

```

1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import warnings
6 import datetime
7 import seaborn as sns
8 from sklearn.preprocessing import MinMaxScaler
9 from sklearn.decomposition import PCA
10 from sklearn.model_selection import train_test_split
11 from tensorflow.keras import Sequential
12 from tensorflow.keras.layers import Dense
13 from tensorflow.keras.layers import LSTM
14 from tensorflow.keras.layers import SimpleRNN
15 from sklearn.metrics import r2_score

```

## ✓ Loading the Dataset

```

1 data = pd.read_csv('/content/Google_Stock_Price_Train.csv',thousands=',')
2 print(data.head(10))
3 data.shape

```

	Date	Open	High	Low	Close	Volume
0	1/3/2012	325.25	332.83	324.97	663.59	7380500
1	1/4/2012	331.27	333.87	329.08	666.45	5749400
2	1/5/2012	329.83	330.75	326.89	657.21	6590300
3	1/6/2012	328.34	328.77	323.68	648.24	5405900
4	1/9/2012	322.04	322.29	309.46	620.76	11688800
5	1/10/2012	313.70	315.72	307.30	621.43	8824000
6	1/11/2012	310.59	313.52	309.40	624.25	4817800
7	1/12/2012	314.43	315.26	312.08	627.92	3764400
8	1/13/2012	311.96	312.30	309.37	623.28	4631800
9	1/17/2012	314.81	314.81	311.67	626.86	3832800

(1258, 6)

## ✓ Plotting the Dataset

```

1 ax1 = data.plot(x="Date", y=["Open", "High", "Low", "Close"], figsize=(18,10),title='Open, High, Low, Close Stock Prices of Google Stoc
2 ax1.set_ylabel("Stock Price")
3
4 ax2 = data.plot(x="Date", y=["Volume"], figsize=(18,9))
5 ax2.set_ylabel("Stock Volume")
6

```

```
Text(0, 0.5, 'Stock Volume')
```



## ✓ Preprocessing Data

### Checking for Missing values

```
1 # Getting a summary of missing values for each field/attribute
2 print(data.isnull().sum())
```

```

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

```

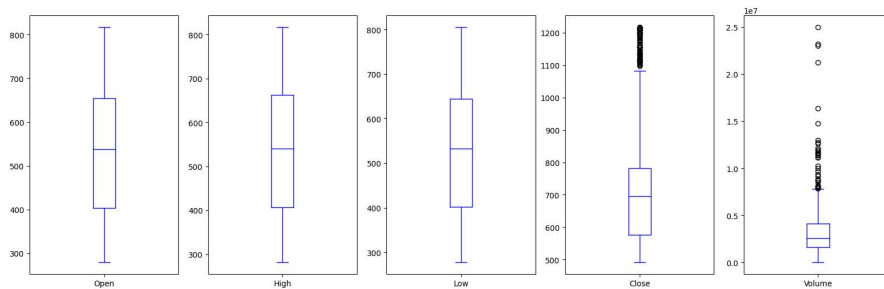
We can see that there are no missing values in the dataset.

### Handling Outliers

```

1 data[['Open', 'High', 'Low', 'Close', 'Volume']].plot(kind= 'box' ,layout=(1,5),subplots=True, sharex=False, sharey=False, figsize=(20,6),co
2 plt.show()

```



We can see that there are outliers in the Close and Volume attributes. However, we will not remove any outliers here since there are only a limited number of datapoints (less than 1300) and if we remove outliers, the dataset will become even smaller.

### Feature Encoding

When we check the 1st 10 records of the dataset, we could see that all the data in the dataset are numerical data, and there are no categorical data that needs encoding. Therefore, no feature encoding process was carried out on this dataset.

### Feature Scaling

MinMaxScaler, which is said to be better to be used with time-series data, was used on this dataset for the feature scaling purposes

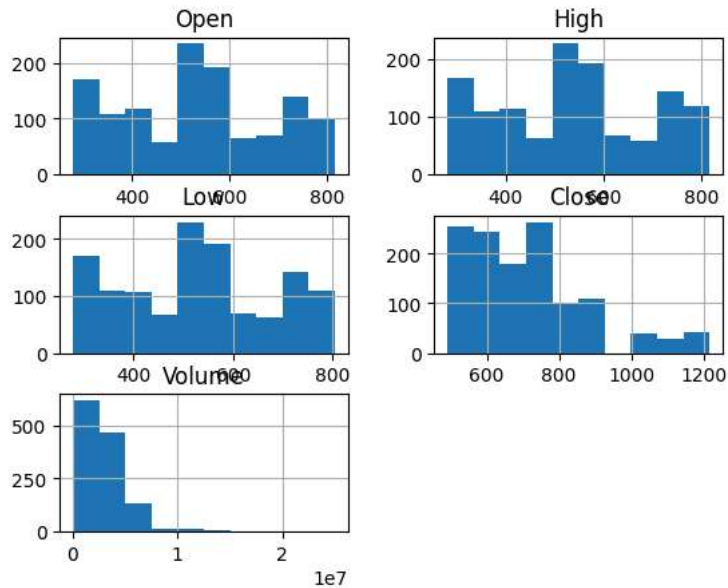
The following set of graphs show the attribute histogram before scaling

```

1 data.hist()

```

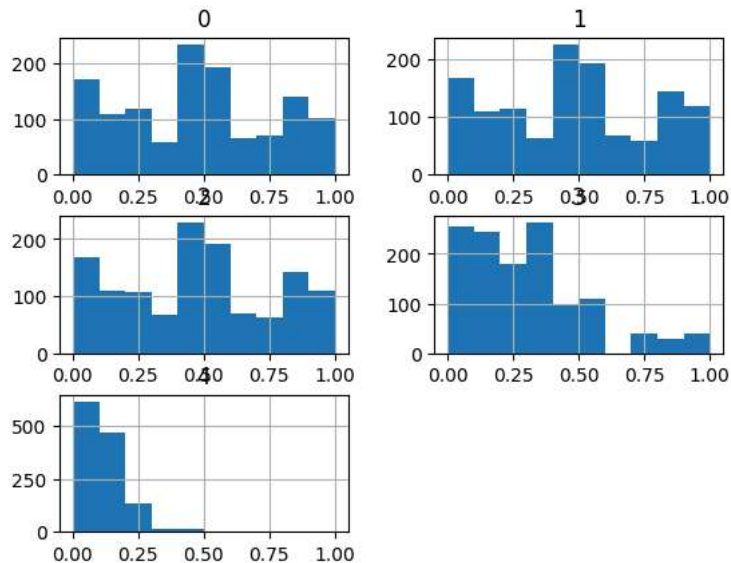
```
array([[<Axes: title={'center': 'Open'}>,
        <Axes: title={'center': 'High'}>],
       [<Axes: title={'center': 'Low'}>,
        <Axes: title={'center': 'Close'}>],
       [<Axes: title={'center': 'Volume'}>, <Axes: >]], dtype=object)
```



```
1 scaler = MinMaxScaler()
2 data_without_date = data[['Open', 'High', 'Low', 'Close', 'Volume']]
3 data_scaled = pd.DataFrame(scaler.fit_transform(data_without_date))
```

```
1 data_scaled.hist()
```

```
array([[<Axes: title={'center': '0'}>, <Axes: title={'center': '1'}>],
       [<Axes: title={'center': '2'}>, <Axes: title={'center': '3'}>],
       [<Axes: title={'center': '4'}>, <Axes: >]], dtype=object)
```



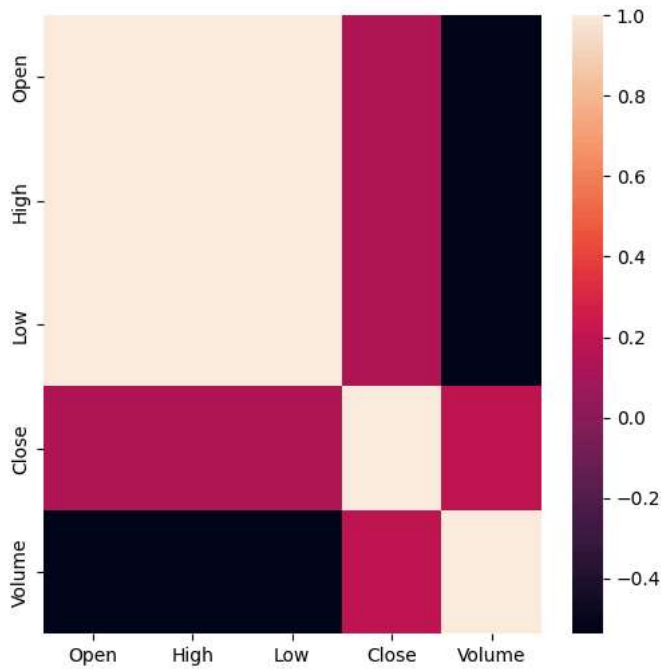
## ✓ Feature Engineering

### Drawing the Correlation Matrix

Correlation Coefficient checking mechanism checks the relationship between the different features with the predicting attribute.

```
1 plt.figure(figsize=(6,6))
2 sns.heatmap(data.corr())
```

```
<ipython-input-9-373595b0d5cd>:2: FutureWarning: The default value of numeric_only in D
sns.heatmap(data.corr())
<Axes: >
```



We can see from the correlation matrix that the features Open, High, and Low, all are highly correlated with each other. Therefore, it is sufficient to consider only one feature from the above 3 features and remove the rest.

Furthermore, I am removing the Close field since it contains many abnormal data values, which is like a contextual outlier.

```
1 #to select final features, we remove open and low because of their perfect correlation (1.0) with High
2 #so we choose High from open low high and also we drop Close because it has a lot of abnormal data values
3 data_scaled=data_scaled.drop([0,2,3], axis=1)
4 data_scaled
```

	1	4	
0	0.096401	0.295258	
1	0.098344	0.229936	
2	0.092517	0.263612	
3	0.088819	0.216179	
4	0.076718	0.467797	
...	...	...	
1253	0.955292	0.024650	
1254	0.964853	0.031286	
1255	0.958074	0.045891	
1256	0.942574	0.029491	
1257	0.936691	0.070569	

1258 rows × 2 columns

Next steps:

[Generate code with data\\_scaled](#)

[View recommended plots](#)

## ✓ Developing the RNN Model

When developing an RNN Model, we have to reshape the data that we are feeding into the model. To do that, first we have to find a pattern in the data available and define the number of timesteps according to the pattern.

When considering the plots we have created for this we could not see a clear cut pattern/trend in the data by just visual inspection. Therefore, I have taken the following code snippet to split the data into a sequence by trying to identify any pattern available.

Since there are two features in our dataset after cleaning, I have used a split sequence method for a multivariate dataset.

```
1 def split_seq_multivariate(sequence, n_past, n_future):
2
3     '''
4     n_past ==> no of past observations
5     n_future ==> no of future observations
6     '''
7     x, y = [], []
8     for window_start in range(len(sequence)):
9         past_end = window_start + n_past
10        future_end = past_end + n_future
11        if future_end > len(sequence):
12            break
13        # slicing the past and future parts of the window
14        past = sequence[window_start:past_end, :]
15        future = sequence[past_end:future_end, -1]
16        x.append(past)
17        y.append(future)
18
19    return np.array(x), np.array(y)
```

In RNN, since we are dealing with time series data, we have to specify how many past data points we will be considering when generating the sequence.

In here, i have taken 60 past data points (time steps) when generating the data sequences.

```
1 # specify the window size
2 n_steps = 60
3
4 data_scaled = data_scaled.to_numpy()
5 data_scaled.shape
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Next, I am using the split\_seq\_multivariate function to split the dataset into sequences.

```
1 # split into samples
2 X, y = split_seq_multivariate(data_scaled, n_steps,1)

3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

### Splitting the Data

In this step, I will be splitting the sequenced data into train and test sections with 0.2 test size.

Furthermore, i have split the training set again to train and validation data. I have not used the test data to do the validation because validation data are used to fine tune the model, and if i used the testing data for validation purposes, then those data will be already seen by the model when trying to predict them later.

Ref: <https://machinelearningmastery.com/difference-test-validation-datasets/>

```

1 # split into train/test
2 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=50)
3
4 print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
5
(958, 60, 2) (240, 60, 2) (958,) (240,)

1 # further dividing the training set into train and validation data
2 X_train, X_val, y_train, y_val = train_test_split(X_train,y_train,test_size=0.2,random_state=30)
3
4 print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)

(766, 60, 2) (192, 60, 2) (766,) (192,)

```

## Define Model

```

1 # Define simple RNN
2 model_rnn = Sequential()
3 model_rnn.add(SimpleRNN(612, input_shape=(n_steps,2))) # Default activation='tanh', recurrent_activation='sigmoid' in keras
4 model_rnn.add(Dense(50, activation='relu'))
5 model_rnn.add(Dense(50, activation='relu'))
6 model_rnn.add(Dense(30, activation='relu'))
7 model_rnn.add(Dense(1))

```

```
1 model_rnn.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
simple_rnn (SimpleRNN)	(None, 612)	376380
dense (Dense)	(None, 50)	30650
dense_1 (Dense)	(None, 50)	2550
dense_2 (Dense)	(None, 30)	1530
dense_3 (Dense)	(None, 1)	31
=====		
Total params: 411141 (1.57 MB)		
Trainable params: 411141 (1.57 MB)		
Non-trainable params: 0 (0.00 Byte)		
=====		

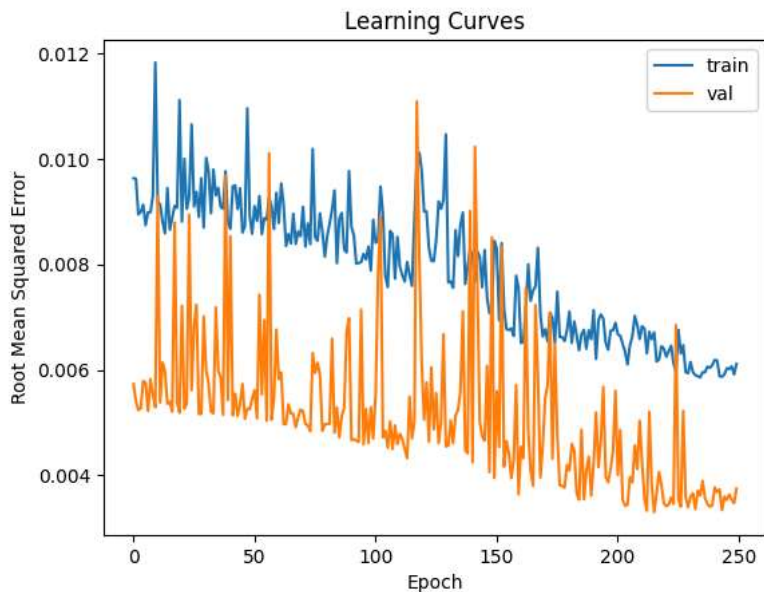
```

1 # compile the model
2 model_rnn.compile(optimizer='adam', loss='mse', metrics=['mae'])

1 # fit the model
2 history = model_rnn.fit(X_train, y_train, epochs=250, batch_size=32, verbose=0, validation_data=(X_val, y_val)) # has used mini batch 1

1 from matplotlib import pyplot
2 # plot learning curves
3 pyplot.title('Learning Curves')
4 pyplot.xlabel('Epoch')
5 pyplot.ylabel('Root Mean Squared Error')
6 pyplot.plot(history.history['loss'], label='train')
7 pyplot.plot(history.history['val_loss'], label='val')
8 pyplot.legend()
9 pyplot.show()

```



```
1 # evaluate the model
2 mse, mae = model_rnn.evaluate(X_test, y_test, verbose=0)
3 print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, np.sqrt(mse), mae))
```

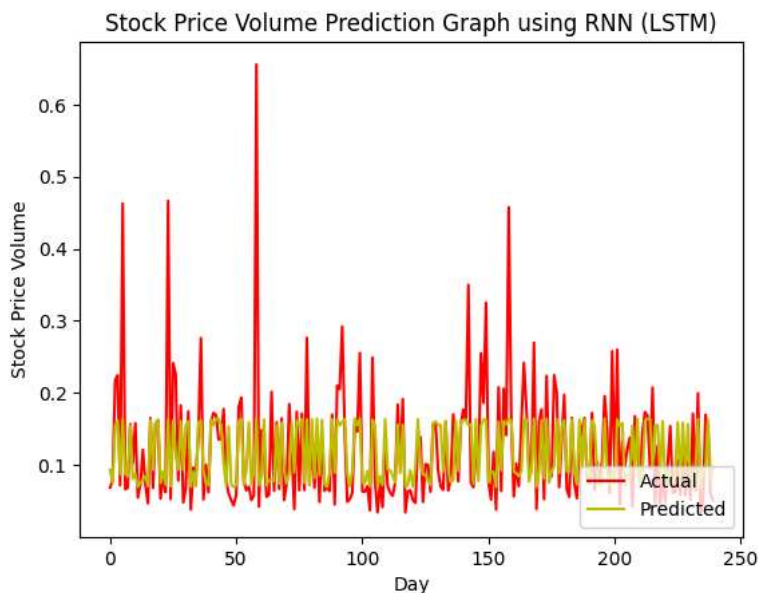
MSE: 0.004, RMSE: 0.067, MAE: 0.041

```
1 # predicting y_test values
2 print('Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence): ',X_test.shape)
3 predicted_values = model_rnn.predict(X_test)
4 print('Shape of Predicted Values: ',predicted_values.shape)
```

Shape of test features (# of data samples) \* (# of dimensions per data point) \* (length of sequence): (240, 60, 2)  
 8/8 [=====] - 1s 49ms/step  
 Shape of Predicted Values: (240, 1)

```
1 plt.plot(y_test,c = 'r')
2 plt.plot(predicted_values,c = 'y')
3 plt.xlabel('Day')
4 plt.ylabel('Stock Price Volume')
5 plt.title('Stock Price Volume Prediction Graph using RNN (LSTM)')
6 plt.legend(['Actual','Predicted'],loc = 'lower right')
7 plt.figure(figsize=(10,6))
```

<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>



```

1 # evaluating using R squared
2 R_square = r2_score(y_test, predicted_values)
3
4 print(R_square)

0.3328745729611804

```

## ✓ Why LSTM over RNN?

- To solve vanishing and exploding gradients problem
- To tackle overfitting in RNN
- To get the Forget Gate Mechanism of LSTM for better predictions
- To Maintain cell state

```

1 # define LSTM model (LSTM is a special type of RNN)
2 model_lstm = Sequential()
3 model_lstm.add(LSTM(612, input_shape=(n_steps,2))) # Default LSTM activation='tanh',recurrent_activation='sigmoid' in keras
4 model_lstm.add(Dense(50, activation='relu'))
5 model_lstm.add(Dense(50, activation='relu'))
6 model_lstm.add(Dense(30, activation='relu'))
7 model_lstm.add(Dense(1))

```

The following code line gives a summary of the model we have created, mentioning each layers information.

```

1 model_lstm.summary()

Model: "sequential"

```

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 612)	1505520
dense (Dense)	(None, 50)	30650
dense_1 (Dense)	(None, 50)	2550
dense_2 (Dense)	(None, 30)	1530
dense_3 (Dense)	(None, 1)	31

```

=====
Total params: 1540281 (5.88 MB)
Trainable params: 1540281 (5.88 MB)
Non-trainable params: 0 (0.00 Byte)

```

## Compiling and Training the Model

```

1 # compile the model
2 model_lstm.compile(optimizer='adam', loss='mse', metrics=['mae'])

1 # fit the model
2 history = model_lstm.fit(X_train, y_train, epochs=250, batch_size=32, verbose=0, validation_data=(X_val, y_val)) # has used mini batch

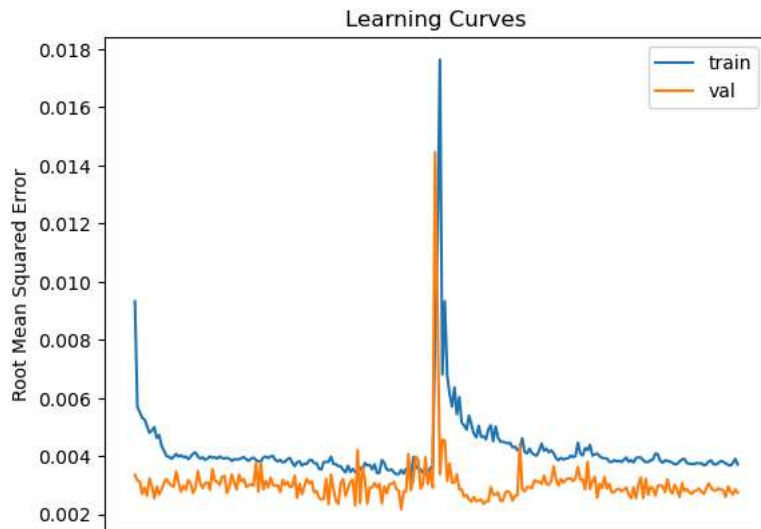
```

We can visualize the training and validation RMSE of the model are as follows.

```

1 from matplotlib import pyplot
2 # plot learning curves
3 pyplot.title('Learning Curves')
4 pyplot.xlabel('Epoch')
5 pyplot.ylabel('Root Mean Squared Error')
6 pyplot.plot(history.history['loss'], label='train')
7 pyplot.plot(history.history['val_loss'], label='val')
8 pyplot.legend()
9 pyplot.show()

```



### Model Evaluation and Predictions

```

1 # evaluate the model
2 mse, mae = model_lstm.evaluate(X_test, y_test, verbose=0)
3 print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, np.sqrt(mse), mae))

MSE: 0.002, RMSE: 0.049, MAE: 0.031

1 # predicting y_test values
2 print("Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence): ",X_test.shape)
3 predicted_values = model_lstm.predict(X_test)
4 print("Shape of Predicted Values: ",predicted_values.shape)

Shape of test features (# of data samples) * (# of dimensions per data point) * (length of sequence): (240, 60, 2)
8/8 [=====] - 0s 30ms/step
Shape of Predicted Values: (240, 1)

1 plt.plot(y_test,c = 'r')
2 plt.plot(predicted_values,c = 'y')
```