Tutorial2-RNN Time series-ALL DATA Standardization

April 9, 2024

1 Tutorial 2 - RNN Time Series

In this notebook, we will predict the weather temperature.

[ ]:
```
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import mean_squared_error

epoch_num = 5 # number of epochs to use for training our models.

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

2 Read the Dataset

[ ]:
```
import pandas as pd

weather = pd.read_csv('https://raw.githubusercontent.com/prof-tcsmith/data/
 ↪master/weather.csv')

weather.head()
```

[ ]:
```
        date  hour       NO2        CO        O3        NO  PM2.5      PM10  \
0  4/25/2021     0  0.039817  0.080700 -0.000867  0.009800    0.2  5.040000
1  4/25/2021     1  0.035900  0.092217 -0.000267  0.009833    0.2  6.293333
```

```
2  4/25/2021     2  0.028083  0.062750  0.002517  0.012883    0.2  5.501667
3  4/25/2021     3  0.025633  0.042300  0.004550  0.014233    0.2  4.201667
4  4/25/2021     4  0.023717  0.036883  0.006267  0.015417    0.2  5.365000

   Air Temp.  Air Hum.    Air Pres.
0  25.795000      99.9  1011.980000
1  25.445000      99.9  1012.131667
2  25.223333      99.9  1012.365000
3  25.075000      99.9  1012.276667
4  24.928333      99.9  1012.030000
```

[ ]:
```python
# Convert the temp to Fahrenheit:

weather['Air Temp F'] = weather['Air Temp.']*1.8 + 32
```

[ ]:
```python
weather
```

[ ]:
```
           date  hour       NO2        CO        O3        NO  PM2.5  \
0     4/25/2021     0  0.039817  0.080700 -0.000867  0.009800  0.200
1     4/25/2021     1  0.035900  0.092217 -0.000267  0.009833  0.200
2     4/25/2021     2  0.028083  0.062750  0.002517  0.012883  0.200
3     4/25/2021     3  0.025633  0.042300  0.004550  0.014233  0.200
4     4/25/2021     4  0.023717  0.036883  0.006267  0.015417  0.200
…           …   …       …       …        …       …     …
7952  4/19/2022    19  0.007500  0.135750  0.035250  0.044000  2.150
7953  4/19/2022    20  0.024000  0.145750  0.025250  0.035000  2.525
7954  4/19/2022    21  0.013400  0.147200  0.038800  0.022200  2.260
7955  4/19/2022    22  0.023000  0.126000  0.039000  0.014000  2.350
7956  4/19/2022    23  0.032500  0.162250  0.041750  0.030500  1.975

           PM10  Air Temp.  Air Hum.    Air Pres.  Air Temp F
0      5.040000  25.795000    99.900  1011.980000      78.431
1      6.293333  25.445000    99.900  1012.131667      77.801
2      5.501667  25.223333    99.900  1012.365000      77.402
3      4.201667  25.075000    99.900  1012.276667      77.135
4      5.365000  24.928333    99.900  1012.030000      76.871
…           …         …        …          …          …
7952   9.275000  28.375000    39.175  1016.000000      83.075
7953  16.100000  28.575000    37.275  1015.550000      83.435
7954  12.980000  28.440000    32.740  1015.340000      83.192
7955  11.375000  28.150000    33.250  1015.600000      82.670
7956  10.675000  25.700000    41.725  1015.725000      78.260

[7957 rows x 12 columns]
```

[ ]:
```python
#Drop the columns we don't need
```

```
weather = weather.drop(['NO2', 'CO', 'O3', 'NO', 'PM2.5', 'PM10', 'Air Temp.',
            'Air Hum.', 'Air Pres.'], axis=1)
```

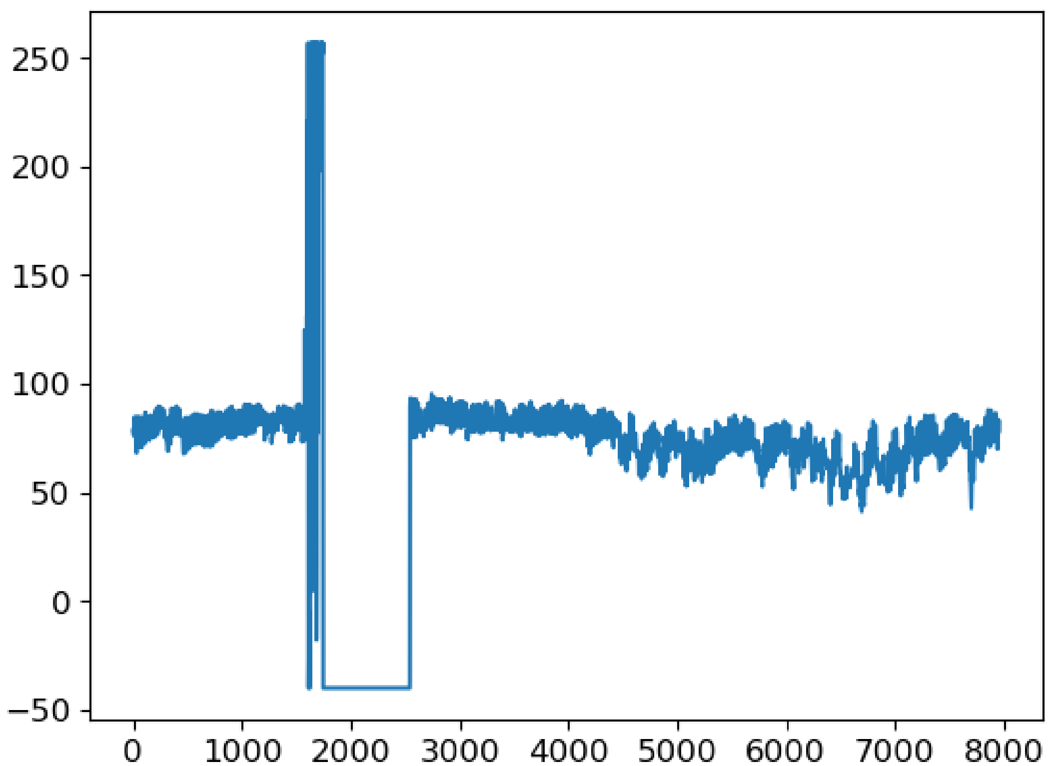
[ ]:
```
weather
```

[ ]:
```
          date  hour  Air Temp F
0     4/25/2021     0      78.431
1     4/25/2021     1      77.801
2     4/25/2021     2      77.402
3     4/25/2021     3      77.135
4     4/25/2021     4      76.871
…          …    …          …
7952  4/19/2022    19      83.075
7953  4/19/2022    20      83.435
7954  4/19/2022    21      83.192
7955  4/19/2022    22      82.670
7956  4/19/2022    23      78.260

[7957 rows x 3 columns]
```

[ ]:
```
#Plot temp

plt.plot(weather['Air Temp F'])
plt.show()
```



3 Data Cleanup

[ ]:
```
# Values higher than 100 degrees are probably incorrect readings

weather[weather['Air Temp F']>100]
```

[ ]:
```
           date  hour  Air Temp F
1578  6/29/2021    21     124.865
1582  6/30/2021     1     108.674
1595  6/30/2021    14     112.001
1596  6/30/2021    15     131.294
1597  6/30/2021    16     128.849
…             …     …           …
1742   7/6/2021    17     256.820
1743   7/6/2021    18     256.820
1744   7/6/2021    19     251.894
1745   7/6/2021    20     256.820
1746   7/6/2021    21     246.926

[114 rows x 3 columns]
```

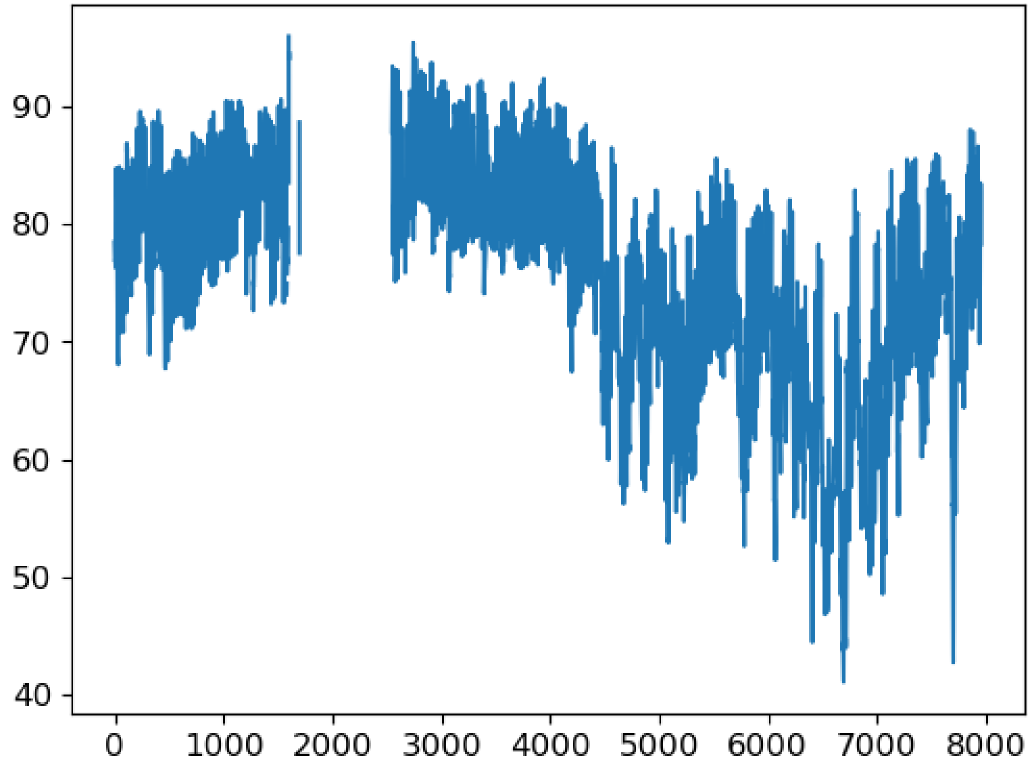
[ ]:
```
# Convert all values higher than 100 degrees to null values

weather['Air Temp F'] = np.where(weather['Air Temp F']>100, np.nan,␣
 ↪weather['Air Temp F'])
```

[ ]:
```
# Values lower than 30 degrees are probably incorrect readings. Convert them to␣
 ↪null

weather['Air Temp F'] = np.where(weather['Air Temp F']<30, np.nan, weather['Air␣
 ↪Temp F'])
```

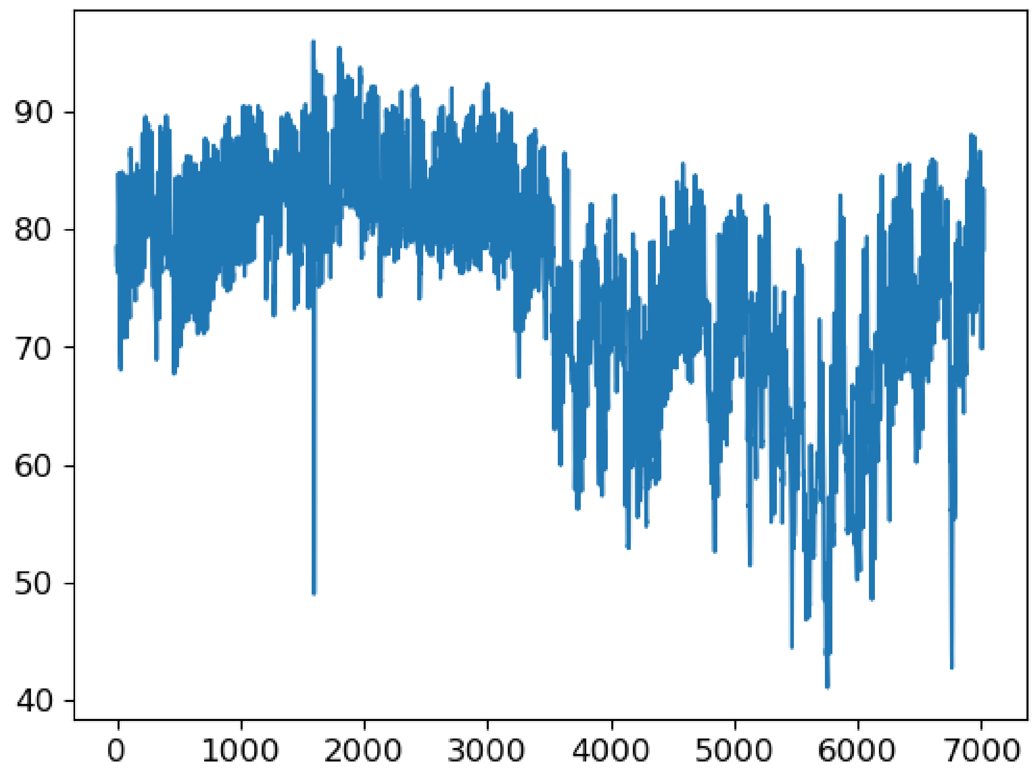
[ ]:
```
plt.plot(weather['Air Temp F'])
plt.show()
```



[ ]:
```
# Remove all null values

weather = weather.dropna().reset_index(drop=True)
```

[ ]:
```
plt.plot(weather['Air Temp F'])
plt.show()
```



3.1 RESHAPE the data set!

[ ]:
```
weather.shape
```

[ ]: (7017, 3)

[ ]:
```
# Note that not all days have 24 readings. Some are missing.

weather.shape[0]/24
```

[ ]: 292.375

[ ]:
```
weather.groupby(['date']).count()
```

[ ]:

	hour	Air Temp F
date		
1/1/2022	24	24
1/10/2022	24	24
1/11/2022	24	24
1/12/2022	24	24
1/13/2022	24	24
…	…	…

```
9/5/2021     24          24
9/6/2021     24          24
9/7/2021     24          24
9/8/2021     24          24
9/9/2021     24          24

[299 rows x 2 columns]
```

[ ]:
```python
# Find the reading count for each day

hour_count = pd.DataFrame(weather.groupby(['date']).count()['hour'])

hour_count
```

[ ]:
```
            hour
date
1/1/2022     24
1/10/2022    24
1/11/2022    24
1/12/2022    24
1/13/2022    24
…           …
9/5/2021     24
9/6/2021     24
9/7/2021     24
9/8/2021     24
9/9/2021     24

[299 rows x 1 columns]
```

[ ]:
```python
# Find the reading counts that are less than 24

hour_count[hour_count['hour']<24]
```

[ ]:
```
            hour
date
3/18/2022     7
4/11/2022     9
4/14/2022    10
6/21/2021    21
6/29/2021    23
6/30/2021    16
7/1/2021      1
7/2/2021      3
7/3/2021      3
7/4/2021      3
8/10/2021     9
```

[ ]:
```
# Identify the dates of these records

hour_count[hour_count['hour']<24].index.values
```

[ ]:
```
array(['3/18/2022', '4/11/2022', '4/14/2022', '6/21/2021', '6/29/2021',
       '6/30/2021', '7/1/2021', '7/2/2021', '7/3/2021', '7/4/2021',
       '8/10/2021'], dtype=object)
```

[ ]:
```
# Find the corresponding index values in the original data set

indexes = weather[weather['date'].isin(hour_count[hour_count['hour']<24].index.
 ↪values)]

indexes
```

[ ]:
```
           date  hour  Air Temp F
1368  6/21/2021     0      84.386
1369  6/21/2021     1      84.137
1370  6/21/2021     2      83.993
1371  6/21/2021     3      83.549
1372  6/21/2021     4      83.210
…           …     …           …
6892  4/14/2022    19      83.732
6893  4/14/2022    20      82.715
6894  4/14/2022    21      81.590
6895  4/14/2022    22      82.085
6896  4/14/2022    23      78.620

[105 rows x 3 columns]
```

[ ]:
```
# Remove these rows from the data set.

weather = weather.drop(indexes.index, axis=0).reset_index(drop=True)

weather.shape
```

[ ]:
```
(6912, 3)
```

[ ]:
```
weather.head()
```

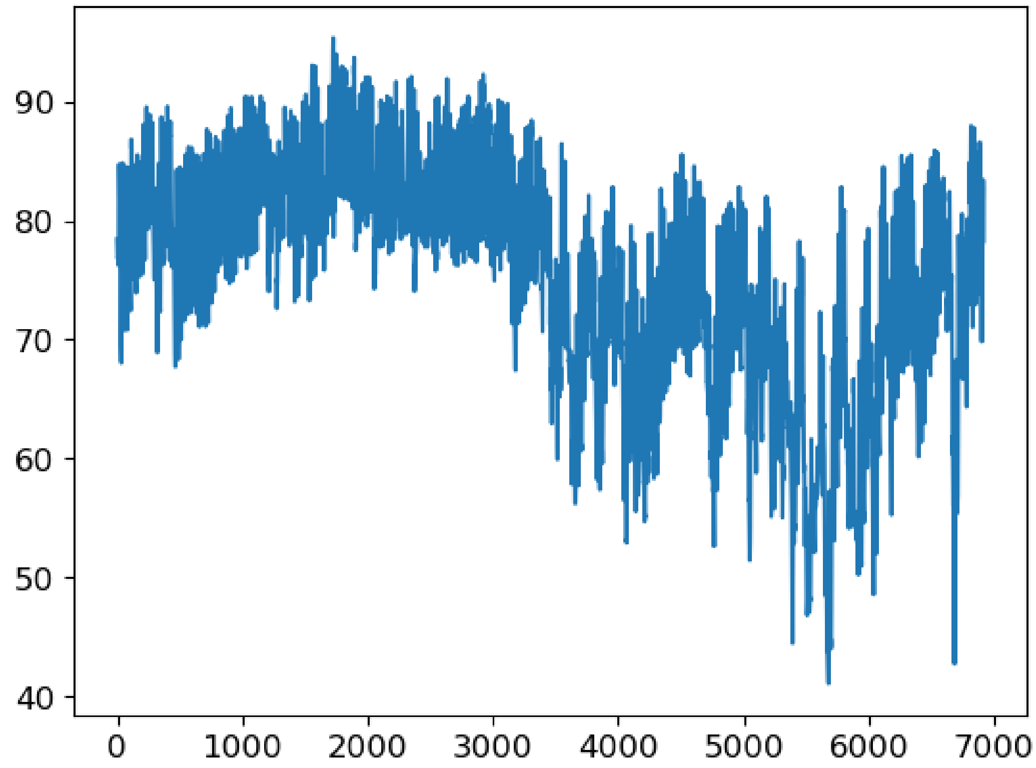
[ ]:
```
        date  hour  Air Temp F
0  4/25/2021     0      78.431
1  4/25/2021     1      77.801
2  4/25/2021     2      77.402
3  4/25/2021     3      77.135
4  4/25/2021     4      76.871
```

```
[ ]: # All remaining days have 24 readings (for 24 hours)
     # There are a total of 288 days

     weather.shape[0]/24
```

```
[ ]: 288.0
```

```
[ ]: plt.plot(weather['Air Temp F'])
     plt.show()
```




```
[ ]: # Re-organize the data set by day and hours

     temp = np.array(weather['Air Temp F']).reshape(288,24)

     temp
```

```
[ ]: array([[78.431     , 77.801     , 77.40199999, …, 82.54700001,
             81.716     , 79.196     ],
            [76.02200001, 73.121     , 71.68699999, …, 84.82699999,
             84.57499999, 82.52900001],
            [80.843     , 78.87500001, 77.05099999, …, 84.52700001,
             83.99899999, 82.44199999],
```

```
…,
[78.26      , 77.54      , 77.495     , …, 83.98999999,
 83.3      , 79.736     ],
[78.305     , 77.63      , 77.432     , …, 84.245     ,
 83.084     , 79.376     ],
[77.27      , 76.136     , 75.29      , …, 83.192     ,
 82.67      , 78.26      ]])
```

[ ]:
```python
# Convert to dataframe

temp_df = pd.DataFrame(temp, columns=np.arange(0,24,1))

temp_df
```

[ ]:
```
          0       1       2       3       4       5       6       7       8  \
0    78.431  77.801  77.402  77.135  76.871  76.814  76.892  76.925  76.580
1    76.022  73.121  71.687  70.664  69.560  68.864  68.603  68.360  68.360
2    80.843  78.875  77.051  74.675  73.499  72.950  72.221  71.330  71.048
3    80.576  78.731  76.739  74.819  73.829  73.052  72.575  71.876  71.306
4    78.632  77.618  76.040  75.278  74.918  74.561  73.859  73.127  72.785
..      …       …       …       …       …       …       …       …       …
283  76.910  76.145  75.590  75.380  75.245  74.525  74.750  74.300  74.030
284  75.200  77.585  77.060  76.055  74.948  74.030  73.400  72.500  71.915
285  78.260  77.540  77.495  77.540  77.540  76.730  75.920  75.245  74.525
286  78.305  77.630  77.432  77.135  77.360  76.640  76.505  75.980  75.065
287  77.270  76.136  75.290  74.948  74.570  74.210  74.030  74.300  74.060

          9  …      14      15      16      17      18      19      20  \
0    76.343  …  81.584  81.575  82.445  84.731  84.272  83.252  83.447
1    68.267  …  75.500  77.594  79.691  81.458  83.012  84.080  84.323
2    70.766  …  77.222  79.241  80.933  81.602  82.478  83.795  84.146
3    70.793  …  77.657  79.562  81.014  81.848  82.955  83.813  83.996
4    72.575  …  79.859  82.376  85.808  86.831  86.108  86.459  86.003
..      …  …       …       …       …       …       …       …       …
283  73.040  …  83.300  84.920  81.428  83.210  83.750  85.400  86.225
284  71.600  …  77.108  79.916  83.012  83.930  83.948  86.000  87.215
285  73.400  …  80.330  82.625  84.695  85.640  85.460  84.992  84.470
286  74.120  …  80.810  83.444  84.155  83.435  80.510  86.585  85.235
287  72.230  …  73.976  74.930  78.485  79.916  81.635  83.075  83.435

          21      22      23
0    82.547  81.716  79.196
1    84.827  84.575  82.529
2    84.527  83.999  82.442
3    84.437  82.913  80.210
4    84.902  84.413  81.434
..      …       …       …
```

```
283  87.980  86.540  76.784
284  87.800  86.540  80.168
285  83.990  83.300  79.736
286  84.245  83.084  79.376
287  83.192  82.670  78.260

[288 rows x 24 columns]
```

4 Reshape for Standardizing Data

```
[ ]: # Let's create a single sequence (i.e., feature) for standardization

     temp_1feature = np.array(temp_df).ravel().reshape(-1,1)

     temp_1feature.shape
```

```
[ ]: (6912, 1)
```

```
[ ]: temp_1feature
```

```
[ ]: array([[78.431     ],
            [77.801     ],
            [77.40199999],
            …,
            [83.192     ],
            [82.67      ],
            [78.26      ]])
```

4.1 Standardize the values

```
[ ]: # Next, standardize

     from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()

     temp_std = scaler.fit_transform(temp_1feature)
```

4.2 Reshape the data back to 24-hour format

```
[ ]: temp_reshaped = temp_std.reshape(288,24)

     temp_reshaped.shape
```

```
[ ]: (288, 24)
```

[ ]:

```python
#Pandas version of the reshaped data

pd.DataFrame(temp_reshaped, columns=np.arange(0,24,1))
```

[ ]:

```
           0         1         2         3         4         5         6   \
0    0.391232  0.325683  0.284169  0.256388  0.228920  0.222990  0.231105
1    0.140586 -0.161251 -0.310453 -0.416891 -0.531758 -0.604174 -0.631330
2    0.642190  0.437428  0.247649  0.000436 -0.121922 -0.179043 -0.254892
3    0.614410  0.422445  0.215186  0.015419 -0.087587 -0.168430 -0.218060
4    0.412145  0.306642  0.142458  0.063175  0.025719 -0.011425 -0.084465
..        …         …         …         …         …         …         …
283  0.232978  0.153383  0.095638  0.073788  0.059742 -0.015171  0.008239
284  0.055060  0.303209  0.248585  0.144019  0.028840 -0.066674 -0.132222
285  0.373440  0.298527  0.293845  0.298527  0.298527  0.214250  0.129973
286  0.378122  0.307891  0.287290  0.256388  0.279799  0.204886  0.190840
287  0.270435  0.152447  0.064424  0.028840 -0.010489 -0.047945 -0.066674

           7         8         9   …        14        15        16  \
0    0.234539  0.198643  0.173984  …  0.719288  0.718351  0.808871
1   -0.656613 -0.656613 -0.666289  …  0.086274  0.304145  0.522329
2   -0.347597 -0.376938 -0.406279  …  0.265440  0.475509  0.651554
3   -0.290788 -0.350094 -0.403470  …  0.310700  0.508907  0.659982
4   -0.160627 -0.196210 -0.218060  …  0.539809  0.801692  1.158777
..        …         …         …  …       …         …         …
283 -0.038581 -0.066674 -0.169679  …  0.897830  1.066384  0.703057
284 -0.225864 -0.286730 -0.319505  …  0.253579  0.545740  0.867865
285  0.059742 -0.015171 -0.132222  …  0.588814  0.827599  1.042974
286  0.136216  0.041014 -0.057309  …  0.638756  0.912813  0.986789
287 -0.038581 -0.063552 -0.253956  … -0.072292  0.026968  0.396850

           17        18        19        20        21        22        23
0    1.046720  0.998963  0.892836  0.913125  0.819484  0.733022  0.470827
1    0.706178  0.867865  0.978986  1.004269  1.056708  1.030489  0.817611
2    0.721161  0.812305  0.949333  0.985853  1.025494  0.970558  0.808559
3    0.746756  0.861934  0.951206  0.970246  1.016130  0.857565  0.576329
4    1.265216  1.189991  1.226511  1.179066  1.064511  1.013633  0.703681
..        …         …         …         …         …         …         …
283  0.888466  0.944651  1.116326  1.202164  1.384764  1.234938  0.219868
284  0.963379  0.965252  1.178754  1.305169  1.366036  1.234938  0.571959
285  1.141297  1.122569  1.073876  1.019564  0.969622  0.897830  0.527011
286  0.911876  0.607543  1.239620  1.099159  0.996153  0.875356  0.489555
287  0.545740  0.724594  0.874420  0.911876  0.886593  0.832281  0.373440

[288 rows x 24 columns]
```

5 Split the Data

In certain cases, we cannot use a random split. For example, if we are trying to predict the stock market, we cannot use a random split. We need to use a chronological split.

BUT, keep in mind if we have something like hourly readings of daily temperature, we can use a random split on days, but the sequence of the temperature within the day is important to remain sequential.

In this case, we are using a random split because each day as an independent sample.

[ ]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(temp_reshaped, test_size=0.3)

# if we neede to maintain the ordering of the data, we can use the following
 ↪code to split the data
#split_point = int(len(temp_reshaped)*0.7)
#train, test = temp_reshaped[:split_point], temp_reshaped[split_point:]
```

[ ]:
```
train.shape
```

[ ]:
```
(201, 24)
```

[ ]:
```
train[:2]
```

[ ]:
```
array([[-1.98631703, -1.8364912 , -1.85053737, -1.75689623, -1.90672206,
        -1.94886057, -1.92545029, -1.85521943, -1.79435268, -1.79435268,
        -1.76157828, -1.7709424 , -1.81776297, -1.87394766, -1.84117326,
        -1.82712708, -1.80839886, -1.74285005, -1.65857302, -1.55556776,
        -1.49001896, -1.46192662, -1.45256251, -1.42447016],
       [ 0.40153216,  0.44835273,  0.48580919,  0.47644508,  0.49985536,
         0.51858359,  0.54199388,  0.52794771,  0.54199388,  0.54667593,
         0.51858359,  0.57476828,  0.79014291,  0.60754268,  0.63095296,
         0.41089627,  0.66372737,  0.99147137,  1.03829194,  1.34730772,
         1.38944624,  1.29580509,  1.0757484 ,  1.01956372]])
```

[ ]:
```
test.shape
```

[ ]:
```
(87, 24)
```

[ ]:
```
test[:2]
```

[ ]:
```
array([[ 1.0236215 ,  0.4564683 ,  0.43274588,  0.54917303,  0.52014428,
         0.53824823,  0.60847909,  0.52825984,  0.44179785,  0.32537069,
         0.27043456,  0.38654958,  0.59536933,  0.77235109,  0.89096321,
         0.95994552,  1.15908902,  1.29642937,  1.46498343,  1.47715677,
         1.54613909,  1.49744569,  1.30329638,  1.18093862],
       [ 0.12529078,  0.05037787, -0.04326328,  0.01760347, -0.09008385,
```

```
-0.17436088, -0.23522762, -0.19308911, -0.27268408, -0.38505346,
-0.40378168, -0.38973551, -0.38505346, -0.20245322, -0.05262739,
 0.00355729,  0.07378815,  0.1674293 ,  0.20956781,  0.31257307,
 0.34534747,  0.3968501 ,  0.24234221,  0.05974198]])
```

6 Create Input and Target values

The first 23 hours will be input to predict the 24th hour reading (i.e., target)

[ ]:
```
# The first 23 columns (from 0 to 22) are inputs

train_inputs = train[:,:23]


pd.DataFrame(train_inputs, columns=np.arange(0,23,1))
```

[ ]:
```
            0         1         2         3         4         5         6   \
0   -1.986317 -1.836491 -1.850537 -1.756896 -1.906722 -1.948861 -1.925450
1    0.401532  0.448353  0.485809  0.476445  0.499855  0.518584  0.541994
2   -1.195049 -1.199731 -1.265280 -1.251234 -1.255916 -1.293373 -1.293373
3   -1.616435 -1.602388 -1.635163 -1.653891 -1.761578 -1.799035 -1.799035
4    0.982107  0.921241  0.696502  0.499855  0.401532  0.307891  0.270435
..        …         …         …         …         …         …         …
196  0.017603 -0.179043 -0.207135 -0.193089 -0.263320 -0.314823 -0.417828
197  1.234938  1.099159  0.935287  0.799507  0.752686  0.748004  0.724594
198  0.209568  0.312573  0.335983  0.335983  0.321937  0.247024  0.228296
199 -0.001125  0.012921 -0.052627 -0.282048 -0.394418 -0.483377 -0.558290
200  0.743322  0.663727  0.593497  0.462399  0.415578  0.424942  0.453035

            7         8         9   …        13        14        15  \
0   -1.855219 -1.794353 -1.794353  … -1.873948 -1.841173 -1.827127
1    0.527948  0.541994  0.546676  …  0.607543  0.630953  0.410896
2   -1.340193 -1.382332 -1.391696  … -1.424470 -1.330829 -1.363603
3   -1.799035 -1.850537 -1.883312  … -1.948861 -1.911404 -1.958225
4    0.247024  0.232978  0.209568  …  0.598179  0.762051  0.949333
..        …         …         …  …       …         …         …
196 -0.469330 -0.502105 -0.539561  … -0.605110 -0.464648 -0.239910
197  0.724594  0.701184  0.701184  …  1.089795  1.281759  1.436267
198  0.214250  0.232978  0.275117  …  0.719912  0.949333  1.127251
199 -0.633202 -0.684705 -0.722162  … -0.427192 -0.164997  0.289163
200  0.443671  0.382804  0.321937  …  0.640317  0.588814  0.701184

             16        17        18        19        20        21        22
0   -1.808399 -1.742850 -1.658573 -1.555568 -1.490019 -1.461927 -1.452563
1    0.663727  0.991471  1.038292  1.347308  1.389446  1.295805  1.075748
2   -1.419788 -1.424470 -1.316783 -1.326147 -1.059270 -1.031177 -1.293373
3   -1.770942 -1.387014 -1.054588 -0.829849 -0.956264 -0.829849 -1.457245
```

```
4    1.057020  1.234938  1.295805  1.239620  1.202164  1.155343  1.206846
..        …         …         …         …         …         …         …
196  0.055060  0.396850  0.626271  0.776097  0.902512  0.804189  0.443671
197  1.623549  1.717190  1.867016  2.040252  2.124529  2.152622  2.040252
198  1.375400  1.637595  1.628231  1.562682  1.431585  1.333262  1.230256
199  0.527948  0.635635  0.930605  0.986789  1.024246  1.103841  0.888466
200  0.822917  0.836963  0.836963  0.949333  0.963379  1.089795  0.860374

[201 rows x 23 columns]
```

6.1 Add one more dimension to make it ready for RNNs

See here for more details: https://keras.io/layers/recurrent/, and https://shiva-verma.medium.com/understanding-input-and-output-shape-in-lstm-keras-c501ee95c65e

[ ]:
```
train_inputs
```

[ ]:
```
array([[-1.98631703e+00, -1.83649120e+00, -1.85053737e+00, …,
        -1.49001896e+00, -1.46192662e+00, -1.45256251e+00],
       [ 4.01532160e-01,  4.48352732e-01,  4.85809190e-01, …,
         1.38944624e+00,  1.29580509e+00,  1.07574840e+00],
       [-1.19504936e+00, -1.19973142e+00, -1.26528022e+00, …,
        -1.05926970e+00, -1.03117735e+00, -1.29337256e+00],
       …,
       [ 2.09567813e-01,  3.12573072e-01,  3.35983359e-01, …,
         1.43158475e+00,  1.33326155e+00,  1.23025629e+00],
       [-1.12476250e-03,  1.29214092e-02, -5.26273921e-02, …,
         1.02424577e+00,  1.10384075e+00,  8.88466112e-01],
       [ 7.43322338e-01,  6.63727365e-01,  5.93496507e-01, …,
         9.63379028e-01,  1.08979457e+00,  8.60373769e-01]])
```

[ ]:
```
train_inputs.shape
```

[ ]:
```
(201, 23)
```

[ ]:
```
#Create an additional dimension for train

train_x = train_inputs[:,:,np.newaxis]

train_x.shape
```

[ ]:
```
(201, 23, 1)
```

[ ]:
```
train_x
```

[ ]:
```
array([[[-1.98631703e+00],
        [-1.83649120e+00],
        [-1.85053737e+00],
```

```
 …,
 [-1.49001896e+00],
 [-1.46192662e+00],
 [-1.45256251e+00]],

[[ 4.01532160e-01],
 [ 4.48352732e-01],
 [ 4.85809190e-01],
 …,
 [ 1.38944624e+00],
 [ 1.29580509e+00],
 [ 1.07574840e+00]],

[[-1.19504936e+00],
 [-1.19973142e+00],
 [-1.26528022e+00],
 …,
 [-1.05926970e+00],
 [-1.03117735e+00],
 [-1.29337256e+00]],

…,

[[ 2.09567813e-01],
 [ 3.12573072e-01],
 [ 3.35983359e-01],
 …,
 [ 1.43158475e+00],
 [ 1.33326155e+00],
 [ 1.23025629e+00]],

[[-1.12476250e-03],
 [ 1.29214092e-02],
 [-5.26273921e-02],
 …,
 [ 1.02424577e+00],
 [ 1.10384075e+00],
 [ 8.88466112e-01]],

[[ 7.43322338e-01],
 [ 6.63727365e-01],
 [ 5.93496507e-01],
 …,
 [ 9.63379028e-01],
 [ 1.08979457e+00],
 [ 8.60373769e-01]]])
```

6.2 Set the target

[ ]:
```python
# The last column (23) is TARGET

train_target = train[:,-1]


pd.DataFrame(train_target, columns=['23'])
```

[ ]:
```
           23
0   -1.424470
1    1.019564
2   -1.485337
3   -1.962907
4    1.047656
..        …
196  0.181475
197  0.410896
198  0.982107
199  0.560722
200  0.719912

[201 rows x 1 columns]
```

6.3 Repeat for TEST

[ ]:
```python
test.shape
```

[ ]:
```
(87, 24)
```

[ ]:
```python
# The first 23 columns (from 0 to 22) are inputs

test_inputs = test[:,:23]
```

[ ]:
```python
#Create an additional dimension for test

test_x = test_inputs[:,:,np.newaxis]

test_x.shape
```

[ ]:
```
(87, 23, 1)
```

[ ]:
```python
# The last column (23) is TARGET

test_target = test[:,-1]


pd.DataFrame(test_target, columns=['23'])
```

```
[ ]:           23
     0   1.180939
     1   0.059742
     2  -1.003085
     3   0.624086
     4  -0.347597
     ..       …
     82 -1.892676
     83  1.145979
     84 -0.033899
     85  1.057020
     86  0.390607

     [87 rows x 1 columns]
```

7 A normal (cross-sectional) NN

This model assumes that the data is NOT a time-series data set. It treats the data as cross-sectional and the columns being independent of each other.

```
[ ]: model = keras.models.Sequential([
         keras.layers.Flatten(input_shape=[23, 1]),
         keras.layers.Dense(23, activation='relu'),
         keras.layers.Dense(1, activation=None)

     ])

     model.compile(loss="mse", optimizer='Adam')

     history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
2024-04-09 07:30:43.848405: I metal_plugin/src/device/metal_device.cc:1154]
Metal device set to: Apple M1 Pro
2024-04-09 07:30:43.848428: I metal_plugin/src/device/metal_device.cc:296]
systemMemory: 16.00 GB
2024-04-09 07:30:43.848434: I metal_plugin/src/device/metal_device.cc:313]
maxCacheSize: 5.33 GB
2024-04-09 07:30:43.848452: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2024-04-09 07:30:43.848465: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
```

```
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)

Epoch 1/5

2024-04-09 07:30:44.428972: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:117]
Plugin optimizer for device_type GPU is enabled.

7/7          1s 24ms/step - loss:
0.7465
Epoch 2/5
7/7          0s 10ms/step - loss:
0.3339
Epoch 3/5
7/7          0s 8ms/step - loss:
0.2328
Epoch 4/5
7/7          0s 12ms/step - loss:
0.1573
Epoch 5/5
7/7          0s 9ms/step - loss:
0.1179
```

7.0.1 Predictions

[ ]:

```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3          0s 21ms/step
```

[ ]:

```
# Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

[ ]:

```
comparison
```

[ ]:

	actual	predicted
0	86.021	83.915245
1	75.245	75.834129
2	65.030	71.919403
3	80.669	83.106102
4	71.330	78.152061
..	…	…
82	56.480	76.081505
83	85.685	85.226166

```
84  74.345  76.045578
85  84.830  85.395248
86  78.425  81.538994

[87 rows x 2 columns]
```

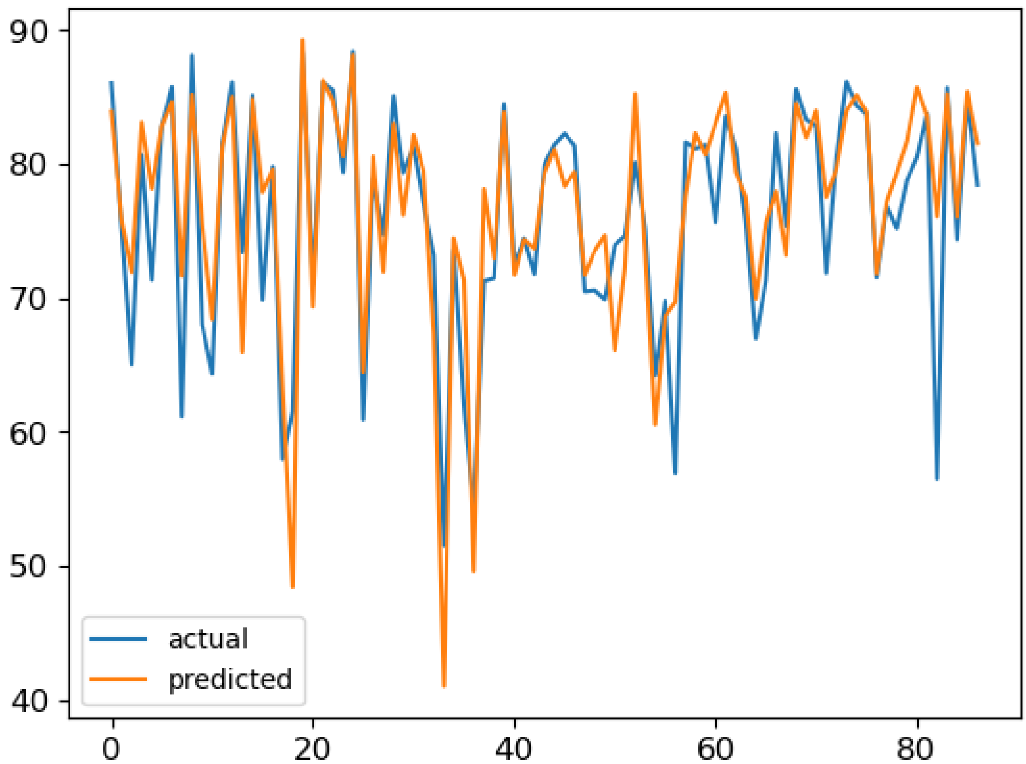
[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]:
```
22.148414581560083
```

[ ]:
```
plt.plot(comparison['actual'], label = 'actual')
plt.plot(comparison['predicted'], label = 'predicted')

plt.legend()

plt.show()
```



8 Simple RNN

Simplest recurrent neural network

[ ]:
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(32, activation='relu', input_shape=[23, 1]),
    keras.layers.Dense(1, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[ ]:
```
np.random.seed(42)
tf.random.set_seed(42)

model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7               3s 271ms/step - loss:
1.1885
Epoch 2/5
7/7               2s 250ms/step - loss:
0.9876
Epoch 3/5
7/7               2s 267ms/step - loss:
0.8507
Epoch 4/5
7/7               2s 255ms/step - loss:
0.7479
Epoch 5/5
7/7               2s 252ms/step - loss:
0.6585
```

8.0.1 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3               0s 119ms/step
```

[ ]:
```
#Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

[ ]:
```
comparison
```

[ ]:
```
    actual  predicted
0   86.021  80.769104
1   75.245  76.220955
2   65.030  74.551300
3   80.669  80.229492
4   71.330  75.179207
..     …        …
82  56.480  76.151115
83  85.685  80.316132
84  74.345  76.672333
85  84.830  80.800354
86  78.425  76.670654

[87 rows x 2 columns]
```

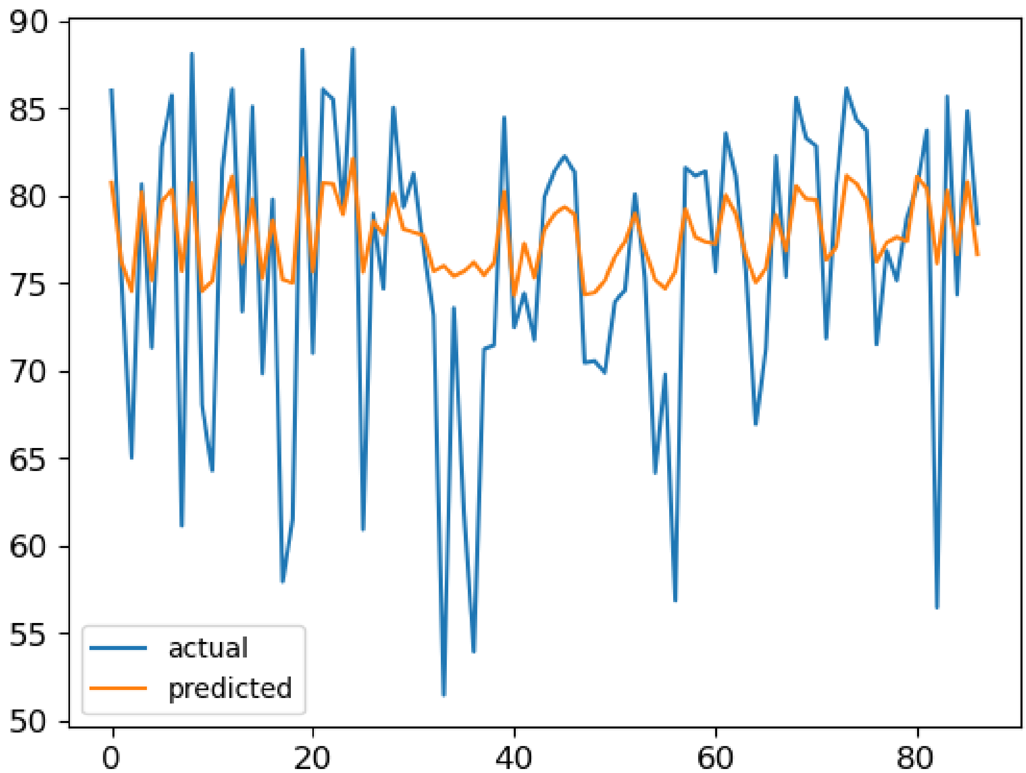
[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]:
```
49.83867603891561
```

[ ]:
```
plt.plot(comparison['actual'], label = 'actual')
plt.plot(comparison['predicted'], label = 'predicted')

plt.legend()

plt.show()
```



8.1 Simple RNN with more layers

Be careful: when stacking RNN layers, you have to set "return_sequences" to True. This enables the layer to send a "sequence" of values to the next layer – jut like how it uses a sequence of values for training. However, if the output of RNN is sent to a DENSE layer, then a single value should be sent. That's why there is no "return sequences" right before DENSE layers.

[ ]:
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(32, activation='relu', return_sequences=True,␣
 ↪input_shape=[23, 1]),
    keras.layers.SimpleRNN(32, activation='relu', return_sequences=False),
    keras.layers.Dense(1, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
[ ]: np.random.seed(42)
tf.random.set_seed(42)

model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7              7s 675ms/step - loss:
0.8615
Epoch 2/5
7/7              5s 692ms/step - loss:
0.3967
Epoch 3/5
7/7              4s 621ms/step - loss:
0.1550
Epoch 4/5
7/7              5s 695ms/step - loss:
0.0933
Epoch 5/5
7/7              5s 669ms/step - loss:
0.0709
```

8.1.1 Predictions

```
[ ]: #Predict:
y_pred = model.predict(test_x)
```

```
WARNING:tensorflow:5 out of the last 7 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x354e9f4c0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/3              0s
132ms/stepWARNING:tensorflow:6 out of the last 9 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x354e9f4c0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
```

```
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
3/3              1s 225ms/step
```

[ ]:
```
#Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

[ ]:
```
comparison
```

[ ]:

	actual	predicted
0	86.021	83.467468
1	75.245	75.912086
2	65.030	70.674118
3	80.669	82.433907
4	71.330	74.155128
..	…	…
82	56.480	59.918369
83	85.685	83.932892
84	74.345	75.744240
85	84.830	83.924927
86	78.425	77.446617

[87 rows x 2 columns]

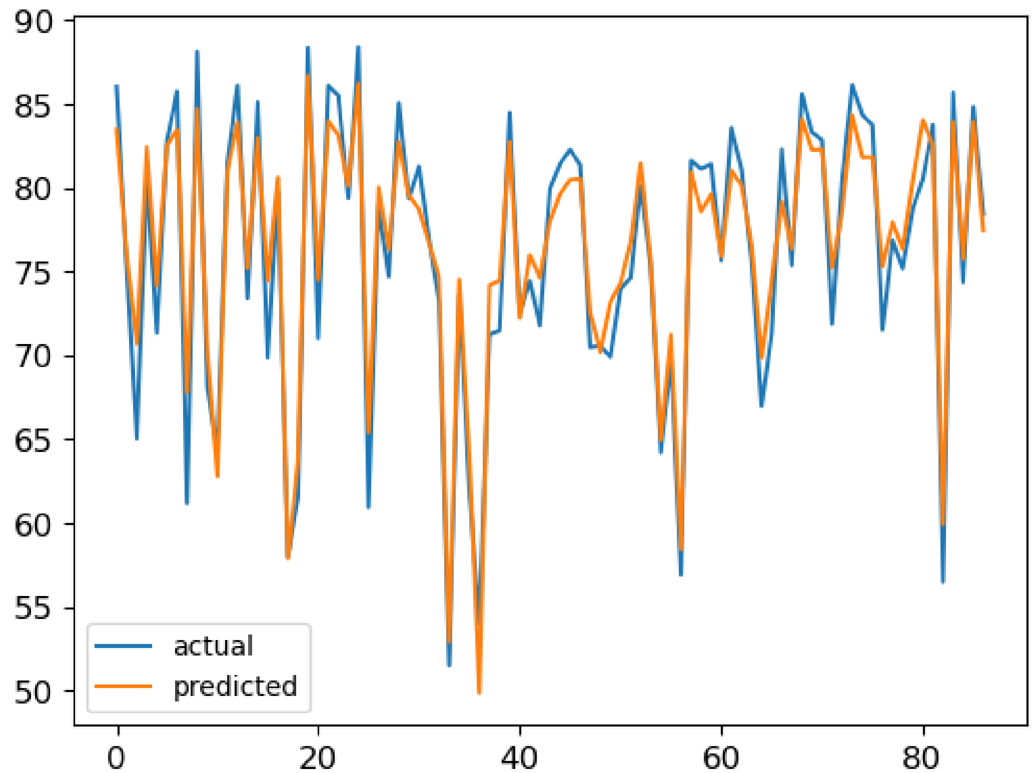
[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]: 5.183264379163942

[ ]:
```
plt.plot(comparison['actual'], label = 'actual')
plt.plot(comparison['predicted'], label = 'predicted')

plt.legend()

plt.show()
```



9 LSTM with one layer

[ ]:
```
model = keras.models.Sequential([
    keras.layers.LSTM(32, activation='relu', input_shape=[23, 1]),
    keras.layers.Dense(1, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[ ]:
```
np.random.seed(42)
tf.random.set_seed(42)

model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7              5s 440ms/step - loss:
```

```
1.2531
Epoch 2/5
7/7              3s 487ms/step - loss:
1.0240
Epoch 3/5
7/7              3s 479ms/step - loss:
0.8281
Epoch 4/5
7/7              3s 481ms/step - loss:
0.6236
Epoch 5/5
7/7              3s 467ms/step - loss:
0.5775
```

9.0.1 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3              1s 154ms/step
```

[ ]:
```
#Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

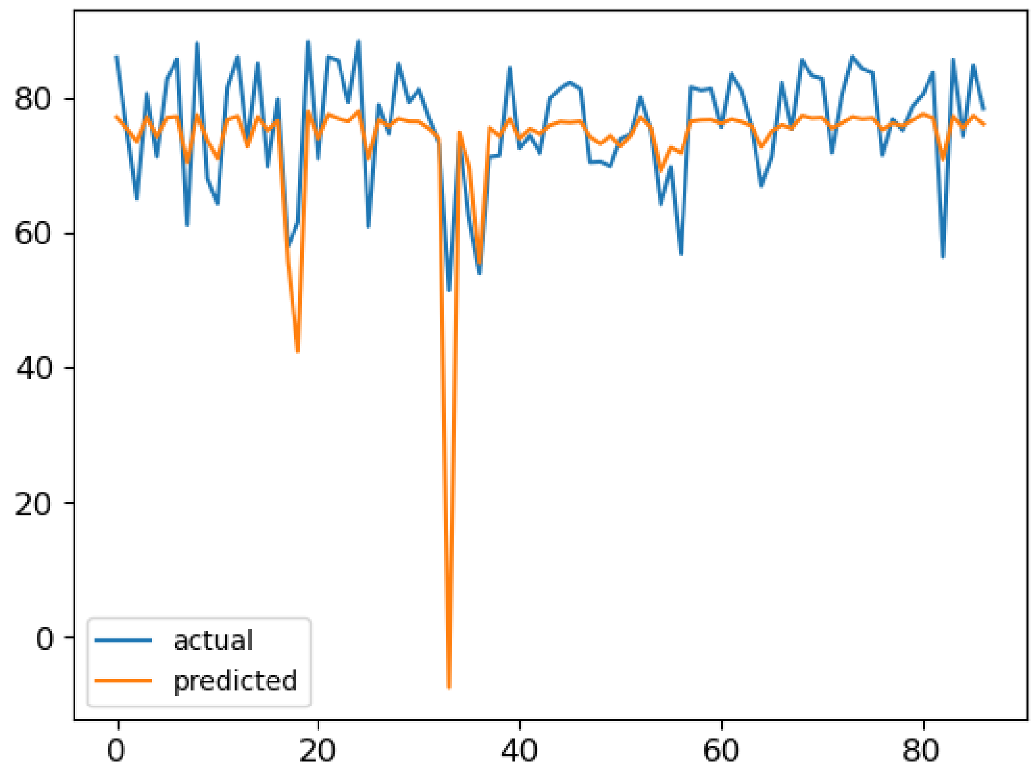
[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]:
```
77.44127969120099
```

[ ]:
```
plt.plot(comparison['actual'], label = 'actual')
plt.plot(comparison['predicted'], label = 'predicted')

plt.legend()

plt.show()
```



10 LSTM with more layers

[ ]:
```
model = keras.models.Sequential([
    keras.layers.LSTM(32, activation='tanh', return_sequences=True,␣
 ↪input_shape=[23, 1]),
    keras.layers.LSTM(32, activation='tanh', return_sequences=False),
    keras.layers.Dense(1, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[ ]:
```
np.random.seed(42)
tf.random.set_seed(42)

model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7            2s 31ms/step - loss:
0.7669
Epoch 2/5
7/7            0s 16ms/step - loss:
0.2524
Epoch 3/5
7/7            0s 14ms/step - loss:
0.1259
Epoch 4/5
7/7            0s 15ms/step - loss:
0.1136
Epoch 5/5
7/7            0s 14ms/step - loss:
0.0880
```

10.0.1 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3            0s 101ms/step
```

[ ]:
```
#Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

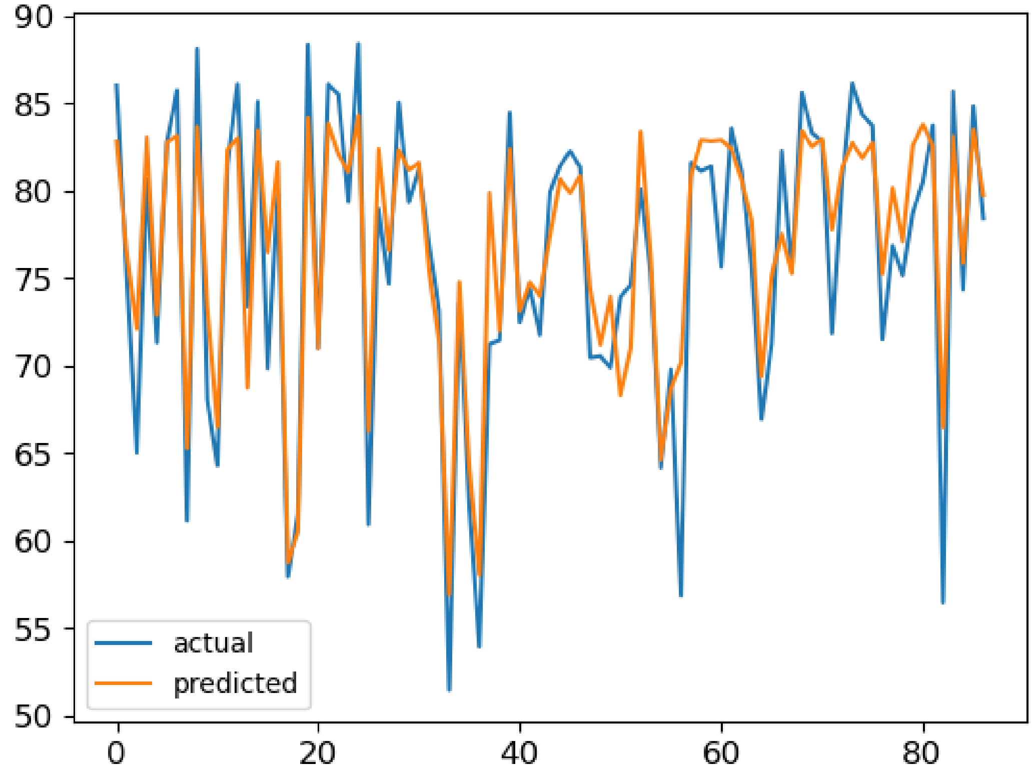
[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]:
```
12.661517235157207
```

[ ]:
```
plt.plot(comparison['actual'], label = 'actual')
plt.plot(comparison['predicted'], label = 'predicted')

plt.legend()

plt.show()
```



11 GRU (with more layers)

```
[ ]: model = keras.models.Sequential([
         keras.layers.GRU(32, activation='relu', return_sequences=True,␣
      ↪input_shape=[23, 1]),
         keras.layers.GRU(32, activation='relu', return_sequences=False),
         keras.layers.Dense(1, activation=None)
     ])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
[ ]: np.random.seed(42)
     tf.random.set_seed(42)

     model.compile(loss="mse", optimizer='RMSprop')

     history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7            13s 2s/step - loss:
0.8036
Epoch 2/5
7/7            11s 2s/step - loss:
0.5600
Epoch 3/5
7/7            11s 2s/step - loss:
0.4047
Epoch 4/5
7/7            11s 2s/step - loss:
0.2556
Epoch 5/5
7/7            11s 2s/step - loss:
0.1227
```

11.0.1 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3            1s 358ms/step
```

[ ]:
```
#Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

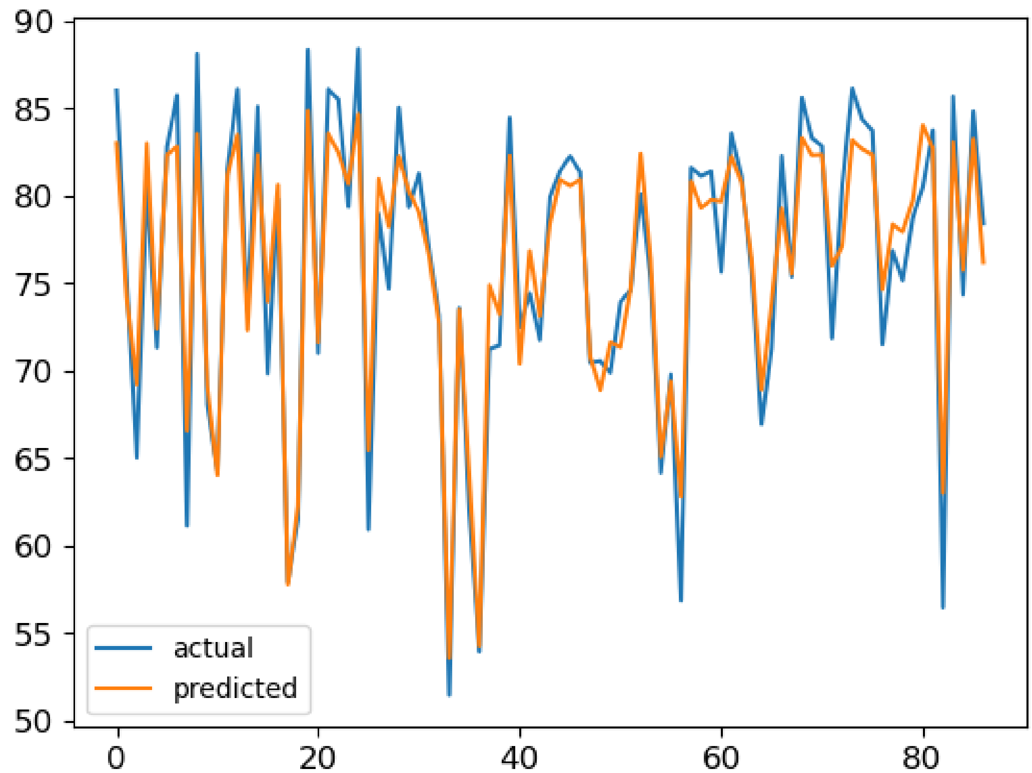
[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]: 5.907716083886514

[ ]:
```
plt.plot(comparison['actual'], label = 'actual')
plt.plot(comparison['predicted'], label = 'predicted')

plt.legend()

plt.show()
```



12 Conv1D

12.0.1 Last Layer: GRU (you can change it to SimpleRNN or LSTM as well)

[ ]:
```
model = keras.models.Sequential([
    keras.layers.Conv1D(filters=20, kernel_size=3, strides=1, padding="valid",␣
  ↪input_shape=[23, 1]),
    keras.layers.GRU(32, activation='relu', return_sequences=True),
    keras.layers.GRU(32, activation='relu', return_sequences=False),
    keras.layers.Dense(1, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:99: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(
```

[ ]:
```
np.random.seed(42)
tf.random.set_seed(42)
```

```
model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7              14s 1s/step - loss:
0.9220
Epoch 2/5
7/7              10s 1s/step - loss:
0.7516
Epoch 3/5
7/7              10s 1s/step - loss:
0.6433
Epoch 4/5
7/7              10s 1s/step - loss:
0.5135
Epoch 5/5
7/7              10s 1s/step - loss:
0.3517
```

12.0.2 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3              1s 325ms/step
```

[ ]:
```
#Remember, these are standardized values.

comparison = pd.DataFrame()

comparison['actual'] = scaler.inverse_transform([test_target]).flatten()
comparison['predicted'] = scaler.inverse_transform(y_pred).flatten()
```

[ ]:
```
mean_squared_error(comparison['actual'], comparison['predicted'])
```

[ ]: 18.93950135820779

13 Forecasting Several Steps Ahead

13.1 Now let's create an RNN that predicts 12 next values at once:

[ ]:
```
# The first 12 columns (from 0 to 11) are inputs

train_inputs = train[:,:12]

pd.DataFrame(train_inputs, columns=np.arange(0,12,1))
```

[ ]:
```
          0         1         2         3         4         5         6   \
0   -1.986317 -1.836491 -1.850537 -1.756896 -1.906722 -1.948861 -1.925450
1    0.401532  0.448353  0.485809  0.476445  0.499855  0.518584  0.541994
2   -1.195049 -1.199731 -1.265280 -1.251234 -1.255916 -1.293373 -1.293373
3   -1.616435 -1.602388 -1.635163 -1.653891 -1.761578 -1.799035 -1.799035
4    0.982107  0.921241  0.696502  0.499855  0.401532  0.307891  0.270435
..        …         …         …         …         …         …         …
196  0.017603 -0.179043 -0.207135 -0.193089 -0.263320 -0.314823 -0.417828
197  1.234938  1.099159  0.935287  0.799507  0.752686  0.748004  0.724594
198  0.209568  0.312573  0.335983  0.335983  0.321937  0.247024  0.228296
199 -0.001125  0.012921 -0.052627 -0.282048 -0.394418 -0.483377 -0.558290
200  0.743322  0.663727  0.593497  0.462399  0.415578  0.424942  0.453035

          7         8         9         10        11
0   -1.855219 -1.794353 -1.794353 -1.761578 -1.770942
1    0.527948  0.541994  0.546676  0.518584  0.574768
2   -1.340193 -1.382332 -1.391696 -1.401060 -1.424470
3   -1.799035 -1.850537 -1.883312 -1.939496 -1.972271
4    0.247024  0.232978  0.209568  0.200204  0.167429
..        …         …         …         …         …
196 -0.469330 -0.502105 -0.539561 -0.548925 -0.516151
197  0.724594  0.701184  0.701184  0.691820  0.738640
198  0.214250  0.232978  0.275117  0.284481  0.211128
199 -0.633202 -0.684705 -0.722162 -0.754936 -0.797074
200  0.443671  0.382804  0.321937  0.331301  0.429625

[201 rows x 12 columns]
```

[ ]:
```python
#Create an additional dimension for train

train_x = train_inputs.reshape(201,12,1)

train_x.shape
```

[ ]:
```
(201, 12, 1)
```

[ ]:
```python
# The last 12 readings (from 12 to 23) are TARGET

train_target = train[:,-12:]

pd.DataFrame(train_target, columns=np.arange(12,24,1))
```

[ ]:
```
          12        13        14        15        16        17        18  \
0   -1.817763 -1.873948 -1.841173 -1.827127 -1.808399 -1.742850 -1.658573
1    0.790143  0.607543  0.630953  0.410896  0.663727  0.991471  1.038292
2   -1.415106 -1.424470 -1.330829 -1.363603 -1.419788 -1.424470 -1.316783
3   -2.000363 -1.948861 -1.911404 -1.958225 -1.770942 -1.387014 -1.054588
```

```
4    0.415578  0.598179  0.762051  0.949333  1.057020  1.234938  1.295805
..        ...       ...       ...       ...       ...       ...       ...
196 -0.614474 -0.605110 -0.464648 -0.239910  0.055060  0.396850  0.626271
197  0.935287  1.089795  1.281759  1.436267  1.623549  1.717190  1.867016
198  0.401532  0.719912  0.949333  1.127251  1.375400  1.637595  1.628231
199 -0.670659 -0.427192 -0.164997  0.289163  0.527948  0.635635  0.930605
200  0.654363  0.640317  0.588814  0.701184  0.822917  0.836963  0.836963

           19        20        21        22        23
0   -1.555568 -1.490019 -1.461927 -1.452563 -1.424470
1    1.347308  1.389446  1.295805  1.075748  1.019564
2   -1.326147 -1.059270 -1.031177 -1.293373 -1.485337
3   -0.829849 -0.956264 -0.829849 -1.457245 -1.962907
4    1.239620  1.202164  1.155343  1.206846  1.047656
..        ...       ...       ...       ...       ...
196  0.776097  0.902512  0.804189  0.443671  0.181475
197  2.040252  2.124529  2.152622  2.040252  0.410896
198  1.562682  1.431585  1.333262  1.230256  0.982107
199  0.986789  1.024246  1.103841  0.888466  0.560722
200  0.949333  0.963379  1.089795  0.860374  0.719912

[201 rows x 12 columns]
```

13.2 Repeat for TEST

[ ]:
```python
# The first 12 columns (from 0 to 11) are inputs

test_inputs = test[:,:12]

pd.DataFrame(test_inputs, columns=np.arange(0,12,1))
```

[ ]:
```
           0         1         2         3         4         5         6  \
0    1.023621  0.456468  0.432746  0.549173  0.520144  0.538248  0.608479
1    0.125291  0.050378 -0.043263  0.017603 -0.090084 -0.174361 -0.235228
2   -0.445920 -0.497423 -0.516151 -0.511469 -0.544243 -0.614474 -0.637885
3    0.791704  0.651554  0.591936  0.510780  0.529196  0.490803  0.448353
4   -0.403782 -0.417828 -0.455284 -0.544243 -0.619156 -0.656613 -0.698751
..        ...       ...       ...       ...       ...       ...       ...
82  -0.047945 -0.076038 -0.108812 -0.155633 -0.174361 -0.146269 -0.090084
83   0.982107  0.907194  0.925923  0.701184  0.574768  0.513902  0.438989
84  -0.202453 -0.272684 -0.347597 -0.342915 -0.445920 -0.534879 -0.600428
85   1.038292  0.982107  0.958697  0.916558  0.846328  0.841646  0.804189
86   0.768293  0.689947  0.624398  0.571959  0.526387  0.453971  0.464584

           7         8         9        10        11
0    0.528260  0.441798  0.325371  0.270435  0.386550
1   -0.193089 -0.272684 -0.385053 -0.403782 -0.389736
```

```
2  -0.609792 -0.619156 -0.651931 -0.605110 -0.708115
3   0.331613  0.312885  0.323810  0.280111  0.374688
4  -0.759618 -0.815803 -0.867305 -0.909444 -0.923490
..       …        …        …        …        …
82 -0.076038 -0.450602 -0.750254 -0.839213 -0.974993
83  0.406214  0.387486  0.387486  0.373440  0.392168
84 -0.633202 -0.628520 -0.670659 -0.689387 -0.708115
85  0.752686  0.752686  0.724594  0.663727  0.743322
86  0.412457  0.359394  0.273244  0.152135  0.148389

[87 rows x 12 columns]
```

[ ]:
```python
#Create an additional dimension for test

test_x = test_inputs.reshape(87,12,1)

test_x.shape
```

[ ]:
```
(87, 12, 1)
```

[ ]:
```python
# The last 12 columns are TARGET

test_target = test[:,-12:]

pd.DataFrame(test_target, columns=np.arange(12,24,1))
```

[ ]:
```
          12        13        14        15        16        17        18  \
0   0.595369  0.772351  0.890963  0.959946  1.159089  1.296429  1.464983
1  -0.385053 -0.202453 -0.052627  0.003557  0.073788  0.167429  0.209568
2  -0.825167 -0.806439 -0.431874 -0.352279 -0.427192 -0.310141 -0.188407
3   0.583820  0.839461  1.129124  1.293932  1.375400  1.511804  1.636659
4  -0.909444 -1.007767 -0.960946 -0.698751 -0.314823  0.415578  0.560722
..       …        …        …        …        …        …        …
82 -1.195049 -1.312101 -1.227824 -1.049906 -1.092044 -1.096726 -1.007767
83  0.626271  0.715230  0.916558  1.136615  1.469041  1.651641  1.796785
84 -0.764300 -0.661295 -0.352279  0.026968  0.331301  0.509219  0.654363
85  0.265752  0.893148  1.267713  1.108523  1.314533  1.628231  1.548636
86  0.182412  0.219556  0.229857  0.283232  0.426503  0.458029  0.474884

          19        20        21        22        23
0   1.477157  1.546139  1.497446  1.303296  1.180939
1   0.312573  0.345347  0.396850  0.242342  0.059742
2   0.031650 -0.033899 -0.108812 -0.436556 -1.003085
3   1.641965  1.550862  1.386013  1.051090  0.624086
4   0.659045  0.340665  0.069106 -0.169679 -0.347597
..       …        …        …        …        …
82 -1.026495 -1.106090 -1.241870 -1.630481 -1.892676
```

```
83  1.515862  1.450313  1.220892  1.384764  1.145979
84  0.630953  0.598179  0.457717  0.223614 -0.033899
85  1.600139  1.454995  1.511180  1.277077  1.057020
86  0.409960  0.395289  0.421509  0.435555  0.390607

[87 rows x 12 columns]
```

14 GRU

[ ]:
```
model = keras.models.Sequential([
    keras.layers.GRU(32, activation='relu', return_sequences=True,␣
 ↪input_shape=[12, 1]),
    keras.layers.GRU(32, activation='relu', return_sequences=False),
    keras.layers.Dense(12, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[ ]:
```
np.random.seed(42)
tf.random.set_seed(42)

model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_x, train_target, epochs=epoch_num)
```

```
Epoch 1/5
7/7               7s 536ms/step - loss:
1.1441
Epoch 2/5
7/7               3s 474ms/step - loss:
1.0978
Epoch 3/5
7/7               3s 487ms/step - loss:
1.0587
Epoch 4/5
7/7               3s 488ms/step - loss:
1.0147
Epoch 5/5
7/7               3s 459ms/step - loss:
0.9574
```

14.0.1 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_x)
```

```
3/3            1s 232ms/step
```

[ ]:
```
#Remember, these are standardized values.

actual = pd.DataFrame(scaler.inverse_transform(test_target))
predicted = pd.DataFrame(scaler.inverse_transform(y_pred))
```

[ ]:
```
actual
```

[ ]:
```
        0       1       2       3       4       5       6       7          8  \
0   80.393  82.094  83.234  83.897  85.811  87.131  88.751  88.868  89.531000
1   70.970  72.725  74.165  74.705  75.380  76.280  76.685  77.675  77.990000
2   66.740  66.920  70.520  71.285  70.565  71.690  72.860  74.975  74.345000
3   80.282  82.739  85.523  87.107  87.890  89.201  90.401  90.452  89.576393
4   65.930  64.985  65.435  67.955  71.645  78.665  80.060  81.005  77.945000
..     …       …       …       …       …       …       …       …          …
82  63.185  62.060  62.870  64.580  64.175  64.130  64.985  64.805  64.040000
83  80.690  81.545  83.480  85.595  88.790  90.545  91.940  89.240  88.610000
84  67.325  68.315  71.285  74.930  77.855  79.565  80.960  80.735  80.420000
85  77.225  83.255  86.855  85.325  87.305  90.320  89.555  90.050  88.655000
86  76.424  76.781  76.880  77.393  78.770  79.073  79.235  78.611  78.470000

        9      10      11
0   89.063  87.197  86.021
1   78.485  77.000  75.245
2   73.625  70.475  65.030
3   87.992  84.773  80.669
4   75.335  73.040  71.330
..     …       …       …
82  62.735  59.000  56.480
83  86.405  87.980  85.685
84  79.070  76.820  74.345
85  89.195  86.945  84.830
86  78.722  78.857  78.425

[87 rows x 12 columns]
```

[ ]:
```
predicted
```

[ ]:
```
           0          1          2          3          4          5  \
0  75.710274  76.159607  75.889503  76.173485  74.491684  74.442482
1  73.025398  73.948112  73.579193  75.142609  74.772263  75.695740
2  72.061836  73.643280  72.786995  74.916153  74.808380  75.914146
```

```
3   75.603302  76.022293  75.804817  76.094887  74.528267  74.488770
4   71.479134  73.488319  72.309883  74.831909  74.780487  76.029640
..        …          …          …          …          …          …
82  72.156029  73.719513  72.835388  74.997971  74.731590  75.878960
83  75.752594  76.221657  75.922379  76.189705  74.491486  74.412437
84  72.015129  73.631264  72.741776  74.907677  74.806061  75.923180
85  76.355865  76.999901  76.385094  76.680244  74.223328  74.172867
86  75.486946  75.841896  75.727425  76.083000  74.508736  74.581299

            6          7          8          9         10         11
0   76.356964  76.361732  76.299683  77.409576  75.419708  77.005798
1   74.771301  74.884758  75.348892  75.261559  74.148224  75.608627
2   74.319107  74.430626  75.067566  74.989601  73.326004  75.468018
3   76.259109  76.263405  76.222855  77.234825  75.389908  76.862595
4   74.029396  74.143074  74.919838  74.866920  72.793411  75.389748
..        …          …          …          …          …          …
82  74.342766  74.434380  75.128151  75.062981  73.335808  75.460693
83  76.390038  76.391083  76.312370  77.477257  75.431549  77.045738
84  74.293282  74.398834  75.052490  74.976540  73.278160  75.452759
85  76.962563  76.986366  76.803223  78.439301  75.542992  77.886246
86  76.179092  76.197670  76.219551  77.085602  75.360512  76.803185

[87 rows x 12 columns]
```

[ ]:
```
mean_squared_error(actual, predicted)
```

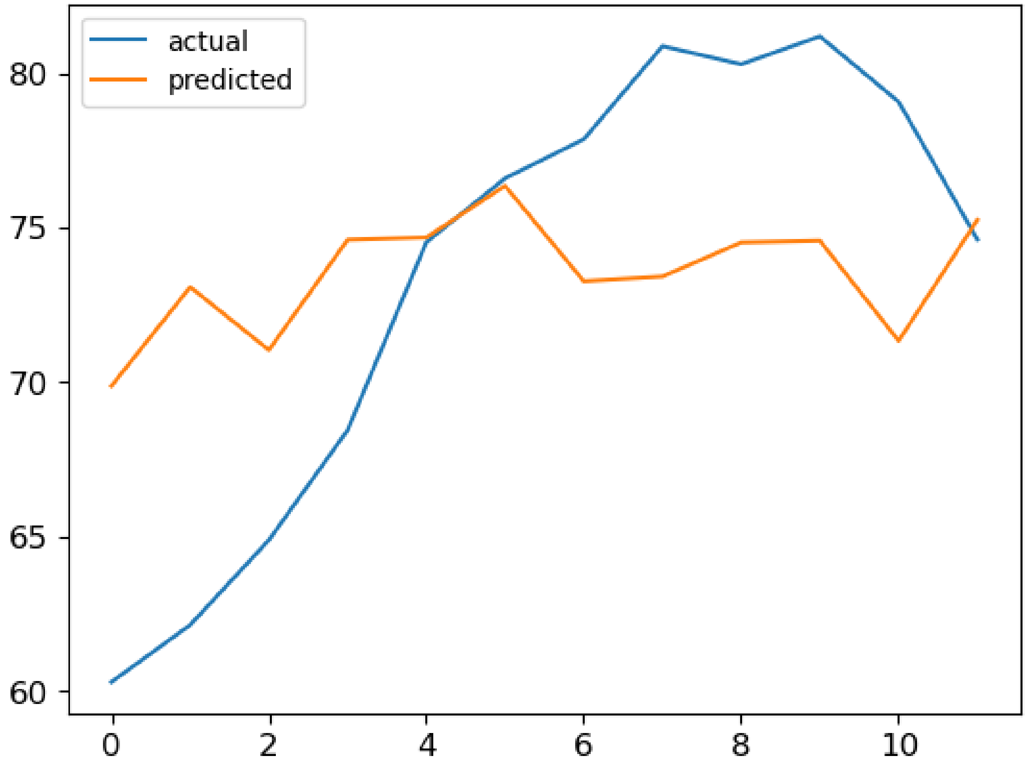
[ ]:
```
78.06923496947906
```

[ ]:
```
# Plot a random row to see the accuracy of predictions

random_row = np.random.randint(low=0, high=86)

plt.plot(actual.iloc[random_row], label='actual')
plt.plot(predicted.iloc[random_row], label='predicted')

plt.legend()
plt.show()
```



15 Sliding window

Prior 18 hours predicts next 6 hours

[ ]:
```
steps_for_prediction = 18
steps_to_predict = 6

#Be careful: sums to 24 hours
```

[ ]:
```
train
```

[ ]:
```
array([[-1.98631703e+00, -1.83649120e+00, -1.85053737e+00, …,
        -1.46192662e+00, -1.45256251e+00, -1.42447016e+00],
       [ 4.01532160e-01,  4.48352732e-01,  4.85809190e-01, …,
         1.29580509e+00,  1.07574840e+00,  1.01956372e+00],
       [-1.19504936e+00, -1.19973142e+00, -1.26528022e+00, …,
        -1.03117735e+00, -1.29337256e+00, -1.48533691e+00],
       …,
       [ 2.09567813e-01,  3.12573072e-01,  3.35983359e-01, …,
         1.33326155e+00,  1.23025629e+00,  9.82107257e-01],
       [-1.12476250e-03,  1.29214092e-02, -5.26273921e-02, …,
         1.10384075e+00,  8.88466112e-01,  5.60722106e-01],
```

```
[ 7.43322338e-01,  6.63727365e-01,  5.93496507e-01, …,
  1.08979457e+00,  8.60373769e-01,  7.19912052e-01]])
```

[ ]:
```
train.flatten().shape
```

[ ]:
```
(4824,)
```

[ ]:
```
train_inputs_sw = []
train_target_sw = []

for i in range(0,4824-24):
    input_row = train.flatten()[i:i+steps_for_prediction]
    target_row = train.flatten()[i+steps_for_prediction:
 ↪i+steps_for_prediction+steps_to_predict]
    train_inputs_sw.append((input_row))
    train_target_sw.append((target_row))
```

[ ]:
```
train_inputs = np.vstack(train_inputs_sw)

train_targets = np.vstack(train_target_sw)
```

[ ]:
```
train_targets.shape
```

[ ]:
```
(4800, 6)
```

[ ]:
```
# Repeat for test

test_inputs_sw = []
test_target_sw = []

for i in range(0,test.flatten().shape[0]-24):
    input_row = test.flatten()[i:i+steps_for_prediction]
    target_row = test.flatten()[i+steps_for_prediction:
 ↪i+steps_for_prediction+steps_to_predict]
    test_inputs_sw.append((input_row))
    test_target_sw.append((target_row))

test_inputs = np.vstack(test_inputs_sw)

test_targets = np.vstack(test_target_sw)
```

16 GRU

[ ]:
```
model = keras.models.Sequential([
    keras.layers.GRU(32, activation='relu', return_sequences=True,␣
 ↪input_shape=[18, 1]),
    keras.layers.GRU(32, activation='relu', return_sequences=False),
```

```
    keras.layers.Dense(steps_to_predict, activation=None)
])
```

```
/Users/timsmith/miniconda3/envs/dsp/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[ ]:
```
np.random.seed(42)
tf.random.set_seed(42)

model.compile(loss="mse", optimizer='Adam')

history = model.fit(train_inputs, train_targets, epochs=epoch_num)
```

```
Epoch 1/5
150/150            85s 549ms/step -
loss: 0.7335
Epoch 2/5
150/150            83s 555ms/step -
loss: 0.2851
Epoch 3/5
150/150            77s 513ms/step -
loss: 0.2646
Epoch 4/5
150/150            54s 362ms/step -
loss: 0.2550
Epoch 5/5
150/150            45s 299ms/step -
loss: 0.2476
```

16.0.1 Predictions

[ ]:
```
#Predict:
y_pred = model.predict(test_inputs)
```

```
65/65            4s 57ms/step
```

[ ]:
```
#Remember, these are standardized values.

actual = pd.DataFrame(scaler.inverse_transform(test_targets))
predicted = pd.DataFrame(scaler.inverse_transform(y_pred))
```

[ ]:
```
actual
```

[ ]:

	0	1	2	3	4	5
0	88.751	88.868	89.531	89.063	87.197	86.021
1	88.868	89.531	89.063	87.197	86.021	75.875

```
2     89.531  89.063  87.197  86.021  75.875  75.155
3     89.063  87.197  86.021  75.875  75.155  74.255
4     87.197  86.021  75.875  75.155  74.255  74.840
…      …       …       …       …       …       …
2059  76.781  76.880  77.393  78.770  79.073  79.235
2060  76.880  77.393  78.770  79.073  79.235  78.611
2061  77.393  78.770  79.073  79.235  78.611  78.470
2062  78.770  79.073  79.235  78.611  78.470  78.722
2063  79.073  79.235  78.611  78.470  78.722  78.857

[2064 rows x 6 columns]
```

[ ]:
```
predicted
```

[ ]:
```
              0          1          2          3          4          5
0     87.054977  87.487541  85.834480  85.526680  84.404007  83.329414
1     88.089355  87.415459  84.871231  84.101242  82.409752  80.780205
2     87.591972  85.875031  83.330467  81.639549  79.749985  78.018005
3     87.657585  85.078636  82.162445  80.146370  78.001503  76.134148
4     86.442322  83.334702  80.434105  77.983383  75.658669  74.115738
…           …          …          …          …          …          …
2059  76.865219  78.937889  80.954544  82.301857  84.409531  85.125687
2060  77.326805  79.606239  81.456772  82.817108  84.677170  85.337708
2061  77.405602  79.738190  81.370369  82.691170  84.216827  84.952332
2062  77.916618  80.251701  81.486099  82.808311  83.912796  84.648819
2063  79.387184  81.782616  82.611473  83.948517  84.647949  85.253090

[2064 rows x 6 columns]
```

[ ]:
```
mean_squared_error(actual, predicted)
```

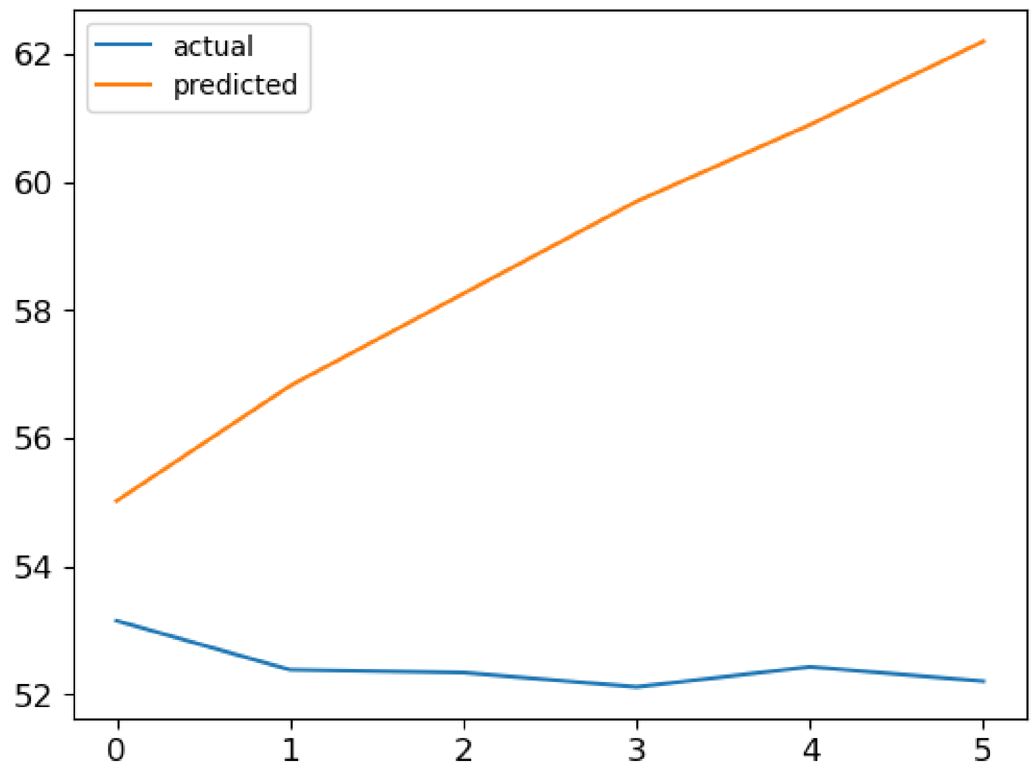
[ ]:
```
24.017596273764052
```

[ ]:
```
# Plot a random row to see the accuracy of predictions

random_row = np.random.randint(low=0, high=2063)

plt.plot(actual.iloc[random_row], label='actual')
plt.plot(predicted.iloc[random_row], label='predicted')

plt.legend()
plt.show()
```



16.1 We could try using 6 steps to predict the next 6 steps (maybe 12 steps is too long)

[ ]:

```

```