# **RNN Multi-Variate analysis**

# Load preprocessed MSFT stock data with the augmented XLK prices for one year (2014) and make short term (intraday) predictions.

**Data**: We have stock data for every minute during trading hours (09:30 to 16:00 EST). This is equivalent to 390 samples per day. We will use two months worth of stock data (~ 44 \* 390 = 17160 samples) to train a multi-step RNN based network.

## [Prologue: Correlation analysis between MSFT and XLK stock prices](#MSFT_XLK_Corr)

Determine the best two months (contiguous) during the year 2014 when MSFT and XLK are most positively correlated. This is needed for Task:5 below.

## [Task:4 Effect of XLK prices on MSFT price - Study1](#RNN_MSFT_XLK_1)

From our single-variate analysis on single stock, we learnt that model that predicted stock prices performed better compared to the model that predicted stock returns or up/down indicators. We will now perform a multi-variate price prediction for MSFT stock (one-minute intervals) with the effect of tech sector (XLK) overall prices. The first experiment uses the same time-period (2014/01 to 2014/02) as the single-variate analysis in Task#3 (see MSFT-RNN-single-variate.ipynb).

## [Task:5 Effect of XLK prices on MSFT price - Study2](#RNN_MSFT_XLK_2)

This is same as above except the 2-month training period is different. We will analyse to pick the best 2-month period during 2014 year where MSFT prices positively correlates the most against XLK prices.

## Prologue: Correlation analysis between MSFT and XLK prices

In [3]:

import pandas as pd

import glob

path ='C:/cygwin64/home/mpatnam/CSCIE63/CSCI E-89 Deep Learning/Project/Processed'

stockfile = path + "/msft\_stock.2014.csv" # Load 2014 year data

stockdata = pd.read\_csv(stockfile)

print('Total rows x columns:', stockdata.shape)

stockdata

Total rows x columns: (97701, 25)

Out[3]:

|  | **Close** | **...** | **DateTime** | **CumReturn** | **etf.Close** | **etf.Return** | **spy.Close** | **spy.Return** | **spy.CumReturn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 33.6842 | ... | 2014-01-02 09:30:00 | 0.000000 | 33.2164 | -0.196504 | 170.524 | -0.048650 | 0.000000 |
| **1** | 33.7157 | ... | 2014-01-02 09:31:00 | 0.093516 | 33.2024 | -0.042148 | 170.529 | 0.002932 | 0.002932 |
| **2** | 33.6120 | ... | 2014-01-02 09:32:00 | -0.214344 | 33.2071 | 0.000000 | 170.589 | 0.038118 | 0.038118 |
| **3** | 33.6120 | ... | 2014-01-02 09:33:00 | -0.214344 | 33.2118 | -0.013849 | 170.598 | 0.010552 | 0.043396 |
| **4** | 33.6300 | ... | 2014-01-02 09:34:00 | -0.160906 | 33.1891 | -0.068349 | 170.515 | -0.048652 | -0.005278 |

97701 rows × 25 columns

In [6]:

**#**

**# Data wrangling:**

**# 1. Convert Date and Time columns into pandas DateTime type**

**# 2. create index on date column for faster lookups**

**## This is similar to RNN single-variate analysis ##**

|  | **Close** | **...** | **DateTime** | **CumReturn** | **etf.Close** | **etf.Return** | **etf.CumReturn** | **spy.Close** | **spy.Return** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DateTime** |  |  |  |  |  |  |  |  |  |
| **2014-01-02 09:30:00** | 33.6842 | ... | 2014-01-02 09:30:00 | 0.000000 | 33.2164 | -0.196504 | 0.000000 | 170.524 | -0.048650 |
| **2014-01-02 09:31:00** | 33.7157 | ... | 2014-01-02 09:31:00 | 0.093516 | 33.2024 | -0.042148 | -0.042148 | 170.529 | 0.002932 |
| **2014-01-02 09:32:00** | 33.6120 | ... | 2014-01-02 09:32:00 | -0.214344 | 33.2071 | 0.000000 | -0.027998 | 170.589 | 0.038118 |

97701 rows × 25 columns

In [20]:

**#**

**# Find the correlation of MSFT stock (close price) against XLK and SPY prices**

**# Entire 2014 data**

**#**

df=stockdata[['Close', 'etf.Close', 'spy.Close']]

df=df.rename(index=str, columns={"Close":"MSFT.Close", "etf.Close":"XLK.Close", "spy.Close":"SPY.Close"})

df.corr()

Out[20]:

|  | **MSFT.Close** | **XLK.Close** | **SPY.Close** |
| --- | --- | --- | --- |
| **MSFT.Close** | 1.000000 | 0.964311 | 0.941512 |
| **XLK.Close** | 0.964311 | 1.000000 | 0.987342 |
| **SPY.Close** | 0.941512 | 0.987342 | 1.000000 |

In [96]:

**# plot cum returns of stock vs etf vs index – Can see strong correlation!**

import matplotlib.pyplot as plt

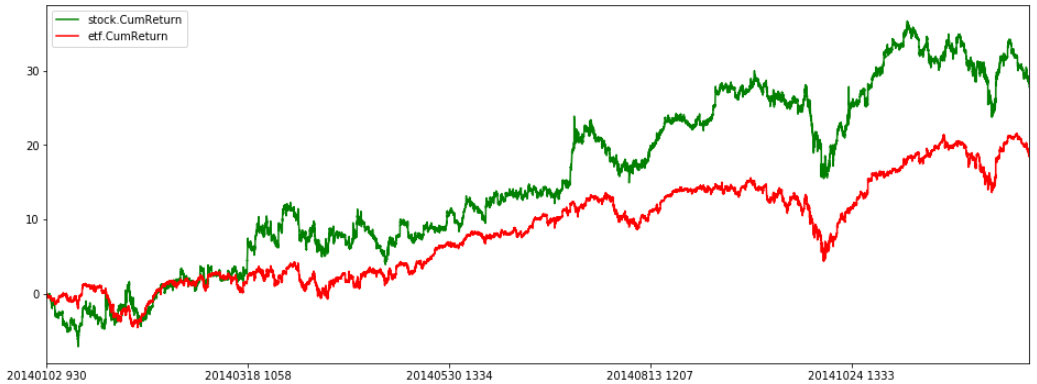
plt.figure(figsize=(16,6))

plt.subplot(1,1,1)

stockdata.plot(x='DateTimeStr', y='stock.CumReturn', color='green', ax=plt.gca())

stockdata.plot(x='DateTimeStr', y='etf.CumReturn', color='red', ax=plt.gca())

plt.show()



#### **OBSERVATION**:

We can visually see that MSFT stock is positively correlated to its sector ETF (XLK) performance. This is for the entire 2014 data. This is also clear from the correlation table above (corr = 0.96). However, we have been training the RNN time-series model with 2 months of data only for short term predictions. Hence, we would determine the best two months during 2014 when MSFT and XLK are most positively correlated. We will run two experiments -

1. Pick the 2 months where MSFT and XLK are highly correlated (06/01 to 07/31)
2. Use the first two months (this is so we can compare with MSFT single-variate analysis for the same training period)

In [166]:

**#**

**# Find the correlation of MSFT stock (close price) against XLK and SPY prices**

**#**

num\_samples\_one\_month = 22 \* 390

month\_idx=0

for idx in range(0, len(stockdata), num\_samples\_one\_month):

month\_idx +=1

print("Month[", month\_idx, "], Time period: ", idx, "->", idx+(2\*num\_samples\_one\_month))

df=stockdata[['Close', 'etf.Close', 'spy.Close']][idx:idx+(2\*num\_samples\_one\_month)]

df=df.rename(index=str, columns={"Close":"MSFT.Close", "etf.Close":"XLK.Close", "spy.Close":"SPY.Close"})

print(df.corr())

print()

Month[ 1 ], Time period: 0 -> 17160

MSFT.Close XLK.Close SPY.Close

MSFT.Close 1.000000 0.631268 0.496683

XLK.Close 0.631268 1.000000 0.940809

SPY.Close 0.496683 0.940809 1.000000

…

Month[ 9 ], Time period: 68640 -> 85800

MSFT.Close XLK.Close SPY.Close

MSFT.Close 1.000000 0.968361 0.954780

XLK.Close 0.968361 1.000000 0.987432

SPY.Close 0.954780 0.987432 1.000000

…

Month[ 12 ], Time period: 94380 -> 111540

MSFT.Close XLK.Close SPY.Close

MSFT.Close 1.000000 0.659188 0.374433

XLK.Close 0.659188 1.000000 0.914817

SPY.Close 0.374433 0.914817 1.000000

#### **OBSERVATION:** From above, the best two contiguous months where MSFT and XLK prices are most correlated (=0.9683) is between 2014/09/01 and 2014/10/31.

In [165]:

**# Check if MSFT stock is correlated (+ve or -ve) against rest of the data columns like market-volume, RSI and Earnings**

**# This is to see if we could use any of this an input during multi-variate analysis.**

df=stockdata[['Close', 'Volume', 'RSI', 'Earnings']]

df=df.rename(index=str, columns={"Close":"MSFT.Close", "Volume":"MSFT.Volume", "RSI":"MSFT.RSI", "Earnings":"MSFT.Earnings"})

df.corr()

Out[165]:

|  | **MSFT.Close** | **MSFT.Volume** | **MSFT.RSI** | **MSFT.Earnings** |
| --- | --- | --- | --- | --- |
| **MSFT.Close** | 1.000000 | -0.008097 | 0.000796 | 0.000563 |
| **MSFT.Volume** | -0.008097 | 1.000000 | 0.004302 | 0.003538 |
| **MSFT.RSI** | 0.000796 | 0.004302 | 1.000000 | 0.108605 |
| **MSFT.Earnings** | 0.000563 | 0.003538 | 0.108605 | 1.000000 |

#### **OBSERVATION**: It is obvious there is no correlation (+ve or -ve) between MSFT stock price and the above data columns. So, we won’t be using any of this columns for further experiments.

## **Task:5 Effect of XLK prices on MSFT price**

### Perform a multi-variate price prediction for MSFT stock price (one-minute intervals) using the effects of tech sector ETF (XLK) prices. Use the 2 month period with the best positive correlation between MSFT and XLK prices.

In [134]:

**# set training and test periods**

train\_startdate='2014-09-01 00:00:00' # train start-date

train\_lastdate='2014-11-01 00:00:00' # train end-date (2 months of trading data)

test\_startdate=train\_startdate # same as train start-date

test\_lastdate='2014-11-03 23:59:00' # test end-date (extends beyond training date for short term predictions)

test\_pred\_start = pd.to\_datetime('2014-11-03 09:30:00', format="%Y-%m-%d %H:%M:%S") # we will predict prices at one hour interval on this day

MSFT\_PRICE\_COLUMN\_NUM = stockdata.columns.get\_loc('Close') # MSFT stock price column

XLK\_PRICE\_COLUMN\_NUM = stockdata.columns.get\_loc('etf.Close')

In [135]:

**# Fetch stock samples into a numpy array (for training purpose)**

print('train: [', train\_startdate, ':', train\_lastdate, ']')

prices\_train = stockdata.loc[(stockdata['DateTime'] > train\_startdate) & (stockdata['DateTime'] < train\_lastdate)]

prices\_test = stockdata.loc[(stockdata['DateTime'] > test\_startdate) & (stockdata['DateTime'] < test\_lastdate)]

prices\_test

train: [ 2014-09-01 00:00:00 : 2014-11-01 00:00:00 ]

test: [ 2014-09-01 00:00:00 : 2014-11-03 23:59:00 ]

#train samples: 17136

#test samples: 17527

Out[135]:

|  | **Close** | **...** | **CumReturn** | **etf.Close** | **etf.Return** | **etf.CumReturn** | **spy.Close** | **spy.Return** | **spy.CumReturn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DateTime** |  |  |  |  |  |  |  |  |  |
| **2014-09-02 09:30:00** | 41.8131 | ... | 24.132679 | 38.0263 | -0.099044 | 14.480498 | 188.043 | -0.024988 | 10.273627 |
| **2014-09-02 09:31:00** | 41.8222 | ... | 24.159695 | 38.0122 | -0.049433 | 14.438049 | 188.057 | 0.007445 | 10.281837 |
| **2014-09-02 09:32:00** | 41.7808 | ... | 24.036789 | 38.0263 | 0.049464 | 14.480498 | 188.071 | 0.007445 | 10.290047 |
| **2014-09-02 09:33:00** | 41.7854 | ... | 24.050445 | 38.0228 | -0.009204 | 14.469961 | 188.062 | -0.004785 | 10.284769 |

17527 rows × 25 columns

In [136]:

**#**

**# Normalize the MSFT stock price using mean/SD -**

**#**

mean = prices\_train['Close'].mean()

std = prices\_train['Close'].std()

prices\_train['stock.CloseNormal'] = (prices\_train['Close'] - mean) / std

prices\_test['stock.CloseNormal'] = (prices\_test['Close'] - mean) / std

prices\_test

MSFT close price mean: 42.11 , std: 1.0413

Out[136]:

|  | **Close** | **Volume** | **...** | **CumReturn** | **etf.Close** | **etf.Return** | **etf.CumReturn** | **spy.Open** | **spy.Close** | **spy.Return** | **spy.CumReturn** | **stock.CloseNormal** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DateTime** |  |  |  |  |  |  |  |  |  |  |  |  |
| **2014-09-02 09:30:00** | 41.8131 | 1592540 | ... | 24.132679 | 38.0263 | -0.099044 | 14.480498 | 188.090 | 188.043 | -0.024988 | 10.273627 | -0.287300 |
| **2014-09-02 09:31:00** | 41.8222 | 104120 | ... | 24.159695 | 38.0122 | -0.049433 | 14.438049 | 188.043 | 188.057 | 0.007445 | 10.281837 | -0.278561 |
| **2014-09-02 09:32:00** | 41.7808 | 77246 | ... | 24.036789 | 38.0263 | 0.049464 | 14.480498 | 188.057 | 188.071 | 0.007445 | 10.290047 | -0.318320 |
| **2014-09-02 09:33:00** | 41.7854 | 44236 | ... | 24.050445 | 38.0228 | -0.009204 | 14.469961 | 188.071 | 188.062 | -0.004785 | 10.284769 | -0.313903 |

17527 rows × 26 columns

In [137]:

**#**

**# Normalize the XLK stock price using mean/SD - similarly**

**#**

mean\_etf = prices\_train['etf.Close'].mean()

std\_etf = prices\_train['etf.Close'].std()

…

XLK close price mean: 37.32 , std: 0.8224

Out[137]:

In [138]:

MSFT\_NORMAL\_PRICE\_COLUMN\_NUM = prices\_train.columns.get\_loc('stock.CloseNormal') # column pos

XLK\_NORMAL\_PRICE\_COLUMN\_NUM = prices\_train.columns.get\_loc('etf.CloseNormal')

# multi-variate input to RNN

model\_cols = [[MSFT\_NORMAL\_PRICE\_COLUMN\_NUM, XLK\_NORMAL\_PRICE\_COLUMN\_NUM]]

model\_cols[0]

Out[138]:

[25, 26]

**In [115]:**

**#**

**# Next batch generation -**

**# data: The original array of floating point data (normalized).**

**# batch\_size: The number of samples per batch. We use randomization to identify the start of each seq in the input data in order to generate 'batch\_size' sequences.**

**# n\_steps: The period, in timesteps, at which we sample data.**

**# n\_lag: How many timesteps in the future should our target be.**

**# NOTE: Multiple inputs being selected from dataframe (MSFT and XLK price)**

**#**

def next\_batch(data, batch\_size, n\_steps, n\_lag=1, debug=0):

t\_min, t\_max = 0, len(data)

t0 = np.random.randint(t\_min, t\_max - n\_lag - n\_steps, batch\_size)

Ts = np.array([np.arange(t, t+n\_steps + n\_lag) for t in t0])

ys = np.array([data.iloc[t, model\_cols[0]].values for t in Ts])

return ys[:, :-n\_lag, :], ys[:, n\_lag:, :1]

In [122]:

**# Set the data batch generation parameters**

**# Generate a sample instance and target for visualization purpose**

**#**

n\_days = 2 # window of contiguous trading days

n\_samples\_hour = 60 # number of samples per hour (one-min samples)

n\_samples\_day = 6.5 \* n\_samples\_hour # 09:30 to 16:00 EST trading hours (6.5 hours)

n\_steps = int(n\_days \* n\_samples\_day)

n\_lag = 60 # n\_steps to predict into future (60 samples = 1 hr)

batch\_size = 50

X\_batch, y\_batch=next\_batch(prices\_train, batch\_size, n\_steps, n\_lag)

**# Plot instance vs target values**

**# note that we cannot use DateTime column due to discontinuity of data (after market)**

x\_label=np.arange(0, n\_steps+n\_lag, 1).tolist()

plt.title("Sample training instance vs target (MSFT)", fontsize=14)

plt.plot(x\_label[:n\_steps], X\_batch[0, :, 0], "b\*", markersize=2, label="instance(MSFT)")

plt.plot(x\_label[:n\_steps], X\_batch[0, :, 1], "g\*", markersize=2, label="instance(XLK)")

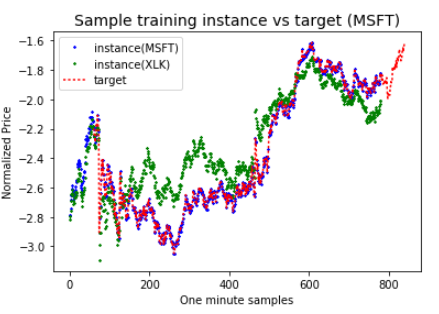
plt.plot(x\_label[n\_lag:n\_steps+n\_lag], y\_batch[0, :, 0], "r:", markersize=2, label="target")

plt.legend(loc="upper left")

plt.xlabel("One minute samples")

plt.ylabel("Normalized Price")

plt.show()



In [123]:

**#**

**# Define RNN model with multi-steps**

**#**

**# Model parameters:**

**# Learning rate = 0.001**

**# Cell type = RNN with OutputProjectionWrapper**

**# #Input steps = 1950 (Samples for 5 consecutive trading days - 5 \* 390)**

**# #Target steps = 1950**

**# #neurons/cell = 100**

**# Output activation: relu function**

**# Optimizer = Adam**

**# Loss function = MSE (mean squared error)**

**# Training #epochs = 800**

**# Batchsize = 50**

**#**

reset\_graph()

**# RNN time-step model parameters**

**n\_inputs = 2 # [MSFT, XLK] multi-variate**

n\_neurons = 100

n\_outputs = 1

**## Rest of the model similar to RNN single variate model with an dynamic RNN with OutputProjectionWrapper**

In [124]:

**# Training parameters**

learning\_rate = 0.001

n\_iterations = 800

**# rest of the code for loss, optimizer similar to RNN single variate model**

In [125]:

**#**

**# Training phase for multi time-step RNN**

**#**

**# We have a vector of time-series values, train to produce the desired output for 'n' steps into future.**

**# We intend to predict every hour for a one full day - trading window of 6.5 hours. This corresponds to a lag of 60, 120, 180, 240, 300, 360 steps into future.**

**#**

start\_time = time.time()

lag\_list = [60, 120, 180, 240, 300, 360] # predicting stock prices next trading day at each hour interval (upto 6hours)

with tf.Session() as sess:

init.run()

for lag in lag\_list:

print("Training for Lag= ", lag)

for iteration in range(n\_iterations):

X\_batch, y\_batch=next\_batch(prices\_train, batch\_size, n\_steps, lag)

sess.run(training\_op, feed\_dict={X: X\_batch, y: y\_batch})

if iteration % 100 == 0:

mse = loss.eval(feed\_dict={X: X\_batch, y: y\_batch})

print("\t", iteration, "\tMSE:", mse)

print("Saving model weights for lag: ", lag)

saver.save(sess, "./Proj\_MSFT\_multi\_variate\_2m\_"+str(n\_days)+"d\_"+str(lag)) # not shown in the book

elapsed\_time = time.time() - start\_time

print("Total training time: ", round(elapsed\_time, 2), "sec")

Training for Lag= 60

0 MSE: 0.6466852

100 MSE: 0.031829707

…

700 MSE: 0.025200555

Saving model weights for lag: 60

Training for Lag= 120

0 MSE: 0.04315259

100 MSE: 0.047666036

…

700 MSE: 0.04441566

Saving model weights for lag: 120

Training for Lag= 180

0 MSE: 0.07498647

100 MSE: 0.07220581

…

700 MSE: 0.06620909

Saving model weights for lag: 180

Training for Lag= 240

0 MSE: 0.09072445

100 MSE: 0.054418877

…

700 MSE: 0.0730878

Saving model weights for lag: 240

Training for Lag= 300

0 MSE: 0.13945991

100 MSE: 0.106082395

…

700 MSE: 0.06758645

Saving model weights for lag: 300

Training for Lag= 360

0 MSE: 0.08884402

100 MSE: 0.10465774

…

700 MSE: 0.13107844

Saving model weights for lag: 360

Total training time: 1002.12 sec

In [140]:

**#**

**# Predict multi time-steps into the future**

**# Lets predict stock prices for the entire test-period. This also includes predicting outside of training period into the near future. We predict 60 steps at a time.**

**#**

results = pd.DataFrame(columns=['n\_lag', 'hour\_idx', 'DateTime', 'loss(MSE)', 'Close(Actual)', 'Close(Pred)', 'loss\_normal'])

sequence = prices\_test['stock.CloseNormal'].tolist() # test input

lag\_list = [60, 120, 180, 240, 300, 360] # num of samples to predict into future

for hour\_idx, lag in enumerate(lag\_list):

with tf.Session() as sess:

print("Restore model with lag: ", lag)

saver.restore(sess, "./Proj\_MSFT\_multi\_variate\_2m\_"+str(n\_days)+"d\_"+str(lag))

**# Collect predictions over the entire test period that stretches beyond the training period**

pred\_values = sequence[:n\_steps] # initialize first n\_steps with original sequence values

cum\_mse = []

for i in range(n\_steps, len(sequence), lag):

X\_batch = np.array(prices\_test.iloc[i-n\_steps:i, model\_cols[0]]).reshape(1, n\_steps, 2) # pick n\_steps values from the sequence

y\_pred = sess.run(outputs, feed\_dict={X: X\_batch}) # predicted values are the last lag values

y\_pred\_values=y\_pred[0, -lag:, 0].tolist() # pick last lag values

pred\_values.extend([round(item, 6) for item in y\_pred\_values]) # accumulate predicted values

mse = tf.reduce\_mean(tf.square(y\_pred - X\_batch)) # calc MSE for test data

cum\_mse.append(mse.eval())

**# Add predicted prices to dataframe**

pred\_col\_name = 'ClosePred'+str(lag)

seq = pd.Series(pred\_values[:prices\_test.shape[0]])

prices\_test[pred\_col\_name] = (seq.values\*std) + mean

**# MSE calculation for pred values**

mse\_normalized = round(np.mean(cum\_mse), 4)

print('\tMSE(normalized): ', mse\_normalized)

mse\_price = round((mse\_normalized\*std + mean), 4)

print('\tMSE(prices):', mse\_price) # MSE in terms of stock price

**# append to results dataframe**

datetime\_lag = test\_pred\_start + pd.Timedelta(str(lag)+' min')

actual\_price\_lag = prices\_test[prices\_test['DateTime']==datetime\_lag]['Close'].values[0]

pred\_price\_lag = prices\_test[prices\_test['DateTime']==datetime\_lag][pred\_col\_name].values[0]

print('\t', datetime\_lag, 'Actual: ', actual\_price\_lag, 'Pred: ', pred\_price\_lag)

results.loc[hour\_idx] = [lag, hour\_idx+1, datetime\_lag, mse\_price, actual\_price\_lag, pred\_price\_lag, mse\_normalized]

Restore model with lag: 60

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_multi\_variate\_2m\_2d\_60

X-batch[ 0 : 780 ], MSE: 0.9774 , y-batch[ 60 : 840 ]

X-batch[ 60 : 840 ], MSE: 0.9693 , y-batch[ 120 : 900 ]

…

X-batch[ 16680 : 17460 ], MSE: 0.0249 , y-batch[ 16740 : 17520 ]

X-batch[ 16740 : 17520 ], MSE: 0.0231 , y-batch[ 16800 : 17580 ]

-------------------------------------------------------------------------------------

MSE(normalized): 0.0776

MSE(prices): 42.1931

2014-11-03 10:30:00 Actual: 43.5169 Pred: 43.15926295377438

Restore model with lag: 120

X-batch[ 0 : 780 ], MSE: 0.9609 , y-batch[ 60 : 900 ]

X-batch[ 120 : 900 ], MSE: 0.9313 , y-batch[ 180 : 1020 ]

…

X-batch[ 16560 : 17340 ], MSE: 0.0212 , y-batch[ 16620 : 17460 ]

X-batch[ 16680 : 17460 ], MSE: 0.0199 , y-batch[ 16740 : 17580 ]

-------------------------------------------------------------------------------------

MSE(normalized): 0.0739

MSE(prices): 42.1892

2014-11-03 11:30:00 Actual: 43.4709 Pred: 43.2021005131652

Restore model with lag: 180

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_multi\_variate\_2m\_2d\_180

X-batch[ 0 : 780 ], MSE: 0.9054 , y-batch[ 60 : 960 ]

…

X-batch[ 16560 : 17340 ], MSE: 0.0282 , y-batch[ 16620 : 17520 ]

X-batch[ 16740 : 17520 ], MSE: 0.0161 , y-batch[ 16800 : 17700 ]

-------------------------------------------------------------------------------------

MSE(normalized): 0.0816

MSE(prices): 42.1972

2014-11-03 12:30:00 Actual: 43.5353 Pred: 43.25490400062917

Restore model with lag: 240

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_multi\_variate\_2m\_2d\_240

X-batch[ 0 : 780 ], MSE: 0.903 , y-batch[ 60 : 1020 ]

…

X-batch[ 16320 : 17100 ], MSE: 0.0547 , y-batch[ 16380 : 17340 ]

X-batch[ 16560 : 17340 ], MSE: 0.0363 , y-batch[ 16620 : 17580 ]

--------------------------------------------------------------------------------------

MSE(normalized): 0.0814

MSE(prices): 42.197

2014-11-03 13:30:00 Actual: 43.6274 Pred: 43.29161372732638

Restore model with lag: 300

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_multi\_variate\_2m\_2d\_300

X-batch[ 0 : 780 ], MSE: 0.898 , y-batch[ 60 : 1080 ]

…

X-batch[ 16200 : 16980 ], MSE: 0.0729 , y-batch[ 16260 : 17280 ]

X-batch[ 16500 : 17280 ], MSE: 0.0492 , y-batch[ 16560 : 17580 ]

-------------------------------------------------------------------------------------

MSE(normalized): 0.1188

MSE(prices): 42.236

2014-11-03 14:30:00 Actual: 43.655 Pred: 43.418352093364604

Restore model with lag: 360

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_multi\_variate\_2m\_2d\_360

X-batch[ 0 : 780 ], MSE: 0.8922 , y-batch[ 60 : 1140 ]

…

X-batch[ 16560 : 17340 ], MSE: 0.0283 , y-batch[ 16620 : 17700 ]

--------------------------------------------------------------------------------------

MSE(normalized): 0.1089

MSE(prices): 42.2256

2014-11-03 15:30:00 Actual: 43.5906 Pred: 43.3413475714971

In [141]:

**# Predictions across different lags into future**

prices\_test[n\_steps:][['DateTime', 'Close', 'stock.CloseNormal', 'ClosePred60', 'ClosePred120', 'ClosePred180', 'ClosePred240', 'ClosePred300', 'ClosePred360']] # print

Out[141]:

|  | **Close** | **ClosePred60** | **ClosePred120** | **ClosePred180** | **ClosePred240** | **ClosePred300** | **ClosePred360** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DateTime** |  |  |  |  |  |  |  |
| **2014-09-03 15:59:00** | 41.4032 | 41.470833 | 41.543857 | 41.557897 | 41.469964 | 41.464019 | 41.398882 |
| **2014-09-03 16:00:00** | 41.4078 | 41.467795 | 41.537166 | 41.557683 | 41.471024 | 41.466222 | 41.403199 |
| **2014-09-04 09:30:00** | 41.3986 | 41.461100 | 41.528405 | 41.548245 | 41.466987 | 41.469330 | 41.410060 |
| **2014-09-04 09:31:00** | 41.3618 | 41.450116 | 41.525893 | 41.546375 | 41.461623 | 41.473308 | 41.411957 |

16747 rows × 9 columns

In [142]:

**# display predicted results at each hour, loss\_normal is the normalized loss values**

results

Out[142]:

|  | **n\_lag** | **hour\_idx** | **DateTime** | **loss(MSE)** | **Close(Actual)** | **Close(Pred)** | **loss\_normal** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 60.0 | 1.0 | 2014-11-03 10:30:00 | 42.1931 | 43.5169 | 43.159263 | 0.0776 |
| **1** | 120.0 | 2.0 | 2014-11-03 11:30:00 | 42.1892 | 43.4709 | 43.202101 | 0.0739 |
| **2** | 180.0 | 3.0 | 2014-11-03 12:30:00 | 42.1972 | 43.5353 | 43.254904 | 0.0816 |
| **3** | 240.0 | 4.0 | 2014-11-03 13:30:00 | 42.1970 | 43.6274 | 43.291614 | 0.0814 |
| **4** | 300.0 | 5.0 | 2014-11-03 14:30:00 | 42.2360 | 43.6550 | 43.418352 | 0.1188 |
| **5** | 360.0 | 6.0 | 2014-11-03 15:30:00 | 42.2256 | 43.5906 | 43.341348 | 0.1089 |

In [144]:

**# Plot instance vs target vs predicted values (for 60min into future)**

**# Print for an approximate time window of last 5 days of the month**

lag=60

start=len(prices\_train)-n\_steps # last 5 days of the month

x\_label=np.arange(0, len(prices\_test['Close']), 1).tolist()

plt.title("Sample instance vs target vs predicted seq (60 steps into the future)", fontsize=14)

plt.figure(figsize=(16,4), dpi=80)

plt.subplot(1,1,1)

plt.plot(x\_label[start-lag:-2\*lag], prices\_test['Close'][start-lag:-2\*lag], "y\*", markersize=4, label="instance")

plt.plot(x\_label[start:-lag], prices\_test['Close'][start:-lag], "g-", markersize=1, label="target")

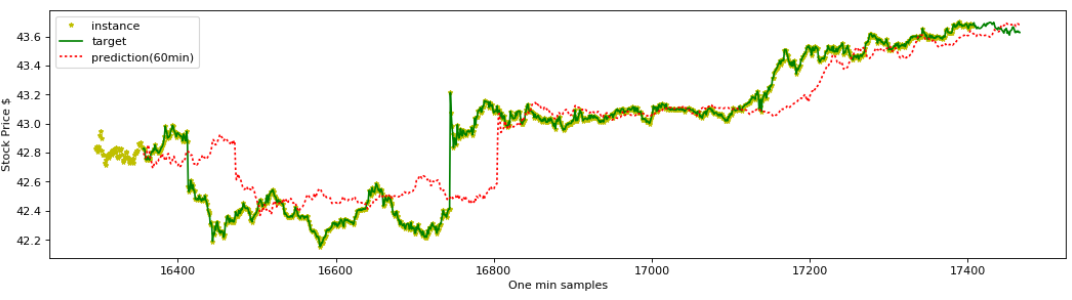
plt.plot(x\_label[start:-lag], prices\_test['ClosePred60'][start:-lag], "r:", markersize=2, label="prediction(60min)")

plt.legend(loc="upper left")

plt.xlabel("One min samples")

plt.ylabel("Stock Price $")

plt.show()



In [145]:

plt.figure(figsize=(16,6))

plt.subplot(111)

lag=450 # skip last batch of values across to have same time frame for all predictions.

prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='Close', color='green', ax=plt.gca()) # Actual close prices

prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred60', color='red', ax=plt.gca())

prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred120', color='blue', ax=plt.gca())

prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred180', color='cyan', ax=plt.gca())

prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred240', color='magenta', ax=plt.gca())

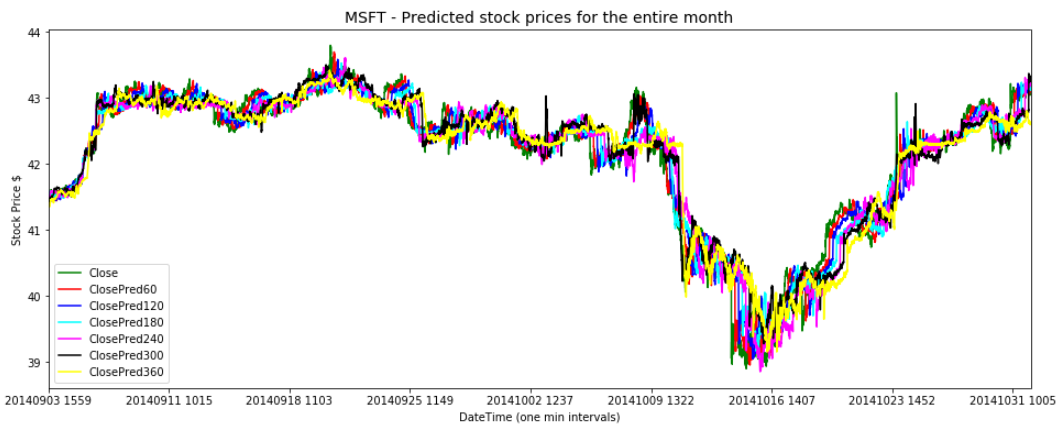
prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred300', color='black', ax=plt.gca())

prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred360', color='yellow', ax=plt.gca())

plt.title("MSFT - Predicted stock prices for the entire month", fontsize=14)

plt.xlabel("DateTime (one min intervals)")

plt.ylabel("Stock Price $")

plt.show()

## **OBSERVATION**:

1. We see that the predictions overall very tight and remains same regardless of prediction window size. The performance of multi-variate analysis is relatively better compared to the single-variate analysis in the previous task (Task#3). This is evident from the lower loss (MSE) values here. Part of the reason could be that we picked the best time-period in 2014 where MSFT prices positively correlated the most against XLK prices (corr=0.968).
2. We do clearly see that the predictions have picked-up for short term (60min window) and overall, picked up the long-term patterns.

## Task:4 Effect of XLK prices on MSFT price

### From our single-variate analysis on single stock, we learnt that model that predicted stock prices performed better compared to the model that predicted stock returns or up/down indicators. We will now perform a multi-variate price prediction for MSFT stock price (one-minute intervals) using the effects of tech sector ETF (XLK) prices. This experiment uses the same time-period (2014/01 to 2014/02) as the single-variate analysis in Task#3.

(see MSFT-RNN-single-variate.ipynb).

In [148]:

In [149]:

**#**

**# Normalize the MSFT stock price using mean/SD -**

**#**

mean = prices\_train['Close'].mean()

std = prices\_train['Close'].std()

print("MSFT close price mean:", round(mean, 2), ", std:", round(std, 4))

prices\_train['stock.CloseNormal'] = (prices\_train['Close'] - mean) / std

prices\_test['stock.CloseNormal'] = (prices\_test['Close'] - mean) / std

In [150]:

**# Normalize the XLK stock price using mean/SD (Similarly) -**

**## same as previous experiment ##**

XLK close price mean: 33.11 , std: 0.5761 Out[150]:

In [151]:

**# Define the same RNN model with multi-steps**

**## Same as previous experiment above ##**

In [153]:

**# Training phase for multi time-step RNN**

**## Same as previous code ##**

print("Total training time: ", round(elapsed\_time, 2), "sec")

Training for Lag= 60

0 MSE: 0.6006964

…

700 MSE: 0.03904175

Training for Lag= 120

0 MSE: 0.07170652

100 MSE: 0.06283994

…

Training for Lag= 360

0 MSE: 0.09124737

…

700 MSE: 0.05929109

Total training time: 989.6 sec

In [160]:

**# Display predicted results into the future (outside of training period)**

results

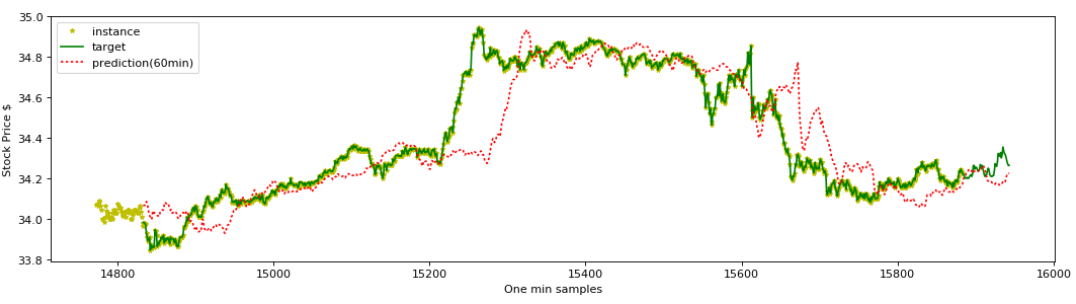
Out[160]:

|  | **n\_lag** | **hour\_idx** | **DateTime** | **loss(MSE)** | **Close(Actual)** | **Close(Pred)** | **loss\_normal** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 60.0 | 1.0 | 2014-03-03 10:30:00 | 33.5700 | 34.3095 | 34.685384 | 0.4398 |
| **1** | 120.0 | 2.0 | 2014-03-03 11:30:00 | 33.5805 | 34.1731 | 34.619664 | 0.4535 |
| **2** | 180.0 | 3.0 | 2014-03-03 12:30:00 | 33.5602 | 34.1731 | 34.660578 | 0.4270 |
| **3** | 240.0 | 4.0 | 2014-03-03 13:30:00 | 33.6010 | 34.2185 | 34.529850 | 0.4802 |
| **4** | 300.0 | 5.0 | 2014-03-03 14:30:00 | 33.6075 | 34.2186 | 34.418596 | 0.4887 |
| **5** | 360.0 | 6.0 | 2014-03-03 15:30:00 | 33.6187 | 34.2299 | 34.471258 | 0.5032 |

In [162]:

**# Plot instance vs target vs predicted values (for 60min into future)**

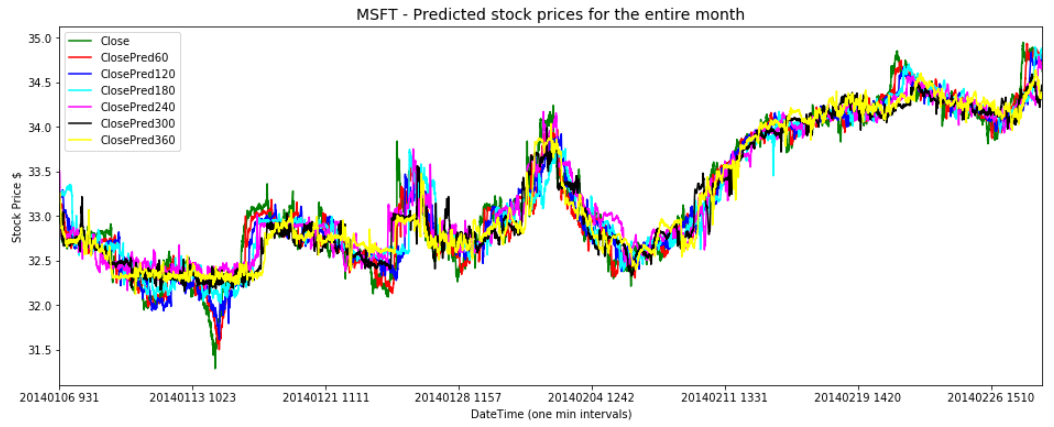
**### Same code as in previous experiment ###**



In [164]:

**# Plot predicted close prices for the entire test period for all lags**

**## Same code as in the previous experiment ##**



### **OBSERVATION**:

1. We can compare the performance of multi-variate price prediction (MSFT stock) here against the single-variate in the previous task (Task#3). It appears multi-variate prediction didn't improve much over single-variate as is evident from the loss (MSE) values. Though MSFT prices are modestly correlated (positiviely) w.r.t XLK stock(corr=0.631), appears it didn't provide enough boost to improve the accuracy of price prediction in the multi-variate scenario.
2. We predicted MSFT prices for different lags (60mins to 360mins) into the future.