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.

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)

print(len(symbols_list))
print(symbols_list[0])
```

```
504
AAL
```

In [4]:
```
csv_file = '{}/{}_data.csv'.format(directory, symbols_list[0])
```

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 [5]:
```
dataset = pd.read_csv(csv_file)
```

In [6]:
```
dataset = dataset.assign(trading_date = pd.to_datetime(dataset['Date']))
```

In [7]:
```
dataset
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Name	trading_date
0	2013-12-09	23.85	25.44	23.45	24.60	43197268	AAL	2013-12-09
1	2013-12-10	24.50	25.17	24.41	24.88	18660625	AAL	2013-12-10
2	2013-12-11	25.48	27.20	25.37	25.99	38843371	AAL	2013-12-11
3	2013-12-12	26.20	26.71	25.45	25.45	19981824	AAL	2013-12-12
4	2013-12-13	25.75	26.30	25.52	26.23	12192421	AAL	2013-12-13
5	2013-12-16	26.63	26.77	26.35	26.61	13190945	AAL	2013-12-16
6	2013-12-17	26.48	26.59	25.95	26.10	11413199	AAL	2013-12-17
7	2013-12-18	25.99	26.23	25.55	26.23	9994162	AAL	2013-12-18
8	2013-12-19	26.12	26.49	25.82	26.12	6916497	AAL	2013-12-19
9	2013-12-20	26.18	26.49	26.14	26.33	8530924	AAL	2013-12-20
10	2013-12-23	26.29	26.49	26.05	26.18	5403084	AAL	2013-12-23
11	2013-12-24	26.00	26.26	26.00	26.25	2652974	AAL	2013-12-24
12	2013-12-26	26.12	26.36	25.98	26.13	4226639	AAL	2013-12-26
13	2013-12-27	25.95	26.10	24.91	24.94	13227018	AAL	2013-12-27
14	2013-12-30	24.87	25.25	24.65	24.78	8841369	AAL	2013-12-30
15	2013-12-31	24.74	25.25	24.63	25.25	7168395	AAL	2013-12-31
16	2014-01-02	25.07	25.82	25.06	25.36	8998943	AAL	2014-01-02
17	2014-01-03	25.75	26.75	25.51	26.54	13836062	AAL	2014-01-03
18	2014-01-06	26.62	27.20	26.60	27.03	11272273	AAL	2014-01-06
19	2014-01-07	27.20	27.40	26.67	26.90	11288775	AAL	2014-01-07
20	2014-01-08	26.37	27.68	26.35	27.63	15736891	AAL	2014-01-08
21	2014-01-09	28.24	29.60	28.20	29.42	26056445	AAL	2014-01-09
22	2014-01-10	29.05	29.83	28.75	29.35	12824160	AAL	2014-01-10
23	2014-01-13	29.18	29.53	28.58	28.65	10591701	AAL	2014-01-13
24	2014-01-14	28.75	29.04	28.71	28.87	10601529	AAL	2014-01-14
25	2014-01-15	28.90	29.44	28.70	28.84	11192558	AAL	2014-01-15
26	2014-01-16	28.94	29.39	28.70	29.34	7034698	AAL	2014-01-16
27	2014-01-17	29.30	30.02	29.17	30.02	18304949	AAL	2014-01-17
28	2014-01-21	30.66	30.80	30.20	30.66	10612821	AAL	2014-01-21
29	2014-01-22	30.71	31.24	30.65	31.20	7583631	AAL	2014-01-22
...	...	...	...	...	...	...	...	...
896	2017-06-30	49.92	50.51	49.60	50.32	6618701	AAL	2017-06-30

	Date	Open	High	Low	Close	Volume	Name	trading_date
897	2017-07-03	50.78	51.28	50.37	50.39	2906524	AAL	2017-07-03
898	2017-07-05	50.44	51.54	50.18	51.26	5157516	AAL	2017-07-05
899	2017-07-06	50.97	52.58	50.86	52.05	7021422	AAL	2017-07-06
900	2017-07-07	52.30	53.38	52.18	53.03	6596952	AAL	2017-07-07
901	2017-07-10	52.98	53.16	52.23	52.66	4600014	AAL	2017-07-10
902	2017-07-11	52.58	52.66	51.47	51.61	4545699	AAL	2017-07-11
903	2017-07-12	53.12	53.82	52.45	53.80	8348363	AAL	2017-07-12
904	2017-07-13	53.40	54.48	53.15	53.81	5346686	AAL	2017-07-13
905	2017-07-14	53.80	54.28	53.34	54.22	4537895	AAL	2017-07-14
906	2017-07-17	54.21	54.28	53.85	53.87	3727804	AAL	2017-07-17
907	2017-07-18	53.83	53.84	53.02	53.15	4101431	AAL	2017-07-18
908	2017-07-19	52.26	53.19	51.78	52.61	5774713	AAL	2017-07-19
909	2017-07-20	52.72	52.78	52.10	52.34	4836234	AAL	2017-07-20
910	2017-07-21	52.13	52.55	51.45	51.91	4544423	AAL	2017-07-21
911	2017-07-24	51.79	52.03	51.24	51.28	4876393	AAL	2017-07-24
912	2017-07-25	51.50	51.90	50.54	50.61	4484890	AAL	2017-07-25
913	2017-07-26	50.63	51.16	50.01	51.01	4776112	AAL	2017-07-26
914	2017-07-27	50.34	50.34	48.75	50.00	10260337	AAL	2017-07-27
915	2017-07-28	49.02	50.67	48.73	50.49	9153445	AAL	2017-07-28
916	2017-07-31	50.85	51.23	50.04	50.44	6062854	AAL	2017-07-31
917	2017-08-01	51.13	52.00	50.32	51.06	5045298	AAL	2017-08-01
918	2017-08-02	50.89	51.18	49.90	50.45	4679746	AAL	2017-08-02
919	2017-08-03	50.56	51.20	50.36	50.55	3231366	AAL	2017-08-03
920	2017-08-04	50.67	50.92	50.39	50.80	2993515	AAL	2017-08-04
921	2017-08-07	50.82	51.13	50.46	50.58	3016726	AAL	2017-08-07
922	2017-08-08	50.68	50.78	49.89	50.00	4274416	AAL	2017-08-08
923	2017-08-09	49.74	49.92	49.23	49.40	5020809	AAL	2017-08-09
924	2017-08-10	49.03	49.34	48.19	48.55	5441375	AAL	2017-08-10
925	2017-08-11	48.50	48.78	47.44	48.35	5610688	AAL	2017-08-11

926 rows × 8 columns

In [8]:
```
dataset = dataset.drop('Name', 1)
dataset = dataset.drop('Date', 1)
dataset.set_index(['trading_date'], inplace=True)
```

In [9]:
```
dataset
```

Out[9]:

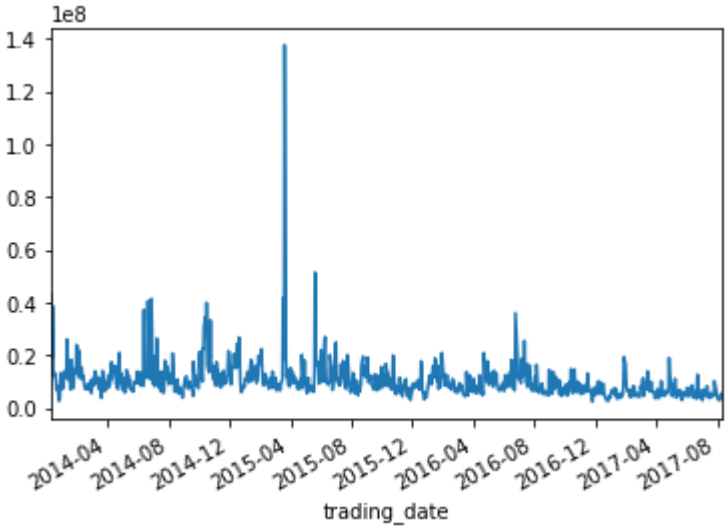
	Open	High	Low	Close	Volume
trading_date					
2013-12-09	23.85	25.44	23.45	24.60	43197268
2013-12-10	24.50	25.17	24.41	24.88	18660625
2013-12-11	25.48	27.20	25.37	25.99	38843371
2013-12-12	26.20	26.71	25.45	25.45	19981824
2013-12-13	25.75	26.30	25.52	26.23	12192421
2013-12-16	26.63	26.77	26.35	26.61	13190945
2013-12-17	26.48	26.59	25.95	26.10	11413199
2013-12-18	25.99	26.23	25.55	26.23	9994162
2013-12-19	26.12	26.49	25.82	26.12	6916497
2013-12-20	26.18	26.49	26.14	26.33	8530924
2013-12-23	26.29	26.49	26.05	26.18	5403084
2013-12-24	26.00	26.26	26.00	26.25	2652974
2013-12-26	26.12	26.36	25.98	26.13	4226639
2013-12-27	25.95	26.10	24.91	24.94	13227018
2013-12-30	24.87	25.25	24.65	24.78	8841369
2013-12-31	24.74	25.25	24.63	25.25	7168395
2014-01-02	25.07	25.82	25.06	25.36	8998943
2014-01-03	25.75	26.75	25.51	26.54	13836062
2014-01-06	26.62	27.20	26.60	27.03	11272273
2014-01-07	27.20	27.40	26.67	26.90	11288775
2014-01-08	26.37	27.68	26.35	27.63	15736891
2014-01-09	28.24	29.60	28.20	29.42	26056445
2014-01-10	29.05	29.83	28.75	29.35	12824160
2014-01-13	29.18	29.53	28.58	28.65	10591701
2014-01-14	28.75	29.04	28.71	28.87	10601529
2014-01-15	28.90	29.44	28.70	28.84	11192558
2014-01-16	28.94	29.39	28.70	29.34	7034698
2014-01-17	29.30	30.02	29.17	30.02	18304949
2014-01-21	30.66	30.80	30.20	30.66	10612821
2014-01-22	30.71	31.24	30.65	31.20	7583631
...	...	...	...	...	...

	Open	High	Low	Close	Volume
trading_date					
2017-06-30	49.92	50.51	49.60	50.32	6618701
2017-07-03	50.78	51.28	50.37	50.39	2906524
2017-07-05	50.44	51.54	50.18	51.26	5157516
2017-07-06	50.97	52.58	50.86	52.05	7021422
2017-07-07	52.30	53.38	52.18	53.03	6596952
2017-07-10	52.98	53.16	52.23	52.66	4600014
2017-07-11	52.58	52.66	51.47	51.61	4545699
2017-07-12	53.12	53.82	52.45	53.80	8348363
2017-07-13	53.40	54.48	53.15	53.81	5346686
2017-07-14	53.80	54.28	53.34	54.22	4537895
2017-07-17	54.21	54.28	53.85	53.87	3727804
2017-07-18	53.83	53.84	53.02	53.15	4101431
2017-07-19	52.26	53.19	51.78	52.61	5774713
2017-07-20	52.72	52.78	52.10	52.34	4836234
2017-07-21	52.13	52.55	51.45	51.91	4544423
2017-07-24	51.79	52.03	51.24	51.28	4876393
2017-07-25	51.50	51.90	50.54	50.61	4484890
2017-07-26	50.63	51.16	50.01	51.01	4776112
2017-07-27	50.34	50.34	48.75	50.00	10260337
2017-07-28	49.02	50.67	48.73	50.49	9153445
2017-07-31	50.85	51.23	50.04	50.44	6062854
2017-08-01	51.13	52.00	50.32	51.06	5045298
2017-08-02	50.89	51.18	49.90	50.45	4679746
2017-08-03	50.56	51.20	50.36	50.55	3231366
2017-08-04	50.67	50.92	50.39	50.80	2993515
2017-08-07	50.82	51.13	50.46	50.58	3016726
2017-08-08	50.68	50.78	49.89	50.00	4274416
2017-08-09	49.74	49.92	49.23	49.40	5020809
2017-08-10	49.03	49.34	48.19	48.55	5441375
2017-08-11	48.50	48.78	47.44	48.35	5610688

926 rows × 5 columns

This code is for comparing the different columns of the raw data.

```
In [10]: dataset[['Open', 'High', 'Low', 'Close']].plot()
         plt.show()
         dataset['Volume'].plot()
         plt.show()
```

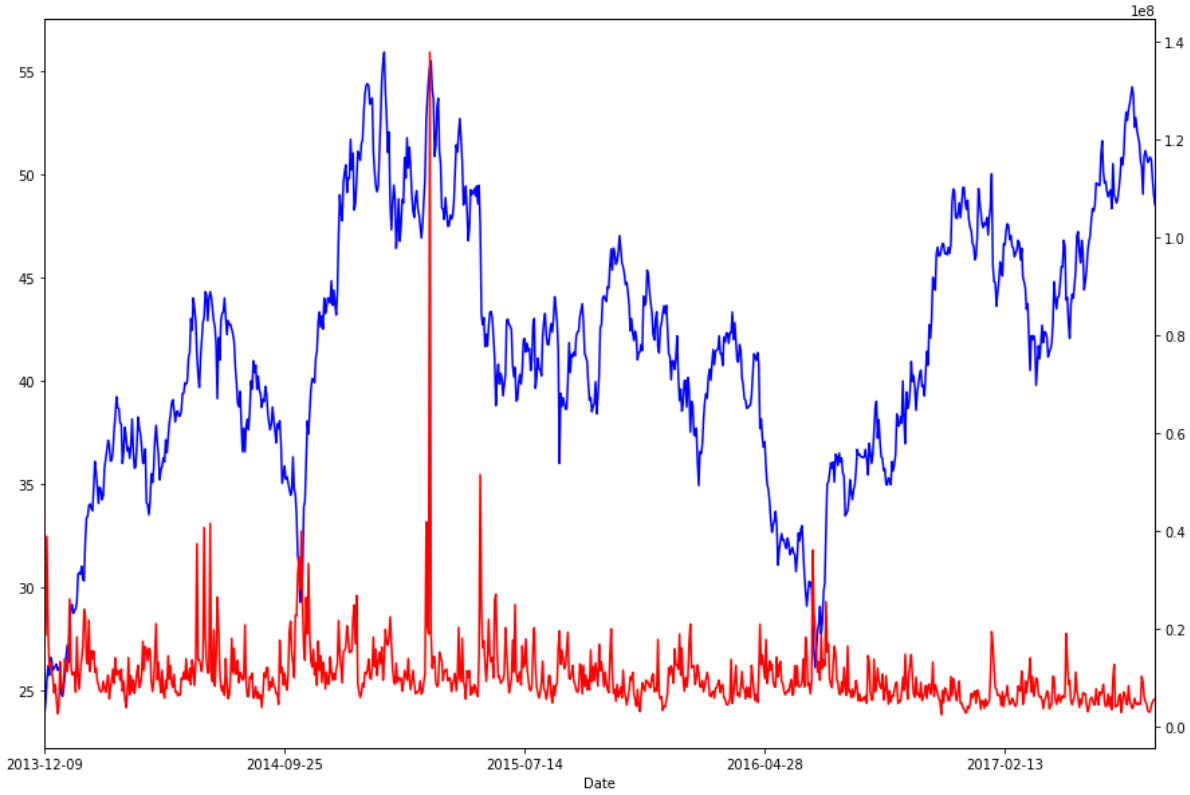


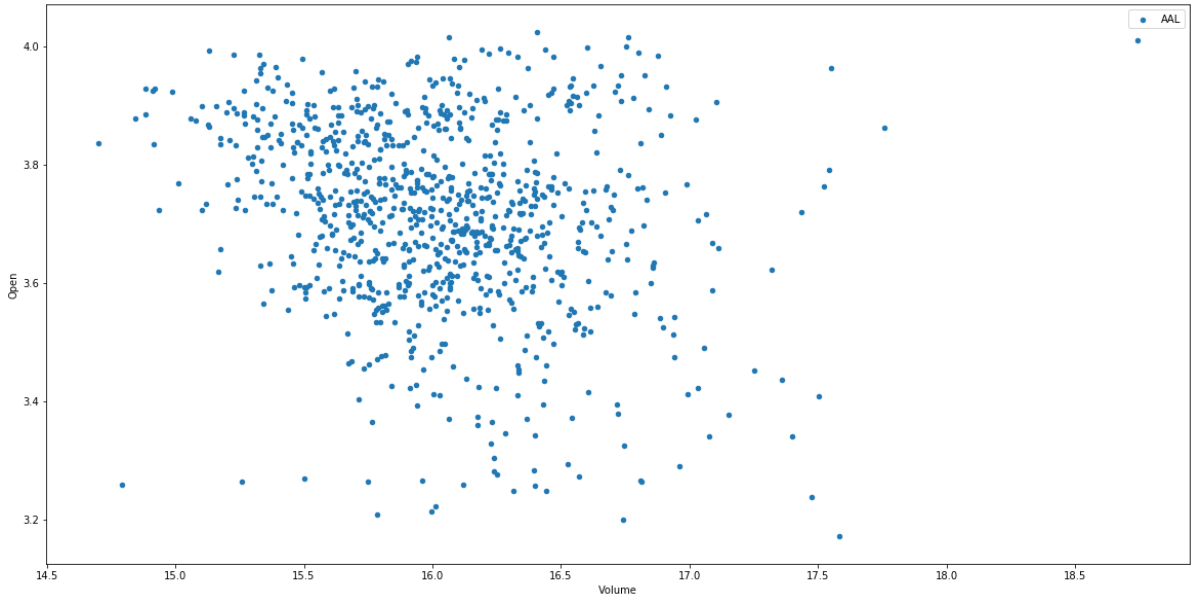
The following is graphing a few of the graphs with the opening price and the volume on one graph to compare with two different axis'. I thought to do this as a comparison between the opening price (which all the raw data features follow roughly the same line) and the volume feature. Since the volume feature is important. [http://www.investopedia.com/terms/v/volume.asp (http://www.investopedia.com/terms/v/volume.asp)]

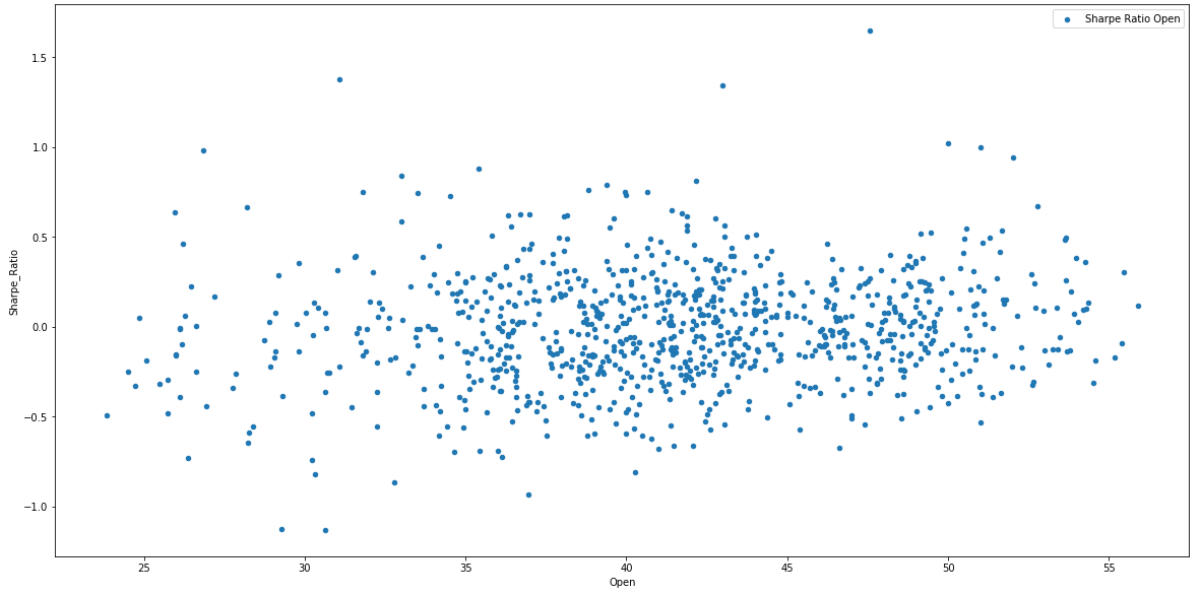
In [11]:
```
def getting_preprocessed_data(symbol):
        csv_file = '{}/{}_data.csv'.format(directory, symbol)
        df = pd.read_csv(csv_file)
        df = df.drop('Name', 1)
        df.set_index('Date', inplace=True)
        # below was found at https://stackoverflow.com/questions/29314033/pyth
on-pandas-dataframe-remove-empty-cells
        df['Open'].replace('', np.nan, inplace=True)
        df.dropna(subset=['Open'], inplace=True)
        return df

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))
        # add legend --> take advantage of pandas providing us access
        # to the line associated with the right part of the axis
        #ax.legend([ax.get_lines()[0], ax.get_lines()[0]], ['Open','Volume'],
 bbox_to_anchor=(1.5, 0.5))
        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.plot.scatter(x='Open', y='Volume', label="AAL")
        log_df = np.log(df)
        log_df.plot.scatter(x='Volume', y='Open', label="AAL", figsize=(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 Close", figsize=(20,10), use_index=True)
        plt.show()
```

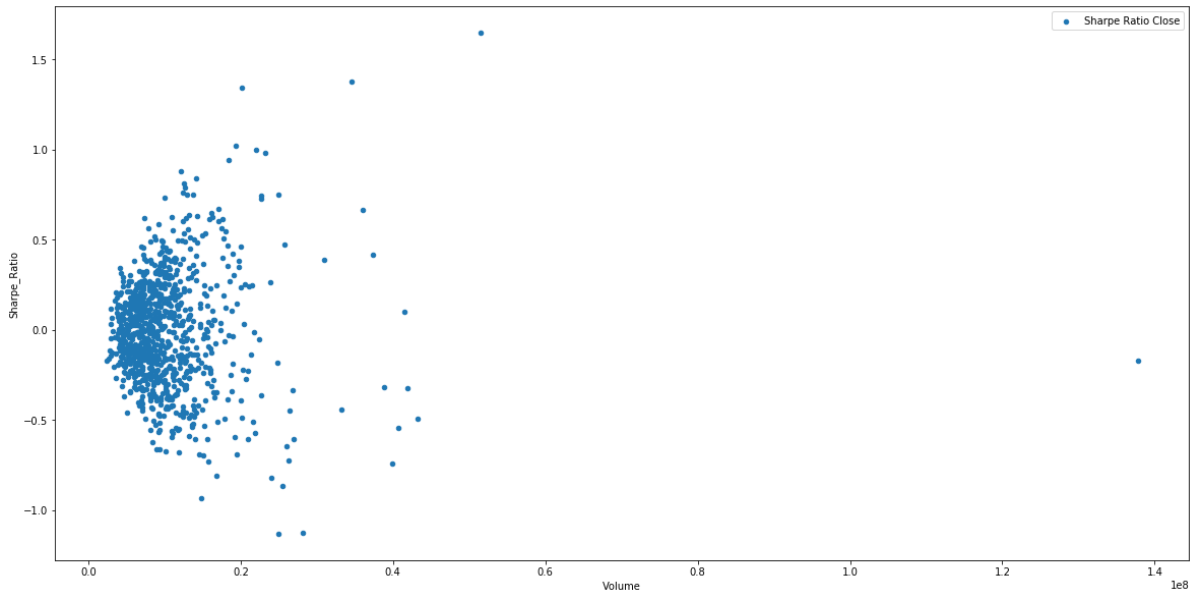
In [12]:
```
# printing out the first four stocks to get an idea of how each stock is indiv
idually represented.
plotting_stocks(symbols_list, 1)
```

AAL


1e8
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
55
50
45
40
35
30
25
2013-12-09
2014-09-25
2015-07-14
2016-04-28
2017-02-13
Date


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





df.plot.scatter(x='Open', y='Volume', label="AAL")

log_df = np.log(df) log_df.plot.scatter(x='Volume', y='Open', label="AAL", figsize=(20,10)) plt.show()

log_df.plot.scatter(x='Volume', y='Close', label="AAL", figsize=(20,10)) plt.show()

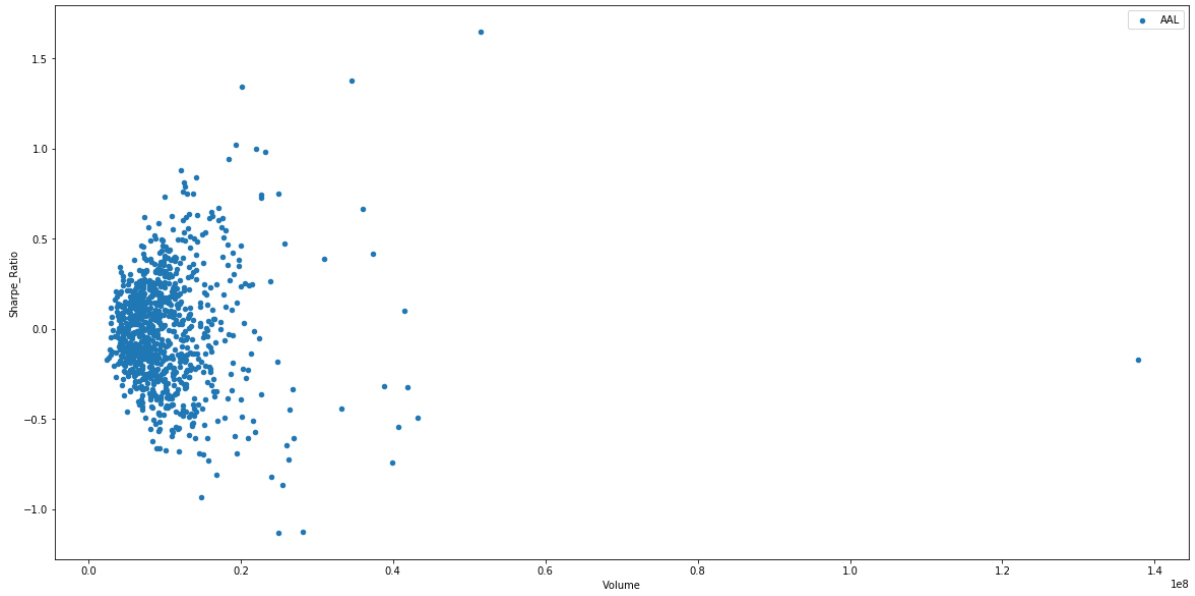
log_df.plot(x=log_df.index, y='Open', label="AAL", figsize=(20,10), use_index=True, style='.') plt.show()

In [13]:
```
#below is the Daily Returns calculation to put into the Sharpe Ratio.
dataset_preprocessed = dataset.assign(Daily_Returns = np.divide((dataset.Open
- dataset.Close), dataset.Close) * 100)
```

In [14]:
```
#Below is the calculation for the Sharpe Ratio column.
dataset_preprocessed = dataset_preprocessed.assign(Sharpe_Ratio = np.divide((d
ataset_preprocessed.Daily_Returns - 0.046), np.std(np.array([dataset_preproces
sed.Open, dataset_preprocessed.High, dataset_preprocessed.Low, dataset_preproc
essed.Close]), ddof=1)))
```

In [15]:
```
#Below is the rate of change for the specific stock.
dataset_preprocessed = dataset_preprocessed.assign(Rate_of_Change = (np.divide
(dataset_preprocessed.Close, dataset_preprocessed.Open) - 1) * 100)
```

In [16]:
```
dataset_preprocessed.plot.scatter(x='Volume', y='Sharpe_Ratio', label="AAL", f
igsize=(20,10))
plt.show()
```



In [17]:
```
from IPython.display import display
display(dataset_preprocessed.head(n=1))
```

	Open	High	Low	Close	Volume	Daily_Returns	Sharpe_Ratio	Rate_o
trading_date								
2013-12-09	23.85	25.44	23.45	24.6	43197268	-3.04878	-0.490892	3.14465

In [18]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

In [19]:
```
# I am using some of the techniques I learned from previous projects.  The bel
ow is from the Finding Donors Project.
# Label encoder example from https://stackoverflow.com/questions/41925157/logi
sticregression-unknown-label-type-continuous-using-sklearn-in-python
lab_enc = LabelEncoder()
closing = lab_enc.fit_transform(dataset_preprocessed['Close'])
features = dataset_preprocessed.drop('Close', axis = 1)
#features = lab_enc.fit_transform(features)

#closing_raw
#features_raw
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(features, closing, test_si
ze=0.2, 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 [21]:
```
clf = SVC()
learner = clf.fit(X_train, y_train)
```

In [22]:
```
pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)*100

print("Accuracy is: {:.4f}%".format(accuracy))
```

```
Accuracy is: 0.0000%
```

This next section will be the creation of the RNN-LSTM model.

Some information gathered from https://machinelearningmastery.com (https://machinelearningmastery.com)

From my research and many hours of trial and error I have discovered that the preprocessing needs to be different for the RNN-LSTM as compared to the SVC. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/ (https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/)

In [23]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

Data preprocessing for the RNN-LSTM model

In [26]:
```
#from machinelearningmastery.com
dataset = dataset_preprocessed.values
dataset = dataset.astype('float32')

print(dataset_preprocessed.shape)
```

```
(926, 8)
```

In [27]:
```
training_set = int(len(dataset) * 0.80)
testing_set = len(dataset) - training_set
```

In [28]:
```
X_train_rnn = dataset[0:training_set]
X_test_rnn = dataset[training_set:len(dataset)]

print(X_train_rnn.shape)
print(X_test_rnn.shape)

#X_train = np.reshape(X_train_rnn, (X_train_rnn.shape[0], 1, X_train_rnn.shape
[1]))
#X_train = np.reshape(X_train_rnn, (X_train_rnn.shape[0], 64))
#X_test = np.reshape(X_test_rnn, (X_test_rnn.shape[0], 1, X_test_rnn.shape
[1]))
```

```
(740, 8)
(186, 8)
```

In [29]:
```
rnn_model = Sequential()
rnn_model.add(LSTM(64, batch_input_shape=(len(X_train), 1, 8)))
rnn_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (740, 64)                 18688
=================================================================
Total params: 18,688
Trainable params: 18,688
Non-trainable params: 0
_________________________________________________________________
```

In [30]:
```
rnn_model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metric
s=['accuracy'])
```

In [31]:
```
print(X_train.shape)
print(rnn_model.input)
```

```
(740, 7)
Tensor("lstm_1_input:0", shape=(740, 1, 8), dtype=float32)
```

In [32]:
```
rnn_model.fit(X_train, y_train, epochs=100)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-32-a1a477a494eb> in <module>()
----> 1 rnn_model.fit(X_train, y_train, epochs=100)

/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)
   1376                                     self._feed_input_shapes,
   1377                                     check_batch_axis=False,
-> 1378                                     exception_prefix='input')
   1379         y = _standardize_input_data(y, self._feed_output_names,
   1380                                     output_shapes,

/usr/local/lib/python3.5/dist-packages/keras/engine/training.py in _standardi
ze_input_data(data, names, shapes, check_batch_axis, exception_prefix)
    130                                 ' to have ' + str(len(shapes[i])) +
    131                                 ' dimensions, but got array with sha
pe ' +
--> 132                                 str(array.shape))
    133             for j, (dim, ref_dim) in enumerate(zip(array.shape, shape
s[i])):
    134                 if not j and not check_batch_axis:

ValueError: Error when checking input: expected lstm_1_input to have 3 dimens
ions, but got array with shape (740, 7)
```

Resource used: https://www.kaggle.com/kwrjcr/predict-stock-prices-with-lstm/editnb (https://www.kaggle.com/kwrjcr/predict-stock-prices-with-lstm/editnb)

One of my major issues was I was treating the RNN like the rest of the techniques and models I have learned. I didn't realize that I needed to think of it as a separate model, with its own needs and quarks. Special thanks to Jason Brownlee of MachineLearningMastery.com for creating a great resource.