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(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(0.99 * 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]
    #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(0.1))

    model.add(LSTM(
        layers[2],
        return_sequences=False))
    model.add(Dropout(0.1))

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

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

```
Compilation Time :  0.025911808013916016
```

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 0x11befb630>
```

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 (0.06 RMSE)
['loss', 'acc']
Test Score: 0.08 MSE (0.28 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 [0.089208091241010257, 0.1236897150590559, 0.12593425551769011, 0.11674520090527052, 0.133997524
76934901, 0.10150636889914777, 0.1250931112540048, 0.1232439778048573, 0.13150554933014891, 0.11697975492292
678, 0.12950798549146825, 0.12528262649273603, 0.1249740865565121, 0.13312055738652551, 0.14172923065415888,
0.15003102358509302, 0.15028772647686095, 0.15582090238749702, 0.15343146524819318, 0.17506863907708325, 0.1
5518743934721879, 0.1922407508313364, 0.17703431743626297, 0.17890826314695718, 0.17648392023379533, 0.05247
5927804712441, 0.14254052215957103, 0.090430030547184348, 0.11132491781174769, 0.1052163513274178, 0.1138746
9735928657, 0.10298511385314235, 0.11264410883323106, 0.11226382801891588, 0.10631630805303471, 0.1175466740
5039254, 0.12730641697762657, 0.11289497866246689, 0.12728452145252778, 0.11819150385154087, 0.1069537976708
923, 0.1147832079147606, 0.10626875489285914, 0.11451513989608508, 0.10849277555395243, 0.12552182856492067,
0.12279878385601117, 0.13254367763315367, 0.13801604133726486, 0.12381428091791635, 0.12718854928485701, 0.1
2896014866939187, 0.14819173302055466, 0.12686153861420735, 0.15245207976337039, 0.16103226488760414, 0.1435
6665214696029, 0.15727761813379026, 0.14020101871009927, 0.1375503218823444, 0.13924076725797452, 0.14414163
948102887, 0.1440797655421302, 0.14590181774418842, 0.10170648026704288, 0.14737554604584613, 0.122129415303
10295, 0.12129166431869609, 0.11304007923670634, 0.13201515526132646, 0.11244388235737413, 0.126592370874103
69, 0.10590814758448452, 0.11936693026359957, 0.12758374160008401, 0.11558778903503342, 0.12619863714248214,
0.10366398638984697, 0.14002203957160453, 0.13320225746826408]
```

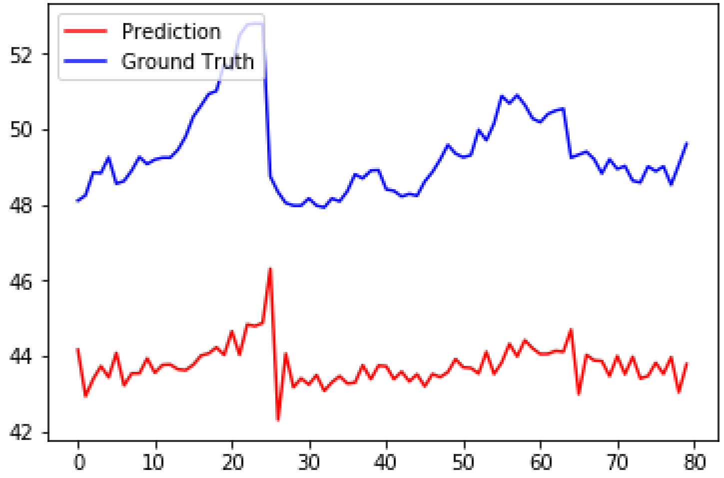
In [18]:

```
# 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)
```

```
[44.160522114003335, 42.930002253749144, 43.386190410859648, 43.725284837057536, 43.430429894472013, 44.0760
03889583653, 43.214200241444182, 43.53462201111882, 43.534915077669666, 43.930966325693078, 43.5499346900115
9, 43.757897652317361, 43.769898870045196, 43.640547051810145, 43.618046786335555, 43.764038506632502, 44.00
6380173281165, 44.063920192823581, 44.224559097686623, 44.022960259265808, 44.650764233678977, 44.0263444806
75817, 44.833017371099317, 44.787199013311351, 44.870992362997512, 46.309849671966788, 42.300470804001883, 4
4.056014282634202, 43.164694889101604, 43.403267552444127, 43.236461079666412, 43.491068372104067, 43.068576
573196509, 43.299079576991296, 43.45954375798204, 43.264410447182627, 43.289027956423425, 43.75075809805350
2, 43.378580180442299, 43.740271529100831, 43.723597228571748, 43.380632805242023, 43.578921294463456, 43.31
9285016174405, 43.509530295222525, 43.188856729664174, 43.516257500921782, 43.433202596477088, 43.5670501988
17334, 43.912947929164581, 43.692778844538992, 43.677361914074162, 43.529315324817155, 44.104798413050908, 4
3.515910883077375, 43.823072397497526, 44.317486789993495, 43.982533838406582, 44.413220273464269, 44.200241
547819985, 44.046878800499442, 44.050491880411705, 44.13154005575386, 44.105001159255139, 44.69430186891948
9, 42.985054169900621, 44.023444467549552, 43.877969100835358, 43.861852695798227, 43.462316534659948, 43.99
3229479847109, 43.511745035133004, 43.9729114087976, 43.399532974017646, 43.464619248990722, 43.815467935769
057, 43.518076104543759, 43.96265584302693, 43.034256617034949, 43.778593515014556]
[ 48.099998  48.240002  48.849998  48.830002  49.25      48.549999
  48.619999  48.900002  49.259998  49.07      49.189999  49.240002
  49.240002  49.450001  49.799999  50.330002  50.619999  50.93      51.009998
  51.73      51.580002  52.490002  52.77      52.799999  52.790001
  48.740002  48.330002  48.040001  47.970001  47.970001  48.16      47.970001
  47.919998  48.16      48.080002  48.349998  48.799999  48.689999
  48.900002  48.91      48.400002  48.360001  48.209999  48.279999  48.23
  48.610001  48.860001  49.189999  49.580002  49.349998  49.25      49.310001
  49.98      49.700001  50.150002  50.880001  50.68      50.900002
  50.639999  50.279999  50.18      50.400002  50.490002  50.540001
  49.240002  49.32      49.400002  49.200001  48.82      49.200001
  48.939999  49.02      48.630001  48.580002  49.009998  48.880001
  49.009998  48.52      49.060001  49.610001]
```

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 [ ]: