In [1]:

```
#weather predictin on time series data
import tensorflow as tf
import os
import pandas as pd
import numpy as np
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
#Download dataset
zip_path = tf.keras.utils.get_file(
    origin='https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip',
    fname='jena_climate_2009_2016.csv.zip',
    extract=True)
csv_path, _ = os.path.splitext(zip_path)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016.csv.zip
13574144/13568290 [==============================] - 2s 0us/step
13582336/13568290 [==============================] - 2s 0us/step
```

In [3]:

```
#Read the data
data = pd.read_csv(csv_path, parse_dates=True, index_col=0)
data.head()
```

Out[3]:

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
Date Time														
2009-01-01 00:10:00	996.52	-8.02	265.40	-8.90	93.3	3.33	3.11	0.22	1.94	3.12	1307.75	1.03	1.75	152.3
2009-01-01 00:20:00	996.57	-8.41	265.01	-9.28	93.4	3.23	3.02	0.21	1.89	3.03	1309.80	0.72	1.50	136.1
2009-01-01 00:30:00	996.53	-8.51	264.91	-9.31	93.9	3.21	3.01	0.20	1.88	3.02	1310.24	0.19	0.63	171.6
2009-01-01 00:40:00	996.51	-8.31	265.12	-9.07	94.2	3.26	3.07	0.19	1.92	3.08	1309.19	0.34	0.50	198.0
2009-01-01 00:50:00	996.51	-8.27	265.15	-9.04	94.1	3.27	3.08	0.19	1.92	3.09	1309.00	0.32	0.63	214.3

In [4]:

```
len(data)
```

Out[4]:

```
420551
```

In [5]:

```
# Limit dataset
data = data[5::6]
```

In [6]:

```
len(data)
```

Out[6]:

```
70091
```

In [7]:

```
data.head()
```

Out[7]:

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)	max. wv (m/s)	wd (deg)
Date Time														
2009-01-01 01:00:00	996.50	-8.05	265.38	-8.78	94.4	3.33	3.14	0.19	1.96	3.15	1307.86	0.21	0.63	192.7
2009-01-01 02:00:00	996.62	-8.88	264.54	-9.77	93.2	3.12	2.90	0.21	1.81	2.91	1312.25	0.25	0.63	190.3
2009-01-01 03:00:00	996.84	-8.81	264.59	-9.66	93.5	3.13	2.93	0.20	1.83	2.94	1312.18	0.18	0.63	167.2
2009-01-01 04:00:00	996.99	-9.05	264.34	-10.02	92.6	3.07	2.85	0.23	1.78	2.85	1313.61	0.10	0.38	240.0
2009-01-01 05:00:00	997.46	-9.63	263.72	-10.65	92.2	2.94	2.71	0.23	1.69	2.71	1317.19	0.40	0.88	157.0

In [8]:

```
#Investigate data,what columns are correlated and not
data.corr()
```

Out[8]:

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	wv (m/s)
p (mbar)	1.000000	-0.045296	-0.124643	-0.066698	-0.018363	-0.031455	-0.054353	-0.003283	-0.069749	-0.069792	0.307583	-0.005740
T (degC)	-0.045296	1.000000	0.996826	0.895706	-0.572593	0.951080	0.867691	0.761672	0.866770	0.867195	-0.963404	-0.004923
Tpot (K)	-0.124643	0.996826	1.000000	0.894909	-0.567306	0.947259	0.866228	0.756886	0.866553	0.866978	-0.981342	-0.004426
Tdew (degC)	-0.066698	0.895706	0.894909	1.000000	-0.156834	0.799182	0.968361	0.435689	0.967614	0.968061	-0.885231	-0.008581
rh (%)	-0.018363	-0.572593	-0.567306	-0.156834	1.000000	-0.616019	-0.151704	-0.843768	-0.151049	-0.151181	0.514461	-0.004227
VPmax (mbar)	-0.031455	0.951080	0.947259	0.799182	-0.616019	1.000000	0.824758	0.875639	0.824349	0.824386	-0.901488	-0.004358
VPact (mbar)	-0.054353	0.867691	0.866228	0.968361	-0.151704	0.824758	1.000000	0.449080	0.999851	0.999856	-0.850271	-0.009390
VPdef (mbar)	-0.003283	0.761672	0.756886	0.435689	-0.843768	0.875639	0.449080	1.000000	0.448561	0.448615	-0.698195	0.001138
sh (g/kg)	-0.069749	0.866770	0.866553	0.967614	-0.151049	0.824349	0.999851	0.448561	1.000000	0.999997	-0.853354	-0.009270
H2OC (mmol/mol)	-0.069792	0.867195	0.866978	0.968061	-0.151181	0.824386	0.999856	0.448615	0.999997	1.000000	-0.853801	-0.009272
rho (g/m3)**	0.307583	-0.963404	-0.981342	-0.885231	0.514461	-0.901488	-0.850271	-0.698195	-0.853354	-0.853801	1.000000	0.003418
wv (m/s)	-0.005740	-0.004923	-0.004426	-0.008581	-0.004227	-0.004358	-0.009390	0.001138	-0.009270	-0.009272	0.003418	1.000000
max. wv (m/s)	-0.007360	-0.003884	-0.003263	-0.009693	-0.008641	-0.003154	-0.010883	0.004315	-0.010736	-0.010736	0.002138	0.865946
wd (deg)	-0.063678	0.041577	0.046465	0.052507	-0.017297	-0.006787	0.020988	-0.028644	0.021961	0.022195	-0.060856	-0.015354

In [9]:

```
#Remove some data
df = data.drop(['wv (m/s)', 'max. wv (m/s)', 'wd (deg)'], axis=1)
```

In [10]:

```
#Add periodic time intervals
timestamp_s = df.index
timestamp_s = timestamp_s.map(pd.Timestamp.timestamp)
```

In [11]:

```
timestamp_s
```

Out[11]:

```
Float64Index([1230771600.0, 1230775200.0, 1230778800.0, 1230782400.0,
              1230786000.0, 1230789600.0, 1230793200.0, 1230796800.0,
              1230800400.0, 1230804000.0,
              ...
              1483193400.0, 1483197000.0, 1483200600.0, 1483204200.0,
              1483207800.0, 1483211400.0, 1483215000.0, 1483218600.0,
              1483222200.0, 1483225800.0],
             dtype='float64', name='Date Time', length=70091)
```

In [12]:

```
df.index
```

Out[12]:

```
DatetimeIndex(['2009-01-01 01:00:00', '2009-01-01 02:00:00',
               '2009-01-01 03:00:00', '2009-01-01 04:00:00',
               '2009-01-01 05:00:00', '2009-01-01 06:00:00',
               '2009-01-01 07:00:00', '2009-01-01 08:00:00',
               '2009-01-01 09:00:00', '2009-01-01 10:00:00',
               ...
               '2016-12-31 14:10:00', '2016-12-31 15:10:00',
               '2016-12-31 16:10:00', '2016-12-31 17:10:00',
               '2016-12-31 18:10:00', '2016-12-31 19:10:00',
               '2016-12-31 20:10:00', '2016-12-31 21:10:00',
               '2016-12-31 22:10:00', '2016-12-31 23:10:00'],
              dtype='datetime64[ns]', name='Date Time', length=70091, freq=None)
```

In [13]:

```
day = 24 * 60 * 60
year = (365.2425) * day
```

In [14]:

```
df['Day sin'] = np.sin(timestamp_s * (2 * np.pi / day))
df['Day cos'] = np.cos(timestamp_s * (2 * np.pi / day))
df['Year sin'] = np.sin(timestamp_s * (2 * np.pi / year))
df['Year cos'] = np.cos(timestamp_s * (2 * np.pi / year))
```

In [15]:

```
df.corr()
```

Out[15]:

	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H2OC (mmol/mol)	rho (g/m**3)	Day sin
p (mbar)	1.000000	-0.045296	-0.124643	-0.066698	-0.018363	-0.031455	-0.054353	-0.003283	-0.069749	-0.069792	0.307583	0.024335
T (degC)	-0.045296	1.000000	0.996826	0.895706	-0.572593	0.951080	0.867691	0.761672	0.866770	0.867195	-0.963404	-0.209414
Tpot (K)	-0.124643	0.996826	1.000000	0.894909	-0.567306	0.947259	0.866228	0.756886	0.866553	0.866978	-0.981342	-0.209994
Tdew (degC)	-0.066698	0.895706	0.894909	1.000000	-0.156834	0.799182	0.968361	0.435689	0.967614	0.968061	-0.885231	-0.042241
rh (%)	-0.018363	-0.572593	-0.567306	-0.156834	1.000000	-0.616019	-0.151704	-0.843768	-0.151049	-0.151181	0.514461	0.393867
VPmax (mbar)	-0.031455	0.951080	0.947259	0.799182	-0.616019	1.000000	0.824758	0.875639	0.824349	0.824386	-0.901488	-0.230615
VPact (mbar)	-0.054353	0.867691	0.866228	0.968361	-0.151704	0.824758	1.000000	0.449080	0.999851	0.999856	-0.850271	-0.040985
VPdef (mbar)	-0.003283	0.761672	0.756886	0.435689	-0.843768	0.875639	0.449080	1.000000	0.448561	0.448615	-0.698195	-0.329381
sh (g/kg)	-0.069749	0.866770	0.866553	0.967614	-0.151049	0.824349	0.999851	0.448561	1.000000	0.999997	-0.853354	-0.041465
H2OC (mmol/mol)	-0.069792	0.867195	0.866978	0.968061	-0.151181	0.824386	0.999856	0.448615	0.999997	1.000000	-0.853801	-0.041463
rho (g/m3)**	0.307583	-0.963404	-0.981342	-0.885231	0.514461	-0.901488	-0.850271	-0.698195	-0.853354	-0.853801	1.000000	0.195175
Day sin	0.024335	-0.209414	-0.209994	-0.042241	0.393867	-0.230615	-0.040985	-0.329381	-0.041465	-0.041463	0.195175	1.000000
Day cos	0.004076	-0.158552	-0.157833	-0.021313	0.312590	-0.185214	-0.023096	-0.272917	-0.023153	-0.023142	0.142375	0.000072
Year sin	-0.056380	-0.142263	-0.136772	-0.216200	-0.088733	-0.131800	-0.214678	-0.024901	-0.213249	-0.213366	0.125191	-0.000037
Year cos	0.019768	-0.462367	-0.460871	-0.430777	0.242826	-0.433101	-0.439461	-0.308988	-0.439367	-0.439501	0.445831	-0.000011

In [16]:

```
#Splitting data
nb_elts = len(df)
train_df = df[:int(nb_elts * .7)]
val_df = df[int(nb_elts * .7):int(nb_elts *.9)]
test_df = df[int(nb_elts * .9):]
```

In [17]:

```
#Normalize data
train_mean = train_df.mean()
train_std = train_df.std()
```

In [18]:

```
train_df = (train_df - train_mean) / train_std
val_df = (val_df - train_mean) / train_std
test_df = (test_df - train_mean) / train_std
```

In [19]:

```
#Create datasets
def create_dataset(df, input_width:int=24, offset:int=0, predict_column:str='T (degC)'):
  x = []
  y = []
  data_x = df.to_numpy()
  data_y = df[predict_column].to_numpy()

  for i in range(input_width, len(data_x) - offset ):
    x.append(data_x[i - input_width:i, :])
    y.append(data_y[i + offset])

  x = np.array(x)
  y = np.array(y)

  return x, y.reshape(-1, 1)
```

In [20]:

```
train_ds = create_dataset(train_df)
val_ds = create_dataset(val_df)
test_ds = create_dataset(test_df)
```

In [21]:

```
train_ds[0].shape
```

Out[21]:

```
(49039, 24, 15)
```

In [22]:

```
#Create model
model = models.Sequential()
model.add(layers.LSTM(32, return_sequences=True, input_shape=train_ds[0].shape[1:]))
model.add(layers.Dense(units=1))
```

```
2022-04-23 19:40:08.085105: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-04-23 19:40:08.130964: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-04-23 19:40:08.131687: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:939] successful NUMA node read
from SysFS had negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-04-23 19:40:08.132473: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1900] Ignoring visible gpu device
(device: 0, name: Quadro K1000M, pci bus id: 0000:01:00.0, compute capability: 3.0) with Cuda compute capability
3.0. The minimum required Cuda capability is 3.5.
2022-04-23 19:40:08.133438: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimi
zed with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critica
l operations:  SSE4.1 SSE4.2 AVX
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [23]:

```
#Train model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
model.fit(x=train_ds[0], y=train_ds[1], validation_data=(val_ds[0], val_ds[1]), epochs=10)
```

```
Epoch 1/10
1533/1533 [==============================] - 27s 17ms/step - loss: 0.1175 - accuracy: 0.0000e+00 - val_loss: 0.09
43 - val_accuracy: 0.0000e+00
Epoch 2/10
1533/1533 [==============================] - 31s 20ms/step - loss: 0.0868 - accuracy: 0.0000e+00 - val_loss: 0.08
81 - val_accuracy: 0.0000e+00
Epoch 3/10
1533/1533 [==============================] - 31s 20ms/step - loss: 0.0832 - accuracy: 0.0000e+00 - val_loss: 0.09
09 - val_accuracy: 0.0000e+00
Epoch 4/10
1533/1533 [==============================] - 31s 20ms/step - loss: 0.0815 - accuracy: 0.0000e+00 - val_loss: 0.08
70 - val_accuracy: 0.0000e+00
Epoch 5/10
1533/1533 [==============================] - 31s 20ms/step - loss: 0.0805 - accuracy: 0.0000e+00 - val_loss: 0.08
47 - val_accuracy: 0.0000e+00
Epoch 6/10
1533/1533 [==============================] - 30s 20ms/step - loss: 0.0795 - accuracy: 0.0000e+00 - val_loss: 0.08
49 - val_accuracy: 0.0000e+00
Epoch 7/10
1533/1533 [==============================] - 30s 20ms/step - loss: 0.0786 - accuracy: 0.0000e+00 - val_loss: 0.08
66 - val_accuracy: 0.0000e+00
Epoch 8/10
1533/1533 [==============================] - 30s 20ms/step - loss: 0.0779 - accuracy: 0.0000e+00 - val_loss: 0.08
80 - val_accuracy: 0.0000e+00
Epoch 9/10
1533/1533 [==============================] - 30s 20ms/step - loss: 0.0771 - accuracy: 0.0000e+00 - val_loss: 0.08
49 - val_accuracy: 0.0000e+00
Epoch 10/10
1533/1533 [==============================] - 30s 19ms/step - loss: 0.0765 - accuracy: 0.0000e+00 - val_loss: 0.08
55 - val_accuracy: 0.0000e+00
```

Out[23]:

```
<keras.callbacks.History at 0x7fc4d25b3f40>
```

In [ ]:

```
# predict data
x, y = test_ds
```

In [25]:

```
y_pred = model.predict(x)
```

In [26]:

```
y_pred.shape
```

Out[26]:

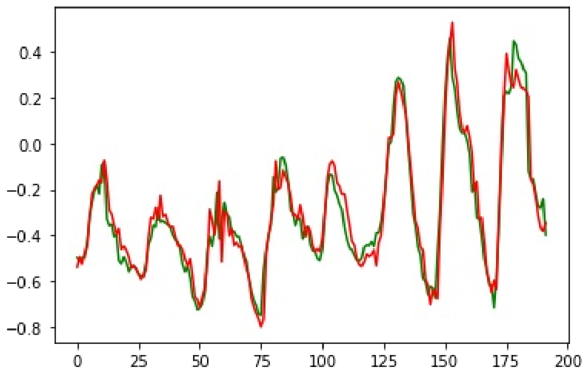
```
(6986, 24, 1)
```

In [27]:

```
#Plot result
fig, ax = plt.subplots()
i = 200
ax.plot(y[i:i+96*2,0], c='g')
ax.plot(y_pred[i:i+96*2,-1,0], c='r')
```

```
[<matplotlib.lines.Line2D at 0x7fc4ce6e99d0>]
```

```
Out[27]: [<matplotlib.lines.Line2D at 0x7fc4ce6e99d0>]
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js