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(with_mean=True, with_std=True).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(0.98 * 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]
    #train_temp = train[:, -2][: ,-1]
    #y_train = (train_temp - y_train)/y_train

    X_test = result[int(row) :, : -1]
    y_test = result[int(row) :, -1][ : ,-1]
    #test_temp = result[int(row) :, -2][ : ,-1]
    #y_test = (test_temp - y_test)/y_test

    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(0.2))

    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)
#print("y_train", y_train)
print("X_test", X_test)
#print("y_test", y_test)
```

```
X_train [[[-0.82250329 -0.8171107  -0.82815553 -0.79244724 -0.82700532]
  [-0.81892169 -0.8204891  -0.8211175  -0.78207499 -0.81587712]
  [-0.81326621 -0.81623053 -0.81826333 -0.7861514  -0.82032545]
  ...,
  [-0.79081306 -0.79431327 -0.79122512 -0.76013564 -0.79275261]
  [-0.78855982 -0.79210131 -0.78855977 -0.75698372 -0.78938873]
  [-0.78844823 -0.78808965 -0.78595885 -0.75248735 -0.78457436]]

 [[-0.8214652  -0.82299134 -0.82367276 -0.78451743 -0.81845483]
  [-0.81582338 -0.81870837 -0.82085518 -0.78852893 -0.82283653]
  [-0.82097174 -0.82357042 -0.82139497 -0.78638918 -0.82058188]
  ...,
  [-0.79081306 -0.79431327 -0.79073528 -0.75902074 -0.79154951]
  [-0.79047707 -0.79021162 -0.78807232 -0.75454271 -0.78675462]
  [-0.78367694 -0.78714922 -0.78547365 -0.75204566 -0.78409772]]

 [[-0.8183509  -0.82120155 -0.82340909 -0.7910055  -0.82545069]
  [-0.82356916 -0.82608557 -0.82400322 -0.78876793 -0.82309427]
  [-0.82148541 -0.82559523 -0.81800233 -0.786627   -0.82083836]
  ...,
  [-0.79273978 -0.79241441 -0.79024548 -0.75656794 -0.78890278]
  [-0.78568389 -0.78926673 -0.78758483 -0.75409886 -0.78627567]
  [-0.78272265 -0.78362271 -0.78401811 -0.74851236 -0.78028477]]


 ...,
 [[ 6.70215084  6.68130102  6.69971893  6.66531556  6.63421696]
  [ 6.69828893  6.78405408  6.81574008  6.91028755  6.88005986]
  [ 6.93895262  6.95266834  6.96193656  6.86190569  6.83216249]
  ...,
  [ 6.45033534  6.55531156  6.4943837   6.66169683  6.63838643]
  [ 6.63325516  6.67170723  6.56410245  6.54180086  6.51868109]
  [ 6.52304135  6.58039767  6.63678795  6.7055691   6.68321531]]

 [[ 6.73454004  6.82022708  6.85261706  6.94852943  6.917783  ]
  [ 6.97688902  6.98967937  7.00008936  6.89891946  6.86867681]
  [ 6.89851437  6.84964638  6.90080693  6.97893428  6.94934933]
  ...,
  [ 6.6676737   6.7057032   6.59736559  6.57501751  6.55153445]
  [ 6.55456941  6.61320128  6.6698659   6.74002203  6.71730491]
  [ 6.75389717  6.69279454  6.67125536  6.6736063   6.65118653]]

 [[ 7.01457792  7.02689814  7.03791821  6.93710121  6.90634034]
```

```
  [ 6.93623955  6.88613376  6.9386396   7.01655081  6.98646127]
  [ 6.97692568  6.88463485  6.98598734  6.92484545  6.89518716]
  ...,
  [ 6.58859916  6.64691343  6.7036388   6.77419611  6.75110991]
  [ 6.78648476  6.72613089  6.70449672  6.70790276  6.68512057]
  [ 6.5454859   6.64991242  6.65029573  6.75168746  6.72942892]]]
X_test [[[ 6.97371867  6.92282591  6.97615116  7.0553547   7.0247416 ]
  [ 7.01506044  6.92130007  7.02426609  6.96218339  6.93202289]
  [ 6.92169193  6.83506845  6.77654373  6.70160586  6.67164668]
  ...,
  [ 6.82166042  6.76039088  6.73843653  6.7419217   6.71877138]
  [ 6.57711697  6.68304552  6.68343772  6.78636615  6.76374297]
  [ 6.75389717  6.68828034  6.80446638  6.71241906  6.69007906]]

 [[ 7.05294634  6.95817108  7.06221978  7.00069974  6.97001814]
  [ 6.95953817  6.87148173  6.81372564  6.73779401  6.70734242]
  [ 6.74810221  6.7028899   6.53152442  6.61899673  6.58892634]
  ...,
  [ 6.61125813  6.71709649  6.71727603  6.82076407  6.79777047]
  [ 6.78648476  6.7215953   6.8383393   6.74690552  6.724202  ]
  [ 6.7355749   6.6932457   6.77186244  6.82200524  6.79989222]]

 [[ 6.99713753  6.90809937  6.85059221  6.77512358  6.7441611 ]
  [ 6.78504171  6.73863132  6.56742316  6.6547594   6.62420023]
  [ 6.56464876  6.46914714  6.43231432  6.3451101   6.31467069]
  ...,
  [ 6.82166042  6.75583329  6.87292421  6.78111284  6.75804005]
  [ 6.7680784   6.7265842   6.80558078  6.85702814  6.8345485 ]
  [ 6.76947057  6.77404526  6.8734005   6.84940218  6.82734585]]


 ...,
 [[ 9.31168813  9.29242182  9.46708247  9.41655739  9.3798271 ]
  [ 9.47140339  9.34533089  9.40166529  9.3846235   9.34857902]
  [ 9.25529771  9.19144134  9.30339954  9.27286253  9.23744238]
  ...,
  [ 9.09955252  9.01626333  9.22363662  9.18795053  9.15387409]
  [ 9.13232261  8.98743982  9.15016461  9.01257472  8.97895412]
  [ 8.95343635  8.82471939  8.83663264  8.81018717  8.77699182]]

 [[ 9.52196592  9.39453005  9.45189447  9.43595195  9.3992289 ]
  [ 9.3053335   9.23983193  9.3538145   9.32229393  9.28622291]
  [ 9.31447606  9.23420517  9.38807914  9.40820402  9.37284707]
```

```
 ...,
 [ 9.17908925  9.03266972  9.19589215  9.05772539  9.02359505]
 [ 8.99611897  8.86816077  8.8801401   8.85494252  8.82125055]
 [ 8.75464369  8.61662757  8.04435294  8.15666007  8.12314395]]

[[ 9.35503898  9.28849454  9.40379661  9.37329272  9.33654626]
 [ 9.36482097  9.28281312  9.43893756  9.45833249  9.42231812]
 [ 9.31447606  9.29689248  9.45622363  9.45446616  9.419131  ]
 ...,
 [ 9.04221261  8.91281203  8.92456617  8.89933181  8.8651359 ]
 [ 8.79641393  8.65908262  8.08410419  8.19821631  8.16422705]
 [ 8.25949558  8.28079174  8.29680163  8.35285569  8.31943586]]]
```

In [14]:
```
model = build_model([X_train.shape[2], window, 50, 1])
```

```
Compilation Time :  0.026410818099975586
```

In [15]:
```
# Training the model
model.fit(
    X_train,
    y_train,
    batch_size=768,
    epochs=500,
    validation_split=0.1,
    verbose=0)
```

Out[15]:
```
<keras.callbacks.History at 0x118fcc7f0>
```

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: 0.23 MSE (0.48 RMSE)
['loss', 'acc']
Test Score: 34.43 MSE (5.87 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)
```

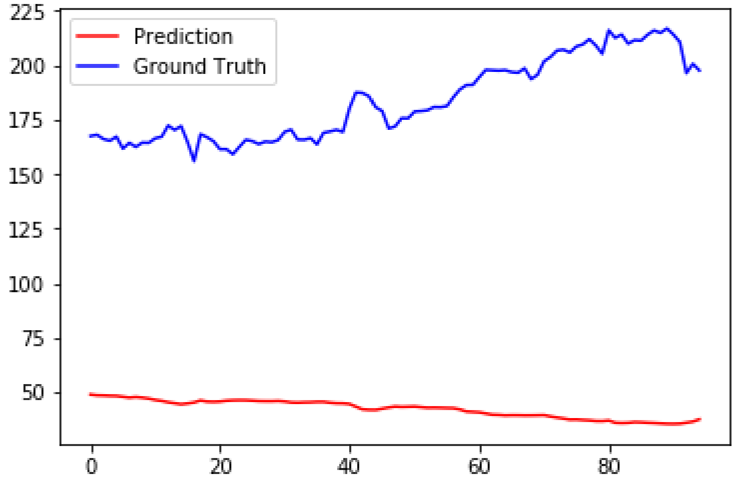
```
error_ratio [2.4213155334486691, 2.4593533107097549, 2.4240023455460791, 2.4183453801351065, 2.4622538139250
838, 2.374424564263486, 2.458856781340609, 2.3990286393200764, 2.4622113027580155, 2.4901392174042916, 2.582
706770449314l, 2.6322391971270802, 2.7955791848026537, 2.7879496317084254, 2.8677059729946257, 2.67287267135
87321, 2.4418937036306385, 2.6410038539899987, 2.6536386444273354, 2.6216312755605231, 2.5297002899716183,
2.5008688118314488, 2.4373546443249601, 2.5057225937968757, 2.5778303937069733, 2.5779882942961616, 2.563886
2446488262, 2.5930570464845832, 2.5880999189151588, 2.5999910881577306, 2.7032912354276903, 2.76074233744593
97, 2.6683616354783029, 2.6554917085658269, 2.6675109379517474, 2.5912118404100837, 2.7071959057922039, 2.75
1293111121282 8, 2.7901231026004014, 2.7720514314186211, 3.0356536410378396, 3.3215217730960944, 3.4428407093
797091, 3.432054748892264 6, 3.314206951693496, 3.2155048908613235, 2.9761903665865233, 2.9574511011573974,
3.0607443383219337, 3.0519712677113784, 3.1132661719055195, 3.1481821530587748, 3.1837184424492655, 3.213503
9633471925, 3.2185636605377175, 3.2402839830228238, 3.3422874354155674, 3.4859001564095715, 3.63900189359199
72, 3.6627249800021398, 3.7708840600345681, 3.9192603933225154, 3.9821771828504584, 3.9900805955211638, 4.02
44740919002631, 3.9911527014431698, 3.9865020903669617, 4.0474420798778805, 3.9184575973238509, 3.9644780535
266761, 4.1119877491704449, 4.2429289423971666, 4.3891390698098585, 4.4621247846776431, 4.50247906145119, 4.
569390396242639, 4.6268261197255312, 4.714533699750878, 4.6864906593306843, 4.5880095558103546, 4.8362728881
923598, 4.9088535336230192, 4.973302890946929, 4.839768941599881, 4.833373817612121, 4.8417525840924363, 4.9
482716603778583, 5.0238171954155248, 5.0240515817505171, 5.1092846713649065, 5.0377462472253827, 4.924100484
9795411, 4.4546072295570642, 4.4957206148864701, 4.2642805467967078]
```

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)
```

```
[48.957776142664287, 48.592898990566233, 48.525081828909045, 48.371357370993543, 48.309570582978402, 47.9311
36678203366, 47.527263888701548, 47.810716602996216, 47.51009993785371, 47.101272688560876, 46.4676588587018
03, 46.034965740211931, 45.408091257872449, 44.958359946088301, 44.499246375427404, 44.853176175872321, 45.3
12267149763386, 46.25097933237798, 45.702383911086571, 45.600996190436199, 45.746093927226426, 46.1314058539
40777, 46.300137887359142, 46.367046636154427, 46.340934240936839, 46.16840202169948 5, 45.963868304147098, 4
5.913549065803245, 45.901730922196862, 46.022334206606473, 45.753895988261775, 45.326159493225418, 45.227275
957586357, 45.359150346712966, 45.420451313781918, 45.580714609502984, 45.5870162501935, 45.213742561775618,
44.9510452267656 5, 44.909248211467464, 44.629697446900209, 43.399064692350812, 42.168967616702389, 41.930889
063689548, 41.8987769998 0169, 42.462292094133055, 43.005989209415027, 43.452212700672042, 43.27531638512826
4, 43.356672935942861, 43.461812712495252, 43.151431975572599, 42.873342809128374, 42.926266730343244, 42.85
1078837804771, 42.756570957484413, 42.694087334699368, 42.116406164335132, 41.159716331168468, 40.9696046452
03056, 40.786989067722189, 40.235721871660509, 39.691482808096573, 39.594551273848666, 39.367304792926454, 3
9.449803638159025, 39.430444715913183, 39.362510724403784, 39.374133082975256, 39.418041512135467, 39.487575
265170996, 38.879030831724464, 38.375331443672827, 37.933955222194413, 37.426767044468612, 37.47088768293055
2, 37.255461700711926, 37.103639621417116, 36.781912875702979, 36.742959178820726, 37.033908993063548, 35.98
498486890496 7, 35.856208350761655, 35.956901394539294, 36.275748411855091, 36.18092309755643, 35.99028662830
1163, 35.865962726164163, 35.678644195397801, 35.513160487826447, 35.466876253437619, 35.568270243600985, 3
6.009925139183686, 36.521144553150883, 37.551188855291429]
[ 167.5       168.100006  166.149994  165.350006  167.259995  161.740005
  164.389999  162.509995  164.490005  164.389999  166.479996  167.210007
  172.350006  170.300003  172.110001  164.740005  155.960007  168.399994
  166.979996  165.149994  161.470001  161.5       159.149994  162.550003
  165.800003  165.190002  163.809998  164.970001  164.699997  165.679993
  169.440002  170.460007  165.910004  165.809998  166.580002  163.690002
  169.        169.610001  170.369995  169.399994  180.110001  187.550003
  187.350006  185.839996  180.759995  179.        171.        171.960007
  175.729996  175.679993  178.770004  179.        179.369995  180.869995
  180.770004  181.300003  185.389999  188.929993  190.940002  191.029999
  194.589996  197.929993  197.75      197.580002  197.800003  196.899994
  196.619995  198.679993  193.660004  195.690002  201.860001  203.839996
  206.809998  207.199997  205.940002  208.690002  209.630005  212.029999
  209.160004  205.320007  216.139999  212.630005  214.179993  209.979996
  211.610001  211.360001  214.080002  216.050003  214.929993  216.960007
  214.139999  210.710007  196.419998  200.710007  197.679993]
```

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 [ ]: