In [1]:
```
import datetime
import pandas as pd
import pandas_datareader.data as web
```

In [2]:
```
start = datetime.datetime(1999, 1, 22) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('NVDA', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                Open      High       Low     Close  Adj Close    Volume
Date
1999-01-22  1.750000  1.953125  1.552083  1.640625   1.521034  67867200
1999-01-25  1.770833  1.833333  1.640625  1.812500   1.680380  12762000
1999-01-26  1.833333  1.869792  1.645833  1.671875   1.550005   8580000
1999-01-27  1.677083  1.718750  1.583333  1.666667   1.545177   6109200
1999-01-28  1.666667  1.677083  1.651042  1.661458   1.540348   5688000
```

In [3]:
```
import time
import math
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers.recurrent import LSTM
import numpy as np
import pandas as pd
import tensorflow as tf
import sklearn.preprocessing as prep
from keras import backend
```

```
Using TensorFlow backend.
```

In [4]:
```
#import os
#s=os.getcwd()
#s
```

In [5]:
```
#df = pd.read_csv('/Users/Yuffie/USA/SCU/COEN281DataMining/TermProject/data/GOOG.csv')
#df.head()
```

In [6]:
```
# Data preparation
col_list = df.columns.tolist()
col_list
```

Out[6]: ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']

In [7]:
```
col_list.remove('Close')
col_list.append('Close')
#col_list.remove('Date')
col_list.remove('Volume')
col_list
```

Out[7]: ['Open', 'High', 'Low', 'Adj Close', 'Close']

In [8]:
```
df = df[col_list]
df.head()
```

Out[8]:

	Open	High	Low	Adj Close	Close
Date					
1999-01-22	1.750000	1.953125	1.552083	1.521034	1.640625
1999-01-25	1.770833	1.833333	1.640625	1.680380	1.812500
1999-01-26	1.833333	1.869792	1.645833	1.550005	1.671875
1999-01-27	1.677083	1.718750	1.583333	1.545177	1.666667
1999-01-28	1.666667	1.677083	1.651042	1.540348	1.661458

In [9]:
```
# Save data
df.to_csv('NVDA-adjust.csv', index=False)
validate_df = pd.read_csv('NVDA-adjust.csv')
validate_df.head()
```

Out[9]:

	Open	High	Low	Adj Close	Close
0	1.750000	1.953125	1.552083	1.521034	1.640625
1	1.770833	1.833333	1.640625	1.680380	1.812500
2	1.833333	1.869792	1.645833	1.550005	1.671875
3	1.677083	1.718750	1.583333	1.545177	1.666667
4	1.666667	1.677083	1.651042	1.540348	1.661458

In [10]:
```
# Standardization the dataset
def standard_scaler(X_train, X_test):
    train_samples, train_nx, train_ny = X_train.shape
    test_samples, test_nx, test_ny = X_test.shape

    X_train = X_train.reshape((train_samples, train_nx * train_ny))
    X_test = X_test.reshape((test_samples, test_nx * test_ny))

    preprocessor = prep.StandardScaler().fit(X_train)
    X_train = preprocessor.transform(X_train)
    X_test = preprocessor.transform(X_test)

    X_train = X_train.reshape((train_samples, train_nx, train_ny))
    X_test = X_test.reshape((test_samples, test_nx, test_ny))

    return X_train, X_test
```

In [11]:
```
# Split the data to X_train, y_train, X_test, y_test
def preprocess_data(stock, seq_len):
    amount_of_features = len(stock.columns)
    data = stock.as_matrix()

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index : index + sequence_length])

    result = np.array(result)
    row = round(0.95 * result.shape[0])
    train = result[: int(row), :]
    y_test_org = result[int(row) :, -1][ : ,-1]

    train, result = standard_scaler(train, result)

    X_train = train[:, : -1]
    y_train = train[:, -1][: ,-1]
    X_test = result[int(row) :, : -1]
    y_test = result[int(row) :, -1][ : ,-1]

    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], amount_of_features))
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], amount_of_features))

    return [X_train, y_train, X_test, y_test, y_test_org]
```

In [12]:
```
# Build LSTM Neural Network
# LSTM --> Dropout --> LSTM --> Dropout --> Fully-Conneted(Dense)
def build_model(layers):
    model = Sequential()

    # By setting return_sequences to True we are able to stack another LSTM layer
    model.add(LSTM(
        return_sequences=True,
        input_shape=(None, 5), units=20))
    model.add(Dropout(0.4))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.3))

    model.add(Dense(
        units=1))
    model.add(Activation("linear"))

    start = time.time()
    model.compile(loss="mse", optimizer="rmsprop", metrics=['accuracy'])
    print("Compilation Time : ", time.time() - start)
    return model
```

In [13]:
```
window = 20
X_train, y_train, X_test, y_test, y_test_org = preprocess_data(df[:: -1], window)
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print("X_test", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train (4492, 20, 5)
y_train (4492,)
X_test (236, 20, 5)
y_test (236,)
```

In [14]:
```
model = build_model([X_train.shape[2], window, 100, 1])
```

```
Compilation Time :  0.056817054748535156
```

In [15]:
```
# Training the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=300,
    validation_split=0.1,
    verbose=0)
```

Out[15]:
```
<keras.callbacks.History at 0x11cc537b8>
```

In [16]:
```
trainScore = model.evaluate(X_train, y_train, verbose=0)
print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore[0], math.sqrt(trainScore[0])))
print(model.metrics_names)
testScore = model.evaluate(X_test, y_test, verbose=0)
print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore[0], math.sqrt(testScore[0])))
```

```
Train Score: 0.01 MSE (0.11 RMSE)
['loss', 'acc']
Test Score: 0.01 MSE (0.08 RMSE)
```

In [17]:
```
#Visualize the Prediction
diff = []
ratio = []
pred = model.predict(X_test)
for u in range(len(y_test)):
    pr = pred[u][0]
    ratio.append((y_test[u] / pr) - 1)
    diff.append(abs(y_test[u] - pr))
print('error_ratio', ratio)
#print('error_abs', diff)
#print(pred)
```

```
error_ratio [0.1342657551982076, 0.14280182793684548, 0.14155707060567391, 0.13226028506682619, 0.1463841691
5918362, 0.14826904824697129, 0.15198380230167596, 0.16055808023283591, 0.16175237413459076, 0.1526062643052
9212, 0.1409174508463511, 0.1355999442226572, 0.13488999796526135, 0.133861098323665, 0.12975807316055477,
0.13986157869723614, 0.13374830113607517, 0.13963565595481953, 0.13529156931333541, 0.14806061670808401, 0.1
5346215104857186, 0.13872752048749626, 0.1306557710728673, 0.1300561065286765, 0.12517925851240341, 0.112496
62928796167, 0.11742069758461171, 0.1575607899215119, 0.15620886739895612, 0.15791945919117389, 0.1656151763
1150613, 0.15436377836641224, 0.15499472931013791, 0.15764907403292705, 0.14979355251242743, 0.1427661360920
1738, 0.14316539176240894, 0.13670961235978019, 0.1443809610045923, 0.15411871305470082, 0.156939432120060З
4, 0.15747270421858017, 0.15837702672402054, 0.14761439397256826, 0.13931122385385986, 0.14689072725486008,
0.14033113724236479, 0.14183536997623314, 0.14475855984631014, 0.1460445136233508, 0.14054154087000525, 0.13
4817448165647, 0.13456044784485299, 0.13359739982093966, 0.13212717521390549, 0.13269458276629598, 0.1318502
8785918296, 0.13850654567411591, 0.13251589985778933, 0.14513292515875831, 0.15178810224763684, 0.1498534031
8122664, 0.145255933272469, 0.14230710477816078, 0.13366832274967688, 0.14184791521005424, 0.146214519327748
43, 0.13648575466868418, 0.12878987718951374, 0.13285342675898226, 0.12491969617509069, 0.1248105977277858,
0.12852867255030609, 0.12915623427485068, 0.13291303120577203, 0.13165962855361868, 0.12987499610866204, 0.1
3415294353508367, 0.13415919611081151, 0.13537887210621968, 0.13598975640959421, 0.13782120874864745, 0.1363
3252433101961, 0.13065272661438465, 0.1416137297476574, 0.1404322704668961, 0.14582780159014597, 0.142159039
85889455, 0.13981139991903491, 0.13779289551589358, 0.12986409975467161, 0.13012937308395922, 0.15316344800
127868, 0.15234384843698812, 0.1483932478950214, 0.14078930050578631, 0.13797680922250644, 0.142154777391892
75, 0.14713191103418888, 0.14671533128953973, 0.14437838486231414, 0.14240855126858154, 0.14220498742243071,
0.14098703972168103, 0.13799047056433955, 0.13871272467707785, 0.13908130749190417, 0.1396676950005975, 0.14
06458218655231, 0.14024578810340738, 0.14006491612634053, 0.13973820905583456, 0.13538154009765435, 0.133096
68636519834, 0.13258857043120664, 0.13185537715040452, 0.13582063785107645, 0.13787481377521527, 0.134891357
7087798, 0.13605846644410735, 0.14134947591807046, 0.14800254076224006, 0.14730724586758037, 0.1428371682652
6849, 0.14115689711097756, 0.14532817734006698, 0.13871560298298258, 0.14454656957719103, 0.1438006782100107
5, 0.14337116549683215, 0.14299775607157539, 0.14009829866956491, 0.14064910144145704, 0.13858721450718914,
0.14211045942028133, 0.13875657817740294, 0.13656038744884014, 0.13928325426057198, 0.13936585369257992, 0.1
3638023206211325, 0.13517252010561864, 0.13633723837651734, 0.13853646312046131, 0.14071504734412232, 0.1412
0212638699781, 0.14108558703380325, 0.14227918828381325, 0.13865711658827262, 0.13854854926079896, 0.1392645
3176153131, 0.13866171348179979, 0.1386074546575784, 0.13931470823983805, 0.13905539493811347, 0.13721891480
640536, 0.13094320355178679, 0.12936434913525163, 0.13583054593189781, 0.14082057048600882, 0.13634216635863
661, 0.1368613872219453, 0.13970025697770905, 0.14371663862161266, 0.14262185303917296, 0.14156247891052987,
0.13877158811305867, 0.13953692617091318, 0.13465773095274014, 0.13580750063257008, 0.13788799206397662, 0.1
3838961692983798, 0.13800709699663605, 0.14552144486130847, 0.14018818349960549, 0.13563010204042425, 0.1325
9308430757, 0.13539359224038905, 0.13513032041620954, 0.13272350473975014, 0.13715251275462248, 0.1401411266
4867939, 0.13776178656512528, 0.13454267849962287, 0.13164243870130465, 0.131843895621075 9, 0.13557861877026
989, 0.14368604679307539, 0.14058291940322487, 0.13710270951791914, 0.13434890665013333, 0.1409226712657845,
0.13281276065464231, 0.13694887870945704, 0.14493830683158127, 0.14485357376700914, 0.14228689452146659, 0.1
3534474982705014, 0.13308625391936779, 0.13491455858353185, 0.13729602910556937, 0.13856275934418627, 0.1359
3651287982711, 0.14055626733979776, 0.13870140407704046, 0.1345630434474645, 0.13381297321109975, 0.13806506
561051313, 0.13957788716201058, 0.14403892108548777, 0.14094921487998957, 0.13368951580068744, 0.13368375176
349123, 0.135497374269546, 0.13227723471317465, 0.13138844744430078, 0.14254790807200202, 0.147235902630451
```

```
5, 0.14580442275234606, 0.14628718937484742, 0.14498586531538349, 0.13758059801668265, 0.13703043371274393,
0.14174302183938892, 0.14760125551487313, 0.14448364193065077, 0.13936064446430962, 0.13585711791546418, 0.1
3878885536171759, 0.14328778692211896, 0.14385031328098874, 0.13581252481117212, 0.1375388484132205, 0.13384
774978864722, 0.13481528016836641, 0.13562883131292347, 0.12954394049658768]
```

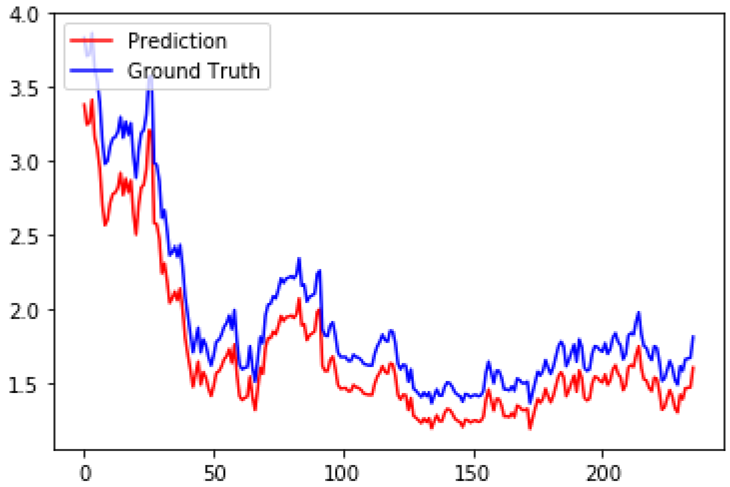
In [18]:
```
# Scale the representation back
y_test = y_test_org
pred_org = []
for u in range(len(y_test)):
    pred_org.append(y_test[u]/(ratio[u] + 1))
    #print(ratio[u])
print(pred_org)
print(y_test)
```

```
[3.3795721879394511, 3.2449484323059146, 3.2576119895844973, 3.413157779151383, 3.162116241241153, 3.0843529
270487253, 2.9478131491216191, 2.7016459179457524, 2.5643734984567881, 2.6027968898886589, 2.716198264564314
3, 2.7747817495332447, 2.7856955349577159, 2.8203780910447485, 2.9182132691265723, 2.7689765660908781, 2.880
3791782776411, 2.7878058951552798, 2.8672880940786563, 2.6403222581502002, 2.5015272476664872, 2.69855338060
54542, 2.8191604213680481, 2.839091777359096, 2.9439753487630473, 3.2069382558789981, 3.1974680688509949, 2.
5781583360320584, 2.5721632862843458, 2.4739052248080875, 2.2430927918025518, 2.3100750820280505, 2.19156844
24914041, 2.0425732227836986, 2.0746480920749617, 2.1193093875552038, 2.059340684177466, 2.1397654894028948,
1.9979867525870389, 1.8051288627717261, 1.6926841160659618, 1.5929101336732874, 1.474764226662789, 1.561209
9407345239, 1.6457311757691482, 1.4895342332124177, 1.575748430710521, 1.5371813189116168, 1.465010229078485
7, 1.413376165362759, 1.4795603137021864, 1.569635717955576, 1.5791728007118082, 1.6126757174008746, 1.66077
45500365287, 1.6829338013999386, 1.7302049758754334, 1.6377446463392711, 1.7613810104127343, 1.5554962754678
039, 1.4108497881067898, 1.3860462521489023, 1.4007061246268617, 1.4088811960182503, 1.5436613733331015, 1.4
048867442253878, 1.3177437334207345, 1.4619259354327405, 1.6057018552583082, 1.563161621946304 7, 1.740864709
4176142, 1.8058595856967385, 1.8045248202670026, 1.849648380448673, 1.8297229733457749, 1.8869781567884416,
1.9544932028813751, 1.9195673849896899, 1.9471102536334219, 1.9496064744394093, 1.9577271603478494, 1.940843
5903815502, 1.9663038346239528, 2.0729176561737943, 1.8910511881082517, 1.8998646882492727, 1.79091744601778
72, 1.824030566056178, 1.8369267057240584, 1.8493409550126754, 1.9821702455067673, 2.0001400316068678, 1.616
9286350792595, 1.5819210580874461, 1.5828271398598148, 1.6572955226366188, 1.6796985531769799, 1.59603324880
5933, 1.4892210595552731, 1.4625103146689729, 1.465496921459049, 1.4680238502571537, 1.4454865966973598, 1.4
470295827398141, 1.48745358048611, 1.4727884949872638, 1.4677398259490644, 1.4532745003350576, 1.42919736236
24898, 1.4251313330460913, 1.4207892700563522, 1.4211965406879221, 1.49545940288404, 1.544401356549136, 1.5
72724682646 0828, 1.6151577638854764, 1.5774207126485713, 1.5654182502644638, 1.6337836986822758, 1.627520109
759216, 1.5492406465405242, 1.4200386690064202, 1.3891222301091848, 1.4218998516355699, 1.419429706906004,
1.3187635036691601, 1.3996031983973944, 1.2787090004914781, 1.2658822709092248, 1.2481362509958749, 1.230317
3759791175, 1.260856192555929, 1.2374164834884955, 1.2625295468667179, 1.1947907391485082, 1.244047259219698
1, 1.283110880956726 9, 1.2434721520764018, 1.243382005357377, 1.2970640973974943, 1.3259729013348145, 1.3200
306646141835, 1.2946076368605948, 1.2601744873506047, 1.245944926953126, 1.2369431496098526, 1.2037337404928
403, 1.2533035443329332, 1.2488488092344858, 1.2343489688261036, 1.2441509038431753, 1.2487842005457541, 1.2
388657758843511, 1.2437208991727478, 1.2732086858109339, 1.3908010544297638, 1.4573091502845879, 1.384816604
5838222, 1.3148430514020051, 1.39335936558064, 1.3927229984202918, 1.3481246411880481, 1.2750824380395109,
1.276304138697409 9, 1.2706417973585873, 1.298914563236492, 1.2706187634176187, 1.3495259039166396, 1.3389883
401483051, 1.3182316805006447, 1.3176508092593127, 1.3272474345601244, 1.1912330459821414, 1.260756795065046
7, 1.3300255050356482, 1.3933733322809521, 1.3669999642479824, 1.3948468924867159, 1.46218383662881, 1.41068
76448033236, 1.3704443805064717, 1.4145096267107575, 1.4782017739675164, 1.5464248601425152, 1.5737594264468
564, 1.5410645912786678, 1.4117362055149065, 1.4521083665417172, 1.5023559311755199, 1.5519184526731902, 1.4
425456180777336, 1.586206531573322, 1.5346266069416432, 1.4010947056520928, 1.3830004432708585, 1.3952274228
512995, 1.4955078624872733, 1.5444543554797487, 1.5373774059059089, 1.5158392853581077, 1.5050044329457979,
1.5589189888003361, 1.4886744728168257, 1.5185429593823621, 1.5929410097021954, 1.6215566794874641, 1.565156
5572346704, 1.5402264468040412, 1.4522740174115532, 1.4881617672865437, 1.6079508318576394, 1.62174151048744
69, 1.609977302832615, 1.6927541605881748, 1.7493257991724684, 1.6046005485176256, 1.5254055386407406, 1.518
2198335570514, 1.4676008033531931, 1.4465244071321266, 1.5383525378782315, 1.5299353020127544, 1.44150913867
50971, 1.3206895624386648, 1.3379396121528708, 1.3988108223180988, 1.4535648665300509, 1.4086606067904155,
```

```
1.3302276271963707, 1.3022534353531987, 1.4215224473496832, 1.3918935623241599, 1.4653272454875015, 1.468668
0987876066, 1.4722019676685307, 1.6046299174542609]
[ 3.833333  3.708333  3.71875   3.864583  3.625     3.541667  3.395833
  3.135417  2.979167  3.        3.098958  3.151042  3.161458  3.197917
  3.296875  3.15625   3.265625  3.177083  3.255208  3.03125   2.885417
  3.072917  3.1875    3.208333  3.3125    3.567708  3.572917  2.984375
  2.973958  2.864583  2.614583  2.666667  2.53125   2.364583  2.385417
  2.421875  2.354167  2.432292  2.286458  2.083333  1.958333  1.84375
  1.708333  1.791667  1.875     1.708333  1.796875  1.755208  1.677083
  1.619792  1.6875    1.78125   1.791667  1.828125  1.880208  1.90625
  1.958333  1.864583  1.994792  1.78125   1.625     1.59375   1.604167
  1.609375  1.75      1.604167  1.510417  1.661458  1.8125    1.770833
  1.958333  2.03125   2.036458  2.088542  2.072917  2.135417  2.208333
  2.177083  2.208333  2.213542  2.223958  2.208333  2.234375  2.34375
  2.15885   2.166667  2.052083  2.083333  2.09375   2.104167  2.239583
  2.260417  1.864583  1.822917  1.817708  1.890625  1.911458  1.822917
  1.708333  1.677083  1.677083  1.677083  1.651042  1.651042  1.692708
  1.677083  1.671875  1.65625   1.630208  1.625     1.619792  1.619792
  1.697917  1.75      1.78125   1.828125  1.791667  1.78125   1.854167
  1.848958  1.768225  1.630208  1.59375   1.625     1.619792  1.510417
  1.59375   1.463542  1.447917  1.427083  1.40625   1.4375    1.411458
  1.4375    1.364583  1.416667  1.458333  1.416667  1.416667  1.473958
  1.505208  1.5       1.473958  1.4375    1.421875  1.411458  1.375
  1.427083  1.421875  1.40625   1.416667  1.421875  1.411458  1.416667
  1.447917  1.572917  1.645833  1.572917  1.5       1.583333  1.583333
  1.536458  1.458333  1.458333  1.450517  1.479167  1.447917  1.53125
  1.520833  1.5       1.5       1.510417  1.364583  1.4375    1.510417
  1.578125  1.552083  1.583333  1.65625   1.604167  1.5625    1.609375
  1.677083  1.75      1.78125   1.75      1.614583  1.65625   1.708333
  1.760417  1.645833  1.796875  1.744792  1.604167  1.583333  1.59375
  1.697917  1.75      1.744792  1.723958  1.713542  1.770833  1.697917
  1.729167  1.807292  1.838542  1.78125   1.755208  1.661458  1.697917
  1.822917  1.838542  1.828125  1.916667  1.979167  1.833333  1.75
  1.739583  1.682292  1.65625   1.75      1.739583  1.645833  1.515625
  1.53125   1.59375   1.651042  1.604167  1.520833  1.489583  1.614583
  1.583333  1.661458  1.666667  1.671875  1.8125  ]
```

In [19]:
```
import matplotlib.pyplot as plt2

plt2.plot(pred_org, color='red', label='Prediction')
plt2.plot(y_test, color='blue', label='Ground Truth')
plt2.legend(loc='upper left')
plt2.show()
```



In [ ]: