2012,1,1)
dttm2 = pd.datetime(2013,12,30)
```

Outlier threshold = above (75 percent quantile + 1.5 * Inter Quartile Range) Outlier threshold = below (25 percent quantile - 1.5 * Inter Quartile Range)

In [70]:
```
quant75=dFCFinal['Yield'][dttm:dttm2].quantile(0.75)
print "quant 75 is ", quant75
quant25=dFCFinal['Yield'][dttm:dttm2].quantile(0.25)
print "quant 25 is ", quant25
q75,q25 = np.percentile(dFCFinal['Yield'][dttm:dttm2],[75,25])
theIQR=q75-q25
print "the interquartile range is ", theIQR
outTopThresh = quant75 + (1.5 * theIQR)
print "the top threshold is ", outTopThresh
outBotThresh = quant25 - (1.5 * theIQR)
print "the bot threshold is ", outBotThresh

rdFCFinal = dFCFinal[dFCFinal['Yield']<outTopThresh][outBotThresh<dFCFina
l['Yield']][dttm:dttm2]
#rdFCFinal = dFCFinal[:][dttm:dttm2]
```

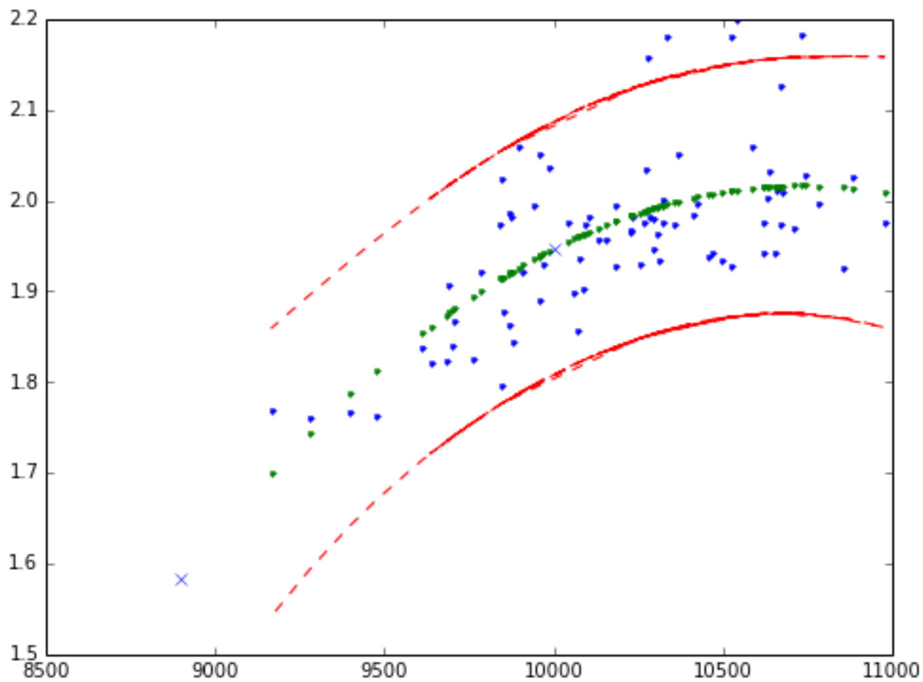
```
quant 75 is  2.0336
quant 25 is  1.9227
the interquartile range is  0.1109
the top threshold is  2.19995
the bot threshold is  1.75635
```

In [71]:
```
res=smf.ols(formula="Yield ~ 1 + Oil_Price + I(Oil_Price ** 2)",data=rdFC
Final[dttm:dttm2]).fit()
print res.summary()
```

```
                            OLS Regression Results
==============================================================================
====
Dep. Variable:                  Yield   R-squared:                       0
.471
Model:                            OLS   Adj. R-squared:                  0
.457
Method:                 Least Squares   F-statistic:                     3
4.22
Date:                Wed, 15 Oct 2014   Prob (F-statistic):           2.33
e-11
Time:                        01:14:25   Log-Likelihood:                 10
1.62
No. Observations:                  80   AIC:                            -1
97.2
Df Residuals:                      77   BIC:                            -1
90.1
Df Model:                           2

==============================================================================
===========
                        coef    std err          t      P>|t|      [95.0%
Conf. Int.]
------------------------------------------------------------------------------
-----------
Intercept           -12.8713      4.250     -3.029      0.003       -21.33
4    -4.409
Oil_Price             0.0028      0.001      3.308      0.001         0.00
1     0.004
I(Oil_Price ** 2) -1.293e-07   4.13e-08     -3.127      0.002    -2.12e-0
7  -4.7e-08
==============================================================================
====
Omnibus:                       15.412   Durbin-Watson:                   0
.385
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17
.328
Skew:                           1.071   Prob(JB):                     0.00
0173
Kurtosis:                       3.779   Cond. No.                     5.77
e+10
==============================================================================
====

Warnings:
[1] The condition number is large, 5.77e+10. This might indicate that ther
e are
strong multicollinearity or other numerical problems.
```

In [72]:
```
prstd, iv_l, iv_u = wls_prediction_std(res)
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(rdFCFinal['Oil_Price'][dttm:dttm2], rdFCFinal['Yield'][dttm:dttm2
], '.', label='data')
ax.plot(rdFCFinal['Oil_Price'][dttm:dttm2], res.fittedvalues, '.', label=
'OLS')
ax.plot(rdFCFinal['Oil_Price'][dttm:dttm2], iv_u, '--', color='red')
ax.plot(rdFCFinal['Oil_Price'][dttm:dttm2], iv_l, '--', color='red')
#ax.plot(rdFCFinal['Oil_Price'][dttm:dttm2], prstd, '--', color='red')
ax.plot(8900,1.5834, "x", color='blue')
ax.plot(10000,1.9479,"x", color='blue')
```

Out[72]: [<matplotlib.lines.Line2D at 0x7fad49330ed0>]



In [73]:
```
X_val=[8900, 10000.]
X_val=np.array(X_val)
X_val = X_val[:,np.newaxis]
X_val = np.column_stack(X_val)
print "x val shape is ", X_val.shape
print "x values are ", X_val
print res.predict(exog=dict(Oil_Price=8900.0))
print res.predict(exog=dict(Oil_Price=10000.0))
print res.mse_model
print res.mse_resid
```

```
x val shape is  (1, 2)
x values are  [[  8900.  10000.]]
[ 1.58338847]
[ 1.94796852]
0.164058167762
0.00479458821397
```

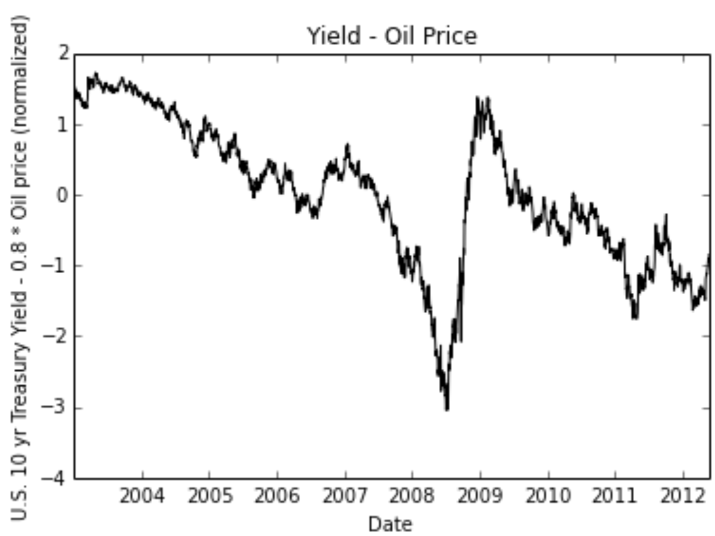
Fascinating post referring to the magazine, The Economist, issue around Feb 2007. The blog was commenting on the article suggesting a link between oil price and US treasury yield, and a graph that the blogger felt was misleading. Rationale: High oil prices mean OPEC has spare cash (no need to prop oil prices up), therefore they buy US treasurys and hence yield falls. This brought up the issue of cointegration. http://epchan.blogspot.com/2007/02/cointegration-between-oil-and-bond.html The following from Wikipedia: The R-squared statistic has been shown to give misleading results when used. Time series can have stochastic trends. De-trending does NOT elimate spurious regression. Cointegration is a measure of the relationship between two time series when regression resulting from stochastic trends has been removed.

So before running a regression between two time series the following process should be observed (?!): 1. Check to see if each time series is a "unit root process". A unit root process means the time series does have a stochastic trend, and as such regressing on 2 such series may well reveal spurious correlations. 2. Cointegration of 2 unit root processes will indicate the true underlying relationship after accounting for any stochastic trends.

I firstly wanted to recreate the graph that appeared on the blog.

In [74]:
```
dFyieldoilprice = dFCFinal['Yield']-0.8 * dFCFinal['Oil_Price']
dFyieldoilprice = (dFyieldoilprice - dFyieldoilprice.mean())/dFyieldoilpr
ice.std()
#dFyieldoilprice.plot(color='black')
ax1=plt.gca()
ax1.plot(dFyieldoilprice.index,dFyieldoilprice, color='black')
ax1.set_ylabel('U.S. 10 yr Treasury Yield - 0.8 * Oil price (normalized)'
)
ax1.set_xlabel('Date')
ax1.set_title('Yield - Oil Price')
```

Out[74]: <matplotlib.text.Text at 0x7fad492c5150>

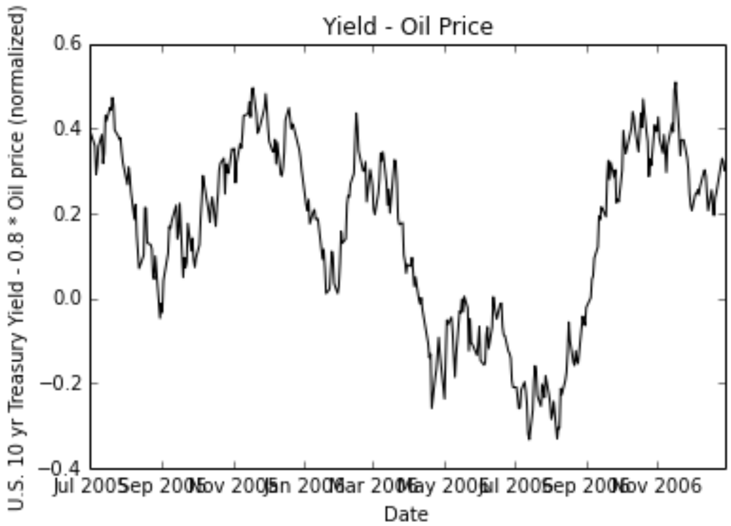


The author of the blog suggested that cointegration could be seen in his graph between the time period mid-2005 through to the end of 2006. I DON'T UNDERSTAND HOW HE MAKES THIS ASSERTION FROM THE GRAPH. I might assume that if the graph is roughly horizontal then the 2 time series are moving together. NOT SURE WHY THIS IS COINTEGRATION.

Zooming in on the period in question:

In [75]:
```
dFyo20052006=dFyieldoilprice[pd.datetime(2005,6,30):pd.datetime(2006,12,3
1)]
ax1=plt.gca()
ax1.plot(dFyo20052006.index,dFyo20052006, color='black')
ax1.set_ylabel('U.S. 10 yr Treasury Yield - 0.8 * Oil price (normalized)'
)
ax1.set_xlabel('Date')
ax1.set_title('Yield - Oil Price')
```

Out[75]: <matplotlib.text.Text at 0x7fad49188390>



I then explored some of the statsmodels tools, most notably, "adfuller" which is meant to test for a unit root process, and "coint" which a test for coinegration.

In [76]:
```
from statsmodels.tsa.stattools import adfuller
#adfuller if the Augmented Dickey-Fuller test used to test for a unit roo
t in a univariate process
```

Try the adfuller routine on the oil price series:

In [77]:
```
adfarray=np.array(dFCFinal['Oil_Price'].values)
print adfarray.shape
print adfarray.ndim
#Null hypothesis of adfuller is that there is a unit root
adfull=adfuller(adfarray, autolag='t-stat')
adfull
#the pvalue is 0.2986 implying that the null hypothesis cannot be rejecte
d. Cannot reject there is a unit root.
```

```
(2294,)
1
```

Out[77]:
```
(-1.9727850106539688,
 0.29863190365868308,
 24,
 2269,
 {'1%': -3.433235285765301,
  '10%': -2.5674485535435454,
  '5%': -2.8628146468918052},
 2.3799471509582437)
```

Try the adfuller routine on the Yield series:

In [78]:
```
adfarray1=np.array(dFCFinal['Yield'].values)
adfull1=adfuller(adfarray1, autolag='t-stat')
adfull1
#the pvalue is 0.6926 again implying that one cannot reject a unit root
```

Out[78]:
```
(-1.1549080723402756,
 0.6926863503312527,
 13,
 2280,
 {'1%': -3.4332213498074471,
  '10%': -2.5674452772006773,
  '5%': -2.8628084932982079},
 1.8506600457212825)
```

Let's investigate over the time-scale mid-2005 till end of 2005:

In [79]:
```
adfarray2 = np.array(dFCFinal['Oil_Price'][pd.datetime(2005,6,30):pd.date
time(2006,12,30)].values)
adfull2=adfuller(adfarray2, autolag='t-stat')
#pvalue is 0.15
adfull2
```

Out[79]:
```
(-2.3567615262964963,
 0.15431373216025335,
 0,
 368,
 {'1%': -3.4482453822848496,
  '10%': -2.5709711770439507,
  '5%': -2.8694261442901396},
 2.1816164089031909)
```

In [80]:
```
adfarray3 = np.array(dFCFinal['Yield'][pd.datetime(2005,6,30):pd.datetime
(2006,12,30)].values)
adfull3=adfuller(adfarray3, autolag='t-stat')
#pvalue is 0.38
adfull3
```

Out[80]:
```
(-1.7977952894790643,
 0.38155681927215979,
 10,
 358,
 {'1%': -3.4487489051519011,
  '10%': -2.5710891239349585,
  '5%': -2.8696473721448728},
 1.8908266754292749)
```

Both these time series appear to be unit root processes over the time period in question.

Now turning to measuring cointegration:

In [81]:
```
from statsmodels.tsa.stattools import coint
#the Null Hypothesis is that there is NO cointegration.
adcoint = coint(adfarray,adfarray1,regression='c')
adcoint
#p value is 0.39 so cannot reject H0. There is NO cointegration of these
2 series over the full time interval
```

Out[81]:
```
(-2.2656359438490559,
 0.39081955970389959,
 array([-3.43320381, -2.86280075, -2.56744115]))
```

In [82]:
```
adcoint = coint(adfarray2,adfarray3,regression='c')
adcoint
#The pvalue has dropped, but is still large
```

Out[82]:
```
(-2.9149533589523728,
 0.13182979117299748,
 array([-3.44819654, -2.86940468, -2.57095974]))
```

Let's finally examine over the time period of 2012 only.

In [83]:
```
adfarray4=np.array(dF_after2012['Oil_Price'].values)
adfull4=adfuller(adfarray4, autolag='t-stat')
#pvalue is 0.71
adfull4
```

Out[83]:
```
(-1.1085027243346215,
 0.71166923290251327,
 6,
 89,
 {'1%': -3.506057133647011,
  '10%': -2.5844100201994697,
  '5%': -2.8946066061911946},
 2.2326180520851207)
```

In [84]:
```
adfarray5=np.array(dF_after2012['Yield'].values)
adfull5=adfuller(adfarray5, autolag='t-stat')
#pvalues is 0.73
adfull5
```

Out[84]:
```
(-1.0451001337399153,
 0.73646190176678128,
 5,
 90,
 {'1%': -3.5051901961591221,
  '10%': -2.5842101234567902,
  '5%': -2.894232085048011},
 1.8291474087957169)
```

In [85]:
```
adcoint4 = coint(adfarray4,adfarray5,regression='c')
adcoint4
#pvalue is 0.66
```

Out[85]:
```
(-1.7271208700812819,
 0.6643997211840349,
 array([-3.50037889, -2.89215197, -2.5830998 ]))
```

This implies there is limited(?), no(?), cointegration between oil price and yield, which suggests the linear regression performed about is of hghly dubious in value.