Final Project of Math 9880

Shirong Zhao

July 16, 2019

1 Introduction

In this project, I use Long-Short-Term-Memory(LSTM) model(one of RNN model) to predict the close price of Alibaba(BABA) at current time t, given the previous prices and trading volumes.

1.1 Methods

Specifically, this project consists of two parts.

In the first part, I use Univariate Time Series Forecasting with LSTMs in Keras to predict the current price of Alibaba, that is to say, I only use the price of Alibaba at $t-1$ period to predict the price of Alibaba at t period. In total, I have 5 different versions based on the different specifications of the LSTM model.

In the second part, I use Multivariate Time Series Forecasting with LSTMs in Keras to predict the current price of Alibaba at t period, to be more specific, I use the previous prices of Alibaba and the previous trading volumes of Alibaba to predict the price of Alibaba at t period. In total, I have 5 different versions based on the different specifications of the LSTM model and the different choice of input data.

1.2 Data

I use the daily close prices and volumes of Alibaba company from 04/25/2013 to 04/25/2018. This data is available in Yahoo Finance Website. The number of total observations is 905. I use the first 80% of the data, in total 723 observations, as training data, and the remaining data, in total 181 observations, as test data.

1.3 Conclusion

Even though I tried different specifications, the outcome is not so different in terms of root squared mean errors of training and test data. Hence here I only report the outcomes of the baseline model for each part. And for the remaining versions, I only attach the codes.

For the baseline models of each part, we can see that my model performs very well(Check Section 2). However, I have to admit that, this project is primary. I should have tried more different specifications of LSTM models, and carefully choose the input data. For example, I should have choose fundamental factors (such as earnings per share, P/E and so on) of Alibaba and some factors reflecting the macroeconomic conditions(such as GDP, CPI). Due to time constraint, I will leave it in future.

Please check the following part for more details about the baseline models of Univariate Time Series Forecasting and Multivariate Time Series Forecasting.

2 Univariate Time Series Forecasting with LSTMs in Keras

2.1 Baseline

In [1]:
```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

```
Using TensorFlow backend.
```

In [2]:
```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        n_vars = 1 if type(data) is list else data.shape[1]
        df = DataFrame(data)
        cols, names = list(), list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
                if i == 0:
                        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_
vars)]
        # put it all together
        agg = concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg
```

In [3]:
```
# now let get all the information for stock
stock =  read_csv('BABA20130425_20180425.csv', header=0)
print(stock.shape)
print(stock.head())
```

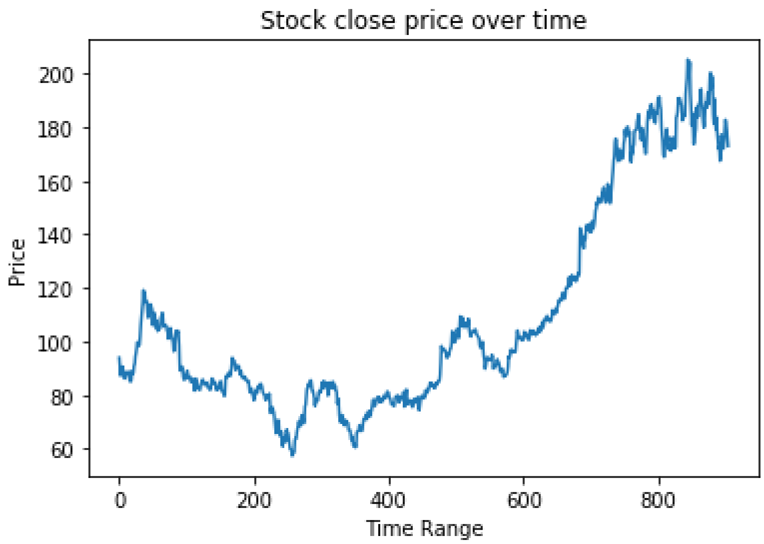
```
(905, 7)
         Date       Open       High        Low      Close  Adj Close  \
0  2014-09-19  92.699997  99.699997  89.949997  93.889999  93.889999
1  2014-09-22  92.699997  92.949997  89.500000  89.889999  89.889999
2  2014-09-23  88.940002  90.480003  86.620003  87.169998  87.169998
3  2014-09-24  88.470001  90.570000  87.220001  90.570000  90.570000
4  2014-09-25  91.089996  91.500000  88.500000  88.919998  88.919998

      Volume
0  271879400
1   66657800
2   39009800
3   32088000
4   28598000
```

In [4]:
```
# now get the stock close price
# astype means "Copy of the array, cast to a specified type."
stock_prices = stock.Close.values.astype("float32")
shape0=stock_prices.shape[0]
stock_prices = stock_prices.reshape(shape0, 1)
print(stock_prices.shape)
# print the prices of last five observations
print(stock_prices[-5:])
```

```
(905, 1)
[[ 182.67999268]
 [ 181.38999939]
 [ 179.11000061]
 [ 175.57000732]
 [ 173.08999634]]
```

In [5]:
```
# Before doing any analysis, first plot the prices series(data)
pyplot.plot(stock_prices)
pyplot.title('Stock close price over time')
pyplot.ylabel('Price')
pyplot.xlabel('Time Range')
pyplot.show()
```



In [8]:
```
values = stock_prices
```

In [9]:
```
# check the last five observations to make sure it's correct
print(values[-5:, :])
```

```
[[ 182.67999268]
 [ 181.38999939]
 [ 179.11000061]
 [ 175.57000732]
 [ 173.08999634]]
```

In [10]:
```
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
print(reframed.head())
# var1 means prices
```

```
   var1(t-1)   var1(t)
1   0.246905  0.219847
2   0.219847  0.201448
3   0.201448  0.224447
4   0.224447  0.213286
5   0.213286  0.223703
```

In [11]:
```
# split into train and test sets
values = reframed.values
n_train = int(reframed.shape[0] * 0.8)
train = values[:n_train, :]
test = values[n_train:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(723, 1, 1) (723,) (181, 1, 1) (181,)
```

In [12]:
```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=300, batch_size=72, validation_da
ta=(test_X, test_y), verbose=2, shuffle=False)
```

```
Train on 723 samples, validate on 181 samples
Epoch 1/300
0s - loss: 0.2208 - val_loss: 0.7733
Epoch 2/300
0s - loss: 0.1814 - val_loss: 0.7129
Epoch 3/300
0s - loss: 0.1427 - val_loss: 0.6496
Epoch 4/300
0s - loss: 0.1081 - val_loss: 0.5843
Epoch 5/300
0s - loss: 0.0864 - val_loss: 0.5287
Epoch 6/300
0s - loss: 0.0815 - val_loss: 0.4941
Epoch 7/300
0s - loss: 0.0815 - val_loss: 0.4732
Epoch 8/300
0s - loss: 0.0811 - val_loss: 0.4587
Epoch 9/300
0s - loss: 0.0799 - val_loss: 0.4473
Epoch 10/300
0s - loss: 0.0785 - val_loss: 0.4367
Epoch 11/300
0s - loss: 0.0769 - val_loss: 0.4264
Epoch 12/300
0s - loss: 0.0752 - val_loss: 0.4159
Epoch 13/300
0s - loss: 0.0736 - val_loss: 0.4052
Epoch 14/300
0s - loss: 0.0719 - val_loss: 0.3942
Epoch 15/300
0s - loss: 0.0702 - val_loss: 0.3829
Epoch 16/300
0s - loss: 0.0685 - val_loss: 0.3714
Epoch 17/300
0s - loss: 0.0667 - val_loss: 0.3596
Epoch 18/300
0s - loss: 0.0649 - val_loss: 0.3474
Epoch 19/300
0s - loss: 0.0631 - val_loss: 0.3348
Epoch 20/300
0s - loss: 0.0612 - val_loss: 0.3218
Epoch 21/300
0s - loss: 0.0593 - val_loss: 0.3083
Epoch 22/300
0s - loss: 0.0574 - val_loss: 0.2943
Epoch 23/300
0s - loss: 0.0554 - val_loss: 0.2804
Epoch 24/300
0s - loss: 0.0533 - val_loss: 0.2663
Epoch 25/300
0s - loss: 0.0511 - val_loss: 0.2521
Epoch 26/300
0s - loss: 0.0488 - val_loss: 0.2374
Epoch 27/300
0s - loss: 0.0464 - val_loss: 0.2220
Epoch 28/300
0s - loss: 0.0439 - val_loss: 0.2054
```

```
Epoch 29/300
0s - loss: 0.0414 - val_loss: 0.1880
Epoch 30/300
0s - loss: 0.0389 - val_loss: 0.1703
Epoch 31/300
0s - loss: 0.0362 - val_loss: 0.1523
Epoch 32/300
0s - loss: 0.0335 - val_loss: 0.1339
Epoch 33/300
0s - loss: 0.0306 - val_loss: 0.1148
Epoch 34/300
0s - loss: 0.0276 - val_loss: 0.0952
Epoch 35/300
0s - loss: 0.0244 - val_loss: 0.0744
Epoch 36/300
0s - loss: 0.0214 - val_loss: 0.0543
Epoch 37/300
0s - loss: 0.0181 - val_loss: 0.0345
Epoch 38/300
0s - loss: 0.0151 - val_loss: 0.0225
Epoch 39/300
0s - loss: 0.0125 - val_loss: 0.0208
Epoch 40/300
0s - loss: 0.0106 - val_loss: 0.0261
Epoch 41/300
0s - loss: 0.0099 - val_loss: 0.0299
Epoch 42/300
0s - loss: 0.0094 - val_loss: 0.0321
Epoch 43/300
0s - loss: 0.0097 - val_loss: 0.0308
Epoch 44/300
0s - loss: 0.0096 - val_loss: 0.0313
Epoch 45/300
0s - loss: 0.0095 - val_loss: 0.0315
Epoch 46/300
0s - loss: 0.0094 - val_loss: 0.0310
Epoch 47/300
0s - loss: 0.0097 - val_loss: 0.0292
Epoch 48/300
0s - loss: 0.0096 - val_loss: 0.0295
Epoch 49/300
0s - loss: 0.0096 - val_loss: 0.0302
Epoch 50/300
0s - loss: 0.0095 - val_loss: 0.0310
Epoch 51/300
0s - loss: 0.0094 - val_loss: 0.0319
Epoch 52/300
0s - loss: 0.0094 - val_loss: 0.0308
Epoch 53/300
0s - loss: 0.0096 - val_loss: 0.0296
Epoch 54/300
0s - loss: 0.0097 - val_loss: 0.0291
Epoch 55/300
0s - loss: 0.0095 - val_loss: 0.0300
Epoch 56/300
0s - loss: 0.0095 - val_loss: 0.0302
Epoch 57/300
```

```
0s - loss: 0.0094 - val_loss: 0.0312
Epoch 58/300
0s - loss: 0.0094 - val_loss: 0.0311
Epoch 59/300
0s - loss: 0.0093 - val_loss: 0.0304
Epoch 60/300
0s - loss: 0.0096 - val_loss: 0.0287
Epoch 61/300
0s - loss: 0.0096 - val_loss: 0.0280
Epoch 62/300
0s - loss: 0.0094 - val_loss: 0.0293
Epoch 63/300
0s - loss: 0.0095 - val_loss: 0.0289
Epoch 64/300
0s - loss: 0.0093 - val_loss: 0.0303
Epoch 65/300
0s - loss: 0.0094 - val_loss: 0.0299
Epoch 66/300
0s - loss: 0.0093 - val_loss: 0.0308
Epoch 67/300
0s - loss: 0.0093 - val_loss: 0.0303
Epoch 68/300
0s - loss: 0.0093 - val_loss: 0.0291
Epoch 69/300
0s - loss: 0.0096 - val_loss: 0.0277
Epoch 70/300
0s - loss: 0.0097 - val_loss: 0.0266
Epoch 71/300
0s - loss: 0.0094 - val_loss: 0.0283
Epoch 72/300
0s - loss: 0.0095 - val_loss: 0.0278
Epoch 73/300
0s - loss: 0.0093 - val_loss: 0.0295
Epoch 74/300
0s - loss: 0.0094 - val_loss: 0.0290
Epoch 75/300
0s - loss: 0.0092 - val_loss: 0.0298
Epoch 76/300
0s - loss: 0.0093 - val_loss: 0.0294
Epoch 77/300
0s - loss: 0.0092 - val_loss: 0.0296
Epoch 78/300
0s - loss: 0.0092 - val_loss: 0.0293
Epoch 79/300
0s - loss: 0.0092 - val_loss: 0.0291
Epoch 80/300
0s - loss: 0.0092 - val_loss: 0.0289
Epoch 81/300
0s - loss: 0.0092 - val_loss: 0.0286
Epoch 82/300
0s - loss: 0.0092 - val_loss: 0.0285
Epoch 83/300
0s - loss: 0.0092 - val_loss: 0.0284
Epoch 84/300
0s - loss: 0.0092 - val_loss: 0.0282
Epoch 85/300
0s - loss: 0.0092 - val_loss: 0.0280
```

```
Epoch 86/300
0s - loss: 0.0092 - val_loss: 0.0278
Epoch 87/300
0s - loss: 0.0092 - val_loss: 0.0277
Epoch 88/300
0s - loss: 0.0092 - val_loss: 0.0275
Epoch 89/300
0s - loss: 0.0092 - val_loss: 0.0276
Epoch 90/300
0s - loss: 0.0093 - val_loss: 0.0272
Epoch 91/300
0s - loss: 0.0092 - val_loss: 0.0273
Epoch 92/300
0s - loss: 0.0092 - val_loss: 0.0270
Epoch 93/300
0s - loss: 0.0092 - val_loss: 0.0269
Epoch 94/300
0s - loss: 0.0092 - val_loss: 0.0268
Epoch 95/300
0s - loss: 0.0092 - val_loss: 0.0267
Epoch 96/300
0s - loss: 0.0092 - val_loss: 0.0265
Epoch 97/300
0s - loss: 0.0091 - val_loss: 0.0266
Epoch 98/300
0s - loss: 0.0092 - val_loss: 0.0261
Epoch 99/300
0s - loss: 0.0092 - val_loss: 0.0262
Epoch 100/300
0s - loss: 0.0092 - val_loss: 0.0260
Epoch 101/300
0s - loss: 0.0092 - val_loss: 0.0261
Epoch 102/300
0s - loss: 0.0092 - val_loss: 0.0259
Epoch 103/300
0s - loss: 0.0092 - val_loss: 0.0257
Epoch 104/300
0s - loss: 0.0092 - val_loss: 0.0257
Epoch 105/300
0s - loss: 0.0092 - val_loss: 0.0255
Epoch 106/300
0s - loss: 0.0092 - val_loss: 0.0255
Epoch 107/300
0s - loss: 0.0092 - val_loss: 0.0253
Epoch 108/300
0s - loss: 0.0091 - val_loss: 0.0253
Epoch 109/300
0s - loss: 0.0092 - val_loss: 0.0250
Epoch 110/300
0s - loss: 0.0091 - val_loss: 0.0250
Epoch 111/300
0s - loss: 0.0092 - val_loss: 0.0250
Epoch 112/300
0s - loss: 0.0092 - val_loss: 0.0248
Epoch 113/300
0s - loss: 0.0091 - val_loss: 0.0248
Epoch 114/300
```

```
0s - loss: 0.0091 - val_loss: 0.0247
Epoch 115/300
0s - loss: 0.0092 - val_loss: 0.0245
Epoch 116/300
0s - loss: 0.0091 - val_loss: 0.0246
Epoch 117/300
0s - loss: 0.0091 - val_loss: 0.0243
Epoch 118/300
0s - loss: 0.0091 - val_loss: 0.0242
Epoch 119/300
0s - loss: 0.0091 - val_loss: 0.0241
Epoch 120/300
0s - loss: 0.0091 - val_loss: 0.0243
Epoch 121/300
0s - loss: 0.0091 - val_loss: 0.0241
Epoch 122/300
0s - loss: 0.0091 - val_loss: 0.0238
Epoch 123/300
0s - loss: 0.0090 - val_loss: 0.0238
Epoch 124/300
0s - loss: 0.0091 - val_loss: 0.0237
Epoch 125/300
0s - loss: 0.0091 - val_loss: 0.0235
Epoch 126/300
0s - loss: 0.0091 - val_loss: 0.0237
Epoch 127/300
0s - loss: 0.0090 - val_loss: 0.0237
Epoch 128/300
0s - loss: 0.0090 - val_loss: 0.0236
Epoch 129/300
0s - loss: 0.0091 - val_loss: 0.0233
Epoch 130/300
0s - loss: 0.0090 - val_loss: 0.0233
Epoch 131/300
0s - loss: 0.0090 - val_loss: 0.0231
Epoch 132/300
0s - loss: 0.0091 - val_loss: 0.0229
Epoch 133/300
0s - loss: 0.0090 - val_loss: 0.0230
Epoch 134/300
0s - loss: 0.0090 - val_loss: 0.0228
Epoch 135/300
0s - loss: 0.0090 - val_loss: 0.0227
Epoch 136/300
0s - loss: 0.0091 - val_loss: 0.0225
Epoch 137/300
0s - loss: 0.0091 - val_loss: 0.0224
Epoch 138/300
0s - loss: 0.0090 - val_loss: 0.0224
Epoch 139/300
0s - loss: 0.0089 - val_loss: 0.0224
Epoch 140/300
0s - loss: 0.0090 - val_loss: 0.0223
Epoch 141/300
0s - loss: 0.0091 - val_loss: 0.0219
Epoch 142/300
0s - loss: 0.0090 - val_loss: 0.0220
```

```
Epoch 143/300
0s - loss: 0.0089 - val_loss: 0.0220
Epoch 144/300
0s - loss: 0.0090 - val_loss: 0.0219
Epoch 145/300
0s - loss: 0.0091 - val_loss: 0.0216
Epoch 146/300
0s - loss: 0.0089 - val_loss: 0.0219
Epoch 147/300
0s - loss: 0.0089 - val_loss: 0.0217
Epoch 148/300
0s - loss: 0.0090 - val_loss: 0.0216
Epoch 149/300
0s - loss: 0.0091 - val_loss: 0.0215
Epoch 150/300
0s - loss: 0.0089 - val_loss: 0.0215
Epoch 151/300
0s - loss: 0.0089 - val_loss: 0.0215
Epoch 152/300
0s - loss: 0.0091 - val_loss: 0.0211
Epoch 153/300
0s - loss: 0.0089 - val_loss: 0.0214
Epoch 154/300
0s - loss: 0.0090 - val_loss: 0.0212
Epoch 155/300
0s - loss: 0.0090 - val_loss: 0.0212
Epoch 156/300
0s - loss: 0.0089 - val_loss: 0.0211
Epoch 157/300
0s - loss: 0.0090 - val_loss: 0.0208
Epoch 158/300
0s - loss: 0.0089 - val_loss: 0.0209
Epoch 159/300
0s - loss: 0.0090 - val_loss: 0.0207
Epoch 160/300
0s - loss: 0.0090 - val_loss: 0.0207
Epoch 161/300
0s - loss: 0.0090 - val_loss: 0.0207
Epoch 162/300
0s - loss: 0.0089 - val_loss: 0.0207
Epoch 163/300
0s - loss: 0.0090 - val_loss: 0.0204
Epoch 164/300
0s - loss: 0.0089 - val_loss: 0.0206
Epoch 165/300
0s - loss: 0.0090 - val_loss: 0.0203
Epoch 166/300
0s - loss: 0.0089 - val_loss: 0.0205
Epoch 167/300
0s - loss: 0.0090 - val_loss: 0.0202
Epoch 168/300
0s - loss: 0.0089 - val_loss: 0.0204
Epoch 169/300
0s - loss: 0.0090 - val_loss: 0.0202
Epoch 170/300
0s - loss: 0.0089 - val_loss: 0.0203
Epoch 171/300
```

```
0s - loss: 0.0089 - val_loss: 0.0202
Epoch 172/300
0s - loss: 0.0089 - val_loss: 0.0201
Epoch 173/300
0s - loss: 0.0089 - val_loss: 0.0201
Epoch 174/300
0s - loss: 0.0089 - val_loss: 0.0200
Epoch 175/300
0s - loss: 0.0089 - val_loss: 0.0200
Epoch 176/300
0s - loss: 0.0089 - val_loss: 0.0199
Epoch 177/300
0s - loss: 0.0089 - val_loss: 0.0199
Epoch 178/300
0s - loss: 0.0090 - val_loss: 0.0198
Epoch 179/300
0s - loss: 0.0089 - val_loss: 0.0197
Epoch 180/300
0s - loss: 0.0089 - val_loss: 0.0198
Epoch 181/300
0s - loss: 0.0089 - val_loss: 0.0197
Epoch 182/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 183/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 184/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 185/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 186/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 187/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 188/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 189/300
0s - loss: 0.0090 - val_loss: 0.0195
Epoch 190/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 191/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 192/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 193/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 194/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 195/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 196/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 197/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 198/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 199/300
0s - loss: 0.0089 - val_loss: 0.0194
```

```
Epoch 200/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 201/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 202/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 203/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 204/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 205/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 206/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 207/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 208/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 209/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 210/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 211/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 212/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 213/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 214/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 215/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 216/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 217/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 218/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 219/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 220/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 221/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 222/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 223/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 224/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 225/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 226/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 227/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 228/300
```

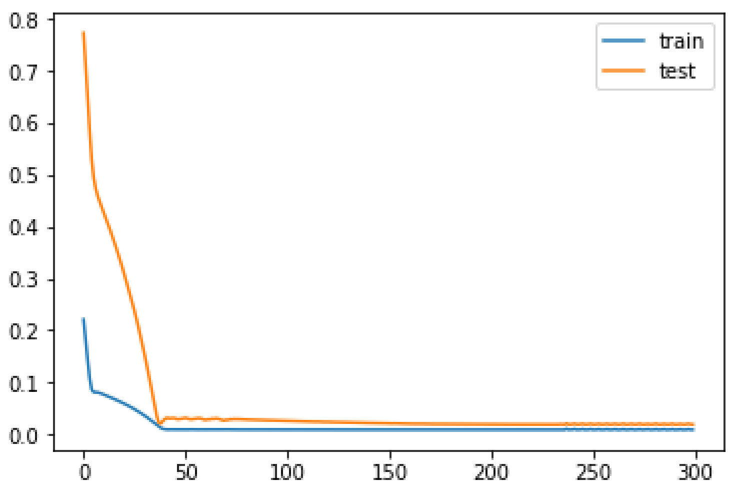
```
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 229/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 230/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 231/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 232/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 233/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 234/300
0s - loss: 0.0089 - val_loss: 0.0192
Epoch 235/300
0s - loss: 0.0093 - val_loss: 0.0198
Epoch 236/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 237/300
0s - loss: 0.0090 - val_loss: 0.0192
Epoch 238/300
0s - loss: 0.0109 - val_loss: 0.0212
Epoch 239/300
0s - loss: 0.0091 - val_loss: 0.0193
Epoch 240/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 241/300
0s - loss: 0.0095 - val_loss: 0.0198
Epoch 242/300
0s - loss: 0.0105 - val_loss: 0.0206
Epoch 243/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 244/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 245/300
0s - loss: 0.0095 - val_loss: 0.0198
Epoch 246/300
0s - loss: 0.0104 - val_loss: 0.0206
Epoch 247/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 248/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 249/300
0s - loss: 0.0094 - val_loss: 0.0198
Epoch 250/300
0s - loss: 0.0104 - val_loss: 0.0206
Epoch 251/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 252/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 253/300
0s - loss: 0.0094 - val_loss: 0.0198
Epoch 254/300
0s - loss: 0.0102 - val_loss: 0.0204
Epoch 255/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 256/300
0s - loss: 0.0092 - val_loss: 0.0192
```

```
Epoch 257/300
0s - loss: 0.0094 - val_loss: 0.0198
Epoch 258/300
0s - loss: 0.0103 - val_loss: 0.0205
Epoch 259/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 260/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 261/300
0s - loss: 0.0094 - val_loss: 0.0198
Epoch 262/300
0s - loss: 0.0102 - val_loss: 0.0204
Epoch 263/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 264/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 265/300
0s - loss: 0.0094 - val_loss: 0.0198
Epoch 266/300
0s - loss: 0.0102 - val_loss: 0.0206
Epoch 267/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 268/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 269/300
0s - loss: 0.0094 - val_loss: 0.0198
Epoch 270/300
0s - loss: 0.0102 - val_loss: 0.0204
Epoch 271/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 272/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 273/300
0s - loss: 0.0093 - val_loss: 0.0198
Epoch 274/300
0s - loss: 0.0101 - val_loss: 0.0205
Epoch 275/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 276/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 277/300
0s - loss: 0.0095 - val_loss: 0.0198
Epoch 278/300
0s - loss: 0.0102 - val_loss: 0.0201
Epoch 279/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 280/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 281/300
0s - loss: 0.0093 - val_loss: 0.0199
Epoch 282/300
0s - loss: 0.0101 - val_loss: 0.0206
Epoch 283/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 284/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 285/300
```

```
0s - loss: 0.0095 - val_loss: 0.0198
Epoch 286/300
0s - loss: 0.0101 - val_loss: 0.0201
Epoch 287/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 288/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 289/300
0s - loss: 0.0093 - val_loss: 0.0198
Epoch 290/300
0s - loss: 0.0101 - val_loss: 0.0204
Epoch 291/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 292/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 293/300
0s - loss: 0.0093 - val_loss: 0.0197
Epoch 294/300
0s - loss: 0.0100 - val_loss: 0.0201
Epoch 295/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 296/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 297/300
0s - loss: 0.0093 - val_loss: 0.0197
Epoch 298/300
0s - loss: 0.0101 - val_loss: 0.0205
Epoch 299/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 300/300
0s - loss: 0.0091 - val_loss: 0.0192
```

In [13]:

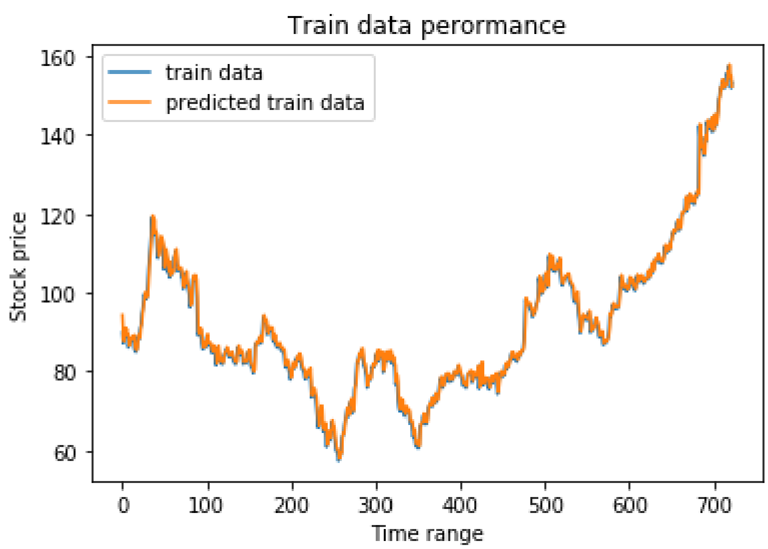
```
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



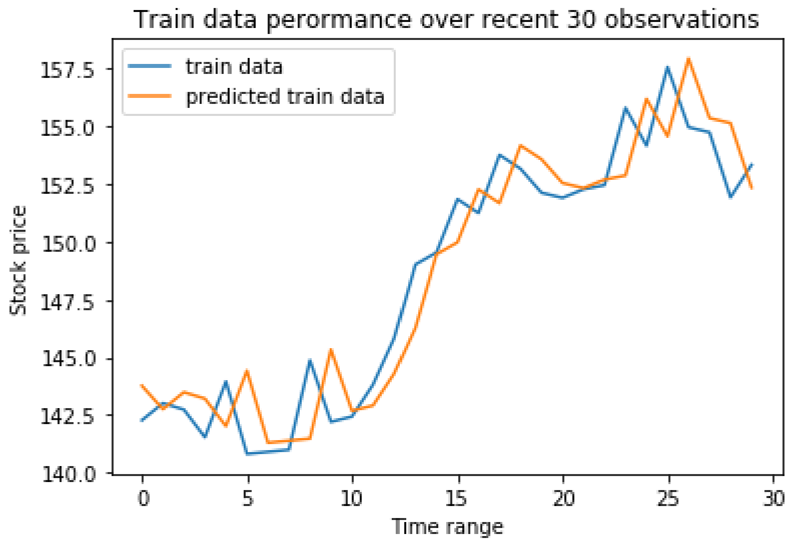
In [14]:
```
# make a prediction for train data
yhat_train = model.predict(train_X)
train_X = train_X.reshape((train_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat_train = concatenate((yhat_train, train_X[:, 1:]), axis=1)
inv_yhat_train = scaler.inverse_transform(inv_yhat_train)
inv_yhat_train = inv_yhat_train[:,0]
# invert scaling for actual
train_y = train_y.reshape((len(train_y), 1))
inv_y_train = concatenate((train_y, train_X[:, 1:]), axis=1)
inv_y_train = scaler.inverse_transform(inv_y_train)
inv_y_train = inv_y_train[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y_train, inv_yhat_train))
print('Train RMSE: %.3f' % rmse)
```

```
Train RMSE: 1.851
```

In [15]:
```
# plot only the train data and predicted train data
trainline, =pyplot.plot(inv_y_train, label='train data')  # blue one
trainPredictline, =pyplot.plot(inv_yhat_train, label='predicted train data') #
 orange one
pyplot.title("Train data perormance")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[trainline, trainPredictline])
pyplot.show()
```



In [16]:
```
# plot only the last 100 train data and predicted train data
trainline, =pyplot.plot(inv_y_train[-30:], label='train data')  # blue one
trainPredictline, =pyplot.plot(inv_yhat_train[-30:], label='predicted train da
ta') # orange one
pyplot.title("Train data perormance over recent 30 observations")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[trainline, trainPredictline])
pyplot.show()
```

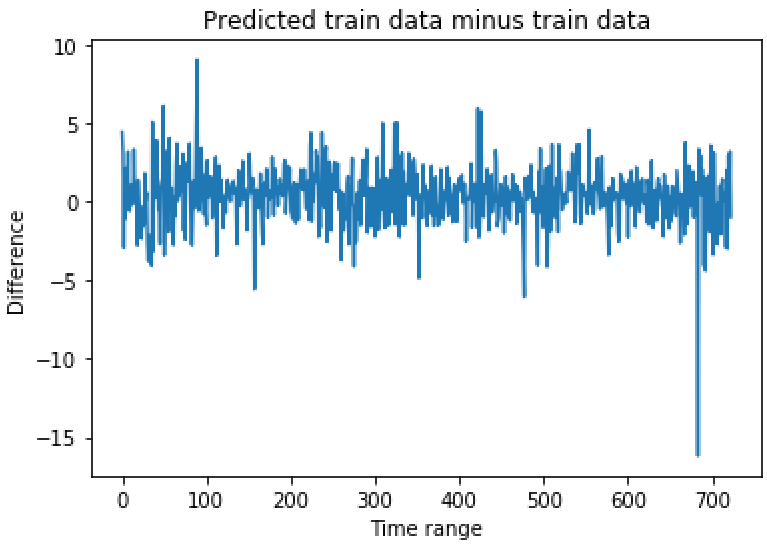


In [17]:
```
# check some observations of train data
print(inv_y_train.shape)
print(inv_yhat_train.shape)
# check the performance
print(inv_y_train[-1])
print(inv_yhat_train[-1])
#
print(inv_y_train[-2])
print(inv_yhat_train[-2])
#
print(inv_y_train[-3])
print(inv_yhat_train[-3])
#
print(inv_y_train[-4])
print(inv_yhat_train[-4])
```

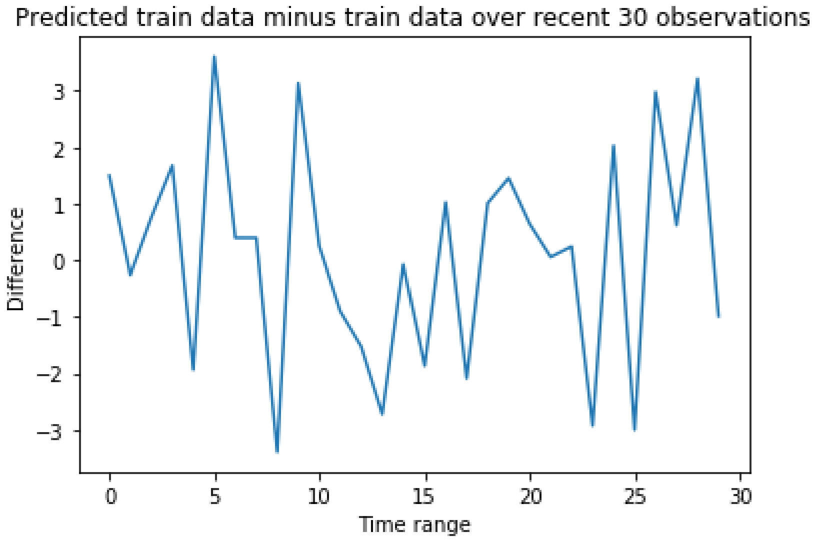
```
(723,)
(723,)
153.32
152.333
151.91
155.129
154.73
155.347
154.95
157.932
```

In [18]:
```
# plot the difference for train data
traindiff = inv_yhat_train - inv_y_train
#
maxtraindiff=abs(max(traindiff, key=abs))
print('The largest absolute difference for train data: %.2f' % (maxtraindiff))
#
pyplot.plot(traindiff)
#
pyplot.title("Predicted train data minus train data")
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```

```
The largest absolute difference for train data: 16.20
```



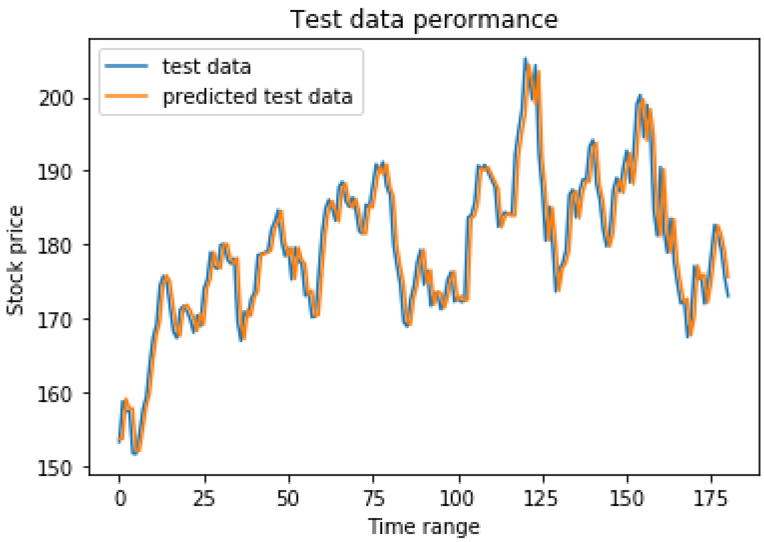
In [19]:
```
# plot the difference for the recent 30 obs of train data
pyplot.plot(traindiff[-30:])
#
pyplot.title("Predicted train data minus train data over recent 30 observation
s")
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```



In [20]:
```
# make a prediction for test data
yhat_test = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat_test = concatenate((yhat_test, test_X[:, 1:]), axis=1)
inv_yhat_test = scaler.inverse_transform(inv_yhat_test)
inv_yhat_test = inv_yhat_test[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y_test = concatenate((test_y, test_X[:, 1:]), axis=1)
inv_y_test = scaler.inverse_transform(inv_y_test)
inv_y_test = inv_y_test[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y_test, inv_yhat_test))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 3.627
```

In [21]:
```
# plot only the test data and predicted test data
testline, =pyplot.plot(inv_y_test, label='test data')  # blue one
testPredictline, =pyplot.plot(inv_yhat_test, label='predicted test data') # or
ange one
pyplot.title("Test data perormance")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[testline, testPredictline])
pyplot.show()
```



In [22]:
```
# plot only the last 30 test data and predicted test data
testline, =pyplot.plot(inv_y_test[-30:], label='test data')  # blue one
testPredictline, =pyplot.plot(inv_yhat_test[-30:], label='predicted test data'
) # orange one
pyplot.title("Test data perormance over recent 30 observations")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[testline, testPredictline])
pyplot.show()
```

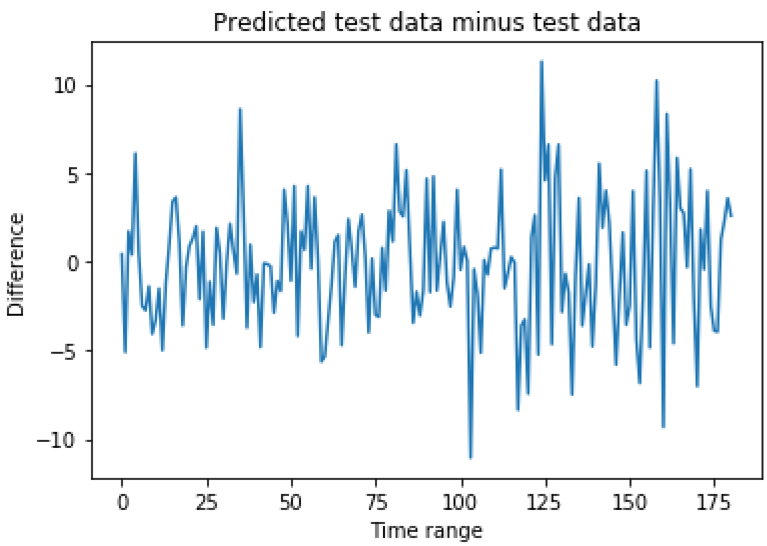
In [23]:

```
# check some observations of test data
print(inv_y_test.shape)
print(inv_yhat_test.shape)
# check the performance
print(inv_y_test[-1])
print(inv_yhat_test[-1])
#
print(inv_y_test[-2])
print(inv_yhat_test[-2])
#
print(inv_y_test[-3])
print(inv_yhat_test[-3])
#
print(inv_y_test[-4])
print(inv_yhat_test[-4])
```

```
(181,)
(181,)
173.09
175.667
175.57
179.13
179.11
181.356
181.39
182.613
```

In [24]:
```
# plot the difference for test data
testdiff = inv_yhat_test - inv_y_test
#
maxtestdiff=abs(max(testdiff, key=abs))
print('The largest absolute difference for test data: %.2f' % (maxtestdiff))
#
pyplot.plot(testdiff)
#
pyplot.title("Predicted test data minus test data")
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```

```
The largest absolute difference for test data: 11.28
```



In [25]:
```
# plot the difference for the recent 30 obs of test data
pyplot.plot(testdiff[-30:])
#
pyplot.title("Predicted test data minus test data over recent 30 observations"
)
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```

3 Multivariate Time Series Forecasting with LSTMs in Keras

3.1 Baseline

In [101]:
```
from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

In [102]:
```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
        n_vars = 1 if type(data) is list else data.shape[1]
        df = DataFrame(data)
        cols, names = list(), list()
        # input sequence (t-n, ... t-1)
        for i in range(n_in, 0, -1):
                cols.append(df.shift(i))
                names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
        # forecast sequence (t, t+1, ... t+n)
        for i in range(0, n_out):
                cols.append(df.shift(-i))
                if i == 0:
                        names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
                else:
                        names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_
vars)]
        # put it all together
        agg = concat(cols, axis=1)
        agg.columns = names
        # drop rows with NaN values
        if dropnan:
                agg.dropna(inplace=True)
        return agg
```

In [103]:
```
# now let get all the information for stock
stock =  read_csv('BABA20130425_20180425.csv', header=0)
print(stock.shape)
print(stock.head())
```

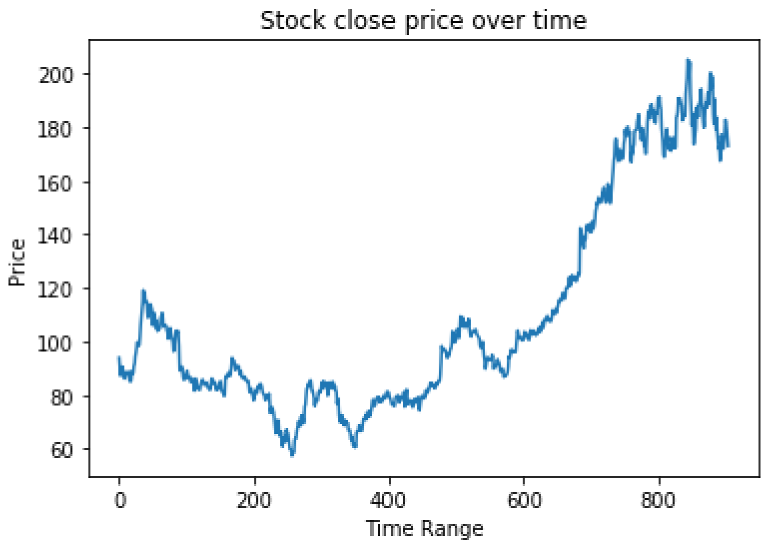
```
(905, 7)
         Date       Open       High        Low      Close  Adj Close  \
0  2014-09-19  92.699997  99.699997  89.949997  93.889999  93.889999
1  2014-09-22  92.699997  92.949997  89.500000  89.889999  89.889999
2  2014-09-23  88.940002  90.480003  86.620003  87.169998  87.169998
3  2014-09-24  88.470001  90.570000  87.220001  90.570000  90.570000
4  2014-09-25  91.089996  91.500000  88.500000  88.919998  88.919998

      Volume
0  271879400
1   66657800
2   39009800
3   32088000
4   28598000
```

In [104]:
```
# now get the stock close price
# astype means "Copy of the array, cast to a specified type."
stock_prices = stock.Close.values.astype("float32")
shape0=stock_prices.shape[0]
stock_prices = stock_prices.reshape(shape0, 1)
print(stock_prices.shape)
# print the prices of last five observations
print(stock_prices[-5:])
```

```
(905, 1)
[[ 182.67999268]
 [ 181.38999939]
 [ 179.11000061]
 [ 175.57000732]
 [ 173.08999634]]
```

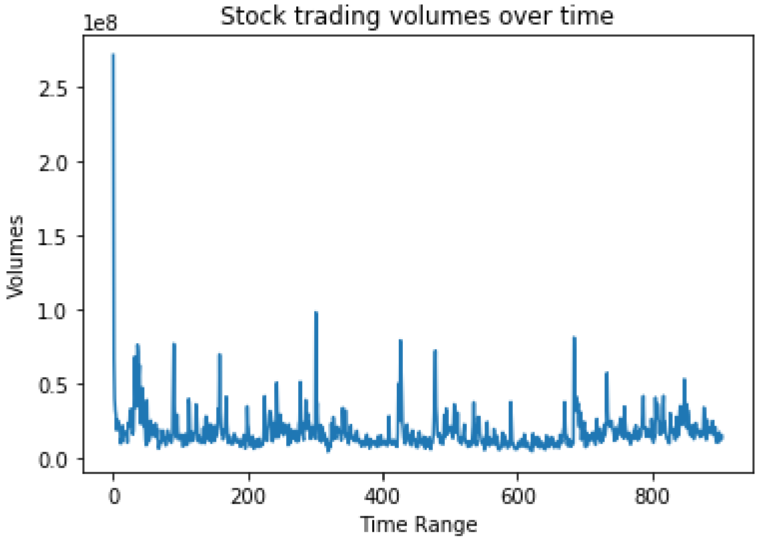
In [105]:
```
# Before doing any analysis, first plot the prices series(data)
pyplot.plot(stock_prices)
pyplot.title('Stock close price over time')
pyplot.ylabel('Price')
pyplot.xlabel('Time Range')
pyplot.show()
```



In [106]:
```
# now get the stock Volume
# astype means "Copy of the array, cast to a specified type."
stock_volumes = stock.Volume.values.astype("float32")
shape0=stock_volumes.shape[0]
stock_volumes = stock_volumes.reshape(shape0, 1)
print(stock_volumes.shape)
# print the volumes of last five observations
print(stock_volumes[-5:])
```

```
(905, 1)
[[ 16972700.]
 [ 11989000.]
 [ 14473100.]
 [ 12033900.]
 [ 14340100.]]
```

In [107]:
```
# Before doing any analysis, first plot the volumes series(data)
pyplot.plot(stock_volumes)
pyplot.title('Stock trading volumes over time')
pyplot.ylabel('Volumes')
pyplot.xlabel('Time Range')
pyplot.show()
```



In [108]:
```
values = concatenate((stock_prices, stock_volumes), axis=1)
```

In [109]:
```
# check the last five observations to make sure it's correct
print(values[-5:, :])
```

```
[[  1.82679993e+02   1.69727000e+07]
 [  1.81389999e+02   1.19890000e+07]
 [  1.79110001e+02   1.44731000e+07]
 [  1.75570007e+02   1.20339000e+07]
 [  1.73089996e+02   1.43401000e+07]]
```

In [110]:
```
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
print(reframed.head())
# var1 means prices, var2 means volumns
```

```
   var1(t-1)  var2(t-1)   var1(t)   var2(t)
1   0.246905   1.000000  0.219847  0.234545
2   0.219847   0.234545  0.201448  0.131421
3   0.201448   0.131421  0.224447  0.105603
4   0.224447   0.105603  0.213286  0.092586
5   0.213286   0.092586  0.223703  0.054325
```

In [111]:
```
# drop columns we don't want to predict
reframed.drop(reframed.columns[[3]], axis=1, inplace=True)
print(reframed.head())
```

```
   var1(t-1)  var2(t-1)   var1(t)
1   0.246905   1.000000  0.219847
2   0.219847   0.234545  0.201448
3   0.201448   0.131421  0.224447
4   0.224447   0.105603  0.213286
5   0.213286   0.092586  0.223703
```

In [112]:
```
# split into train and test sets
values = reframed.values
n_train = int(reframed.shape[0] * 0.8)
train = values[:n_train, :]
test = values[n_train:, :]

# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))

print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(723, 1, 2) (723,) (181, 1, 2) (181,)
```

In [113]:
```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
# fit network
history = model.fit(train_X, train_y, epochs=300, batch_size=72, validation_da
ta=(test_X, test_y), verbose=2, shuffle=False)
```

```
Train on 723 samples, validate on 181 samples
Epoch 1/300
0s - loss: 0.2326 - val_loss: 0.8140
Epoch 2/300
0s - loss: 0.1903 - val_loss: 0.7502
Epoch 3/300
0s - loss: 0.1501 - val_loss: 0.6860
Epoch 4/300
0s - loss: 0.1151 - val_loss: 0.6217
Epoch 5/300
0s - loss: 0.0926 - val_loss: 0.5663
Epoch 6/300
0s - loss: 0.0867 - val_loss: 0.5308
Epoch 7/300
0s - loss: 0.0867 - val_loss: 0.5095
Epoch 8/300
0s - loss: 0.0862 - val_loss: 0.4953
Epoch 9/300
0s - loss: 0.0851 - val_loss: 0.4840
Epoch 10/300
0s - loss: 0.0836 - val_loss: 0.4738
Epoch 11/300
0s - loss: 0.0820 - val_loss: 0.4637
Epoch 12/300
0s - loss: 0.0803 - val_loss: 0.4535
Epoch 13/300
0s - loss: 0.0787 - val_loss: 0.4432
Epoch 14/300
0s - loss: 0.0771 - val_loss: 0.4326
Epoch 15/300
0s - loss: 0.0755 - val_loss: 0.4218
Epoch 16/300
0s - loss: 0.0739 - val_loss: 0.4107
Epoch 17/300
0s - loss: 0.0723 - val_loss: 0.3994
Epoch 18/300
0s - loss: 0.0707 - val_loss: 0.3879
Epoch 19/300
0s - loss: 0.0690 - val_loss: 0.3762
Epoch 20/300
0s - loss: 0.0673 - val_loss: 0.3642
Epoch 21/300
0s - loss: 0.0656 - val_loss: 0.3519
Epoch 22/300
0s - loss: 0.0638 - val_loss: 0.3391
Epoch 23/300
0s - loss: 0.0619 - val_loss: 0.3260
Epoch 24/300
0s - loss: 0.0601 - val_loss: 0.3127
Epoch 25/300
0s - loss: 0.0582 - val_loss: 0.2993
Epoch 26/300
0s - loss: 0.0561 - val_loss: 0.2856
Epoch 27/300
0s - loss: 0.0540 - val_loss: 0.2715
Epoch 28/300
0s - loss: 0.0518 - val_loss: 0.2569
```

```
Epoch 29/300
0s - loss: 0.0496 - val_loss: 0.2418
Epoch 30/300
0s - loss: 0.0472 - val_loss: 0.2258
Epoch 31/300
0s - loss: 0.0448 - val_loss: 0.2092
Epoch 32/300
0s - loss: 0.0423 - val_loss: 0.1919
Epoch 33/300
0s - loss: 0.0397 - val_loss: 0.1739
Epoch 34/300
0s - loss: 0.0370 - val_loss: 0.1556
Epoch 35/300
0s - loss: 0.0341 - val_loss: 0.1369
Epoch 36/300
0s - loss: 0.0311 - val_loss: 0.1174
Epoch 37/300
0s - loss: 0.0280 - val_loss: 0.0963
Epoch 38/300
0s - loss: 0.0248 - val_loss: 0.0745
Epoch 39/300
0s - loss: 0.0216 - val_loss: 0.0536
Epoch 40/300
0s - loss: 0.0181 - val_loss: 0.0328
Epoch 41/300
0s - loss: 0.0150 - val_loss: 0.0215
Epoch 42/300
0s - loss: 0.0123 - val_loss: 0.0221
Epoch 43/300
0s - loss: 0.0106 - val_loss: 0.0290
Epoch 44/300
0s - loss: 0.0098 - val_loss: 0.0328
Epoch 45/300
0s - loss: 0.0095 - val_loss: 0.0345
Epoch 46/300
0s - loss: 0.0098 - val_loss: 0.0328
Epoch 47/300
0s - loss: 0.0096 - val_loss: 0.0312
Epoch 48/300
0s - loss: 0.0100 - val_loss: 0.0298
Epoch 49/300
0s - loss: 0.0099 - val_loss: 0.0300
Epoch 50/300
0s - loss: 0.0096 - val_loss: 0.0311
Epoch 51/300
0s - loss: 0.0096 - val_loss: 0.0319
Epoch 52/300
0s - loss: 0.0095 - val_loss: 0.0327
Epoch 53/300
0s - loss: 0.0094 - val_loss: 0.0319
Epoch 54/300
0s - loss: 0.0097 - val_loss: 0.0307
Epoch 55/300
0s - loss: 0.0097 - val_loss: 0.0301
Epoch 56/300
0s - loss: 0.0096 - val_loss: 0.0310
Epoch 57/300
```

```
0s - loss: 0.0096 - val_loss: 0.0315
Epoch 58/300
0s - loss: 0.0094 - val_loss: 0.0310
Epoch 59/300
0s - loss: 0.0096 - val_loss: 0.0300
Epoch 60/300
0s - loss: 0.0098 - val_loss: 0.0288
Epoch 61/300
0s - loss: 0.0095 - val_loss: 0.0306
Epoch 62/300
0s - loss: 0.0096 - val_loss: 0.0304
Epoch 63/300
0s - loss: 0.0094 - val_loss: 0.0316
Epoch 64/300
0s - loss: 0.0094 - val_loss: 0.0301
Epoch 65/300
0s - loss: 0.0096 - val_loss: 0.0294
Epoch 66/300
0s - loss: 0.0097 - val_loss: 0.0281
Epoch 67/300
0s - loss: 0.0095 - val_loss: 0.0295
Epoch 68/300
0s - loss: 0.0095 - val_loss: 0.0294
Epoch 69/300
0s - loss: 0.0094 - val_loss: 0.0307
Epoch 70/300
0s - loss: 0.0094 - val_loss: 0.0307
Epoch 71/300
0s - loss: 0.0093 - val_loss: 0.0296
Epoch 72/300
0s - loss: 0.0095 - val_loss: 0.0286
Epoch 73/300
0s - loss: 0.0097 - val_loss: 0.0272
Epoch 74/300
0s - loss: 0.0094 - val_loss: 0.0286
Epoch 75/300
0s - loss: 0.0094 - val_loss: 0.0288
Epoch 76/300
0s - loss: 0.0093 - val_loss: 0.0299
Epoch 77/300
0s - loss: 0.0093 - val_loss: 0.0300
Epoch 78/300
0s - loss: 0.0093 - val_loss: 0.0302
Epoch 79/300
0s - loss: 0.0093 - val_loss: 0.0302
Epoch 80/300
0s - loss: 0.0093 - val_loss: 0.0300
Epoch 81/300
0s - loss: 0.0093 - val_loss: 0.0300
Epoch 82/300
0s - loss: 0.0093 - val_loss: 0.0298
Epoch 83/300
0s - loss: 0.0092 - val_loss: 0.0298
Epoch 84/300
0s - loss: 0.0092 - val_loss: 0.0296
Epoch 85/300
0s - loss: 0.0093 - val_loss: 0.0293
```

```
Epoch 86/300
0s - loss: 0.0092 - val_loss: 0.0294
Epoch 87/300
0s - loss: 0.0093 - val_loss: 0.0291
Epoch 88/300
0s - loss: 0.0092 - val_loss: 0.0291
Epoch 89/300
0s - loss: 0.0092 - val_loss: 0.0287
Epoch 90/300
0s - loss: 0.0092 - val_loss: 0.0287
Epoch 91/300
0s - loss: 0.0092 - val_loss: 0.0285
Epoch 92/300
0s - loss: 0.0092 - val_loss: 0.0282
Epoch 93/300
0s - loss: 0.0092 - val_loss: 0.0280
Epoch 94/300
0s - loss: 0.0092 - val_loss: 0.0281
Epoch 95/300
0s - loss: 0.0092 - val_loss: 0.0276
Epoch 96/300
0s - loss: 0.0092 - val_loss: 0.0275
Epoch 97/300
0s - loss: 0.0092 - val_loss: 0.0274
Epoch 98/300
0s - loss: 0.0092 - val_loss: 0.0273
Epoch 99/300
0s - loss: 0.0092 - val_loss: 0.0269
Epoch 100/300
0s - loss: 0.0092 - val_loss: 0.0269
Epoch 101/300
0s - loss: 0.0092 - val_loss: 0.0265
Epoch 102/300
0s - loss: 0.0092 - val_loss: 0.0265
Epoch 103/300
0s - loss: 0.0092 - val_loss: 0.0263
Epoch 104/300
0s - loss: 0.0092 - val_loss: 0.0261
Epoch 105/300
0s - loss: 0.0091 - val_loss: 0.0263
Epoch 106/300
0s - loss: 0.0092 - val_loss: 0.0260
Epoch 107/300
0s - loss: 0.0092 - val_loss: 0.0258
Epoch 108/300
0s - loss: 0.0091 - val_loss: 0.0258
Epoch 109/300
0s - loss: 0.0092 - val_loss: 0.0256
Epoch 110/300
0s - loss: 0.0092 - val_loss: 0.0254
Epoch 111/300
0s - loss: 0.0091 - val_loss: 0.0255
Epoch 112/300
0s - loss: 0.0091 - val_loss: 0.0253
Epoch 113/300
0s - loss: 0.0091 - val_loss: 0.0252
Epoch 114/300
```

```
0s - loss: 0.0091 - val_loss: 0.0251
Epoch 115/300
0s - loss: 0.0091 - val_loss: 0.0249
Epoch 116/300
0s - loss: 0.0091 - val_loss: 0.0249
Epoch 117/300
0s - loss: 0.0091 - val_loss: 0.0247
Epoch 118/300
0s - loss: 0.0091 - val_loss: 0.0246
Epoch 119/300
0s - loss: 0.0091 - val_loss: 0.0246
Epoch 120/300
0s - loss: 0.0091 - val_loss: 0.0246
Epoch 121/300
0s - loss: 0.0091 - val_loss: 0.0242
Epoch 122/300
0s - loss: 0.0091 - val_loss: 0.0244
Epoch 123/300
0s - loss: 0.0091 - val_loss: 0.0241
Epoch 124/300
0s - loss: 0.0090 - val_loss: 0.0241
Epoch 125/300
0s - loss: 0.0091 - val_loss: 0.0239
Epoch 126/300
0s - loss: 0.0091 - val_loss: 0.0238
Epoch 127/300
0s - loss: 0.0090 - val_loss: 0.0237
Epoch 128/300
0s - loss: 0.0091 - val_loss: 0.0234
Epoch 129/300
0s - loss: 0.0091 - val_loss: 0.0233
Epoch 130/300
0s - loss: 0.0091 - val_loss: 0.0233
Epoch 131/300
0s - loss: 0.0091 - val_loss: 0.0231
Epoch 132/300
0s - loss: 0.0090 - val_loss: 0.0232
Epoch 133/300
0s - loss: 0.0090 - val_loss: 0.0229
Epoch 134/300
0s - loss: 0.0091 - val_loss: 0.0226
Epoch 135/300
0s - loss: 0.0090 - val_loss: 0.0228
Epoch 136/300
0s - loss: 0.0090 - val_loss: 0.0226
Epoch 137/300
0s - loss: 0.0091 - val_loss: 0.0224
Epoch 138/300
0s - loss: 0.0090 - val_loss: 0.0223
Epoch 139/300
0s - loss: 0.0090 - val_loss: 0.0222
Epoch 140/300
0s - loss: 0.0090 - val_loss: 0.0222
Epoch 141/300
0s - loss: 0.0090 - val_loss: 0.0220
Epoch 142/300
0s - loss: 0.0091 - val_loss: 0.0218
```

```
Epoch 143/300
0s - loss: 0.0089 - val_loss: 0.0221
Epoch 144/300
0s - loss: 0.0089 - val_loss: 0.0218
Epoch 145/300
0s - loss: 0.0090 - val_loss: 0.0215
Epoch 146/300
0s - loss: 0.0092 - val_loss: 0.0212
Epoch 147/300
0s - loss: 0.0089 - val_loss: 0.0216
Epoch 148/300
0s - loss: 0.0089 - val_loss: 0.0216
Epoch 149/300
0s - loss: 0.0090 - val_loss: 0.0213
Epoch 150/300
0s - loss: 0.0091 - val_loss: 0.0210
Epoch 151/300
0s - loss: 0.0089 - val_loss: 0.0214
Epoch 152/300
0s - loss: 0.0089 - val_loss: 0.0211
Epoch 153/300
0s - loss: 0.0090 - val_loss: 0.0209
Epoch 154/300
0s - loss: 0.0090 - val_loss: 0.0206
Epoch 155/300
0s - loss: 0.0089 - val_loss: 0.0209
Epoch 156/300
0s - loss: 0.0090 - val_loss: 0.0206
Epoch 157/300
0s - loss: 0.0089 - val_loss: 0.0208
Epoch 158/300
0s - loss: 0.0090 - val_loss: 0.0205
Epoch 159/300
0s - loss: 0.0089 - val_loss: 0.0207
Epoch 160/300
0s - loss: 0.0089 - val_loss: 0.0206
Epoch 161/300
0s - loss: 0.0090 - val_loss: 0.0205
Epoch 162/300
0s - loss: 0.0090 - val_loss: 0.0204
Epoch 163/300
0s - loss: 0.0089 - val_loss: 0.0203
Epoch 164/300
0s - loss: 0.0089 - val_loss: 0.0205
Epoch 165/300
0s - loss: 0.0090 - val_loss: 0.0200
Epoch 166/300
0s - loss: 0.0089 - val_loss: 0.0203
Epoch 167/300
0s - loss: 0.0090 - val_loss: 0.0199
Epoch 168/300
0s - loss: 0.0089 - val_loss: 0.0201
Epoch 169/300
0s - loss: 0.0090 - val_loss: 0.0198
Epoch 170/300
0s - loss: 0.0089 - val_loss: 0.0200
Epoch 171/300
```

```
0s - loss: 0.0090 - val_loss: 0.0198
Epoch 172/300
0s - loss: 0.0089 - val_loss: 0.0199
Epoch 173/300
0s - loss: 0.0090 - val_loss: 0.0198
Epoch 174/300
0s - loss: 0.0089 - val_loss: 0.0199
Epoch 175/300
0s - loss: 0.0090 - val_loss: 0.0197
Epoch 176/300
0s - loss: 0.0089 - val_loss: 0.0197
Epoch 177/300
0s - loss: 0.0089 - val_loss: 0.0199
Epoch 178/300
0s - loss: 0.0090 - val_loss: 0.0195
Epoch 179/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 180/300
0s - loss: 0.0090 - val_loss: 0.0195
Epoch 181/300
0s - loss: 0.0089 - val_loss: 0.0196
Epoch 182/300
0s - loss: 0.0090 - val_loss: 0.0195
Epoch 183/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 184/300
0s - loss: 0.0088 - val_loss: 0.0195
Epoch 185/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 186/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 187/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 188/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 189/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 190/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 191/300
0s - loss: 0.0088 - val_loss: 0.0194
Epoch 192/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 193/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 194/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 195/300
0s - loss: 0.0088 - val_loss: 0.0194
Epoch 196/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 197/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 198/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 199/300
0s - loss: 0.0089 - val_loss: 0.0193
```

```
Epoch 200/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 201/300
0s - loss: 0.0090 - val_loss: 0.0193
Epoch 202/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 203/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 204/300
0s - loss: 0.0088 - val_loss: 0.0193
Epoch 205/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 206/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 207/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 208/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 209/300
0s - loss: 0.0089 - val_loss: 0.0195
Epoch 210/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 211/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 212/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 213/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 214/300
0s - loss: 0.0090 - val_loss: 0.0195
Epoch 215/300
0s - loss: 0.0088 - val_loss: 0.0194
Epoch 216/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 217/300
0s - loss: 0.0090 - val_loss: 0.0192
Epoch 218/300
0s - loss: 0.0093 - val_loss: 0.0196
Epoch 219/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 220/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 221/300
0s - loss: 0.0106 - val_loss: 0.0212
Epoch 222/300
0s - loss: 0.0096 - val_loss: 0.0195
Epoch 223/300
0s - loss: 0.0091 - val_loss: 0.0197
Epoch 224/300
0s - loss: 0.0094 - val_loss: 0.0193
Epoch 225/300
0s - loss: 0.0092 - val_loss: 0.0194
Epoch 226/300
0s - loss: 0.0107 - val_loss: 0.0203
Epoch 227/300
0s - loss: 0.0088 - val_loss: 0.0193
Epoch 228/300
```

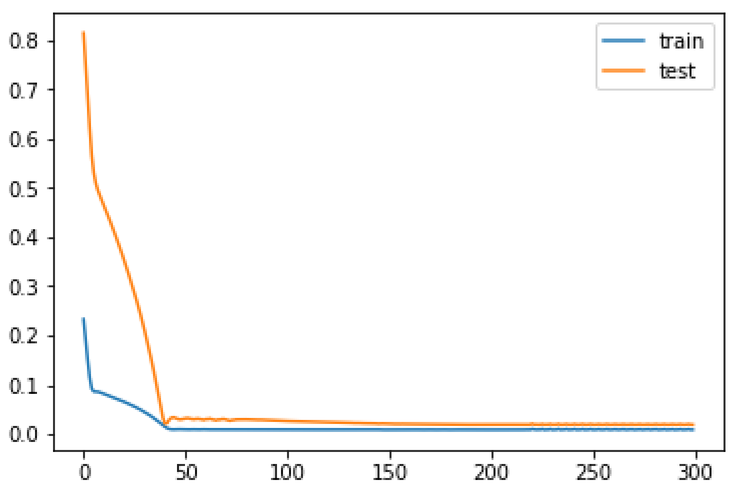
```
0s - loss: 0.0093 - val_loss: 0.0192
Epoch 229/300
0s - loss: 0.0093 - val_loss: 0.0195
Epoch 230/300
0s - loss: 0.0106 - val_loss: 0.0205
Epoch 231/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 232/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 233/300
0s - loss: 0.0094 - val_loss: 0.0195
Epoch 234/300
0s - loss: 0.0108 - val_loss: 0.0207
Epoch 235/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 236/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 237/300
0s - loss: 0.0095 - val_loss: 0.0196
Epoch 238/300
0s - loss: 0.0105 - val_loss: 0.0204
Epoch 239/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 240/300
0s - loss: 0.0093 - val_loss: 0.0192
Epoch 241/300
0s - loss: 0.0095 - val_loss: 0.0196
Epoch 242/300
0s - loss: 0.0105 - val_loss: 0.0204
Epoch 243/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 244/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 245/300
0s - loss: 0.0094 - val_loss: 0.0197
Epoch 246/300
0s - loss: 0.0105 - val_loss: 0.0204
Epoch 247/300
0s - loss: 0.0089 - val_loss: 0.0193
Epoch 248/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 249/300
0s - loss: 0.0096 - val_loss: 0.0196
Epoch 250/300
0s - loss: 0.0104 - val_loss: 0.0204
Epoch 251/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 252/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 253/300
0s - loss: 0.0094 - val_loss: 0.0196
Epoch 254/300
0s - loss: 0.0103 - val_loss: 0.0202
Epoch 255/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 256/300
0s - loss: 0.0092 - val_loss: 0.0192
```

```
Epoch 257/300
0s - loss: 0.0095 - val_loss: 0.0196
Epoch 258/300
0s - loss: 0.0104 - val_loss: 0.0204
Epoch 259/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 260/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 261/300
0s - loss: 0.0094 - val_loss: 0.0196
Epoch 262/300
0s - loss: 0.0102 - val_loss: 0.0201
Epoch 263/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 264/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 265/300
0s - loss: 0.0095 - val_loss: 0.0195
Epoch 266/300
0s - loss: 0.0103 - val_loss: 0.0202
Epoch 267/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 268/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 269/300
0s - loss: 0.0095 - val_loss: 0.0196
Epoch 270/300
0s - loss: 0.0102 - val_loss: 0.0202
Epoch 271/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 272/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 273/300
0s - loss: 0.0094 - val_loss: 0.0196
Epoch 274/300
0s - loss: 0.0102 - val_loss: 0.0201
Epoch 275/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 276/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 277/300
0s - loss: 0.0095 - val_loss: 0.0197
Epoch 278/300
0s - loss: 0.0102 - val_loss: 0.0201
Epoch 279/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 280/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 281/300
0s - loss: 0.0095 - val_loss: 0.0197
Epoch 282/300
0s - loss: 0.0102 - val_loss: 0.0202
Epoch 283/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 284/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 285/300
```

```
0s - loss: 0.0094 - val_loss: 0.0196
Epoch 286/300
0s - loss: 0.0102 - val_loss: 0.0201
Epoch 287/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 288/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 289/300
0s - loss: 0.0094 - val_loss: 0.0196
Epoch 290/300
0s - loss: 0.0101 - val_loss: 0.0200
Epoch 291/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 292/300
0s - loss: 0.0092 - val_loss: 0.0192
Epoch 293/300
0s - loss: 0.0094 - val_loss: 0.0196
Epoch 294/300
0s - loss: 0.0102 - val_loss: 0.0202
Epoch 295/300
0s - loss: 0.0090 - val_loss: 0.0194
Epoch 296/300
0s - loss: 0.0091 - val_loss: 0.0192
Epoch 297/300
0s - loss: 0.0094 - val_loss: 0.0197
Epoch 298/300
0s - loss: 0.0102 - val_loss: 0.0203
Epoch 299/300
0s - loss: 0.0089 - val_loss: 0.0194
Epoch 300/300
0s - loss: 0.0091 - val_loss: 0.0192
```

In [114]:

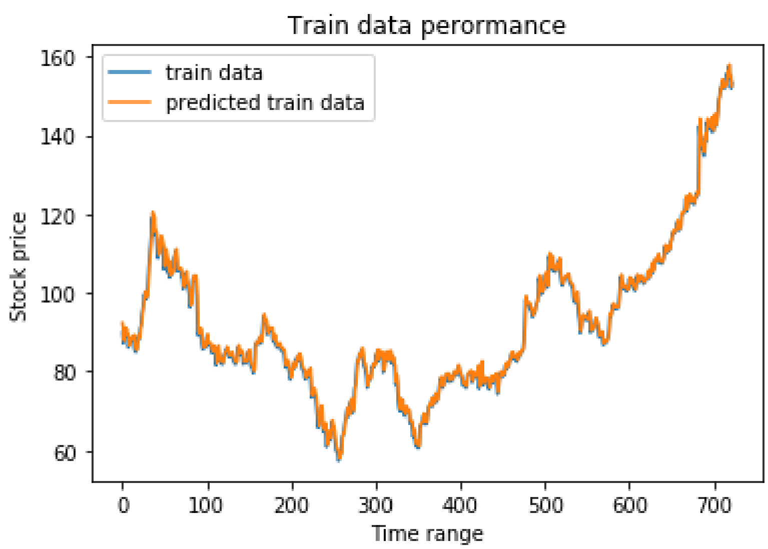
```
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



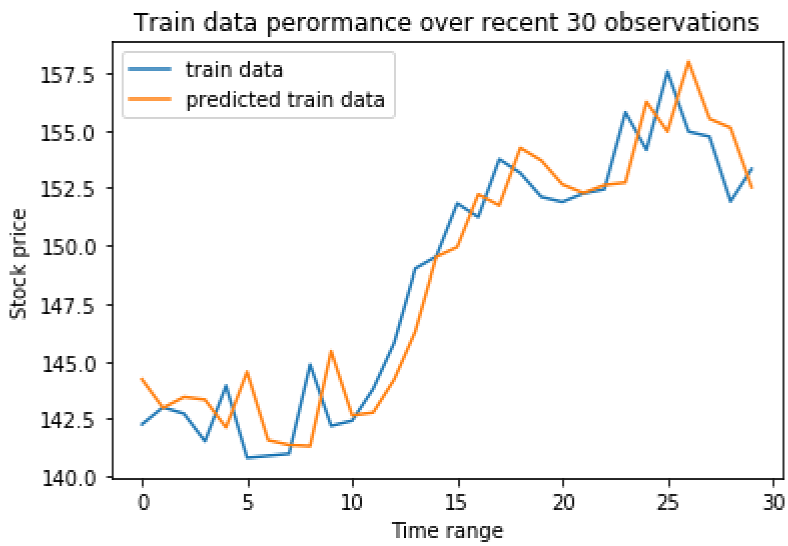
In [115]:
```
# make a prediction for train data
yhat_train = model.predict(train_X)
train_X = train_X.reshape((train_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat_train = concatenate((yhat_train, train_X[:, 1:]), axis=1)
inv_yhat_train = scaler.inverse_transform(inv_yhat_train)
inv_yhat_train = inv_yhat_train[:,0]
# invert scaling for actual
train_y = train_y.reshape((len(train_y), 1))
inv_y_train = concatenate((train_y, train_X[:, 1:]), axis=1)
inv_y_train = scaler.inverse_transform(inv_y_train)
inv_y_train = inv_y_train[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y_train, inv_yhat_train))
print('Train RMSE: %.3f' % rmse)
```

```
Train RMSE: 1.860
```

In [116]:
```
# plot only the train data and predicted train data
trainline, =pyplot.plot(inv_y_train, label='train data')  # blue one
trainPredictline, =pyplot.plot(inv_yhat_train, label='predicted train data') #
 orange one
pyplot.title("Train data perormance")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[trainline, trainPredictline])
pyplot.show()
```



In [117]:
```
# plot only the last 100 train data and predicted train data
trainline, =pyplot.plot(inv_y_train[-30:], label='train data')  # blue one
trainPredictline, =pyplot.plot(inv_yhat_train[-30:], label='predicted train da
ta') # orange one
pyplot.title("Train data perormance over recent 30 observations")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[trainline, trainPredictline])
pyplot.show()
```

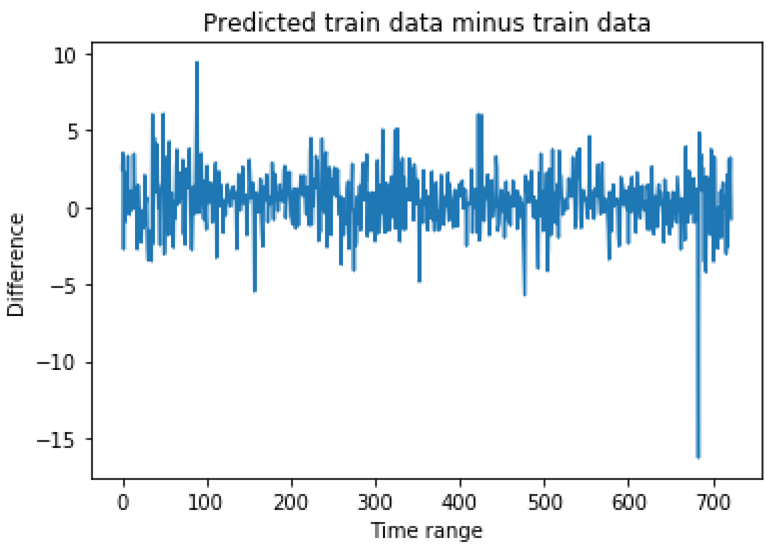


In [118]:
```
# check some observations of train data
print(inv_y_train.shape)
print(inv_yhat_train.shape)
# check the performance
print(inv_y_train[-1])
print(inv_yhat_train[-1])
#
print(inv_y_train[-2])
print(inv_yhat_train[-2])
#
print(inv_y_train[-3])
print(inv_yhat_train[-3])
#
print(inv_y_train[-4])
print(inv_yhat_train[-4])
```

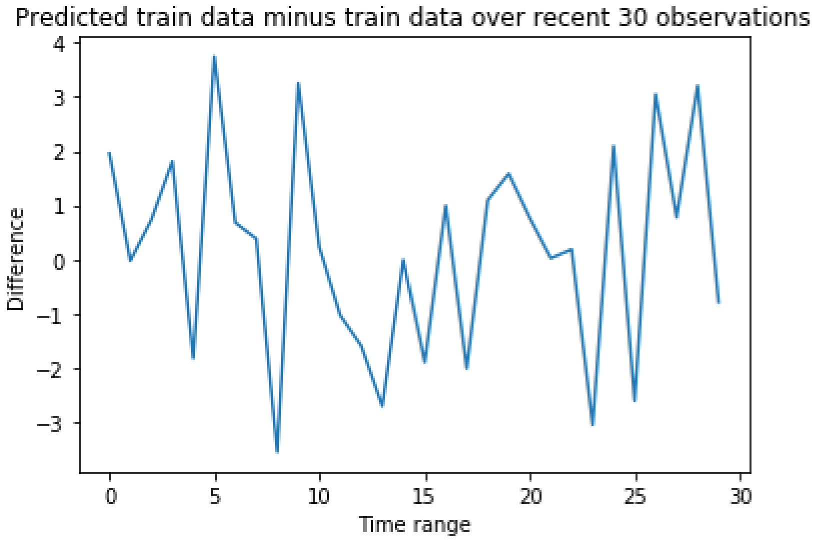
```
(723,)
(723,)
153.32
152.536
151.91
155.117
154.73
155.508
154.95
157.997
```

In [119]:
```
# plot the difference for train data
traindiff = inv_yhat_train - inv_y_train
#
maxtraindiff=abs(max(traindiff, key=abs))
print('The largest absolute difference for train data: %.2f' % (maxtraindiff))
#
pyplot.plot(traindiff)
#
pyplot.title("Predicted train data minus train data")
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```

```
The largest absolute difference for train data: 16.27
```



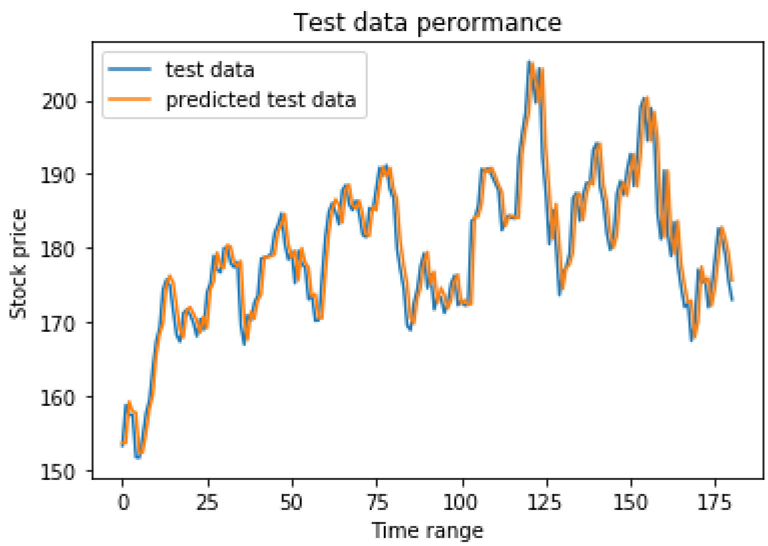
In [120]:
```
# plot the difference for the recent 30 obs of train data
pyplot.plot(traindiff[-30:])
#
pyplot.title("Predicted train data minus train data over recent 30 observation
s")
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```



In [121]:
```
# make a prediction for test data
yhat_test = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], test_X.shape[2]))
# invert scaling for forecast
inv_yhat_test = concatenate((yhat_test, test_X[:, 1:]), axis=1)
inv_yhat_test = scaler.inverse_transform(inv_yhat_test)
inv_yhat_test = inv_yhat_test[:,0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y_test = concatenate((test_y, test_X[:, 1:]), axis=1)
inv_y_test = scaler.inverse_transform(inv_y_test)
inv_y_test = inv_y_test[:,0]
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y_test, inv_yhat_test))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 3.650
```

In [122]:
```
# plot only the test data and predicted test data
testline, =pyplot.plot(inv_y_test, label='test data')  # blue one
testPredictline, =pyplot.plot(inv_yhat_test, label='predicted test data') # or
ange one
pyplot.title("Test data perormance")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[testline, testPredictline])
pyplot.show()
```



In [123]:
```
# plot only the last 30 test data and predicted test data
testline, =pyplot.plot(inv_y_test[-30:], label='test data')  # blue one
testPredictline, =pyplot.plot(inv_yhat_test[-30:], label='predicted test data'
) # orange one
pyplot.title("Test data perormance over recent 30 observations")
pyplot.ylabel('Stock price')
pyplot.xlabel('Time range')
pyplot.legend(handles=[testline, testPredictline])
pyplot.show()
```

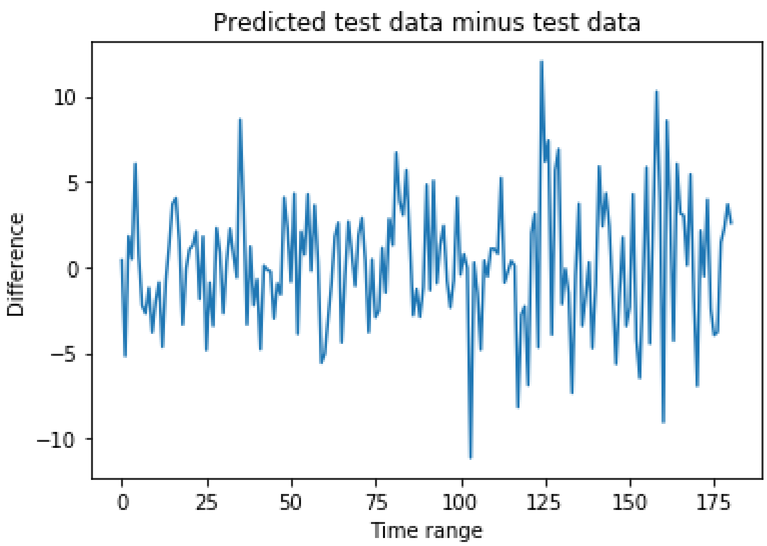
In [124]:

```
# check some observations of test data
print(inv_y_test.shape)
print(inv_yhat_test.shape)
# check the performance
print(inv_y_test[-1])
print(inv_yhat_test[-1])
#
print(inv_y_test[-2])
print(inv_yhat_test[-2])
#
print(inv_y_test[-3])
print(inv_yhat_test[-3])
#
print(inv_y_test[-4])
print(inv_yhat_test[-4])
```

```
(181,)
(181,)
173.09
175.684
175.57
179.246
179.11
181.384
181.39
182.837
```

In [125]:
```
# plot the difference for test data
testdiff = inv_yhat_test - inv_y_test
#
maxtestdiff=abs(max(testdiff, key=abs))
print('The largest absolute difference for test data: %.2f' % (maxtestdiff))
#
pyplot.plot(testdiff)
#
pyplot.title("Predicted test data minus test data")
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```

The largest absolute difference for test data: 12.04



In [126]:
```
# plot the difference for the recent 30 obs of test data
pyplot.plot(testdiff[-30:])
#
pyplot.title("Predicted test data minus test data over recent 30 observations"
)
pyplot.ylabel('Difference')
pyplot.xlabel('Time range')
pyplot.show()
```

4 References

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

https://finance.yahoo.com/quote/BABA/history?period1=1366862400&period2=1524628800&interval=1d&fi