This is my capstone project for the Udacity Machine Learning Nanodegree.

Import the libraries needed.

In [1]:
```
import pandas as pd
import numpy as np
import keras as kr
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from os import listdir
```

```
Using TensorFlow backend.
```

Get the data.

Since we already know the name of the specific stock we are trying to get from the name of the file, we can drop that column in the dataframe.

In [2]:
```
directory = 'sandp500/individual_stocks_5yr'
dir_listing = listdir(directory)
```

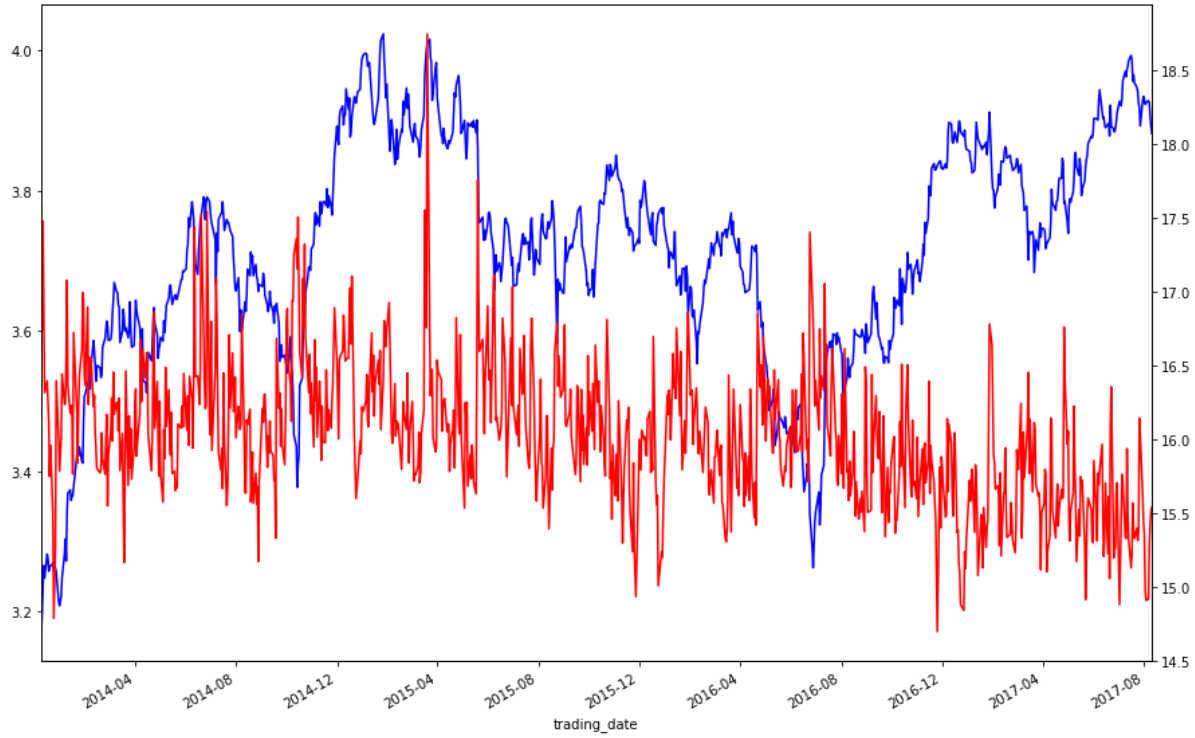
In [3]:
```
symbols_list = []

for symbol in dir_listing:
    symb = symbol.split('_')[0]
    symbols_list.append(symb)

csv_file = '{}/{}_data.csv'.format(directory, symbols_list[0])
dataset = pd.read_csv(csv_file)
```

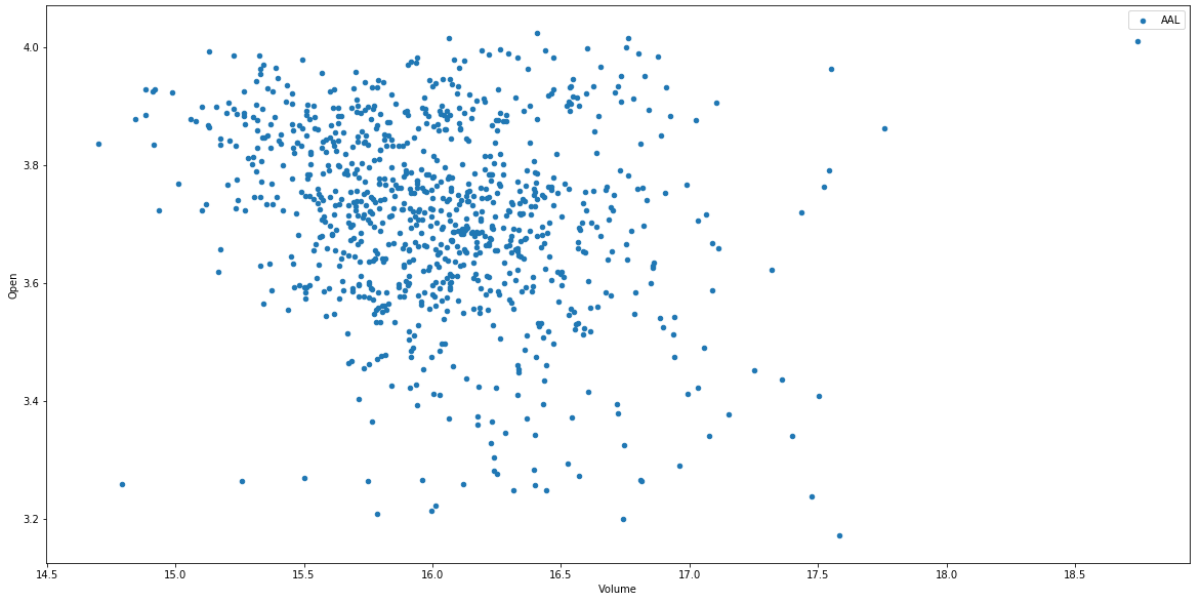
In [4]:
```
def getting_preprocessed_data(symbol):
        csv_file = '{}/{}_data.csv'.format(directory, symbol)
        df = pd.read_csv(csv_file)
        df = df.assign(trading_date = pd.to_datetime(df['Date']))
        df = df.drop('Name', 1)
        df = df.drop('Date', 1)
        df.set_index(['trading_date'], inplace=True)
        df = np.log(df)
        #below was found at https://stackoverflow.com/questions/29314033/pytho
n-pandas-dataframe-remove-empty-cells
        df.replace('', np.nan, inplace=True)
        df.dropna(subset=['Open'], inplace=True)
        return df
```

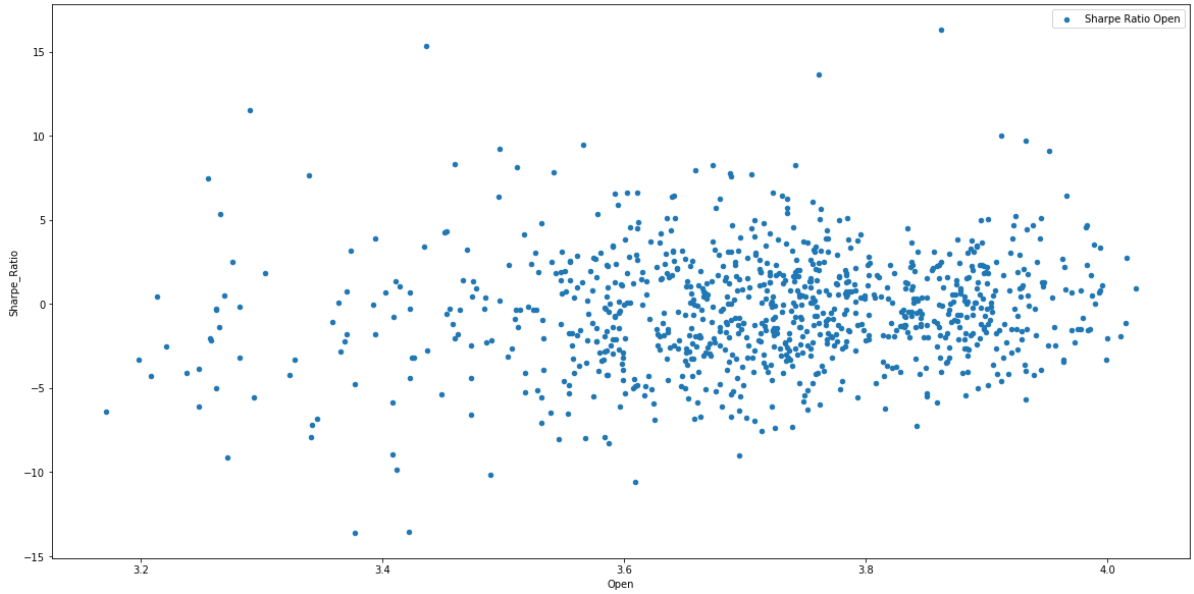
In [5]:
```
def plotting_stocks(symbols_list, amount_of_stocks=0):
    if amount_of_stocks == 0:
        amount_of_stocks = len(symbols_list)

    for symbol in symbols_list[:amount_of_stocks]:
        fig, ax = plt.subplots()
        fig.subplots_adjust(right=0.7)
        df = getting_preprocessed_data(symbol)
        print(symbol)
        df.Open.plot(ax=ax, style='b-', figsize=(20,10))
        # same ax as above since it's automatically added on the right
        df.Volume.plot(ax=ax, style='r-', secondary_y=True, figsize=(20,10))
        plt.show()

        #below is the Daily Returns calculation to put into the Sharpe Ratio.
        df_preprocessed = df.assign(Daily_Returns = np.divide((df.Open - df.Cl
ose), df.Close) * 100)

        #Below is the calculation for the Sharpe Ratio column.
        df_preprocessed = df_preprocessed.assign(Sharpe_Ratio = np.divide((df_
preprocessed.Daily_Returns - 0.046), np.std(np.array([df_preprocessed.Open, df
_preprocessed.High, df_preprocessed.Low, df_preprocessed.Close]))))

        #Below is the rate of change (momentum) for the specific stock.
        df_preprocessed = df_preprocessed.assign(Rate_of_Change = (np.divide(d
f_preprocessed.Close, df_preprocessed.Open) - 1) * 100)

        df_preprocessed.plot.scatter(x='Volume', y='Open', label="AAL", figsiz
e=(20,10))
        plt.show()
        df_preprocessed.plot.scatter(x='Open', y='Sharpe_Ratio', label="Sharpe
 Ratio Open", figsize=(20,10))
        plt.show()
        df_preprocessed.plot.scatter(x='Volume',y='Sharpe_Ratio', label="Sharp
e Ratio Volume", figsize=(20,10), use_index=True)
        plt.show()
        df_preprocessed.plot.scatter(x='Volume',y='Close', label="Volume Clos
e", figsize=(20,10), use_index=True)
        plt.show()
        return df_preprocessed
```

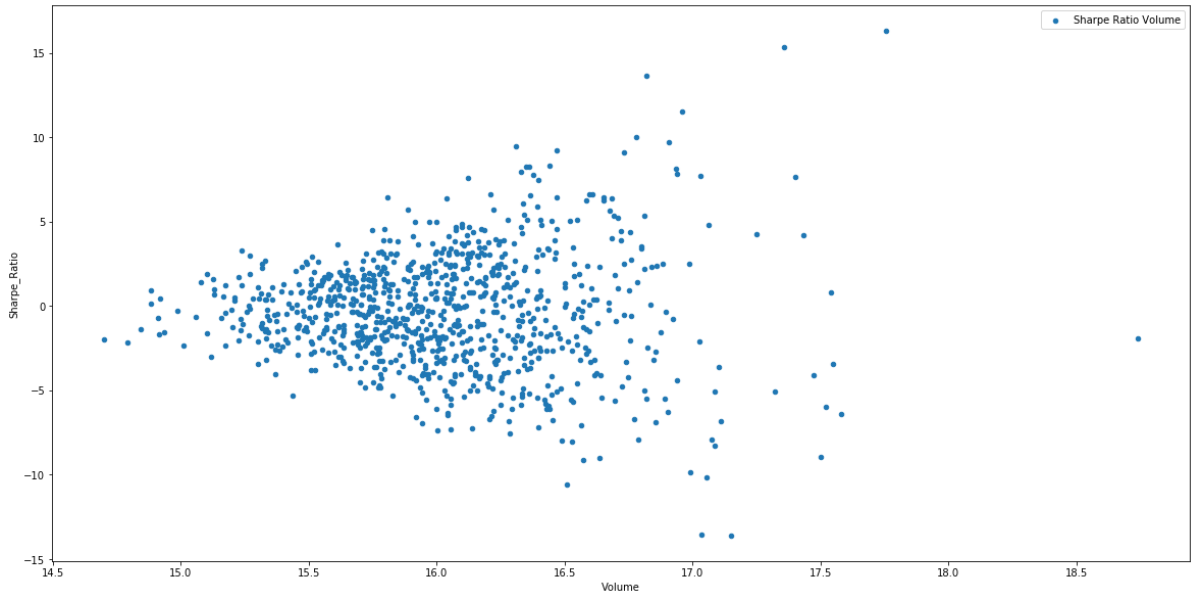
In [6]:
```
# printing out the first four stocks to get an idea of how each stock is indiv
idually represented.
raw_features = plotting_stocks(symbols_list, 1)
```

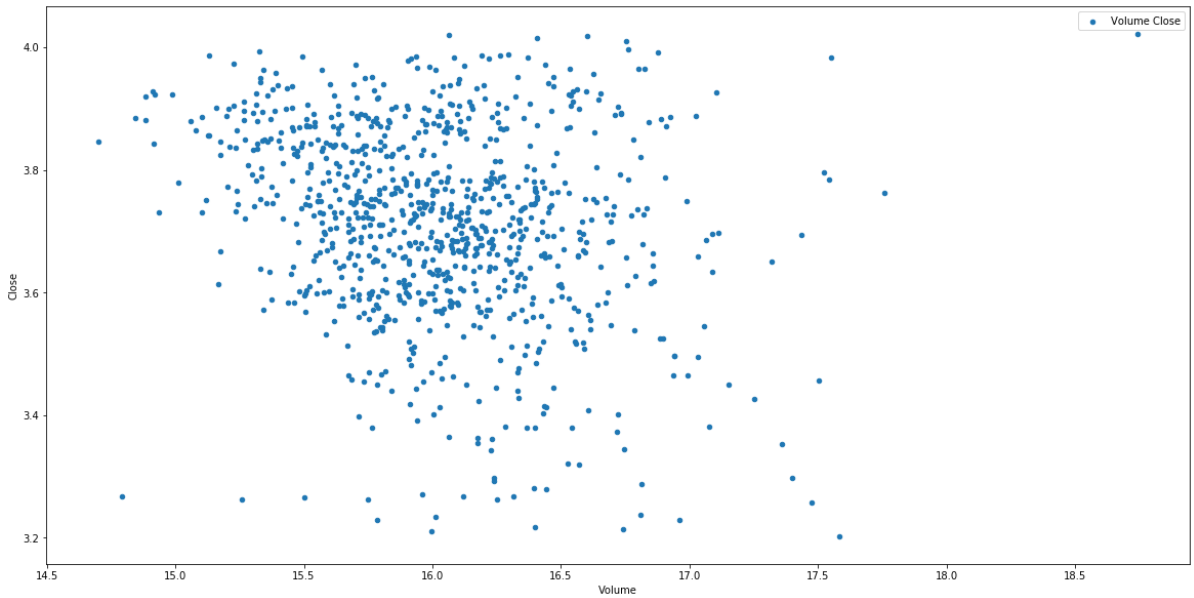
AAL


4.0
3.8
3.6
3.4
3.2
18.5
18.0
17.5
17.0
16.5
16.0
15.5
15.0
14.5
2014-04
2014-08
2014-12
2015-04
2015-08
2015-12
2016-04
2016-08
2016-12
2017-04
2017-08
trading_date


AAL
Open
Volume
4.0
3.8
3.6
3.4
3.2
14.5
15.0
15.5
16.0
16.5
17.0
17.5
18.0
18.5


Sharpe Ratio Open
Sharpe_Ratio
Open
15
10
5
0
−5
−10
−15
3.2
3.4
3.6
3.8
4.0


Sharpe Ratio Volume
Sharpe_Ratio
Volume
15
10
5
0
−5
−10
−15
14.5
15.0
15.5
16.0
16.5
17.0
17.5
18.0
18.5


Volume Close
Close
Volume
4.0
3.8
3.6
3.4
3.2
14.5
15.0
15.5
16.0
16.5
17.0
17.5
18.0
18.5

In [7]:
```
raw_closing = raw_features['Close']
raw_features = raw_features.drop('Close', axis = 1)
display(raw_features.head(n=1))
```

	Open	High	Low	Volume	Daily_Returns	Sharpe_Ratio	Rate
trading_date							
2013-12-09	3.171784	3.236323	3.15487	17.581288	-0.96674	-6.369843	0.976

In [8]:
```
# from the Finding Donors project
scaler = MinMaxScaler()
numerical = ['Open', 'High', 'Low', 'Volume']
raw_features[numerical] = scaler.fit_transform(dataset[numerical])

display(raw_features.head(n=1))
```

	Open	High	Low	Volume	Daily_Returns	Sharpe_Ratio	Rate_of_Cha
trading_date							
2013-12-09	0.0	0.008701	0.0	0.301245	-0.96674	-6.369843	0.976177

In [9]:
```
raw_closing = raw_closing.values.reshape(-1,1)
raw_closing = scaler.fit_transform(raw_closing)
```

In [10]:
```
from sklearn.model_selection import train_test_split
```

In [11]:
```
X_train, X_test, y_train, y_test = train_test_split(raw_features, raw_closing,
 test_size=0.20, random_state=0)

print("Training set has {} samples".format(X_train.shape[0]))
print("Testing set has {} samples".format(X_test.shape[0]))
```

```
Training set has 740 samples
Testing set has 186 samples
```

In [12]:
```
# From my Finding Donors project.

from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
```

In [13]:
```
learner = SVR()
```

In [14]:
```
train = learner.fit(X_train, y_train.ravel())
```

In [15]:
```
pred = learner.predict(X_test)
accuracy = mean_squared_error(y_test, pred)

print(accuracy)
```

```
0.00609825958261
```

This is the RNN-LSTM Section

This section is the main analysis. The above Support Vector Machine section is to compare against and see what type of correlation and accuracy I can achieve.

In [16]:
```
X_train_array = np.array(X_train)
X_test_array = np.array(X_test)

test_sample_size = int(len(X_train_array)*0.80)

X_train_sample_size = X_train_array[:test_sample_size]
X_test_sample_size = X_train_array[test_sample_size:]
y_train_sample_size = y_train[:test_sample_size]
y_test_sample_size = y_test[test_sample_size:]

print(test_sample_size)
print(X_train_sample_size.shape)
print(y_train_sample_size.shape)
```

```
592
(592, 7)
(592, 1)
```

In [41]:
```
train_X = np.reshape(X_train_sample_size, (1, X_train_sample_size.shape[0], X_
train_sample_size.shape[1]))
train_y = np.reshape(y_train_sample_size, (1, y_train_sample_size.shape[0], 1
))
```

In [42]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, LSTM, Flatten
```

In [43]:
```
model = Sequential()
model.add(Dense(7, input_shape=(X_train_sample_size.shape[0], 7)))
model.add(LSTM(7, dropout=0.2, return_sequences=True))
#model.add(Flatten())
model.add(Dense((7)))
```

In [44]:
```
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=['acc
uracy'])
```

In [45]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 592, 7)            56
_________________________________________________________________
lstm_5 (LSTM)                (None, 592, 7)            420
_________________________________________________________________
dense_7 (Dense)              (None, 592, 7)            56
=================================================================
Total params: 532
Trainable params: 532
Non-trainable params: 0
_________________________________________________________________
```

In [46]:
```
print("train_X shape {}".format(train_X.shape))
print("train_y shape {}".format(train_y.shape))
```

```
train_X shape (1, 592, 7)
train_y shape (1, 592, 1)
```

In [47]:
```
model.fit(train_X, train_y, epochs=10, batch_size=7)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-47-7f48a1fcb1dd> in <module>()
----> 1 model.fit(train_X, train_y, epochs=10, batch_size=7)

/usr/local/lib/python3.5/dist-packages/keras/models.py in fit(self, x, y, bat
ch_size, epochs, verbose, callbacks, validation_split, validation_data, shuff
le, class_weight, sample_weight, initial_epoch, **kwargs)
    865                                 class_weight=class_weight,
    866                                 sample_weight=sample_weight,
--> 867                                 initial_epoch=initial_epoch)
    868
    869     def evaluate(self, x, y, batch_size=32, verbose=1,

/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in fit(self,
 x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_d
ata, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, va
lidation_steps, **kwargs)
   1520             class_weight=class_weight,
   1521             check_batch_axis=False,
-> 1522             batch_size=batch_size)
   1523         # Prepare validation data.
   1524         do_validation = False

/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in _standardi
ze_user_data(self, x, y, sample_weight, class_weight, check_batch_axis, batch
_size)
   1380                                     output_shapes,
   1381                                     check_batch_axis=False,
-> 1382                                     exception_prefix='target')
   1383         sample_weights = _standardize_sample_weights(sample_weight,
   1384                                                      self._feed_outpu
t_names)

/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in _standardi
ze_input_data(data, names, shapes, check_batch_axis, exception_prefix)
    142                             ' to have shape ' + str(shapes[i]) +
    143                             ' but got array with shape ' +
--> 144                             str(array.shape))
    145     return arrays
    146

ValueError: Error when checking target: expected dense_7 to have shape (None,
 592, 7) but got array with shape (1, 592, 1)
```

In [ ]:
```

```