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

In [2]:
```
start = datetime.datetime(1986, 3, 12) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('ORCL', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                Open      High       Low     Close  Adj Close     Volume
Date
1986-03-12  0.063272  0.064043  0.063272  0.063272   0.057184  393012000
1986-03-13  0.064815  0.065586  0.064815  0.064815   0.058579  125290800
1986-03-14  0.067130  0.067901  0.067130  0.067130   0.060671   57866400
1986-03-17  0.066358  0.066358  0.065586  0.065586   0.059277   28285200
1986-03-18  0.064815  0.064815  0.064043  0.064043   0.057882   32335200
```

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					
1986-03-12	0.063272	0.064043	0.063272	0.057184	0.063272
1986-03-13	0.064815	0.065586	0.064815	0.058579	0.064815
1986-03-14	0.067130	0.067901	0.067130	0.060671	0.067130
1986-03-17	0.066358	0.066358	0.065586	0.059277	0.065586
1986-03-18	0.064815	0.064815	0.064043	0.057882	0.064043

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

Out[9]:

	Open	High	Low	Adj Close	Close
0	0.063272	0.064043	0.063272	0.057184	0.063272
1	0.064815	0.065586	0.064815	0.058579	0.064815
2	0.067130	0.067901	0.067130	0.060671	0.067130
3	0.066358	0.066358	0.065586	0.059277	0.065586
4	0.064815	0.064815	0.064043	0.057882	0.064043

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(0.99 * 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 (7900, 20, 5)
y_train (7900,)
X_test (80, 20, 5)
y_test (80,)
```

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

```
Compilation Time :  0.02963876724243164
```

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 0x11f9422b0>
```

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: 0.00 MSE (0.05 RMSE)
['loss', 'acc']
Test Score: 0.00 MSE (0.03 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)
```

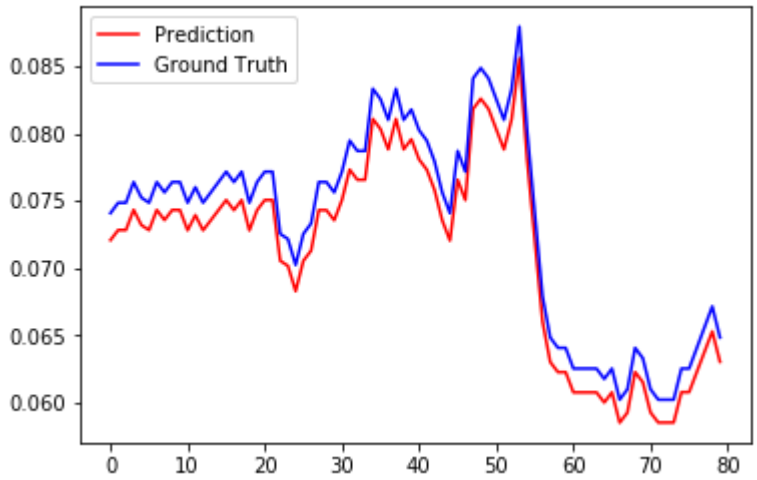
```
error_ratio [0.027922291025096913, 0.027891289065870017, 0.027914793080967337, 0.027837497289325164, 0.02794
2707192179173, 0.027998542254387804, 0.02792703046045597, 0.028012968878575961, 0.027996139944994169, 0.0280
28512369295822, 0.028162063747391164, 0.02811373009188944 4, 0.028216424895885162, 0.028187848461752596, 0.02
8154004318578396, 0.028110771350192687, 0.028165704281799098, 0.028117507429527011, 0.028273392093076888, 0.
028172558929406621, 0.028123652701703472, 0.028126843545077662, 0.028434899947494818, 0.028460331806053007,
0.028585769172227105, 0.028431826361786205, 0.028382684182614693, 0.028179531854369388, 0.02818106827428024
1, 0.028232523191804582, 0.028128970784997209, 0.027973491120569571, 0.028023697717975971, 0.028021570701249
443, 0.027710207560735167, 0.027758878325420566, 0.02785782152750671, 0.027700993413335429, 0.02785486785094
804, 0.027805725076799348, 0.027912702096977382, 0.02796982810568327, 0.02808236109631479, 0.028248833981281
196, 0.028365745915239771, 0.028070730670158639, 0.028187236945217187, 0.02773707615831511, 0.02769648659026
136, 0.027756688309151345, 0.027863557880672962, 0.02796546508251252, 0.027808738388822807, 0.02749881623165
3844, 0.028009124900948423, 0.028412914301389369, 0.028814214797224125, 0.029011635287785609, 0.029058243286
069319, 0.029057296938051147, 0.029162038896346276, 0.029164878240467873, 0.02916641622507532, 0.02916641622
507532, 0.029214088353749501, 0.029153994173080733, 0.029293155251652214, 0.029218983050458114, 0.0289820678
3982471, 0.028999746081750155, 0.029119733070195108, 0.02913652589110316, 0.029102225219764355, 0.0290685181
66897928, 0.028886933445888774, 0.028870379770831844, 0.028758456341975824, 0.028651499821336968, 0.02854707
8115045199, 0.028699211171270056]
```

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)
```

```
[0.07206186756211853 4, 0.07281509318754785, 0.07281342821778467 9, 0.07432011402722478 5, 0.0731859854383257
7 4, 0.07280749624008578 4, 0.07431364069274629 1, 0.07355646503417946 6, 0.07430864478157224 5, 0.074306304816338
584, 0.07279591675188368 2, 0.07392470091145178 5, 0.07279206807805929 7, 0.07354395416472660 2, 0.0742972353160
53395, 0.07505027877362636 9, 0.07429638985416239 8, 0.07504978705489945 3, 0.07278803533722588 7, 0.07429589453
3054546, 0.07504933846939417, 0.07504910554999721 9, 0.07052561129897766 4, 0.07014854901920038 2, 0.0682646037
93330325, 0.07052582207280380 7, 0.07127891312003403 8, 0.0742953906719276, 0.07429527965167928 6, 0.0735407588
2104201, 0.07504895027039923 4, 0.07731230492467873 5, 0.07655854643692401 6, 0.07655870483954266 3, 0.081086087
680096564, 0.08033207179345648 6, 0.07882315851777739 5, 0.08106814680623703, 0.07882338502652185, 0.07957729
5596294242, 0.07806791358477557 8, 0.07731258041537514 3, 0.07580326533070343 2, 0.07353959226699843 8, 0.072030
792832441681, 0.07655504397902269 9, 0.07504469733474347 6, 0.08183513269209358 2, 0.08258955937624036 5, 0.0818
33571074461389, 0.08032389062438655 1, 0.07881490453912942 7, 0.08107831436676782 3, 0.08560885775285251 3, 0.07
8060591152565917, 0.07202748912417068 8, 0.06599928249765003 9, 0.06298762596777933 4, 0.06223457264721273, 0.0
622346298797542 62, 0.06072901801452355 2, 0.06072885047034869 8, 0.06072875971725398 6, 0.06072875971725398 6,
0.05997585992894373 4, 0.06072949272301895 3, 0.05847216577019338 1, 0.05922646298199063 5, 0.0622391798668051 8
9, 0.06148883927418703 8, 0.05923217487837462 2, 0.05848106493731464 9, 0.05848301415065691 7, 0.058484929756872
604, 0.06074525583746952 9, 0.06074623317848950 5, 0.06225270820880724 6, 0.06375920320088136 8, 0.0652668229081
21039, 0.06300675580979796 2]
[ 0.074074  0.074846  0.074846  0.076389  0.075231  0.074846  0.076389
  0.075617  0.076389  0.076389  0.074846  0.076003  0.074846  0.075617
  0.076389  0.07716   0.076389  0.07716   0.074846  0.076389  0.07716
  0.07716   0.072531  0.072145  0.070216  0.072531  0.073302  0.076389
  0.076389  0.075617  0.07716   0.079475  0.078704  0.078704  0.083333
  0.082562  0.081019  0.083333  0.081019  0.08179   0.080247  0.079475
  0.077932  0.075617  0.074074  0.078704  0.07716   0.084105  0.084877
  0.084105  0.082562  0.081019  0.083333  0.087963  0.080247  0.074074
  0.067901  0.064815  0.064043  0.064043  0.0625    0.0625    0.0625
  0.0625    0.061728  0.0625    0.060185  0.060957  0.064043  0.063272
  0.060957  0.060185  0.060185  0.060185  0.0625    0.0625    0.064043
  0.065586  0.06713   0.064815]
```

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 [ ]: