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

In [3]:
```
df=pd.read_csv("C:/Users/Preetha/OneDrive/Desktop/Stock.csv")
df.head()
```

Out[3]:

	Unnamed: 0	symbol	date	close	high	low	open	volume	adjClose	a
0	0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.682558	121.8
1	1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.438354	121.5
2	2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.90	131.23	50884452	120.056069	121.1
3	3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.05	131.20	32112797	120.291057	121.0
4	4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.32	129.86	33667627	119.761181	120.4

In [4]:
```
df.describe()
```

Out[4]:

	Unnamed: 0	close	high	low	open	volume	adjClose
count	1258.000000	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000
mean	628.500000	167.723998	169.230475	166.039780	167.548266	3.500397e+07	162.666715
std	363.297628	56.850796	57.500128	56.006773	56.612707	1.729100e+07	58.733820
min	0.000000	90.340000	91.670000	89.470000	90.000000	1.136204e+07	84.954351
25%	314.250000	116.327500	117.405000	115.602500	116.482500	2.359205e+07	109.484490
50%	628.500000	160.485000	162.080000	158.974250	160.345000	3.064771e+07	154.710645
75%	942.750000	199.785000	201.277500	198.170000	199.520000	4.100487e+07	196.960053
max	1257.000000	327.200000	327.850000	323.350000	324.730000	1.622063e+08	326.337147

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1258 entries, 0 to 1257
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   1258 non-null   int64
 1   symbol       1258 non-null   object
 2   date         1258 non-null   object
 3   close        1258 non-null   float64
 4   high         1258 non-null   float64
 5   low          1258 non-null   float64
 6   open         1258 non-null   float64
 7   volume       1258 non-null   int64
 8   adjClose     1258 non-null   float64
 9   adjHigh      1258 non-null   float64
 10  adjLow       1258 non-null   float64
 11  adjOpen      1258 non-null   float64
 12  adjVolume    1258 non-null   int64
 13  divCash      1258 non-null   float64
 14  splitFactor  1258 non-null   float64
dtypes: float64(10), int64(3), object(2)
memory usage: 147.5+ KB
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Unnamed: 0     0
symbol         0
date           0
close          0
high           0
low            0
open           0
volume         0
adjClose       0
adjHigh        0
adjLow         0
adjOpen        0
adjVolume      0
divCash        0
splitFactor    0
dtype: int64
```

In [7]:
```
df.shape
```

Out[7]:
```
(1258, 15)
```

In [8]:
```
df1=df.reset_index()["close"]
```

In [9]:
```
df1
```

Out[9]:
```
0       132.045
1       131.780
2       130.280
3       130.535
```

In [9]:
```
df1
```

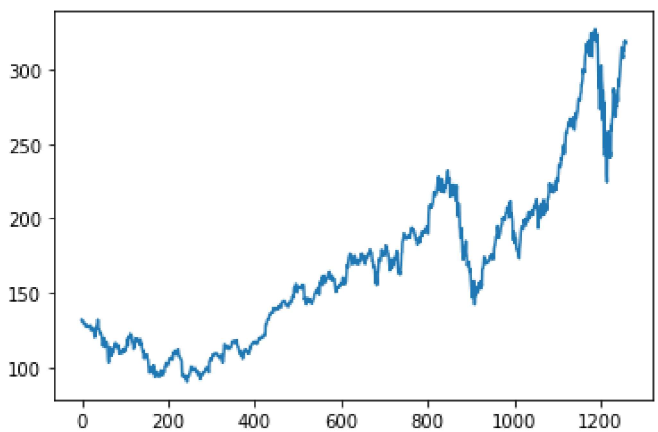
Out[9]:
```
0       132.045
1       131.780
2       130.280
3       130.535
4       129.960
          ...
1253    314.960
1254    313.140
1255    319.230
1256    316.850
1257    318.890
Name: close, Length: 1258, dtype: float64
```

In [10]:
```
df1.shape
```

Out[10]:
```
(1258,)
```

In [11]:
```
import matplotlib.pyplot as plt
plt.plot(df1)
```

Out[11]:
```
[<matplotlib.lines.Line2D at 0x1f9cae9de50>]
```



Data Preprocessing

In [12]:
```
#LSTM are sensitive to the scaled data, so we apply minmax scaler
from sklearn.preprocessing import MinMaxScaler
#to set the range of values between 0 and 1
scaler=MinMaxScaler(feature_range=(0,1))
df1=scaler.fit_transform(np.array(df1).reshape(-1,1))
```

In [13]:
```
df1.shape
```

Out[13]:
```
(1258, 1)
```

Splitting the dataset into training and testing set

In [14]:
```
#splitting the dataset into training and test consecutively as it is an time seri
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

In [15]:
```
training_size,test_size
```

Splitting the dataset into training and testing set

In [14]:
```
#splitting the dataset into training and test consecutively as it is an time seri
training_size=int(len(df1)*0.65)
test_size=len(df1)-training_size
train_data,test_data=df1[0:training_size,:],df1[training_size:len(df1),:1]
```

In [15]:
```
training_size,test_size
```

Out[15]:
```
(817, 441)
```

In [16]:
```
train_data
```

```
       [0.10512539],
       [0.10474542],
       [0.10816516],
       [0.11323144],
       [0.11044499],
       [0.10415435],
       [0.09419066],
       [0.06510175],
       [0.05395592],
       [0.0565735 ],
       [0.08169383],
       [0.09533058],
       [0.09689268],
       [0.09465507],
       [0.07337668],
       [0.09288187],
       [0.08456472],
       [0.07992063],
       [0.09275521],
       [0.0836359 ],
```

In [17]:
```
test_data
```

```
       [0.56831039],
       [0.5716457 ],
       [0.57806299],
       [0.58659124],
       [0.59837035],
       [0.58114498],
       [0.56552394],
       [0.56332855],
       [0.57641645],
       [0.53204425],
       [0.52398041],
       [0.55632019],
       [0.53626615],
       [0.55648907],
       [0.55243604],
       [0.5306088 ],
       [0.54449886],
       [0.55015621],
       [0.55893777],
```

In [18]:
```
def create_dataset(dataset,time_step=1):
    #Convert an array of values into a dataset matrix
    dataX,dataY=[],[]
    for i in range(len(dataset)-time_step-1):
        a=dataset[i:(i+time_step),0]   #i=0 then values will be from "i" to "n-1'
        dataX.append(a)
```

In [18]:
```
def create_dataset(dataset,time_step=1):
    #Convert an array of values into a dataset matrix
    dataX,dataY=[],[]
    for i in range(len(dataset)-time_step-1):
        a=dataset[i:(i+time_step),0]   #i=0 then values will be from "i" to "n-1'
        dataX.append(a)
        dataY.append(dataset[i+time_step,0])
    return np.array(dataX) , np.array(dataY)
```

In [19]:
```
#reshape into X=t,t+1,t+2,t+3 and y=t+4
time_step=100
X_train,y_train = create_dataset(train_data,time_step)
X_test,y_test = create_dataset(test_data,time_step)
```

In [20]:
```
X_train
```

Out[20]:
```
array([[0.17607447, 0.17495567, 0.16862282, ..., 0.09055982, 0.08388922,
        0.09085536],
       [0.17495567, 0.16862282, 0.1696994 , ..., 0.08388922, 0.09085536,
        0.0873934 ],
       [0.16862282, 0.1696994 , 0.16727181, ..., 0.09085536, 0.0873934 ,
        0.09030651],
       ...,
       [0.34801148, 0.32930845, 0.32145571, ..., 0.50042219, 0.50413747,
        0.5062062 ],
       [0.32930845, 0.32145571, 0.32694419, ..., 0.50413747, 0.5062062 ,
        0.51920966],
       [0.32145571, 0.32694419, 0.32230009, ..., 0.5062062 , 0.51920966,
        0.53719497]])
```

In [21]:
```
X_train.shape, y_train.shape
```

Out[21]:
```
((716, 100), (716,))
```

In [22]:
```
X_test.shape , y_test.shape
```

Out[22]:
```
((340, 100), (340,))
```

In [23]:
```
#reshape the input to be [samples , time_step , features] which is required for L
X_train = X_train.reshape(X_train.shape[0] , X_train.shape[1] , 1)
X_test = X_test.reshape(X_test.shape[0] , X_test.shape[1] , 1)
```

In [ ]:
```
model.summary()
```

In [ ]:
```
model.fit(X_train,y_train,validation_data=(X_test,y_test),epochs=100,batch_size=6
```

Predicting the values

In [ ]:
```
#Doing prediction and checking performance matrix
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

In [ ]:
```
#Inverse transforming the values back to original
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

In [ ]:
```
#Calculating RMS performance matrix
import math
```

```
test_predict=model.predict(X_test)
```

In [ ]:
```
#Inverse transforming the values back to original
train_predict=scaler.inverse_transform(train_predict)
test_predict=scaler.inverse_transform(test_predict)
```

In [ ]:
```
#Calculating RMS performance matrix
import math
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(y_train,train_predict))
```

In [ ]:
```
#RMS for test data
math.sqrt(mean_squared_error(y_test,test_predict))
```

In [ ]:
```
### Plotting
# shift train predictions for plotting
look_back=100
trainPredictPlot = np.empty_like(df1)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back, :] = train_predict
# shift test predictions for plotting
testPredictPlot = np.empty_like(df1)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(df1)-1, :] = test_predict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(df1))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Here "Blue" line represents the whole dataset , "Yellow" line represents the training data and "Green" line represents the predicted values

Predicting the values for next 30 days

In [27]:
```
len(test_data)
```

Out[27]:
```
441
```

In [28]:
```
#because if we predict the output for next 30 days we will use the previous 100 d
x_input=test_data[341:].reshape(1,-1)
x_input.shape
```

Out[28]:
```
(1, 100)
```

In [29]:
```
temp_input=list(x_input)
temp_input=temp_input[0].tolist()
```

In [30]:
```
temp_input
```

In [30]:
```
temp_input
```

Out[30]:
```
[0.8583551465000423,
 0.8866418981676942,
 0.8743139407244789,
 0.8843198513890065,
 0.8783669678290975,
 0.8986321033521913,
 0.925821160179009,
```

Out[30]:

```
[0.8583551465000423,
 0.8866418981676942,
 0.8743139407244789,
 0.8843198513890065,
 0.8783669678290975,
 0.8986321033521913,
 0.925821160179009,
 0.9287764924427933,
 0.9567677108840666,
 0.9386979650426415,
 0.933040614709111,
 0.9495060373216249,
 0.9642404796082076,
 0.9551211686228154,
 0.9598919192772104,
 0.9663514312251966,
 0.9624672802499368,
 0.9229502659799038,
 0.9598497002448705,
 0.9879253567508233,
 0.985941062230854,
 0.9253145317909315,
 0.9217259140420504,
 0.964747107996285,
 0.9757240564046274,
 0.9915984125643842,
 0.9697289538123788,
 0.9761462467280253,
 0.9679557544541082,
 1.0000000000000002,
 0.9901629654648318,
 0.9905007177235499,
 0.9653803934813816,
 0.9848855864223593,
 0.9708688676855528,
 0.9402600692392133,
 0.8774803681499621,
 0.8348391454867856,
 0.8541332432660644,
 0.7733682344000676,
 0.7726927298826314,
 0.8801401671873683,
 0.8400743054969182,
 0.8967322468969012,
 0.8552731571392387,
 0.8388499535590646,
 0.742337245630303,
 0.8232711306256861,
 0.7814320695769654,
 0.6665963016127672,
 0.7921557037912694,
 0.6411804441442204,
 0.6861437135860848,
 0.6600101325677616,
 0.6520307354555435,
 0.5864223591995272,
 0.5658616904500551,
 0.6609867322346897,
 0.6526640209400402,
 0.6551549438486872,
 0.6378029215557038,
 0.7097019336216812,
 0.7267162036646122,
 0.664527569038118,
 0.7138816178333194,
 0.6943764248823416,
 0.7419150553069325,
 0.692181035210673,
 0.7500211095181702,
 0.6356919609400492,
 0.7722283205268936,
```

```
0.6608967322346897,
0.6528640269408402,
0.6551549438486872,
0.6378029215570358,
0.7097019336216812,
0.7267162036646122,
0.6645275690381118,
0.7138816178333194,
0.6943764248023416,
0.7419150553069325,
0.6921810352106773,
0.7500211095161702,
0.6356919609400492,
0.7722283205268936,
0.8304905851557884,
0.8194291986827664,
0.8289706999915563,
0.8125474964113824,
0.7877649244279323,
0.7516254327450818,
0.7842607447437306,
0.7797433082833742,
0.8132652199611587,
0.8141096006079542,
0.7947310647639958,
0.8333614793548934,
0.8589884319851391,
0.839018829688423,
0.8562864139153934,
0.8748627881448958,
0.887824031073208,
0.9009541501308793,
0.9279321117959978,
0.9485349995778098,
0.9333361479354896,
0.9174617917757326,
0.925441188887951,
0.9177151059697712,
0.9483239044161109,
0.9406400405302711,
0.9663514312251966,
0.9563033015283293,
0.964915984125644]
```

In [42]:

```
day_new=np.arange(1,101)
#taking the next 30 values as per predictions
day_pred=np.arange(101,131)
```

In [43]:

```
len(df1)
```

Out[43]: 1258

In [44]:

```
#Plotting for the previous values in the dataset
plt.plot(day_new,scaler.inverse_transform(df1[1158:]) , color="red")
#Plotting for the predicted values
plt.plot(day_pred,scaler.inverse_transform(lst_output) , color="green")
```

```
---------------------------------------------------------------------------
```

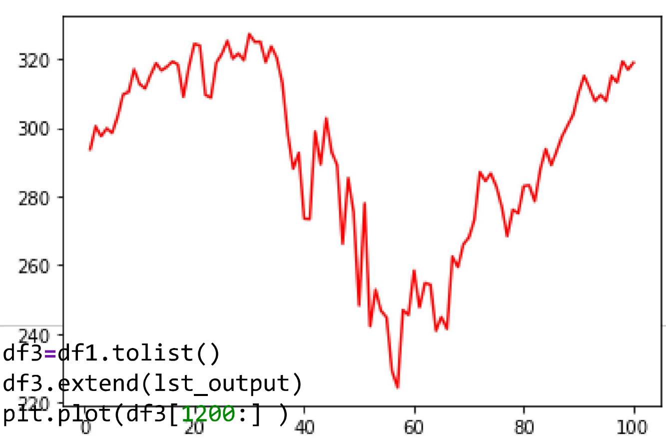
In [44]:
```python
#Plotting for the previous values in the dataset
plt.plot(day_new,scaler.inverse_transform(df1[1158:]) , color="red")
#Plotting for the predicted values
plt.plot(day_pred,scaler.inverse_transform(lst_output) , color="green")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [44], in <cell line: 4>()
      2 plt.plot(day_new,scaler.inverse_transform(df1[1158:]) , color="red")
      3 #Plotting for the predicted values
----> 4 plt.plot(day_pred,scaler.inverse_transform(lst_output) , color="green")

File ~\anaconda3\lib\site-packages\sklearn\preprocessing\_data.py:525, in MinMa
xScaler.inverse_transform(self, X)
    511 """Undo the scaling of X according to feature_range.
    512 
    513 Parameters
   (...)
    521     Transformed data.
    522 """
    523 check_is_fitted(self)
--> 525 X = check_array(
    526     X, copy=self.copy, dtype=FLOAT_DTYPES, force_all_finite="allow-nan"
    527 )
    529 X -= self.min_
    530 X /= self.scale_

File ~\anaconda3\lib\site-packages\sklearn\utils\validation.py:769, in check_ar
ray(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_fi
nite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    767     # If input is 1D raise error
    768     if array.ndim == 1:
--> 769         raise ValueError(
    770             "Expected 2D array, got 1D array instead:\narray={}.\n"
    771             "Reshape your data either using array.reshape(-1, 1) if "
    772             "your data has a single feature or array.reshape(1, -1) "
    773             "if it contains a single sample.".format(array)
    774         )
    776 # make sure we actually converted to numeric:
    777 if dtype_numeric and array.dtype.kind in "OUSV":

ValueError: Expected 2D array, got 1D array instead:
array=[].
Reshape your data either using array.reshape(-1, 1) if your data has a single f
eature or array.reshape(1, -1) if it contains a single sample.
```

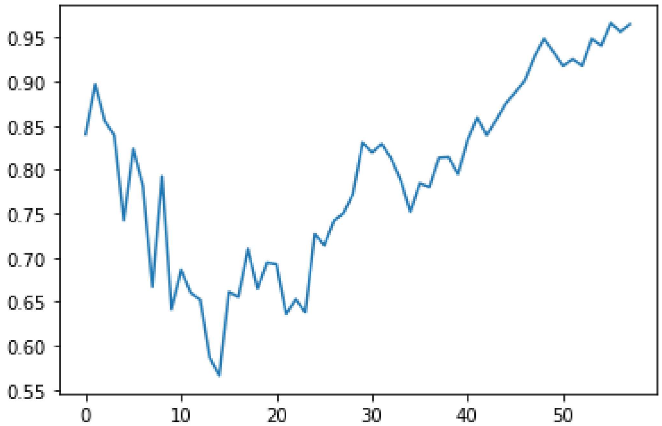


In [45]:
```python
df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[1200:] )
```

Out[45]: [<matplotlib.lines.Line2D at 0x1f9cf839d90>]

In [45]:
```
df3=df1.tolist()
df3.extend(lst_output)
plt.plot(df3[1200:] )
```

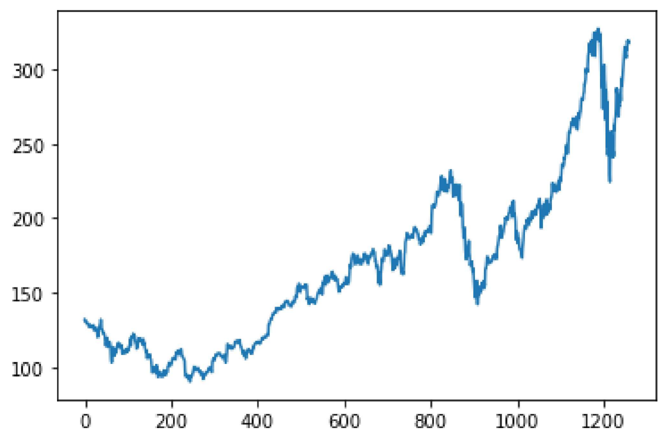
Out[45]: [<matplotlib.lines.Line2D at 0x1f9cf839d90>]



In [46]:
```
df3=scaler.inverse_transform(df3).tolist()
```

In [47]:
```
plt.plot(df3)
```

Out[47]: [<matplotlib.lines.Line2D at 0x1f9cf9a0a60>]



In [ ]: