In [1]:
```python
import pandas as pd
import numpy as np

import tensorflow as tf

import keras

import matplotlib.pyplot as plt
import seaborn as sns
```

```
Using TensorFlow backend.
```

In [2]:
```python
ibm_stock_data = pd.read_csv('../data/ibm_stock_prices.csv')
```

In [3]:
```python
ibm_stock_data.shape
```

Out[3]:
```
(14059, 7)
```

In [4]:
```python
ibm_stock_data.head()
```

Out[4]:

	Date	Open	High	Low	Close	Volume	OpenInt
0	1962-01-02	6.4130	6.4130	6.3378	6.3378	467056	0
1	1962-01-03	6.3378	6.3963	6.3378	6.3963	350294	0
2	1962-01-04	6.3963	6.3963	6.3295	6.3295	314365	0
3	1962-01-05	6.3211	6.3211	6.1958	6.2041	440112	0
4	1962-01-08	6.2041	6.2041	6.0373	6.0870	655676	0

In [5]:
```python
ibm_stock_data.drop('OpenInt', axis=1, inplace=True)
```

In [6]:
```
ibm_stock_data.isnull().any()
```

Out[6]:
```
Date      False
Open      False
High      False
Low       False
Close     False
Volume    False
dtype: bool
```

Convert data to number

In [7]:
```
ibm_stock_data.dtypes
```

Out[7]:
```
Date       object
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
dtype: object
```

In [8]:
```
ibm_stock_data['Date'] = ibm_stock_data['Date'].astype('datetime64')
```

In [9]:
```
ibm_stock_data.head()
```

Out[9]:

	Date	Open	High	Low	Close	Volume
0	1962-01-02	6.4130	6.4130	6.3378	6.3378	467056
1	1962-01-03	6.3378	6.3963	6.3378	6.3963	350294
2	1962-01-04	6.3963	6.3963	6.3295	6.3295	314365
3	1962-01-05	6.3211	6.3211	6.1958	6.2041	440112
4	1962-01-08	6.2041	6.2041	6.0373	6.0870	655676

In [10]:
```
ibm_stock_data.dtypes
```

Out[10]:
```
Date      datetime64[ns]
Open             float64
High             float64
Low              float64
Close            float64
Volume             int64
dtype: object
```

In [11]:
```
min_date = ibm_stock_data['Date'].min()
print("min_date : ", min_date)
```

```
min_date :  1962-01-02 00:00:00
```

In [12]:
```
max_date = ibm_stock_data['Date'].max()
print("max_date : ", max_date)
```

```
max_date :  2017-11-10 00:00:00
```

In [13]:
```
def calculate_days_since_min_date(row_date):
    return row_date - min_date
```

In [14]:
```
ibm_stock_data['numeric_date'] = ibm_stock_data['Date'].apply(lambda x:calculate_days_since_min_date(x))
```

In [15]:
```
ibm_stock_data.head()
```

Out[15]:

	Date	Open	High	Low	Close	Volume	numeric_date
0	1962-01-02	6.4130	6.4130	6.3378	6.3378	467056	0 days
1	1962-01-03	6.3378	6.3963	6.3378	6.3963	350294	1 days
2	1962-01-04	6.3963	6.3963	6.3295	6.3295	314365	2 days
3	1962-01-05	6.3211	6.3211	6.1958	6.2041	440112	3 days
4	1962-01-08	6.2041	6.2041	6.0373	6.0870	655676	6 days

In [16]:
```python
ibm_stock_data.dtypes
```

Out[16]:
```
Date             datetime64[ns]
Open                    float64
High                    float64
Low                     float64
Close                   float64
Volume                    int64
numeric_date    timedelta64[ns]
dtype: object
```

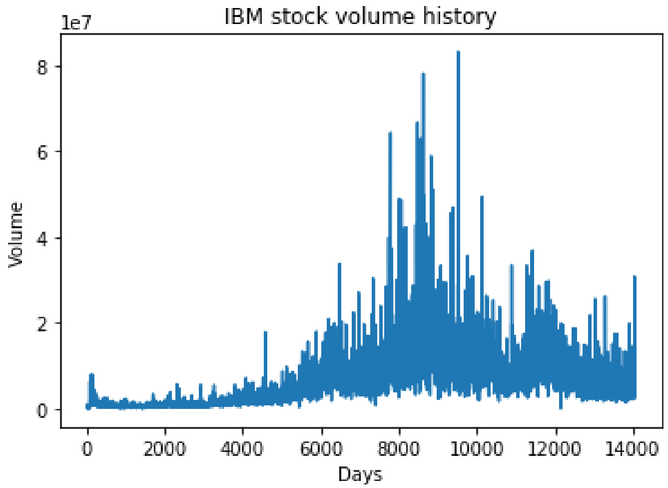
In [17]:
```python
ibm_stock_data['numeric_date'] = ibm_stock_data['numeric_date'].astype('timedelta64[D]')
```

In [18]:
```python
ibm_stock_data.dtypes
```

Out[18]:
```
Date            datetime64[ns]
Open                   float64
High                   float64
Low                    float64
Close                  float64
Volume                   int64
numeric_date           float64
dtype: object
```

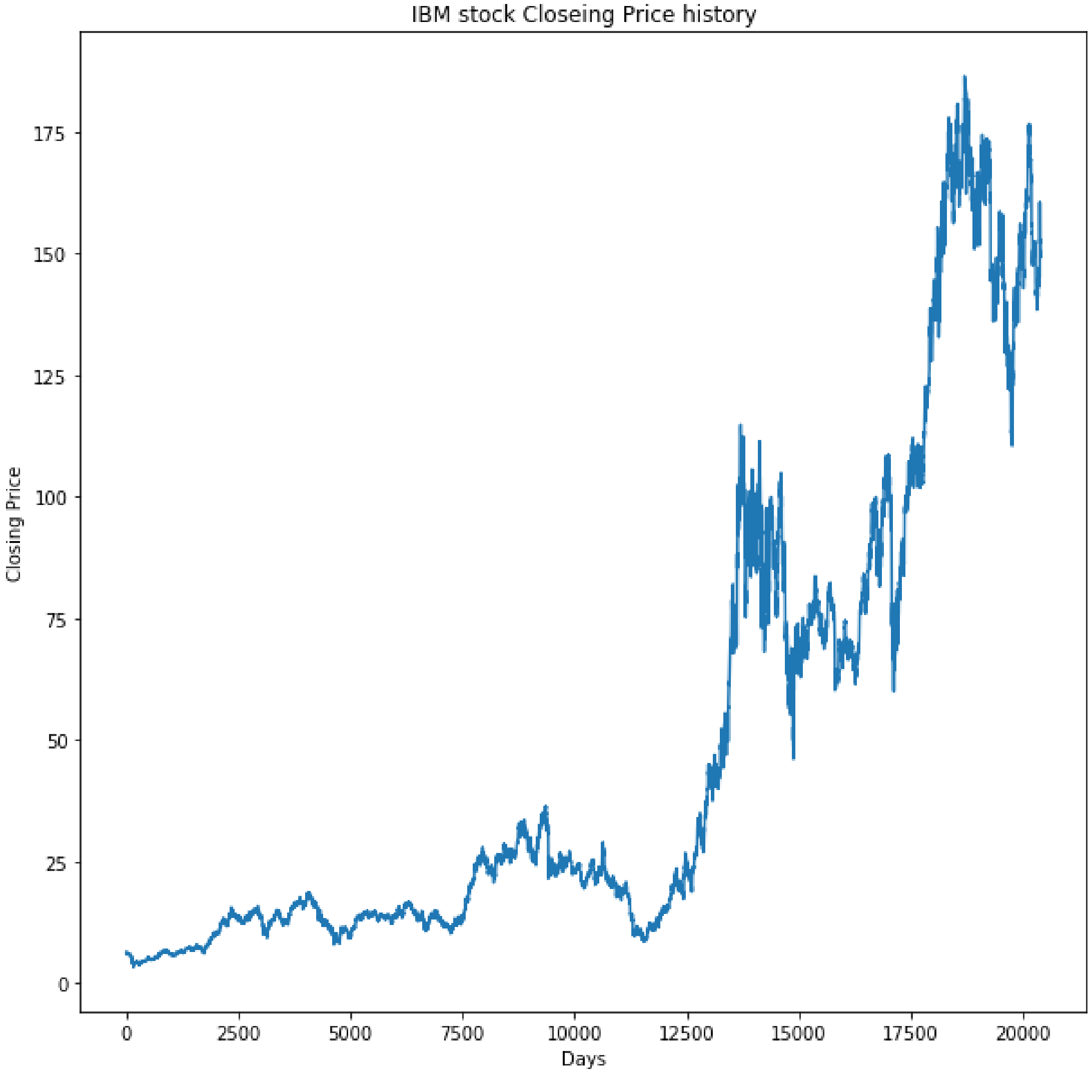
In [19]:

```
plt.figure()
plt.plot(ibm_stock_data["Volume"])
plt.title('IBM stock volume history')
plt.ylabel('Volume')
plt.xlabel('Days')
plt.show()
```



In [20]:

```
plt.figure(figsize=(10,10))
plt.plot(ibm_stock_data["numeric_date"], ibm_stock_data["Close"])
plt.title('IBM stock Closeing Price history')
plt.ylabel('Closing Price')
plt.xlabel('Days')
plt.show()
```



In [21]:
```python
ibm_stock_data.drop('Date', axis=1, inplace=True)
```

In [22]:
```python
ibm_stock_data['numeric_date'].min()
```

Out[22]:
```
0.0
```

In [23]:
```python
ibm_stock_data['numeric_date'].max()
```

Out[23]:
```
20401.0
```

In [24]:
```python
TIME_STEPS = 60
```

In [25]:
```python
stock_train_set = ibm_stock_data[ibm_stock_data.numeric_date <= 19000.0]
stock_test_set = ibm_stock_data[ibm_stock_data.numeric_date > 19000.0]
```

In [26]:
```python
stock_train_set.shape
```

Out[26]:
```
(13091, 6)
```

In [27]:
```python
stock_train_set = stock_train_set.iloc[0:13020]
stock_test_set = stock_test_set.iloc[0:900]
```

In [28]:
```python
stock_train_set.shape
```

Out[28]:
```
(13020, 6)
```

In [29]:
```python
#stock_test_set[:100]
```

In [30]:
```python
stock_test_set.shape
```

Out[30]:
```
(900, 6)
```

In [31]:
```python
X_train_orig = stock_train_set[['Open', 'High', 'Low', 'Volume', 'numeric_date']].values
#X_train = stock_train_set[['Open', 'High', 'Low', 'Volume']].values
#X_train = stock_train_set[['Open', 'High', 'Low', 'Volume', 'numeric_date']].values
y_train = stock_train_set['Close'].values
```

In [32]:
```
X_test_orig = stock_test_set[['Open', 'High', 'Low', 'Volume', 'numeric_date']].values
#X_test = stock_test_set[['Open', 'High', 'Low', 'Volume']].values
#X_test = stock_test_set[['Open', 'High', 'Low', 'Volume', 'numeric_date']].values
y_test = stock_test_set['Close'].values
```

In [33]:
```
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()
X_train = sc.fit_transform(X_train_orig)
X_test = sc.transform(X_test_orig)
```

In [34]:
```
X_test.shape
```

Out[34]:
```
(900, 5)
```

In [35]:
```
a = [0,1,2,3,4,5,6,7,8,9]
print(a[:-2])
print(a[2:])
print('(0,1:2), (1,2:3), (2,3:4), (3,4:5), (4,5:6), (5,6:7),(6,7:8), (7,8:9)')
```

```
[0, 1, 2, 3, 4, 5, 6, 7]
[2, 3, 4, 5, 6, 7, 8, 9]
(0,1:2), (1,2:3), (2,3:4), (3,4:5), (4,5:6), (5,6:7),(6,7:8), (7,8:9)
```

In [36]:
```python
# Each TIME_STEP is a row 'Open', 'High', 'Low', 'Volume', 'numeric_date', we are considering 4 of these to predict 5th.
# Hence number of TIME_STEPS = 4

def build_time_series_data(x, y):

    #iters = x.shape[0]//TIME_STEPS

    row_dim = TIME_STEPS
    col_dim = x.shape[1]
    third_dim = x.shape[0] - TIME_STEPS

    print(third_dim, row_dim, col_dim)

    x_time_series_data = np.zeros((third_dim, row_dim, col_dim))
    y_time_series_data = np.zeros((third_dim))
    num_date_corresponding_to_y = np.zeros((third_dim))

    for i in range(third_dim):
        x_time_series_data[i] = x[i:TIME_STEPS+i]
        y_time_series_data[i] = y[TIME_STEPS+i]


        num_date_corresponding_to_y[i] = sc.inverse_transform(x)[TIME_STEPS+i][4]  # as 5th columns in X is the "numeric_date"

    return x_time_series_data, y_time_series_data, num_date_corresponding_to_y
```

In [37]:
```python
x_train_time_series_data, y_train_time_series_data, train_num_date_corresponding_to_y = build_time_series_data(X_train, y_train)
x_test_time_series_data, y_test_time_series_data, test_num_date_corresponding_to_y = build_time_series_data(X_test, y_test)
```

```
12960 60 5
840 60 5
```

In [38]:
```python
y_test_time_series_data[y_test_time_series_data >0].shape
```

Out[38]:
```
(840,)
```

In [39]:
```python
y_train_time_series_data[y_train_time_series_data!=0].shape
```

Out[39]:
```
(12960,)
```

In [40]:

```python
from keras import Sequential
from keras.layers import GRU
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers.core import Dropout
from keras.layers.core import Flatten
from keras.layers import GRU
from keras import optimizers

from keras import regularizers
```

In [41]:

```python
BATCH_SIZE = 60

t_row_dim = TIME_STEPS
t_col_dim = X_train.shape[1]
t_third_dim = X_train.shape[0] - TIME_STEPS
```

In [42]:

```
rnn_model = Sequential()
rnn_model.add(GRU(3, return_sequences=True, input_shape=(t_row_dim, t_col_dim) ))
#rnn_model.add(Dropout(0.7))

rnn_model.add(GRU(3, return_sequences=False))
#rnn_model.add(Dropout(0.7))


#rnn_model.add(Flatten())
#rnn_model.add(Dense(25))
#rnn_model.add(Dropout(0.6))

rnn_model.add(Dense(5))
rnn_model.add(Dropout(0.3))

rnn_model.add(Dense(1))

optimizer = optimizers.Adam(lr=0.7)
rnn_model.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

```
WARNING:tensorflow:From C:\Users\thisi\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\framework\op_def_libr
ary.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\thisi\AppData\Local\Continuum\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:344
5: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

In [ ]:

```
```

In [43]:

```
rnn_model.fit(x_train_time_series_data, y_train_time_series_data,
          batch_size=BATCH_SIZE, epochs=1000, shuffle=False,
          validation_data=(x_test_time_series_data, y_test_time_series_data), verbose=2)
```

```
Epoch 849/1000
 - 23s - loss: 133.4733 - val_loss: 37.4652

Epoch 850/1000
 - 23s - loss: 145.5879 - val_loss: 36.4476
Epoch 851/1000
 - 23s - loss: 128.7842 - val_loss: 59.1308
Epoch 852/1000
 - 22s - loss: 137.5519 - val_loss: 70.2305
Epoch 853/1000
 - 23s - loss: 134.0128 - val_loss: 69.4804
Epoch 854/1000
 - 23s - loss: 133.6396 - val_loss: 77.5564
Epoch 855/1000
 - 23s - loss: 141.4769 - val_loss: 30.5555
Epoch 856/1000
 - 22s - loss: 134.5154 - val_loss: 158.0352
Epoch 857/1000
 - 22s - loss: 140.4537 - val_loss: 37.4736
Epoch 858/1000
```

In [44]:

```
rnn_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_1 (GRU)                  (None, 60, 3)             81
_________________________________________________________________
gru_2 (GRU)                  (None, 3)                 63
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 20
_________________________________________________________________
dropout_1 (Dropout)          (None, 5)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 6
=================================================================
Total params: 170
Trainable params: 170
Non-trainable params: 0
_________________________________________________________________
```

In [45]:
```
rnn_model.save('./models/RNN_GRU_stock_price_prediction_model.h5')
```

In [46]:
```
train_predict= rnn_model.predict(x_train_time_series_data)
```

In [47]:
```
x_train_time_series_data.shape
```

Out[47]:
```
(12960, 60, 5)
```

In [48]:
```
train_num_date_corresponding_to_y.shape
```

Out[48]:
```
(12960,)
```

In [49]:
```
y_train_time_series_data
```

Out[49]:
```
array([  5.9787,   5.9622,   5.9037, ..., 165.22  , 165.88  , 163.    ])
```

In [50]:
```
y_train_time_series_data
```

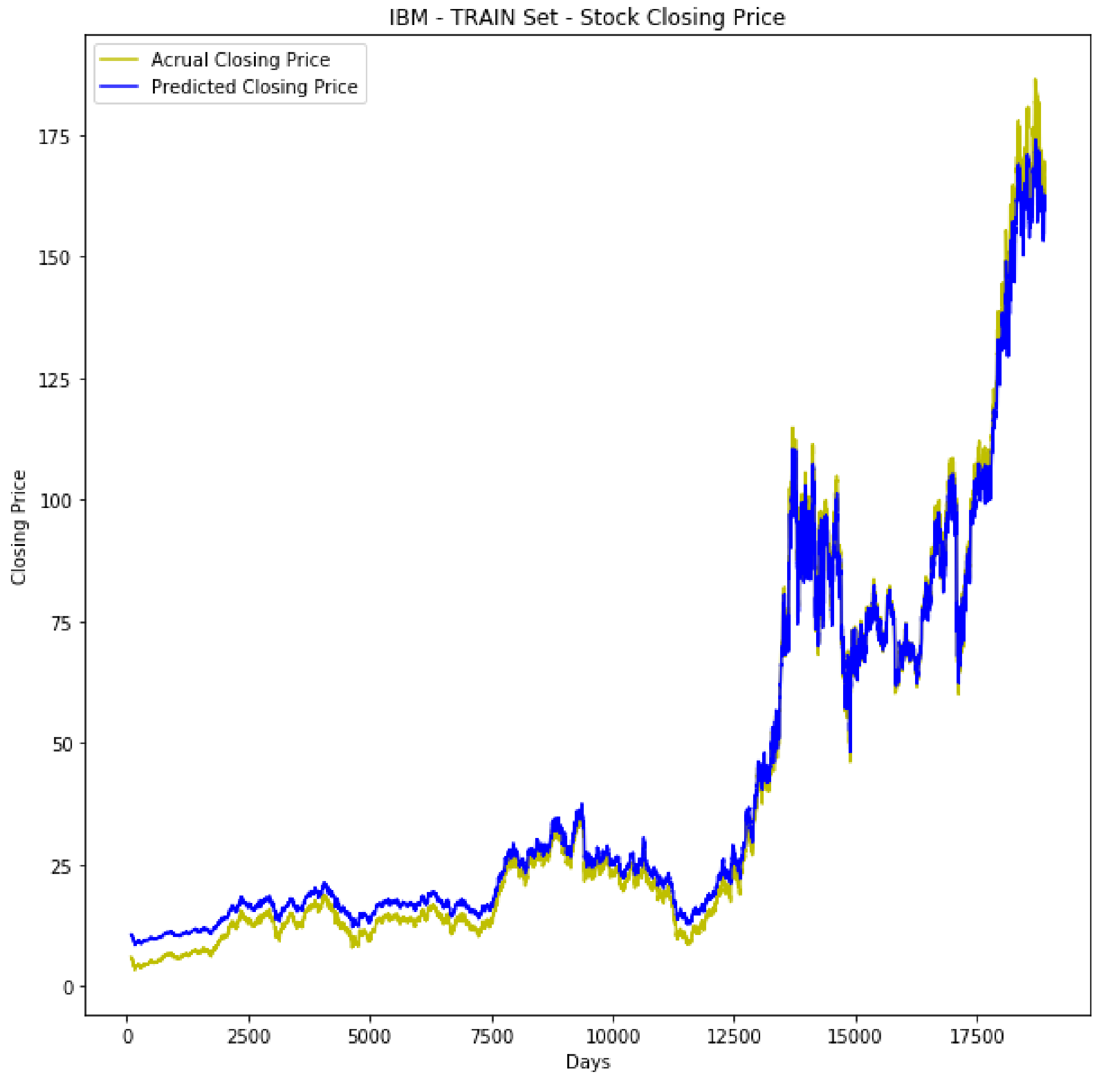
Out[50]:
```
array([  5.9787,   5.9622,   5.9037, ..., 165.22  , 165.88  , 163.    ])
```

In [64]:

```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax.set_title('IBM - TRAIN Set - Stock Closing Price')
#ax.scatter(x=data[:,0],y=data[:,1],label='Data')

plt.plot(train_num_date_corresponding_to_y, y_train_time_series_data, color='y', label='Acrual Closing Price')
plt.plot(train_num_date_corresponding_to_y, train_predict.ravel(),color='b', label='Predicted Closing Price')

#plt.plot(data[:,0], m*data[:,0] + b,color='red',label='Our Fitting Line')
ax.set_xlabel('Days')
ax.set_ylabel('Closing Price')
ax.legend(loc='best')
plt.show()
```



In [62]:

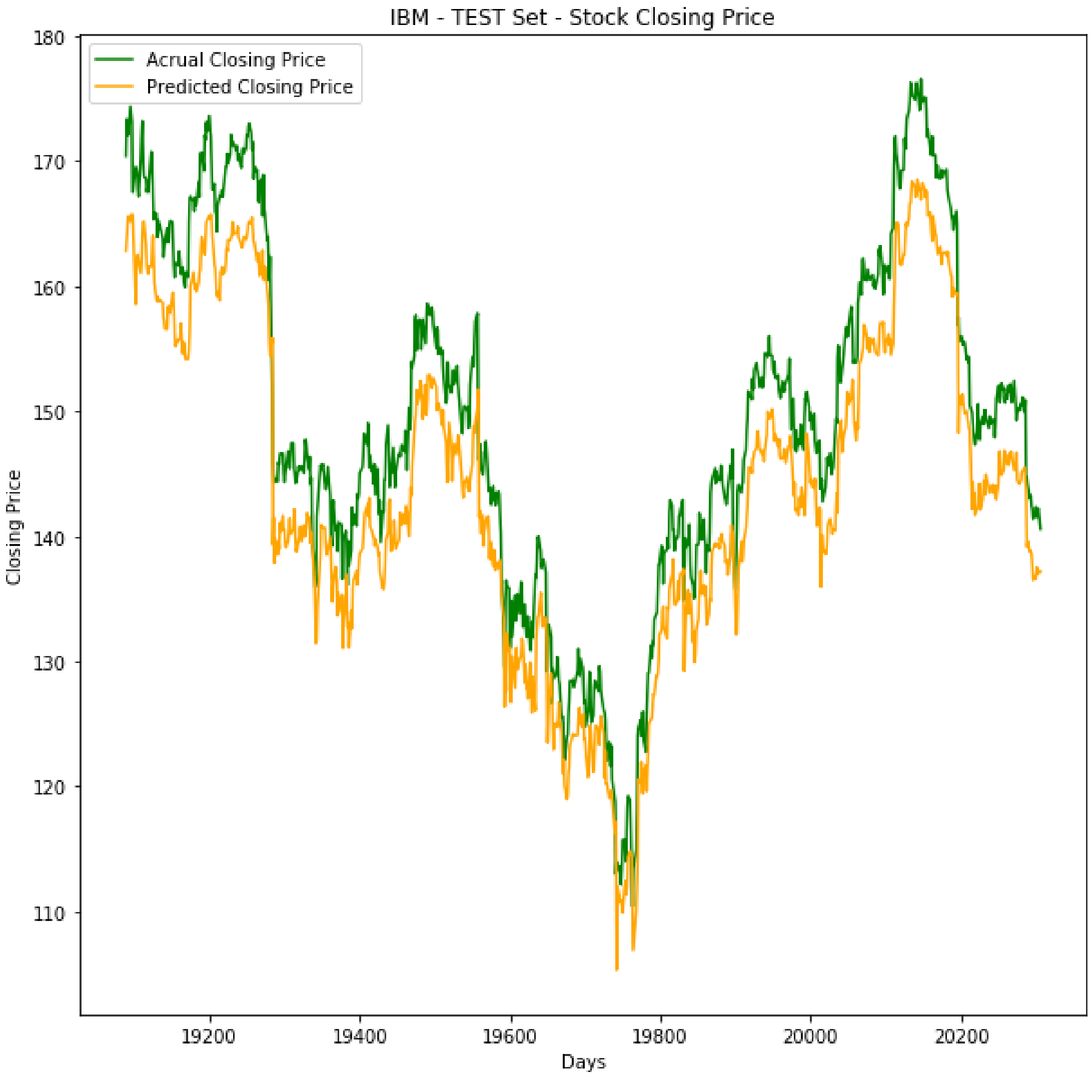
```
test_predict= rnn_model.predict(x_test_time_series_data)
```

In [65]:

```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)
ax.set_title('IBM - TEST Set - Stock Closing Price')
#ax.scatter(x=data[:,0],y=data[:,1],label='Data')

plt.plot(test_num_date_corresponding_to_y, y_test_time_series_data, color='g', label='Acrual Closing Price')
plt.plot(test_num_date_corresponding_to_y, test_predict.ravel(),color='orange', label='Predicted Closing Price')

#plt.plot(data[:,0], m*data[:,0] + b,color='red',label='Our Fitting Line')
ax.set_xlabel('Days')
ax.set_ylabel('Closing Price')
ax.legend(loc='best')
plt.show()
```



https://www.dlology.com/blog/how-to-use-return_state-or-return_sequences-in-keras/ (https://www.dlology.com/blog/how-to-use-return_state-or-return_sequences-in-keras/)

```
In [ ]:
```