In [1]:
```
# LSTM RNN model is used to examine model performance with two data sets with
 contrasting behavior
# Data:
    # dataset 1: used cars monthly sales in millions of dolars from 1992-01-01
to 2019-12-01
        # https://fred.stlouisfed.org/series/MRTSSM44112USN
    # dataset 2: gold price daily in USD from 2015-02-23 to 2020-02-21
        # https://fred.stlouisfed.org/series/GOLDPMGBD228NLBM
# Note: Here we use GPU computing, so processing time will be different for th
ose who use CPU computing
```

In [2]:
```
# import libraries

import torch
import torch.nn as nn

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
%matplotlib inline
sns.set(style = "whitegrid", font_scale = 1.2)

# for plotting datetime values with matplotlib
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

# ignore non-critical warnings
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# read dataset 1 csv file

data_1 = pd.read_csv('used_car_sales.csv', index_col = 0, parse_dates = True)
# set date column as index
data_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 336 entries, 1992-01-01 to 2019-12-01
Data columns (total 1 columns):
MRTSSM44112USN    336 non-null int64
dtypes: int64(1)
memory usage: 5.2 KB
```

In [4]:
```
# there are 336 non-null entries of type int64
```

In [5]:
```
# call first 10 entries

data_1.head(10)
```

Out[5]:

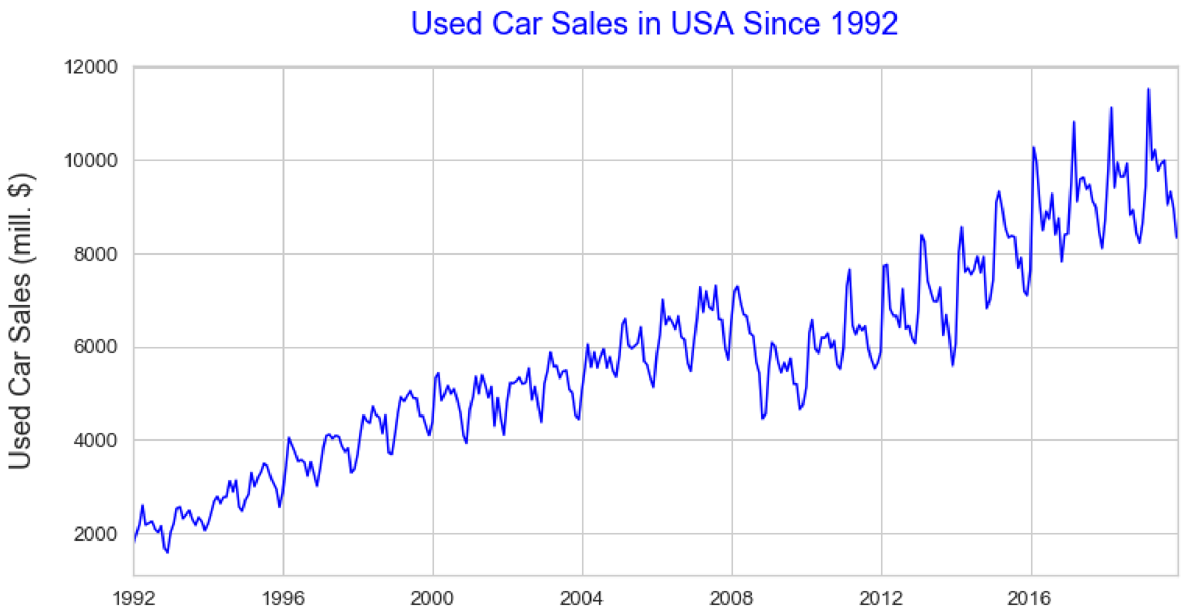
	MRTSSM44112USN
DATE	
1992-01-01	1744
1992-02-01	1990
1992-03-01	2177
1992-04-01	2601
1992-05-01	2171
1992-06-01	2207
1992-07-01	2251
1992-08-01	2087
1992-09-01	2016
1992-10-01	2149

In [6]:
```
# call last 10 entries

data_1.tail(10)
```

Out[6]:

	MRTSSM44112USN
DATE	
2019-03-01	11525
2019-04-01	10009
2019-05-01	10223
2019-06-01	9770
2019-07-01	9916
2019-08-01	9998
2019-09-01	9043
2019-10-01	9326
2019-11-01	8969
2019-12-01	8336

In [7]:
```
# plot data

plt.figure(figsize = (12,6))
plt.plot(data_1.index, data_1['MRTSSM44112USN'], c = 'blue')
plt.autoscale(axis='x',tight=True)
plt.ylabel('Used Car Sales (mill. $)', fontsize = 18, labelpad = 15)
plt.title('Used Car Sales in USA Since 1992', fontsize = 20, pad = 20, color =
'blue')
plt.show()
```



In [8]:
```
# plot shows highly cyclical data with a yearly cycle
# the big drop at 2008 corresponds to the 2008-2009 reccession
```

In [9]:
```
# read dataset 2 csv file

data_2 = pd.read_csv('GOLDPMGBD228NLBM.csv', index_col = 0, parse_dates = True
) # set date column as index

data_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1305 entries, 2015-02-23 to 2020-02-21
Data columns (total 1 columns):
GOLDPMGBD228NLBM    1305 non-null object
dtypes: object(1)
memory usage: 20.4+ KB
```

In [10]:
```
# data have 1305 entries
# the data have missing entries since five years will result in 5 * 365 = 1825
data points
```

In [11]:
```
# call first 10 entries

data_2.head(10)
```

Out[11]:

DATE	GOLDPMGBD228NLBM
2015-02-23	1204.500
2015-02-24	1192.500
2015-02-25	1204.750
2015-02-26	1208.250
2015-02-27	1214.000
2015-03-02	1212.500
2015-03-03	1212.750
2015-03-04	1199.500
2015-03-05	1202.000
2015-03-06	1175.750

In [12]:
```
# call last 10 entries

data_2.tail(10)
```

Out[12]:

DATE	GOLDPMGBD228NLBM
2020-02-10	1573.20
2020-02-11	1570.50
2020-02-12	1563.70
2020-02-13	1575.05
2020-02-14	1581.40
2020-02-17	1580.80
2020-02-18	1589.85
2020-02-19	1604.20
2020-02-20	1619.00
2020-02-21	1643.30

In [13]:
```
# select gold prices column to work with

y = data_2.iloc[:, -1].values
y
```

Out[13]:
```
array(['1204.500', '1192.500', '1204.750', ..., '1604.20', '1619.00',
       '1643.30'], dtype=object)
```

In [14]:
```
# gold price values are in string format --> need to convert to floats
# however, strings of the type 'x.y' cannot be converted directly
# we will use split('.') and select only the digits before the decimal point
    # all values are >= 1000, thus the error introduced is negligible
```

In [15]:
```
# convert from strings to floats and at the same time check for missing values
and impute

count_null = 0 # set counter for null values

for i in range(len(y)):
    if y[i] == '.':
        y[i] = round(np.mean(y[i-10: i]), 1) # impute with 10-day running avg
        count_null = count_null + 1 # update null counter

    else:
        y[i] = y[i].split('.')[0] # split the string at '.' and drop the digit
s after the decimal point
        y[i] = float(y[i])

    print(i)
    print(y[i])
```

```
0
1204.0
1
1192.0
2
1204.0
3
1208.0
4
1214.0
5
1212.0
6
1212.0
7
1199.0
8
1202.0
9
1175.0
10
1168.0
11
1162.0
12
1150.0
13
1152.0
14
1152.0
15
1150.0
16
1150.0
17
1147.0
18
1166.0
19
1183.0
20
1186.0
21
1191.0
22
1195.0
23
1203.0
24
1195.0
25
1185.0
26
1187.0
27
1197.0
28
```

```
1198.0
29
1192.0
30
1192.9
31
1211.0
32
1207.0
33
1194.0
34
1207.0
35
1198.0
36
1194.0
37
1192.0
38
1204.0
39
1203.0
40
1196.0
41
1195.0
42
1189.0
43
1185.0
44
1183.0
45
1200.0
46
1209.0
47
1209.0
48
1180.0
49
1175.0
50
1192.1
51
1197.0
52
1194.0
53
1187.0
54
1186.0
55
1189.0
56
1191.0
```

```
57
1210.0
58
1225.0
59
1220.0
60
1223.0
61
1214.0
62
1210.0
63
1205.0
64
1204.0
65
1209.1
66
1185.0
67
1185.0
68
1185.0
69
1191.0
70
1199.0
71
1192.0
72
1190.0
73
1176.0
74
1164.0
75
1172.0
76
1177.0
77
1188.0
78
1178.0
79
1182.0
80
1181.0
81
1177.0
82
1178.0
83
1201.0
84
1203.0
85
```

```
1185.0
86
1178.0
87
1173.0
88
1172.0
89
1170.0
90
1176.0
91
1171.0
92
1168.0
93
1165.0
94
1167.0
95
1166.0
96
1156.0
97
1158.0
98
1164.0
99
1159.0
100
1154.0
101
1157.0
102
1147.0
103
1144.0
104
1132.0
105
1104.0
106
1105.0
107
1088.0
108
1097.0
109
1080.0
110
1100.0
111
1096.0
112
1090.0
113
1087.0
```

```
114
1098.0
115
1091.0
116
1090.0
117
1085.0
118
1089.0
119
1093.0
120
1097.0
121
1108.0
122
1119.0
123
1116.0
124
1118.0
125
1118.0
126
1111.0
127
1126.0
128
1147.0
129
1156.0
130
1166.0
131
1137.0
132
1120.0
133
1119.0
134
1135.0
135
1133.5
136
1142.0
137
1137.0
138
1128.0
139
1118.0
140
1119.0
141
1121.0
142
```

```
1109.0
143
1109.0
144
1100.0
145
1104.0
146
1105.0
147
1117.0
148
1117.0
149
1141.0
150
1133.0
151
1122.0
152
1131.0
153
1154.0
154
1146.0
155
1131.0
156
1132.0
157
1114.0
158
1119.0
159
1140.0
160
1139.0
161
1147.0
162
1144.0
163
1140.0
164
1151.0
165
1164.0
166
1165.0
167
1173.0
168
1184.0
169
1180.0
170
1175.0
```

```
171
1177.0
172
1167.0
173
1167.0
174
1161.0
175
1166.0
176
1165.0
177
1179.0
178
1148.0
179
1142.0
180
1134.0
181
1123.0
182
1114.0
183
1106.0
184
1088.0
185
1089.0
186
1087.0
187
1085.0
188
1087.0
189
1081.0
190
1084.0
191
1079.0
192
1067.0
193
1082.0
194
1081.0
195
1070.0
196
1076.0
197
1068.0
198
1071.0
199
```

```
1057.0
200
1061.0
201
1065.0
202
1055.0
203
1055.0
204
1079.0
205
1075.0
206
1072.0
207
1081.0
208
1071.0
209
1072.0
210
1068.0
211
1061.0
212
1075.0
213
1049.0
214
1062.0
215
1078.0
216
1074.0
217
1068.0
218
1067.8
219
1067.5
220
1067.0
221
1070.0
222
1060.0
223
1066.3
224
1068.1
225
1082.0
226
1077.0
227
1091.0
```

```
228
1106.0
229
1101.0
230
1100.0
231
1085.0
232
1088.0
233
1088.0
234
1093.0
235
1089.0
236
1086.0
237
1101.0
238
1096.0
239
1096.0
240
1106.0
241
1113.0
242
1116.0
243
1114.0
244
1111.0
245
1126.0
246
1128.0
247
1132.0
248
1156.0
249
1150.0
250
1193.0
251
1191.0
252
1190.0
253
1241.0
254
1239.0
255
1208.0
256
```

```
1209.0
257
1210.0
258
1210.0
259
1231.0
260
1211.0
261
1221.0
262
1250.0
263
1236.0
264
1226.0
265
1234.0
266
1236.0
267
1239.0
268
1250.0
269
1277.0
270
1267.0
271
1267.0
272
1246.0
273
1266.0
274
1264.0
275
1242.0
276
1232.0
277
1228.0
278
1266.0
279
1252.0
280
1244.0
281
1252.0
282
1217.0
283
1221.0
284
1241.8
```

```
285
1239.6
286
1226.0
287
1236.0
288
1237.0
289
1213.0
290
1219.0
291
1231.0
292
1221.0
293
1242.0
294
1239.0
295
1254.0
296
1254.0
297
1245.0
298
1233.0
299
1227.0
300
1234.0
301
1255.0
302
1252.0
303
1249.0
304
1243.0
305
1238.0
306
1241.0
307
1247.0
308
1256.0
309
1285.0
310
1250.0
311
1294.0
312
1283.0
313
```

```
1280.0
314
1289.0
315
1265.0
316
1262.0
317
1276.0
318
1279.0
319
1265.0
320
1285.0
321
1277.0
322
1272.0
323
1246.0
324
1254.0
325
1245.0
326
1236.0
327
1220.0
328
1223.0
329
1216.0
330
1247.4
331
1212.0
332
1214.0
333
1212.0
334
1240.0
335
1244.0
336
1241.0
337
1263.0
338
1263.0
339
1275.0
340
1280.0
341
1287.0
```

```
342
1283.0
343
1310.0
344
1290.0
345
1281.0
346
1272.0
347
1264.0
348
1262.0
349
1315.0
350
1324.0
351
1309.0
352
1321.0
353
1320.0
354
1340.0
355
1350.0
356
1350.0
357
1366.0
358
1356.0
359
1354.0
360
1357.0
361
1342.0
362
1342.0
363
1323.0
364
1327.0
365
1334.0
366
1330.0
367
1315.0
368
1321.0
369
1320.0
370
```

```
1313.0
371
1323.0
372
1329.0
373
1341.0
374
1342.0
375
1349.0
376
1363.0
377
1358.0
378
1362.0
379
1340.0
380
1336.0
381
1341.0
382
1347.0
383
1355.0
384
1352.0
385
1339.0
386
1344.0
387
1343.0
388
1350.0
389
1346.0
390
1335.0
391
1342.0
392
1327.0
393
1321.0
394
1318.0
395
1336.5
396
1318.0
397
1309.0
398
1309.0
```

```
399
1324.0
400
1326.0
401
1337.0
402
1348.0
403
1343.0
404
1330.0
405
1324.0
406
1323.0
407
1321.0
408
1310.0
409
1308.0
410
1314.0
411
1313.0
412
1326.0
413
1339.0
414
1338.0
415
1340.0
416
1327.0
417
1322.0
418
1318.0
419
1322.0
420
1313.0
421
1283.0
422
1269.0
423
1254.0
424
1258.0
425
1259.0
426
1253.0
427
```

```
1256.0
428
1261.0
429
1251.0
430
1254.0
431
1258.0
432
1269.0
433
1271.0
434
1266.0
435
1265.0
436
1269.0
437
1270.0
438
1266.0
439
1273.0
440
1272.0
441
1288.0
442
1303.0
443
1301.0
444
1302.0
445
1283.0
446
1282.0
447
1281.0
448
1267.0
449
1236.0
450
1213.0
451
1226.0
452
1229.0
453
1226.0
454
1211.0
455
1214.0
```

```
456
1212.0
457
1185.0
458
1186.0
459
1187.0
460
1187.0
461
1186.0
462
1178.0
463
1161.0
464
1173.0
465
1162.0
466
1172.0
467
1177.0
468
1171.0
469
1163.0
470
1156.0
471
1158.0
472
1162.0
473
1126.0
474
1131.0
475
1136.0
476
1125.0
477
1133.0
478
1131.0
479
1142.1
480
1140.0
481
1138.4
482
1134.0
483
1145.0
484
```

```
1135.5
485
1136.0
486
1151.0
487
1164.0
488
1176.0
489
1175.0
490
1178.0
491
1189.0
492
1178.0
493
1205.0
494
1190.0
495
1203.0
496
1216.0
497
1214.0
498
1196.0
499
1200.0
500
1212.0
501
1216.0
502
1195.0
503
1189.0
504
1184.0
505
1192.0
506
1212.0
507
1203.0
508
1221.0
509
1215.0
510
1226.0
511
1231.0
512
1242.0
```

```
513
1236.0
514
1228.0
515
1222.0
516
1230.0
517
1224.0
518
1240.0
519
1241.0
520
1237.0
521
1233.0
522
1236.0
523
1247.0
524
1253.0
525
1257.0
526
1255.0
527
1240.0
528
1238.0
529
1226.0
530
1230.0
531
1216.0
532
1209.0
533
1206.0
534
1202.0
535
1204.0
536
1204.0
537
1198.0
538
1229.0
539
1229.0
540
1232.0
541
```

```
1241.0
542
1249.0
543
1247.0
544
1247.0
545
1257.0
546
1257.0
547
1251.0
548
1248.0
549
1244.0
550
1247.0
551
1257.0
552
1245.0
553
1252.0
554
1266.0
555
1250.0
556
1252.0
557
1274.0
558
1284.0
559
1257.1
560
1258.4
561
1278.0
562
1279.0
563
1282.0
564
1281.0
565
1269.0
566
1267.0
567
1261.0
568
1262.0
569
1266.0
```

```
570
1270.3
571
1255.0
572
1250.0
573
1228.0
574
1228.0
575
1229.0
576
1220.0
577
1222.0
578
1223.0
579
1231.0
580
1233.0
581
1234.0
582
1257.0
583
1255.0
584
1252.0
585
1258.0
586
1260.0
587
1252.0
588
1256.0
589
1265.0
590
1252.2
591
1262.0
592
1266.0
593
1264.0
594
1274.0
595
1279.0
596
1293.0
597
1291.0
598
```

```
1273.0
599
1266.0
600
1266.0
601
1262.0
602
1275.0
603
1254.0
604
1255.0
605
1248.0
606
1242.0
607
1242.0
608
1250.0
609
1255.0
610
1245.0
611
1249.0
612
1248.0
613
1243.0
614
1242.0
615
1229.0
616
1223.0
617
1220.0
618
1224.0
619
1215.0
620
1211.0
621
1211.0
622
1218.0
623
1218.0
624
1230.0
625
1234.0
626
1240.0
```

```
627
1242.0
628
1238.0
629
1248.0
630
1255.0
631
1254.0
632
1248.0
633
1261.0
634
1264.0
635
1267.0
636
1270.0
637
1269.0
638
1268.0
639
1257.0
640
1258.0
641
1261.0
642
1271.0
643
1284.0
644
1286.0
645
1282.0
646
1270.6
647
1272.0
648
1285.0
649
1295.0
650
1292.0
651
1284.0
652
1286.0
653
1289.0
654
1285.0
655
```

```
1284.1
656
1318.0
657
1308.0
658
1311.0
659
1320.0
660
1333.0
661
1335.0
662
1337.0
663
1343.0
664
1346.0
665
1334.0
666
1326.0
667
1327.0
668
1324.0
669
1322.0
670
1312.0
671
1309.0
672
1311.0
673
1292.0
674
1294.0
675
1293.0
676
1300.0
677
1282.0
678
1283.0
679
1283.0
680
1273.0
681
1271.0
682
1274.0
683
1274.0
```

```
684
1261.0
685
1278.0
686
1291.0
687
1289.0
688
1290.0
689
1299.0
690
1303.0
691
1284.0
692
1280.0
693
1286.0
694
1281.0
695
1274.0
696
1276.0
697
1275.0
698
1273.0
699
1266.0
700
1272.0
701
1270.0
702
1277.0
703
1279.0
704
1267.0
705
1270.0
706
1275.0
707
1284.0
708
1284.0
709
1284.0
710
1277.0
711
1274.0
712
```

```
1282.0
713
1280.0
714
1284.0
715
1286.0
716
1283.0
717
1286.0
718
1290.0
719
1290.0
720
1294.0
721
1291.0
722
1283.0
723
1280.0
724
1275.0
725
1273.0
726
1266.0
727
1263.0
728
1255.0
729
1250.0
730
1247.0
731
1240.0
732
1242.0
733
1251.0
734
1254.0
735
1260.0
736
1260.0
737
1264.0
738
1264.0
739
1253.2
740
1253.5
```

```
741
1254.2
742
1279.0
743
1291.0
744
1263.3
745
1264.2
746
1312.0
747
1314.0
748
1314.0
749
1317.0
750
1319.0
751
1311.0
752
1319.0
753
1323.0
754
1326.0
755
1339.0
756
1333.0
757
1335.0
758
1332.0
759
1334.0
760
1332.0
761
1333.0
762
1353.0
763
1354.0
764
1353.0
765
1343.0
766
1344.0
767
1345.0
768
1341.0
769
```

```
1331.0
770
1333.0
771
1331.0
772
1324.0
773
1315.0
774
1314.0
775
1322.0
776
1325.0
777
1336.0
778
1352.0
779
1352.0
780
1346.0
781
1339.0
782
1330.0
783
1328.0
784
1327.0
785
1333.0
786
1325.0
787
1317.0
788
1307.0
789
1322.0
790
1320.0
791
1331.0
792
1329.0
793
1321.0
794
1320.0
795
1319.0
796
1322.0
797
1323.0
```

```
798
1318.0
799
1310.0
800
1312.0
801
1311.0
802
1321.0
803
1329.0
804
1346.0
805
1352.0
806
1341.0
807
1332.0
808
1323.0
809
1327.7
810
1329.5
811
1333.0
812
1337.0
813
1327.0
814
1331.0
815
1331.0
816
1338.0
817
1350.0
818
1341.0
819
1343.0
820
1349.0
821
1342.0
822
1351.0
823
1348.0
824
1336.0
825
1324.0
826
```

```
1328.0
827
1321.0
828
1320.0
829
1321.0
830
1313.0
831
1307.0
832
1304.0
833
1315.0
834
1309.0
835
1316.2
836
1306.0
837
1313.0
838
1318.0
839
1324.0
840
1319.0
841
1295.0
842
1291.0
843
1289.0
844
1288.0
845
1288.0
846
1293.0
847
1289.0
848
1304.0
849
1303.0
850
1295.9
851
1295.0
852
1300.0
853
1305.0
854
1294.0
```

```
855
1295.0
856
1292.0
857
1300.0
858
1297.0
859
1298.0
860
1299.0
861
1298.0
862
1296.0
863
1302.0
864
1285.0
865
1281.0
866
1276.0
867
1274.0
868
1266.0
869
1269.0
870
1268.0
871
1260.0
872
1254.0
873
1251.0
874
1250.0
875
1247.0
876
1251.0
877
1255.0
878
1255.0
879
1255.0
880
1262.0
881
1254.0
882
1251.0
883
```

```
1245.0
884
1241.0
885
1241.0
886
1232.0
887
1224.0
888
1217.0
889
1228.0
890
1224.0
891
1228.0
892
1231.0
893
1228.0
894
1223.0
895
1223.0
896
1220.0
897
1219.0
898
1215.0
899
1216.0
900
1209.0
901
1212.0
902
1209.0
903
1214.0
904
1214.0
905
1200.0
906
1197.0
907
1182.0
908
1180.0
909
1178.0
910
1184.0
911
1190.0
```

```
912
1196.0
913
1192.0
914
1197.0
915
1189.6
916
1212.0
917
1204.0
918
1197.0
919
1202.0
920
1200.0
921
1190.0
922
1196.0
923
1205.0
924
1198.0
925
1196.0
926
1189.0
927
1195.0
928
1209.0
929
1201.0
930
1201.0
931
1200.0
932
1203.0
933
1208.0
934
1198.0
935
1202.0
936
1201.0
937
1194.0
938
1185.0
939
1187.0
940
```

```
1189.0
941
1204.0
942
1201.0
943
1203.0
944
1203.0
945
1186.0
946
1185.0
947
1188.0
948
1205.0
949
1219.0
950
1229.0
951
1230.0
952
1229.0
953
1223.0
954
1227.0
955
1222.0
956
1235.0
957
1230.0
958
1230.0
959
1233.0
960
1230.0
961
1225.0
962
1214.0
963
1231.0
964
1232.0
965
1232.0
966
1231.0
967
1229.0
968
1224.0
```

```
969
1211.0
970
1205.0
971
1202.0
972
1203.0
973
1211.0
974
1222.0
975
1221.0
976
1223.0
977
1226.0
978
1227.0
979
1223.0
980
1223.0
981
1221.0
982
1213.0
983
1226.0
984
1217.0
985
1230.0
986
1240.0
987
1235.0
988
1242.0
989
1243.0
990
1245.0
991
1245.0
992
1245.0
993
1242.0
994
1235.0
995
1241.0
996
1246.0
997
```

```
1255.0
998
1259.0
999
1258.0
1000
1247.1
1001
1247.3
1002
1247.5
1003
1268.0
1004
1279.0
1005
1254.8
1006
1256.2
1007
1282.0
1008
1290.0
1009
1279.0
1010
1292.0
1011
1286.0
1012
1288.0
1013
1291.0
1014
1288.0
1015
1292.0
1016
1294.0
1017
1292.0
1018
1290.0
1019
1284.0
1020
1279.0
1021
1282.0
1022
1279.0
1023
1283.0
1024
1293.0
1025
1302.0
```

```
1026
1307.0
1027
1310.0
1028
1323.0
1029
1318.0
1030
1312.0
1031
1314.0
1032
1312.0
1033
1310.0
1034
1314.0
1035
1306.0
1036
1310.0
1037
1312.0
1038
1311.0
1039
1316.0
1040
1325.0
1041
1334.0
1042
1343.0
1043
1331.0
1044
1329.0
1045
1331.0
1046
1325.0
1047
1322.0
1048
1319.0
1049
1311.0
1050
1285.0
1051
1283.0
1052
1285.0
1053
1285.0
1054
```

```
1296.0
1055
1292.0
1056
1297.0
1057
1306.0
1058
1295.0
1059
1303.0
1060
1305.0
1061
1307.0
1062
1303.0
1063
1309.0
1064
1311.0
1065
1319.0
1066
1316.0
1067
1309.0
1068
1295.0
1069
1295.0
1070
1293.0
1071
1290.0
1072
1290.0
1073
1283.0
1074
1288.0
1075
1300.0
1076
1303.0
1077
1305.0
1078
1298.0
1079
1294.0
1080
1285.0
1081
1276.0
1082
1275.0
```

```
1083
1275.0
1084
1289.9
1085
1290.1
1086
1269.0
1087
1271.0
1088
1280.0
1089
1284.0
1090
1279.0
1091
1282.0
1092
1283.0
1093
1270.0
1094
1278.0
1095
1278.6
1096
1281.0
1097
1285.0
1098
1286.0
1099
1287.0
1100
1295.0
1101
1298.0
1102
1299.0
1103
1291.0
1104
1280.0
1105
1276.0
1106
1271.0
1107
1273.0
1108
1283.0
1109
1282.0
1110
1284.8
1111
```

```
1278.0
1112
1281.0
1113
1280.0
1114
1295.0
1115
1317.0
1116
1324.0
1117
1335.0
1118
1335.0
1119
1340.0
1120
1328.0
1121
1324.0
1122
1332.0
1123
1335.0
1124
1351.0
1125
1341.0
1126
1341.0
1127
1344.0
1128
1379.0
1129
1397.0
1130
1405.0
1131
1431.0
1132
1403.0
1133
1402.0
1134
1409.0
1135
1390.0
1136
1391.0
1137
1413.0
1138
1414.0
1139
1388.0
```

```
1140
1400.0
1141
1391.0
1142
1408.0
1143
1413.0
1144
1407.0
1145
1412.0
1146
1409.0
1147
1410.0
1148
1417.0
1149
1439.0
1150
1427.0
1151
1425.0
1152
1426.0
1153
1416.0
1154
1420.0
1155
1419.0
1156
1425.0
1157
1427.0
1158
1406.0
1159
1441.0
1160
1465.0
1161
1465.0
1162
1506.0
1163
1495.0
1164
1497.0
1165
1504.0
1166
1498.0
1167
1513.0
1168
```

```
1515.0
1169
1515.0
1170
1496.0
1171
1504.0
1172
1503.0
1173
1502.0
1174
1503.0
1175
1505.3
1176
1532.0
1177
1537.0
1178
1540.0
1179
1528.0
1180
1525.0
1181
1537.0
1182
1546.0
1183
1529.0
1184
1523.0
1185
1509.0
1186
1498.0
1187
1490.0
1188
1515.0
1189
1503.0
1190
1497.0
1191
1502.0
1192
1503.0
1193
1500.0
1194
1501.0
1195
1522.0
1196
1520.0
```

```
1197
1528.0
1198
1506.0
1199
1489.0
1200
1485.0
1201
1473.0
1202
1492.0
1203
1517.0
1204
1499.0
1205
1501.0
1206
1505.0
1207
1507.0
1208
1494.0
1209
1479.0
1210
1490.0
1211
1487.0
1212
1485.0
1213
1492.0
1214
1493.9
1215
1491.0
1216
1485.0
1217
1494.0
1218
1496.0
1219
1513.0
1220
1492.0
1221
1486.0
1222
1492.0
1223
1510.0
1224
1508.0
1225
```

```
1509.0
1226
1488.0
1227
1486.0
1228
1484.0
1229
1464.0
1230
1458.0
1231
1452.0
1232
1462.0
1233
1466.0
1234
1466.0
1235
1467.0
1236
1468.0
1237
1471.0
1238
1467.0
1239
1464.0
1240
1458.0
1241
1454.0
1242
1454.0
1243
1454.0
1244
1460.0
1245
1461.0
1246
1477.0
1247
1475.0
1248
1475.0
1249
1459.0
1250
1461.0
1251
1464.0
1252
1466.0
1253
1467.0
```

```
1254
1466.0
1255
1477.0
1256
1475.0
1257
1474.0
1258
1476.0
1259
1479.0
1260
1482.0
1261
1472.6
1262
1473.5
1263
1474.2
1264
1511.0
1265
1514.0
1266
1483.1
1267
1483.9
1268
1527.0
1269
1548.0
1270
1573.0
1271
1567.0
1272
1571.0
1273
1550.0
1274
1553.0
1275
1549.0
1276
1545.0
1277
1549.0
1278
1554.0
1279
1557.0
1280
1560.0
1281
1551.0
1282
```

```
1556.0
1283
1562.0
1284
1564.0
1285
1580.0
1286
1574.0
1287
1573.0
1288
1578.0
1289
1584.0
1290
1574.0
1291
1558.0
1292
1553.0
1293
1563.0
1294
1570.1
1295
1573.0
1296
1570.0
1297
1563.0
1298
1575.0
1299
1581.0
1300
1580.0
1301
1589.0
1302
1604.0
1303
1619.0
1304
1643.0
```

In [16]:

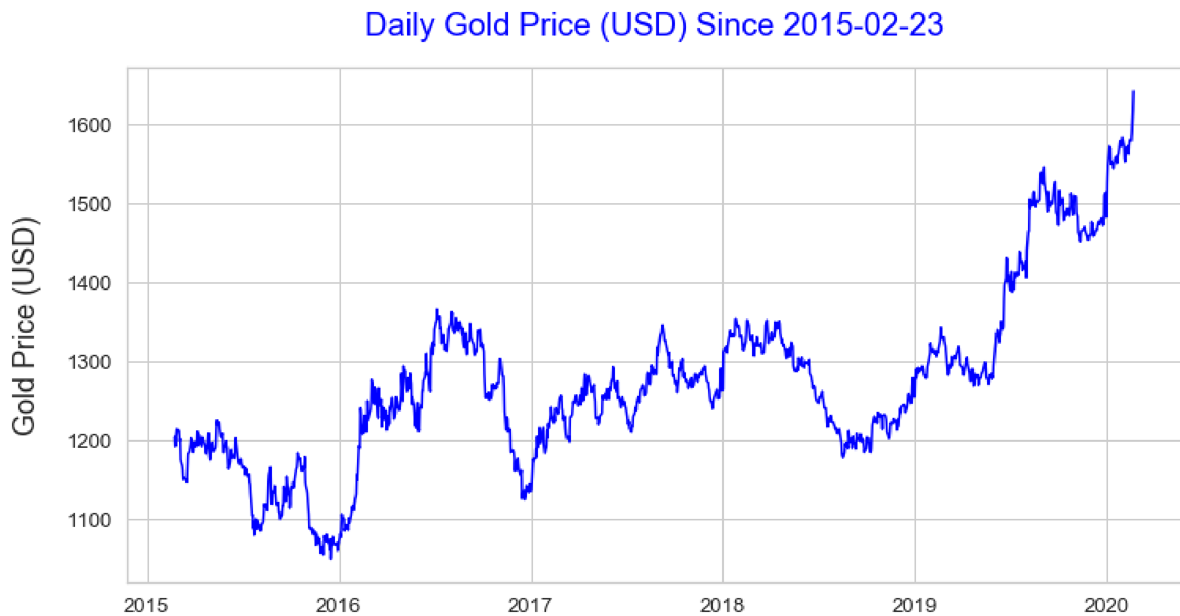
```
# print total null count
print(f'Total Null Count: {count_null}')
```

```
Total Null Count: 53
```

In [17]:

```
# number of nulls is 53 which is small relative to the total number of data po
ints
```

In [18]:
```
# plot gold prices

plt.figure(figsize = (12,6))
plt.plot(data_2.index, y, color = 'blue')
plt.ylabel('Gold Price (USD)', fontsize = 18, labelpad = 15)
plt.title('Daily Gold Price (USD) Since 2015-02-23', fontsize = 20, pad = 20,
color ='blue')
plt.show()
```



In [19]:
```
# in contrast with dataset 1, values here do not have clear cyclical nature an
d rather resemble "random walk"
```

In [20]:
```
# Prepare data
```

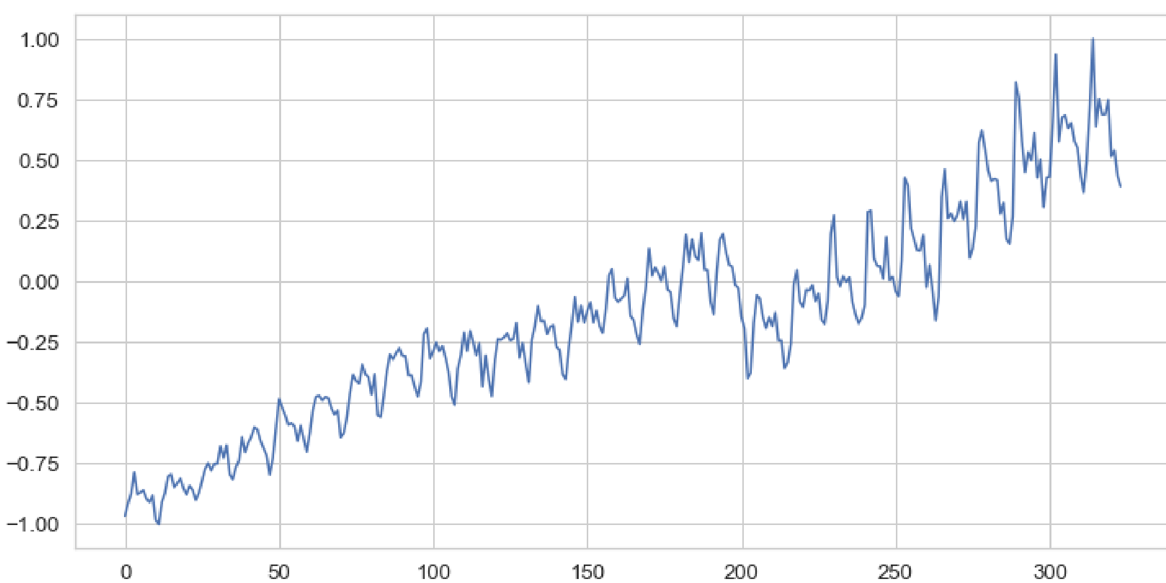
In [21]:
```
# get values from both datasets assigned as y_1 and y_2

# values from dataset 1
y_1 = data_1.iloc[:, -1].values.astype(float)

# values from dataset 2
y_2 = y[0:1290]
# for convenience we select 1290 out of 1305 points to use with a window size
 of 30 (one month) later on
```

In [22]:
```
# create train and test sets from y_1 and y_2

# dataset 1
test_size_1 = 12 # test size corresponds to 1 year

train_set_1 = y_1[:-test_size_1]
test_set_1 = y_1[-test_size_1:]

# dataset 2
test_size_2 = 30 # test size corresponds to 1 month

train_set_2 = y_2[:-test_size_2]
test_set_2 = y_2[-test_size_2:]
```

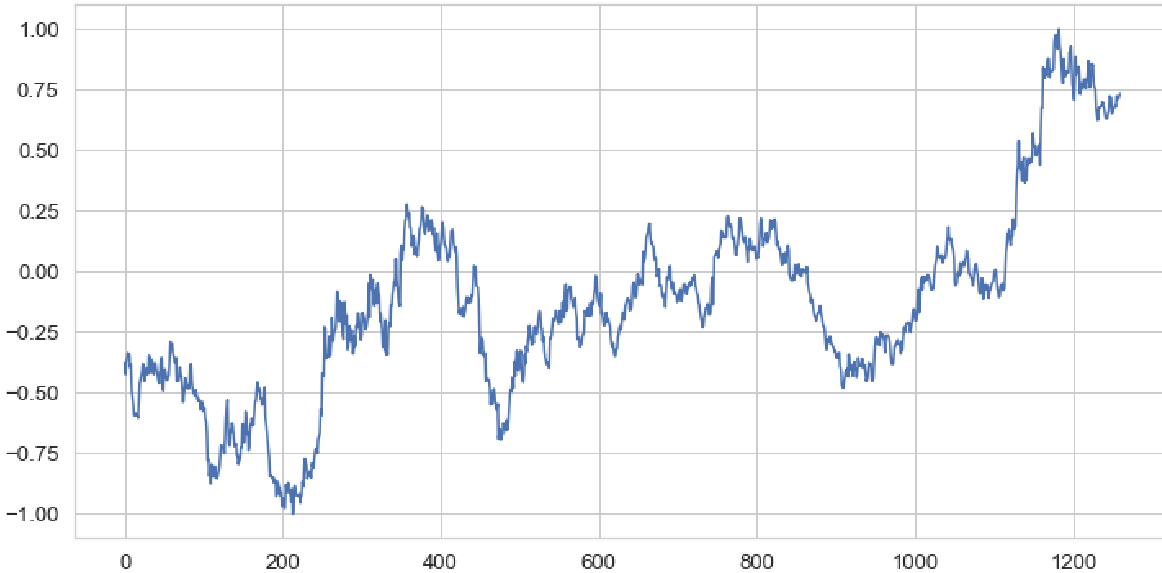
In [23]:
```
# NNs perform better with normalized data --> normalize data using MinMaxScale
r
    # normalize train_set only to avoid information leakage from test_set

from sklearn.preprocessing import MinMaxScaler

# instantiate a scaler with a feature range from -1 to 1
scaler_1 = MinMaxScaler(feature_range=(-1, 1)) # for dataset 1
scaler_2 = MinMaxScaler(feature_range=(-1, 1)) # for dataset 2
```

In [24]:
```
# normalize the training sets

train_set_1 = scaler_1.fit_transform(train_set_1.reshape(-1, 1))

train_set_2 = scaler_2.fit_transform(train_set_2.reshape(-1, 1))
```

In [25]:
```
# plot normalized train set 1

plt.figure(figsize = (12,6))
plt.plot(train_set_1)
plt.show()
```



In [26]:
```
# plot normalized train set 2

plt.figure(figsize = (12,6))
plt.plot(train_set_2)
plt.show()
```



In [27]:
```
# normalized data is bound within -1 and 1, while preserving the ratio between data points
```

In [28]:
```
# Prepare data for LSTM model
```

In [29]:
```
# first, check if GPU computing is available
# torch.cuda.is_available() checks and returns a Boolean True if a GPU is available, else it'll return False

is_cuda = torch.cuda.is_available()

# set device to GPU or CPU depending on the outcome --> we will use this device variable later on in our code
if is_cuda:
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
```

In [30]:
```
# convert train_set_1 and train_set_2 to tensors and set window sizes for both sets
train_set_1 = torch.FloatTensor(train_set_1).view(-1)
train_set_2 = torch.FloatTensor(train_set_2).view(-1)

# window size for dataset 1
window_size_1 = 12 # 1 year

# window size for dataset 2
window_size_2 = 30 # 1 month
```

In [31]:
```
# define function to create seq/label tuples

def input_data(seq, ws):  # ws is the window size
    out = []
    L = len(seq)
    for i in range(L-ws):
        window = seq[i:i+ws]
        label = seq[i+ws:i+ws+1]
        out.append((window, label))
    return out
```

In [32]:
```
# apply the input_data function to train_set_1 and train_set_2

train_data_1 = input_data(train_set_1, window_size_1)
train_data_2 = input_data(train_set_2, window_size_2)
```

In [33]:
```
len(train_data_1)  # this should equal 336 - 12 - 12
```

Out[33]:
```
312
```

In [34]:
```
# show first element of train_data_1
train_data_1[0]
```

Out[34]:
```
(tensor([-0.9663, -0.9148, -0.8756, -0.7868, -0.8768, -0.8693, -0.8601, -0.89
44,
         -0.9093, -0.8815, -0.9824, -1.0000]),
 tensor([-0.9081]))
```

In [35]:
```
# first tensor is the input data for the model
# second tensor is the target value to be predicted by model based on input da
ta
```

In [36]:
```
len(train_data_2)  # this should equal 1290 - 30 - 30
```

Out[36]:
```
1230
```

In [37]:
```
# show first element of train_data_2
train_data_2[0]
```

Out[37]:
```
(tensor([-0.3763, -0.4245, -0.3763, -0.3602, -0.3360, -0.3441, -0.3441, -0.39
64,
         -0.3843, -0.4930, -0.5211, -0.5453, -0.5936, -0.5855, -0.5855, -0.59
36,
         -0.5936, -0.6056, -0.5292, -0.4608, -0.4487, -0.4286, -0.4125, -0.38
03,
         -0.4125, -0.4527, -0.4447, -0.4044, -0.4004, -0.4245]),
 tensor([-0.4209]))
```

In [38]:
```
# Define the LSTM model
```

In [39]:
```
class LSTMnetwork(nn.Module):
    def __init__(self, input_size = 1, hidden_size = 256, output_size = 1): #
 use LSTM layer of size 256
        super().__init__()
        self.hidden_size = hidden_size

        # add an LSTM layer:
        self.lstm = nn.LSTM(input_size, hidden_size)

        # add a fully-connected layer:
        self.linear = nn.Linear(hidden_size, output_size)

        # initialize h0 and c0 -- use .to(device) to select GPU or CPU computa
tion, respectively
        self.hidden = (torch.zeros(1, 1, self.hidden_size).to(device),
                       torch.zeros(1, 1, self.hidden_size).to(device))

    def forward(self, seq):
        lstm_out, self.hidden = self.lstm(seq.view(len(seq), 1, -1), self.hidd
en)
        pred = self.linear(lstm_out.view(len(seq), -1))
        return pred[-1]  # we only want the last value
```

In [40]:
```
# Training
```

In [41]:
```
# define train_model function to be used with the two datasets

def train_model(epochs, train_data):

    # instantiate model, define loss and optimization functions

    torch.manual_seed(42)
    model = LSTMnetwork()

    model.to(device) # use .to(device) to select GPU or CPU computation, respe
ctively

    criterion = nn.MSELoss() # use MSE

    # optimizer has to be defined after model has been associated with the dev
ice!
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001) # use Adam opti
mizer


    # start training

    start_time = time.time()
    for epoch in range(epochs):

        # extract the sequence & label from the training data
        for seq, y_train in train_data:

            # reset the parameters and hidden states -- use .to(device) to sel
ect GPU or CPU computation, respectively
            optimizer.zero_grad()
            model.hidden = (torch.zeros(1, 1, model.hidden_size).to(device),
                            torch.zeros(1, 1, model.hidden_size).to(device))
            y_pred = model(seq.to(device))

            loss = criterion(y_pred, y_train.to(device))
            loss.backward()
            optimizer.step()

        # print training result every 10 epochs starting with 1st epoch
        if epoch%10 == 0:
            print(f'Epoch: {epoch+1:2} Loss: {loss.item():10.8f}')

    print(f'\nDuration: {time.time() - start_time:.0f} seconds')
    return model
```

In [42]:
```
# define model_predictions function to be used with both datasets

def model_predictions(model, future, preds, window_size):

    # set the model to evaluation mode
    model.eval()

    for i in range(future):
        seq = torch.FloatTensor(preds[-window_size:])
        with torch.no_grad():
            model.hidden = (torch.zeros(1, 1, model.hidden_size).to(device),
                            torch.zeros(1, 1, model.hidden_size).to(device))
            preds.append(model(seq.to(device)).item())
```

In [43]:
```
import time

epochs = 200

# train model with train_data_1
train_data = train_data_1

model_1 = train_model(epochs, train_data) # provide separate name for model in
case it will be used later on
```

```
Epoch:  1 Loss: 0.02258122
Epoch: 11 Loss: 0.01771832
Epoch: 21 Loss: 0.00304347
Epoch: 31 Loss: 0.00054948
Epoch: 41 Loss: 0.01037953
Epoch: 51 Loss: 0.00080097
Epoch: 61 Loss: 0.00018010
Epoch: 71 Loss: 0.00063973
Epoch: 81 Loss: 0.00016539
Epoch: 91 Loss: 0.00000044
Epoch: 101 Loss: 0.00001165
Epoch: 111 Loss: 0.00004981
Epoch: 121 Loss: 0.00000134
Epoch: 131 Loss: 0.00000192
Epoch: 141 Loss: 0.00000058
Epoch: 151 Loss: 0.00008903
Epoch: 161 Loss: 0.00176829
Epoch: 171 Loss: 0.00005219
Epoch: 181 Loss: 0.00018570
Epoch: 191 Loss: 0.00011390

Duration: 325 seconds
```

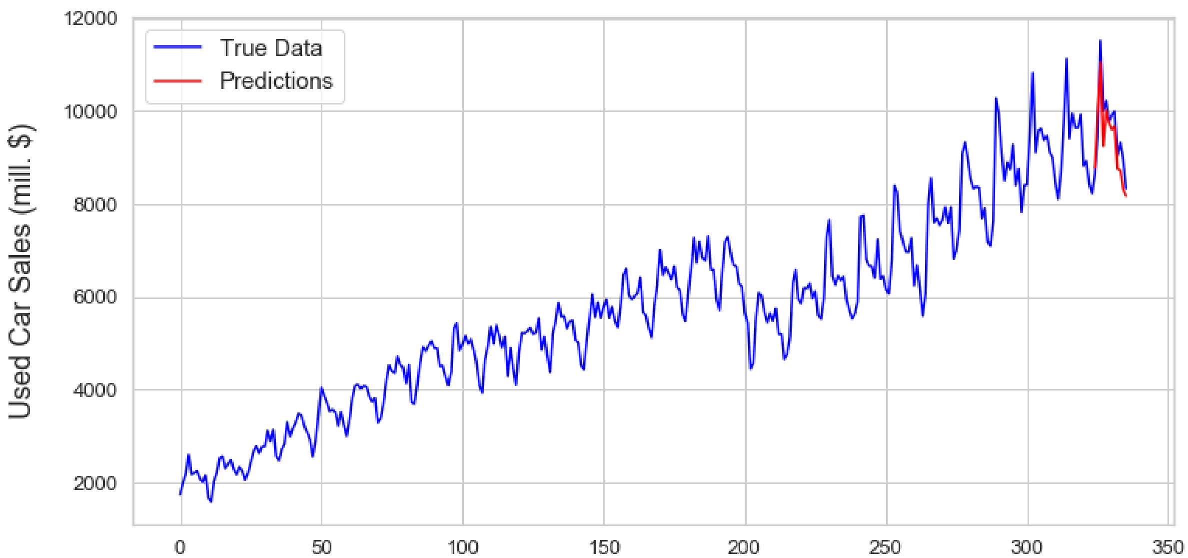
In [44]:
```
# make predictions for train_set_1

future = 12
window_size = window_size_1
preds = train_set_1[-window_size:].tolist()

model_predictions(model_1, future, preds, window_size)
```

In [45]:
```
preds
```

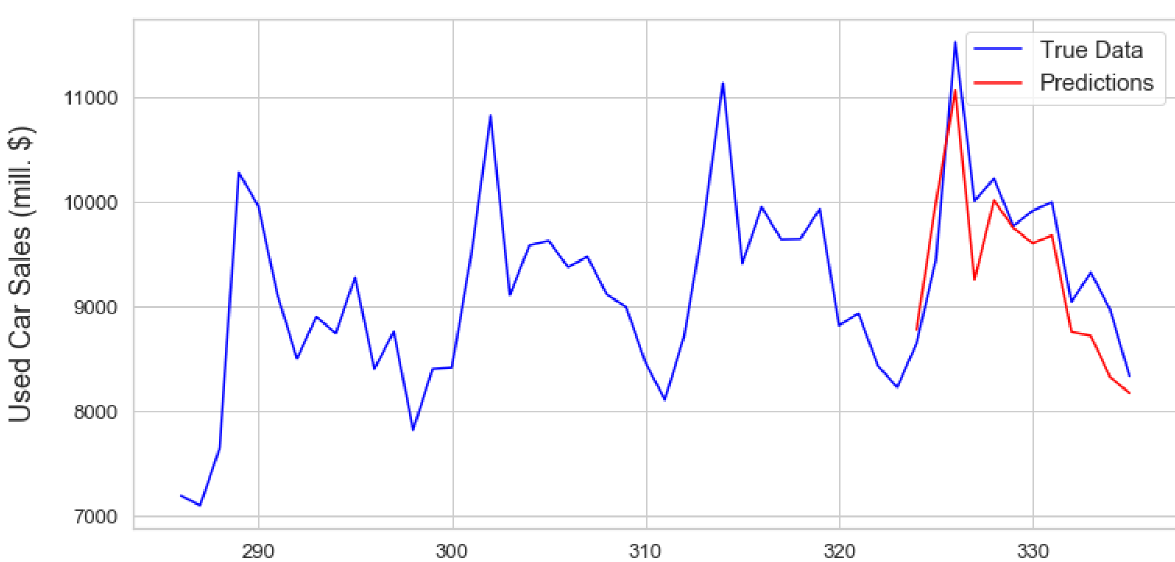
Out[45]:
```
[0.4943973124027252,
 0.7203895449638367,
 1.0,
 0.6391245126724243,
 0.7526442408561707,
 0.6879254579544067,
 0.6887632012367249,
 0.7486647963523865,
 0.5155513882637024,
 0.5398470759391785,
 0.4353335499763489,
 0.39176878333091736,
 0.5067726373672485,
 0.7620344758033752,
 0.9861121773719788,
 0.6068530082702637,
 0.7663792967796326,
 0.7105640172958374,
 0.679915189743042,
 0.6957989931106567,
 0.5026361346244812,
 0.4948640465736389,
 0.4121510982513428,
 0.3802984952926636]
```

In [46]:
```
# invert the normalization for the predicted values to be able to compare to t
est data

preds_1 = scaler_1.inverse_transform(np.array(preds[future:]).reshape(-1, 1))
# use the coresponding scaler
```

In [47]:
```
# plot y_1 and preds_1 to compare predictions to data

plt.figure(figsize = (12,6))
plt.plot(y_1, c = 'blue', label = 'True Data')
plt.plot(np.arange(len(y_1) - future, len(y_1)), preds_1, c = 'red', label =
'Predictions')
plt.ylabel('Used Car Sales (mill. $)', fontsize = 18, labelpad = 15)
plt.legend(fontsize = 15)
plt.show()
```



In [48]:
```
# plot only last portion of graph for more detail view

plt.figure(figsize = (12,6))
plt.plot(np.arange(len(y_1) - 50, len(y_1)), y_1[-50:], c = 'blue', label = 'T
rue Data')
plt.plot(np.arange(len(y_1) - future, len(y_1)), preds_1, c = 'red', label =
'Predictions')
plt.ylabel('Used Car Sales (mill. $)', fontsize = 18, labelpad = 15)
plt.legend(fontsize = 15)
plt.show()
```



In [49]:
```
# model predictions matches well data
# we note, however, that similar result can be obtained simply by appropriate
 averaging and translating the closest data cycles
```

In [50]:
```
# repeat the same process with dataset 2
```

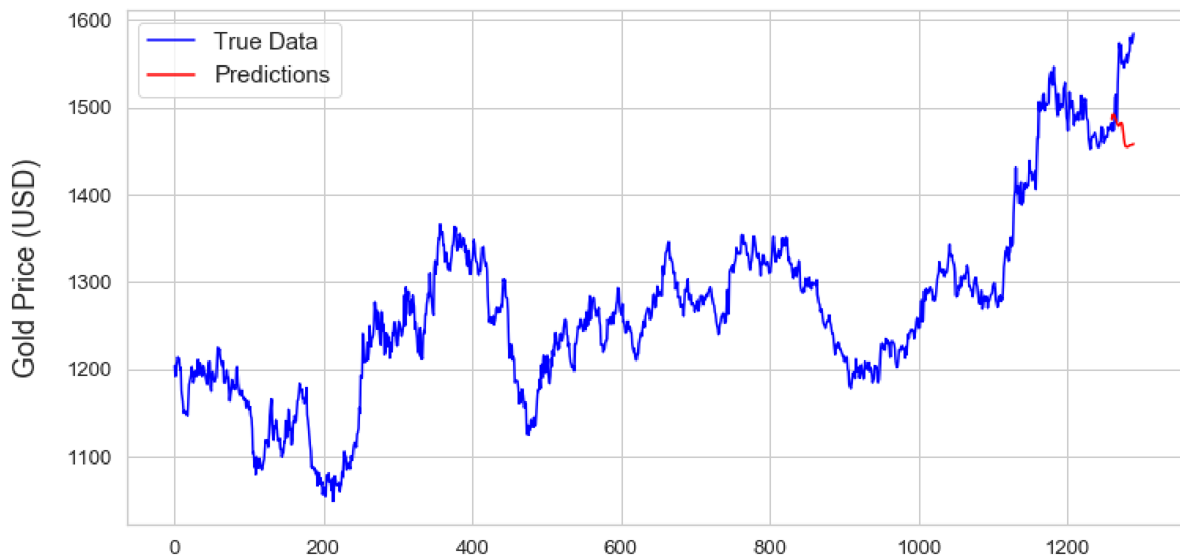
In [51]:
```
# train model with train_data_2
train_data = train_data_2

model_2 = train_model(epochs, train_data)
```

```
Epoch:  1 Loss: 0.00132558
Epoch: 11 Loss: 0.00002865
Epoch: 21 Loss: 0.00001599
Epoch: 31 Loss: 0.00171120
Epoch: 41 Loss: 0.00000009
Epoch: 51 Loss: 0.00002835
Epoch: 61 Loss: 0.00002389
Epoch: 71 Loss: 0.00090776
Epoch: 81 Loss: 0.00011469
Epoch: 91 Loss: 0.00009155
Epoch: 101 Loss: 0.00002225
Epoch: 111 Loss: 0.00004643
Epoch: 121 Loss: 0.00026000
Epoch: 131 Loss: 0.00033497
Epoch: 141 Loss: 0.00008077
Epoch: 151 Loss: 0.00012586
Epoch: 161 Loss: 0.00000982
Epoch: 171 Loss: 0.00009436
Epoch: 181 Loss: 0.00024117
Epoch: 191 Loss: 0.00010930

Duration: 1435 seconds
```

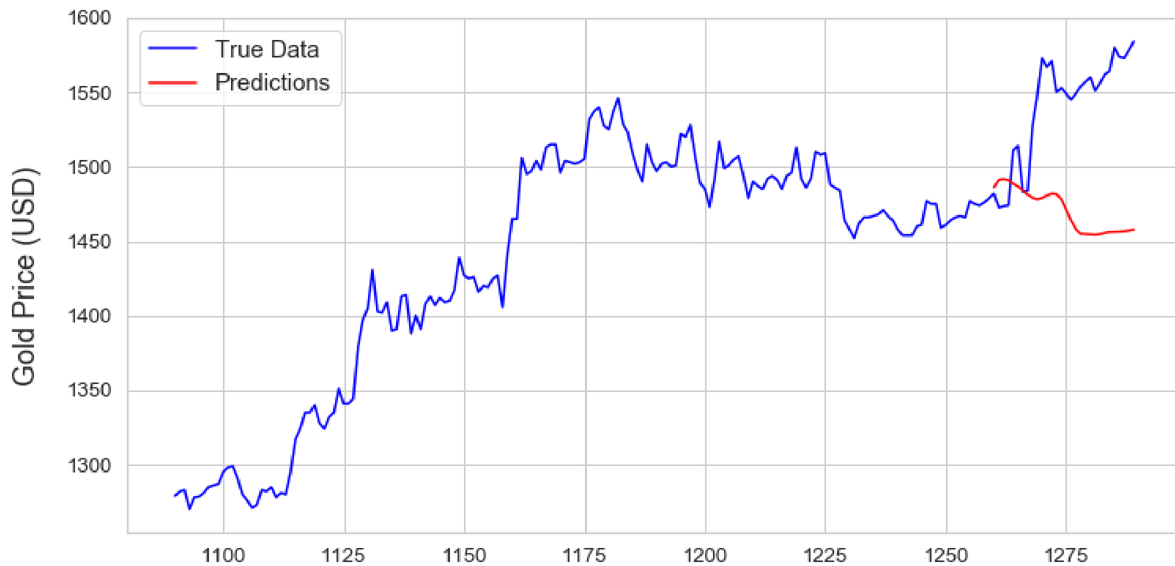
In [52]:
```
# make predictions for train_set_2

future = 30
window_size = window_size_2
preds = train_set_2[-window_size:].tolist()

model_predictions(model_2, future, preds, window_size)
```

In [53]:
```
# invert the normalization for the predicted values to be able to compare to t
est data

preds_2 = scaler_2.inverse_transform(np.array(preds[future:]).reshape(-1, 1))
# use the coresponding scaler
```

In [54]:
```
# plot y_2 and preds_2 to compare predictions to data

plt.figure(figsize = (12,6))
plt.plot(y_2, c = 'blue', label = 'True Data')
plt.plot(np.arange(len(y_2) - future, len(y_2)), preds_2, c = 'red', label =
'Predictions')
plt.ylabel('Gold Price (USD)', fontsize = 18, labelpad = 15)
plt.legend(fontsize = 15)
plt.show()
```



In [55]:
```
# plot only last portion of graph for more detail view

plt.figure(figsize = (12,6))
plt.plot(np.arange(len(y_2) - 200, len(y_2)), y_2[-200:], c = 'blue', label =
'True Data')
plt.plot(np.arange(len(y_2) - future, len(y_2)), preds_2, c = 'red', label =
'Predictions')
plt.ylabel('Gold Price (USD)', fontsize = 18, labelpad = 15)
plt.legend(fontsize = 15)
plt.show()
```



In [56]:
```
# predictions diverge dramatically from data
# extensive research on LSTM models used for predicting "random walk" type of
 data (e.g. stock prices) showed similar results
```

In [57]:
```
# Conclusion:
    # 1) LSTM model provides good predictions for data with well-defined cycli
cal behavior
            # we note that for such data much simpler mathematical operations
 would provide equally good predictions

    # 2) LSTM (and other versions of RNNs) model does not provide good predict
ions for data with random behavior
            # this finding is supported by numerous other studies

    # For data with random behavior different types of analysis are needed for
providing good predictions (if at all possible)
```