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

In [2]:
```
start = datetime.datetime(2012, 5, 18) # or start = '1/1/2016'
end = datetime.date.today()
df = web.DataReader('FB', 'yahoo', start, end)
print (df.head())  # print first rows of the prices data
```

```
                 Open       High        Low      Close  Adj Close     Volume
Date
2012-05-18  42.049999  45.000000  38.000000  38.230000  38.230000  573576400
2012-05-21  36.529999  36.660000  33.000000  34.029999  34.029999  168192700
2012-05-22  32.610001  33.590000  30.940001  31.000000  31.000000  101786600
2012-05-23  31.370001  32.500000  31.360001  32.000000  32.000000   73600000
2012-05-24  32.950001  33.209999  31.770000  33.029999  33.029999   50237200
```

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					
2012-05-18	42.049999	45.000000	38.000000	38.230000	38.230000
2012-05-21	36.529999	36.660000	33.000000	34.029999	34.029999
2012-05-22	32.610001	33.590000	30.940001	31.000000	31.000000
2012-05-23	31.370001	32.500000	31.360001	32.000000	32.000000
2012-05-24	32.950001	33.209999	31.770000	33.029999	33.029999

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

Out[9]:

	Open	High	Low	Adj Close	Close
0	42.049999	45.000000	38.000000	38.230000	38.230000
1	36.529999	36.660000	33.000000	34.029999	34.029999
2	32.610001	33.590000	30.940001	31.000000	31.000000
3	31.370001	32.500000	31.360001	32.000000	32.000000
4	32.950001	33.209999	31.770000	33.029999	33.029999

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(0.9 * 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 :  0.03105306625366211
```

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 0x119cc0ba8>
```

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 (0.09 RMSE)
['loss', 'acc']
Test Score: 0.21 MSE (0.46 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 [0.13171406582972178, 0.13753617490568026, 0.13651725194460584, 0.16853305046814882, 0.195842201
39950337, 0.1901242163236414, 0.18488811191215282, 0.16251894021180902, 0.15777569209052777, 0.1868864619081
3318, 0.18231673952518568, 0.16344961195247576, 0.16413727163259351, 0.18563615367998199, 0.1040922344559078
7, 0.088723485478876318, 0.1347325594681521, 0.13453923131020251, 0.13079310128124799, 0.15033481278266692,
0.18780117153509868, 0.17338535133510113, 0.19305361951439126, 0.17860916606761412, 0.19822596941474413, 0.1
694834060249395, 0.11780149843024934, 0.16017028039248093, 0.12900634765202956, 0.13233897408546902, 0.09759
3659966452861, 0.13293550961206924, 0.1152183187315412, 0.16233631969441165, 0.19542250142930007, 0.21688984
665594724, 0.26209661205962131, 0.2491152424344576, 0.24126983811349478, 0.22124308443752883, 0.254203874041
67522, 0.2587883298130782, 0.24819821640402417, 0.23365139041402849, 0.24591559316368072, 0.2267141747837719
8, 0.22522972772918259, 0.2859924911453593, 0.29447234911530451, 0.23704073489889166, 0.23667731210138498,
0.22516029241969182, 0.21419697867215826, 0.22805401680848814, 0.25872295569088322, 0.2448901490981048, 0.23
971713673556816, 0.18867633036549525, 0.20060830954554665, 0.23994535873953171, 0.242061371480226, 0.2261169
9276697683, 0.18435098366924918, 0.19448208910511733, 0.20428898761047654, 0.23312150034082801, 0.2208961797
9478771, 0.20805055898591229, 0.19027750326072801, 0.20542883212157514, 0.21873876901182365, 0.2457097207407
7213, 0.27011715019018134, 0.26352616467510082, 0.23858872145042653, 0.25249311474599168, 0.2634328975160142
9, 0.22949626868152784, 0.26026134362948006, 0.24921014906536643, 0.24710277197736086, 0.2183305766508703,
0.22857320251549451, 0.20919922536158464, 0.24415404884632341, 0.23982637218669711, 0.2251955068152478, 0.21
866148711827527, 0.12163534361232031, 0.14993617258242753, 0.20826148580586179, 0.22725186243236273, 0.25603
582532780789, 0.23298970613077796, 0.23512670400676394, 0.21234593848325156, 0.24903343979846926, 0.25956045
588957477, 0.25889901657669245, 0.24052818525766084, 0.25181009599033755, 0.24644193113444723, 0.25885301721
185749, 0.26466842553680392, 0.28017910393704204, 0.27328853636820605, 0.24866275344469746, 0.25233798818304
431, 0.20367150668069534, 0.21305369743889679, 0.20209457145110266, 0.20825798274925078, 0.3060827069278138
1, 0.32426773301588829, 0.31371321458432333, 0.33486372308457968, 0.2755800769352974, 0.2706523337543596, 0.
28487571982872928, 0.28427264386135498, 0.27360559089368985, 0.26965923735684383, 0.26046456627505354, 0.265
95552859833593, 0.25892210563878959, 0.25896728589975559, 0.28144793078030395, 0.2727778166016146, 0.2681915
9921741464, 0.3064249373819754, 0.2903132668164361, 0.31021573215644827, 0.30891740152514013, 0.296431374387
5957, 0.2036142993189809, 0.23385362273658461, 0.2150681223778359]
```

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)
```

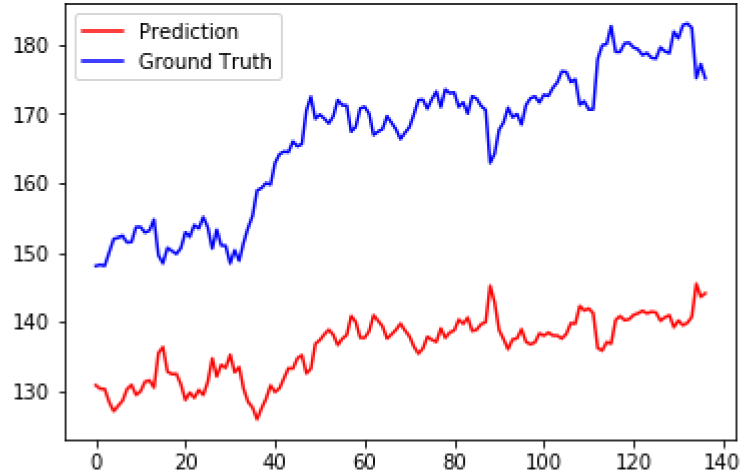
```
[130.82809737055717, 130.31673916857321, 130.28399414672236, 128.40029893881865, 127.07362796041154, 127.826
99731120327, 128.60286424352057, 130.28605535872325, 130.8802732992184, 129.4226582996358, 129.939803661832,
131.3421711006968, 131.53087589512836, 130.48691752508603, 135.49593170875102, 136.34316149128398, 132.7889
9221030861, 132.43261744813043, 132.47339661894711, 130.95317756714752, 128.69998671783833, 129.752770329684
89, 129.00510210316114, 130.15340319455802, 129.41632960579352, 131.3314880816057, 134.71086074894555, 132.0
8406351191786, 133.78134969224453, 133.33462810633949, 135.23218875420312, 132.699517955332, 133.44472961067
314, 130.28931423205884, 128.40648374651522, 127.5957757612055, 125.90160886391284, 127.49824002596506, 128.
87608808985917, 130.79295844984642, 129.85129799925051, 130.3952341410519, 131.81399944153097, 133.287243282
57377, 133.23534989917405, 134.73391145018215, 135.16648939537112, 132.53576764526744, 133.22030178385765, 1
36.8184532854801, 137.35191819066426, 138.18600231128326, 138.84896681621581, 138.12095614557302, 136.630539
08920273, 137.54626954358369, 138.07987961733946, 140.82891172614475, 139.99570106558861, 137.7076810655399
9, 137.67435645809582, 138.64908569316961, 140.92951017180272, 140.1527955311748, 139.31871895042846, 137.56
957360090829, 138.18538364855672, 138.85180860377889, 139.73212678923323, 138.7390118300512, 137.88845261422
028, 136.40416797819927, 135.39696001605051, 136.14281113381827, 137.8342932108097, 137.39795770047024, 137.
09474190559814, 139.04067979264488, 137.67778871983901, 138.45549296041597, 138.7616216469619 4, 140.32316866
737474, 139.70677420650912, 140.59717491893218, 138.66450393339474, 138.86621696580113, 139.65934420113931,
139.94041397276757, 145.2076166532552, 142.79923609258358, 138.77790111646584, 137.48603784196675, 136.03910
935852909, 137.44640377559247, 137.60532133962286, 138.92074254870505, 137.09801478784223, 136.7381733799837
2, 137.02449340938958, 138.32011077149392, 137.99218072557656, 138.43404870289777, 138.01452800645794, 137.9
9664835146245, 137.56668926902037, 138.24831840714549, 139.79755343741914, 139.72266085601211, 142.289655482
75104, 141.62604950029743, 141.91895550618869, 141.21984496369828, 136.19352285768477, 135.82600445180668, 1
37.06187621548241, 136.837941462603, 140.26559463822321, 140.80956155123113, 140.22367706039009, 140.3518177
0909003, 140.98556043084159, 141.21899618772031, 141.58272415970254, 141.21349444078331, 141.4464057803207,
141.34600556584212, 140.14615161978793, 140.63727200866813, 140.94085239982527, 139.20432456260392, 140.1752
5794046657, 139.50374317301731, 139.83311612079945, 140.70933610826179, 145.50342671991405, 143.598875697288
58, 144.10715150467189]
[ 148.059998  148.240005  148.070007  150.039993  151.960007  152.130005
  152.380005  151.460007  151.529999  153.610001  153.630005  152.809998
  153.119995  154.710007  149.600006  148.440002  150.679993  150.25
  149.800003  150.639999  152.869995  152.25      153.910004  153.399994
  155.070007  153.589996  150.580002  153.240005  151.039993  150.979996
  148.429993  150.339996  148.820007  151.440002  153.5       155.270004
  158.899994  159.259995  159.970001  159.729996  162.860001  164.139999
  164.529999  164.429993  166.        165.279999  165.610001  170.440002
  172.449997  169.25      169.860001  169.300003  168.589996  169.619995
  171.979996  171.229996  171.179993  167.399994  168.080002  170.75      171.
  170.        166.910004  167.410004  167.779999  169.639999  168.710007
  167.740005  166.320007  167.240005  168.050003  169.919998  171.970001
  172.020004  170.720001  172.089996  173.210007  170.949997  173.509995
  172.960007  173.050003  170.960007  171.639999  170.009995  172.520004
  172.169998  171.110001  170.539993  162.869995  164.210007  167.679993
  168.729996  170.869995  169.470001  169.960007  168.419998  171.240005
```

```
 172.229996  172.5       171.589996  172.740005  172.550003  173.740005
 174.520004  176.110001  176.029999  174.559998  174.979996  171.270004
 171.800003  170.600006  170.630005  177.880005  179.869995  180.059998
 182.660004  178.919998  178.919998  180.169998  180.25      179.559998
 179.300003  178.460007  178.770004  178.070007  177.949997  179.589996
 179.        178.740005  181.860001  180.869995  182.779999  183.029999
 182.419998  175.130005  177.179993  175.100006]
```

In [19]:

```python
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 [ ]: