NAME: Oluwaseun A. Dada Assignnment #2: Stock forecasting

-Set up the most basic analytics pipeline for the forecast project. –Use close price as input –Create a label 1-day ahead –Train your linear model –Create a prediction –Visualize prediction and price in same chart

Step 1 - get data

In [1]:

```
import datetime as dt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# https://stackoverflow.com/a/50970152
pd.core.common.is_list_like = pd.api.types.is_list_like
# you have to install pandas_datareader module i Anaconda Navigator:
# http://docs.anaconda.com/anaconda/navigator/tutorials/manage-packages/
from pandas_datareader.data import DataReader
```

In [2]:

```
# Define timeframe of stocks we retreive
end = dt.datetime.now()
start = end - dt.timedelta(days=5*365)
```

In [3]:

```
# Use DataReader to get Apples stock data from IEX https://iextrading.com/developer/
df = DataReader('AAPL','iex', start, end)
#df = DataReader('MU','iex', start, end)
```

5y

In [4]:

```
df.dtypes
```

Out[4]:

```
open      float64
high      float64
low       float64
close     float64
volume      int64
dtype: object
```

In [5]:

```
df.columns
```

Out[5]:

```
Index(['open', 'high', 'low', 'close', 'volume'], dtype='object')
```

In [6]:

```
df.head()
```

Out[6]:

date	open	high	low	close	volume
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122

In [7]:

```
df.tail()
```

Out[7]:

date	open	high	low	close	volume
2018-10-09	223.64	227.27	222.2462	226.87	26891029
2018-10-10	225.46	226.35	216.0500	216.36	41990554
2018-10-11	214.52	219.50	212.3200	214.45	53124392
2018-10-12	220.42	222.88	216.8400	222.11	40337851
2018-10-15	221.16	221.83	217.2700	217.36	30791007

In [8]:

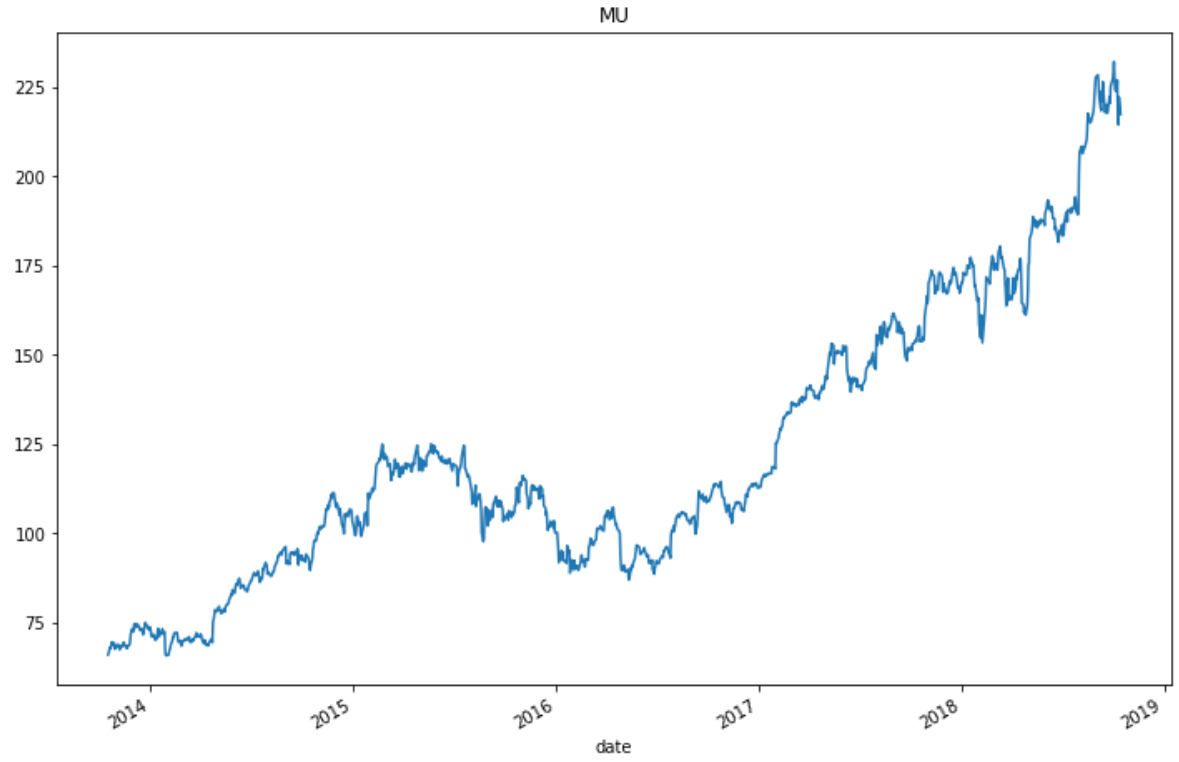
```
df.index = pd.to_datetime(df.index, format='%Y-%m-%d')
```

In [9]:

```
df.close.plot(figsize=(12,8), title='MU')
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef663af3c8>
```



In [10]:

```
df.head()
```

Out[10]:

date	open	high	low	close	volume
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122

Step 2 - implement 2 features and visualize the price and the features in the same graph

In [11]:

```
# https://www.investopedia.com/articles/technical/081501.asp
df['momentum'] = df.close - df.close.shift(1)
df.head(10)
```

Out[11]:

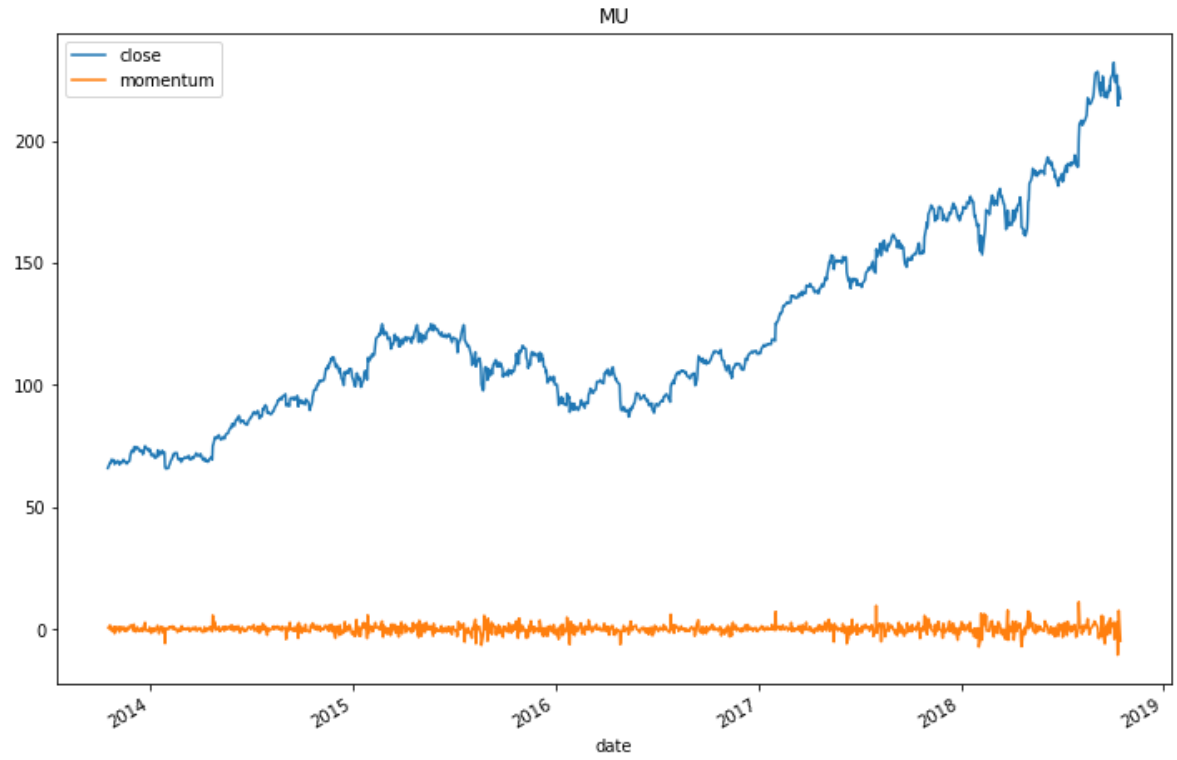
date	open	high	low	close	volume	momentum
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661
2013-10-24	68.6722	69.6493	68.3386	69.5760	96191095	0.9091
2013-10-25	69.4982	69.7487	68.6866	68.7975	84448133	-0.7785
2013-10-28	69.2006	69.4570	68.4380	69.3100	137610123	0.5125
2013-10-29	70.1463	70.5361	67.3040	67.5836	158952115	-1.7264
2013-10-30	67.9671	69.0018	67.6284	68.6586	88540697	1.0750

In [12]:

```
df[['close', 'momentum']].plot(figsize=(12,8), title='MU')
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef66727828>
```



Step 3 - do a regression based on the features, forecast

1-day ahead

In [13]:

```
df.head()
```

Out[13]:

date	open	high	low	close	volume	momentum
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661

Calculate change between days using log returns between two consecutive days

In [14]:

```
df["log_ret"] = np.log(df["close"]/df["close"].shift(1))
```

In [15]:

```
df.head(10)
```

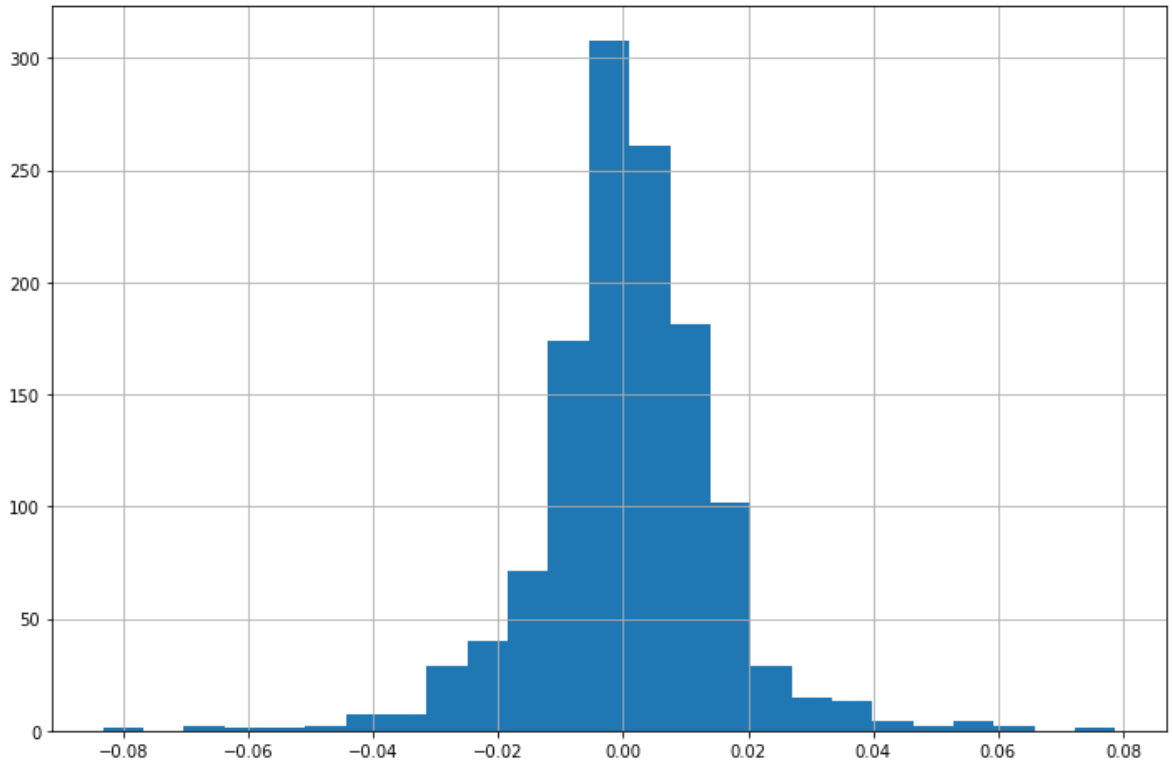
Out[15]:

date	open	high	low	close	volume	momentum	log_ret
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748
2013-10-24	68.6722	69.6493	68.3386	69.5760	96191095	0.9091	0.013152
2013-10-25	69.4982	69.7487	68.6866	68.7975	84448133	-0.7785	-0.011252
2013-10-28	69.2006	69.4570	68.4380	69.3100	137610123	0.5125	0.007422
2013-10-29	70.1463	70.5361	67.3040	67.5836	158952115	-1.7264	-0.025224
2013-10-30	67.9671	69.0018	67.6284	68.6586	88540697	1.0750	0.015781

In [16]:

```
print("Max value: ", df['log_ret'].max())
print("Min value: ", df['log_ret'].min())
df['log_ret'].hist(bins=25, figsize=(12,8))
plt.show()
```

```
Max value:  0.07879405422991285
Min value:  -0.08330285327371977
```



Step 4 - plot (as a line) the regression and expected output, make the plot zoomable

In [17]:

```
df.head()
```

Out[17]:

date	open	high	low	close	volume	momentum	log_ret
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748

In [18]:

```
# we are using a numpy WHERE function to filter the 'log_ret' column of the data frame.
# Condition (WHERE clause): if the corresponding 'log_ret' value is less than 25 percentile
# result will be stored in a new column for 'crash'
# notice the new column: 'crash'
df['crash'] = np.where(df['log_ret'] < df['log_ret'].quantile(0.25), 1,0)
df.head(10)
```

Out[18]:

date	open	high	low	close	volume	momentum	log_ret	crash
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN	0
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0
2013-10-24	68.6722	69.6493	68.3386	69.5760	96191095	0.9091	0.013152	0
2013-10-25	69.4982	69.7487	68.6866	68.7975	84448133	-0.7785	-0.011252	1
2013-10-28	69.2006	69.4570	68.4380	69.3100	137610123	0.5125	0.007422	0
2013-10-29	70.1463	70.5361	67.3040	67.5836	158952115	-1.7264	-0.025224	1
2013-10-30	67.9671	69.0018	67.6284	68.6586	88540697	1.0750	0.015781	0

In [19]:

```
# we are using a numpy WHERE function to filter the 'log_ret' column of the data frame.
# Condition (WHERE clause): if the corresponding 'log_ret' value is less than 50 percentile
# result will be stored in a new column for 'down'
# notice the new column: 'down'
a = df['log_ret'] < df['log_ret'].quantile(0.5)
b = df['log_ret'] > df['log_ret'].quantile(0.25)
#b *a
```

In [20]:

```
# outcome should only be true when both are true
df['down'] = np.where(a * b, 1,0)
#df
```

```
C:\tools\Anaconda3\lib\site-packages\pandas\core\computation\expressions.py:
180: UserWarning: evaluating in Python space because the '*' operator is not
supported by numexpr for the bool dtype, use '&' instead
  .format(op=op_str, alt_op=unsupported[op_str]))
```

In [21]:

```
# we are using a numpy WHERE function to filter the 'log_ret' column of the data frame.
# Condition (WHERE clause): if the corresponding 'log_ret' value is less than 75 percentile
# result will be stored in a new column for 'up'
# notice the new column: 'up'

a = df['log_ret'] < df['log_ret'].quantile(0.75)
b = df['log_ret'] > df['log_ret'].quantile(0.5)

df['up'] = np.where(a * b, 1,0)
#df
```

```
C:\tools\Anaconda3\lib\site-packages\pandas\core\computation\expressions.py:
180: UserWarning: evaluating in Python space because the '*' operator is not
supported by numexpr for the bool dtype, use '&' instead
  .format(op=op_str, alt_op=unsupported[op_str]))
```

In [22]:

```
# we are using a numpy WHERE function to filter the 'log_ret' column of the data frame.
# Condition (WHERE clause): if the corresponding 'log_ret' value is less than 100 percentil
# result will be stored in a new column for 'jump'
# notice the new column: 'jump'
a = df['log_ret'] < df['log_ret'].quantile(1)
b = df['log_ret'] > df['log_ret'].quantile(0.75)

df['jump'] = np.where(a * b, 1,0)
```

```
C:\tools\Anaconda3\lib\site-packages\pandas\core\computation\expressions.py:
180: UserWarning: evaluating in Python space because the '*' operator is not
supported by numexpr for the bool dtype, use '&' instead
  .format(op=op_str, alt_op=unsupported[op_str]))
```

In [23]:

```
df[['close','momentum','crash','up','down','jump']]
```

Out[23]:

date	close	momentum	crash	up	down	jump
2013-10-17	65.9907	NaN	0	0	0	0
2013-10-18	66.5649	0.5742	0	0	0	1
2013-10-21	68.1963	1.6314	0	0	0	1
2013-10-22	68.0008	-0.1955	0	0	1	0
2013-10-23	68.6669	0.6661	0	0	0	1
2013-10-24	69.5760	0.9091	0	0	0	1
2013-10-25	68.7975	-0.7785	1	0	0	0
2013-10-28	69.3100	0.5125	0	1	0	0
2013-10-29	67.5836	-1.7264	1	0	0	0
2013-10-30	68.6586	1.0750	0	0	0	1
2013-10-31	68.3716	-0.2870	0	0	1	0
2013-11-01	68.0221	-0.3495	0	0	1	0
2013-11-04	68.9011	0.8790	0	0	0	1
2013-11-05	68.7309	-0.1702	0	0	1	0
2013-11-06	68.5363	-0.1946	0	0	1	0
2013-11-07	67.4275	-1.1088	1	0	0	0
2013-11-08	68.4890	1.0615	0	0	0	1
2013-11-11	68.2901	-0.1989	0	0	1	0
2013-11-12	68.4166	0.1265	0	1	0	0
2013-11-13	68.4987	0.0821	0	1	0	0
2013-11-14	69.4889	0.9902	0	0	0	1
2013-11-15	69.0720	-0.4169	1	0	0	0
2013-11-18	68.2349	-0.8371	1	0	0	0
2013-11-19	68.3561	0.1212	0	1	0	0
2013-11-20	67.7575	-0.5986	1	0	0	0
2013-11-21	68.5648	0.8073	0	0	0	1
2013-11-22	68.3890	-0.1758	0	0	1	0
2013-11-25	68.9074	0.5184	0	1	0	0
2013-11-26	70.1783	1.2709	0	0	0	1
2013-11-27	71.8308	1.6525	0	0	0	1
...	...	...	...	...	...	...
2018-09-04	228.3600	0.7300	0	1	0	0
2018-09-05	226.8700	-1.4900	1	0	0	0
2018-09-06	223.1000	-3.7700	1	0	0	0

date	close	momentum	crash	up	down	jump
2018-09-07	221.3000	-1.8000	1	0	0	0
2018-09-10	218.3300	-2.9700	1	0	0	0
2018-09-11	223.8500	5.5200	0	0	0	1
2018-09-12	221.0700	-2.7800	1	0	0	0
2018-09-13	226.4100	5.3400	0	0	0	1
2018-09-14	223.8400	-2.5700	1	0	0	0
2018-09-17	217.8800	-5.9600	1	0	0	0
2018-09-18	218.2400	0.3600	0	1	0	0
2018-09-19	218.3700	0.1300	0	0	1	0
2018-09-20	220.0300	1.6600	0	1	0	0
2018-09-21	217.6600	-2.3700	1	0	0	0
2018-09-24	220.7900	3.1300	0	0	0	1
2018-09-25	222.1900	1.4000	0	1	0	0
2018-09-26	220.4200	-1.7700	1	0	0	0
2018-09-27	224.9500	4.5300	0	0	0	1
2018-09-28	225.7400	0.7900	0	1	0	0
2018-10-01	227.2600	1.5200	0	1	0	0
2018-10-02	229.2800	2.0200	0	0	0	1
2018-10-03	232.0700	2.7900	0	0	0	1
2018-10-04	227.9900	-4.0800	1	0	0	0
2018-10-05	224.2900	-3.7000	1	0	0	0
2018-10-08	223.7700	-0.5200	0	0	1	0
2018-10-09	226.8700	3.1000	0	0	0	1
2018-10-10	216.3600	-10.5100	1	0	0	0
2018-10-11	214.4500	-1.9100	1	0	0	0
2018-10-12	222.1100	7.6600	0	0	0	1
2018-10-15	217.3600	-4.7500	1	0	0	0

1258 rows × 6 columns

In [24]:

```
# calculate standardisation value for the data
std = (df['log_ret'] - df['log_ret'].mean())/(df['log_ret'].std())
std.head()
```

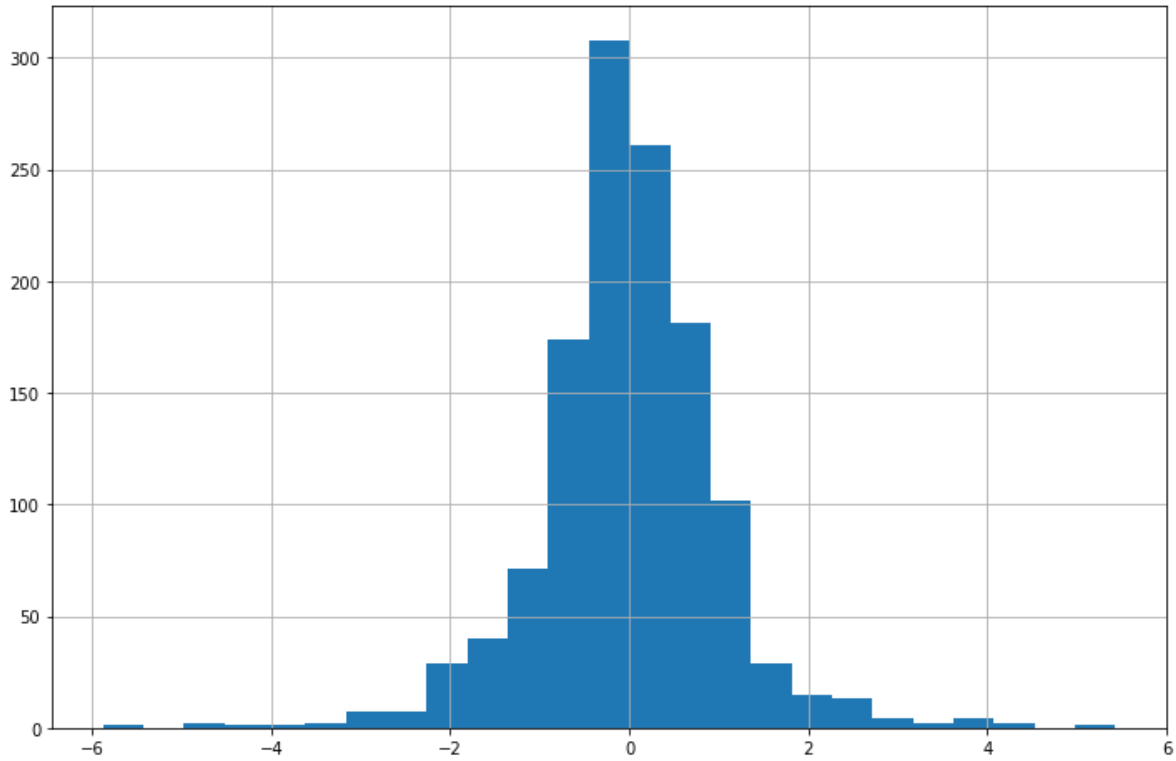
Out[24]:

```
date
2013-10-17         NaN
2013-10-18    0.538254
2013-10-21    1.623049
2013-10-22   -0.266443
2013-10-23    0.613894
Name: log_ret, dtype: float64
```

In [25]:

```
print("Max value:", std.max())
print("Max value:", std.min())
std.hist(bins=25,figsize=(12,8))
plt.show()
```

```
Max value: 5.430891757839674
Max value: -5.87776624057741
```



In [26]:

```
#Given that date is our indexes, we can resample as follows
# Here we make each row the mean values of the data that month
df.resample("M").mean().mean()
```

Out[26]:

```
open        1.225821e+02
high        1.236078e+02
low         1.215642e+02
close       1.226104e+02
volume      4.490681e+07
momentum    1.130997e-01
log_ret     9.404273e-04
crash       2.505810e-01
down        2.465762e-01
up          2.464028e-01
jump        2.518129e-01
dtype: float64
```

In [27]:

```
df.resample("M").mean().count()
```

Out[27]:

```
open        61
high        61
low         61
close       61
volume      61
momentum    61
log_ret     61
crash       61
down        61
up          61
jump        61
dtype: int64
```

In [28]:

```
df.head()
```

Out[28]:

date	open	high	low	close	volume	momentum	log_ret	crash	down	up
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN	0	0	0
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0	0	0
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0	0	0
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0	1	0
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0	0

In [29]:

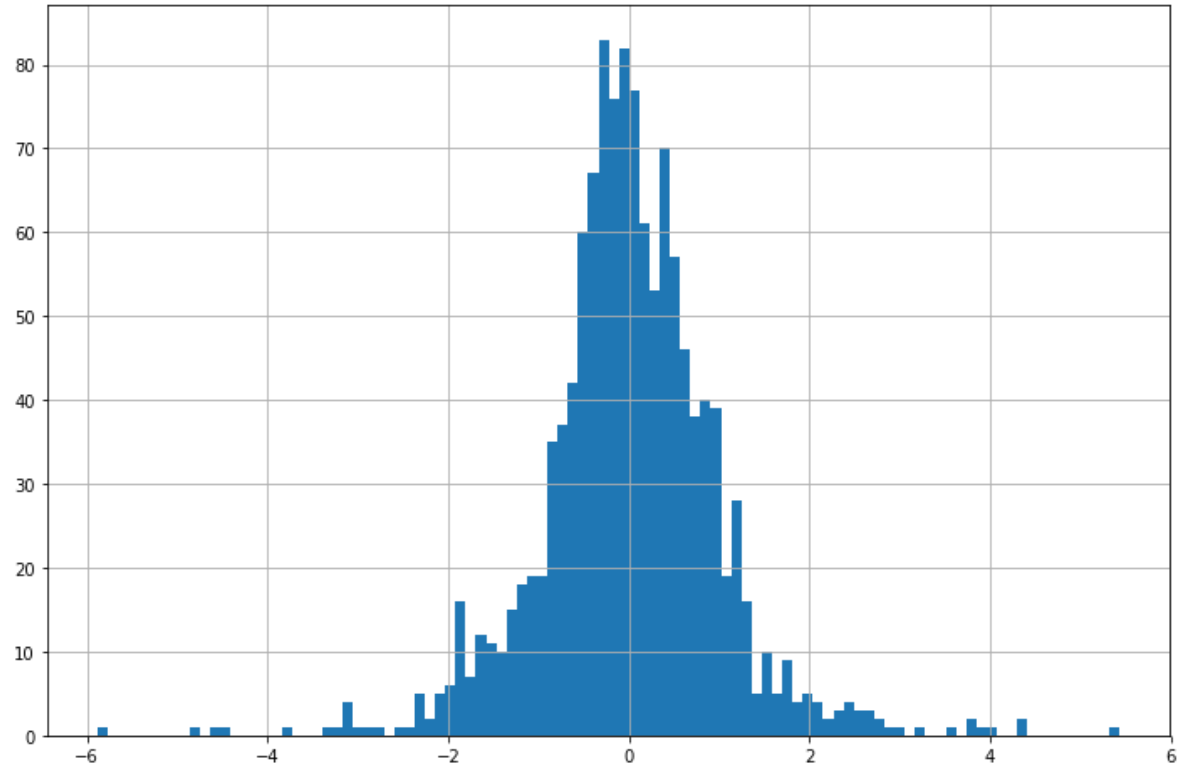
```
# stdNorm: standard normal distribution - In statistics, stdNorm is the process of putting
# it's a normal distribution with a mean of 0 and a standard deviation of 1.
# This process allows you to compare scores between different types of variables.

df["stdNorm"] = (df['log_ret'] - df['log_ret'].mean())/(df['log_ret'].max() - df['log_ret']
df["stdL_R"] = (df['log_ret'] - df['log_ret'].mean())/(df['log_ret'].std())
df.describe()
```

Out[29]:

	open	high	low	close	volume	momentum	lo
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1257.000000	1257.0(
mean	122.283337	123.300450	121.283381	122.319742	4.443072e+07	0.120421	0.0(
std	38.427186	38.726654	38.151584	38.440432	2.516639e+07	1.792652	0.01
min	65.149800	65.985200	64.935300	65.755300	1.147592e+07	-10.510000	-0.08
25%	94.160700	95.210250	93.432050	94.143325	2.663289e+07	-0.677600	-0.0(
50%	110.901200	112.050500	110.032900	111.194950	3.760330e+07	0.070000	0.0(
75%	150.912775	151.549300	149.917400	150.841475	5.527668e+07	0.978800	0.0(
max	230.780000	233.470000	229.780000	232.070000	2.668336e+08	11.170900	0.07

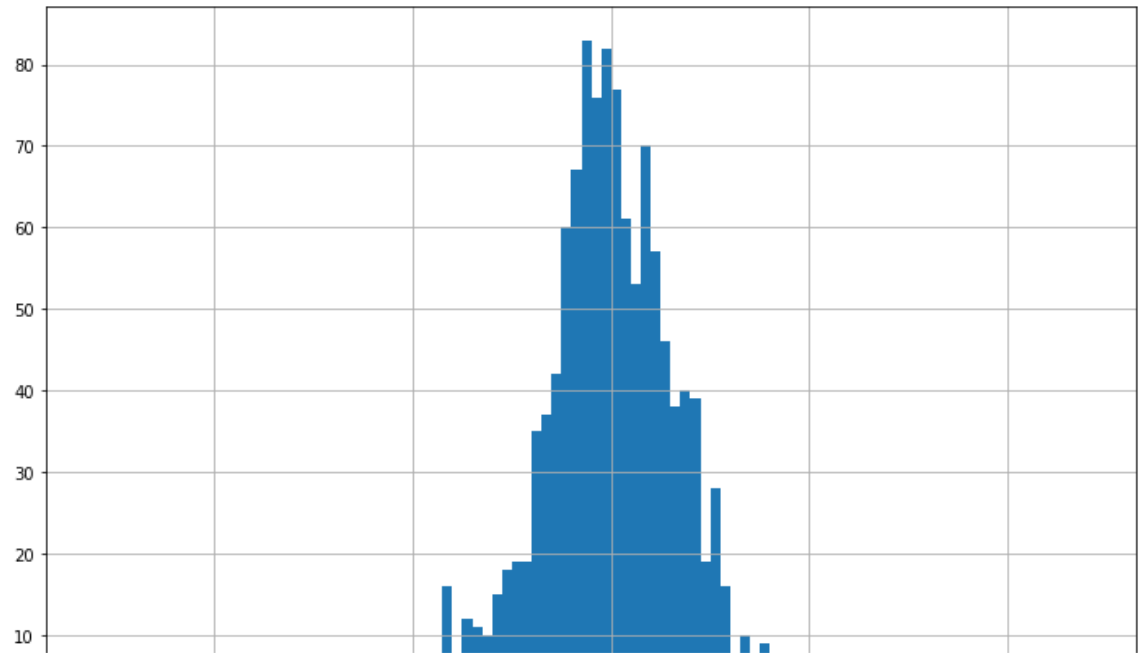
In [30]:

```
df["stdL_R"].hist(bins=100, figsize=(12,8))
plt.show()
```



In [31]:

```
df["stdNorm"].hist(bins=100, figsize=(12,8))
plt.show()
```



CALCULATE MOVING AVERAGES

In [32]:

```
df.head()
```

Out[32]:

date	open	high	low	close	volume	momentum	log_ret	crash	down	up
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN	0	0	0
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0	0	0
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0	0	0
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0	1	0
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0	0

In [33]:

```
df.reset_index(inplace=True)
```

In [34]:

```
df.head()
```

Out[34]:

	date	open	high	low	close	volume	momentum	log_ret	crash	down
0	2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN	0	0
1	2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0	0
2	2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0	0
3	2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0	1
4	2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0

In [35]:

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline
import seaborn as sns
sns.set(style='darkgrid', context='talk', palette='Dark2')

my_year_month_fmt = mdates.DateFormatter('%m/%y')
data = df
```

In [36]:

```
data['MA_14'] = data.close.rolling(14).mean()
data[['close','MA_14']].head(30)
```

Out[36]:

	close	MA_14
0	65.9907	NaN
1	66.5649	NaN
2	68.1963	NaN
3	68.0008	NaN
4	68.6669	NaN
5	69.5760	NaN
6	68.7975	NaN
7	69.3100	NaN
8	67.5836	NaN
9	68.6586	NaN
10	68.3716	NaN
11	68.0221	NaN
12	68.9011	NaN
13	68.7309	68.240786
14	68.5363	68.422614
15	67.4275	68.484229
16	68.4890	68.505136
17	68.2901	68.525800
18	68.4166	68.507921
19	68.4987	68.430971
20	69.4889	68.480357
21	69.0720	68.463357
22	68.2349	68.509879
23	68.3561	68.488271
24	67.7575	68.444407
25	68.5648	68.483171
26	68.3890	68.446593
27	68.9074	68.459200
28	70.1783	68.576486
29	71.8308	68.891007

In [37]:

```
data['MA_3'] = data.close.rolling(3).mean()
data['MA_7'] = data.close.rolling(7).mean()
data['MA_14'] = data.close.rolling(14).mean()
data['MA_21'] = data.close.rolling(21).mean()
```

In [38]:

```
data.head()
```

Out[38]:

	date	open	high	low	close	volume	momentum	log_ret	crash	down
0	2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN	0	0
1	2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0	0
2	2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0	0
3	2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0	1
4	2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0

In [39]:

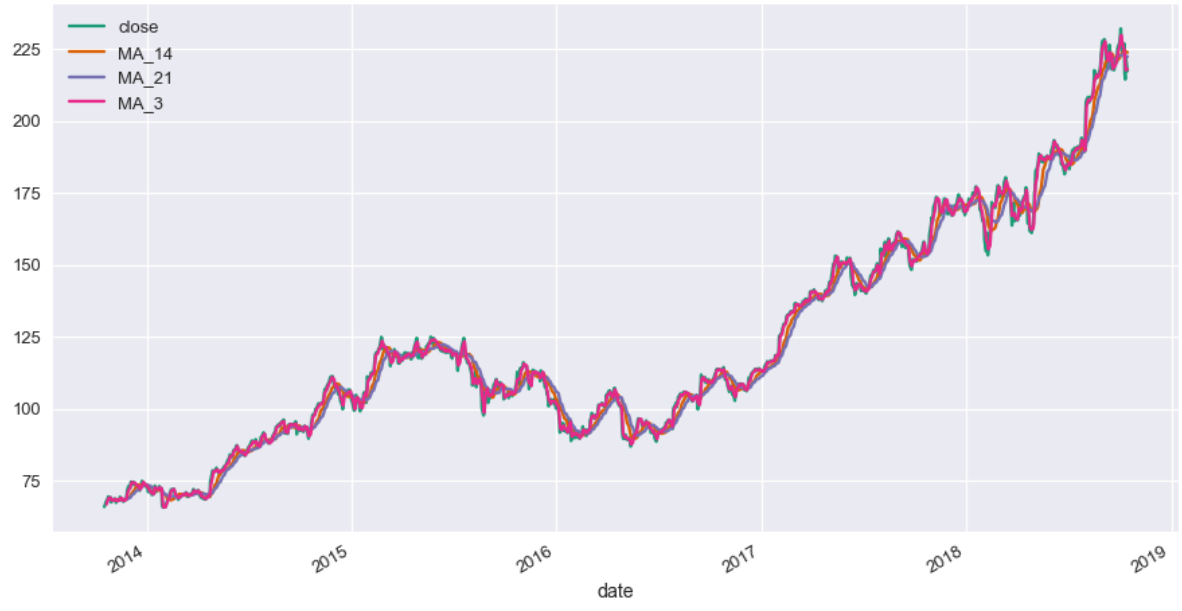
```
data.set_index('date', inplace=True)
```

In [40]:

```
data[['close','MA_14','MA_21','MA_3']].plot(figsize=(15,8))
```

Out[40]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef66841400>
```



In [41]:

```
data[['close','MA_14','MA_21','MA_3']].plot(figsize=(20,12))
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef67ffee48>
```

In [42]:

```
data[['close','MA_21']].plot(figsize=(15,8))
```

Out[42]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef67f9d940>
```

In [43]:

```
data.head()
```

Out[43]:

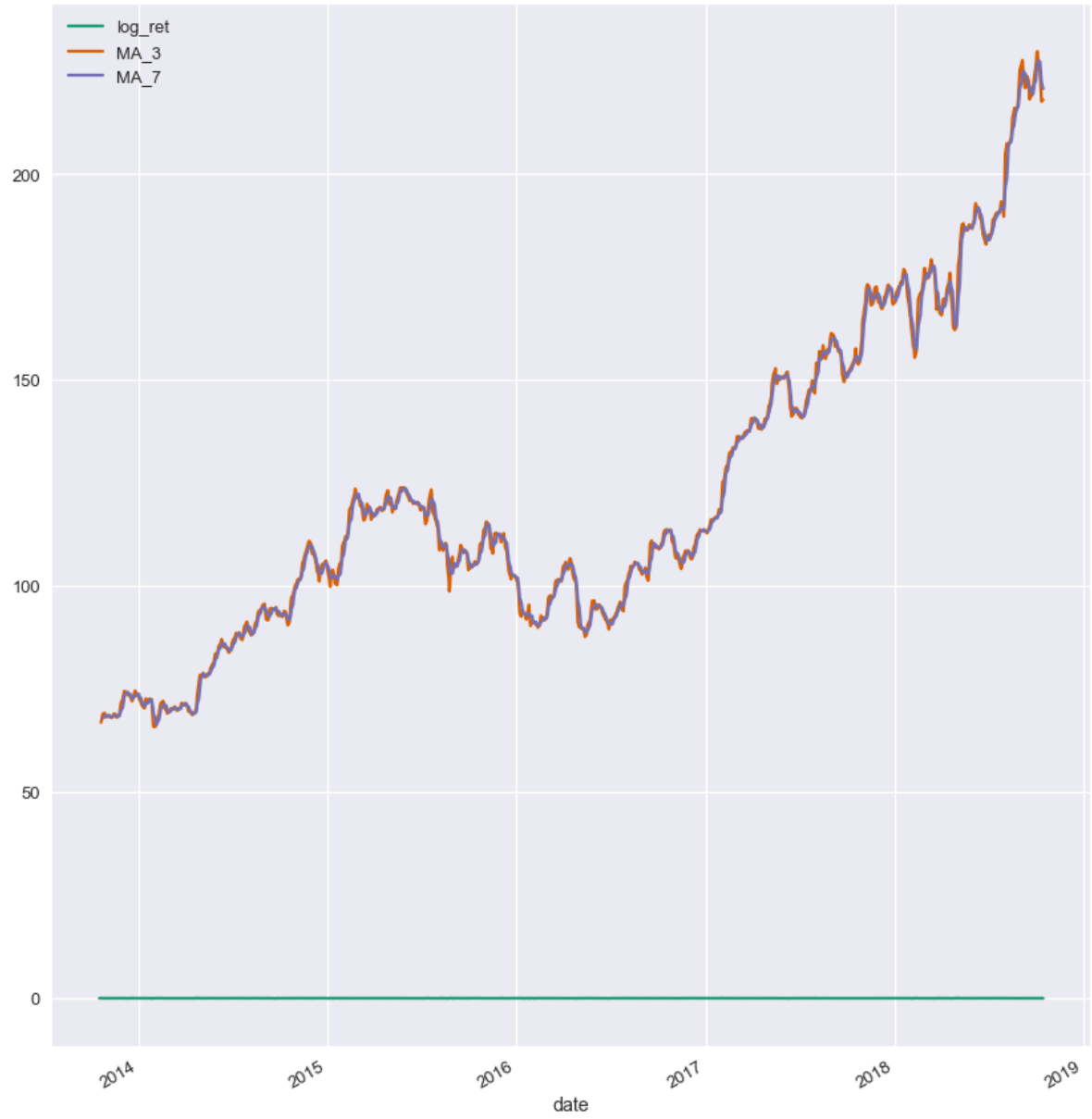
date	open	high	low	close	volume	momentum	log_ret	crash	down	up
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	NaN	NaN	0	0	0
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0	0	0
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0	0	0
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0	1	0
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0	0

In [44]:

```
# create plot for M(3,7) on the log return with price
# assumption: price = closing price ('close')

data[['log_ret','MA_3','MA_7']].plot(figsize=(14,16))
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef66bdf160>
```



In [45]:

```
# create plot for M(3,7) on the log return with price
# assumption: price = closing price ('close')

data[['log_ret','close','MA_3','MA_7']].plot(figsize=(27,10))
```

Out[45]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef67db8dd8>
```

CONSTRUCTING A FORECAST FOR THE TIME SERIES

In [46]:

```
data['label'] = data['close'].shift(-1)
data[['close','label']].head()
```

Out[46]:

date	close	label
2013-10-17	65.9907	66.5649
2013-10-18	66.5649	68.1963
2013-10-21	68.1963	68.0008
2013-10-22	68.0008	68.6669
2013-10-23	68.6669	69.5760

In [47]:

```
data[['close','label']].tail()
```

Out[47]:

date	close	label
2018-10-09	226.87	216.36
2018-10-10	216.36	214.45
2018-10-11	214.45	222.11
2018-10-12	222.11	217.36
2018-10-15	217.36	NaN

Turn data intro arrays and scale data

In [48]:

```
data.fillna(0,inplace=True)
```

- Set up the most basic analytics pipeline for the forecast project.

– Use close price as input

– Create a label 1-day ahead

– Train your linear model –Create a prediction

– Visualize prediction and price in same char

```
# X is the featureset, do not include labels
# X holds the input data

X = np.array(data[['MA_3','MA_7', 'MA_21']])
```

In [49]:

```
data['ASY_3'] = (data.close - data.close.shift(1)).rolling(3).sum()/3
data.ASY_3
```

Out[49]:

```
date
2013-10-17         NaN
2013-10-18         NaN
2013-10-21         NaN
2013-10-22    0.670033
2013-10-23    0.700667
2013-10-24    0.459900
2013-10-25    0.265567
2013-10-28    0.214367
2013-10-29   -0.664133
2013-10-30   -0.046300
2013-10-31   -0.312800
2013-11-01    0.146167
2013-11-04    0.080833
2013-11-05    0.119767
2013-11-06    0.171400
2013-11-07   -0.491200
2013-11-08   -0.080633
2013-11-11   -0.082067
2013-11-12    0.329700
2013-11-13    0.003233
2013-11-14    0.399600
2013-11-15    0.218467
2013-11-18   -0.087933
2013-11-19   -0.377600
2013-11-20   -0.438167
2013-11-21    0.109967
2013-11-22    0.010967
2013-11-25    0.383300
2013-11-26    0.537833
2013-11-27    1.147267
                 ...
2018-09-04    1.793333
2018-09-05    0.613333
2018-09-06   -1.510000
2018-09-07   -2.353333
2018-09-10   -2.846667
2018-09-11    0.250000
2018-09-12   -0.076667
2018-09-13    2.693333
2018-09-14   -0.003333
2018-09-17   -1.063333
2018-09-18   -2.723333
2018-09-19   -1.823333
2018-09-20    0.716667
2018-09-21   -0.193333
2018-09-24    0.806667
2018-09-25    0.720000
2018-09-26    0.920000
2018-09-27    1.386667
2018-09-28    1.183333
2018-10-01    2.280000
2018-10-02    1.443333
2018-10-03    2.110000
```

```
2018-10-04    0.243333
2018-10-05   -1.663333
2018-10-08   -2.766667
2018-10-09   -0.373333
2018-10-10   -2.643333
2018-10-11   -3.106667
2018-10-12   -1.586667
2018-10-15    0.333333
Name: ASY_3, Length: 1258, dtype: float64
```

In [50]:

```
data['MA_5'] = data.close.rolling(5).mean()
data.MA_5
```

Out[50]:

```
date
2013-10-17          NaN
2013-10-18          NaN
2013-10-21          NaN
2013-10-22          NaN
2013-10-23     67.48392
2013-10-24     68.20098
2013-10-25     68.64750
2013-10-28     68.87024
2013-10-29     68.78680
2013-10-30     68.78514
2013-10-31     68.54426
2013-11-01     68.38918
2013-11-04     68.30740
2013-11-05     68.53686
2013-11-06     68.51240
2013-11-07     68.32358
2013-11-08     68.41696
2013-11-11     68.29476
2013-11-12     68.23190
2013-11-13     68.22438
2013-11-14     68.63666
2013-11-15     68.75326
2013-11-18     68.74222
2013-11-19     68.73012
2013-11-20     68.58188
2013-11-21     68.39706
2013-11-22     68.26046
2013-11-25     68.39496
2013-11-26     68.75940
2013-11-27     69.57406
                  ...
2018-09-04    224.74000
2018-09-05    226.17400
2018-09-06    226.19800
2018-09-07    225.45200
2018-09-10    223.59200
2018-09-11    222.69000
2018-09-12    221.53000
2018-09-13    222.19200
2018-09-14    222.70000
2018-09-17    222.61000
2018-09-18    221.48800
2018-09-19    220.94800
2018-09-20    219.67200
2018-09-21    218.43600
2018-09-24    219.01800
2018-09-25    219.80800
2018-09-26    220.21800
2018-09-27    221.20200
2018-09-28    222.81800
2018-10-01    224.11200
2018-10-02    225.53000
2018-10-03    227.86000
```

```
2018-10-04    228.46800
2018-10-05    228.17800
2018-10-08    227.48000
2018-10-09    226.99800
2018-10-10    223.85600
2018-10-11    221.14800
2018-10-12    220.71200
2018-10-15    219.43000
Name: MA_5, Length: 1258, dtype: float64
```

In [51]:

```
data.head()
```

Out[51]:

date	open	high	low	close	volume	momentum	log_ret	crash	down	up
2013-10-17	65.3995	66.0273	65.3602	65.9907	63398335	0.0000	0.000000	0	0	0
2013-10-18	66.1856	66.6133	66.1490	66.5649	72635570	0.5742	0.008664	0	0	0
2013-10-21	66.9416	68.5806	66.9089	68.1963	99526945	1.6314	0.024213	0	0	0
2013-10-22	68.8560	69.1234	66.4524	68.0008	133515753	-0.1955	-0.002871	0	1	0
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0	0

In [52]:

```
data.dropna(inplace=True)
```

In [53]:

```
# X is the featureset, do not include labels
# X holds the input data

X = np.array(data[['close', 'ASY_3', 'MA_5']])
```

In [54]:

```
X
```

Out[54]:

```
array([[ 68.6669    ,   0.70066667,  67.48392   ],
       [ 69.576     ,   0.4599    ,  68.20098   ],
       [ 68.7975    ,   0.26556667,  68.6475    ],
       ...,
       [214.45      ,  -3.10666667, 221.148     ],
       [222.11      ,  -1.58666667, 220.712     ],
       [217.36      ,   0.33333333, 219.43      ]])
```

In [55]:

```
# y refer to the labels
# y holds the expected output
y = np.array(data['label'])
y
```

Out[55]:

```
array([ 69.576 ,  68.7975,  69.31  , ..., 222.11  , 217.36  ,   0.    ])
```

In [56]:

```
from sklearn import model_selection
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression, ElasticNetCV, ridge
from sklearn.neural_network import MLPRegressor
```

In [57]:

```
y[:6]
```

Out[57]:

```
array([69.576 , 68.7975, 69.31  , 67.5836, 68.6586, 68.3716])
```

In [58]:

```
X[:6]
```

Out[58]:

```
array([[ 6.86669000e+01,  7.00666667e-01,  6.74839200e+01],
       [ 6.95760000e+01,  4.59900000e-01,  6.82009800e+01],
       [ 6.87975000e+01,  2.65566667e-01,  6.86475000e+01],
       [ 6.93100000e+01,  2.14366667e-01,  6.88702400e+01],
       [ 6.75836000e+01, -6.64133333e-01,  6.87868000e+01],
       [ 6.86586000e+01, -4.63000000e-02,  6.87851400e+01]])
```

In [59]:

```
# scale the values down, fit standard scalar to y so x and y are using the same scale
#y = y.reshape(-1,1)

# ensure all na values are removed
y = y[np.logical_not(np.isnan(y))]
```

In [60]:

```
y.shape
```

Out[60]:

```
(1254,)
```

In [61]:

```
X.shape
```

Out[61]:

```
(1254, 3)
```

In [62]:

```
y = y.reshape(-1, 1)
y
```

Out[62]:

```
array([[ 69.576 ],
       [ 68.7975],
       [ 69.31  ],
       ...,
       [222.11  ],
       [217.36  ],
       [  0.    ]])
```

In [63]:

```
try:
    X = X.drop('date')
except: _
#X
```

In [64]:

```
# StandardScaler() will normalize the features (each column of X) so
# that each column/feature/variable will have mean = 0 and standard deviation = 1.
# https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler

scalar = preprocessing.StandardScaler().fit(y)
```

In [65]:

```
np.info(X)
```

```
class:  ndarray
shape:  (1254, 3)
strides:  (8, 10032)
itemsize:  8
aligned:  True
contiguous:  False
fortran:  True
data pointer: 0x1ef6793fc50
byteorder:  little
byteswap:  False
type: float64
```

In [66]:

```
np.info(y)
```

```
class:  ndarray
shape:  (1254, 1)
strides:  (8, 8)
itemsize:  8
aligned:  True
contiguous:  True
fortran:  True
data pointer: 0x1ef67a9f260
byteorder:  little
byteswap:  False
type: float64
```

In [67]:

```
y = scalar.transform(y)
```

In [68]:

```
X = scalar.transform(X)
```

Split the dataset into train and test

In [69]:

```
from sklearn.model_selection import TimeSeriesSplit
```

In [70]:

```
# this should give 80/20 split

tscv = TimeSeriesSplit(n_splits=5)

# x_train and y_train follow the same ordering index
# the same for x_test and y_test

for train_index, test_index in tscv.split(X):
    print(train_index[-1:], test_index[-1:])
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
[208] [417]
[417] [626]
[626] [835]
[835] [1044]
[1044] [1253]
```

In [71]:

```
print(train_index[-1:], test_index[-1:])
```

```
[1044] [1253]
```

Model setup

In [72]:

```
# Assign sklearn model to a variable
linear = LinearRegression()
```

In [73]:

```
X_train
```

Out[73]:

```
array([[-1.39725609, -3.16328308, -1.42799451],
       [-1.37363414, -3.16953913, -1.4093625 ],
       [-1.3938626 , -3.17458867, -1.39776018],
       ...,
       [ 1.22928884, -3.16110996,  1.19625397],
       [ 1.24393336, -3.15665632,  1.21300318],
       [ 1.24264716, -3.18534254,  1.22790234]])
```

In [74]:

```
y_train
```

Out[74]:

```
array([[-1.37363414],
       [-1.3938626 ],
       [-1.38054585],
       ...,
       [ 1.24393336],
       [ 1.24264716],
       [ 1.28760448]])
```

In [75]:

```
len(y_train)
```

Out[75]:

```
1045
```

In [76]:

```
y_train.shape
```

Out[76]:

```
(1045, 1)
```

In [77]:

```
# model sucdessfully trained
linear.fit(X_train, y_train.reshape(len(y_train),))
```

Out[77]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

Test model

In [78]:

```
# A simple score measure.  Run your regression forecaset and store your output in a array

# testing model
coefficient_of_determination =linear.score(X_test, y_test)# score returns the coefficient d
coefficient_of_determination
```

Out[78]:

```
0.5955890170447927
```

In [79]:

```
# The following are the 5 featuresets of the testing data
X_test[:5]
```

Out[79]:

```
array([[ 1.28760448, -3.16205058,  1.25153625],
       [ 1.35054266, -3.14595269,  1.2708033 ],
       [ 1.3022464 , -3.16162271,  1.28539481],
       [ 1.29736663, -3.17823508,  1.29608147],
       [ 1.31432112, -3.19356297,  1.31041626]])
```

In [80]:

```
# .predict uses the model to predict the values for the input
forecase_set = linear.predict(X_test)
forecase_set
```

Out[80]:

```
array([1.28920494, 1.35331043, 1.30101162, 1.29907144, 1.32112482,
       1.31540384, 1.2007748 , 1.20616398, 1.2213227 , 1.16444814,
       1.24728205, 1.24655097, 1.2612296 , 1.31892122, 1.29781689,
       1.29790218, 1.30055381, 1.32360073, 1.3702867 , 1.34486053,
       1.42159904, 1.42705843, 1.40105754, 1.36989731, 1.37077125,
       1.29701157, 1.2161397 , 1.23192995, 1.13637743, 1.10904377,
       1.12726642, 1.12961984, 0.94089955, 0.84351022, 1.02161434,
       0.91504037, 0.79654642, 0.86691464, 1.02133253, 1.0519254 ,
       1.13188885, 1.28563091, 1.26334798, 1.24923491, 1.23839247,
       1.27146034, 1.34839352, 1.43445316, 1.41797979, 1.41360211,
       1.33749117, 1.36782362, 1.38499233, 1.37394434, 1.33550304,
       1.38683503, 1.46506869, 1.50373056, 1.45956998, 1.42421481,
       1.43321131, 1.41312648, 1.34092471, 1.34330615, 1.24057979,
       1.17569839, 1.08037619, 1.28124382, 1.16125707, 1.10457105,
       1.16092392, 1.11846375, 1.16247261, 1.24973293, 1.27407936,
       1.15988066, 1.21068026, 1.29596141, 1.2619476 , 1.31090682,
       1.33058558, 1.35342322, 1.41908669, 1.40603148, 1.27500178,
       1.09846491, 1.0922307 , 1.02839508, 1.03890017, 1.05910535,
       1.00461396, 1.08517305, 1.18499458, 1.37051235, 1.37768791,
       1.55802443, 1.59901344, 1.61138549, 1.65703982, 1.72276112,
       1.70179727, 1.69075607, 1.65112118, 1.69483137, 1.6633766 ,
       1.64220866, 1.68268936, 1.66636816, 1.6966665 , 1.69369563,
       1.7023438 , 1.68707478, 1.67552046, 1.66082429, 1.74803513,
       1.78732431, 1.82147005, 1.84373017, 1.82919868, 1.78478967,
       1.77438515, 1.80196051, 1.75720757, 1.76058061, 1.71279818,
       1.70771621, 1.63083271, 1.65043274, 1.62611145, 1.60633116,
       1.53938646, 1.59825456, 1.59126697, 1.61991343, 1.61531634,
       1.66656646, 1.58291861, 1.62074028, 1.69350721, 1.75096414,
       1.74641921, 1.68402079, 1.77150676, 1.77650886, 1.75840404,
       1.78050385, 1.75125766, 1.78928355, 1.77881926, 1.78007945,
       1.82024196, 1.8647496 , 1.84711553, 1.76437087, 1.7417107 ,
       1.75226672, 2.03998857, 2.18579212, 2.19053836, 2.23547052,
       2.1862553 , 2.18742798, 2.23301512, 2.21088149, 2.24699602,
       2.27260756, 2.2814471 , 2.36429547, 2.47416325, 2.41349782,
       2.40654302, 2.41468771, 2.41929708, 2.43727648, 2.48348367,
       2.52870827, 2.61314082, 2.66620115, 2.73262686, 2.75346457,
       2.71353247, 2.6187489 , 2.57489445, 2.49762888, 2.63967609,
       2.56414686, 2.69776534, 2.64137521, 2.47357638, 2.50165991,
       2.50026661, 2.53375079, 2.47532895, 2.55794175, 2.59724071,
       2.54168287, 2.66859507, 2.68773391, 2.71925602, 2.78065053,
       2.84947403, 2.74131076, 2.64801108, 2.6440007 , 2.71843594,
       2.43665057, 2.39365684, 2.61006889, 2.45858187])
```

In [81]:

```
# The first 5 predictions, compare to the featureset above
forecase_set[:5]
```

Out[81]:

```
array([1.28920494, 1.35331043, 1.30101162, 1.29907144, 1.32112482])
```

In [82]:

```
# Here we can see what the actual labels were for the feature_sets
y_test[:5]
```

Out[82]:

```
array([[1.35054266],
       [1.3022464 ],
       [1.29736663],
       [1.31432112],
       [1.31432112]])
```

In [83]:

```
# reverse the scaling back to dollars
c = scalar.inverse_transform(forecase_set)
```

In [84]:

```
bunchZeroes=[0]*(len(data)-len(forecase_set))
```

In [85]:

```
data['pred']=  bunchZeroes+ c.tolist()
```

In [86]:

```
data.tail(15)
```

Out[86]:

date	open	high	low	close	volume	momentum	log_ret	crash	down	up	...
2018-09-25	219.75	222.82	219.7000	222.19	24554379	1.40	0.006321	0	0	1	...
2018-09-26	221.00	223.75	219.7600	220.42	23984706	-1.77	-0.007998	1	0	0	...
2018-09-27	223.82	226.44	223.5400	224.95	30181227	4.53	0.020343	0	0	0	...
2018-09-28	224.79	225.84	224.0200	225.74	22929364	0.79	0.003506	0	0	1	...
2018-10-01	227.95	229.42	226.3500	227.26	23600802	1.52	0.006711	0	0	1	...
2018-10-02	227.25	230.00	226.6300	229.28	24788170	2.02	0.008849	0	0	0	...
2018-10-03	230.05	233.47	229.7800	232.07	28654799	2.79	0.012095	0	0	0	...
2018-10-04	230.78	232.35	226.7300	227.99	32042000	-4.08	-0.017737	1	0	0	...
2018-10-05	227.96	228.41	220.5800	224.29	33580463	-3.70	-0.016362	1	0	0	...
2018-10-08	222.21	224.80	220.2000	223.77	29663923	-0.52	-0.002321	0	1	0	...
2018-10-09	223.64	227.27	222.2462	226.87	26891029	3.10	0.013758	0	0	0	...
2018-10-10	225.46	226.35	216.0500	216.36	41990554	-10.51	-0.047433	1	0	0	...
2018-10-11	214.52	219.50	212.3200	214.45	53124392	-1.91	-0.008867	1	0	0	...
2018-10-12	220.42	222.88	216.8400	222.11	40337851	7.66	0.035096	0	0	0	...
2018-10-15	221.16	221.83	217.2700	217.36	30791007	-4.75	-0.021618	1	0	0	...

15 rows × 21 columns

In [87]:

```
data.head()
```

Out[87]:

date	open	high	low	close	volume	momentum	log_ret	crash	down	up
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0	0
2013-10-24	68.6722	69.6493	68.3386	69.5760	96191095	0.9091	0.013152	0	0	0
2013-10-25	69.4982	69.7487	68.6866	68.7975	84448133	-0.7785	-0.011252	1	0	0
2013-10-28	69.2006	69.4570	68.4380	69.3100	137610123	0.5125	0.007422	0	0	1
2013-10-29	70.1463	70.5361	67.3040	67.5836	158952115	-1.7264	-0.025224	1	0	0

5 rows × 21 columns

In [88]:

```
result= data[['close','label','pred']]
```

In [89]:

```
len(result)
```

Out[89]:

```
1254
```

In [90]:

```
len(data)
```

Out[90]:

```
1254
```

In [91]:

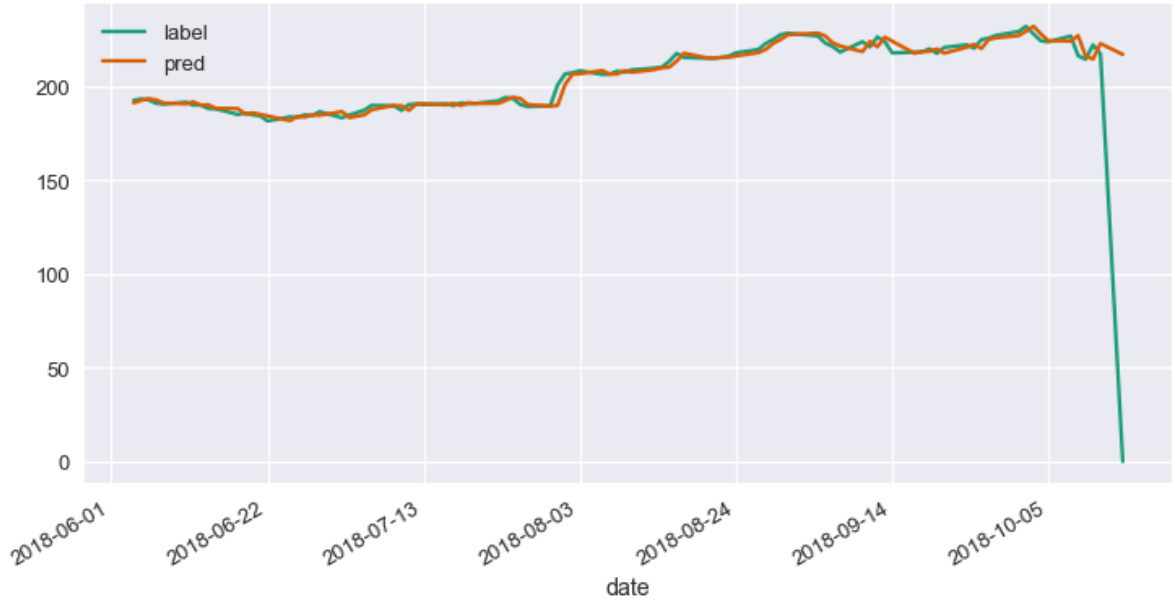
```
result= data[['close','label','pred']]
```

In [92]:

```
data[['label', 'pred']][data.index > pd.Timestamp('2018-06-01')].plot(figsize=(12,6))
```

Out[92]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef68d88438>
```



In [93]:

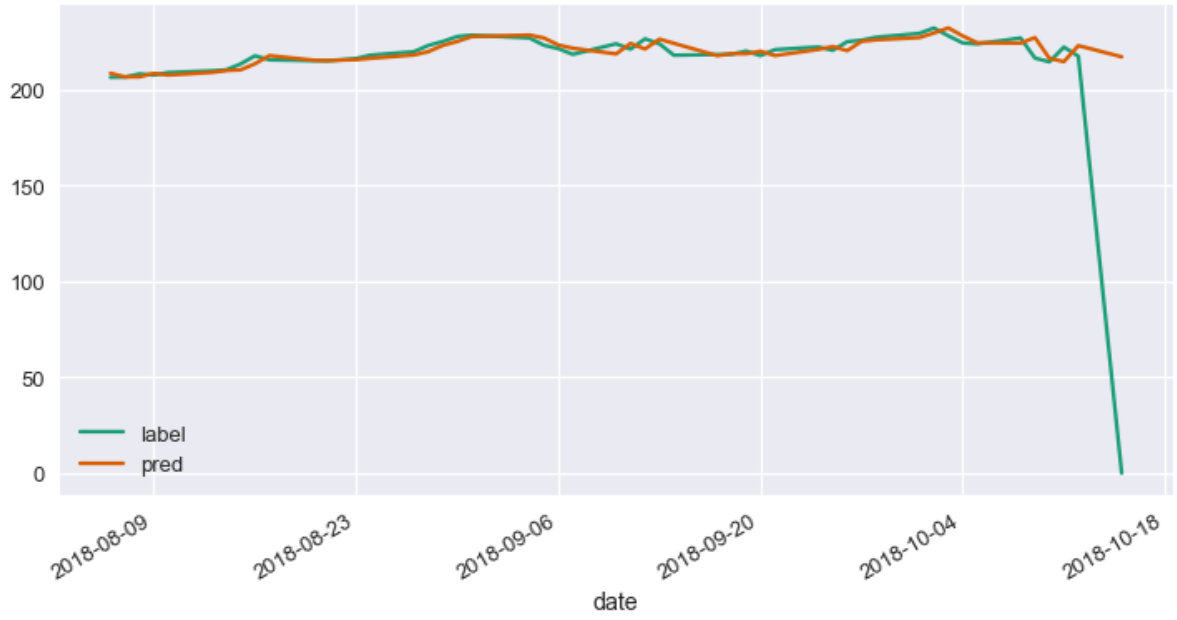
```
result = result[result.pred > 0]
```

In [94]:

```
result[['label', 'pred']][-50:].plot(figsize=(12,6))
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef68df0fd0>
```

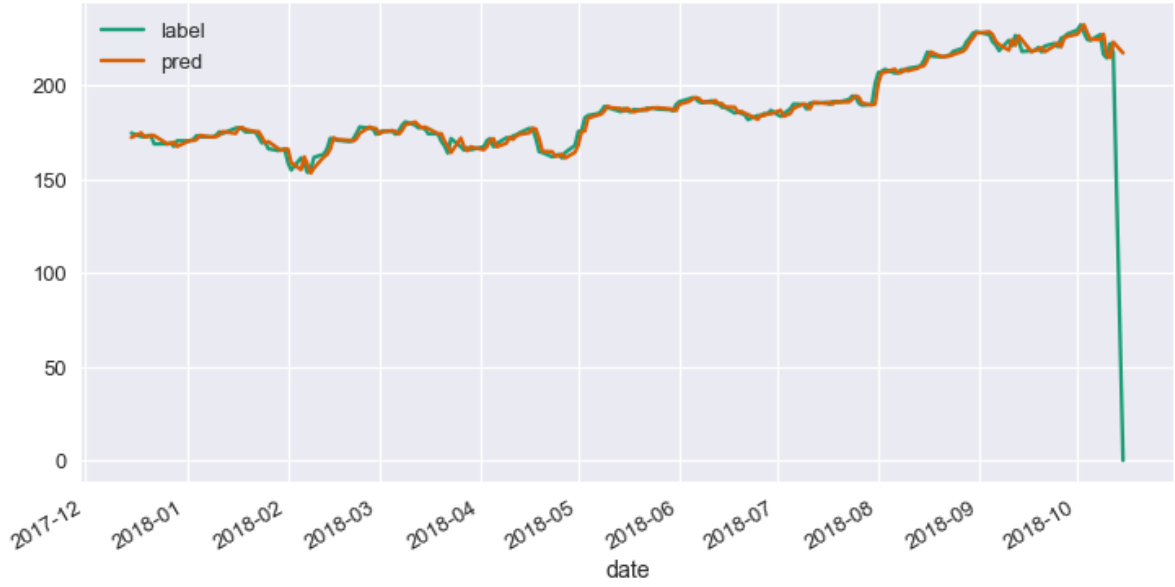


In [95]:

```
result[['label', 'pred']].plot(figsize=(12,6))
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef68e3f630>
```




```
print(len(direction))
print(len(result))
```

```
# Here am trying to detect changes in the direction
np.info(direction)
dir = np.array(direction)
dir
```

In [96]:

```
result.columns
```

Out[96]:

```
Index(['close', 'label', 'pred'], dtype='object')
```

In [97]:

```
result.head()
```

Out[97]:

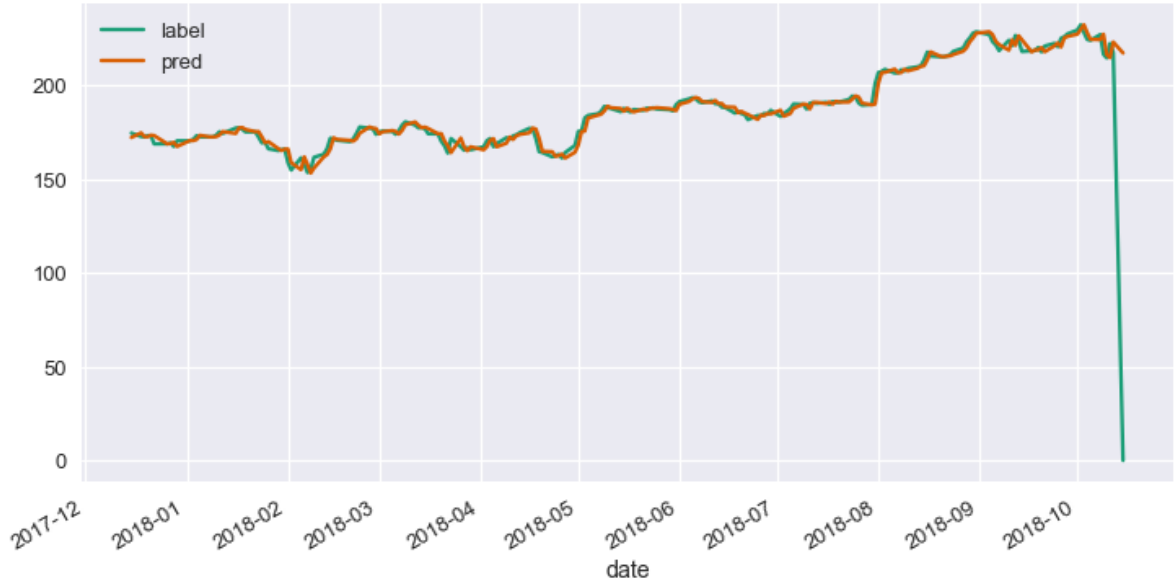
date	close	label	pred
2017-12-15	171.9948	174.4170	172.056394
2017-12-18	174.4170	172.5583	174.523519
2017-12-19	172.5583	172.3705	172.510779
2017-12-20	172.3705	173.0230	172.436110
2017-12-21	173.0230	173.0230	173.284843

In [98]:

```
# print(direction)
result[['label', 'pred']].plot(figsize=(12,6))
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef698fae10>
```



In [99]:

```
trueExp = []
predExp = []

def conf_matrix(close, label, pred):
    for d in range(len(pred)):
        trueExp.append(True) if (label[d] > close[d]) else trueExp.append(False)
        predExp.append(True) if (pred[d] > close[d]) else predExp.append(False)


conf_matrix(data['close'], data['label'], data['pred'])
```

In [100]:

```
# print("predExp: ", predExp)
# print("trueExp: ", trueExp)

print("\nCONFUSION MATRIX:")

# How many predictions are correct?
b = np.array(trueExp).sum()
hits = len(predExp) - (np.array(predExp) ^ np.array(trueExp)).sum()
print(hits, "hits out of ", len(data['pred']))
print("Accuracy: ", 100 * hits/(len(predExp)),"%")
```

```
CONFUSION MATRIX:
611 hits out of  1254
Accuracy:  48.72408293460925 %
```

In [101]:

```
# Measuring the error

from sklearn import metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error, confusion_matrix

mae = mean_absolute_error(data['label'],data['pred'])
mse = mean_squared_error(data['label'],data['pred'])


def mean_absolute_percentage_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

k=len(result['pred'])
y_pred = result['pred'][:k-3]
y_true = result['label'][:k-3]

mape = mean_absolute_percentage_error(y_true, y_pred)
```

In [102]:

```
# mape_result = mape(np.array(data['pred']), np.array(data['label']))
# g = 100 * mape_result

print('Mean Absolute Error: %f' %mae)
print('Mean Squared Error: %f' %mse)
print('Mean Absolute Percentage Error: %f' %mape)
```

```
Mean Absolute Error: 91.797381
Mean Squared Error: 10593.622091
Mean Absolute Percentage Error: 1.062298
```

In [103]:

```
close_val= np.array(result['close'])
act_val= np.array(result['label'])
pred_val= np.array(result['pred'])

original_value = []
prediction_value = []


for i in range(len(pred_val)):
    original_value.append(True) if (act_val[i] > close_val[i]) else original_value.append(F
    prediction_value.append(True) if (pred_val[i] > close_val[i]) else prediction_value.app


original_value = np.array(original_value)
prediction_value = np.array(prediction_value)

#original_value
#prediction_value
```

In [104]:

```
TP = sum(np.bitwise_and(original_value == True, prediction_value == True))
FN = sum(np.bitwise_and(original_value == True, prediction_value == False))
TN = sum(np.bitwise_and(original_value == False, prediction_value == False))
FP = sum(np.bitwise_and(original_value == False, prediction_value == True))
```

In [105]:

```
print("True Positives TP:",TP)
print("False Negatives FN:",FN)
print("True Negatives TN:",TN)
print("False Positives FP:",FP)
```

```
True Positives TP: 86
False Negatives FN: 25
True Negatives TN: 24
False Positives FP: 74
```

In [106]:

```
total = TN + FN

print("ACCURACY: Overall, how often is this model correct?", (TP+TN)/total)
print("MISCLASSIFICATION RATE: Overall, how often is it wrong?", (FP+FN)/total)
print("TRUE POSITIVE RATE: When it is actually yes, how often does it predict yes?", (TP)/F
print("FALSE POSITIVE RATE: When it is actual No, how often does it predict yes?", (FP)/TN+
print("\n")
```

```
ACCURACY: Overall, how often is this model correct? 2.2448979591836733
MISCLASSIFICATION RATE: Overall, how often is it wrong? 2.020408163265306
TRUE POSITIVE RATE: When it is actually yes, how often does it predict yes?
89.44
FALSE POSITIVE RATE: When it is actual No, how often does it predict yes? 7
7.08333333333333
```

In [ ]:

```
```

Step 5 - add 2 more features from the “Type 2” category of features presented in the paper

In [107]:

```
data_new = df
data_new['ASY_3'] = (data_new.close - data_new.close.shift(1)).rolling(3).sum()/3

data_new.head()
```

Out[107]:

date	open	high	low	close	volume	momentum	log_ret	crash	down	up
2013-10-23	67.8873	68.7598	67.8873	68.6669	78431122	0.6661	0.009748	0	0	0
2013-10-24	68.6722	69.6493	68.3386	69.5760	96191095	0.9091	0.013152	0	0	0
2013-10-25	69.4982	69.7487	68.6866	68.7975	84448133	-0.7785	-0.011252	1	0	0
2013-10-28	69.2006	69.4570	68.4380	69.3100	137610123	0.5125	0.007422	0	0	1
2013-10-29	70.1463	70.5361	67.3040	67.5836	158952115	-1.7264	-0.025224	1	0	0

5 rows × 21 columns

In [108]:

```
data_new['MA_5'] = data_new.close.rolling(5).mean()
data_new['label'] = data_new['close'].shift(-5)
data_new[['close','MA_5','ASY_3']].head(30)
```

Out[108]:

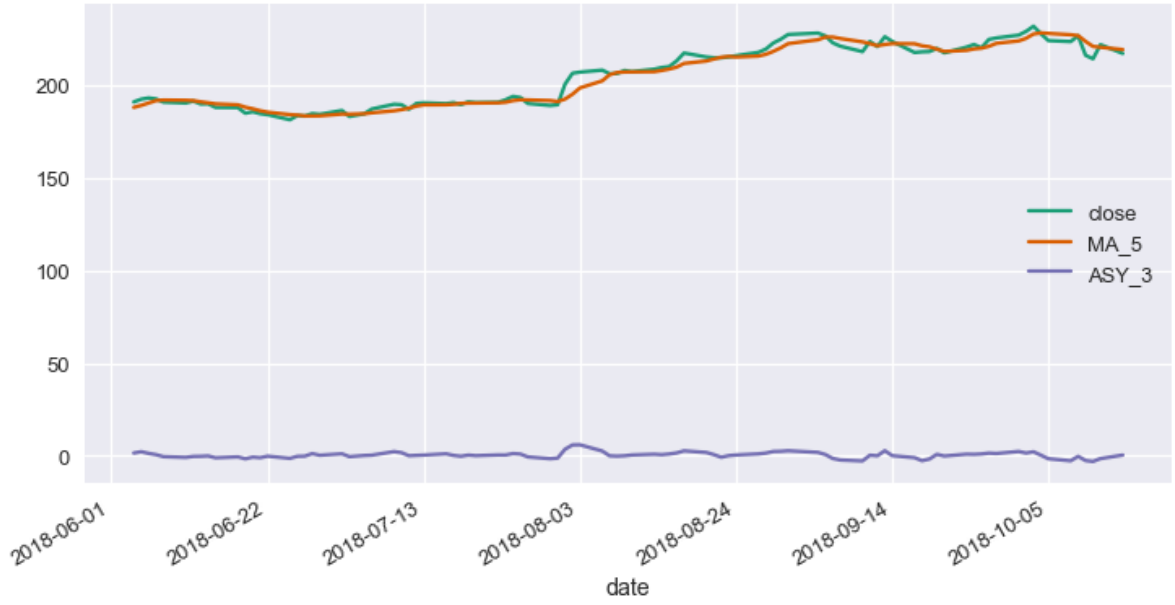
date	close	MA_5	ASY_3
2013-10-23	68.6669	NaN	NaN
2013-10-24	69.5760	NaN	NaN
2013-10-25	68.7975	NaN	NaN
2013-10-28	69.3100	NaN	0.214367
2013-10-29	67.5836	68.78680	-0.664133
2013-10-30	68.6586	68.78514	-0.046300
2013-10-31	68.3716	68.54426	-0.312800
2013-11-01	68.0221	68.38918	0.146167
2013-11-04	68.9011	68.30740	0.080833
2013-11-05	68.7309	68.53686	0.119767
2013-11-06	68.5363	68.51240	0.171400
2013-11-07	67.4275	68.32358	-0.491200
2013-11-08	68.4890	68.41696	-0.080633
2013-11-11	68.2901	68.29476	-0.082067
2013-11-12	68.4166	68.23190	0.329700
2013-11-13	68.4987	68.22438	0.003233
2013-11-14	69.4889	68.63666	0.399600
2013-11-15	69.0720	68.75326	0.218467
2013-11-18	68.2349	68.74222	-0.087933
2013-11-19	68.3561	68.73012	-0.377600
2013-11-20	67.7575	68.58188	-0.438167
2013-11-21	68.5648	68.39706	0.109967
2013-11-22	68.3890	68.26046	0.010967
2013-11-25	68.9074	68.39496	0.383300
2013-11-26	70.1783	68.75940	0.537833
2013-11-27	71.8308	69.57406	1.147267
2013-11-29	73.1610	70.49330	1.417867
2013-12-02	72.5242	71.32034	0.781967
2013-12-03	74.5098	72.44082	0.893000
2013-12-04	74.3359	73.27234	0.391633

In [109]:

```
data[['close','MA_5', 'ASY_3']][data.index > pd.Timestamp('2018-06-01')].plot(figsize=(12,6
```

Out[109]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef68db9f60>
```

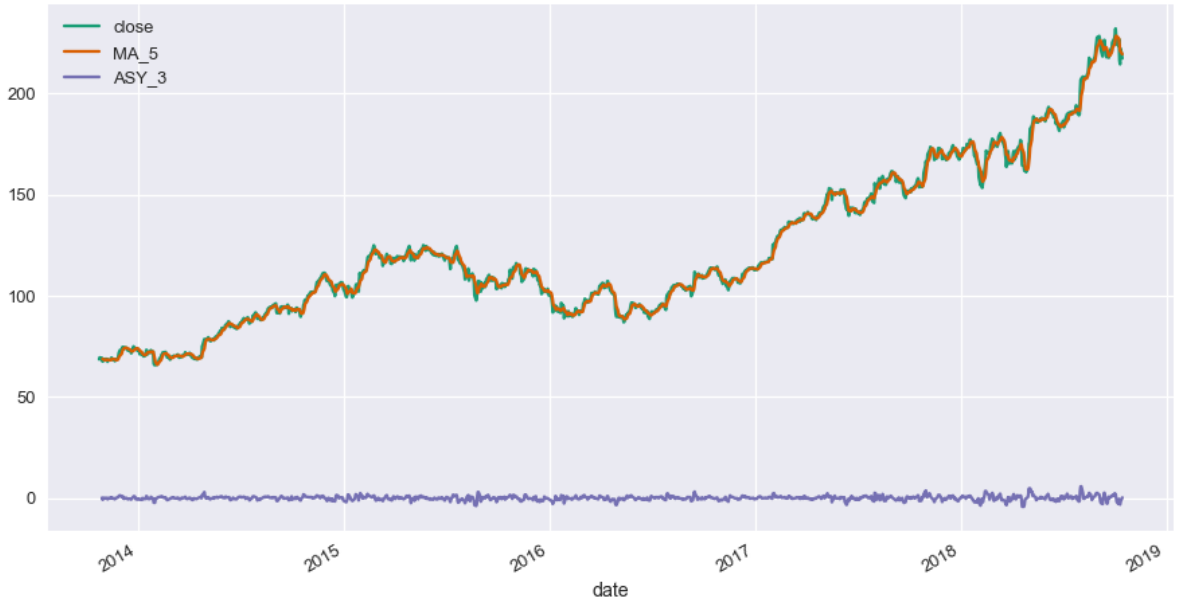


In [110]:

```
data[['close','MA_5', 'ASY_3']].plot(figsize=(15,8))
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ef699f0c50>
```



In [111]:

```
data[['close','label','pred','MA_5','ASY_3']]
```

Out[111]:

date	close	label	pred	MA_5	ASY_3
2013-10-23	68.6669	68.6586	0.000000	NaN	NaN
2013-10-24	69.5760	68.3716	0.000000	NaN	NaN
2013-10-25	68.7975	68.0221	0.000000	NaN	NaN
2013-10-28	69.3100	68.9011	0.000000	NaN	0.214367
2013-10-29	67.5836	68.7309	0.000000	68.78680	-0.664133
2013-10-30	68.6586	68.5363	0.000000	68.78514	-0.046300
2013-10-31	68.3716	67.4275	0.000000	68.54426	-0.312800
2013-11-01	68.0221	68.4890	0.000000	68.38918	0.146167
2013-11-04	68.9011	68.2901	0.000000	68.30740	0.080833
2013-11-05	68.7309	68.4166	0.000000	68.53686	0.119767
2013-11-06	68.5363	68.4987	0.000000	68.51240	0.171400
2013-11-07	67.4275	69.4889	0.000000	68.32358	-0.491200
2013-11-08	68.4890	69.0720	0.000000	68.41696	-0.080633
2013-11-11	68.2901	68.2349	0.000000	68.29476	-0.082067
2013-11-12	68.4166	68.3561	0.000000	68.23190	0.329700
2013-11-13	68.4987	67.7575	0.000000	68.22438	0.003233
2013-11-14	69.4889	68.5648	0.000000	68.63666	0.399600
2013-11-15	69.0720	68.3890	0.000000	68.75326	0.218467
2013-11-18	68.2349	68.9074	0.000000	68.74222	-0.087933
2013-11-19	68.3561	70.1783	0.000000	68.73012	-0.377600
2013-11-20	67.7575	71.8308	0.000000	68.58188	-0.438167
2013-11-21	68.5648	73.1610	0.000000	68.39706	0.109967
2013-11-22	68.3890	72.5242	0.000000	68.26046	0.010967
2013-11-25	68.9074	74.5098	0.000000	68.39496	0.383300
2013-11-26	70.1783	74.3359	0.000000	68.75940	0.537833
2013-11-27	71.8308	74.7175	0.000000	69.57406	1.147267
2013-11-29	73.1610	73.6807	0.000000	70.49330	1.417867
2013-12-02	72.5242	74.5240	0.000000	71.32034	0.781967
2013-12-03	74.5098	74.4082	0.000000	72.44082	0.893000
2013-12-04	74.3359	73.8570	0.000000	73.27234	0.391633
...	...	...	...	...	...
2018-09-04	228.3600	223.8500	228.408994	224.74000	1.793333
2018-09-05	226.8700	221.0700	226.872192	226.17400	0.613333
2018-09-06	223.1000	226.4100	223.224409	226.19800	-1.510000

date	close	label	pred	MA_5	ASY_3
2018-09-07	221.3000	223.8400	221.536653	225.45200	-2.353333
2018-09-10	218.3300	217.8800	218.563058	223.59200	-2.846667
2018-09-11	223.8500	218.2400	224.029800	222.69000	0.250000
2018-09-12	221.0700	218.3700	221.123028	221.53000	-0.076667
2018-09-13	226.4100	220.0300	226.265387	222.19200	2.693333
2018-09-14	223.8400	217.6600	224.095191	222.70000	-0.003333
2018-09-17	217.8800	220.7900	217.637388	222.61000	-1.063333
2018-09-18	218.2400	222.1900	218.718194	221.48800	-2.723333
2018-09-19	218.3700	220.4200	218.664572	220.94800	-1.823333
2018-09-20	220.0300	224.9500	219.953224	219.67200	0.716667
2018-09-21	217.6600	225.7400	217.704837	218.43600	-0.193333
2018-09-24	220.7900	227.2600	220.884222	219.01800	0.806667
2018-09-25	222.1900	229.2800	222.396658	219.80800	0.720000
2018-09-26	220.4200	232.0700	220.258493	220.21800	0.920000
2018-09-27	224.9500	227.9900	225.142758	221.20200	1.386667
2018-09-28	225.7400	224.2900	225.879324	222.81800	1.183333
2018-10-01	227.2600	223.7700	227.092464	224.11200	2.280000
2018-10-02	229.2800	226.8700	229.455256	225.53000	1.443333
2018-10-03	232.0700	216.3600	232.103955	227.86000	2.110000
2018-10-04	227.9900	214.4500	227.941250	228.46800	0.243333
2018-10-05	224.2900	222.1100	224.350575	228.17800	-1.663333
2018-10-08	223.7700	217.3600	224.196234	227.48000	-2.766667
2018-10-09	226.8700	NaN	227.060903	226.99800	-0.373333
2018-10-10	216.3600	NaN	216.216284	223.85600	-2.643333
2018-10-11	214.4500	NaN	214.561654	221.14800	-3.106667
2018-10-12	222.1100	NaN	222.890355	220.71200	-1.586667
2018-10-15	217.3600	NaN	217.060319	219.43000	0.333333

1254 rows × 5 columns

In [112]:

```
data[['close','label','pred','MA_5','ASY_3','crash','down','up','jump']]
```

Out[112]:

date	close	label	pred	MA_5	ASY_3	crash	down	up	jump
2013-10-23	68.6669	68.6586	0.000000	NaN	NaN	0	0	0	1
2013-10-24	69.5760	68.3716	0.000000	NaN	NaN	0	0	0	1
2013-10-25	68.7975	68.0221	0.000000	NaN	NaN	1	0	0	0
2013-10-28	69.3100	68.9011	0.000000	NaN	0.214367	0	0	1	0
2013-10-29	67.5836	68.7309	0.000000	68.78680	-0.664133	1	0	0	0
2013-10-30	68.6586	68.5363	0.000000	68.78514	-0.046300	0	0	0	1
2013-10-31	68.3716	67.4275	0.000000	68.54426	-0.312800	0	1	0	0
2013-11-01	68.0221	68.4890	0.000000	68.38918	0.146167	0	1	0	0
2013-11-04	68.9011	68.2901	0.000000	68.30740	0.080833	0	0	0	1
2013-11-05	68.7309	68.4166	0.000000	68.53686	0.119767	0	1	0	0
2013-11-06	68.5363	68.4987	0.000000	68.51240	0.171400	0	1	0	0
2013-11-07	67.4275	69.4889	0.000000	68.32358	-0.491200	1	0	0	0
2013-11-08	68.4890	69.0720	0.000000	68.41696	-0.080633	0	0	0	1
2013-11-11	68.2901	68.2349	0.000000	68.29476	-0.082067	0	1	0	0
2013-11-12	68.4166	68.3561	0.000000	68.23190	0.329700	0	0	1	0
2013-11-13	68.4987	67.7575	0.000000	68.22438	0.003233	0	0	1	0
2013-11-14	69.4889	68.5648	0.000000	68.63666	0.399600	0	0	0	1
2013-11-15	69.0720	68.3890	0.000000	68.75326	0.218467	1	0	0	0
2013-11-18	68.2349	68.9074	0.000000	68.74222	-0.087933	1	0	0	0
2013-11-19	68.3561	70.1783	0.000000	68.73012	-0.377600	0	0	1	0
2013-11-20	67.7575	71.8308	0.000000	68.58188	-0.438167	1	0	0	0
2013-11-21	68.5648	73.1610	0.000000	68.39706	0.109967	0	0	0	1
2013-11-22	68.3890	72.5242	0.000000	68.26046	0.010967	0	1	0	0
2013-11-25	68.9074	74.5098	0.000000	68.39496	0.383300	0	0	1	0
2013-11-26	70.1783	74.3359	0.000000	68.75940	0.537833	0	0	0	1
2013-11-27	71.8308	74.7175	0.000000	69.57406	1.147267	0	0	0	1
2013-11-29	73.1610	73.6807	0.000000	70.49330	1.417867	0	0	0	1
2013-12-02	72.5242	74.5240	0.000000	71.32034	0.781967	1	0	0	0
2013-12-03	74.5098	74.4082	0.000000	72.44082	0.893000	0	0	0	1
2013-12-04	74.3359	73.8570	0.000000	73.27234	0.391633	0	1	0	0
...	...	...	...	...	...	...	...	...	...
2018-09-04	228.3600	223.8500	228.408994	224.74000	1.793333	0	0	1	0
2018-09-05	226.8700	221.0700	226.872192	226.17400	0.613333	1	0	0	0
2018-09-06	223.1000	226.4100	223.224409	226.19800	-1.510000	1	0	0	0

date	close	label	pred	MA_5	ASY_3	crash	down	up	jump
2018-09-07	221.3000	223.8400	221.536653	225.45200	-2.353333	1	0	0	0
2018-09-10	218.3300	217.8800	218.563058	223.59200	-2.846667	1	0	0	0
2018-09-11	223.8500	218.2400	224.029800	222.69000	0.250000	0	0	0	1
2018-09-12	221.0700	218.3700	221.123028	221.53000	-0.076667	1	0	0	0
2018-09-13	226.4100	220.0300	226.265387	222.19200	2.693333	0	0	0	1
2018-09-14	223.8400	217.6600	224.095191	222.70000	-0.003333	1	0	0	0
2018-09-17	217.8800	220.7900	217.637388	222.61000	-1.063333	1	0	0	0
2018-09-18	218.2400	222.1900	218.718194	221.48800	-2.723333	0	0	1	0
2018-09-19	218.3700	220.4200	218.664572	220.94800	-1.823333	0	1	0	0
2018-09-20	220.0300	224.9500	219.953224	219.67200	0.716667	0	0	1	0
2018-09-21	217.6600	225.7400	217.704837	218.43600	-0.193333	1	0	0	0
2018-09-24	220.7900	227.2600	220.884222	219.01800	0.806667	0	0	0	1
2018-09-25	222.1900	229.2800	222.396658	219.80800	0.720000	0	0	1	0
2018-09-26	220.4200	232.0700	220.258493	220.21800	0.920000	1	0	0	0
2018-09-27	224.9500	227.9900	225.142758	221.20200	1.386667	0	0	0	1
2018-09-28	225.7400	224.2900	225.879324	222.81800	1.183333	0	0	1	0
2018-10-01	227.2600	223.7700	227.092464	224.11200	2.280000	0	0	1	0
2018-10-02	229.2800	226.8700	229.455256	225.53000	1.443333	0	0	0	1
2018-10-03	232.0700	216.3600	232.103955	227.86000	2.110000	0	0	0	1
2018-10-04	227.9900	214.4500	227.941250	228.46800	0.243333	1	0	0	0
2018-10-05	224.2900	222.1100	224.350575	228.17800	-1.663333	1	0	0	0
2018-10-08	223.7700	217.3600	224.196234	227.48000	-2.766667	0	1	0	0
2018-10-09	226.8700	NaN	227.060903	226.99800	-0.373333	0	0	0	1
2018-10-10	216.3600	NaN	216.216284	223.85600	-2.643333	1	0	0	0
2018-10-11	214.4500	NaN	214.561654	221.14800	-3.106667	1	0	0	0
2018-10-12	222.1100	NaN	222.890355	220.71200	-1.586667	0	0	0	1
2018-10-15	217.3600	NaN	217.060319	219.43000	0.333333	1	0	0	0

1254 rows × 9 columns

Step 6 - design a decision for when to invest and when to sell based on your regression.

The model can be naïve, meaning you can create a rule (if .. X .. then .. Y)

In [113]:

```
data["decision"] = ""

for i in range (1, len(data.close)):
   if data.pred.iloc[i] > data.close.shift(1).iloc[i]:
       data.decision.iloc[i] = "BUY"
       if data.decision.shift(1).iloc[i] == "BUY":
           data.decision.iloc[i] = "HOLD"
   elif data.pred.iloc[i] == data.close.shift(1).iloc[i]:
       data.decision.iloc[i] = "HOLD"
   elif data.pred.iloc[i] < data.close.shift(1).iloc[i]:
       data.decision.iloc[i] = "SELL"
       if data.decision.shift(1).iloc[i] == "SELL":
           data.decision.iloc[i] = "HOLD"
   else:
       data.decision.iloc[i] = "HOLD"
```

```
C:\tools\Anaconda3\lib\site-packages\pandas\core\indexing.py:189: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pand
as-docs/stable/indexing.html#indexing-view-versus-copy)
  self._setitem_with_indexer(indexer, value)
```

In [114]:

```
data[['close','label','pred','decision']]
```

Out[114]:

date	close	label	pred	decision
2013-10-23	68.6669	68.6586	0.000000	
2013-10-24	69.5760	68.3716	0.000000	SELL
2013-10-25	68.7975	68.0221	0.000000	HOLD
2013-10-28	69.3100	68.9011	0.000000	SELL
2013-10-29	67.5836	68.7309	0.000000	HOLD
2013-10-30	68.6586	68.5363	0.000000	SELL
2013-10-31	68.3716	67.4275	0.000000	HOLD
2013-11-01	68.0221	68.4890	0.000000	SELL
2013-11-04	68.9011	68.2901	0.000000	HOLD
2013-11-05	68.7309	68.4166	0.000000	SELL
2013-11-06	68.5363	68.4987	0.000000	HOLD
2013-11-07	67.4275	69.4889	0.000000	SELL
2013-11-08	68.4890	69.0720	0.000000	HOLD
2013-11-11	68.2901	68.2349	0.000000	SELL
2013-11-12	68.4166	68.3561	0.000000	HOLD
2013-11-13	68.4987	67.7575	0.000000	SELL
2013-11-14	69.4889	68.5648	0.000000	HOLD
2013-11-15	69.0720	68.3890	0.000000	SELL
2013-11-18	68.2349	68.9074	0.000000	HOLD
2013-11-19	68.3561	70.1783	0.000000	SELL
2013-11-20	67.7575	71.8308	0.000000	HOLD
2013-11-21	68.5648	73.1610	0.000000	SELL
2013-11-22	68.3890	72.5242	0.000000	HOLD
2013-11-25	68.9074	74.5098	0.000000	SELL
2013-11-26	70.1783	74.3359	0.000000	HOLD
2013-11-27	71.8308	74.7175	0.000000	SELL
2013-11-29	73.1610	73.6807	0.000000	HOLD
2013-12-02	72.5242	74.5240	0.000000	SELL
2013-12-03	74.5098	74.4082	0.000000	HOLD
2013-12-04	74.3359	73.8570	0.000000	SELL
...	...	...	...	...
2018-09-04	228.3600	223.8500	228.408994	BUY
2018-09-05	226.8700	221.0700	226.872192	SELL
2018-09-06	223.1000	226.4100	223.224409	HOLD

date	close	label	pred	decision
2018-09-07	221.3000	223.8400	221.536653	SELL
2018-09-10	218.3300	217.8800	218.563058	HOLD
2018-09-11	223.8500	218.2400	224.029800	BUY
2018-09-12	221.0700	218.3700	221.123028	SELL
2018-09-13	226.4100	220.0300	226.265387	BUY
2018-09-14	223.8400	217.6600	224.095191	SELL
2018-09-17	217.8800	220.7900	217.637388	HOLD
2018-09-18	218.2400	222.1900	218.718194	BUY
2018-09-19	218.3700	220.4200	218.664572	HOLD
2018-09-20	220.0300	224.9500	219.953224	BUY
2018-09-21	217.6600	225.7400	217.704837	SELL
2018-09-24	220.7900	227.2600	220.884222	BUY
2018-09-25	222.1900	229.2800	222.396658	HOLD
2018-09-26	220.4200	232.0700	220.258493	SELL
2018-09-27	224.9500	227.9900	225.142758	BUY
2018-09-28	225.7400	224.2900	225.879324	HOLD
2018-10-01	227.2600	223.7700	227.092464	BUY
2018-10-02	229.2800	226.8700	229.455256	HOLD
2018-10-03	232.0700	216.3600	232.103955	BUY
2018-10-04	227.9900	214.4500	227.941250	SELL
2018-10-05	224.2900	222.1100	224.350575	HOLD
2018-10-08	223.7700	217.3600	224.196234	SELL
2018-10-09	226.8700	NaN	227.060903	BUY
2018-10-10	216.3600	NaN	216.216284	SELL
2018-10-11	214.4500	NaN	214.561654	HOLD
2018-10-12	222.1100	NaN	222.890355	BUY
2018-10-15	217.3600	NaN	217.060319	SELL

1254 rows × 4 columns

In [ ]: