In [1]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
```

In [3]:
```
import matplotlib.pyplot as plt
def loss_plot(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    #plt.legend(['Train'], loc='upper right')
```

In [25]:
```
data = pd.read_csv('data/q2_dataset.csv')
```

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

Out[9]:

	Date	Close/Last	Volume	Open	High	Low
0	07/08/20	$381.37	29272970	376.72	381.50	376.36
1	07/07/20	$372.69	28106110	375.41	378.62	372.23
2	07/06/20	$373.85	29663910	370.00	375.78	369.87
3	07/02/20	$364.11	28510370	367.85	370.47	363.64
4	07/01/20	$364.11	27684310	365.12	367.36	363.91

In [8]:
```
data.tail()
```

Out[8]:

	Date	Close/Last	Volume	Open	High	Low
1254	07/15/2015	$126.82	33559770	125.72	127.15	125.58
1255	07/14/2015	$125.61	31695870	126.04	126.37	125.04
1256	07/13/2015	$125.66	41365600	125.03	125.76	124.32
1257	07/10/15	$123.28	61292800	121.94	123.85	121.21
1258	07/09/15	$120.07	78291510	123.85	124.06	119.22

In [ ]:
```
type(data)
```

Out[ ]:
```
pandas.core.frame.DataFrame
```

In [26]:
```
list(data.columns)
```

Out[26]:
```
['Date', ' Close/Last', ' Volume', ' Open', ' High', ' Low']
```

In [27]:
```
data.columns = data.columns.str.strip()
list(data.columns)
```

Out[27]:
```
['Date', 'Close/Last', 'Volume', 'Open', 'High', 'Low']
```

In [28]:
```
data['target']= data['Open']
data['Date'] =pd.to_datetime(data.Date)
data=data.sort_values(by='Date')
data.reset_index(inplace=True, drop=True)
data.head()
```

Out[28]:

	Date	Close/Last	Volume	Open	High	Low	target
0	2015-07-09	$120.07	78291510	123.85	124.06	119.22	123.85
1	2015-07-10	$123.28	61292800	121.94	123.85	121.21	121.94
2	2015-07-13	$125.66	41365600	125.03	125.76	124.32	125.03
3	2015-07-14	$125.61	31695870	126.04	126.37	125.04	126.04
4	2015-07-15	$126.82	33559770	125.72	127.15	125.58	125.72

In [19]:
```
data.head(20)
```

Out[19]:

	Date	Close/Last	Volume	Open	High	Low	target
0	2015-07-09	$120.07	78291510	123.85	124.06	119.22	123.85
1	2015-07-10	$123.28	61292800	121.94	123.85	121.21	121.94
2	2015-07-13	$125.66	41365600	125.03	125.76	124.32	125.03
3	2015-07-14	$125.61	31695870	126.04	126.37	125.04	126.04
4	2015-07-15	$126.82	33559770	125.72	127.15	125.58	125.72
5	2015-07-16	$128.51	35987630	127.74	128.57	127.35	127.74
6	2015-07-17	$129.62	45970470	129.08	129.62	128.31	129.08
7	2015-07-20	$132.07	55204920	130.97	132.97	130.70	130.97
8	2015-07-21	$130.75	73006780	132.85	132.92	130.32	132.85
9	2015-07-22	$125.22	115288400	121.99	125.50	121.99	121.99
10	2015-07-23	$125.16	50832950	126.20	127.09	125.06	126.20
11	2015-07-24	$124.50	42090320	125.32	125.74	123.90	125.32
12	2015-07-27	$122.77	44371580	123.09	123.61	122.12	123.09
13	2015-07-28	$123.38	33570380	123.38	123.91	122.55	123.38
14	2015-07-29	$122.99	36912040	123.15	123.50	122.27	123.15
15	2015-07-30	$122.37	33400950	122.32	122.57	121.71	122.32
16	2015-07-31	$121.30	42832890	122.60	122.64	120.91	122.60
17	2015-08-03	$118.44	69639900	121.50	122.57	117.52	121.50

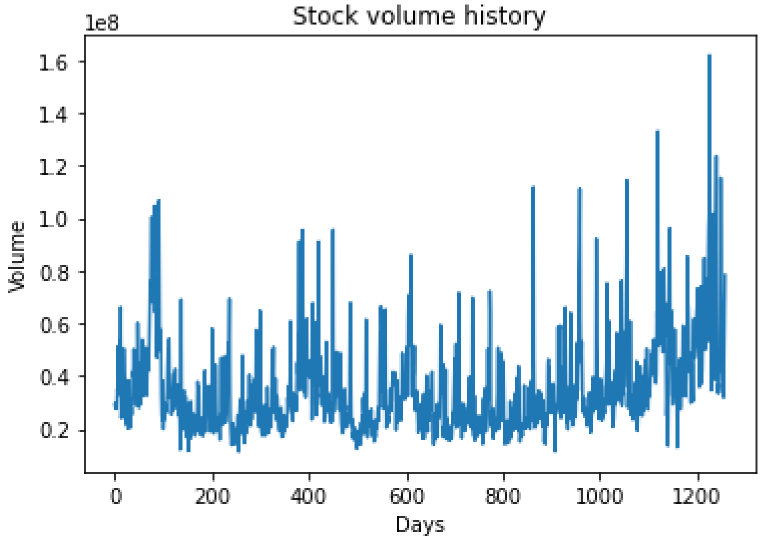
	Date	Close/Last	Volume	Open	High	Low	target
18	2015-08-04	$114.64	123601900	117.42	117.70	113.25	117.42
19	2015-08-05	$115.40	99202400	112.95	117.44	112.10	112.95

In [ ]:

```
from matplotlib import pyplot as plt
plt.figure()
plt.plot(data["Open"])
plt.plot(data["High"])
plt.plot(data["Low"])
#plt.plot(data["Close"])
plt.title('Stock price history')
plt.ylabel('Price (USD)')
plt.xlabel('Days')
plt.legend(['Open','High','Low'], loc='upper left')
plt.show()
```

In [ ]:

```
plt.figure()
plt.plot(data["Volume"])
plt.title('Stock volume history')
plt.ylabel('Volume')
plt.xlabel('Days')
plt.show()
```



In [29]:

```
#create features using columns from previous 3 days
data['Volume_t-3'] = data.shift(3)['Volume']
data['Volume_t-2'] = data.shift(2)['Volume']
data['Volume_t-1'] = data.shift(1)['Volume']
data['Open_t-3'] = data.shift(3)['Open']
data['Open_t-2'] = data.shift(2)['Open']
data['Open_t-1'] = data.shift(1)['Open']
data['High_t-3'] = data.shift(3)['High']
data['High_t-2'] = data.shift(2)['High']
data['High_t-1'] = data.shift(1)['High']
data['Low_t-3'] = data.shift(3)['Low']
data['Low_t-2'] = data.shift(2)['Low']
data['Low_t-1'] = data.shift(1)['Low']
data['target']= data['Open']
data.head()
```

Out[29]:

	Date	Close/Last	Volume	Open	High	Low	target	Volume_t-3	Volume_t-2	Volume_
0	2015-07-09	$120.07	78291510	123.85	124.06	119.22	123.85	NaN	NaN	N
1	2015-07-10	$123.28	61292800	121.94	123.85	121.21	121.94	NaN	NaN	7829151
2	2015-07-13	$125.66	41365600	125.03	125.76	124.32	125.03	NaN	78291510.0	6129280
3	2015-07-14	$125.61	31695870	126.04	126.37	125.04	126.04	78291510.0	61292800.0	4136560
4	2015-07-15	$126.82	33559770	125.72	127.15	125.58	125.72	61292800.0	41365600.0	3169587

In [30]:

```
data = data.drop(['Close/Last','Volume','Open','High','Low'], axis = 1)
data.head()
```

Out[30]:

	Date	target	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	Higl

	Date	target	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	Higl
0	2015-07-09	123.85	NaN	NaN	NaN	NaN	NaN	NaN	NaN	I
1	2015-07-10	121.94	NaN	NaN	78291510.0	NaN	NaN	123.85	NaN	I
2	2015-07-13	125.03	NaN	78291510.0	61292800.0	NaN	123.85	121.94	NaN	12
3	2015-07-14	126.04	78291510.0	61292800.0	41365600.0	123.85	121.94	125.03	124.06	12
4	2015-07-15	125.72	61292800.0	41365600.0	31695870.0	121.94	125.03	126.04	123.85	12

In [31]:

```
data.isna().sum()
```

Out[31]:

```
Date           0
target         0
Volume_t-3     3
Volume_t-2     2
Volume_t-1     1
Open_t-3       3
Open_t-2       2
Open_t-1       1
High_t-3       3
High_t-2       2
High_t-1       1
Low_t-3        3
Low_t-2        2
Low_t-1        1
dtype: int64
```

In [32]:

```
#drop columns with null values
data = data.dropna()
data.reset_index(inplace=True, drop=True)
data.head()
```

Out[32]:

	Date	target	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	Hig
0	2015-07-14	126.04	78291510.0	61292800.0	41365600.0	123.85	121.94	125.03	124.06	12
1	2015-07-15	125.72	61292800.0	41365600.0	31695870.0	121.94	125.03	126.04	123.85	12
2	2015-07-16	127.74	41365600.0	31695870.0	33559770.0	125.03	126.04	125.72	125.76	12
3	2015-07-17	129.08	31695870.0	33559770.0	35987630.0	126.04	125.72	127.74	126.37	12
4	2015-07-20	130.97	33559770.0	35987630.0	45970470.0	125.72	127.74	129.08	127.15	12

In [33]:
```
list(data.columns)
```

Out[33]:
```
['Date',
 'target',
 'Volume_t-3',
 'Volume_t-2',
 'Volume_t-1',
 'Open_t-3',
 'Open_t-2',
 'Open_t-1',
 'High_t-3',
 'High_t-2',
 'High_t-1',
 'Low_t-3',
 'Low_t-2',
 'Low_t-1']
```

In [34]:
```
data = data[[
 'Date',
 'Volume_t-3',
 'Volume_t-2',
 'Volume_t-1',
 'Open_t-3',
 'Open_t-2',
 'Open_t-1',
 'High_t-3',
 'High_t-2',
 'High_t-1',
 'Low_t-3',
 'Low_t-2',
 'Low_t-1',
 'target']]
data.head()
```

Out[34]:

	Date	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2	Hi
0	2015-07-14	78291510.0	61292800.0	41365600.0	123.85	121.94	125.03	124.06	123.85	1
1	2015-07-15	61292800.0	41365600.0	31695870.0	121.94	125.03	126.04	123.85	125.76	1
2	2015-07-16	41365600.0	31695870.0	33559770.0	125.03	126.04	125.72	125.76	126.37	'
3	2015-07-17	31695870.0	33559770.0	35987630.0	126.04	125.72	127.74	126.37	127.15	1
4	2015-07-20	33559770.0	35987630.0	45970470.0	125.72	127.74	129.08	127.15	128.57	1

Dataset Creation

We sorted the dataset in ascending order, since our intention is to predict the opening price from the **previous** three days. Using the pandas shift function which shifts the index by desired number of periods, we were able to create new features by specifying the index that was needed. For example, to get the Volume from three days prior, we shift by 3 - data.shift(3)['Volume']. This process was repeated for all necessary columns and indices.

In [35]:
```
len(data)
```

Out[35]:
```
1256
```

In [36]:
```
from sklearn.model_selection import train_test_split
#split the data into train and test set
train, test = train_test_split(data, test_size=0.30, random_state=0)
#save the data
train.to_csv('train_data_RNN.csv',index=False)
test.to_csv('test_data_RNN.csv',index=False)
```

In [ ]:
```
type(train)
```

Out[ ]:
```
pandas.core.frame.DataFrame
```

In [38]:
```
train.head()
```

Out[38]:

	Date	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2
689	2018-04-09	34581850.0	26750260.0	34949690.0	164.88	172.58	170.97	172.01	174.23
1134	2020-01-14	42621540.0	35217270.0	30521720.0	307.24	310.60	311.64	310.43	312.67
901	2019-02-11	28204640.0	31644240.0	23793830.0	174.65	172.40	168.99	175.57	173.94
579	2017-10-27	17633730.0	21175670.0	16916650.0	156.29	156.91	157.23	157.42	157.55
367	2016-12-23	21337310.0	23724430.0	26043820.0	116.74	116.80	116.35	117.50	117.40

In [40]:
```
data_train = pd.read_csv('train_data_RNN.csv')
data_test = pd.read_csv('test_data_RNN.csv')
```

Preprocessing

Scaling the data

The range of the data is widely varied. The values of Volume are very high and could skew the model. Normalizing data helps the algorithm in converging i.e. to find local/ global minimum efficiently. We utilise the Minmax scaler to keep feature values between 0 and 1.

Scaled values of X are created using the following formula:

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min

We also tried the Standard scaler, however there was no significant difference in training or test loss with this scaler.

Splitting Features and Target

The target is the opening price of the day we wish to predict.

In [41]:

```
#separate features and target
X_train = data_train.drop(['Date','target'], axis = 1)
y_train = data_train['target']
X_test_date = data_test
X_test = data_test.drop(['Date','target'], axis = 1)
y_test = data_test['target']
```

In [ ]:

```
X_train
```

Out[ ]:

	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2	High_t
0	34581850.0	26750260.0	34949690.0	164.88	172.58	170.97	172.01	174.23	172.4
1	42621540.0	35217270.0	30521720.0	307.24	310.60	311.64	310.43	312.67	317.0
2	28204640.0	31644240.0	23793830.0	174.65	172.40	168.99	175.57	173.94	170.6
3	17633730.0	21175670.0	16916650.0	156.29	156.91	157.23	157.42	157.55	157.8
4	21337310.0	23724430.0	26043820.0	116.74	116.80	116.35	117.50	117.40	116.5
...	...	...	...	...	...	...	...	...	.
874	20182050.0	20670830.0	15955820.0	189.69	191.78	190.68	192.55	192.43	191.9
875	36487930.0	38016810.0	52954070.0	211.15	216.88	219.05	215.18	220.45	222.3
876	28803760.0	33511990.0	36486560.0	303.22	305.64	308.10	305.17	310.35	317.0
877	35907770.0	25402270.0	21983410.0	151.78	153.80	153.89	153.92	154.72	154.2
878	39824200.0	41464880.0	38116290.0	173.68	167.25	167.81	175.15	170.02	171.7

879 rows × 12 columns

In [42]:

```
X_test
```

Out[42]:

	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2	High_t
0	35987630.0	45970470.0	55204920.0	127.74	129.08	130.97	128.57	129.62	132.9
1	35421310.0	25674500.0	24725210.0	145.13	147.17	145.01	147.16	148.28	146.1
2	50278030.0	35678360.0	50061580.0	113.38	113.63	113.25	114.18	114.72	115.5
3	29773430.0	22526310.0	30684390.0	184.28	183.08	186.51	184.99	185.47	191.9

	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2	High_t
4	26560420.0	26178840.0	31735810.0	109.51	110.23	109.95	110.73	110.98	110.4
...	...	...	...	...	...	...	...	...	.
372	64678220.0	53168580.0	56157370.0	112.18	111.94	111.07	112.68	112.80	111.9
373	33935720.0	69281360.0	54017920.0	208.76	216.42	213.90	210.16	221.37	218.0
374	24833800.0	25080500.0	20117070.0	145.87	145.50	147.97	146.18	148.49	149.3
375	53812480.0	32503750.0	45247890.0	284.69	277.95	276.28	286.95	281.68	277.2
376	69032740.0	24677880.0	12119710.0	282.23	280.53	284.69	282.65	284.25	284.8

377 rows × 12 columns

In [43]:

```
y_train
```

Out[43]:

```
0      169.88
1      316.70
2      171.05
3      159.29
4      115.59
        ...
874    192.45
875    209.55
876    317.83
877    153.21
878    167.88
Name: target, Length: 879, dtype: float64
```

In [44]:

```
y_test
```

Out[44]:

```
0      132.85
1      144.49
2      116.44
3      191.81
4      108.91
        ...
372    112.02
373    205.53
374    148.82
375    273.61
376    284.82
Name: target, Length: 377, dtype: float64
```

In [45]:

```
X_test_date
```

Out[45]:

	Date	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2
0	2015-07-21	35987630.0	45970470.0	55204920.0	127.74	129.08	130.97	128.57	129.62
1	2017-06-28	35421310.0	25674500.0	24725210.0	145.13	147.17	145.01	147.16	148.28

	Date	Volume_t-3	Volume_t-2	Volume_t-1	Open_t-3	Open_t-2	Open_t-1	High_t-3	High_t-2
2	2015-09-25	50278030.0	35678360.0	50061580.0	113.38	113.63	113.25	114.18	114.72
3	2019-06-10	29773430.0	22526310.0	30684390.0	184.28	183.08	186.51	184.99	185.47
4	2016-04-08	26560420.0	26178840.0	31735810.0	109.51	110.23	109.95	110.73	110.98
...	...	...	...	...	...	...	...	...	...
372	2015-12-17	64678220.0	53168580.0	56157370.0	112.18	111.94	111.07	112.68	112.80
373	2019-08-02	33935720.0	69281360.0	54017920.0	208.76	216.42	213.90	210.16	221.37
374	2017-07-17	24833800.0	25080500.0	20117070.0	145.87	145.50	147.97	146.18	148.49
375	2020-04-22	53812480.0	32503750.0	45247890.0	284.69	277.95	276.28	286.95	281.68
376	2019-12-26	69032740.0	24677880.0	12119710.0	282.23	280.53	284.69	282.65	284.25

377 rows × 14 columns

In [46]:

```
#scale the data
#scaling the dataset using minmaxscaler
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler(feature_range=(0,1))
X_train=scaler.fit_transform(X_train)
X_test=scaler.transform(X_test)
```

In [50]:

```
X_train[:10]
```

Out[50]:

```
array([[0.15426342, 0.12644697, 0.15606877, 0.26949793, 0.29492857,
        0.28369714, 0.28620284, 0.29059167, 0.28161701, 0.27437691,
        0.29461484, 0.27707898],
       [0.20767603, 0.19602151, 0.12662877, 0.78186072, 0.78785714,
        0.77656704, 0.77930961, 0.77786773, 0.7855027 , 0.78971724,
        0.78024251, 0.78358077],
       [0.11189569, 0.16666145, 0.08189741, 0.30466079, 0.29428571,
        0.27675975, 0.29888497, 0.28957094, 0.27527444, 0.30381869,
        0.28840942, 0.27785848],
       [0.04166663, 0.08063982, 0.03617348, 0.23858197, 0.23896429,
        0.23555587, 0.23422749, 0.23188202, 0.23056282, 0.24314969,
        0.23466476, 0.23661553],
       [0.06627179, 0.10158332, 0.0968568 , 0.09623898, 0.09571429,
        0.09232332, 0.09201667, 0.09056351, 0.0865656 , 0.09914735,
        0.09739658, 0.09084789],
       [0.06032861, 0.23632515, 0.11907544, 0.29656289, 0.29867857,
        0.29739673, 0.29019273, 0.29038049, 0.29806587, 0.29944614,
        0.29597004, 0.30067675],
       [0.10418206, 0.2385764 , 0.12959566, 0.31034731, 0.30571429,
        0.29935882, 0.30750597, 0.3035796 , 0.30723122, 0.31161638,
```

```
 0.28815977, 0.29947206],
[0.1531356 , 0.10127863, 0.09360209, 0.10199748, 0.10714286,
 0.10300971, 0.10177764, 0.10147478, 0.09904164, 0.10475878,
 0.10784593, 0.10406406],
[0.11052279, 0.34197841, 0.68454705, 0.05398596, 0.04967857,
 0.02102239, 0.04980229, 0.04797438, 0.02453389, 0.05480251,
 0.05149786, 0.02012543],
[0.04760117, 0.07985845, 0.04767943, 0.1903905 , 0.19325   ,
 0.19175922, 0.18909195, 0.19105276, 0.18881338, 0.19468736,
 0.19222539, 0.19267973]])
```

In [51]:

```
#numpy array conversion
X_train=np.array(X_train)
X_test=np.array(X_test)
```

In [52]:

```
# reshape input to be [samples, time steps, features] which is required for LSTM
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 [54]:

```
X_train.shape
```

Out[54]:

```
(879, 12, 1)
```

In [55]:

```
X_test.shape
```

Out[55]:

```
(377, 12, 1)
```

Design Steps

RNNs were not chosen because of the vanishing gradient problem. Long short-term memory (LSTM) is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called “forget gates”.LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space.

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks introduced in 2014. They are used in the full form and several simplified variants. They have fewer parameters than LSTM, as they lack an output gate.

LSTMs and GRUs take 3D input so data was reshaped. We considered various RNN architectures but the following gave the best performance for our problem.

Step 1: Model Architecture

MODEL 1

LSTM LAYER 1 - 50 units --> Dropout 0.2 --> LSTM LAYER 2 - 50 units --> Dropout 0.2 --> LSTM LAYER 3 - 50 units --> Dropout 0.2 --> Dense Layer - 1 unit

This model uses three LSTM layers. 20 % of the nodes at each layer are unused to avoid overfitting and improve model performance.

MODEL 2

GRU Layer 75 units --> GRU Layer 30 units --> GRU Layer 30 units --> Dropout 0.2 --> Dense Layer - 1 layer

This model uses three GRU layers. 20 % of the nodes at the final GRU layer are unused to avoid overfitting and improve model performance.

Step 2: Optimizers considered

Adagrad - Resulted in poor model performance. Model did not train.

Stochastic Gradient Descents - Resulted in poor model performance. Model did not train.

Adam: Model performed well with this. It is also recommended as the best optimizer for LSTMs as referenced in [1]

Step 3: Number of Epochs

Epochs	Model 1 Training Loss	Model 2 Training Loss
100	13621	18898
256	4851	8759
512	4203	4312
800	1033	2159
1500	189	292

Step 4: Runtime

Model 1 - 13 min 26s for 1500 epochs Model 2 - 12 min 26s for 1500 epochs

Model 2 has a shorter run time, perhaps because of the smaller width in its 2nd and 3rd layer.

Step 5: Loss Metric

Mean Squared Error.

In [56]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Embedding
from keras.layers import LSTM, SimpleRNN, GRU, Bidirectional
from keras import callbacks
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau, C
```

Model #1

In [57]:

```
model = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, lstm_units]
```

```
    tf.keras.layers.LSTM(50, return_sequences=True, input_shape=(12,1)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(50, return_sequences=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(50),
    tf.keras.layers.Dropout(0.2),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1, activation='linear')
])
```

In [58]:

```
model.compile(loss='mean_squared_error',optimizer='adam')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=
```

In [59]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 12, 50)            10400
_________________________________________________________________
dropout (Dropout)            (None, 12, 50)            0
_________________________________________________________________
lstm_1 (LSTM)                (None, 12, 50)            20200
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 50)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 50)                20200
_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 51
=================================================================
Total params: 50,851
Trainable params: 50,851
Non-trainable params: 0
_________________________________________________________________
```

In [60]:

```
%%time
history = model.fit(X_train,y_train,validation_split=0.05,epochs=1500,batch_size
```

```
Epoch 1/1500
14/14 [==============================] - 7s 142ms/step - loss: 33012.8359 - val_
loss: 33627.8398
Epoch 2/1500
14/14 [==============================] - 0s 31ms/step - loss: 31456.1758 - val_l
oss: 31396.8242
Epoch 3/1500
14/14 [==============================] - 0s 30ms/step - loss: 30037.9141 - val_l
oss: 30665.1504
Epoch 4/1500
14/14 [==============================] - 0s 31ms/step - loss: 29497.0449 - val_l
oss: 30262.4941
Epoch 5/1500
14/14 [==============================] - 0s 31ms/step - loss: 29174.7969 - val_l
oss: 29956.0996
Epoch 6/1500
```

```
s: 23.7803
Epoch 1491/1500
14/14 [==============================] - 1s 40ms/step - loss: 192.2093 - val_los
s: 31.8610
Epoch 1492/1500
14/14 [==============================] - 1s 39ms/step - loss: 215.0018 - val_los
s: 25.3716
Epoch 1493/1500
14/14 [==============================] - 1s 43ms/step - loss: 188.5737 - val_los
s: 26.7372
Epoch 1494/1500
14/14 [==============================] - 1s 43ms/step - loss: 165.1457 - val_los
s: 22.6986
Epoch 1495/1500
14/14 [==============================] - 1s 46ms/step - loss: 179.4910 - val_los
s: 24.8382
Epoch 1496/1500
14/14 [==============================] - 1s 40ms/step - loss: 171.4703 - val_los
s: 27.4761
Epoch 1497/1500
14/14 [==============================] - 1s 43ms/step - loss: 182.3687 - val_los
s: 28.1025
Epoch 1498/1500
14/14 [==============================] - 1s 40ms/step - loss: 169.7404 - val_los
s: 30.3197
Epoch 1499/1500
14/14 [==============================] - 1s 38ms/step - loss: 176.8685 - val_los
s: 32.3442
Epoch 1500/1500
14/14 [==============================] - 1s 40ms/step - loss: 189.4727 - val_los
s: 38.6774
CPU times: user 16min 31s, sys: 33 s, total: 17min 4s
Wall time: 13min 26s
```

In [62]:

```
print(history.history["loss"][-1])
```

```
189.47265625
```

In [63]:

```
print('Training MSE for Model 1', model.evaluate(X_train, y_train, verbose=0))
```

```
Training MSE for Model 1 24.01258659362793
```

In [83]:

```
model.predict(X_train) - y_train.values
```

Out[83]:

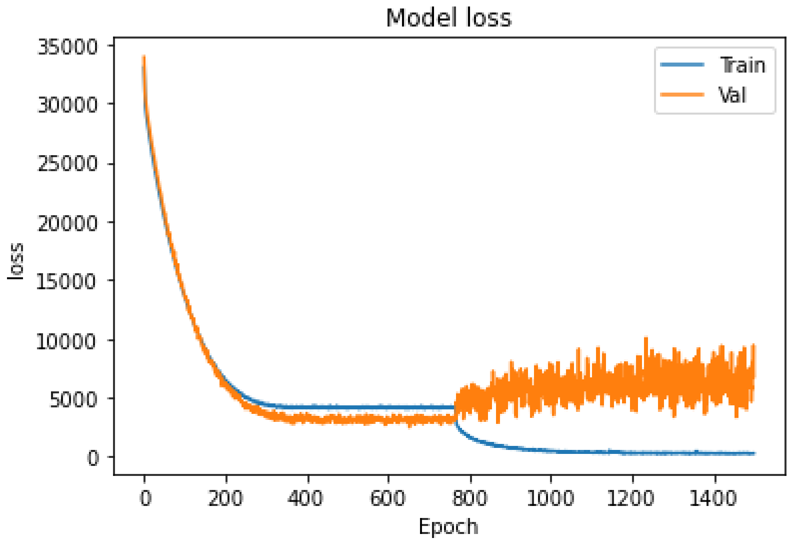
```
array([[   2.95023071, -143.86976929,    1.78023071, ..., -144.99976929,
          19.62023071,    4.95023071],
       [ 147.15701782,    0.33701782,  145.98701782, ...,   -0.79298218,
         163.82701782,  149.15701782],
       [   4.44254028, -142.37745972,    3.27254028, ..., -143.50745972,
          21.11254028,    6.44254028],
       ...,
       [ 143.92276489,   -2.89723511,  142.75276489, ...,   -4.02723511,
         160.59276489,  145.92276489],
       [ -14.04496277, -160.86496277,  -15.21496277, ..., -161.99496277,
           2.62503723,  -12.04496277],
       [   0.6628772 , -146.1571228 ,   -0.5071228 , ..., -147.2871228 ,
          17.3328772 ,    2.6628772 ]])
```

In [64]:

```
#save for the best model
model.save('models/Group3_RNN_model.h5')
```

In [ ]:

```
loss_plot(history)
```



Comment on Model #1 Training Output

- The training and validation losses start at a high value of approximately 35000
- There is a drastic decrease in the first 200 epochs
- The gap between training and validation loss remains steady until about 800 epochs
- The final training loss achieved is approximately 189

In [85]:

```
y_test=np.array(y_test)
y_pred = model.predict(X_test, verbose = 0)
```

In [87]:

```
y_pred[:20]
```

Out[87]:

```
array([[130.02972 ],
       [147.80804 ],
       [114.17725 ],
       [188.1433  ],
       [111.11356 ],
       [114.54795 ],
       [116.98694 ],
       [152.87953 ],
       [116.82906 ],
       [160.26447 ],
       [172.04114 ],
       [248.34659 ],
       [176.55379 ],
       [154.58133 ],
       [190.42662 ],
       [285.1028  ],
       [233.29152 ],
       [333.38208 ],
```

```
       [120.063614],
       [195.56525 ]], dtype=float32)
```

In [88]:
```
#calculate test loss/mse
mean_squared_error(y_pred, y_test)
```

Out[88]: 21.196446387090102

In [ ]:
```
score = model.evaluate(X_test, y_test, verbose=False)
print('Metric Names',model.metrics_names)
print('Test Score:', score)
```

In [90]:
```
score = model.evaluate(X_train, y_train, verbose=False)
print('Metric Names',model.metrics_names)
print('Training Score:', score)
```

```
Metric Names ['loss']
Training Score: 24.01258659362793
```

In [99]:
```
result_array=pd.DataFrame({'y_test':y_test, 'y_predicted':y_pred.ravel(),'Date':
```

In [101…
```
#result_array = result_array.sort_values(by=['Date'])
result_array=result_array.reset_index(drop=True, inplace=False)
result_array
```

Out[101…

	y_test	y_predicted	Date
0	132.85	130.029724	2015-07-21
1	144.49	147.808044	2017-06-28
2	116.44	114.177254	2015-09-25
3	191.81	188.143295	2019-06-10
4	108.91	111.113564	2016-04-08
...	...	...	...
372	112.02	110.918327	2015-12-17
373	205.53	215.097290	2019-08-02
374	148.82	149.203873	2017-07-17
375	273.61	281.391571	2020-04-22
376	284.82	287.908386	2019-12-26

377 rows × 3 columns

In [103…
```
result_array['Date'] =pd.to_datetime(result_array.Date)
```

In [104…
```
result_array=result_array.sort_values(by='Date')
result_array
```

Out[104…

	y_test	y_predicted	Date
45	125.72	124.952179	2015-07-15
357	127.74	126.632896	2015-07-16
0	132.85	130.029724	2015-07-21
128	125.32	126.351768	2015-07-24
241	123.38	124.806816	2015-07-28
...	...	...	...
49	319.25	326.039917	2020-05-29
34	344.72	347.023743	2020-06-12
76	351.46	345.197784	2020-06-16
236	365.00	348.616974	2020-06-24
232	353.25	348.882324	2020-06-29

377 rows × 3 columns

Comments about y_true/y_pred dataframe

The model has a good output. The predicted values of y and close to the true values. From the 10 values shown above, the largest value of |y_pred - y_true| is 12, although most have a difference of less than 5.

In [105…

```
result_array=result_array.reset_index(drop=True, inplace=False)
```

In [106…

```
result_array.iloc[0:,0:2].plot.line(figsize=(13,8))
plt.xticks(np.arange(0, 377, step=20), result_array["Date"].dt.date.iloc[lambda
plt.xlabel('Date')
plt.ylabel('Opening price')
plt.title('Stock price over time')
```

Out[106…

```
Text(0.5, 1.0, 'Stock price over time')
```

Comments about Stock Price over time plot

The plots of y_test(y_true) and y_pred mostly overlap. The largest gaps in both plots occurs sometime in 2020. This could be due to the Coronavirus pandemic.

Model #2

In [108…

```
model_1 = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, lstm_units]
    tf.keras.layers.GRU(75, return_sequences=True, input_shape=(12,1)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GRU(30, return_sequences=True),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GRU(30),
    tf.keras.layers.Dropout(0.2),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])
```

In [109…

```
model_1.compile(optimizer='adam', loss='mean_squared_error')
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3, patience=5, verbose=
```

In [110…

```
model_1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
```

```
14/14 [==============================] - 1s 36ms/step - loss: 383.2633 - val_los
s: 38.9415
Epoch 1498/1500
14/14 [==============================] - 1s 40ms/step - loss: 364.9526 - val_los
s: 36.0181
Epoch 1499/1500
14/14 [==============================] - 1s 36ms/step - loss: 328.2225 - val_los
s: 44.9635
Epoch 1500/1500
14/14 [==============================] - 1s 40ms/step - loss: 292.2680 - val_los
s: 42.8146
CPU times: user 15min 23s, sys: 37.4 s, total: 16min
Wall time: 12min 26s
```

In [112…

```
print(history.history["loss"][-1])
```

```
292.26800537109375
```

In [113…

```
print('training MSE', model_1.evaluate(X_train, y_train, verbose=0))
```

```
training MSE 89.08911895751953
```

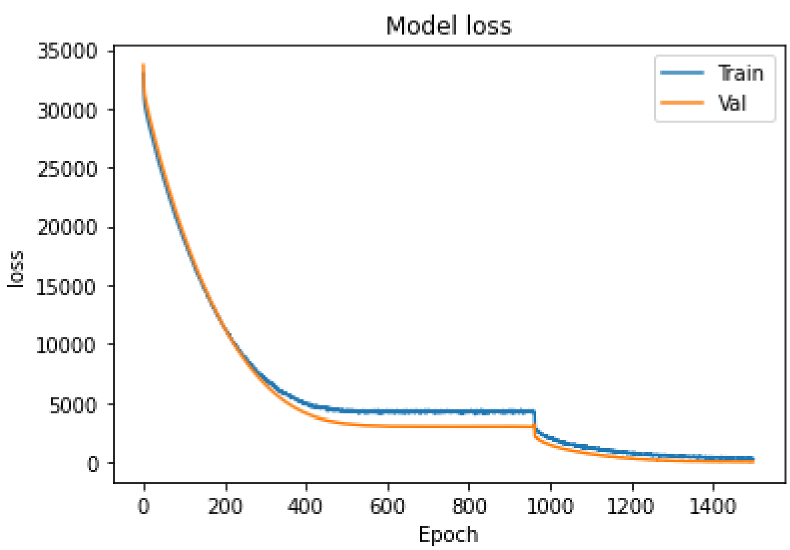
In [114…

```
print(history.history.keys())
```

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

In [115…

```
loss_plot(history)
```



Comment on Model #2 Training Output

- The training and validation losses start at a high value of approximately 35000
- There is a drastic decrease in the first 200 epochs
- The training and validation losses continue to decrease
- The final training loss achieved is approximately 292

In [116…

```
y_test=np.array(y_test)
```

```
y_pred = model_1.predict(X_test, verbose=0)
```

In [117…

```
y_pred[:20]
```

Out[117…

```
array([[125.660934],
       [144.5896  ],
       [110.064964],
       [184.48605 ],
       [107.21594 ],
       [113.757805],
       [112.20062 ],
       [149.78496 ],
       [111.87024 ],
       [156.93988 ],
       [168.75972 ],
       [245.05467 ],
       [173.32153 ],
       [151.28717 ],
       [186.90234 ],
       [283.3356  ],
       [228.92291 ],
       [302.5869  ],
       [116.39398 ],
       [191.75362 ]], dtype=float32)
```

In [118…

```
#calculate test loss/mse
mean_squared_error(y_pred, y_test)
```

Out[118…

```
52.513931808736615
```

In [119…

```
score = model_1.evaluate(X_test, y_test, verbose=False)
print('Metric Names',model_1.metrics_names)
print('Test Score for Model 2:', score)
```

```
Metric Names ['loss']
Test Score for Model 2: 52.5139274597168
```

In [120…

```
score = model_1.evaluate(X_train, y_train, verbose=False)
print('Metric Names',model_1.metrics_names)
print('Training Score for Model 2:', score)
```

```
Metric Names ['loss']
Training Score for Model 2: 89.08911895751953
```

In [121…

```
result_array=pd.DataFrame({'y_test':y_test, 'y_predicted':y_pred.ravel(),'Date':
```

In [ ]:

```
result_array=result_array.reset_index(drop=True, inplace=False)
result_array
```

In [124…

```
result_array['Date'] =pd.to_datetime(result_array.Date)
```

In [125…

```
result_array=result_array.sort_values(by='Date')
```

```
result_array
```

Out [125…

	y_test	y_predicted	Date
45	125.72	120.705017	2015-07-15
357	127.74	122.633720	2015-07-16
0	132.85	125.660934	2015-07-21
128	125.32	123.803505	2015-07-24
241	123.38	120.971870	2015-07-28
...	...	...	...
49	319.25	302.511902	2020-05-29
34	344.72	302.652496	2020-06-12
76	351.46	302.647675	2020-06-16
236	365.00	302.658325	2020-06-24
232	353.25	302.659149	2020-06-29

377 rows × 3 columns

Comments about y_true/y_pred dataframe

The model has a fairly good output, though not as good as Model #1. The predicted values of y are close to the true values. From the 10 values shown above, there are much larger values of |y_pred - y_true| than model #1. Some datapoints have a difference of over 40.

In [126…

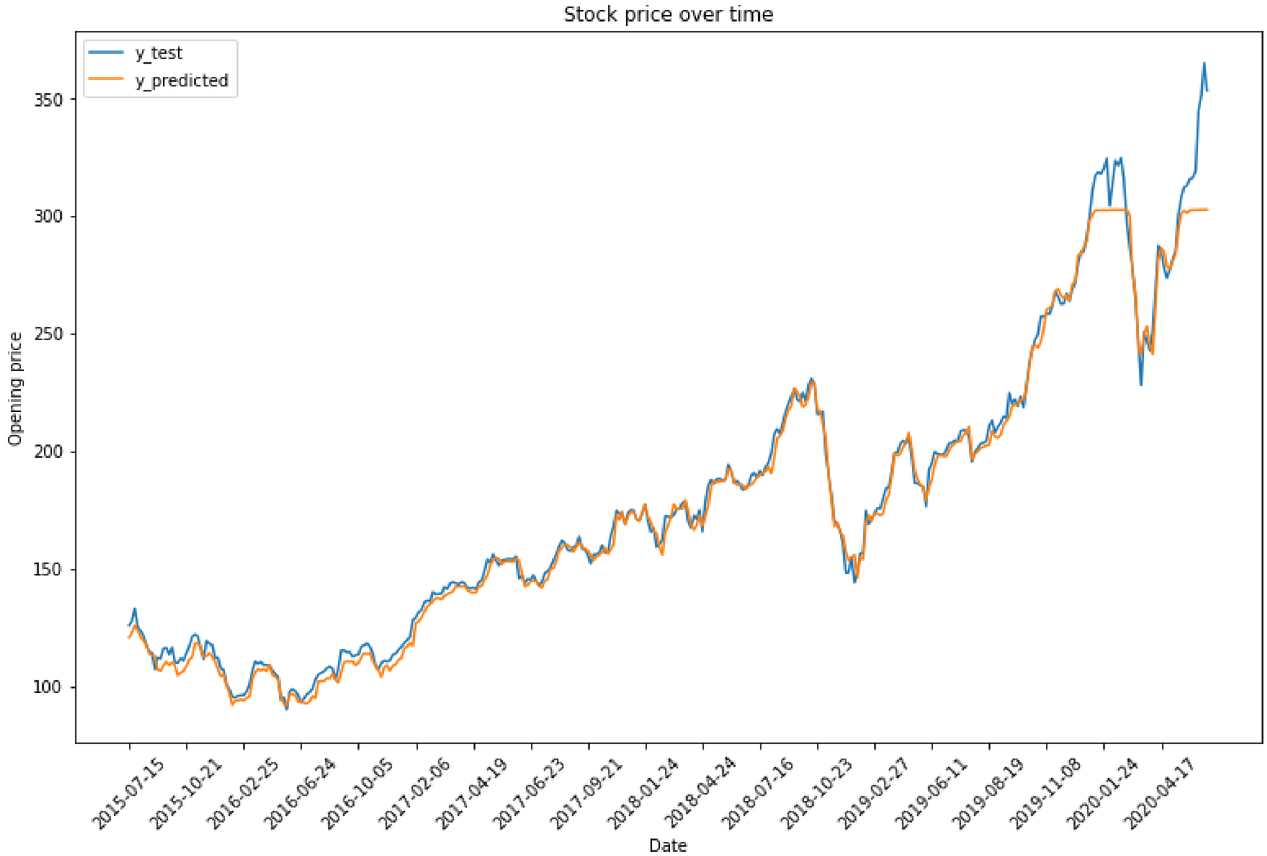
```
result_array=result_array.reset_index(drop=True, inplace=False)
```

In [127…

```
result_array.iloc[0:,0:2].plot.line(figsize=(13,8))
plt.xticks(np.arange(0, 377, step=20), result_array["Date"].dt.date.iloc[lambda
plt.xlabel('Date')
plt.ylabel('Opening price')
plt.title('Stock price over time')
```

Out [127…

```
Text(0.5, 1.0, 'Stock price over time')
```



Comments about Stock Price over time plot

The plots of y_test(y_true) and y_pred mostly overlap. There is significant variation in the plots in 2019 and 2020. The plot for Model #1 seems to be more accurate.

Final Network Architecture

Model #1 is chosen because it has a better performance.

MODEL 1

LSTM LAYER 1 - 50 units --> Dropout 0.2 --> LSTM LAYER 2 - 50 units --> Dropout 0.2 --> LSTM LAYER 3 - 50 units --> Dropout 0.2 --> Dense Layer - 1 unit

This model uses three LSTM layers. 20 % of the nodes at each layer are unused to avoid overfitting and improve model performance.

Optimizer - **Adam**

Loss Metric - **Mean Squared Error**

Activation Function in Dense layer - **Linear**

Batch Size - **64**

Number of Epochs - **1500**

The model utilises Early Stopping in order to converge faster and avoid overfitting.

Effect of Adding More Features

After increasing the features to 40 i.e (using data from the latest 10 days) we observed the following:

- The model trained for a longer time with the same number of epochs.
- The model performance was significantly improved. The training loss was approximately 15 using Model #1 as compared to 189 using Model #1 with 12 Features
- The final plot of predicted values against true values in the test set are almost identical.
- External Resources suggest that the prices and volumes are not the best features for stock prediction. Return value is suggested to be a better input.

References

1 "LSTM Optimizer Choice ?" https://deepdatascience.wordpress.com/2016/11/18/which-lstm-optimizer-to-use/

```
In [ ]:
```