```python
# Importing the libraries
import numpy as np
#import matplotlib.pyplot as plt

import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout # GRU, Bidirectional
#from keras.optimizers import SGD
import math
from sklearn.metrics import mean_squared_error
```

```python
data = pd.read_csv('/content/drive/MyDrive/IBM_2006-01-01_to_2018-01-01.csv', index_col='Date', parse_dates=['Date'])
data.head() #view first five columns
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 3020,\n  \"fields\":
[\n    {\n      \"column\": \"Date\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3020,\n
\"samples\": [\n          \"11-08-2011\",\n          \"20-03-2012\",\n
\"23-10-2006\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Open\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 37.554946213280544,\n        \"min\": 72.74,\n
\"max\": 215.38,\n        \"num_unique_values\": 2613,\n
\"samples\": [\n          166.96,\n          134.4,\n          148.4\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"High\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
37.61344637562722,\n        \"min\": 73.94,\n        \"max\": 215.9,\n
\"num_unique_values\": 2603,\n        \"samples\": [\n
103.65,\n          126.39,\n          197.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Low\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 37.47764087967782,\n
\"min\": 69.5,\n        \"max\": 214.3,\n
\"num_unique_values\": 2590,\n        \"samples\": [\n
171.23,\n          162.73,\n          155.07\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Close\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 37.52938665470685,\n
\"min\": 71.74,\n        \"max\": 215.8,\n
\"num_unique_values\": 2659,\n        \"samples\": [\n
119.58,\n          73.7,\n          205.49\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Volume\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 3192830,\n
\"min\": 254256,\n        \"max\": 30774276,\n
\"num_unique_values\": 3020,\n        \"samples\": [\n
9176528,\n          3695026,\n          8862300\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Name\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 1,\n
\"samples\": [\n          \"IBM\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"data"}
```


```python
import pandas as pd

# Example DataFrame initialization (replace this with your actual
DataFrame creation)
data = pd.DataFrame({
    'date': pd.date_range('2016-01-01', '2017-12-31', freq='D'),
    'value': range(731)  # Example values, adjust as per your actual
data
})

# Step 1: Set 'date' column as the index (assuming 'date' is your
datetime column)
data.set_index('date', inplace=True)

# Step 2: Slice using datetime objects
mytrain = data.loc[:pd.to_datetime('2016-12-31'), 'value'].values
mytest = data.loc[pd.to_datetime('2017-01-01'):, 'value'].values

# Now mytrain and mytest will contain the values sliced according to
the datetime range
print(mytrain)
print(mytest)
```

```
[  0   1   2   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]
```

```
# Scaling the training set
sc = MinMaxScaler(feature_range=(0,1))  #MinMaxScaler(feature_range=
(start,stop))
mytrain_scaled = sc.fit_transform(mytrain.reshape(-1, 1))  # Reshape
mytrain to a 2D array

mytrain_scaled #view the scaled values!
#82.55 ==> 0.06065...
```

```
array([[0.        ],
       [0.00273973],
       [0.00547945],
       [0.00821918],
       [0.0109589 ],
       [0.01369863],
       [0.01643836],
       [0.01917808],
       [0.02191781],
       [0.02465753],
       [0.02739726],
       [0.03013699],
       [0.03287671],
       [0.03561644],
       [0.03835616],
       [0.04109589],
       [0.04383562],
       [0.04657534],
       [0.04931507],
       [0.05205479],
       [0.05479452],
       [0.05753425],
       [0.06027397],
       [0.0630137 ],
       [0.06575342],
       [0.06849315],
       [0.07123288],
```

```
       [0.0739726 ],
       [0.07671233],
       [0.07945205],
       [0.08219178],
       [0.08493151],
       [0.08767123],
       [0.09041096],
       [0.09315068],
       [0.09589041],
       [0.09863014],
       [0.10136986],
       [0.10410959],
       [0.10684932],
       [0.10958904],
       [0.11232877],
       [0.11506849],
       [0.11780822],
       [0.12054795],
       [0.12328767],
       [0.1260274 ],
       [0.12876712],
       [0.13150685],
       [0.13424658],
       [0.1369863 ],
       [0.13972603],
       [0.14246575],
       [0.14520548],
       [0.14794521],
       [0.15068493],
       [0.15342466],
       [0.15616438],
       [0.15890411],
       [0.16164384],
       [0.16438356],
       [0.16712329],
       [0.16986301],
       [0.17260274],
       [0.17534247],
       [0.17808219],
       [0.18082192],
       [0.18356164],
       [0.18630137],
       [0.1890411 ],
       [0.19178082],
       [0.19452055],
       [0.19726027],
       [0.2       ],
       [0.20273973],
       [0.20547945],
```

```
       [0.20821918],
       [0.2109589 ],
       [0.21369863],
       [0.21643836],
       [0.21917808],
       [0.22191781],
       [0.22465753],
       [0.22739726],
       [0.23013699],
       [0.23287671],
       [0.23561644],
       [0.23835616],
       [0.24109589],
       [0.24383562],
       [0.24657534],
       [0.24931507],
       [0.25205479],
       [0.25479452],
       [0.25753425],
       [0.26027397],
       [0.2630137 ],
       [0.26575342],
       [0.26849315],
       [0.27123288],
       [0.2739726 ],
       [0.27671233],
       [0.27945205],
       [0.28219178],
       [0.28493151],
       [0.28767123],
       [0.29041096],
       [0.29315068],
       [0.29589041],
       [0.29863014],
       [0.30136986],
       [0.30410959],
       [0.30684932],
       [0.30958904],
       [0.31232877],
       [0.31506849],
       [0.31780822],
       [0.32054795],
       [0.32328767],
       [0.3260274 ],
       [0.32876712],
       [0.33150685],
       [0.33424658],
       [0.3369863 ],
       [0.33972603],
```

```
       [0.34246575],
       [0.34520548],
       [0.34794521],
       [0.35068493],
       [0.35342466],
       [0.35616438],
       [0.35890411],
       [0.36164384],
       [0.36438356],
       [0.36712329],
       [0.36986301],
       [0.37260274],
       [0.37534247],
       [0.37808219],
       [0.38082192],
       [0.38356164],
       [0.38630137],
       [0.3890411 ],
       [0.39178082],
       [0.39452055],
       [0.39726027],
       [0.4       ],
       [0.40273973],
       [0.40547945],
       [0.40821918],
       [0.4109589 ],
       [0.41369863],
       [0.41643836],
       [0.41917808],
       [0.42191781],
       [0.42465753],
       [0.42739726],
       [0.43013699],
       [0.43287671],
       [0.43561644],
       [0.43835616],
       [0.44109589],
       [0.44383562],
       [0.44657534],
       [0.44931507],
       [0.45205479],
       [0.45479452],
       [0.45753425],
       [0.46027397],
       [0.4630137 ],
       [0.46575342],
       [0.46849315],
       [0.47123288],
       [0.4739726 ],
```

```
       [0.47671233],
       [0.47945205],
       [0.48219178],
       [0.48493151],
       [0.48767123],
       [0.49041096],
       [0.49315068],
       [0.49589041],
       [0.49863014],
       [0.50136986],
       [0.50410959],
       [0.50684932],
       [0.50958904],
       [0.51232877],
       [0.51506849],
       [0.51780822],
       [0.52054795],
       [0.52328767],
       [0.5260274 ],
       [0.52876712],
       [0.53150685],
       [0.53424658],
       [0.5369863 ],
       [0.53972603],
       [0.54246575],
       [0.54520548],
       [0.54794521],
       [0.55068493],
       [0.55342466],
       [0.55616438],
       [0.55890411],
       [0.56164384],
       [0.56438356],
       [0.56712329],
       [0.56986301],
       [0.57260274],
       [0.57534247],
       [0.57808219],
       [0.58082192],
       [0.58356164],
       [0.58630137],
       [0.5890411 ],
       [0.59178082],
       [0.59452055],
       [0.59726027],
       [0.6       ],
       [0.60273973],
       [0.60547945],
       [0.60821918],
```

```
       [0.6109589 ],
       [0.61369863],
       [0.61643836],
       [0.61917808],
       [0.62191781],
       [0.62465753],
       [0.62739726],
       [0.63013699],
       [0.63287671],
       [0.63561644],
       [0.63835616],
       [0.64109589],
       [0.64383562],
       [0.64657534],
       [0.64931507],
       [0.65205479],
       [0.65479452],
       [0.65753425],
       [0.66027397],
       [0.6630137 ],
       [0.66575342],
       [0.66849315],
       [0.67123288],
       [0.6739726 ],
       [0.67671233],
       [0.67945205],
       [0.68219178],
       [0.68493151],
       [0.68767123],
       [0.69041096],
       [0.69315068],
       [0.69589041],
       [0.69863014],
       [0.70136986],
       [0.70410959],
       [0.70684932],
       [0.70958904],
       [0.71232877],
       [0.71506849],
       [0.71780822],
       [0.72054795],
       [0.72328767],
       [0.7260274 ],
       [0.72876712],
       [0.73150685],
       [0.73424658],
       [0.7369863 ],
       [0.73972603],
       [0.74246575],
```

```
       [0.74520548],
       [0.74794521],
       [0.75068493],
       [0.75342466],
       [0.75616438],
       [0.75890411],
       [0.76164384],
       [0.76438356],
       [0.76712329],
       [0.76986301],
       [0.77260274],
       [0.77534247],
       [0.77808219],
       [0.78082192],
       [0.78356164],
       [0.78630137],
       [0.7890411 ],
       [0.79178082],
       [0.79452055],
       [0.79726027],
       [0.8       ],
       [0.80273973],
       [0.80547945],
       [0.80821918],
       [0.8109589 ],
       [0.81369863],
       [0.81643836],
       [0.81917808],
       [0.82191781],
       [0.82465753],
       [0.82739726],
       [0.83013699],
       [0.83287671],
       [0.83561644],
       [0.83835616],
       [0.84109589],
       [0.84383562],
       [0.84657534],
       [0.84931507],
       [0.85205479],
       [0.85479452],
       [0.85753425],
       [0.86027397],
       [0.8630137 ],
       [0.86575342],
       [0.86849315],
       [0.87123288],
       [0.8739726 ],
       [0.87671233],
```

```
        [0.87945205],
        [0.88219178],
        [0.88493151],
        [0.88767123],
        [0.89041096],
        [0.89315068],
        [0.89589041],
        [0.89863014],
        [0.90136986],
        [0.90410959],
        [0.90684932],
        [0.90958904],
        [0.91232877],
        [0.91506849],
        [0.91780822],
        [0.92054795],
        [0.92328767],
        [0.9260274 ],
        [0.92876712],
        [0.93150685],
        [0.93424658],
        [0.9369863 ],
        [0.93972603],
        [0.94246575],
        [0.94520548],
        [0.94794521],
        [0.95068493],
        [0.95342466],
        [0.95616438],
        [0.95890411],
        [0.96164384],
        [0.96438356],
        [0.96712329],
        [0.96986301],
        [0.97260274],
        [0.97534247],
        [0.97808219],
        [0.98082192],
        [0.98356164],
        [0.98630137],
        [0.9890411 ],
        [0.99178082],
        [0.99452055],
        [0.99726027],
        [1.        ]])
```

```
len(mytrain_scaled)
```

```
366
```

```
I_train = []
O_train = []
for i in range(60, len(mytrain_scaled)): # Change the upper bound to
len(mytrain_scaled)
    I_train.append(mytrain_scaled[i-60:i,0]) #every sequence will have
60 rows/values as input
    O_train.append(mytrain_scaled[i,0])
```

```
I_train[0]
```

```
array([0.        , 0.00273973, 0.00547945, 0.00821918, 0.0109589 ,
       0.01369863, 0.01643836, 0.01917808, 0.02191781, 0.02465753,
       0.02739726, 0.03013699, 0.03287671, 0.03561644, 0.03835616,
       0.04109589, 0.04383562, 0.04657534, 0.04931507, 0.05205479,
       0.05479452, 0.05753425, 0.06027397, 0.0630137 , 0.06575342,
       0.06849315, 0.07123288, 0.0739726 , 0.07671233, 0.07945205,
       0.08219178, 0.08493151, 0.08767123, 0.09041096, 0.09315068,
       0.09589041, 0.09863014, 0.10136986, 0.10410959, 0.10684932,
       0.10958904, 0.11232877, 0.11506849, 0.11780822, 0.12054795,
       0.12328767, 0.1260274 , 0.12876712, 0.13150685, 0.13424658,
       0.1369863 , 0.13972603, 0.14246575, 0.14520548, 0.14794521,
       0.15068493, 0.15342466, 0.15616438, 0.15890411, 0.16164384])
```

```
O_train[0]
```

```
0.1643835616438356
```

```
I_train = np.array(I_train)   #converting into arrays!
O_train = np.array(O_train)
```

```
I_train = np.array(I_train)   #converting into arrays!
```

```
# Calculate the appropriate dimensions based on the array's size
num_sequences = I_train.size // 60  # Assuming 60 inputs per sequence
I_train = I_train.reshape(num_sequences, 60, 1)
```

```
print(I_train.shape)  # Verify the new shape
```

```
(306, 60, 1)
```

```
I_train.shape
```

```
(306, 60, 1)
```

```
#building the model

model = Sequential()
# First LSTM layer
model.add(LSTM(units=50, return_sequences=True,
input_shape=(I_train.shape[1],1)))
model.add(Dropout(0.2))
# Second LSTM layer
```

```
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
# Third LSTM layer
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
# Fourth LSTM layer
model.add(LSTM(units=50))
model.add(Dropout(0.2))
# The output layer
model.add(Dense(units=1))

#compile
model.compile(optimizer='rmsprop',loss='mean_squared_error')

model.fit(I_train,O_train,epochs=50,batch_size=32)
```

```
Epoch 1/50
10/10 [==============================] - 13s 105ms/step - loss: 0.0827
Epoch 2/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0088
Epoch 3/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0174
Epoch 4/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0165
Epoch 5/50
10/10 [==============================] - 1s 103ms/step - loss: 0.0141
Epoch 6/50
10/10 [==============================] - 1s 103ms/step - loss: 0.0080
Epoch 7/50
10/10 [==============================] - 1s 100ms/step - loss: 0.0142
Epoch 8/50
10/10 [==============================] - 1s 99ms/step - loss: 0.0102
Epoch 9/50
10/10 [==============================] - 1s 149ms/step - loss: 0.0087
Epoch 10/50
10/10 [==============================] - 2s 172ms/step - loss: 0.0091
Epoch 11/50
10/10 [==============================] - 2s 178ms/step - loss: 0.0137
Epoch 12/50
10/10 [==============================] - 1s 104ms/step - loss: 0.0086
Epoch 13/50
10/10 [==============================] - 1s 118ms/step - loss: 0.0110
Epoch 14/50
10/10 [==============================] - 2s 159ms/step - loss: 0.0098
Epoch 15/50
10/10 [==============================] - 2s 166ms/step - loss: 0.0079
Epoch 16/50
10/10 [==============================] - 1s 143ms/step - loss: 0.0114
Epoch 17/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0073
```

```
Epoch 18/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0070
Epoch 19/50
10/10 [==============================] - 1s 105ms/step - loss: 0.0114
Epoch 20/50
10/10 [==============================] - 1s 154ms/step - loss: 0.0071
Epoch 21/50
10/10 [==============================] - 2s 165ms/step - loss: 0.0065
Epoch 22/50
10/10 [==============================] - 2s 167ms/step - loss: 0.0066
Epoch 23/50
10/10 [==============================] - 1s 104ms/step - loss: 0.0081
Epoch 24/50
10/10 [==============================] - 1s 104ms/step - loss: 0.0043
Epoch 25/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0070
Epoch 26/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0072
Epoch 27/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0072
Epoch 28/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0071
Epoch 29/50
10/10 [==============================] - 1s 103ms/step - loss: 0.0037
Epoch 30/50
10/10 [==============================] - 1s 100ms/step - loss: 0.0095
Epoch 31/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0040
Epoch 32/50
10/10 [==============================] - 1s 117ms/step - loss: 0.0072
Epoch 33/50
10/10 [==============================] - 2s 161ms/step - loss: 0.0071
Epoch 34/50
10/10 [==============================] - 2s 168ms/step - loss: 0.0048
Epoch 35/50
10/10 [==============================] - 1s 130ms/step - loss: 0.0077
Epoch 36/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0045
Epoch 37/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0070
Epoch 38/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0034
Epoch 39/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0062
Epoch 40/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0051
Epoch 41/50
10/10 [==============================] - 1s 99ms/step - loss: 0.0044
Epoch 42/50
```

```
10/10 [==============================] - 1s 102ms/step - loss: 0.0047
Epoch 43/50
10/10 [==============================] - 1s 102ms/step - loss: 0.0032
Epoch 44/50
10/10 [==============================] - 1s 104ms/step - loss: 0.0053
Epoch 45/50
10/10 [==============================] - 1s 146ms/step - loss: 0.0050
Epoch 46/50
10/10 [==============================] - 2s 162ms/step - loss: 0.0049
Epoch 47/50
10/10 [==============================] - 2s 168ms/step - loss: 0.0058
Epoch 48/50
10/10 [==============================] - 1s 103ms/step - loss: 0.0029
Epoch 49/50
10/10 [==============================] - 1s 101ms/step - loss: 0.0069
Epoch 50/50
10/10 [==============================] - 1s 99ms/step - loss: 0.0036

<keras.src.callbacks.History at 0x7bf506718c10>
```

```
model.save("StockIBM.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/
training.py:3103: UserWarning: You are saving your model as an HDF5
file via `model.save()`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

```
#Test
from tensorflow.keras.models import load_model
import numpy as np
#load the model
model = load_model("/content/StockIBM.h5")
#60 values
user_input = [83.58746017, 80.34122742, 80.81285462, 82.00291895,
83.29159961, 81.44688209, 80.76357678, 83.80165525, 82.51120409,
82.48569342, 80.65383659, 81.69740582, 82.78338862, 83.37704145,
83.53829805, 80.20374715, 80.42476455, 82.69989097, 83.73417143,
82.95666579, 83.94381372, 82.37856671, 83.60787851, 81.80850007,
82.61239819, 82.06021048, 81.0673498, 83.7835985, 81.74964384,
83.10783276, 80.54146548, 80.49215929, 82.03237911, 81.79802128,
80.93914228, 83.90444224, 83.2143073, 81.31844381, 83.23807452,
83.4603476, 81.85039395, 81.3031393, 80.6351211, 83.65161771,
80.54893536, 82.22940581, 82.50289209, 80.59368723, 83.24620825,
81.77694404, 83.20275117, 83.43996357, 83.38335394, 82.69282925,
80.80822433, 80.30044039, 83.6842126, 82.82960781, 80.12425038,
82.25607533]

#2D array
```

```python
user_input_array = np.array(user_input).reshape(60, 1)
#down-scale
sc = MinMaxScaler(feature_range=(0,1))
user_scaled = sc.fit_transform(user_input_array)
#3D array
user_scaled = user_scaled.reshape(1, 60, 1)
#predict
pred = model.predict(user_scaled)
print(pred) #down scale output
#up-scale
pred_original = sc.inverse_transform(pred)
print("The stock price is",pred_original[0][0]) #up scaled/original
o/p
```

```
1/1 [==============================] - 2s 2s/step
[[0.54982406]]
The stock price is 82.22434
```