Stock Market Analysis using CNN-LSTM model

This project is about analysis of Stock Market and providing suggestions and predictions to the stockholders. For this, we used CNN-LSTM approach to create a blank model, then use it to train on stock market data. Further implementation is discussed below...

In [42]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
#for dirname, _, filenames in os.walk('/kaggle/input'):
#    for filename in filenames:
#        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

Data Preprocessing and Analysis

In [43]:

```
import math
import seaborn as sns
import datetime as dt
from datetime import datetime
sns.set_style("whitegrid")
from pandas.plotting import autocorrelation_plot
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use("ggplot")
```

First we'd read the CSV file and then drop the null columns. Then we'd check the columns (some not all)

In [44]:

```
#1DP18XAREYFRWP4I
import requests
import csv
from tqdm import tqdm
key = "1DP18XAREYFRWP4I"

def request_stock_price_list(symbol, size, token):
    q_string = 'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY_ADJUSTED&sym
bol={}&outputsize={}&apikey={}'

    print("Retrieving stock price data from Alpha Vantage (This may take a while)...")
    r = requests.get(q_string.format(symbol, size, token))
    print("Data has been successfully downloaded...")
    date = []
    colnames = list(range(0, 7))
    df = pd.DataFrame(columns = colnames)
```

```python
    print("Sorting the retrieved data into a dataframe...")
    for i in tqdm(r.json()['Time Series (Daily)'].keys()):
        date.append(i)
        row = pd.DataFrame.from_dict(r.json()['Time Series (Daily)'][i], orient='index')
.reset_index().T[1:]
        df = pd.concat([df, row], ignore_index=True)
    df.columns = ["open", "high", "low", "close", "adjusted close", "volume", "dividend
amount", "split cf"]
    df['date'] = date
    return df
```

In [45]:

```python
cv1 = request_stock_price_list('IBM', 'full', key)
print(cv1.head)
cv1.to_csv('data.csv')
```

```
Retrieving stock price data from Alpha Vantage (This may take a while)...
Data has been successfully downloaded...
Sorting the retrieved data into a dataframe...
```

```
100%|██████████| 5555/5555 [01:43<00:00, 53.74it/s]
```

```
<bound method NDFrame.head of          open     high      low   close adjusted close     vol
ume  \
0       115.0  116.335  114.56  115.81         115.81   3322012
1      116.16   117.27  116.08  116.73         116.73   3220802
2      116.79   117.94  116.04  116.79         116.79   4914995
3       116.0   118.81  115.19  116.47         116.47   6417218
4      116.49   116.56  115.27  116.05         116.05   5384548
...       ...      ...     ...     ...            ...       ...
5550    92.75    92.94   90.19   90.25  52.2266076272  13737600
5551    94.44    94.44    90.0   91.56  52.9846891341  16697600
5552    95.87    95.94    93.5   94.37  54.6108029006  10369100
5553    96.75    96.81   93.69   94.81  54.8654256968  11105400
5554     98.5    98.81   96.37   96.75  55.9880807527   9551800

     dividend amount split cf        date
0             0.0000      1.0  2021-11-26
1             0.0000      1.0  2021-11-24
2             0.0000      1.0  2021-11-23
3             0.0000      1.0  2021-11-22
4             0.0000      1.0  2021-11-19
...              ...      ...         ...
5550          0.0000      1.0  1999-11-05
5551          0.0000      1.0  1999-11-04
5552          0.0000      1.0  1999-11-03
5553          0.0000      1.0  1999-11-02
5554          0.0000      1.0  1999-11-01

[5555 rows x 9 columns]>
```

In [46]:

```python
# For data preprocessing and analysis part
data = pd.read_csv('../input/price-volume-data-for-all-us-stocks-etfs/Stocks/abe.us.txt')
#data = pd.read_csv('../input/nifty50-stock-market-data/COALINDIA.csv')
#data = pd.read_csv('../input/stock-market-data/stock_market_data/nasdaq/csv/ABCO.csv')
#data = pd.read_csv('./data.csv')
# Any CSV or TXT file can be added here....
data.dropna(inplace=True)
data.head()
```

Out[46]:

	Date	Open	High	Low	Close	Volume	OpenInt
0	2005-02-25	6.4987	6.6009	6.4668	6.5753	55766	0
1	2005-02-28	6.6072	6.7669	6.5944	6.6263	49343	0
2	2005-03-01	6.6391	6.6773	6.6072	6.6072	31643	0

	Date	Open	High	Low	Close	Volume	OpenInt
3	2005-03-02	[illegible]	[illegible]	6.5484	[illegible]	27101	0
4	2005-03-03	6.5753	6.6135	6.5562	6.5944	17387	0

In [47]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3190 entries, 0 to 3189
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     3190 non-null   object
 1   Open     3190 non-null   float64
 2   High     3190 non-null   float64
 3   Low      3190 non-null   float64
 4   Close    3190 non-null   float64
 5   Volume   3190 non-null   int64
 6   OpenInt  3190 non-null   int64
dtypes: float64(4), int64(2), object(1)
memory usage: 199.4+ KB
```

In [48]:

```
data.describe()
```

Out[48]:

	Open	High	Low	Close	Volume	OpenInt
count	3190.000000	3190.000000	3190.000000	3190.000000	3190.000000	3190.0
mean	11.599416	11.712848	11.484610	11.605599	28444.870846	0.0
std	2.350376	2.365621	2.327065	2.341989	37525.175821	0.0
min	5.860300	5.905000	5.834700	5.841100	106.000000	0.0
25%	10.534000	10.655000	10.413750	10.554000	8147.750000	0.0
50%	11.981000	12.067000	11.899000	11.988500	17741.500000	0.0
75%	13.271000	13.386750	13.189000	13.295750	36167.250000	0.0
max	18.130000	19.151000	17.842000	17.925000	634041.000000	0.0

In [49]:

```
data.isnull().sum()
```

Out[49]:

```
Date       0
Open       0
High       0
Low        0
Close      0
Volume     0
OpenInt    0
dtype: int64
```

In [50]:

```
data.reset_index(drop=True, inplace=True)
data.fillna(data.mean(), inplace=True)
data.head()
```

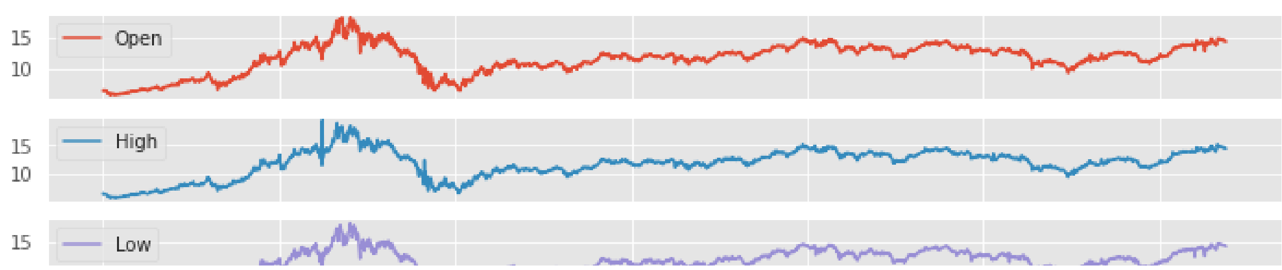
Out[50]:

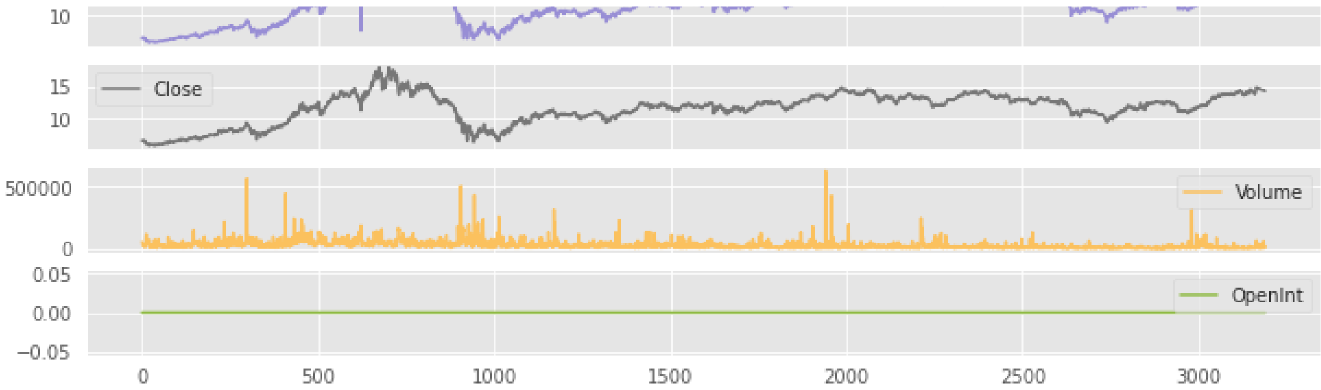
	Date	Open	High	Low	Close	Volume	OpenInt
0	2005-02-25	6.4987	6.6009	6.4668	6.5753	55766	0
1	2005-02-28	6.6072	6.7669	6.5944	6.6263	49343	0

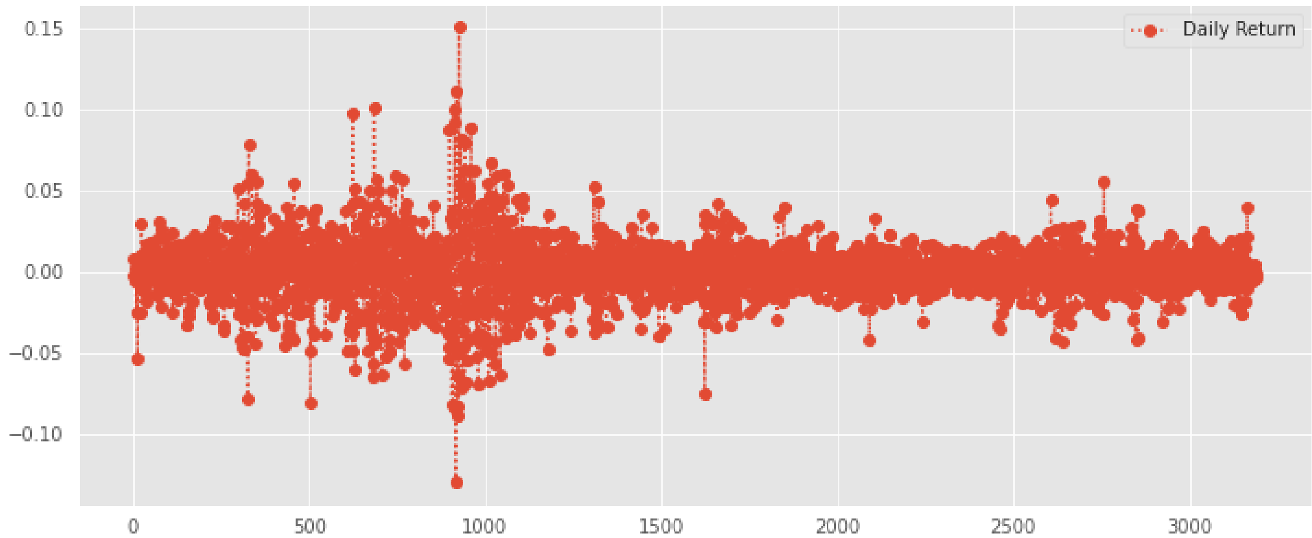
	Date	Open	High	Low	Close	Volume	OpenInt
1	2005-02-28	6.6072	6.7009	6.5944	[illegible]	[illegible]	0
~~2~~	~~2005-03-01~~	~~6.6391~~	~~6.6773~~	~~6.6072~~	~~6.6072~~	~~31643~~	~~0~~
3	2005-03-02	6.5753	6.6072	6.5434	6.5816	27101	0
4	2005-03-03	6.5753	6.6135	6.5562	6.5944	17387	0

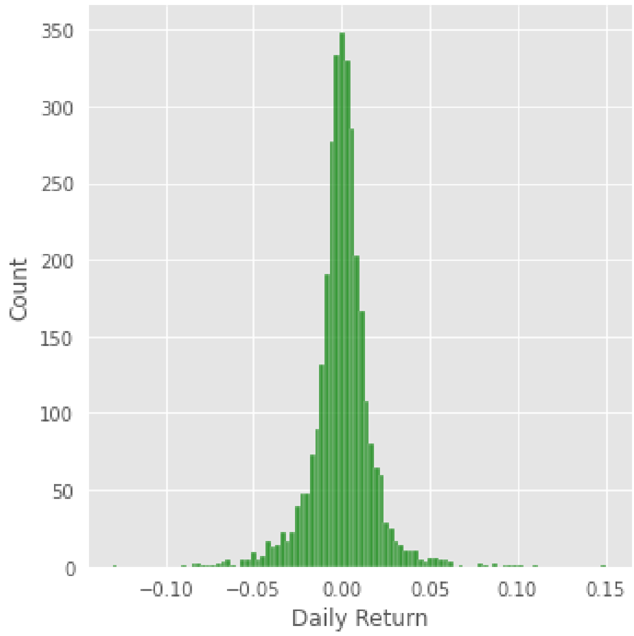
In [51]:

```
data.plot(legend=True,subplots=True, figsize = (12, 6))
plt.show()
#data['Close'].plot(legend=True, figsize = (12, 6))
#plt.show()
#data['Volume'].plot(legend=True,figsize=(12,7))
#plt.show()

data.shape
data.size
data.describe(include='all').T
data.dtypes
data.nunique()
ma_day = [10,50,100]

for ma in ma_day:
    column_name = "MA for %s days" %(str(ma))
    data[column_name]=pd.DataFrame.rolling(data['Close'],ma).mean()

data['Daily Return'] = data['Close'].pct_change()
# plot the daily return percentage
data['Daily Return'].plot(figsize=(12,5),legend=True,linestyle=':',marker='o')
plt.show()

sns.displot(data['Daily Return'].dropna(),bins=100,color='green')
plt.show()

date=pd.DataFrame(data['Date'])
closing_df1 = pd.DataFrame(data['Close'])
close1  = closing_df1.rename(columns={"Close": "data_close"})
close2=pd.concat([date,close1],axis=1)
close2.head()

data.reset_index(drop=True, inplace=True)
data.fillna(data.mean(), inplace=True)
data.head()

data.nunique()

data.sort_index(axis=1,ascending=True)

cols_plot = ['Open', 'High', 'Low','Close','Volume','MA for 10 days','MA for 50 days','MA
for 100 days','Daily Return']
axes = data[cols_plot].plot(marker='.', alpha=0.7, linestyle='None', figsize=(11, 9), su
bplots=True)
for ax in axes:
    ax.set_ylabel('Daily trade')

plt.plot(data['Close'], label="Close price")
plt.xlabel("Timestamp")
plt.ylabel("Closing price")
df = data
print(df)

data.isnull().sum()
```









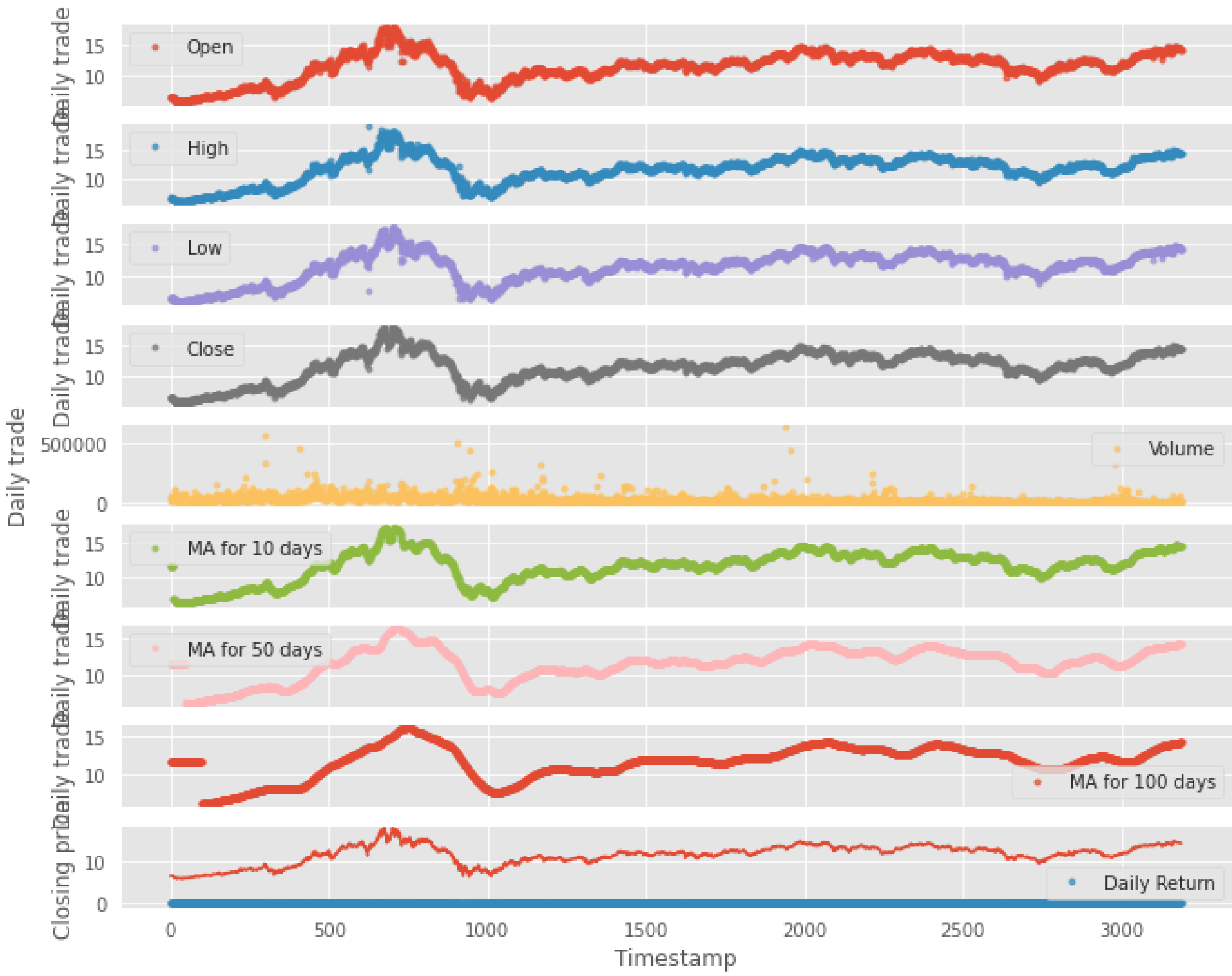
	Date	Open	High	Low	Close	Volume	OpenInt	\
0	2005-02-25	6.4987	6.6009	6.4668	6.5753	55766	0	
1	2005-02-28	6.6072	6.7669	6.5944	6.6263	49343	0	
2	2005-03-01	6.6391	6.6773	6.6072	6.6072	31643	0	
3	2005-03-02	6.5753	6.6072	6.5434	6.5816	27101	0	
4	2005-03-03	6.5753	6.6135	6.5562	6.5944	17387	0	
...	...	...	...	...	...	...	...	
3185	2017-11-06	14.3998	14.4802	14.3900	14.4400	62423	0	
3186	2017-11-07	14.4400	14.4400	14.4000	14.4000	6722	0	
3187	2017-11-08	14.3400	14.4352	14.3400	14.3781	6304	0	
3188	2017-11-09	14.3300	14.3737	14.2800	14.3200	18761	0	
3189	2017-11-10	14.2500	14.3000	14.2400	14.3000	10658	0	

	MA for 10 days	MA for 50 days	MA for 100 days	Daily Return
0	11.60878	11.625272	11.649354	0.000388
1	11.60878	11.625272	11.649354	0.007756
2	11.60878	11.625272	11.649354	-0.002882
3	11.60878	11.625272	11.649354	-0.003875

```
4          11.60878      11.625272      11.649354       0.001945
...             ...            ...            ...            ...
3185       14.44648      14.344662      14.136796       0.003893
3186       14.43071      14.355862      14.142926      -0.002770
3187       14.42077      14.361972      14.150117      -0.001521
3188       14.40677      14.369792      14.155817      -0.004041
3189       14.39377      14.371792      14.160597      -0.001397

[3190 rows x 11 columns]
```

Out[51]:

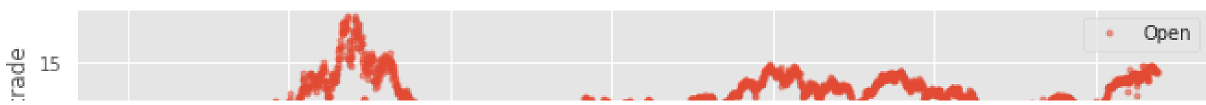
```
Date                 0
Open                 0
High                 0
Low                  0
Close                0
Volume               0
OpenInt              0
MA for 10 days       0
MA for 50 days       0
MA for 100 days      0
Daily Return         0
dtype: int64
```

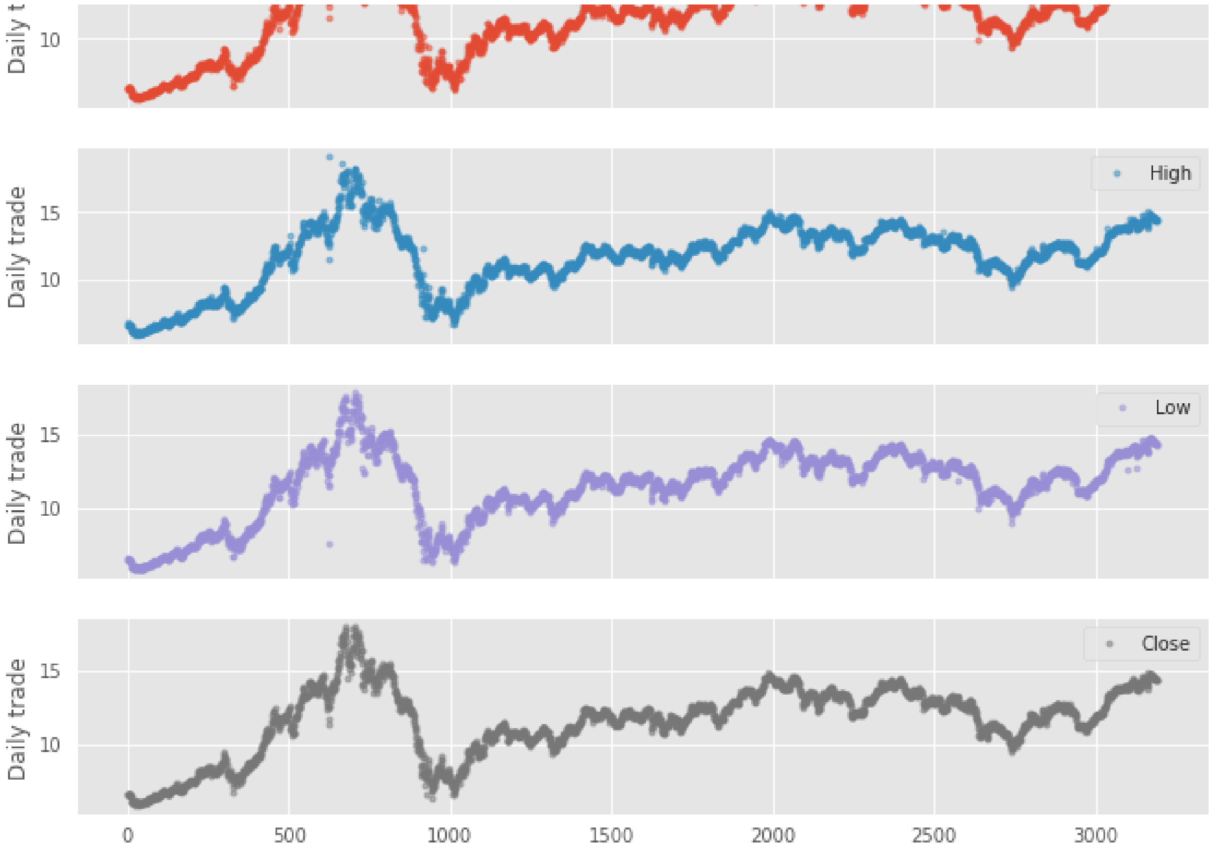


After that, we'll visualize the data for understanding, this is shown below...

In [52]:

```
cols_plot = ['Open', 'High', 'Low','Close']
axes = data[cols_plot].plot(marker='.', alpha=0.5, linestyle='None', figsize=(11, 9), su
bplots=True)
for ax in axes:
    ax.set_ylabel('Daily trade')
```





Then we'd print the data after making changes and dropping null data

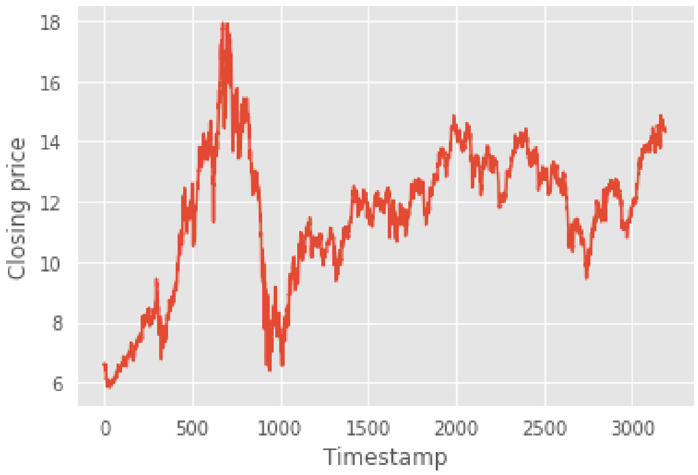
In [53]:

```
plt.plot(data['Close'], label="Close price")
plt.xlabel("Timestamp")
plt.ylabel("Closing price")
df = data
print(df)

df.describe().transpose()
```

```
            Date     Open     High      Low    Close  Volume  OpenInt  \
0     2005-02-25   6.4987   6.6009   6.4668   6.5753   55766        0
1     2005-02-28   6.6072   6.7669   6.5944   6.6263   49343        0
2     2005-03-01   6.6391   6.6773   6.6072   6.6072   31643        0
3     2005-03-02   6.5753   6.6072   6.5434   6.5816   27101        0
4     2005-03-03   6.5753   6.6135   6.5562   6.5944   17387        0
...          ...      ...      ...      ...      ...     ...      ...
3185  2017-11-06  14.3998  14.4802  14.3900  14.4400   62423        0
3186  2017-11-07  14.4400  14.4400  14.4000  14.4000    6722        0
3187  2017-11-08  14.3400  14.4352  14.3400  14.3781    6304        0
3188  2017-11-09  14.3300  14.3737  14.2800  14.3200   18761        0
3189  2017-11-10  14.2500  14.3000  14.2400  14.3000   10658        0

      MA for 10 days  MA for 50 days  MA for 100 days  Daily Return
0           11.60878       11.625272        11.649354      0.000388
1           11.60878       11.625272        11.649354      0.007756
2           11.60878       11.625272        11.649354     -0.002882
3           11.60878       11.625272        11.649354     -0.003875
4           11.60878       11.625272        11.649354      0.001945
...              ...             ...              ...           ...
3185        14.44648       14.344662        14.136796      0.003893
3186        14.43071       14.355862        14.142926     -0.002770
3187        14.42077       14.361972        14.150117     -0.001521
3188        14.40677       14.369792        14.155817     -0.004041
3189        14.39377       14.371792        14.160597     -0.001397

[3190 rows x 11 columns]
```

Out[53]:

	count	mean	std	min	25%	50%	75%	max
Open	3190.0	11.599416	2.350376	5.860300	10.534000	11.981000	13.271000	18.130000
High	3190.0	11.712848	2.365621	5.905000	10.655000	12.067000	13.386750	19.151000
Low	3190.0	11.484610	2.327065	5.834700	10.413750	11.899000	13.189000	17.842000
Close	3190.0	11.605599	2.341989	5.841100	10.554000	11.988500	13.295750	17.925000
Volume	3190.0	28444.870846	37525.175821	106.000000	8147.750000	17741.500000	36167.250000	634041.000000
OpenInt	3190.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
MA for 10 days	3190.0	11.608780	2.321162	5.963080	10.577125	11.962700	13.297200	17.329400
MA for 50 days	3190.0	11.625272	2.231059	6.037646	10.591696	11.933450	13.269480	16.618460
MA for 100 days	3190.0	11.649354	2.113346	6.221377	10.632551	11.876775	13.200810	16.042560
Daily Return	3190.0	0.000388	0.017010	-0.130345	-0.006439	0.000484	0.007807	0.150503



In [54]:

```
X = data.drop(['Date', 'Close'], axis=1)
Y = data['Close']

X.shape,Y.shape

from mlxtend.feature_selection import SequentialFeatureSelector as sfs
from sklearn.linear_model import LinearRegression

lreg = LinearRegression()
sfs1 = sfs(lreg, k_features=2, forward=False, verbose=2, scoring='neg_mean_squared_error
')

sfs1 = sfs1.fit(X, Y)

feat_names = list(sfs1.k_feature_names_)
print(feat_names)

# creating a new dataframe using the above variables and adding the target variable
new_data = data[feat_names]
new_data['Close'] = data['Close']

# first five rows of the new data
new_data.head()

new_data.shape, data.shape

df = new_data
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.1s finished

[2021-11-29 11:49:40] Features: 8/2 -- score: -0.01130510673631423[Parallel(n_jobs=1)]: U
sing backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
```

```
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.1s finished

[2021-11-29 11:49:40] Features: 7/2 -- score: -0.011114316874792215[Parallel(n_jobs=1)]:
Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.1s finished

[2021-11-29 11:49:40] Features: 6/2 -- score: -0.011080371541763374[Parallel(n_jobs=1)]:
Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.1s finished

[2021-11-29 11:49:40] Features: 5/2 -- score: -0.011080371541730121[Parallel(n_jobs=1)]:
Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
```

```
['High', 'Low']
```

```
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.1s finished

[2021-11-29 11:49:40] Features: 4/2 -- score: -0.011086733169734618[Parallel(n_jobs=1)]:
Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s finished

[2021-11-29 11:49:40] Features: 3/2 -- score: -0.011860213917250834[Parallel(n_jobs=1)]:
Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s finished

[2021-11-29 11:49:40] Features: 2/2 -- score: -0.014047232655157732/opt/conda/lib/python3
.7/site-packages/ipykernel_launcher.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
```

The data has been analysed but it must be converted into data of shape [100,1] to make it easier for CNN to train on... Else it won't select necessary features and the model will fail

In [55]:

```
from sklearn.model_selection import train_test_split

X = []
Y = []
window_size=100
for i in range(1 , len(df) - window_size -1 , 1):
    first = df.iloc[i,2]
    temp = []
    temp2 = []
    for j in range(window_size):
        temp.append((df.iloc[i + j, 2] - first) / first)
    temp2.append((df.iloc[i + window_size, 2] - first) / first)
    X.append(np.array(temp).reshape(100, 1))
    Y.append(np.array(temp2).reshape(1, 1))

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=True)

train_X = np.array(x_train)
test_X = np.array(x_test)
train_Y = np.array(y_train)
test_Y = np.array(y_test)

train_X = train_X.reshape(train_X.shape[0],1,100,1)
test_X = test_X.reshape(test_X.shape[0],1,100,1)

print(len(train_X))
print(len(test_X))
```

```
2470
```

Training part

This part has 2 subparts: CNN and LSTM

For CNN, the layers are created with sizes 64,128,64. In every layer, TimeDistributed function is added to track the features with respect to time. In between them, Pooling layers are added.

After that, it's passed to Bi-LSTM layers

In [56]:

```
# For creating model and training
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, Bidirectional, TimeDist
ributed
from tensorflow.keras.layers import MaxPooling1D, Flatten
from tensorflow.keras.regularizers import L1, L2
from tensorflow.keras.metrics import Accuracy
from tensorflow.keras.metrics import RootMeanSquaredError

model = tf.keras.Sequential()

# Creating the Neural Network model here...
model.add(TimeDistributed(Conv1D(64, kernel_size=1, activation='relu', input_shape=(None
, 100, 1))))
model.add(TimeDistributed(MaxPooling1D(2)))
model.add(TimeDistributed(Conv1D(128, kernel_size=1, activation='relu')))
model.add(TimeDistributed(MaxPooling1D(2)))
model.add(TimeDistributed(Conv1D(64, kernel_size=1, activation='relu')))
model.add(TimeDistributed(MaxPooling1D(2)))
model.add(TimeDistributed(Flatten()))
# model.add(Dense(5, kernel_regularizer=L2(0.01)))
model.add(Bidirectional(LSTM(100, return_sequences=True)))
model.add(Dropout(0.25))
model.add(Bidirectional(LSTM(100, return_sequences=False)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='linear'))
model.compile(optimizer='adam', loss='mse', metrics=['mse', 'mae'])

history = model.fit(train_X, train_Y, validation_data=(test_X,test_Y), epochs=40,batch_s
ize=40, verbose=1, shuffle =True)
```

```
Epoch 1/40
62/62 [==============================] - 8s 35ms/step - loss: 0.0130 - mse: 0.0130 - mae:
0.0805 - val_loss: 0.0023 - val_mse: 0.0023 - val_mae: 0.0343
Epoch 2/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0028 - mse: 0.0028 - mae:
0.0382 - val_loss: 0.0026 - val_mse: 0.0026 - val_mae: 0.0379
Epoch 3/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0025 - mse: 0.0025 - mae:
0.0366 - val_loss: 0.0018 - val_mse: 0.0018 - val_mae: 0.0307
Epoch 4/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0022 - mse: 0.0022 - mae:
0.0341 - val_loss: 0.0021 - val_mse: 0.0021 - val_mae: 0.0349
Epoch 5/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0022 - mse: 0.0022 - mae:
0.0349 - val_loss: 0.0018 - val_mse: 0.0018 - val_mae: 0.0310
Epoch 6/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0021 - mse: 0.0021 - mae:
0.0343 - val_loss: 0.0017 - val_mse: 0.0017 - val_mae: 0.0307
Epoch 7/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0020 - mse: 0.0020 - mae:
0.0329 - val_loss: 0.0016 - val_mse: 0.0016 - val_mae: 0.0277
Epoch 8/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0021 - mse: 0.0021 - mae:
0.0328 - val_loss: 0.0014 - val_mse: 0.0014 - val_mae: 0.0265
Epoch 9/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0019 - mse: 0.0019 - mae:
```

```
62/62 [==============================] - 1s 10ms/step - loss: 0.0019 - mse: 0.0019 - mae:
0.0315 - val_loss: 0.0016 - val_mse: 0.0016 - val_mae: 0.0282
Epoch 10/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0021 - mse: 0.0021 - mae:
0.0330 - val_loss: 0.0015 - val_mse: 0.0015 - val_mae: 0.0272
Epoch 11/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0021 - mse: 0.0021 - mae:
0.0328 - val_loss: 0.0017 - val_mse: 0.0017 - val_mae: 0.0292
Epoch 12/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0018 - mse: 0.0018 - mae:
0.0308 - val_loss: 0.0016 - val_mse: 0.0016 - val_mae: 0.0280
Epoch 13/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0019 - mse: 0.0019 - mae:
0.0321 - val_loss: 0.0016 - val_mse: 0.0016 - val_mae: 0.0281
Epoch 14/40
62/62 [==============================] - 1s 11ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0301 - val_loss: 0.0019 - val_mse: 0.0019 - val_mae: 0.0308
Epoch 15/40
62/62 [==============================] - 1s 11ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0309 - val_loss: 0.0013 - val_mse: 0.0013 - val_mae: 0.0261
Epoch 16/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0019 - mse: 0.0019 - mae:
0.0308 - val_loss: 0.0016 - val_mse: 0.0016 - val_mae: 0.0292
Epoch 17/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0303 - val_loss: 0.0013 - val_mse: 0.0013 - val_mae: 0.0260
Epoch 18/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0296 - val_loss: 0.0016 - val_mse: 0.0016 - val_mae: 0.0295
Epoch 19/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0301 - val_loss: 0.0013 - val_mse: 0.0013 - val_mae: 0.0257
Epoch 20/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0293 - val_loss: 0.0015 - val_mse: 0.0015 - val_mae: 0.0285
Epoch 21/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0300 - val_loss: 0.0017 - val_mse: 0.0017 - val_mae: 0.0288
Epoch 22/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0302 - val_loss: 0.0014 - val_mse: 0.0014 - val_mae: 0.0288
Epoch 23/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0289 - val_loss: 0.0017 - val_mse: 0.0017 - val_mae: 0.0291
Epoch 24/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0297 - val_loss: 0.0017 - val_mse: 0.0017 - val_mae: 0.0293
Epoch 25/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0296 - val_loss: 0.0018 - val_mse: 0.0018 - val_mae: 0.0307
Epoch 26/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0307 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0251
Epoch 27/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0292 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0254
Epoch 28/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0305 - val_loss: 0.0020 - val_mse: 0.0020 - val_mae: 0.0312
Epoch 29/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0019 - mse: 0.0019 - mae:
0.0312 - val_loss: 0.0014 - val_mse: 0.0014 - val_mae: 0.0266
Epoch 30/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0288 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0251
Epoch 31/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0294 - val_loss: 0.0018 - val_mse: 0.0018 - val_mae: 0.0299
Epoch 32/40
62/62 [==============================] - 1s 12ms/step - loss: 0.0016 - mse: 0.0016 - mae:
0.0292 - val_loss: 0.0013 - val_mse: 0.0013 - val_mae: 0.0260
Epoch 33/40
62/62 [==============================] - 1s 11ms/step - loss: 0.0015 - mse: 0.0015 - mae:
```

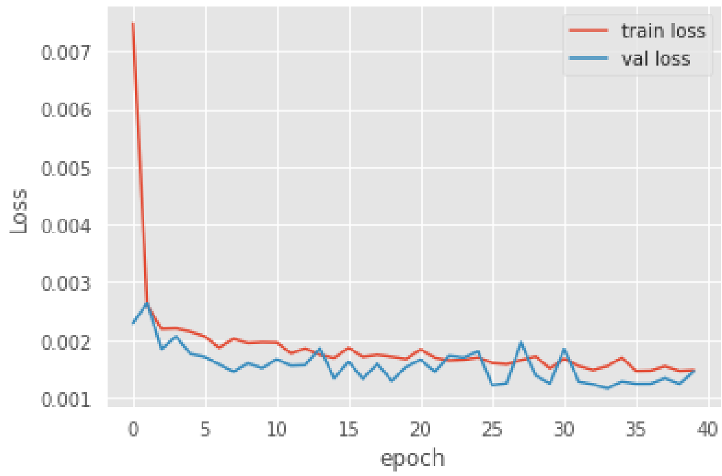
```
62/62 [==============================] - 1s 11ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0284 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0252
Epoch 34/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0290 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0243
Epoch 35/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0017 - mse: 0.0017 - mae:
0.0308 - val_loss: 0.0013 - val_mse: 0.0013 - val_mae: 0.0259
Epoch 36/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0014 - mse: 0.0014 - mae:
0.0284 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0252
Epoch 37/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0289 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0256
Epoch 38/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0290 - val_loss: 0.0013 - val_mse: 0.0013 - val_mae: 0.0265
Epoch 39/40
62/62 [==============================] - 1s 9ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0285 - val_loss: 0.0012 - val_mse: 0.0012 - val_mae: 0.0253
Epoch 40/40
62/62 [==============================] - 1s 10ms/step - loss: 0.0014 - mse: 0.0014 - mae:
0.0276 - val_loss: 0.0015 - val_mse: 0.0015 - val_mae: 0.0277
```

In [57]:

```
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[57]:

```
<matplotlib.legend.Legend at 0x7f382ca22110>
```

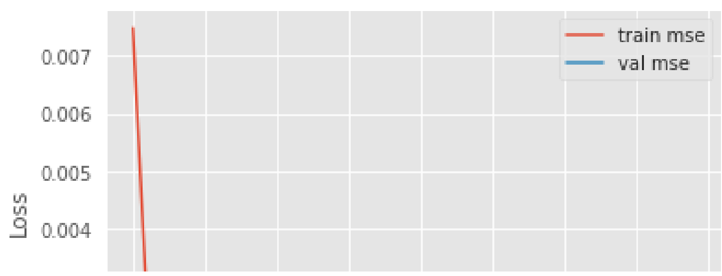


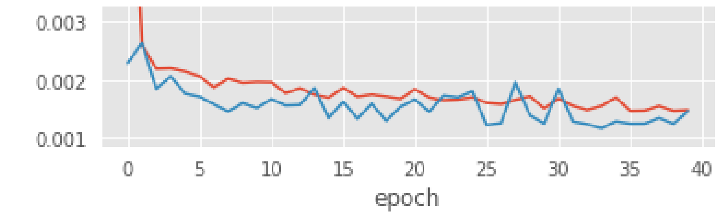
In [58]:

```
plt.plot(history.history['mse'], label='train mse')
plt.plot(history.history['val_mse'], label='val mse')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[58]:

```
<matplotlib.legend.Legend at 0x7f382c421790>
```



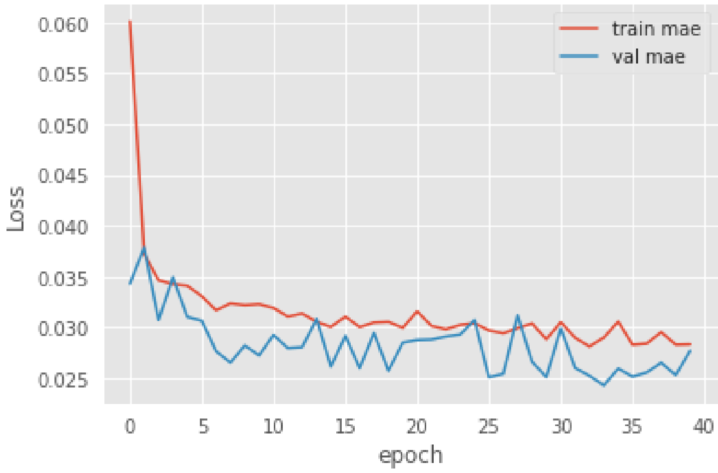


In [59]:

```
plt.plot(history.history['mae'], label='train mae')
plt.plot(history.history['val_mae'], label='val mae')
plt.xlabel("epoch")
plt.ylabel("Loss")
plt.legend()
```

Out[59]:

```
<matplotlib.legend.Legend at 0x7f3335291410>
```



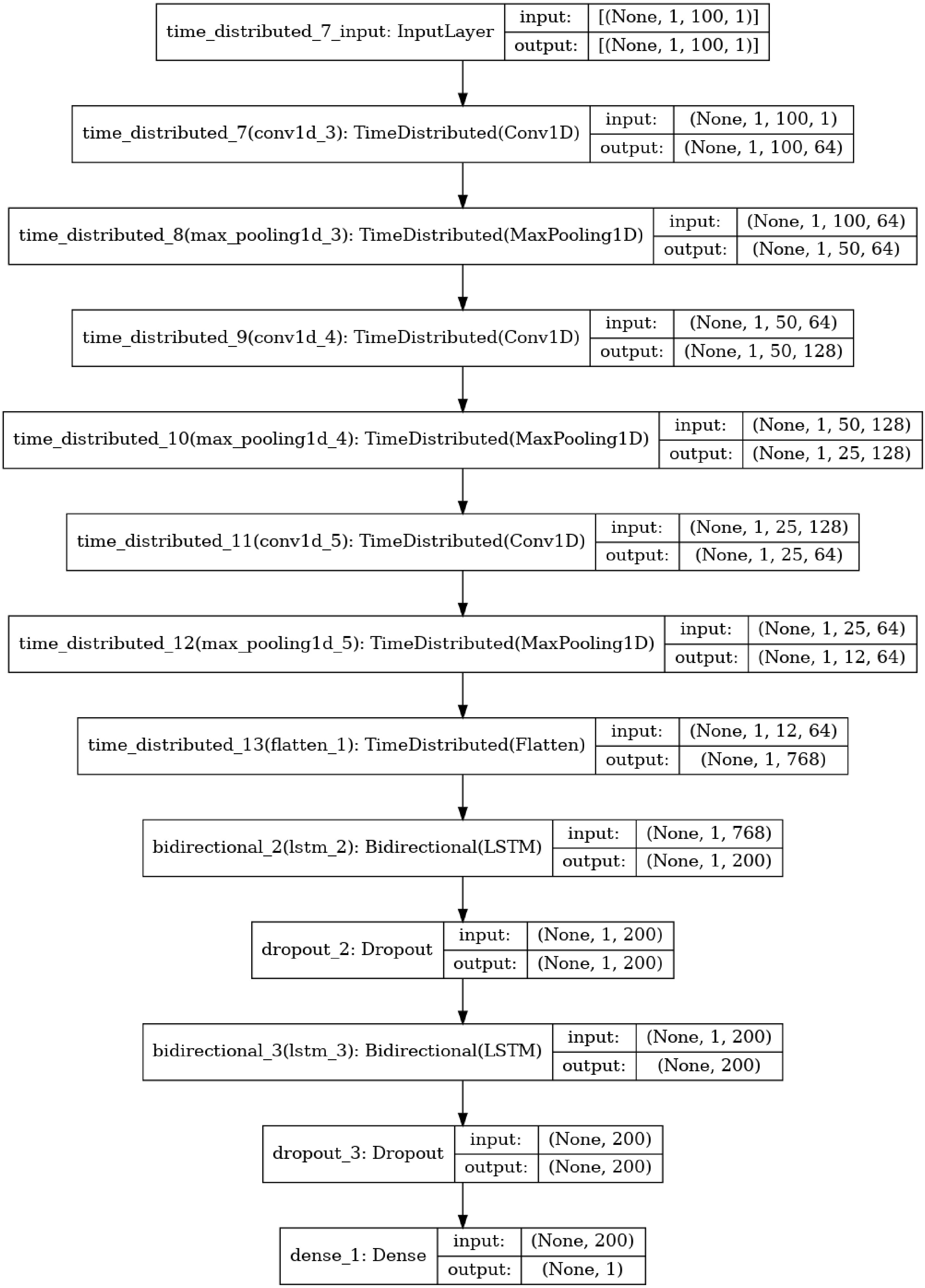
In [60]:

```
# After the model has been constructed, we need to train
from tensorflow.keras.utils import plot_model
print(model.summary())
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
time_distributed_7 (TimeDist (None, 1, 100, 64)        128
_________________________________________________________________
time_distributed_8 (TimeDist (None, 1, 50, 64)         0
_________________________________________________________________
time_distributed_9 (TimeDist (None, 1, 50, 128)        8320
_________________________________________________________________
time_distributed_10 (TimeDis (None, 1, 25, 128)        0
_________________________________________________________________
time_distributed_11 (TimeDis (None, 1, 25, 64)         8256
_________________________________________________________________
time_distributed_12 (TimeDis (None, 1, 12, 64)         0
_________________________________________________________________
time_distributed_13 (TimeDis (None, 1, 768)            0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 1, 200)            695200
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 200)            0
_________________________________________________________________
bidirectional_3 (Bidirection (None, 200)               240800
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 201
=================================================================
Total params: 952,905
```

```
Trainable params: 952,905
Non-trainable params: 0
_________________________________________________________________
None
```

Out[60]:



In [61]:

```
model.evaluate(test_X, test_Y)
```

```
20/20 [==============================] - 0s 4ms/step - loss: 0.0015 - mse: 0.0015 - mae:
0.0277
```

Out[61]:

```
[0.0014527516905218363, 0.0014527516905218363, 0.027694158256053925]
```

In [62]:

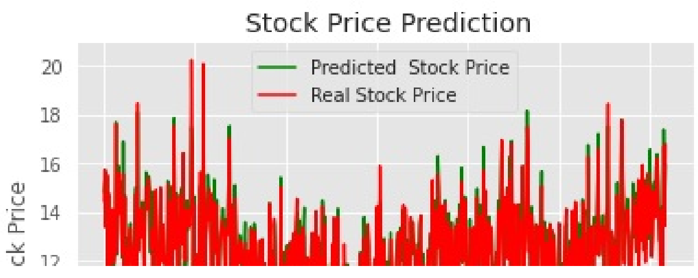
```
from sklearn.metrics import explained_variance_score
from sklearn.metrics import r2_score
from sklearn.metrics import max_error

# predict probabilities for test set
yhat_probs = model.predict(test_X, verbose=0)
# predict crisp classes for test set
yhat_classes = model.predict_classes(test_X, verbose=0)
# reduce to 1d array
yhat_probs = yhat_probs[:, 0]
yhat_classes = yhat_classes[:, 0]

var = explained_variance_score(test_Y.reshape(-1,1), yhat_probs)
print('Variance: %f' % var)

r2 = r2_score(test_Y.reshape(-1,1), yhat_probs)
print('R2 Score: %f' % var)

var2 = max_error(test_Y.reshape(-1,1), yhat_probs)
print('Max Error: %f' % var2)
```

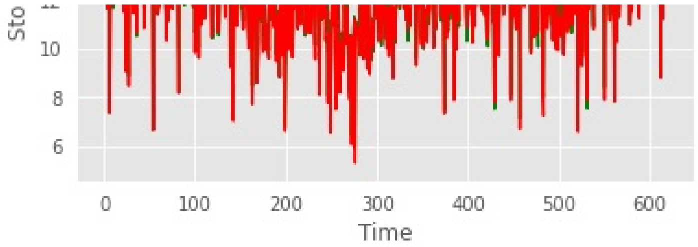
```
Variance: 0.947681
R2 Score: 0.947681
Max Error: 0.216366
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/sequential.py:450:
UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01
. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi
-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.pre
dict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it
uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

In [63]:

```
predicted  = model.predict(test_X)
test_label = test_Y.reshape(-1,1)
predicted = np.array(predicted[:,0]).reshape(-1,1)
len_t = len(train_X)
for j in range(len_t , len_t + len(test_X)):
    temp = data.iloc[j,3]
    test_label[j - len_t] = test_label[j - len_t] * temp + temp
    predicted[j - len_t] = predicted[j - len_t] * temp + temp
plt.plot(predicted, color = 'green', label = 'Predicted  Stock Price')
plt.plot(test_label, color = 'red', label = 'Real Stock Price')
plt.title(' Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel(' Stock Price')
plt.legend()
plt.show()
```





Testing part

In this part, the model is saved and loaded back again. Then, it's made to train again but with different data to check it's loss and prediction

In [64]:

```
# First we need to save a model
model.save("model.h5")
```

In [65]:

```
# Load model
new_model = tf.keras.models.load_model("./model.h5")
```

In [66]:

```
new_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
time_distributed_7 (TimeDist (None, 1, 100, 64)        128       
_________________________________________________________________
time_distributed_8 (TimeDist (None, 1, 50, 64)         0         
_________________________________________________________________
time_distributed_9 (TimeDist (None, 1, 50, 128)        8320      
_________________________________________________________________
time_distributed_10 (TimeDis (None, 1, 25, 128)        0         
_________________________________________________________________
time_distributed_11 (TimeDis (None, 1, 25, 64)         8256      
_________________________________________________________________
time_distributed_12 (TimeDis (None, 1, 12, 64)         0         
_________________________________________________________________
time_distributed_13 (TimeDis (None, 1, 768)            0         
_________________________________________________________________
bidirectional_2 (Bidirection (None, 1, 200)            695200    
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 200)            0         
_________________________________________________________________
bidirectional_3 (Bidirection (None, 200)               240800    
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 201       
=================================================================
Total params: 952,905
Trainable params: 952,905
Non-trainable params: 0
_________________________________________________________________
```

In [67]:

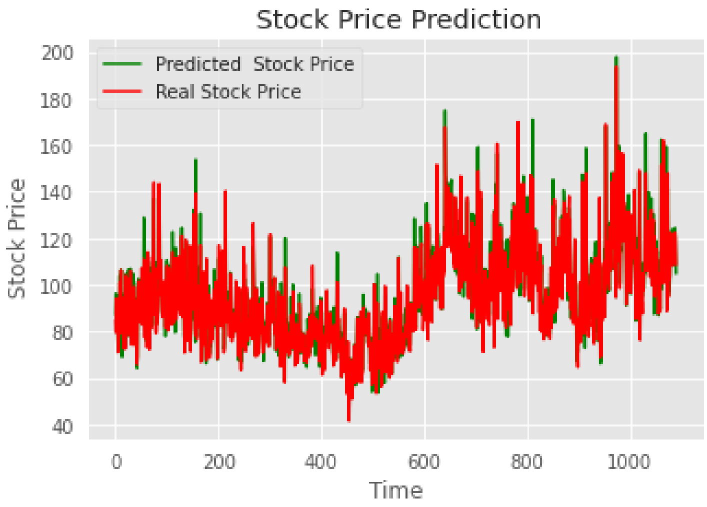
```
# For data preprocessing and analysis part
#data2 = pd.read_csv('../input/price-volume-data-for-all-us-stocks-etfs/Stocks/aaoi.us.tx
t')
#data2 = pd.read_csv('../input/nifty50-stock-market-data/SBIN.csv')
#data2 = pd.read_csv('../input/stock-market-data/stock_market_data/nasdaq/csv/ACTG.csv')
```

```
data2 = pd.read_csv('./data.csv')
# Any CSV or TXT file can be added here....
data2.dropna(inplace=True)
data2.head()

data2.reset_index(drop=True, inplace=True)
data2.fillna(data.mean(), inplace=True)
data2.head()
df2 = data2.drop('date', axis=1)

print(df2)

X = []
Y = []
window_size=100
for i in range(1 , len(df2) - window_size -1 , 1):
    first = df2.iloc[i,4]
    temp = []
    temp2 = []
    for j in range(window_size):
        temp.append((df2.iloc[i + j, 4] - first) / first)
    # for j in range(week):
    temp2.append((df2.iloc[i + window_size, 4] - first) / first)
    # X.append(np.array(stock.iloc[i:i+window_size,4]).reshape(50,1))
    # Y.append(np.array(stock.iloc[i+window_size,4]).reshape(1,1))
    # print(stock2.iloc[i:i+window_size,4])
    X.append(np.array(temp).reshape(100, 1))
    Y.append(np.array(temp2).reshape(1, 1))

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, shuffle=True)

train_X = np.array(x_train)
test_X = np.array(x_test)
train_Y = np.array(y_train)
test_Y = np.array(y_test)

train_X = train_X.reshape(train_X.shape[0],1,100,1)
test_X = test_X.reshape(test_X.shape[0],1,100,1)

print(len(train_X))
print(len(test_X))
```

```
      Unnamed: 0    open     high     low   close  adjusted close    volume  \
0              0  115.00  116.335  114.56  115.81      115.810000   3322012
1              1  116.16  117.270  116.08  116.73      116.730000   3220802
2              2  116.79  117.940  116.04  116.79      116.790000   4914995
3              3  116.00  118.810  115.19  116.47      116.470000   6417218
4              4  116.49  116.560  115.27  116.05      116.050000   5384548
...          ...     ...      ...     ...     ...             ...       ...
5550        5550   92.75   92.940   90.19   90.25       52.226608  13737600
5551        5551   94.44   94.440   90.00   91.56       52.984689  16697600
5552        5552   95.87   95.940   93.50   94.37       54.610803  10369100
5553        5553   96.75   96.810   93.69   94.81       54.865426  11105400
5554        5554   98.50   98.810   96.37   96.75       55.988081   9551800

      dividend amount  split cf
0                 0.0       1.0
1                 0.0       1.0
2                 0.0       1.0
3                 0.0       1.0
4                 0.0       1.0
...               ...       ...
5550              0.0       1.0
5551              0.0       1.0
5552              0.0       1.0
5553              0.0       1.0
5554              0.0       1.0

[5555 rows x 9 columns]
4362
1091
```

In [68]:

```
predicted  = model.predict(test_X)
test_label = test_Y.reshape(-1,1)
predicted = np.array(predicted[:,0]).reshape(-1,1)
len_t = len(train_X)
for j in range(len_t , len_t + len(test_X)):
    temp = data2.iloc[j,3]
    test_label[j - len_t] = test_label[j - len_t] * temp + temp
    predicted[j - len_t] = predicted[j - len_t] * temp + temp
plt.plot(predicted, color = 'green', label = 'Predicted  Stock Price')
plt.plot(test_label, color = 'red', label = 'Real Stock Price')
plt.title(' Stock Price Prediction')
plt.xlabel('Time')
plt.ylabel(' Stock Price')
plt.legend()
plt.show()
```



In [ ]: