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'
#directory = '/OneDrive/Documents/Projects/MachineLearning/Udacity/Capstone/sa
ndp500/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]:
```
# below code is copied from: https://machinelearningmastery.com/multivariate-t
ime-series-forecasting-lstms-keras/

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
	n_vars = 1 if type(data) is list else data.shape[1]
	df = pd.DataFrame(data)
	cols, names = list(), list()
	# input sequence (t-n, ... t-1)
	for i in range(n_in, 0, -1):
		cols.append(df.shift(i))
		names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
	# forecast sequence (t, t+1, ... t+n)
	for i in range(0, n_out):
		cols.append(df.shift(-i))
		if i == 0:
			names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
		else:
			names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_
vars)]
	# put it all together
	agg = pd.concat(cols, axis=1)
	agg.columns = names
	# drop rows with NaN values
	if dropnan:
		agg.dropna(inplace=True)
	return agg
```

In [8]:
```
dataset
```

Out[8]:

	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 [1]:
```
# load dataset
#read_csv('pollution.csv', header=0, index_col=0)
dataset = dataset.drop('Name', 1)
dataset = dataset.drop('Date', 1)
dataset.set_index(['trading_date'], inplace=True)
values = dataset.values
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-96b402d84e02> in <module>()
      1 # load dataset
      2 #read_csv('pollution.csv', header=0, index_col=0)
----> 3 dataset = dataset.drop('Name', 1)
      4 dataset = dataset.drop('Date', 1)
      5 dataset.set_index(['trading_date'], inplace=True)

NameError: name 'dataset' is not defined
```

In [10]:
```
# integer encode direction
encoder = LabelEncoder()
values[:,4] = encoder.fit_transform(values[:,4])
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
reframed.drop(reframed.columns[[5, 6, 7, 9]], axis=1, inplace=True)
print(reframed.head())
```

```
   var1(t-1)  var2(t-1)  var3(t-1)  var4(t-1)  var5(t-1)   var4(t)
1   0.000000   0.008701   0.000000   0.000000   0.997838  0.008986
2   0.020287   0.000000   0.030563   0.008986   0.934054  0.044608
3   0.050874   0.065421   0.061127   0.044608   0.992432  0.027279
4   0.073346   0.049629   0.063674   0.027279   0.948108  0.052311
5   0.059301   0.036416   0.065903   0.052311   0.753514  0.064506
```

In [11]:
```
values = reframed.values
train_len = int(len(values) * 0.80)
test_len = len(values) - train_len
```

In [14]:
```
print(train_len)
print(test_len)
```

```
740
185
```

In [12]:
```
train = values[0:train_len]
test = values[train_len:]
```

In [16]:
```
print("Train")
print(train)
print("Test")
print(test)
```

```
Train
[[ 0.          0.00870132  0.          0.          0.99783784  0.00898588]
 [ 0.0202871   0.          0.03056347  0.00898588  0.93405408  0.04460847]
 [ 0.05087388  0.06542063  0.06112701  0.04460847  0.99243248  0.0272786 ]
 ...,
 [ 0.58863914  0.5871737   0.59535176  0.56803596  0.47027028  0.58825421]
 [ 0.56866407  0.57299387  0.58038837  0.58825421  0.24108109  0.60333776]
 [ 0.59800243  0.59426367  0.6217764   0.60333776  0.37513515  0.64698339]]
Test
[[ 0.66011226  0.64808249  0.64119703  0.64698339  0.80216217  0.63703477]
 [ 0.64950061  0.63261354  0.65202159  0.63703477  0.19351351  0.68132234]
 [ 0.64107358  0.66645181  0.65584201  0.68132234  0.45297298  0.695122  ]
 ...,
 [ 0.83739066  0.82533026  0.84176999  0.81514776  0.04108108  0.79589236]
 [ 0.80805242  0.79761517  0.82075769  0.79589236  0.08972973  0.7686137 ]
 [ 0.78589249  0.77892363  0.78764719  0.7686137   0.12        0.76219511]]
```

In [22]:
```
X_train = train[:, :-1]
y_train = train[:, -1]
X_test = test[:, :-1]
y_test = test[:, -1]

# reshape
X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(740, 1, 5) (740,) (185, 1, 5) (185,)
```

In [35]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from numpy import concatenate
from sklearn.metrics import mean_squared_error
```

In [25]:
```
rnn_model = Sequential()
rnn_model.add(LSTM(50, input_shape=(X_train.shape[1], X_train.shape[2])))
rnn_model.add(Dense(1))
rnn_model.compile(loss='mae', optimizer='adam')
```

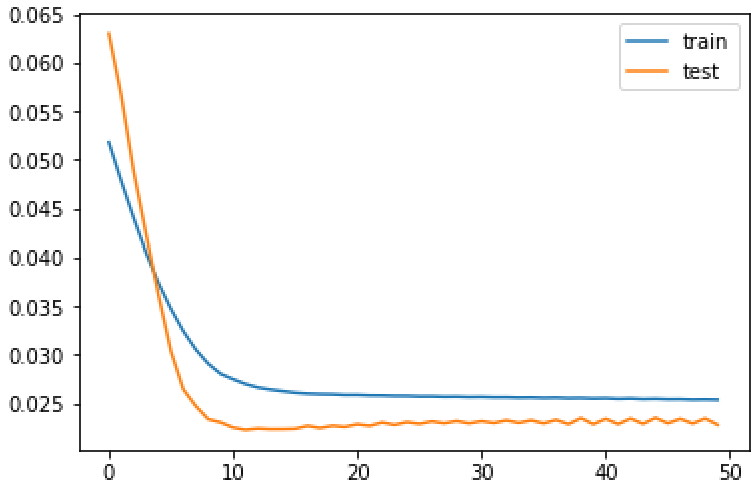
In [27]:

```
history = rnn_model.fit(X_train, y_train, epochs=50, batch_size=72, validation
_data=(X_test, y_test), verbose=2, shuffle=False)
```

```
Train on 740 samples, validate on 185 samples
Epoch 1/50
0s - loss: 0.0518 - val_loss: 0.0630
Epoch 2/50
0s - loss: 0.0478 - val_loss: 0.0566
Epoch 3/50
0s - loss: 0.0440 - val_loss: 0.0488
Epoch 4/50
0s - loss: 0.0405 - val_loss: 0.0424
Epoch 5/50
0s - loss: 0.0374 - val_loss: 0.0362
Epoch 6/50
0s - loss: 0.0347 - val_loss: 0.0304
Epoch 7/50
0s - loss: 0.0324 - val_loss: 0.0264
Epoch 8/50
0s - loss: 0.0306 - val_loss: 0.0247
Epoch 9/50
0s - loss: 0.0291 - val_loss: 0.0234
Epoch 10/50
0s - loss: 0.0281 - val_loss: 0.0231
Epoch 11/50
0s - loss: 0.0275 - val_loss: 0.0225
Epoch 12/50
0s - loss: 0.0270 - val_loss: 0.0223
Epoch 13/50
0s - loss: 0.0266 - val_loss: 0.0224
Epoch 14/50
0s - loss: 0.0264 - val_loss: 0.0224
Epoch 15/50
0s - loss: 0.0263 - val_loss: 0.0224
Epoch 16/50
0s - loss: 0.0261 - val_loss: 0.0224
Epoch 17/50
0s - loss: 0.0260 - val_loss: 0.0227
Epoch 18/50
0s - loss: 0.0260 - val_loss: 0.0225
Epoch 19/50
0s - loss: 0.0259 - val_loss: 0.0227
Epoch 20/50
0s - loss: 0.0259 - val_loss: 0.0226
Epoch 21/50
0s - loss: 0.0259 - val_loss: 0.0229
Epoch 22/50
0s - loss: 0.0258 - val_loss: 0.0227
Epoch 23/50
0s - loss: 0.0258 - val_loss: 0.0231
Epoch 24/50
0s - loss: 0.0258 - val_loss: 0.0228
Epoch 25/50
0s - loss: 0.0258 - val_loss: 0.0231
Epoch 26/50
0s - loss: 0.0257 - val_loss: 0.0229
Epoch 27/50
0s - loss: 0.0257 - val_loss: 0.0232
Epoch 28/50
0s - loss: 0.0257 - val_loss: 0.0230
```

```
Epoch 29/50
0s - loss: 0.0257 - val_loss: 0.0232
Epoch 30/50
0s - loss: 0.0257 - val_loss: 0.0230
Epoch 31/50
0s - loss: 0.0257 - val_loss: 0.0232
Epoch 32/50
0s - loss: 0.0256 - val_loss: 0.0230
Epoch 33/50
0s - loss: 0.0256 - val_loss: 0.0233
Epoch 34/50
0s - loss: 0.0256 - val_loss: 0.0230
Epoch 35/50
0s - loss: 0.0256 - val_loss: 0.0233
Epoch 36/50
0s - loss: 0.0256 - val_loss: 0.0230
Epoch 37/50
0s - loss: 0.0256 - val_loss: 0.0233
Epoch 38/50
0s - loss: 0.0256 - val_loss: 0.0229
Epoch 39/50
0s - loss: 0.0256 - val_loss: 0.0235
Epoch 40/50
0s - loss: 0.0255 - val_loss: 0.0229
Epoch 41/50
0s - loss: 0.0255 - val_loss: 0.0234
Epoch 42/50
0s - loss: 0.0255 - val_loss: 0.0229
Epoch 43/50
0s - loss: 0.0255 - val_loss: 0.0235
Epoch 44/50
0s - loss: 0.0255 - val_loss: 0.0229
Epoch 45/50
0s - loss: 0.0255 - val_loss: 0.0235
Epoch 46/50
0s - loss: 0.0254 - val_loss: 0.0230
Epoch 47/50
0s - loss: 0.0254 - val_loss: 0.0234
Epoch 48/50
0s - loss: 0.0254 - val_loss: 0.0229
Epoch 49/50
0s - loss: 0.0254 - val_loss: 0.0235
Epoch 50/50
0s - loss: 0.0254 - val_loss: 0.0228
```

In [30]:
```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



In [32]:
```
yhat = rnn_model.predict(X_test)
X_test = X_test.reshape((X_test.shape[0], X_test.shape[2]))
inv_yhat = concatenate((yhat, X_test[:, 1:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]
```

In [34]:
```
y_test = y_test.reshape((len(y_test), 1))
inv_y = concatenate((y_test, X_test[:,1:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]
```

In [ ]:
```
rmse = sqrt(mean)
```

In [ ]:
```
 l;asjfsdlk f
```

++

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

In [ ]:
```
df[['Open', 'High', 'Low', 'Close']].plot()
plt.show()
df['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 [ ]:
```
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 [ ]:
```
# printing out the first four stocks to get an idea of how each stock is indiv
idually represented.
plotting_stocks(symbols_list, 10)
```

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

In [ ]:
```
#Below is the calculation for the Sharpe Ratio column.
df_preprocessed = df_preprocessed.assign(Sharpe_Ratio = np.divide((df_preproce
ssed.Daily_Returns - 0.046), np.std(np.array([df_preprocessed.Open, df_preproc
essed.High, df_preprocessed.Low, df_preprocessed.Close]), ddof=1)))
```

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

In [ ]:
```
df_preprocessed.plot.scatter(x='Volume', y='Sharpe_Ratio', label="AAL", figsiz
e=(20,10))
plt.show()
```

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

In [ ]:
```
# I am using some of the techniques I learned from previous projects.  The bel
ow is from the Finding Donors Project.
closing = df_preprocessed['Close'].astype(int)
features = df_preprocessed.drop('Close', axis = 1)

#closing_raw
#features_raw
```

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

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]))
```

In [ ]:

```
clf = SVC(random_state=2)

learner = clf.fit(X_train, y_train)
```

In [ ]:

```
pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)*100

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

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 [ ]:

```
features_rnn = np.array(features)
closing_rnn = np.array(closing)

features_train_len = int(len(features_rnn) * 0.80)
features_test_len = int(len(features_rnn) - features_train_len)

X_train = features_rnn[0:features_train_len]
X_test = features_rnn[features_test_len:len(features_rnn)]

y_train = closing_rnn[0:features_train_len]
y_test = closing_rnn[features_test_len:len(closing_rnn)]
```

In [ ]:

```
X_train = X_train.reshape(1, features_train_len, 7)
y_train = y_train.reshape(1, features_train_len, 1)
```

In [ ]:

```
    kaldkfjasdfklj
```

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

In [ ]:
```
rnn_model = Sequential()
rnn_model.add(LSTM(32, input_shape=(740, 7)))
rnn_model.add(Dense(1))
```

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

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

In [ ]:
```
```