# **RNN Single-Variate analysis**

# Load preprocessed MSFT stock data 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.

## [Task:1 Use Stock close price](#RNN_Stock_Price)

Using stock close price at one-min intervals, train the model to predict the stock prices on a short term hourly basis.

## [Task:2 Use Stock returns (% gain or loss)](#RNN_Stock_Returns)

Using stock returns at one-min intervals, train the model to predict the stock returns on a short term hourly basis.

## [Task:3 Use Stock Up/Down Indicator](#RNN_Stock_UpDown)

Using stock up/down indicator at one-min intervals, train the model to predict if stock will be up/down on a short term hourly basis.

## Task:1 Predict using stock close price.

In [6]:

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)

stockdata

|  | **Close** | **Volume** | **...** | **DateTime** | **CumReturn** | **etf.Close** | **etf.Return** | **etf.CumReturn** | **spy.Close** | **spy.CumReturn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 33.6842 | 1319510.0 | ... | 2014-01-02 09:30:00 | 0.000000 | 33.2164 | -0.196504 | 0.000000 | 170.524 | 0.000000 |
| **1** | 33.7157 | 206817.0 | ... | 2014-01-02 09:31:00 | 0.093516 | 33.2024 | -0.042148 | -0.042148 | 170.529 | 0.002932 |
| **2** | 33.6120 | 227216.0 | ... | 2014-01-02 09:32:00 | -0.214344 | 33.2071 | 0.000000 | -0.027998 | 170.589 | 0.038118 |

97701 rows × 25 columns

In [7]:

print('Column datatypes:\n', stockdata.dtypes)

**# Highlighted columns are augmented during pre-process stage.**

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

Column datatypes:

Date int64

Time int64

Open float64

High float64

Low float64

Close float64

Volume float64

SplitFactor int64

Earnings int64

Dividends float64

stock.Return float64

Gain float64

Loss float64

RSI float64

DateTimeStr object

DateTime object

stock.CumReturn float64

etf.Open float64

etf.Close float64

etf.Return float64

etf.CumReturn float64

spy.Open float64

spy.Close float64

spy.Return float64

spy.CumReturn float64

dtype: object

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

In [7]:

**# Data wrangling:**

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

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

stockdata['Volume'] = stockdata['Volume'].map(int)

**# convert Date/Time columns into native datetime type**

stockdata['DateTime'] = pd.to\_datetime(stockdata['DateTime'].map(str), format="%Y-%m-%d %H:%M:%S")

stockdata=stockdata.set\_index(pd.DatetimeIndex(stockdata['DateTime']))

In [8]:

**#**

**# set training and test periods. Test period stretches one day beyond training period.**

**#**

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

train\_lastdate='2014-03-01 00:00:00' # train end-date (2 months of trading data)

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

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

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

CLOSE\_PRICE\_COLUMN\_NUM = stockdata.columns.get\_loc('Close') # column pos

RETURN\_COLUMN\_NUM = stockdata.columns.get\_loc('stock.Return') # column pos

In [9]:

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

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

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

prices = stockdata.loc[stockdata['DateTime'] < train\_lastdate].values[:,CLOSE\_PRICE\_COLUMN\_NUM].astype(float) # training purpose

print()

tmp = stockdata.loc[stockdata['DateTime'] < test\_lastdate] # testing purpose

prices\_test = tmp.loc[tmp['DateTime'] > test\_startdate]

print(len(prices\_test['DateTime']))

prices\_test

train: [ 2014-01-01 00:00:00 : 2014-03-01 00:00:00 ]

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

16004

Out[9]:

|  | **Close** | **...** | **stock.CumReturn** | **etf.Close** | **etf.Return** | **etf.CumReturn** | **spy.Return** | **spy.CumReturn** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **DateTime** |  |  |  |  |  |  |  |  |
| **2014-01-02 09:30:00** | 33.6842 | ... | 0.000000 | 33.2164 | -0.196504 | 0.000000 | -0.048650 | 0.000000 |
| **2014-01-02 09:31:00** | 33.7157 | ... | 0.093516 | 33.2024 | -0.042148 | -0.042148 | 0.002932 | 0.002932 |

16004 rows × 25 columns

In [214]:

**# plot stock prices for different time periods just to understand the pattern**

**# plot the stock prices for the entire data**

plt.subplot(411)

stockdata.plot(x='DateTimeStr', y='Close', ax=plt.gca())

plt.xlabel("Date")

plt.ylabel("Close Price", horizontalalignment = 'right')

plt.title("MSFT Stock data (one year)")

plt.tight\_layout()

**# plot the prices for the test period (two months)**

…

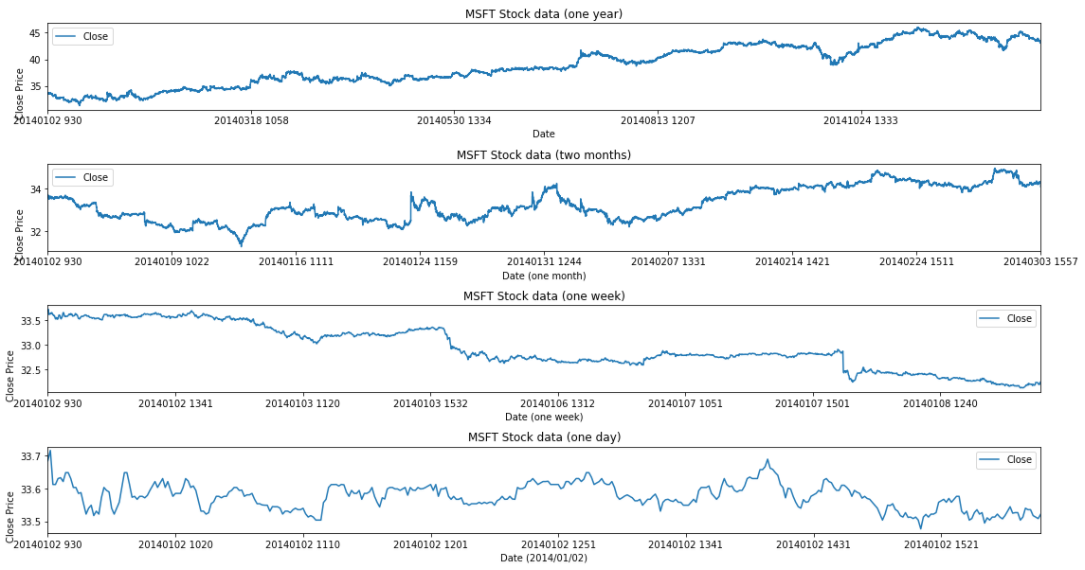
**# plot the prices for one week**

…

**# plot the price for one day**

…

plt.show()



#### OBSERVATION: From the plot, it is clear that stock price varies widely regardless of how long the time period. In reality, there are numerous factors that affect stock price movement including company's performance, sector overall performance, macro-economic trends, geo-political events etc.

In [12]:

**#**

**# Next batch generation -**

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

**# batch\_size: The number of samples per batch. We use randomization to identiy 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.**

**#**

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\_steps - n\_lag, batch\_size)

Ts = np.array([np.arange(t, t + n\_steps + n\_lag) for t in t0]) # batch\_size x (n\_steps+n\_lag) matrix of indices into data vector

ys = np.array([data[t] for t in Ts])

return ys[:, :-n\_lag].reshape(-1, n\_steps, 1), ys[:, n\_lag:].reshape(-1, n\_steps, 1)

In [13]:

**#**

**# Normalize the stock prices using mean/SD -**

**#**

prices\_normal = prices.tolist()

mean = np.mean(prices)

std = np.std(prices)

print('mean:', round(mean, 5), ', std:', round(std, 5))

prices\_normal -= mean

prices\_normal /= std

mean: 33.23274 , std: 0.76691

Train data size: 15613

In [14]:

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

In [15]:

**# Set the data batch generation parameters**

n\_days = 2 **# window of contiguous trading days**

n\_samples\_hour = 60 **# number of samples per hour (input: 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 hour)

batch\_size = 50

In [216]:

**#**

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

**#**

X\_batch, y\_batch=next\_batch(prices\_normal, 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")

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

plt.legend(loc="upper right")

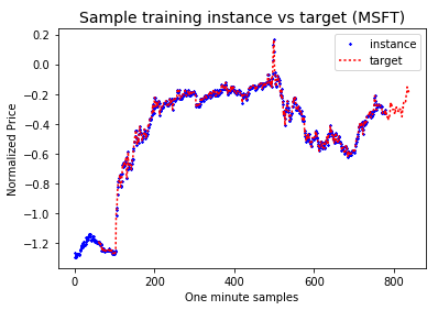
plt.xlabel("One minute samples")

plt.ylabel("Normalized Price")

plt.show()

X\_batch shape: (50, 780, 1)

y\_batch shape: (50, 780, 1)



In [210]:

**#**

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

**#**

**# Model parameters:**

**# Learning rate = 0.001**

**# Cell type = RNN with OutputProjectionWrapper**

**# #Input steps = 780 (Samples for 2 consecutive trading days - 2 \* 390 = 780)**

**# #Target steps = 780**

**# #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 = 1

n\_neurons = 100

n\_outputs = 1

**# define RNN cell with output projection to get single value out of #n\_neurons by having a FC layer**

X = tf.placeholder(tf.float32, [None, n\_steps, n\_inputs])

y = tf.placeholder(tf.float32, [None, n\_steps, n\_outputs])

**# add FC layer to convert output vector of size 100 into one output that corresponds to time\_series(t)**

cell = tf.contrib.rnn.OutputProjectionWrapper(

tf.contrib.rnn.BasicRNNCell(num\_units=n\_neurons, activation=tf.nn.relu),

output\_size=n\_outputs)

outputs, states = tf.nn.dynamic\_rnn(cell, X, dtype=tf.float32)

In [211]:

**# Training parameters**

learning\_rate = 0.001

n\_iterations = 800

loss = tf.reduce\_mean(tf.square(outputs - y)) # Loss function: MSE

optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate)

training\_op = optimizer.minimize(loss)

init = tf.global\_variables\_initializer()

saver = tf.train.Saver()

In [212]:

**#**

**# 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.**

**# y\_batch[t] = X\_batch[t+n\_lag]**

**#**

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\_normal, 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\_time\_series\_1m\_"+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: 0.7363748

100 MSE: 0.05171514

…

700 MSE: 0.047164723

Saving model weights for lag: 60

Training for Lag= 120

0 MSE: 0.09848139

…

700 MSE: 0.08530672

Saving model weights for lag: 120

Training for Lag= 180

0 MSE: 0.13831215

…

700 MSE: 0.13906056

Saving model weights for lag: 180

Training for Lag= 240

0 MSE: 0.184146

…

700 MSE: 0.18146883

Saving model weights for lag: 240

Training for Lag= 300

0 MSE: 0.22791417

…

600 MSE: 0.15030272

700 MSE: 0.16450556

Saving model weights for lag: 300

Training for Lag= 360

0 MSE: 0.24133901

…

700 MSE: 0.27187183

Saving model weights for lag: 360

Total training time: 818.34 sec

In [135]:

**#**

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

**# Lets predict stock prices for the last five days in the input sequence. 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['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\_time\_series\_1m\_"+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(sequence[i-n\_steps:i]).reshape(1, n\_steps, 1) # 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())

print('\tX-batch[', i-n\_steps, ':', i, '], MSE: ', round(mse.eval(), 4), ', y-batch[', i-n\_steps+n\_lag, ':', len(pred\_values), ']')

# 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]

#Test Prices(initial): 16004

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

Restore model with lag: 60

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_time\_series\_1m\_2d\_60

X-batch[ 0 : 780 ], MSE: 0.0028 , y-batch[ 60 : 840 ]

X-batch[ 60 : 840 ], MSE: 0.0028 , y-batch[ 120 : 900 ]

X-batch[ 120 : 900 ], MSE: 0.0028 , y-batch[ 180 : 960 ]

…

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

MSE(normalized): 0.0012

MSE(prices): 33.2337

2014-03-03 10:30:00 Actual: 34.3095 Pred: 34.622044485870376

Restore model with lag: 120

INFO:tensorflow:Restoring parameters from ./Proj\_MSFT\_time\_series\_1m\_2d\_120

X-batch[ 0 : 780 ], MSE: 0.0408 , y-batch[ 60 : 900 ]

X-batch[ 120 : 900 ], MSE: 0.0439 , y-batch[ 180 : 1020 ]

…

X-batch[ 15120 : 15900 ], MSE: 0.0026 , y-batch[ 15180 : 16020 ]

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

MSE(normalized): 0.0062

MSE(prices): 33.2375

2014-03-03 11:30:00 Actual: 34.1731 Pred: 34.616619395815015

Restore model with lag: 180

X-batch[ 0 : 780 ], MSE: 0.0075 , y-batch[ 60 : 960 ]

X-batch[ 180 : 960 ], MSE: 0.0058 , y-batch[ 240 : 1140 ]

…

X-batch[ 15120 : 15900 ], MSE: 0.0136 , y-batch[ 15180 : 16080 ]

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

MSE(normalized): 0.0118

MSE(prices): 33.2418

2014-03-03 12:30:00 Actual: 34.1731 Pred: 34.558905916578894

Restore model with lag: 240

X-batch[ 0 : 780 ], MSE: 0.0284 , y-batch[ 60 : 1020 ]

X-batch[ 240 : 1020 ], MSE: 0.0264 , y-batch[ 300 : 1260 ]

…

X-batch[ 15120 : 15900 ], MSE: 0.0357 , y-batch[ 15180 : 16140 ]

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

MSE(normalized): 0.0232

MSE(prices): 33.2505

2014-03-03 13:30:00 Actual: 34.2185 Pred: 34.505929613095105

Restore model with lag: 300

X-batch[ 0 : 780 ], MSE: 0.0384 , y-batch[ 60 : 1080 ]

X-batch[ 300 : 1080 ], MSE: 0.0372 , y-batch[ 360 : 1380 ]

…

X-batch[ 15000 : 15780 ], MSE: 0.0145 , y-batch[ 15060 : 16080 ]

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

MSE(normalized): 0.0394

MSE(prices): 33.263

2014-03-03 14:30:00 Actual: 34.2186 Pred: 34.561902216670255

Restore model with lag: 360

X-batch[ 0 : 780 ], MSE: 0.051 , y-batch[ 60 : 1140 ]

X-batch[ 360 : 1140 ], MSE: 0.072 , y-batch[ 420 : 1500 ]

…

X-batch[ 15120 : 15900 ], MSE: 0.0381 , y-batch[ 15180 : 16260 ]

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

#Test Prices: 16004

#Predicted values: 16260

MSE(normalized): 0.0477

MSE(prices): 33.2693

2014-03-03 15:30:00 Actual: 34.2299 Pred: 34.473164339201226

In [136]:

**# display predicted results at each hour (outside of training period)**

results

Out[136]:

|  | **n\_lag** | **hour\_idx** | **DateTime** | **loss(MSE)** | **Close(Actual)** | **Close(Pred)** | **loss\_normal** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 60.0 | 1.0 | 2014-03-03 10:30:00 | 33.2337 | 34.3095 | 34.622044 | 0.0012 |
| **1** | 120.0 | 2.0 | 2014-03-03 11:30:00 | 33.2375 | 34.1731 | 34.616619 | 0.0062 |
| **2** | 180.0 | 3.0 | 2014-03-03 12:30:00 | 33.2418 | 34.1731 | 34.558906 | 0.0118 |
| **3** | 240.0 | 4.0 | 2014-03-03 13:30:00 | 33.2505 | 34.2185 | 34.505930 | 0.0232 |
| **4** | 300.0 | 5.0 | 2014-03-03 14:30:00 | 33.2630 | 34.2186 | 34.561902 | 0.0394 |
| **5** | 360.0 | 6.0 | 2014-03-03 15:30:00 | 33.2693 | 34.2299 | 34.473164 | 0.0477 |

In [222]:

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

lag=60

start=len(prices\_normal)-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.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")

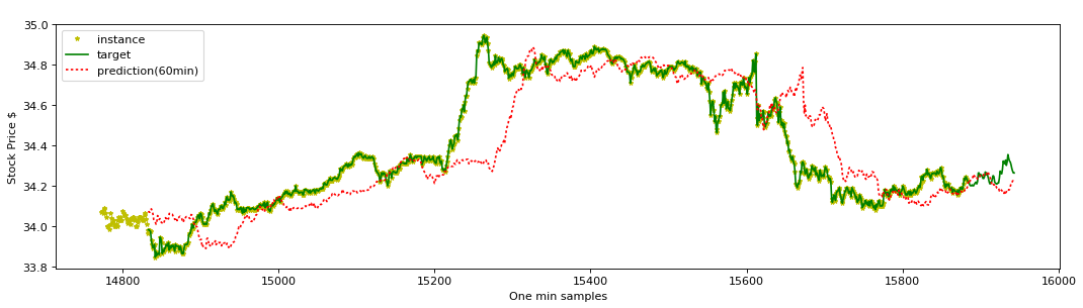
plt.plot(x\_label[start:-lag], prices\_test['ClosePred60'][start:-lag], "r:", markersize=2, label="prediction(60min)")

plt.legend(loc="upper left")

plt.xlabel("One min samples")

plt.ylabel("Stock Price $")

plt.show()



In [224]:

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())

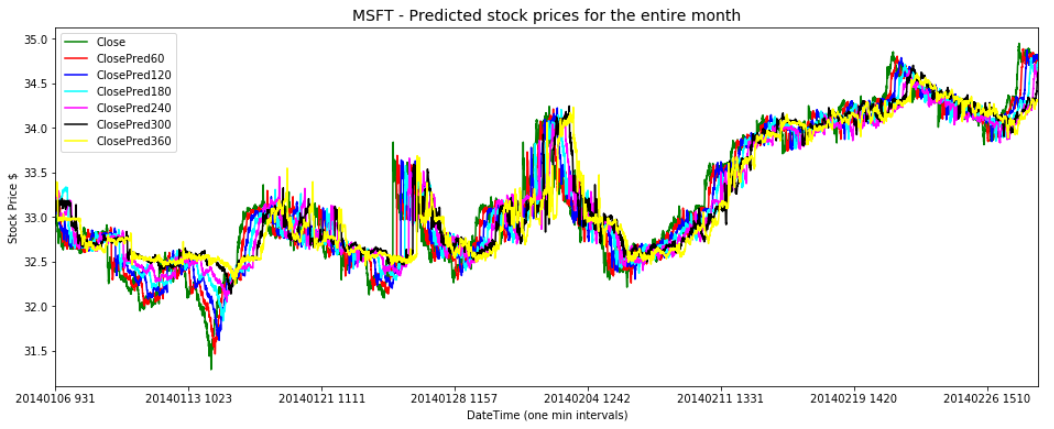
prices\_test.iloc[n\_steps:-lag].plot(x='DateTimeStr', y='ClosePred360', color='yellow', ax=plt.gca())

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 here are much worse as the prediction window grows. We do see that the predictions have picked-up for short term (60min window) and overall, picked up the long-term patterns.
2. We are using #steps equivalent to 2 days worth of trading data. From the experiment with XLK, we have observed that there is not much improvement with increasing #steps to 5 days worth of trading data (see XLK-RLK-single-variate-5days.ipynb).

## **Task:2 Predict Stock returns (% gain or loss).**

#### Using stock returns, train the model using 2 month data to predict the stock returns on a short term basis. We will predict multiple values on an hourly basis outside the training period.

**In [183]:**

**#**

**# Use one-min returns to train the model**

**#**

returns = stockdata.loc[stockdata['DateTime'] < train\_lastdate].values[:,RETURN\_COLUMN\_NUM].astype(float) # training purpose

returns\_normal = returns.tolist()

mean\_ret = np.mean(returns)

std\_ret = np.std(returns)

print('mean:', round(mean\_ret, 5), ', std:', round(std\_ret, 5))

returns\_normal -= mean\_ret

returns\_normal /= std\_ret

prices\_test['stock.ReturnNormal'] = (prices\_test['stock.Return'] - mean\_ret)/std\_ret

prices\_test[['stock.Return', 'stock.ReturnNormal']]

mean: -0.00043 , std: 0.05971

Out[183]:

|  | **stock.Return** | **stock.ReturnNormal** |
| --- | --- | --- |
| **DateTime** |  |  |
| **2014-01-02 09:30:00** | -0.026712 | -0.440180 |
| **2014-01-02 09:31:00** | 0.066779 | 1.125690 |
| **2014-01-02 09:32:00** | -0.294264 | -4.921403 |
| **2014-01-02 09:33:00** | 0.000000 | 0.007212 |
| **2014-03-03 16:00:00** | -0.741189 | -12.406929 |

16004 rows × 2 columns

In [64]:

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

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

In [70]:

Training for Lag= 60

0 MSE: 0.9495059

100 MSE: 0.9489427

…

700 MSE: 0.62334543

Saving model weights for lag: 60

Training for Lag= 120

0 MSE: 1.3654504

100 MSE: 0.9759403

…

700 MSE: 0.81444603

Saving model weights for lag: 120

Training for Lag= 180

0 MSE: 1.1427897

…

700 MSE: 0.7382939

Saving model weights for lag: 180

Training for Lag= 240

0 MSE: 1.3111365

100 MSE: 0.9471651

…

700 MSE: 0.70238394

Saving model weights for lag: 240

Training for Lag= 300

0 MSE: 1.0586054

100 MSE: 1.0340484

…

700 MSE: 0.678944

Saving model weights for lag: 300

Training for Lag= 360

0 MSE: 1.1575825

100 MSE: 0.89751625

…

700 MSE: 0.6237951

Saving model weights for lag: 360

Total training time: 768.77 sec

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

results

Out[158]:

|  | **n\_lag** | **hour\_idx** | **DateTime** | **loss(MSE)** | **Return(Actual)** | **Return(Pred)** | **loss\_normal** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 60.0 | 1.0 | 2014-03-03 10:30:00 | 0.0817 | -0.013114 | -0.099755 | 1.3755 |
| **1** | 120.0 | 2.0 | 2014-03-03 11:30:00 | 0.0690 | 0.026636 | 0.064145 | 1.1628 |
| **2** | 180.0 | 3.0 | 2014-03-03 12:30:00 | 0.0749 | 0.000000 | 0.004021 | 1.2622 |
| **3** | 240.0 | 4.0 | 2014-03-03 13:30:00 | 0.0773 | -0.079426 | -0.010672 | 1.3025 |
| **4** | 300.0 | 5.0 | 2014-03-03 14:30:00 | 0.0795 | 0.053216 | -0.075354 | 1.3391 |
| **5** | 360.0 | 6.0 | 2014-03-03 15:30:00 | 0.0824 | -0.099521 | -0.034892 | 1.3875 |

In [92]:

prices\_test[n\_steps:][['DateTime', 'stock.Return', 'stock.ReturnNormal', 'ReturnPred60', 'ReturnPred120', 'ReturnPred180', 'ReturnPred240', 'ReturnPred300', 'ReturnPred360']] # print

Out[92]:

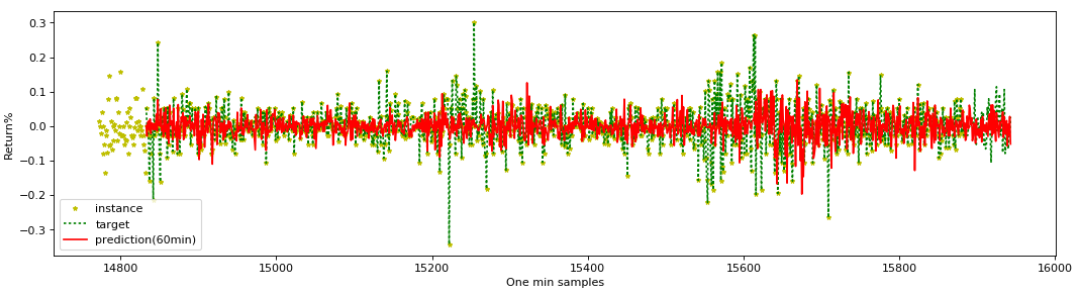
|  | **stock.Return** | **ReturnPred60** | **ReturnPred120** | **ReturnPred180** | **ReturnPred240** | **ReturnPred300** | **ReturnPred360** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **DateTime** |  |  |  |  |  |  |  |
| **2014-01-06 09:31:00** | -0.083927 | -0.013503 | -0.011847 | -0.023297 | 0.004405 | 0.004891 | -0.012725 |
| **2014-01-06 09:32:00** | -0.027401 | -0.009089 | 0.011176 | 0.009483 | 0.007524 | 0.016508 | -0.017545 |
| **2014-01-06 09:33:00** | 0.027103 | 0.000874 | -0.017829 | -0.019089 | 0.033005 | 0.020141 | -0.026737 |

15224 rows × 9 columns

In [220]:

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

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



### **OBSERVATION:**

1. The prediction of "stock" one-min returns is worse than prediction of stock prices, as seen in the previous experiment. Normalized loss as measured by MSE of returns (see results table above) is also worse.

## **Task:3 Predict Stock Up/Down Indicator.**

#### 1. We will predict on a logistic basis whether stock prices will be up or down using the 2 month training data. We will predict multiple times on an hourly basis outside the training period.

#### 2. We already calculated single minute returns during pre-processing stage. We will simply use it to decide if stock went up (return >= 0) or down (return < 0).

In [10]:

**# add stock up or down indicator column**

prices\_test['stock.UpDownIndicator']=(prices\_test['stock.Return']>=0).astype(int)

updown = prices\_test['stock.UpDownIndicator'].values

prices\_test[['stock.Return', 'stock.UpDownIndicator']]

Out[10]:

|  | **stock.Return** | **stock.UpDownIndicator** |
| --- | --- | --- |
| **DateTime** |  |  |
| **2014-01-02 09:30:00** | -0.026712 | 0 |
| **2014-01-02 09:31:00** | 0.066779 | 1 |
| **2014-01-02 09:32:00** | -0.294264 | 0 |
| **...** | ... | ... |
| **2014-03-03 15:31:00** | 0.045273 | 1 |
| **2014-03-03 16:00:00** | -0.741189 | 0 |

16004 rows × 2 columns

In [16]:

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

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

In [193]:

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

Training for Lag= 60

0 MSE: 0.53467673

100 MSE: 0.24599327

…

700 MSE: 0.22243455

Saving model weights for lag: 60

Training for Lag= 120

0 MSE: 0.2665217

100 MSE: 0.2419027

…

700 MSE: 0.21621259

Saving model weights for lag: 120

Training for Lag= 180

0 MSE: 0.27192664

100 MSE: 0.24415417

…

700 MSE: 0.2273739

Saving model weights for lag: 180

Training for Lag= 240

0 MSE: 0.25982863

100 MSE: 0.24284238

…

700 MSE: 0.21824239

Saving model weights for lag: 240

Training for Lag= 300

0 MSE: 0.27259532

100 MSE: 0.24348618

…

700 MSE: 0.22282723

Saving model weights for lag: 300

Training for Lag= 360

0 MSE: 0.26766855

…

700 MSE: 0.2264033

Saving model weights for lag: 360

Total training time: 908.72 sec

In [205]:

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

**# Lets predict stock up/down for the entire test period and into the future.**

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

In [207]:

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

results

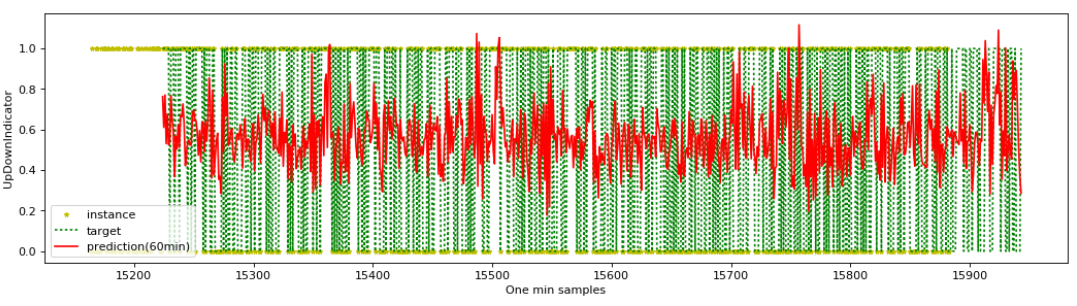
Out[207]:

|  | **n\_lag** | **hour\_idx** | **DateTime** | **UpDown(Actual)** | **UpDown(Pred)** | **loss\_normal** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 60.0 | 1.0 | 2014-03-03 10:30:00 | 0.0 | 0.575 | 0.2651 |
| **1** | 120.0 | 2.0 | 2014-03-03 11:30:00 | 1.0 | 0.796 | 0.2711 |
| **2** | 180.0 | 3.0 | 2014-03-03 12:30:00 | 1.0 | 0.599 | 0.2617 |
| **3** | 240.0 | 4.0 | 2014-03-03 13:30:00 | 0.0 | 0.697 | 0.2725 |
| **4** | 300.0 | 5.0 | 2014-03-03 14:30:00 | 1.0 | 0.644 | 0.2747 |
| **5** | 360.0 | 6.0 | 2014-03-03 15:30:00 | 0.0 | 0.676 | 0.2691 |

In [218]:

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

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



### **OBSERVATION:**

#### Model performed reasonably well at predicting stock prices going up or down at min-intervals. The loss function as measured by LSE is better compared to predicting actual stock returns (%). Rounding-off predictions to the nearest value (0 or 1) would have given even better results with a lower MSE loss.