In [1]:
```
import pandas as pd
import numpy as np
import datetime as dt
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import statsmodels.api as sm
import re
```

Task 1: Palindromic Function

Given a list of words, write a function to see if they are palindromic. Example words: “Hannah”, “Bob”, “Tony”, “Sally” You need to return a list of True or False.

In [2]:
```
def get_palindromic(words):
    return [w.lower()==w.lower()[::-1] for w in words]
'''
Test Case 1
words = ['Hannah', 'Bob', 'Tony', 'Sally']
Test Case 2
words = []
Test Case 3
words = ['45&54','D^&beD']
'''
words = ['Hannah', 'Bob', 'Tony', 'Sally']
get_palindromic(words)
```

Out[2]:
```
[True, True, False, False]
```

Task 3: Fibonacci Numbers

Each new value in a Fibonacci series is calculated by summing the previous two values, e.g. 1, 2, 3, 5 and so on. Write a function that returns the sum of odd-valued terms where those values do not exceed 10 million.

In [3]:
```
def sum_odd_fibonacci(limit):
    s = 0
    n0, n1 = 0,1
    while True:
        n0, n1 = n1, n0 + n1
        if n1 >= limit:
            break
        if n1%2 == 1:
            s += n1
    return s

'''
Test Case 1
sum_odd_fibonacci(1)
Test Case 2
sum_odd_fibonacci(5)
'''
print("sum of odd-valued terms where those values do not exceed 10 million is:",su
m_odd_fibonacci(1e7))
```

```
sum of odd-valued terms where those values do not exceed 10 million is: 19544083
```

Task 2: Integrating Alternative Data

An analyst comes to you and wants to understand if the transaction data can be used to forecast stock price movements

Using a modelling package of your choice, prove out whether the transaction data is predictive for MCD (using the closing price data provided)

Present your results, this can be done using any visualisation package of your choice

Write a short commentary

What other datasets do you think could be included/blended to enhance your analysis?

Data Processing

Load daily credit transaction price and volume data for differnet brands/companies, aggregated transaction volume data across the panel, as well as close prices for MCD. Data looks to be daily but needs further testing for us to align the sources

In [4]:
```
#load credit transaction data
xls = pd.ExcelFile('Excel_Credit_Card_Data.xlsx')
transaction_daily = pd.read_excel(xls, 'Daily Data')
transaction_panel = pd.read_excel(xls, 'Total Panel Information')

#load MCD market return data
price_mcd = pd.read_csv('Coding_test_data.csv')
```

Exploratory Data Analysis

PRICE_MCD

start with mcdonald price data since it is our target variable

1. There appears to be duplicate date
2. Null Prices is represented unconventionaly as False
3. We have 2.5 year worth of data, which is relatively short term.

In [5]:
```
print(price_mcd.describe())
price_mcd.Date = pd.to_datetime(price_mcd['Date'])
price_mcd.set_index('Date',inplace=True)
print(price_mcd.describe())
print('Date Range',price_mcd.index.min(),price_mcd.index.max())
```

```
                   Date Close Price
count               905         905
unique              899         513
top     26 October 2014       FALSE
freq                  2         258
       Close Price
count          905
unique         513
top          FALSE
freq           258
Date Range 2012-06-21 00:00:00 2014-12-31 00:00:00
```

confirm duplicate index does not produce multiple distinct values and drop them

In [6]:
```
price_mcd[price_mcd.index.duplicated()]
```

Out[6]:

Date	Close Price
2014-10-26	FALSE
2014-10-27	92.01
2014-10-28	92.6
2014-10-29	92.73
2014-10-30	93.38
2014-10-31	93.73

In [7]:
```
price_mcd = price_mcd[~price_mcd.index.duplicated()]
```

~30% missing data, this is likely due to weekends and market holidays (or rare market events).

However, it is possible that if the date index is missing, the missing rate would not infer that

In [8]:
```
price_mcd[price_mcd['Close Price'] == 'FALSE'].count() / 899
```

Out[8]:
```
Close Price    0.285873
dtype: float64
```

Missing ~20 days of data, this can potentially be an data error

In [9]:
```
import numpy as np
start = dt.date( 2012, 6, 21 )
end = dt.date( 2014, 12, 31 )
days = np.busday_count( start, end )
print(days)
```

```
659
```

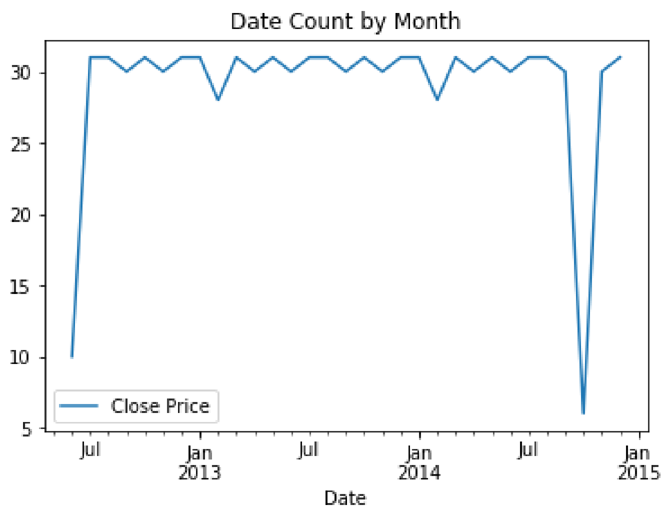
In [10]:
```
price_mcd[price_mcd['Close Price'] != 'FALSE'].count()
```

Out[10]:
```
Close Price    642
dtype: int64
```

to verfiy this, we plot monthly count of date index, looks like a large portion of data for 2014-10 is missing, for simplicity we will drop the missing period for later analysis since the grap exists in both market price and credit card transaction data

In [11]:
```
price_mcd.groupby(pd.Grouper(freq='M')).count().plot()
plt.title('Date Count by Month')
```

Out[11]:
```
Text(0.5, 1.0, 'Date Count by Month')
```



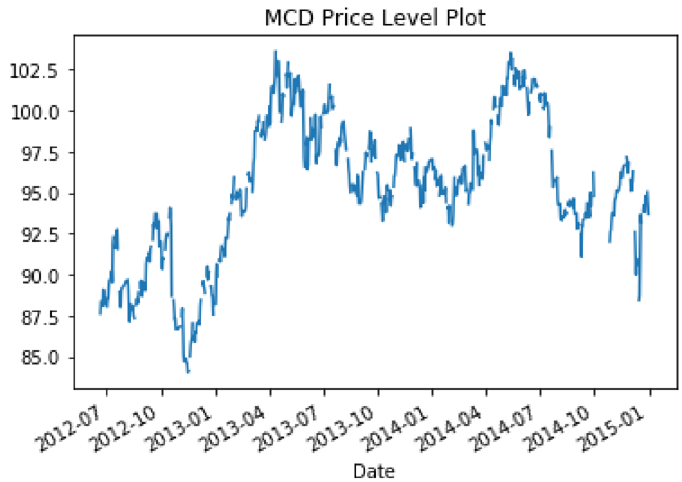
Price Level

visually assesss normality of price level,in case we need to apply transformation for any normality assumptions

In [12]:
```
price_mcd['Close Price'] = price_mcd['Close Price'].apply(lambda x: None if x =='F
ALSE' else float(x))
```

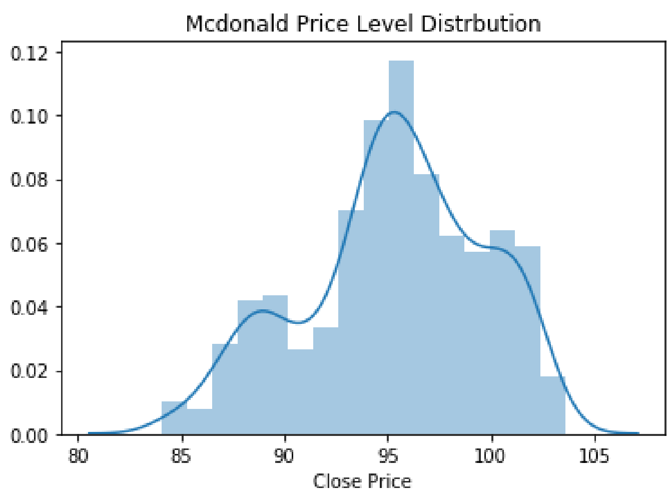
In [13]:
```
price_mcd['Close Price'].plot()
plt.title('MCD Price Level Plot')
```

Out[13]: Text(0.5, 1.0, 'MCD Price Level Plot')

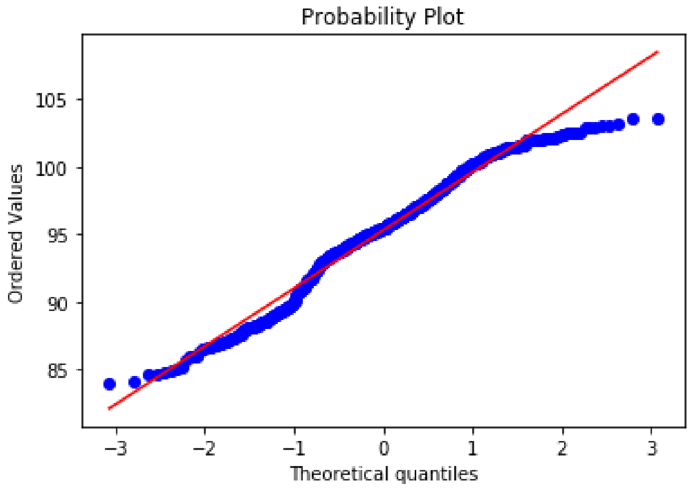


In [14]:
```
sns.distplot(price_mcd['Close Price'].dropna())
print("Skewness: %f" % price_mcd['Close Price'].skew())
print("Kurtosis: %f" % price_mcd['Close Price'].kurt())
plt.title('Mcdonald Price Level Distrbution')
plt.show()
```

```
Skewness: -0.347842
Kurtosis: -0.507665

/Users/charles-18/anaconda3/lib/python3.7/site-packages/scipy/stats/stats.py:171
3: FutureWarning: Using a non-tuple sequence for multidimensional indexing is dep
recated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will be
interpreted as an array index, `arr[np.array(seq)]`, which will result either in
an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



In [15]:
```
res = stats.probplot(price_mcd['Close Price'].dropna(),plot = plt)
plt.show()
```

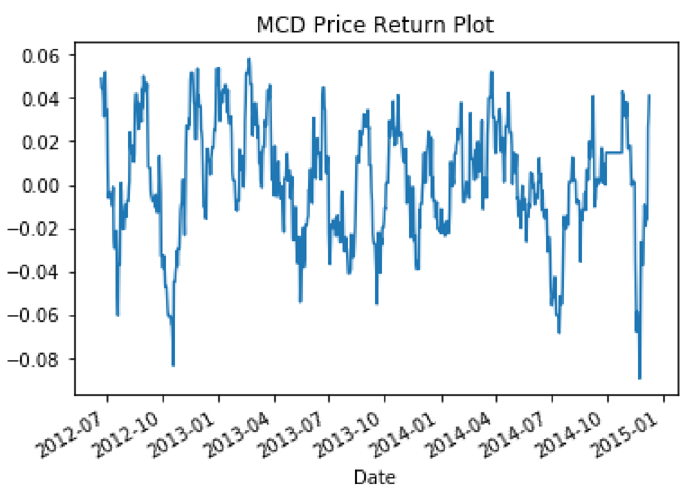


Price return

We are also interested to see return data since level data is generally not stationary and exibits a certain degree of auto correlation.omit dicker fuller test here since time series anaylsis is not the focus. we use 1-month lookahead return.The return looks stationary and exibits strong reversal behaviour

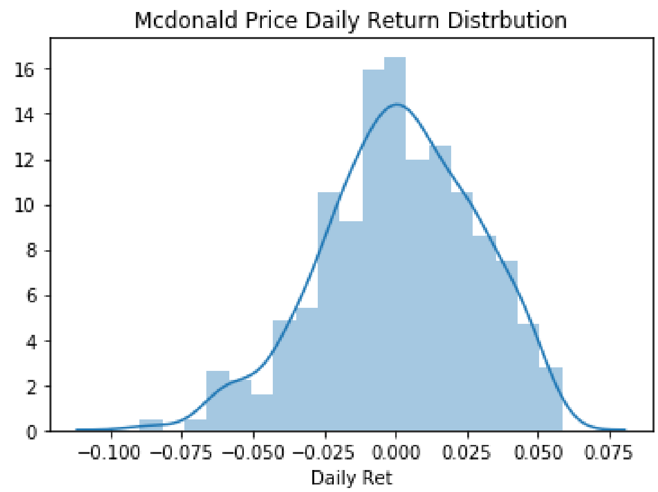
In [16]:
```
price_mcd['Daily Ret'] = price_mcd['Close Price'].pct_change(21).shift(-21)
price_mcd['Daily Ret'].plot()
plt.title('MCD Price Return Plot')
```

Out[16]:
```
Text(0.5, 1.0, 'MCD Price Return Plot')
```

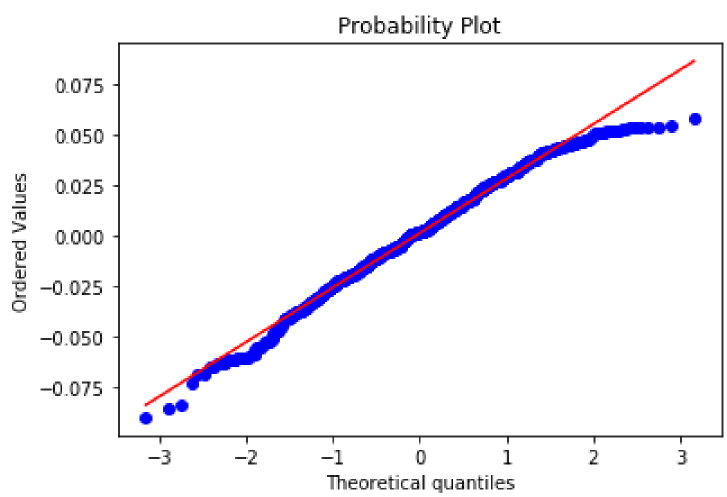


In [17]:
```
sns.distplot(price_mcd['Daily Ret'].dropna())
print("Skewness: %f" % price_mcd['Daily Ret'].skew())
print("Kurtosis: %f" % price_mcd['Daily Ret'].kurt())
plt.title('Mcdonald Price Daily Return Distrbution')
plt.show()
```

```
Skewness: -0.314500
Kurtosis: -0.147144
```



In [18]:
```
fig = plt.figure()
res = stats.probplot(price_mcd['Daily Ret'].dropna(),plot = plt)
plt.show()
```



Transaction Daily

from the data summary:

1. Same date range as MCD market price data
2. There are more merchants than companies, which may indicate companies own different brands. A filter on company should take higher priority than merchant, since market price generally embeds/refect aggregated company-level info
3. Both transaction value and volume data are available
4. There are transaction count of 2 and negative transaction value, these could potentailly be outliers. We will filter out non-MC related companies and perform in-depth analysis

In [19]:
```
for col in transaction_daily.columns:
    print(transaction_daily[col].describe())
```

```
count                    35791
unique                     899
top        2014-10-26 00:00:00
freq                        80
first      2012-06-21 00:00:00
last       2014-12-31 00:00:00
Name: TRANSACTION_DATE, dtype: object
count         35791
unique           23
top        YUM-USAA
freq           5931
Name: COMPANY, dtype: object
count       35791
unique         40
top        Carvel
freq          905
Name: MERCHANT, dtype: object
count     35791.000000
mean       8603.883742
std       18424.202775
min           2.000000
25%         910.000000
50%        3020.000000
75%        7954.000000
max      188184.000000
Name: TRANSACTION_COUNT, dtype: float64
count    3.579100e+04
mean     9.023590e+04
std      1.605152e+05
min     -6.039529e+04
25%      8.066316e+03
50%      3.184630e+04
75%      1.015911e+05
max      1.469060e+06
Name: TRANSACTED_VALUE, dtype: float64
```

find brand that corresponds to Mcdonald

In [20]:
```
mcd_company = [i for i in set(transaction_daily['COMPANY']) if (('mcd' in i.lower
()) or ('mcdonald' in i.lower()))]
print(mcd_company)
```

```
['MCD-USAA']
```

In [21]:
```
tran_mcd = transaction_daily[transaction_daily.COMPANY == mcd_company[0]]
```

vefify there is only one merchant under MCD

In [22]:
```
len(set(tran_mcd.MERCHANT))
```

Out[22]:
```
1
```

In [23]:
```
for col in tran_mcd.columns:
    print(tran_mcd[col].describe())
```

```
count                      905
unique                     899
top        2014-10-31 00:00:00
freq                         2
first      2012-06-21 00:00:00
last       2014-12-31 00:00:00
Name: TRANSACTION_DATE, dtype: object
count           905
unique            1
top        MCD-USAA
freq            905
Name: COMPANY, dtype: object
count            905
unique             1
top        McDonalds
freq             905
Name: MERCHANT, dtype: object
count       905.000000
mean      93117.048619
std       43847.347624
min       20976.000000
25%       59340.000000
50%       68952.000000
75%      146788.000000
max      188184.000000
Name: TRANSACTION_COUNT, dtype: float64
count    9.050000e+02
mean     7.013062e+05
std      3.420348e+05
min      2.099331e+05
25%      4.394585e+05
50%      5.271563e+05
75%      1.051377e+06
max      1.469060e+06
Name: TRANSACTED_VALUE, dtype: float64
```

Get rid of duplicates

In [24]:
```
tran_mcd.set_index(['TRANSACTION_DATE','MERCHANT'],inplace = True)
```

In [25]:
```
print(sum(tran_mcd.index.duplicated()))
print(sum(tran_mcd.duplicated()))
```

```
6
0
```

In [26]:
```
tran_mcd = tran_mcd.loc[~tran_mcd.index.duplicated(keep='first')]
tran_mcd = tran_mcd.reset_index()
```

In [27]:
```
tran_mcd.head()
```

Out[27]:

	TRANSACTION_DATE	MERCHANT	COMPANY	TRANSACTION_COUNT	TRANSACTED_VALUE
0	2012-06-21	McDonalds	MCD-USAA	56662	413671.839212
1	2012-06-22	McDonalds	MCD-USAA	59521	455681.487347
2	2012-06-23	McDonalds	MCD-USAA	54730	461245.695234
3	2012-06-24	McDonalds	MCD-USAA	54592	458026.880000
4	2012-06-25	McDonalds	MCD-USAA	52526	383162.014475

There does not seem to be missing data

In [28]:
```
tran_mcd.isnull().sum()
```

Out[28]:
```
TRANSACTION_DATE      0
MERCHANT              0
COMPANY               0
TRANSACTION_COUNT     0
TRANSACTED_VALUE      0
dtype: int64
```

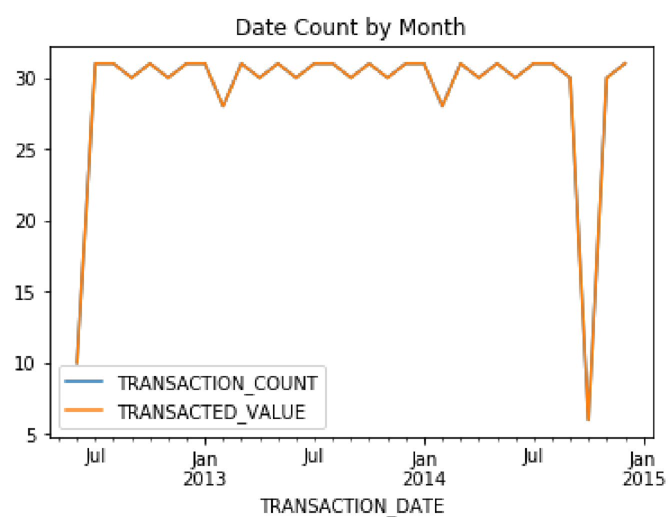
In [29]:
```
tran_mcd.TRANSACTION_DATE = pd.to_datetime(tran_mcd['TRANSACTION_DATE'])
tran_mcd.set_index('TRANSACTION_DATE',inplace=True)
```

In [30]:
```
tran_mcd.drop(columns = ['COMPANY','MERCHANT'],inplace = True)
```

There are missing data in 2014-10, we observed similar behavior in makret price data for mcdonald. It is unlikly MCD suddenly stopped trading or ceased operation for that specific error. Therefore, this error is likely related to data loading script using the wrong datetime parameters

In [31]:
```
tran_mcd.groupby(pd.Grouper(freq='M')).count().plot()
plt.title('Date Count by Month')
```

Out[31]:
```
Text(0.5, 1.0, 'Date Count by Month')
```

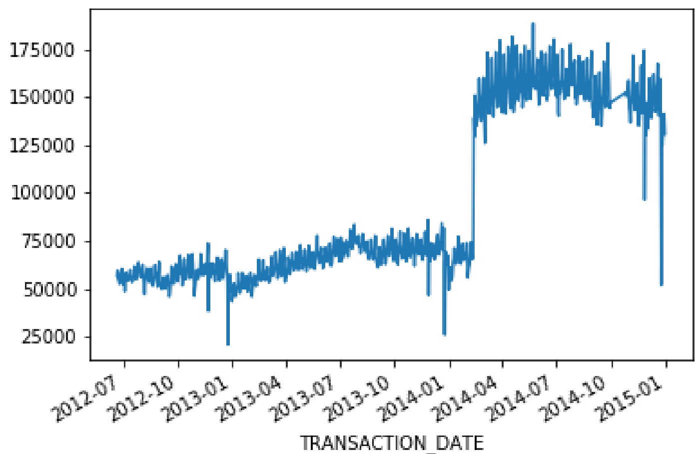


Based on the transaction count and value time plot :

1. There is a jump in both transaction count and value starting 2014 Feburary. This could be caused by:
   1) Data generating process change. If this is the case, we need to ask vendor to provide justification and apply a discount factor to normalize time series by applying the panel normalization factor
   2) Mcdonald doubled its sales by either acquiring another chain of business or aggresively doubling its revenue channel.
2. There appears to be a dip in transaction value and count in the month of Janauray, this is likely related to seasonality, for example, families tend to eat eat at home to celebrate new year.

In [32]:
```
tran_mcd['TRANSACTION_COUNT'].plot()
```

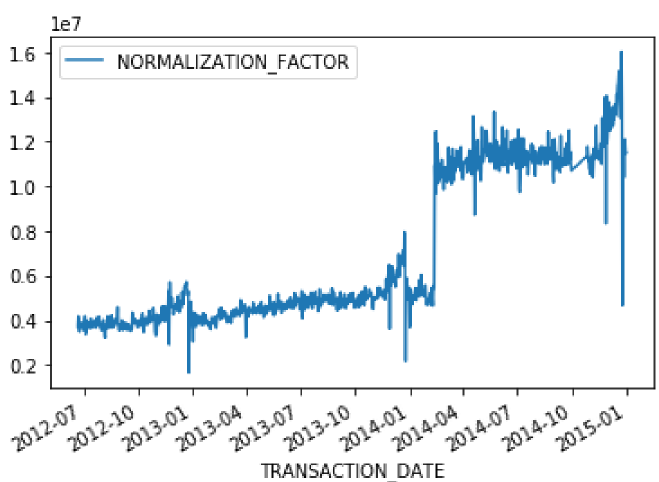
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x1c207b2b38>



To test the hypothesis, we can compare using normalized panel data, it appears that we can rule out 1.2) just by observation

In [33]:
```
transaction_panel['TRANSACTION_DATE'] = pd.to_datetime(transaction_panel['TRANSACT
ION_DATE'])
transaction_panel.set_index('TRANSACTION_DATE',inplace= True)
transaction_panel.plot()
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1c218a9f60>



Apply Normalization factor to mcdonald data

for simplicity, we directly use normalization factor consturcted based transaction count instead of total transaction volumes, also get rid of duplicates

In [34]:
```
transaction_panel = transaction_panel.loc[~transaction_panel.index.duplicated(keep
='first')]
```

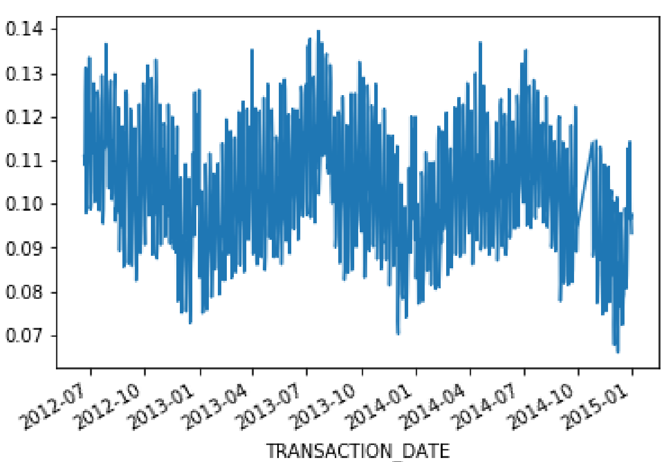
In [35]:
```
tran_mcd['TRANSACTED_VALUE'] = (tran_mcd['TRANSACTED_VALUE'] /transaction_panel['N
ORMALIZATION_FACTOR'] )
```

In [36]:
```
tran_mcd['TRANSACTION_COUNT'] = (tran_mcd['TRANSACTION_COUNT']/transaction_panel[
'NORMALIZATION_FACTOR'])
```

Normalized data has strong seasonality

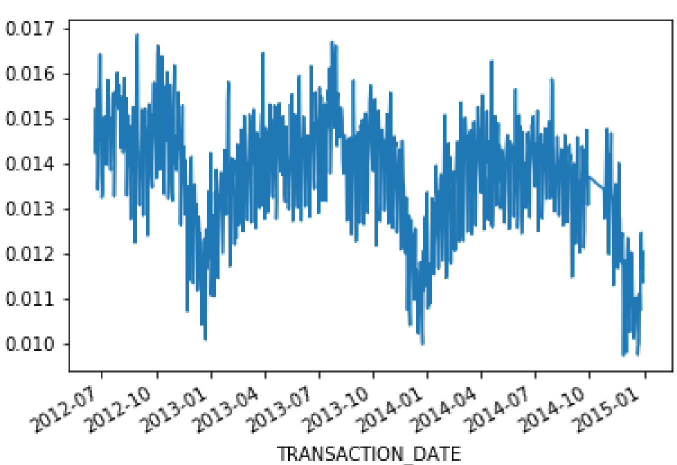
In [37]:
```
tran_mcd.TRANSACTED_VALUE.plot()
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x1c21a68da0>



In [38]:
```
tran_mcd.TRANSACTION_COUNT.plot()
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1c21ba02e8>



Build Data Panel

In [39]:
```
data = pd.merge(price_mcd.drop(columns= ['Daily Ret']),tran_mcd,left_on=price_mcd.
index,right_on=tran_mcd.index)
```

In [40]:
```
data.rename(columns={'key_0':'Date'},inplace=True)
data.set_index('Date',inplace = True)
```

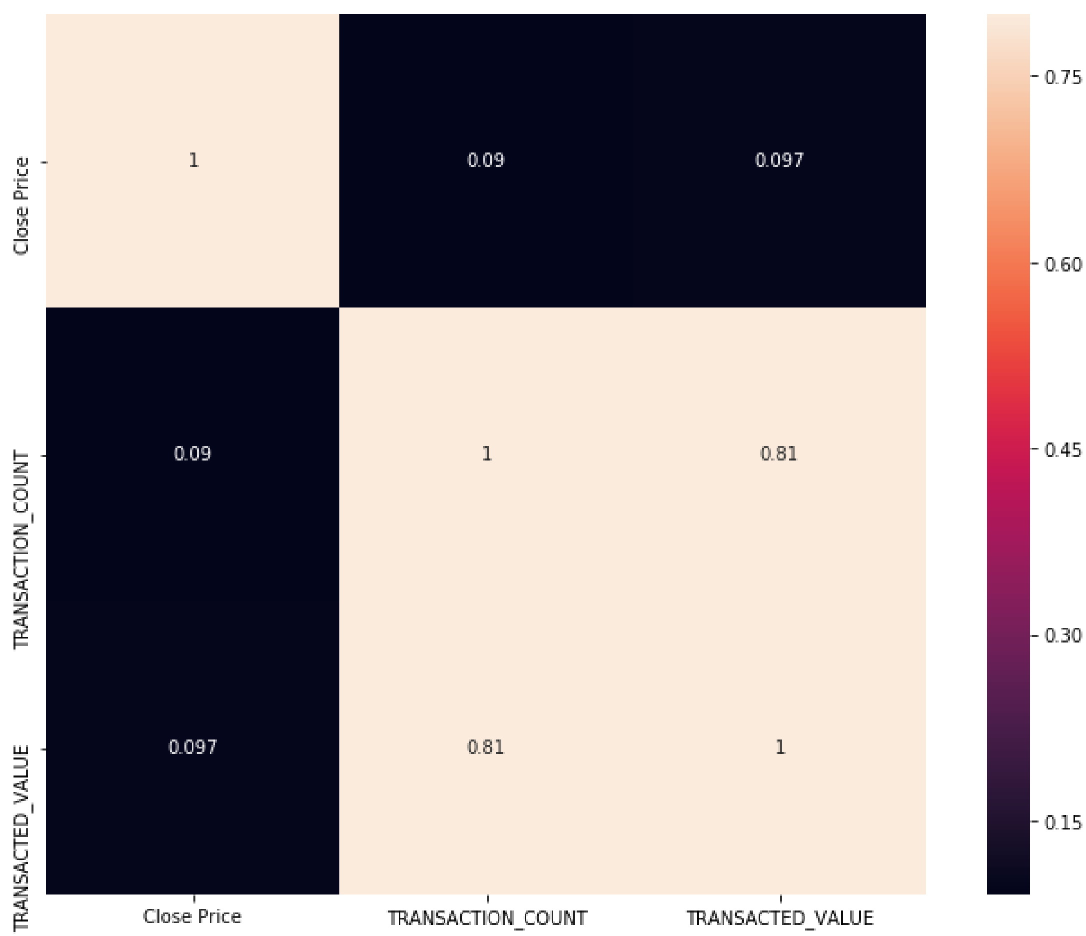
observe missing values and drop them if number of missing rows is not significant (<5%)

In [42]:
```
print((data.isnull().sum()) / len(data))
data = data.dropna()
```

```
Close Price          0.285873
TRANSACTION_COUNT    0.000000
TRANSACTED_VALUE     0.000000
dtype: float64
```

As expected transaction count and trasaction value are higly correlated

In [43]:
```
corrmat = data.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True,annot=True);
plt.show()
```



Feature Engineering

Return data is generally better as a target variable due to stationarity, for simpicity, we pick 5-day,10-day, 1-month,3-month lookahead return as our target variable

In [44]:
```
tenor = {'5d':5,'1m':21,'3m':63}
```

In [45]:
```
for t in tenor.keys():
    data[('Ret_'+t)] = data['Close Price'].pct_change(tenor[t]).shift(-tenor[t])
```

Need to remove outliers from the time series, simply apply a 5-day median filter

In [46]:
```
data['TRANSACTED_VALUE'] = data['TRANSACTED_VALUE'].rolling(5).median()
data['TRANSACTED_VALUE'] = data['TRANSACTED_VALUE'].rolling(5).median()
```

average transaction value can measure the size of each transaction. A higher value indicate each transaciton tends to have a larger dollar value, indicating people's willingness to purchase more expensive items such as meals instead of individual food items.

In [47]:
```
data['TRANSACTED_AVG_VALUE'] = data['TRANSACTED_VALUE'] / data['TRANSACTION_COUNT'
]
```

To adjust for return, we can compute change in transaction count/value to capture the momentum which could lead to lookahead return differences. Here we use percent change of transaction value and count for the past 5-day, 1 month and 3 month transaction momentum

In [48]:
```
for t in tenor.keys():
    data[('TRANSACTED_VALUE_'+t)] = data['TRANSACTED_VALUE'].pct_change(tenor[t])
    data[('TRANSACTION_COUNT_'+t)] = data['TRANSACTION_COUNT'].pct_change(tenor[t
])
    data[('TRANSACTED_AVG_VALUE_'+t)] = data['TRANSACTED_VALUE'].pct_change(tenor[
t])
```

We also compute a rolling sum of total sales value to establish relationship between level data

In [49]:
```
for t in tenor.keys():
    data[('TRANSACTED_VALUE_MV'+t)] = data['TRANSACTED_VALUE'].rolling(tenor[t]).s
um()
    data[('TRANSACTION_COUNT_MV'+t)] = data['TRANSACTION_COUNT'].rolling(tenor[t])
.sum()
    data[('TRANSACTED_AVG_VALUE_MV'+t)] = data['TRANSACTED_VALUE'].rolling(tenor[t
]).sum()
```

log transform skewed numeric features to assume normality

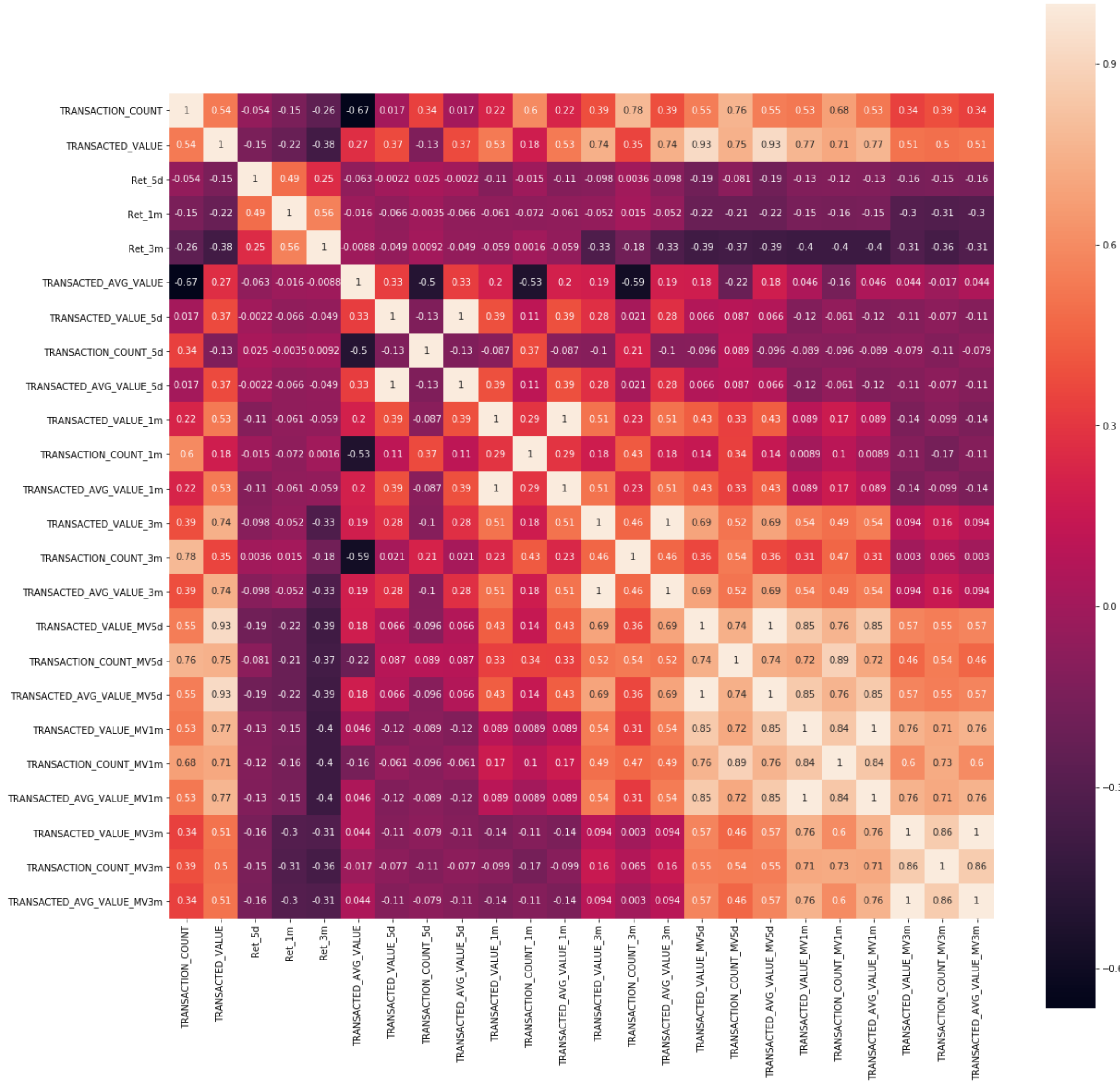
In [50]:
```
numeric_feats = data.dtypes[data.dtypes != "object"].index
# Check the skew of all numerical features
skewed_feats = data[numeric_feats].apply(lambda x: (x.dropna()).skew()).sort_value
s(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
print(skewness)
```

```
Skew in numerical features:

                               Skew
TRANSACTED_AVG_VALUE       0.575978
TRANSACTION_COUNT_3m       0.434584
Ret_3m                     0.371933
TRANSACTION_COUNT_5d       0.371172
TRANSACTED_AVG_VALUE_3m    0.182507
TRANSACTED_VALUE_3m        0.182507
TRANSACTED_AVG_VALUE_MV3m  0.084985
TRANSACTED_VALUE_MV3m      0.084985
TRANSACTED_AVG_VALUE_MV1m  0.046871
TRANSACTED_VALUE_MV1m      0.046871
TRANSACTION_COUNT_1m       0.036149
TRANSACTED_VALUE_5d       -0.076889
TRANSACTED_AVG_VALUE_5d   -0.076889
TRANSACTED_VALUE          -0.157506
TRANSACTED_AVG_VALUE_MV5d -0.157821
TRANSACTED_VALUE_MV5d     -0.157821
TRANSACTED_AVG_VALUE_1m   -0.224342
TRANSACTED_VALUE_1m       -0.224342
Close Price               -0.347842
Ret_1m                    -0.443628
TRANSACTION_COUNT         -0.457377
Ret_5d                    -0.544733
TRANSACTION_COUNT_MV3m    -0.591246
TRANSACTION_COUNT_MV5d    -1.071821
TRANSACTION_COUNT_MV1m    -1.100810
```

In [51]:
```
skewness = skewness[abs(skewness) > 0.4].dropna()
print("There are {} skewed numerical features to Box Cox transform".format(skewnes
s.shape[0]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0
for feat in skewed_features:
    data[feat] += 1
    data[feat] = boxcox1p(data[feat], lam)
```

```
There are 8 skewed numerical features to Box Cox transform
```

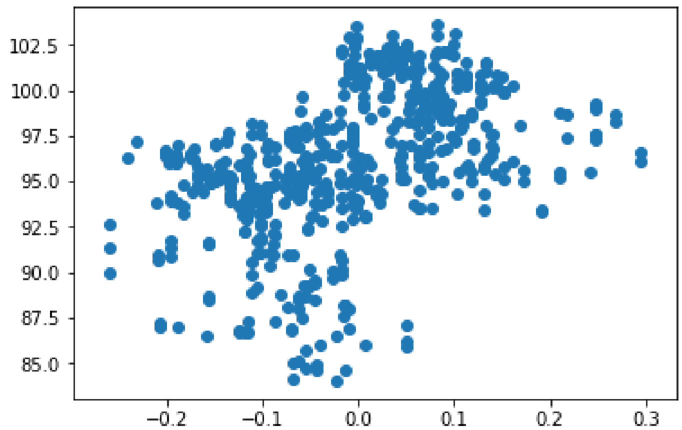
In [52]:
```
corrmat = data.drop(columns =['Close Price']).corr()
f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(corrmat, vmax=1, square=True,annot=True);
plt.show()
```



identified some highly correlated features, need to remove redundant features to avoid multicoliearity

In [53]:
```
plt.scatter(data['TRANSACTED_AVG_VALUE_3m'],data['Close Price'])
```

Out[53]: <matplotlib.collections.PathCollection at 0x1c2322d4a8>



Model Building - Linear Model

In Sample Regression

Hypothesis: sales momentum predicts lookahead market returns , this can be modeled as a linear relationship. Specifically we think sales value and volume should be positive indicators of future returns. A 3-month (quaterly) sales revision should lead to market return changes

$$R(t, t+h) = \beta_0 + \beta_1 * x_1(t-n_0, t) + \beta_2 * x_1(t-n_1, t)\beta_1 * x_1(t-n_0, t) + \beta_3 * x_1(t-n_1, t) + \ldots$$

R(t,t+h): h-day look ahead return
x_i(t_n_j,t): n_j day momentum of i th feature

Model 0

Pick 5-day look ahead return and spot transaction value and volume data

In [110]:
```
data = data.dropna()
data_Y = data['Ret_5d']
#data_X = data.drop(columns= ['Ret_1m','Ret_3m','Ret_5d','Close Price'])
data_X = data[['TRANSACTION_COUNT','TRANSACTED_VALUE']]
```

This returned very low adjusted R^2 and indicated low predicatbility and multicolinearity. The high condition number refers to high correlation between TRANSACTION_COUNT and TRANSACTED_VALUE, only TRANSACTED_VALUE is predictive in this case

In [111]:
```
data_X = sm.add_constant(data_X)
model = sm.OLS(data_Y, data_X).fit()
predictions = model.predict(data_X) # make the predictions by the model
# Print out the statistics
model.summary()
```

Out[111]:

OLS Regression Results

Dep. Variable:	Ret_5d	**R-squared:**	0.052
Model:	OLS	**Adj. R-squared:**	0.048
Method:	Least Squares	**F-statistic:**	13.76
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	1.52e-06
Time:	21:35:46	**Log-Likelihood:**	1732.8
No. Observations:	508	**AIC:**	-3460.
Df Residuals:	505	**BIC:**	-3447.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.4703	0.449	1.048	0.295	-0.412	1.352
TRANSACTION_COUNT	0.3598	0.645	0.558	0.577	-0.908	1.627
TRANSACTED_VALUE	-0.2947	0.060	-4.892	0.000	-0.413	-0.176

Omnibus:	24.613	**Durbin-Watson:**	0.420
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	29.561
Skew:	-0.466	**Prob(JB):**	3.81e-07
Kurtosis:	3.726	**Cond. No.**	2.71e+03

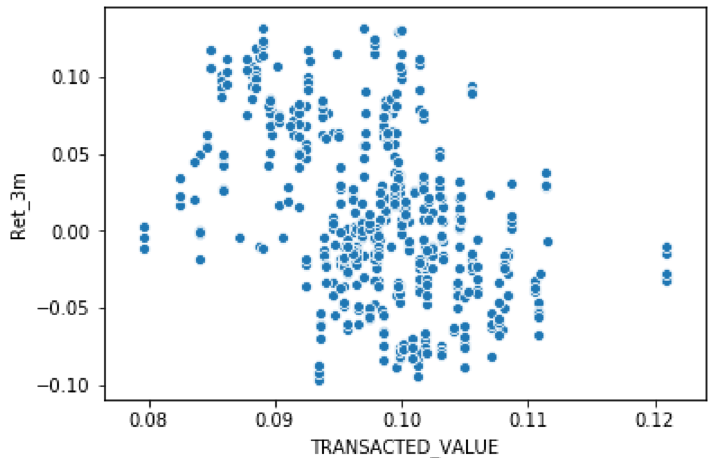
Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.71e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

Model 1

We chose a longer term look ahead return, get rid of the related volume data. We observed improved adjusted R^2, it might be indicative that it takes time for sales data to be absorbed by market and be embedded into pricing of the equity. However, it is counter intuitive to observe that sales value contribute negatively to market price

In [209]:
```
sns.scatterplot(data['TRANSACTED_VALUE'],data['Ret_3m'])
```

Out[209]: <matplotlib.axes._subplots.AxesSubplot at 0x1c30ef3400>



In [210]:
```
data = data.dropna()
data_Y = data['Ret_3m']
#data_X = data.drop(columns= ['Ret_1m','Ret_3m','Ret_5d','Close Price'])
data_X = data[['TRANSACTED_VALUE']]
data_X = sm.add_constant(data_X)
model = sm.OLS(data_Y, data_X).fit()
predictions = model.predict(data_X) # make the predictions by the model
# Print out the statistics
model.summary()
```

Out[210]:

OLS Regression Results

Dep. Variable:	Ret_3m	**R-squared:**	0.167
Model:	OLS	**Adj. R-squared:**	0.166
Method:	Least Squares	**F-statistic:**	101.8
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	6.27e-22
Time:	21:55:30	**Log-Likelihood:**	794.47
No. Observations:	508	**AIC:**	-1585.
Df Residuals:	506	**BIC:**	-1576.
Df Model:	1		
Covariance Type:	nonrobust		

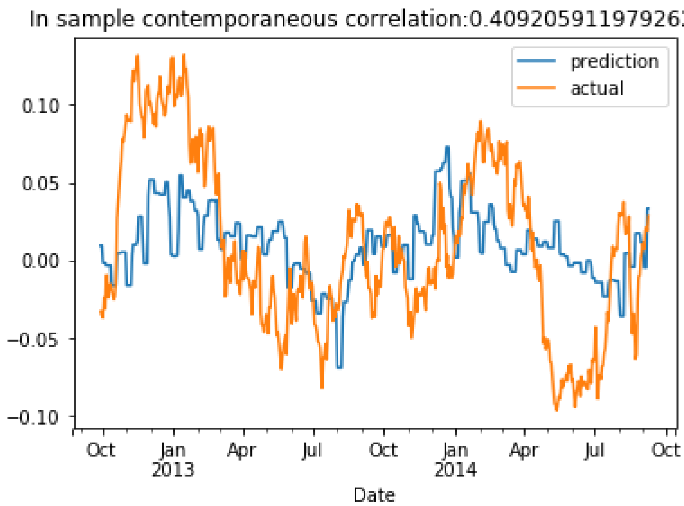
	coef	std err	t	P>|t|	[0.025	0.975]
const	0.3452	0.033	10.346	0.000	0.280	0.411
TRANSACTED_VALUE	-3.4225	0.339	-10.088	0.000	-4.089	-2.756

Omnibus:	12.084	**Durbin-Watson:**	0.071
Prob(Omnibus):	0.002	**Jarque-Bera (JB):**	8.217
Skew:	0.179	**Prob(JB):**	0.0164
Kurtosis:	2.490	**Cond. No.**	152.

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

In [211]:
```
predictions.plot(label='prediction')
data_Y.plot(label='actual')
plt.title('In sample contemporaneous correlation:' + str(predictions.corr(data_Y
)))
plt.legend()
```

Out[211]: <matplotlib.legend.Legend at 0x1c30fdee10>



Model 3

We now consider all variables, ignoring multicolinearity effects for now, observing transaction momentum features become signficant and all spot transaction data became insignificant. Namely,3-month sales revision, 3-month average sale revision, transaction volume 1/3 month moving average are the signficiant factors, interestingly they contribute negatively to 3-month lookahead stock prices, indicating a reversal effect.

In [214]:
```
data = data.dropna()

data_Y = data['Ret_3m']
data_X = data.drop(columns= ['Ret_1m','Ret_3m','Ret_5d','Close Price'])
data_X = sm.add_constant(data_X)
model = sm.OLS(data_Y, data_X).fit()
predictions = model.predict(data_X) # make the predictions by the model
# Print out the statistics
model.summary()
```

Out[214]:

OLS Regression Results

Dep. Variable:	Ret_3m	**R-squared:**	0.249
Model:	OLS	**Adj. R-squared:**	0.226
Method:	Least Squares	**F-statistic:**	10.86
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	7.31e-23
Time:	21:55:48	**Log-Likelihood:**	820.56
No. Observations:	508	**AIC:**	-1609.
Df Residuals:	492	**BIC:**	-1541.
Df Model:	15		
Covariance Type:	nonrobust		

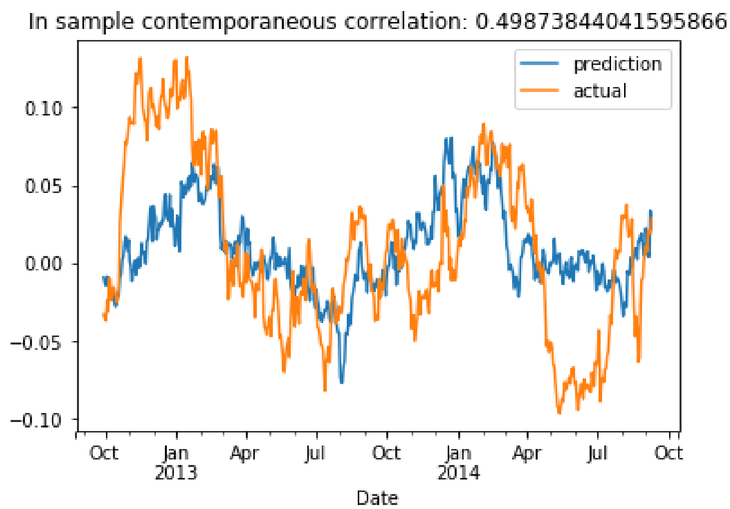
	coef	std err	t	P>|t|	[0.025	0.975]
const	-29.8612	37.319	-0.800	0.424	-103.186	43.463
TRANSACTION_COUNT	48.3554	52.341	0.924	0.356	-54.484	151.195
TRANSACTED_VALUE	-4.1845	4.106	-1.019	0.309	-12.252	3.883
TRANSACTED_AVG_VALUE	0.3614	0.445	0.813	0.417	-0.512	1.235
TRANSACTED_VALUE_5d	0.0329	0.041	0.794	0.428	-0.049	0.114
TRANSACTION_COUNT_5d	-0.0479	0.044	-1.094	0.274	-0.134	0.038
TRANSACTED_AVG_VALUE_5d	0.0329	0.041	0.794	0.428	-0.049	0.114
TRANSACTED_VALUE_1m	0.0063	0.026	0.248	0.804	-0.044	0.057
TRANSACTION_COUNT_1m	0.0114	0.034	0.341	0.733	-0.054	0.077
TRANSACTED_AVG_VALUE_1m	0.0063	0.026	0.248	0.804	-0.044	0.057
TRANSACTED_VALUE_3m	-0.0748	0.020	-3.664	0.000	-0.115	-0.035
TRANSACTION_COUNT_3m	-0.0810	0.063	-1.285	0.199	-0.205	0.043
TRANSACTED_AVG_VALUE_3m	-0.0748	0.020	-3.664	0.000	-0.115	-0.035
TRANSACTED_VALUE_MV5d	-0.0117	0.145	-0.081	0.935	-0.296	0.273
TRANSACTION_COUNT_MV5d	-5.7700	2.853	-2.022	0.044	-11.377	-0.164
TRANSACTED_AVG_VALUE_MV5d	-0.0117	0.145	-0.081	0.935	-0.296	0.273
TRANSACTED_VALUE_MV1m	0.0439	0.032	1.355	0.176	-0.020	0.108
TRANSACTION_COUNT_MV1m	1.3454	1.061	1.268	0.206	-0.740	3.431
TRANSACTED_AVG_VALUE_MV1m	0.0439	0.032	1.355	0.176	-0.020	0.108
TRANSACTED_VALUE_MV3m	-0.0148	0.011	-1.291	0.197	-0.037	0.008
TRANSACTION_COUNT_MV3m	-1.1569	0.416	-2.779	0.006	-1.975	-0.339
TRANSACTED_AVG_VALUE_MV3m	-0.0148	0.011	-1.291	0.197	-0.037	0.008

Omnibus:	9.488	**Durbin-Watson:**	0.086
Prob(Omnibus):	0.009	**Jarque-Bera (JB):**	7.955
Skew:	0.228	**Prob(JB):**	0.0187
Kurtosis:	2.591	**Cond. No.**	3.40e+18

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.17e-33. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [216]:
```
predictions.plot(label='prediction')
data_Y.plot(label='actual')
plt.title('In sample contemporaneous correlation: ' + str(predictions.corr(data_Y
)))
plt.legend()
```

Out[216]: <matplotlib.legend.Legend at 0x1c31165860>



Model 4

recursive feature selection

In [217]:
```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression

data_Y = data['Ret_3m'].shift(-60).dropna()
data_X = data.drop(columns= ['Ret_1m','Ret_3m','Ret_5d','Close Price']).iloc[:-60]
data_X = sm.add_constant(data_X)

model = LinearRegression()
rfe = RFE(model, 5)
fit = rfe.fit(data_X, data_Y)
select_feat = data_X.columns.values[fit.support_]
print("Selected Features: %s" % select_feat)

data = data.dropna()
data_Y = data['Ret_3m']
data_X = data[select_feat]
data_X = sm.add_constant(data_X)
model = sm.OLS(data_Y, data_X).fit()
predictions = model.predict(data_X) # make the predictions by the model
# Print out the statistics
model.summary()
```

```
Selected Features: ['TRANSACTED_VALUE' 'TRANSACTED_VALUE_3m' 'TRANSACTION_COUNT_M
V5d'
 'TRANSACTION_COUNT_MV1m' 'TRANSACTION_COUNT_MV3m']
```

Out[217]:

OLS Regression Results

Dep. Variable:	Ret_3m	**R-squared:**	0.238
Model:	OLS	**Adj. R-squared:**	0.231
Method:	Least Squares	**F-statistic:**	31.40
Date:	Sun, 28 Jul 2019	**Prob (F-statistic):**	7.81e-28
Time:	21:56:08	**Log-Likelihood:**	817.03
No. Observations:	508	**AIC:**	-1622.
Df Residuals:	502	**BIC:**	-1597.
Df Model:	5		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	5.0169	1.273	3.942	0.000	2.517	7.517
TRANSACTED_VALUE	-0.3022	0.614	-0.492	0.623	-1.509	0.905
TRANSACTED_VALUE_3m	-0.1514	0.035	-4.382	0.000	-0.219	-0.084
TRANSACTION_COUNT_MV5d	-6.7898	2.276	-2.983	0.003	-11.262	-2.317
TRANSACTION_COUNT_MV1m	1.7360	0.783	2.217	0.027	0.198	3.274
TRANSACTION_COUNT_MV3m	-1.4056	0.259	-5.420	0.000	-1.915	-0.896

Omnibus:	9.115	**Durbin-Watson:**	0.067
Prob(Omnibus):	0.010	**Jarque-Bera (JB):**	8.695
Skew:	0.278	**Prob(JB):**	0.0129
Kurtosis:	2.680	**Cond. No.**	2.26e+03

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.26e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

split data 50-50 into train and test set, with no validation set and hyperparameter tunning, observe the out of sample performance

In [218]:

```
split = int(len(data_X)*0.5)
train_X = data_X.iloc[:split]
test_X = data_X.iloc[split:]
train_Y = data_Y.iloc[:split]
test_Y = data_Y.iloc[split:]
```

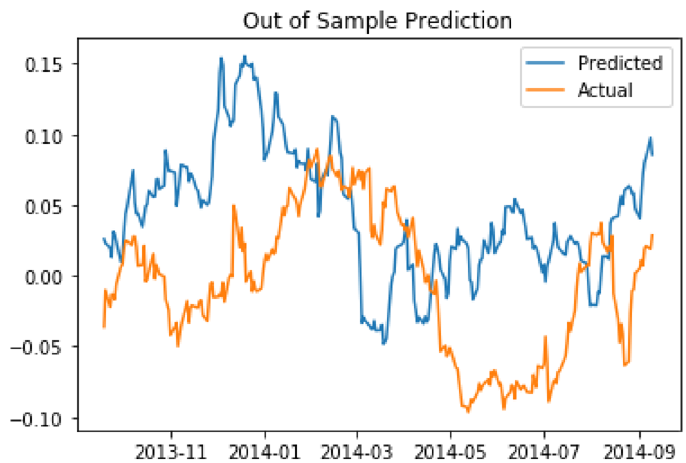
In [219]:

```
model = sm.OLS(train_Y, train_X).fit()
predictions = model.predict(test_X)
```

In [220]:
```
print(predictions.corr(test_Y))
plt.plot(predictions,label='Predicted')
plt.plot(test_Y,label = 'Actual')
plt.legend()
plt.title('Out of Sample Prediction')
```

```
0.1684717811588413
```

Out[220]:
```
Text(0.5, 1.0, 'Out of Sample Prediction')
```



Conlusion: Using simple linear model, it is questionable that there is predicability of transaction data over stock prices that we can take action on. Model 0~5 lacks robustness due to short time span and we can not rule out the possibility that any predictability is a result of spurious relationship (confunding of seasonality).

Model Building - NonLinear Model

First fit an arbitrary gradient boosting tree to the data and observe in-sample performance

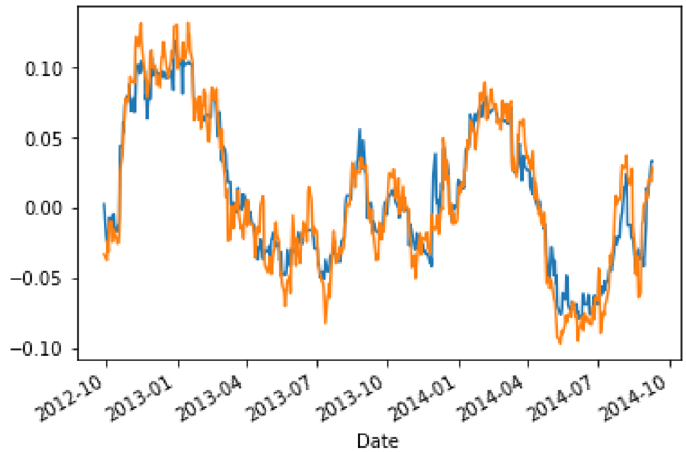
In [223]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
xgb_reg = xgb.XGBRegressor()
xgb_reg.fit(data_X.values, data_Y.values)
xgb_train_pred = xgb_reg.predict(data_X.values)
#lgb_pred = np.exp(lgb_train_pred)-1
#print(np.sqrt(mean_squared_error(train_Y.values, xgb_train_pred)))
```

```
[21:57:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
```

This is a result of overfitting

In [224]:
```
plt.plot(data_Y.index,xgb_train_pred)
data_Y.plot()
```

Out[224]: <matplotlib.axes._subplots.AxesSubplot at 0x1c3134ef98>



If more data is available, referring to a longer lookback history as well as other data features, it would be intersting to evalute predictability based on cross validation. Although the current data volume is not enough to justify the validality of such analysis. For exploration purposes, use a set of individually tuned parameters and applying a stacking model to the data

In [225]:
```
##############################method 2#########################################3

from sklearn.metrics import mean_squared_error
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.linear_model import RidgeCV, ElasticNet, LassoCV, LassoLarsCV
from sklearn.kernel_ridge import KernelRidge
from vecstack import stacking
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.feature_selection import SelectFromModel
from xgboost import XGBRFRegressor

models = [
    make_pipeline(RobustScaler(),xgb.XGBRegressor()),
    make_pipeline(RobustScaler(), RidgeCV(alphas =[0.0005,0.001,0.0015],cv=3)),
    make_pipeline(RobustScaler(), LassoCV(alphas =[0.0005,0.001,0.0015], random_st
ate=1,cv=3)),
    make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9, random_sta
te=3)),
    KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5),
    ]

# Compute stacking features
S_train, S_test = stacking(models, train_X.values, train_Y.values, test_X.values,
    regression = True, metric = mean_squared_error, n_folds = 20,
    shuffle = True, random_state = 0, verbose = 2)


model = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0001, l1_ratio=.9, random
_state=3))


# Fit 2-nd level model
model = model.fit(S_train, train_Y.values)

# Predict
y_pred = model.predict(S_train)

# Final prediction score
print('Final prediction score: [%.8f]' % mean_squared_error(train_Y.values, y_pred
))
y_pred = model.predict(S_test)
```

```
task:         [regression]
metric:       [mean_squared_error]
mode:         [oof_pred_bag]
n_models:     [5]

model  0:     [Pipeline]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  0:  [0.00017147]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  1:  [0.00034821]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  2:  [0.00020887]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  3:  [0.00016366]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  4:  [0.00012306]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  5:  [0.00047590]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  6:  [0.00040894]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  7:  [0.00041673]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  8:  [0.00031523]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold  9:  [0.00032134]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 10:  [0.00037872]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 11:  [0.00039564]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 12:  [0.00021877]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 13:  [0.00023099]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 14:  [0.00063558]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 15:  [0.00020129]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 16:  [0.00071858]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 17:  [0.00044136]
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 18:  [0.00026173]
```

```
[21:57:22] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprec
ated in favor of reg:squarederror.
    fold 19:  [0.00027533]
    ----
    MEAN:     [0.00033557] + [0.00015086]
    FULL:     [0.00033352]

model  1:     [Pipeline]
    fold  0:  [0.00103898]
```

```
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
```

```
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

```
    fold  1:  [0.00135275]
    fold  2:  [0.00138632]
    fold  3:  [0.00095472]
    fold  4:  [0.00054556]
    fold  5:  [0.00194000]
    fold  6:  [0.00121455]
    fold  7:  [0.00165588]
    fold  8:  [0.00117936]
    fold  9:  [0.00166096]
    fold 10:  [0.00138585]
    fold 11:  [0.00107081]
    fold 12:  [0.00096102]
    fold 13:  [0.00086598]
    fold 14:  [0.00168253]
```

```
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
/Users/charles-18/anaconda3/lib/python3.7/site-packages/sklearn/model_selection/_
search.py:841: DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24. This will change
numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

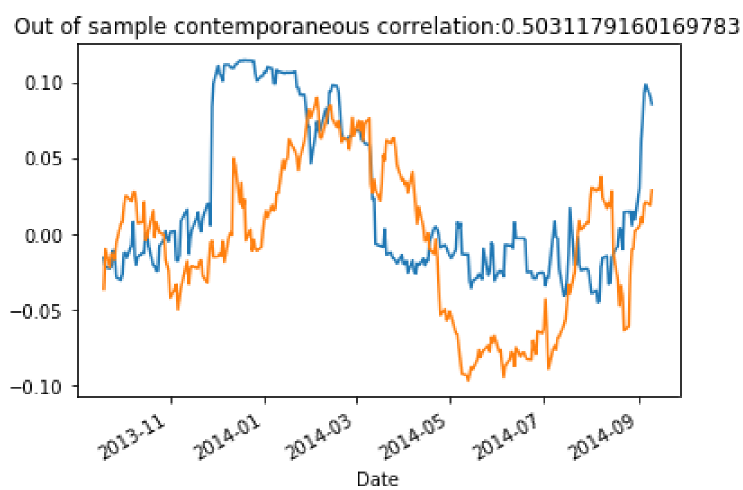
```
    fold 15:  [0.00328011]
    fold 16:  [0.00195563]
    fold 17:  [0.00161717]
    fold 18:  [0.00076929]
    fold 19:  [0.00113841]
    ----
    MEAN:     [0.00138279] + [0.00057387]
    FULL:     [0.00137434]

model  2:     [Pipeline]
    fold  0:  [0.00118215]
    fold  1:  [0.00136037]
    fold  2:  [0.00135056]
    fold  3:  [0.00084748]
    fold  4:  [0.00063076]
    fold  5:  [0.00207556]
    fold  6:  [0.00136003]
    fold  7:  [0.00183442]
    fold  8:  [0.00091313]
    fold  9:  [0.00178628]
    fold 10:  [0.00157356]
    fold 11:  [0.00135274]
    fold 12:  [0.00101596]
    fold 13:  [0.00106644]
    fold 14:  [0.00158478]
    fold 15:  [0.00347262]
    fold 16:  [0.00239560]
    fold 17:  [0.00173504]
    fold 18:  [0.00072716]
    fold 19:  [0.00114301]
    ----
    MEAN:     [0.00147038] + [0.00063740]
    FULL:     [0.00146158]

model  3:     [Pipeline]
    fold  0:  [0.00108670]
    fold  1:  [0.00133761]
    fold  2:  [0.00135325]
    fold  3:  [0.00089532]
    fold  4:  [0.00056857]
    fold  5:  [0.00199347]
    fold  6:  [0.00126632]
    fold  7:  [0.00171716]
    fold  8:  [0.00101977]
    fold  9:  [0.00170290]
    fold 10:  [0.00142506]
    fold 11:  [0.00116851]
    fold 12:  [0.00096232]
    fold 13:  [0.00093612]
    fold 14:  [0.00159929]
    fold 15:  [0.00335776]
    fold 16:  [0.00215852]
    fold 17:  [0.00165312]
    fold 18:  [0.00073846]
    fold 19:  [0.00111235]
    ----
    MEAN:     [0.00140263] + [0.00060210]
    FULL:     [0.00139395]

model  4:     [KernelRidge]
    fold  0:  [0.00160948]
    fold  1:  [0.00178283]
    fold  2:  [0.00185254]
```

```
    fold  3:  [0.00206942]
    fold  4:  [0.00130600]
    fold  5:  [0.00189474]
    fold  6:  [0.00345517]
    fold  7:  [0.00153742]
    fold  8:  [0.00203865]
    fold  9:  [0.00194081]
    fold 10:  [0.00250277]
    fold 11:  [0.00220335]
    fold 12:  [0.00194299]
    fold 13:  [0.00234699]
    fold 14:  [0.00260725]
    fold 15:  [0.00298737]
    fold 16:  [0.00245369]
    fold 17:  [0.00238008]
    fold 18:  [0.00167007]
    fold 19:  [0.00206515]
    ----
    MEAN:     [0.00213234] + [0.00049649]
    FULL:     [0.00212695]

Final prediction score: [0.00032653]
```

We are able to increase out of sample correlation from 0.16 to 0.48. We can visually obersve some similiarities in market microstructures especially regarding the stock price increase 2013 Q1 and 2014Q4 indicating the data could potentially be helpful as a risk factor.

In [226]:
```
plt.plot(test_Y.index,y_pred)
test_Y.plot()
plt.title("Out of sample contemporaneous correlation:" + str(np.corrcoef(y_pred,te
st_Y)[0][1]))
```

Out[226]:
```
Text(0.5, 1.0, 'Out of sample contemporaneous correlation:0.5031179160169783')
```



Commentary

To summarize, transactional data exhibits strong seasonablity. We initially constructed a linear model for which momentum of sales data would be predictive of mcdonald's market returns. Specficially, we assumed 3-month aggreagete sales(to model a rolling quater) should be somewhat predictive over market prices over a short period of time, generally referring to market reaction to quaterly release. However, we did not find a strong evidence potentially due to:

1. Spurious relationship between credit sales and equity prices (for example, seasonality as confunding variable)
2. Equity price seems to be driven by many factors in addition to coporate sales revisions.
3. Cash is widely used at Mcdonalds chain stores so credit card transaction does not provide the full picutre
4. Quaterly releases are also discrete events instead of being continous.
5. The majority of investor are likely not reactive enough to real-time sales numbers to make investment desicions.

As a result, the conclusion is that Mcdonald's sales transaction data is somewhat helpful for us to understand stock prices (potentially in a format of a risk factor) does not grant us the premise to build a robust direational based trading strategy. We need data with a longer lookback (higher frequency) and a cross section of stock prices to increase the confidence in its usefulness

Enhancement / Additional Information

Mcdonalds quanterly sales consensus number:

1. we can use it to justify the quality of the data, we expect a high correlation between transaction value and sales figure over quaterly timeframe.
2. A large discrepancy between consensus number and forcasted sales is likely an indicator of earnings surprise, which usually drives equity prices

Stock Prices of Other fast food restaurants which are similar to Mcdonalds:

1. We can test predictability of the data by ranking the stocks based on sales and evaluate returns of a long-short portfolio