[View in Colaboratory]

```
import quandl
import datetime
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import datetime
from math import sqrt
```

```
df = quandl.get("NSE/MRF", start_date="2013-01-01", end_date="2018-05-18")
```

```
df.head()
```

Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
2013-01-01	12927.0	13380.00	12879.85	13350.0	13243.25	20619.0	2695.81
2013-01-02	13312.4	13435.00	13256.00	13295.0	13316.20	12217.0	1631.71
2013-01-03	13351.0	13365.95	13124.15	13273.0	13279.10	10213.0	1353.29
2013-01-04	13240.0	13418.40	13221.30	13365.0	13361.10	7307.0	973.81
2013-01-07	13375.0	13439.95	13265.00	13295.0	13288.80	7509.0	1001.57

```
df.tail()
```

Date	Open	High	Low	Last	Close	Total Trade Quantity	Turnover (Lacs)
2018-05-14	74750.0	75152.70	74515.45	74950.00	74737.35	3327.0	2490.00
2018-05-15	74850.0	75599.00	74341.65	74500.00	74604.95	4574.0	3422.04
2018-05-16	74500.0	75098.85	73978.05	74760.90	74873.40	7566.0	5659.91
2018-05-17	74803.4	75276.95	74400.00	74569.85	74559.95	4063.0	3034.76
2018-05-18	74555.0	75509.00	73925.10	74200.00	74206.20	5414.0	4034.38

```
df.columns
```

```
Index(['Open', 'High', 'Low', 'Last', 'Close', 'Total Trade Quantity',
       'Turnover (Lacs)'],
      dtype='object')
```

```
df.drop(df.columns[[3,5,6]], axis=1, inplace=True)
df.head()
```

Date	Open	High	Low	Close
2013-01-01	12927.0	13380.00	12879.85	13243.25
2013-01-02	13312.4	13435.00	13256.00	13316.20
2013-01-03	13351.0	13365.95	13124.15	13279.10
2013-01-04	13240.0	13418.40	13221.30	13361.10
2013-01-07	13375.0	13439.95	13265.00	13288.80

```
df['High'] = df['High'] / 100000
df['Open'] = df['Open'] / 100000
df['Low'] = df['Low'] / 100000
df['Close'] = df['Close'] / 100000
print(df.head())
print(df.tail())
```

```
                Open      High       Low     Close
Date
2013-01-01  0.129270  0.133800  0.128799  0.132433
2013-01-02  0.133124  0.134350  0.132560  0.133162
2013-01-03  0.133510  0.133660  0.131241  0.132791
2013-01-04  0.132400  0.134184  0.132213  0.133611
2013-01-07  0.133750  0.134400  0.132650  0.132888
```

```
                  Open      High       Low     Close
Date
2018-05-14  0.747500  0.751527  0.745154  0.747374
2018-05-15  0.748500  0.755990  0.743416  0.746049
2018-05-16  0.745000  0.750989  0.739781  0.748734
2018-05-17  0.748034  0.752769  0.744000  0.745599
2018-05-18  0.745550  0.755090  0.739251  0.742062
```

```
data = df.as_matrix()
```

```
data
```

```
array([[0.12927  , 0.1338   , 0.1287985, 0.1324325],
       [0.133124 , 0.13435  , 0.13256  , 0.133162 ],
       [0.13351  , 0.1336595, 0.1312415, 0.132791 ],
       ...,
       [0.745    , 0.7509885, 0.7397805, 0.748734 ],
       [0.748034 , 0.7527695, 0.744    , 0.7455995],
       [0.74555  , 0.75509  , 0.739251 , 0.742062 ]])
```

```
result = []
sequence_length = 6
for index in range(len(data) - sequence_length):
    result.append(data[index: index + sequence_length])

result = np.array(result)

row = round(0.8 * result.shape[0])

#creating training data
train = result[:int(row), :]

x_train = train[:, :-1]
y_train = train[:, -1][:,-1]
x_test = result[int(row):, :-1]
y_test = result[int(row):, -1][:,-1]

amount_of_features = len(df.columns)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], amount_of_features))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], amount_of_features))

print("X_train", x_train.shape)
print("y_train", y_train.shape)
print("X_test", x_test.shape)
print("y_test", y_test.shape)
```

```
X_train (1060, 5, 4)
y_train (1060,)
X_test (265, 5, 4)
y_test (265,)
```

```
from __future__ import print_function
import math
#importing keras modules
from keras.models import Sequential
from keras.layers import Dense, Activation ,Dropout , Flatten , Conv1D ,MaxPooling1D
from keras.layers.recurrent import LSTM
from keras import losses
from keras import optimizers
```

```
Using TensorFlow backend.
```

```
def build_model(input):
    model = Sequential()
    model.add(Dense(128,input_shape=(input[1],input[0])))
    model.add(Conv1D(filters = 112, kernel_size= 1,padding='valid', activation='relu', kernel_initializer="uniform"))
    model.add(MaxPooling1D(pool_size=2, padding='valid'))
    model.add(Conv1D(filters = 64,kernel_size = 1,padding='valid', activation='relu', kernel_initializer="uniform"))
    model.add(MaxPooling1D(pool_size=1, padding='valid'))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(100, activation="relu", kernel_initializer="uniform"))
    #model.add(Dropout(0.2))
    model.add(Dense(1, activation="relu", kernel_initializer="uniform"))
    model.compile(loss='mse',optimizer='adam',metrics=['mae'])
    return model
```

```
model = build_model([4,5,1])
#Summary of the Model
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 5, 128)            640
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 5, 112)            14448
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 2, 112)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 2, 64)             7232
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 2, 64)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 64)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               12900
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 35,321
Trainable params: 35,321
Non-trainable params: 0
_________________________________________________________________
None
```

```
from timeit import default_timer as timer
start = timer()
history = model.fit(x_train,
                    y_train,
                    batch_size=128,
                    epochs=25,
                    validation_split=0.2,
                    verbose=2)
end = timer()
print(end - start)
```

```
Train on 848 samples, validate on 212 samples
Epoch 1/25
 - 0s - loss: 0.0795 - mean_absolute_error: 0.2619 - val_loss: 0.1858 - val_mean_absolute_error: 0.4233
Epoch 2/25
 - 0s - loss: 0.0455 - mean_absolute_error: 0.1891 - val_loss: 0.0390 - val_mean_absolute_error: 0.1907
Epoch 3/25
 - 0s - loss: 0.0092 - mean_absolute_error: 0.0843 - val_loss: 0.0026 - val_mean_absolute_error: 0.0453
Epoch 4/25
 - 0s - loss: 0.0043 - mean_absolute_error: 0.0558 - val_loss: 0.0278 - val_mean_absolute_error: 0.1598
Epoch 5/25
 - 0s - loss: 0.0054 - mean_absolute_error: 0.0595 - val_loss: 0.0110 - val_mean_absolute_error: 0.0973
Epoch 6/25
 - 0s - loss: 0.0027 - mean_absolute_error: 0.0438 - val_loss: 0.0010 - val_mean_absolute_error: 0.0268
Epoch 7/25
 - 0s - loss: 0.0023 - mean_absolute_error: 0.0408 - val_loss: 0.0091 - val_mean_absolute_error: 0.0881
Epoch 8/25
 - 0s - loss: 0.0019 - mean_absolute_error: 0.0372 - val_loss: 0.0048 - val_mean_absolute_error: 0.0614
Epoch 9/25
 - 0s - loss: 0.0015 - mean_absolute_error: 0.0333 - val_loss: 0.0022 - val_mean_absolute_error: 0.0388
Epoch 10/25
 - 0s - loss: 0.0013 - mean_absolute_error: 0.0310 - val_loss: 0.0039 - val_mean_absolute_error: 0.0558
Epoch 11/25
 - 0s - loss: 0.0011 - mean_absolute_error: 0.0276 - val_loss: 0.0020 - val_mean_absolute_error: 0.0381
Epoch 12/25
 - 0s - loss: 9.8639e-04 - mean_absolute_error: 0.0266 - val_loss: 0.0018 - val_mean_absolute_error: 0.0360
Epoch 13/25
 - 0s - loss: 7.7335e-04 - mean_absolute_error: 0.0228 - val_loss: 0.0013 - val_mean_absolute_error: 0.0302
Epoch 14/25
 - 0s - loss: 6.0797e-04 - mean_absolute_error: 0.0202 - val_loss: 0.0013 - val_mean_absolute_error: 0.0307
Epoch 15/25
 - 0s - loss: 5.9640e-04 - mean_absolute_error: 0.0192 - val_loss: 6.8688e-04 - val_mean_absolute_error: 0.0211
Epoch 16/25
 - 0s - loss: 5.4487e-04 - mean_absolute_error: 0.0181 - val_loss: 5.8812e-04 - val_mean_absolute_error: 0.0195
Epoch 17/25
 - 0s - loss: 4.1810e-04 - mean_absolute_error: 0.0154 - val_loss: 4.7904e-04 - val_mean_absolute_error: 0.0175
Epoch 18/25
 - 0s - loss: 4.6870e-04 - mean_absolute_error: 0.0155 - val_loss: 3.0016e-04 - val_mean_absolute_error: 0.0138
Epoch 19/25
 - 0s - loss: 4.4264e-04 - mean_absolute_error: 0.0155 - val_loss: 5.8311e-04 - val_mean_absolute_error: 0.0197
Epoch 20/25
 - 0s - loss: 4.5042e-04 - mean_absolute_error: 0.0158 - val_loss: 3.3479e-04 - val_mean_absolute_error: 0.0145
Epoch 21/25
 - 0s - loss: 4.7675e-04 - mean_absolute_error: 0.0159 - val_loss: 5.1621e-04 - val_mean_absolute_error: 0.0183
Epoch 22/25
 - 0s - loss: 4.8802e-04 - mean_absolute_error: 0.0163 - val_loss: 3.3061e-04 - val_mean_absolute_error: 0.0144
Epoch 23/25
```

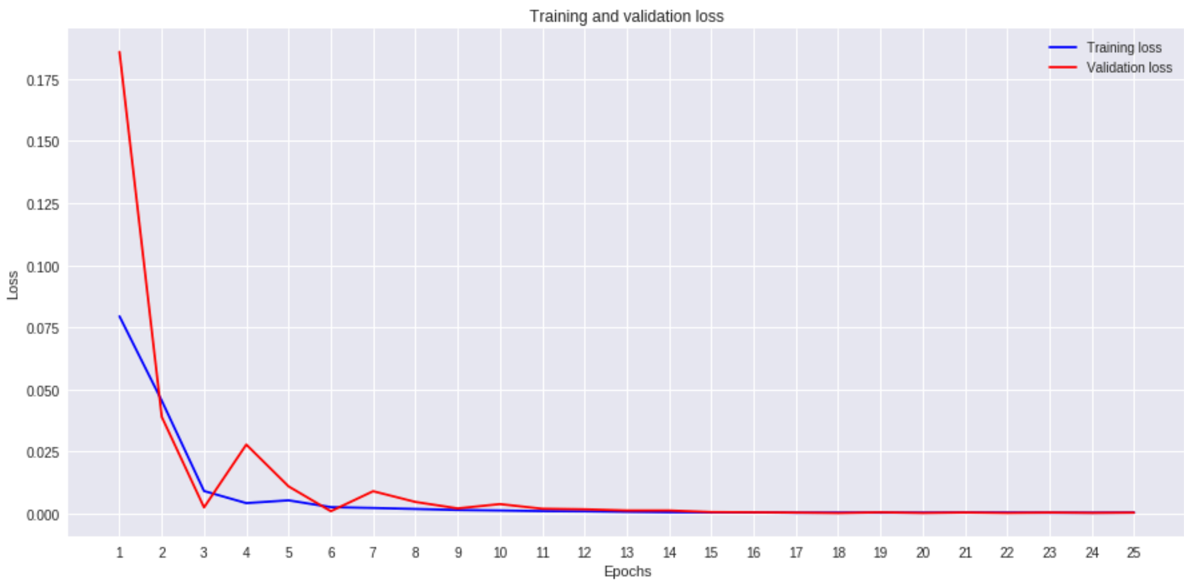
```
 - 0s - loss: 4.1876e-04 - mean_absolute_error: 0.0150 - val_loss: 4.9814e-04 - val_mean_absolute_error: 0.0180
Epoch 24/25
 - 0s - loss: 4.1497e-04 - mean_absolute_error: 0.0150 - val_loss: 3.6232e-04 - val_mean_absolute_error: 0.0151
Epoch 25/25
 - 0s - loss: 4.6069e-04 - mean_absolute_error: 0.0158 - val_loss: 4.6297e-04 - val_mean_absolute_error: 0.0172
2.4056871799999726
```

```
history_dict = history.history
history_dict.keys()
```

```
dict_keys(['val_loss', 'val_mean_absolute_error', 'loss', 'mean_absolute_error'])
```

```
import matplotlib.pyplot as plt

loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
loss_values50 = loss_values[0:150]
val_loss_values50 = val_loss_values[0:150]
epochs = range(1, len(loss_values50) + 1)
plt.plot(epochs, loss_values50, 'b',color = 'blue', label='Training loss')
plt.plot(epochs, val_loss_values50, 'b',color='red', label='Validation loss')
plt.rc('font', size = 18)
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.xticks(epochs)
fig = plt.gcf()
fig.set_size_inches(15,7)
#fig.savefig('img/25/mrftest&validationlosscnn.png', dpi=300)
plt.show()
```




```
mae = history_dict['mean_absolute_error']
vmae = history_dict['val_mean_absolute_error']
epochs = range(1, len(mae) + 1)
plt.plot(epochs, mae, 'b',color = 'blue', label='Training error')
plt.plot(epochs, vmae, 'b',color='red', label='Validation error')
plt.title('Training and validation error')
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.legend()
plt.xticks(epochs)
fig = plt.gcf()
fig.set_size_inches(15,7)
#fig.savefig('img/25/mrftest&validationerrorcnn.png', dpi=300)
plt.show()
```


```
model.metrics_names
```

```
['loss', 'mean_absolute_error']
```

```
trainScore = model.evaluate(x_train, y_train, verbose=0)
testScore = model.evaluate(x_test, y_test, verbose=0)
```

```
#predicting values for y_test
p = model.predict(x_test)
```

```
plt.plot(p,color='red', label='prediction')
plt.plot(y_test,color='blue', label='y_test')
plt.xlabel('No. of Trading Days')
plt.ylabel('Close Value (scaled)')
plt.legend(loc='upper left')
fig = plt.gcf()
fig.set_size_inches(15, 5)
#fig.savefig('img/25/mrftestcnn.png', dpi=300)
plt.show()
```


```
p1= model.predict(x_train)
```

```
plt.plot(p1[:848],color='red', label='prediction on training samples')
x = np.array(range(848,1060))
plt.plot(x,p1[848:],color = 'magenta',label ='prediction on validating samples')
plt.plot(y_train,color='blue', label='y_train')
plt.xlabel('No. of Trading Days')
plt.ylabel('Close Value (scaled)')
plt.legend(loc='upper left')
fig = plt.gcf()
fig.set_size_inches(20,10)
#fig.savefig('img/25/mrftraincnn.png', dpi=300)
plt.show()
```


```
y = y_test * 100000
y_pred = p.reshape(265)
y_pred = y_pred * 100000
```

```
from sklearn.metrics import mean_absolute_error
```

```
print('Trainscore RMSE \tTrain Mean abs Error \tTestscore Rmse \t Test Mean abs Error')
print('%.9f \t\t %.9f \t\t %.9f \t\t %.9f' % (math.sqrt(trainScore[0]),trainScore[1],math.sqrt(testScore[0]),testScore[1]))
```

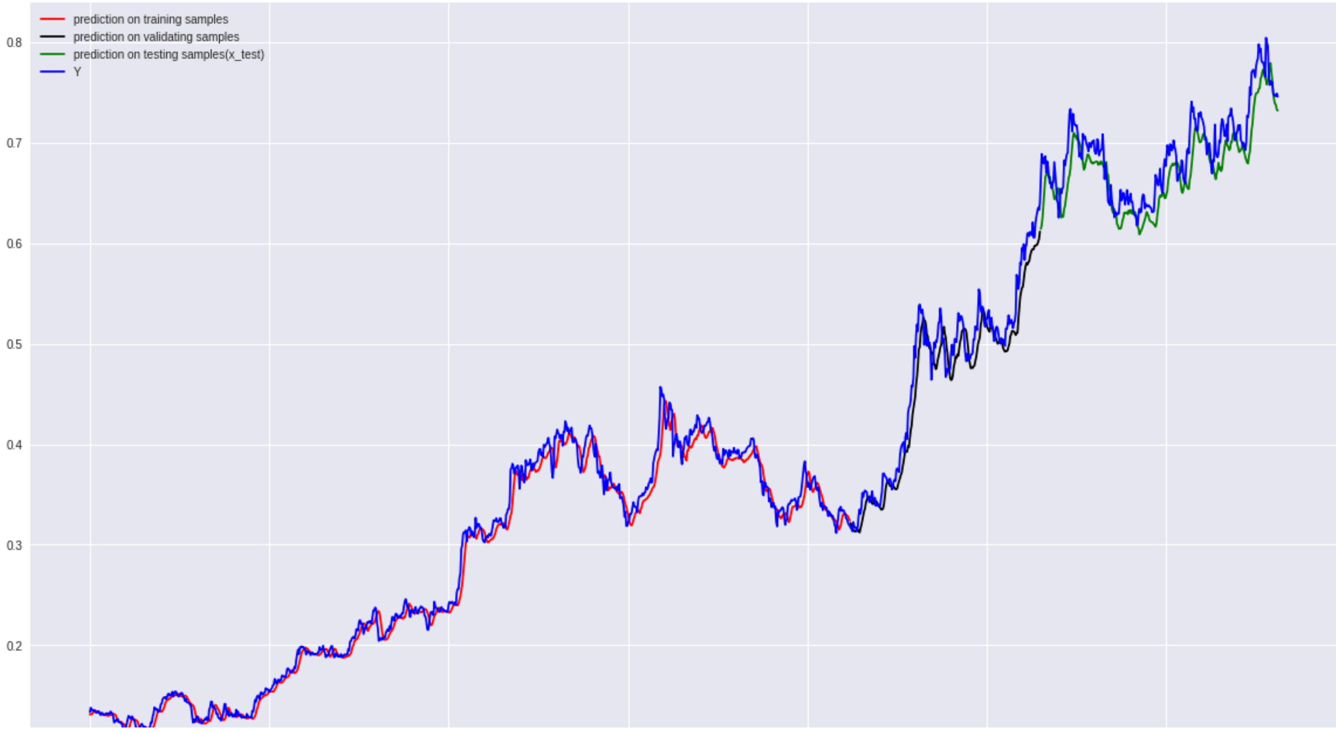
```
Trainscore RMSE         Train Mean abs Error    Testscore Rmse    Test Mean abs Error
0.013779589              0.009754905             0.024719553            0.019698331
```

```
print('mean absolute error \t mean absolute percentage error')
print(' %.9f \t\t\t %.9f' % (mean_absolute_error(y,y_pred),(np.mean(np.abs((y - y_pred) / y)) * 100)))
```

```
mean absolute error      mean absolute percentage error
 1969.833048938                         2.819180747
```

```
Y = np.concatenate((y_train,y_test),axis = 0)
P = np.concatenate((p1,p),axis = 0)
#plotting the complete Y set with predicted values on x_train and x_test(variable p1 & p respectively given above)
#for
plt.plot(P[:848],color='red', label='prediction on training samples')
#for validating samples
z = np.array(range(848,1060))
plt.plot(z,P[848:1060],color = 'black',label ='prediction on validating samples')
#for testing samples
x = np.array(range(1060,1325))
plt.plot(x,P[1060:],color = 'green',label ='prediction on testing samples(x_test)')

plt.plot(Y,color='blue', label='Y')
plt.legend(loc='upper left')
fig = plt.gcf()
fig.set_size_inches(20,12)
plt.show()
```


prediction on training samples
prediction on validating samples
prediction on testing samples(x_test)
Y
0.8
0.7
0.6
0.5
0.4
0.3
0.2