vwdhspf48

April 8, 2024

Assignment 2

GROUP 13 - Phani Varshitha, Durga Chowdary

Importing all the modules

[123]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
```

Reading the csv file

[124]:
```
df = pd.read_csv("C:/Users/varshitha/Desktop/jena_climate_2009_2016.csv")
```

Data exploration

[125]:
```
df.head()
```

[125]:
```
             Date Time  p (mbar)  T (degC)  Tpot (K)  Tdew (degC)  rh (%)  \
0  01.01.2009 00:10:00    996.52     -8.02    265.40        -8.90    93.3
1  01.01.2009 00:20:00    996.57     -8.41    265.01        -9.28    93.4
2  01.01.2009 00:30:00    996.53     -8.51    264.91        -9.31    93.9
3  01.01.2009 00:40:00    996.51     -8.31    265.12        -9.07    94.2
4  01.01.2009 00:50:00    996.51     -8.27    265.15        -9.04    94.1

   VPmax (mbar)  VPact (mbar)  VPdef (mbar)  sh (g/kg)  H2OC (mmol/mol)  \
0          3.33          3.11          0.22       1.94             3.12
1          3.23          3.02          0.21       1.89             3.03
2          3.21          3.01          0.20       1.88             3.02
3          3.26          3.07          0.19       1.92             3.08
4          3.27          3.08          0.19       1.92             3.09

   rho (g/m**3)  wv (m/s)  max. wv (m/s)  wd (deg)
0       1307.75      1.03           1.75     152.3
1       1309.80      0.72           1.50     136.1
2       1310.24      0.19           0.63     171.6
3       1309.19      0.34           0.50     198.0
4       1309.00      0.32           0.63     214.3
```

Checking shape of our shape

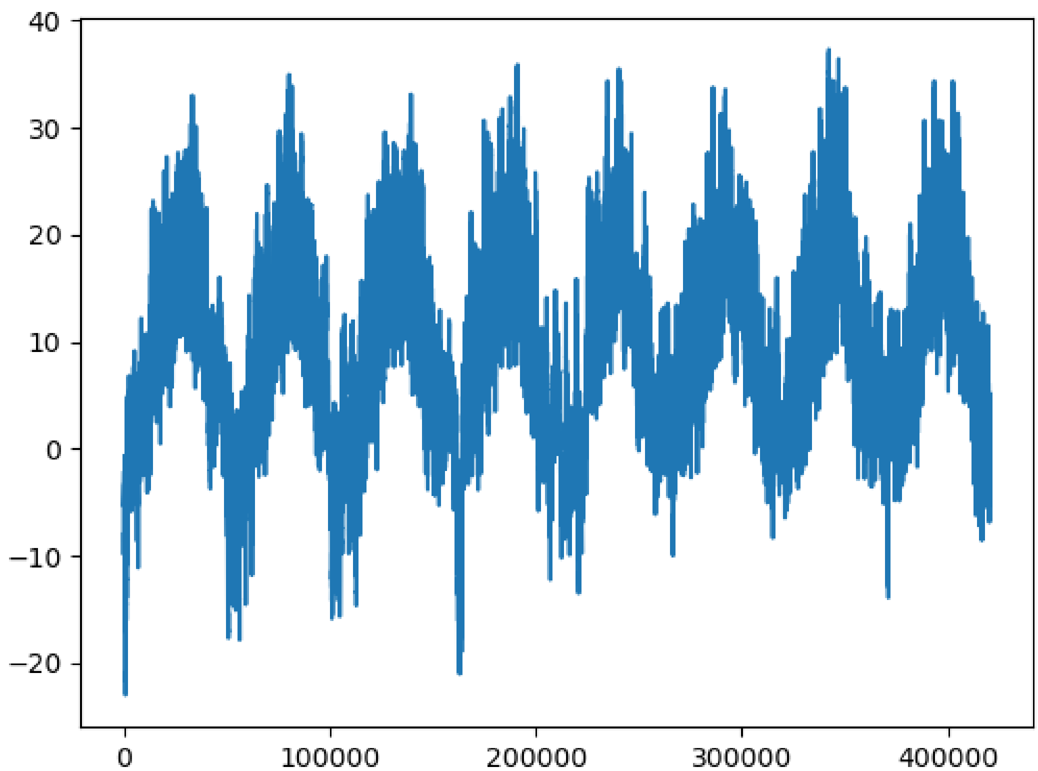
[126]:
```
df.shape
```

[126]:
```
(420451, 15)
```

plotting temperatures

[128]:
```
plt.plot(range(420451),df.iloc[:,2])
```

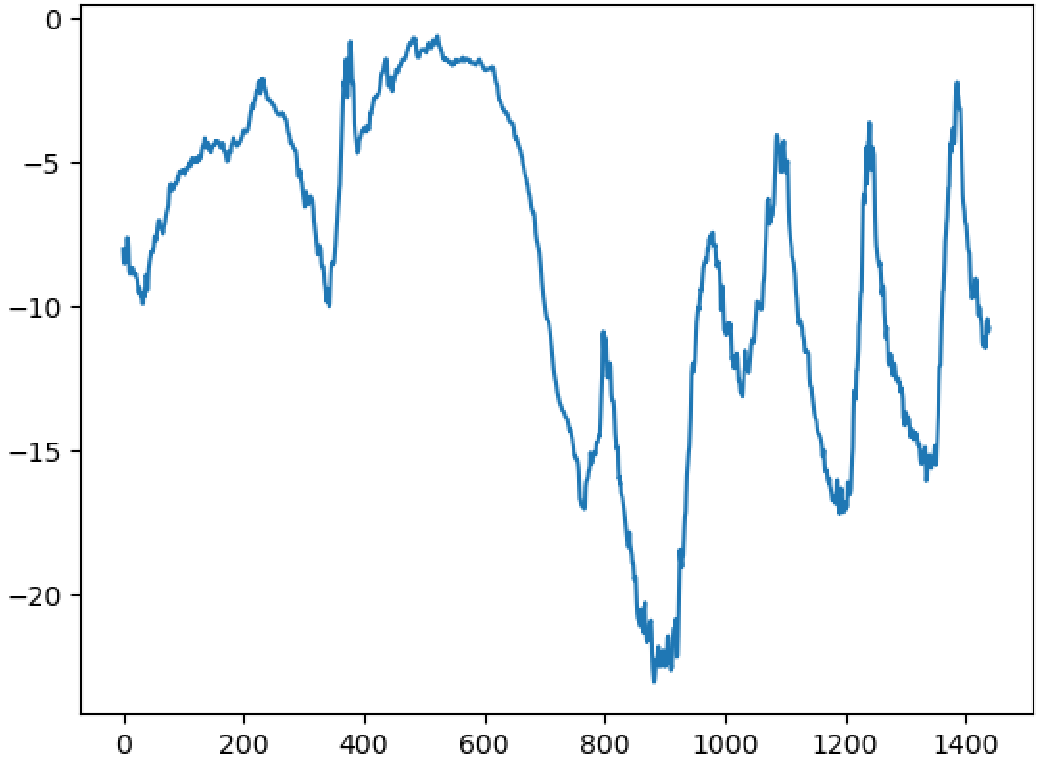
[128]:
```
[<matplotlib.lines.Line2D at 0x2e812aa3f50>]
```



Plotting the temperatures for only first 10 days

[129]:
```
plt.plot(range(1440), df.iloc[0:1440,2])
```

[129]:
```
[<matplotlib.lines.Line2D at 0x2e812ae3f50>]
```



Printing the size of training,validation,test samples that we will be using

[130]:
```
n_train = int(0.5*len(df))
n_val = int(0.25*len(df))
n_test = int(n_train-n_val)
print("Train samples : ",n_train)
print("Validation samples : ",n_val)
print("Test samples : ",n_test)
```

```
Train samples :  210225
Validation samples :  105112
Test samples :  105113
```

Data preparation

[131]:
```
# Storing standard devidation and mean for furthur use  and normalizing data:
dfs = df.drop('Date Time',axis=1).to_numpy()
mean = dfs[:n_train].mean(axis=0)
dfs -=mean
std = dfs[:n_train].std(axis=0)
dfs /= std
print(std[1])
print(mean[1])
```

```
8.770983608349352
8.825903294089667
```

[132]:
```python
temperature = dfs[:,1]
```

Model Construction

Diving data into training, validation and test dataset

[133]:
```python
sampling_rate = 6
sequence_length = 120
delay = sampling_rate * (sequence_length + 24 - 1)
batch_size = 256
```

Converting data frame to array, discarding date-time and converting values to float

[134]:
```python
dfs=dfs.astype('float32')
temperature = temperature.astype('float32')
```

[135]:
```python
from tensorflow import keras
Train = keras.utils.timeseries_dataset_from_array(
    dfs[:-delay],
    targets = temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length = sequence_length,
    batch_size=batch_size,
    start_index=0,
    shuffle=True,
    end_index=n_train
    )
Validation = keras.utils.timeseries_dataset_from_array(
    dfs[:-delay],
    targets = temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length = sequence_length,
    batch_size=batch_size,
    start_index=n_train,
    shuffle=True,
    end_index=n_train+n_val
    )
Test = keras.utils.timeseries_dataset_from_array(
    dfs[:-delay],
    targets = temperature[delay:],
    sampling_rate=sampling_rate,
    sequence_length = sequence_length,
    batch_size=batch_size,
    start_index=n_train+n_val,
    shuffle=True
```

```
    )
```

Inspecting the output of our Train dataset :

[136]:
```
for samples,targets in Train :
    print("Sample shape : ",samples.shape)
    print("Target shape :",targets.shape)
    break
```

```
Sample shape :  (256, 120, 14)
Target shape : (256,)
```

Making a simple dense network model to check performance :

[140]:
```
from tensorflow import keras
from tensorflow.keras import layers
inputs = keras.Input(shape=(sequence_length, dfs.shape[-1]))
x = layers.Reshape((sequence_length * dfs.shape[-1],))(inputs)
x = layers.Flatten()(x)
x = layers.Dense(16, activation="relu")(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("C:/Users/varshitha/Desktop/jena_dense/
 ↪model_1.keras",
                                    save_best_only=True)
]

model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(Train,
                    epochs=10,
                    validation_data=Validation,
                    callbacks=callbacks)
model = keras.models.load_model("C:/Users/varshitha/Desktop/jena_dense/model_1.
 ↪keras")
print(f"Test MAE: {model.evaluate(Test)[1]:.2f}")
```

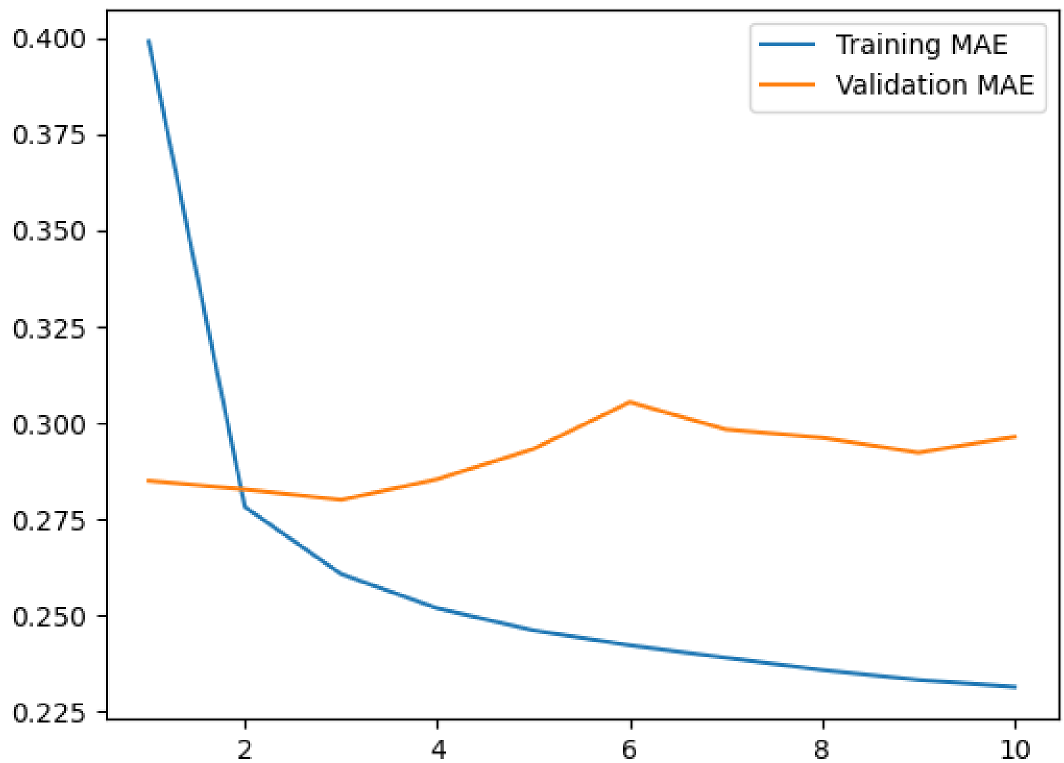
```
Epoch 1/10
819/819             93s 112ms/step -
loss: 0.5493 - mae: 0.5064 - val_loss: 0.1306 - val_mae: 0.2849
Epoch 2/10
819/819             99s 120ms/step -
loss: 0.1323 - mae: 0.2852 - val_loss: 0.1285 - val_mae: 0.2826
Epoch 3/10
819/819             101s 123ms/step -
loss: 0.1123 - mae: 0.2636 - val_loss: 0.1271 - val_mae: 0.2799
Epoch 4/10
```

```
819/819              95s 115ms/step -
loss: 0.1035 - mae: 0.2532 - val_loss: 0.1304 - val_mae: 0.2853
Epoch 5/10
819/819              98s 119ms/step -
loss: 0.0975 - mae: 0.2462 - val_loss: 0.1379 - val_mae: 0.2931
Epoch 6/10
819/819              95s 115ms/step -
loss: 0.0949 - mae: 0.2429 - val_loss: 0.1508 - val_mae: 0.3053
Epoch 7/10
819/819              155s 129ms/step -
loss: 0.0921 - mae: 0.2395 - val_loss: 0.1416 - val_mae: 0.2982
Epoch 8/10
819/819              106s 129ms/step -
loss: 0.0895 - mae: 0.2359 - val_loss: 0.1407 - val_mae: 0.2960
Epoch 9/10
819/819              106s 129ms/step -
loss: 0.0874 - mae: 0.2330 - val_loss: 0.1371 - val_mae: 0.2921
Epoch 10/10
819/819              100s 121ms/step -
loss: 0.0860 - mae: 0.2315 - val_loss: 0.1407 - val_mae: 0.2963
405/405              36s 85ms/step -
loss: 0.1497 - mae: 0.3041
Test MAE: 0.30
```

Plotting the results:

[141]:
```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[141]:
```
<matplotlib.legend.Legend at 0x2e873854d10>
```



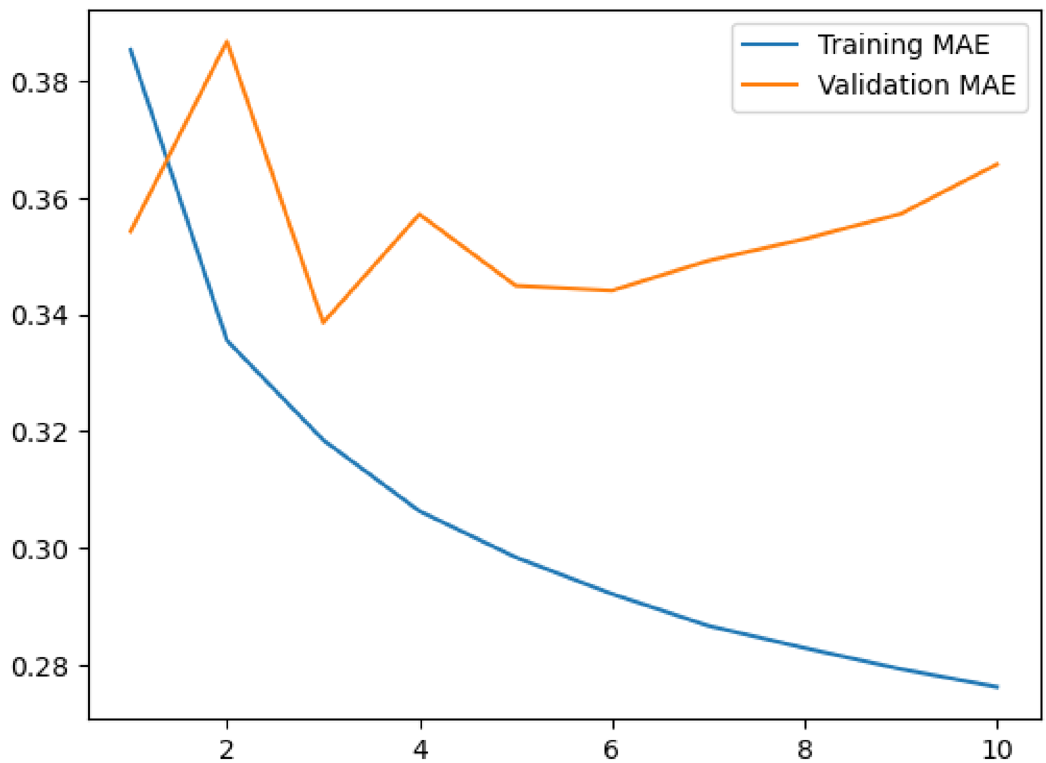
[142]:
```
inputs = keras.Input(shape=(sequence_length, dfs.shape[-1]))
x = layers.Conv1D(8, 24, activation="relu")(inputs)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 12, activation="relu")(x)
x = layers.MaxPooling1D(2)(x)
x = layers.Conv1D(8, 6, activation="relu")(x)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
callbacks = [
            keras.callbacks.ModelCheckpoint("C:/Users/varshitha/Desktop/
 ↪jena_dense/model_2.keras",
                                            save_best_only=True)
        ]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(Train,
                            epochs=10,
                            validation_data=Validation,
                            callbacks=callbacks)
model = keras.models.load_model("C:/Users/varshitha/Desktop/jena_dense/model_2.
 ↪keras")
print(f"Test MAE: {model.evaluate(Test)[1]:.2f}")
```

```
Epoch 1/10
819/819             119s 142ms/step -
loss: 0.2960 - mae: 0.4260 - val_loss: 0.2021 - val_mae: 0.3543
Epoch 2/10
819/819             121s 147ms/step -
loss: 0.1847 - mae: 0.3409 - val_loss: 0.2445 - val_mae: 0.3868
Epoch 3/10
819/819             124s 151ms/step -
loss: 0.1661 - mae: 0.3222 - val_loss: 0.1840 - val_mae: 0.3386
Epoch 4/10
819/819             138s 143ms/step -
loss: 0.1528 - mae: 0.3088 - val_loss: 0.2021 - val_mae: 0.3572
Epoch 5/10
819/819             132s 160ms/step -
loss: 0.1444 - mae: 0.3002 - val_loss: 0.1883 - val_mae: 0.3449
Epoch 6/10
819/819             126s 153ms/step -
loss: 0.1374 - mae: 0.2923 - val_loss: 0.1910 - val_mae: 0.3441
Epoch 7/10
819/819             130s 158ms/step -
loss: 0.1326 - mae: 0.2870 - val_loss: 0.1980 - val_mae: 0.3492
Epoch 8/10
819/819             119s 145ms/step -
loss: 0.1295 - mae: 0.2838 - val_loss: 0.1994 - val_mae: 0.3529
Epoch 9/10
819/819             149s 152ms/step -
loss: 0.1257 - mae: 0.2796 - val_loss: 0.2055 - val_mae: 0.3573
Epoch 10/10
819/819             118s 143ms/step -
loss: 0.1226 - mae: 0.2763 - val_loss: 0.2152 - val_mae: 0.3658
405/405             37s 90ms/step -
loss: 0.2133 - mae: 0.3677
Test MAE: 0.37
```

Plotting the results. :

[143]:
```
loss = history.history["mae"]
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[143]:
```
<matplotlib.legend.Legend at 0x2e87672cd10>
```



Constructing a simple recurrent neural network using LSTM model :

[144]:
```
inputs = keras.Input(shape=(sequence_length,dfs.shape[-1]))
x = layers.LSTM(16)(inputs)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs,outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("C:/Users/varshitha/Desktop/jena_dense/
 ↪model_3.keras",
                                    save_best_only=True)
]

model.compile(optimizer="rmsprop",loss = "mse",metrics=["mae"])

history=model.fit(Train,
                 epochs=10,
                 validation_data=Validation,
                 callbacks=callbacks)

model=keras.models.load_model("C:/Users/varshitha/Desktop/jena_dense/model_3.
 ↪keras")
```

```
Epoch 1/10
819/819                    145s 171ms/step -
loss: 0.1808 - mae: 0.3230 - val_loss: 0.1167 - val_mae: 0.2646
Epoch 2/10
819/819                    178s 214ms/step -
loss: 0.1232 - mae: 0.2747 - val_loss: 0.1159 - val_mae: 0.2625
Epoch 3/10
819/819                    132s 161ms/step -
loss: 0.1144 - mae: 0.2652 - val_loss: 0.1214 - val_mae: 0.2697
Epoch 4/10
819/819                    125s 152ms/step -
loss: 0.1075 - mae: 0.2570 - val_loss: 0.1248 - val_mae: 0.2732
Epoch 5/10
819/819                    159s 170ms/step -
loss: 0.1020 - mae: 0.2500 - val_loss: 0.1250 - val_mae: 0.2741
Epoch 6/10
819/819                    147s 179ms/step -
loss: 0.0979 - mae: 0.2448 - val_loss: 0.1272 - val_mae: 0.2768
Epoch 7/10
819/819                    141s 171ms/step -
loss: 0.0946 - mae: 0.2406 - val_loss: 0.1298 - val_mae: 0.2795
Epoch 8/10
819/819                    167s 203ms/step -
loss: 0.0917 - mae: 0.2369 - val_loss: 0.1318 - val_mae: 0.2828
Epoch 9/10
819/819                    148s 180ms/step -
loss: 0.0891 - mae: 0.2335 - val_loss: 0.1343 - val_mae: 0.2851
Epoch 10/10
819/819                    181s 154ms/step -
loss: 0.0866 - mae: 0.2303 - val_loss: 0.1354 - val_mae: 0.2865
```

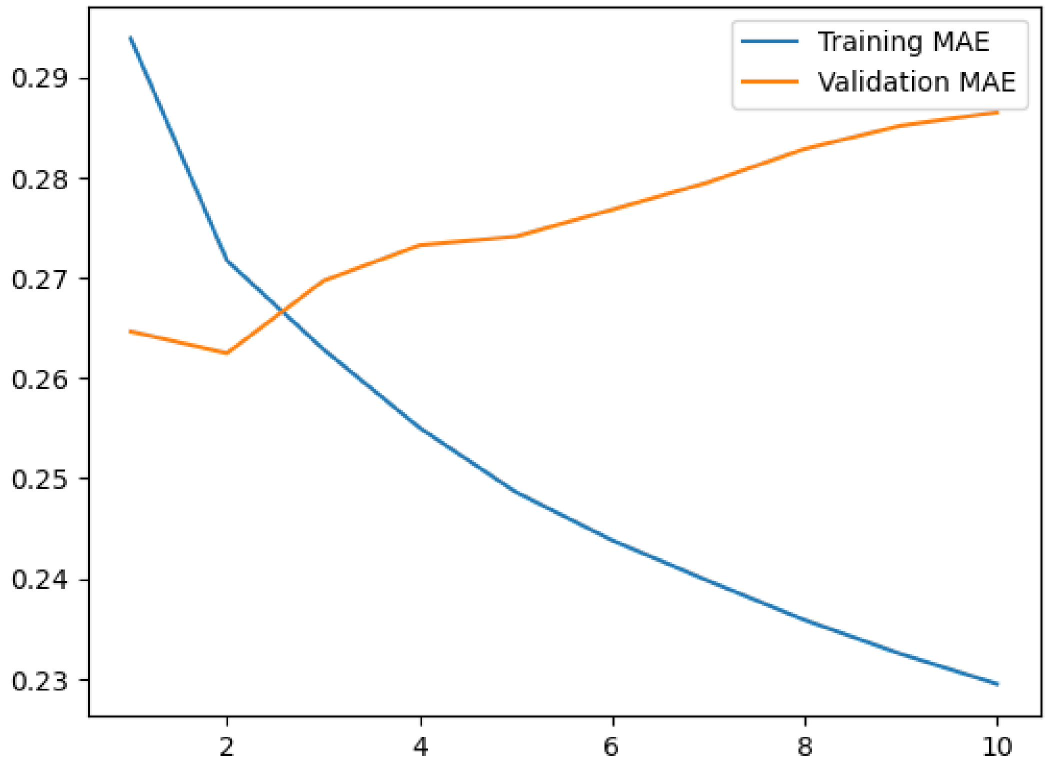
[145]:
```
print("Test MAE: ",round(model.evaluate(Test)[1],2))
```

```
405/405                    41s 97ms/step -
loss: 0.1308 - mae: 0.2809
Test MAE:  0.28
```

Plotting results of LSTM model :

[146]:
```
loss = history.history['mae']
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
plt.legend()
```

[146]: <matplotlib.legend.Legend at 0x2e887f00e90>



Hence, we can conclude that recurrent neural networks work best on a time - series problem

Improving Forecasting

Now we will use different model architecures to acheive the best possible accuracy in our model.

Constructing a RNN with only one layer of GRU but increased units to check initial accuracy.

[147]:
```
inputs = keras.Input(shape=(sequence_length, dfs.shape[-1]))
x = layers.GRU(32, recurrent_dropout=0.25)(inputs)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1)(x)
model = keras.Model(inputs, outputs)
callbacks = [
            keras.callbacks.ModelCheckpoint("C:/Users/varshitha/Desktop/
 ↪jena_dense/model_4.keras",
                                            save_best_only=True)
        ]
model.compile(optimizer="rmsprop", loss="mse", metrics=["mae"])
history = model.fit(Train,
                    epochs=10,
                    validation_data=Validation,
                    callbacks=callbacks)
```

```
model = keras.models.load_model("C:/Users/varshitha/Desktop/jena_dense/model_4.
↪keras")
```

```
Epoch 1/10
819/819              260s 309ms/step -
loss: 0.3526 - mae: 0.4251 - val_loss: 0.1180 - val_mae: 0.2676
Epoch 2/10
819/819              208s 253ms/step -
loss: 0.1646 - mae: 0.3165 - val_loss: 0.1136 - val_mae: 0.2630
Epoch 3/10
819/819              211s 257ms/step -
loss: 0.1594 - mae: 0.3109 - val_loss: 0.1135 - val_mae: 0.2622
Epoch 4/10
819/819              206s 251ms/step -
loss: 0.1566 - mae: 0.3086 - val_loss: 0.1144 - val_mae: 0.2630
Epoch 5/10
819/819              213s 259ms/step -
loss: 0.1540 - mae: 0.3062 - val_loss: 0.1125 - val_mae: 0.2618
Epoch 6/10
819/819              212s 258ms/step -
loss: 0.1498 - mae: 0.3020 - val_loss: 0.1152 - val_mae: 0.2643
Epoch 7/10
819/819              209s 254ms/step -
loss: 0.1459 - mae: 0.2985 - val_loss: 0.1136 - val_mae: 0.2623
Epoch 8/10
819/819              227s 277ms/step -
loss: 0.1433 - mae: 0.2954 - val_loss: 0.1109 - val_mae: 0.2575
Epoch 9/10
819/819              134s 163ms/step -
loss: 0.1401 - mae: 0.2919 - val_loss: 0.1189 - val_mae: 0.2683
Epoch 10/10
819/819              117s 143ms/step -
loss: 0.1376 - mae: 0.2895 - val_loss: 0.1119 - val_mae: 0.2591
```

[148]:
```
model.evaluate(Test)
```

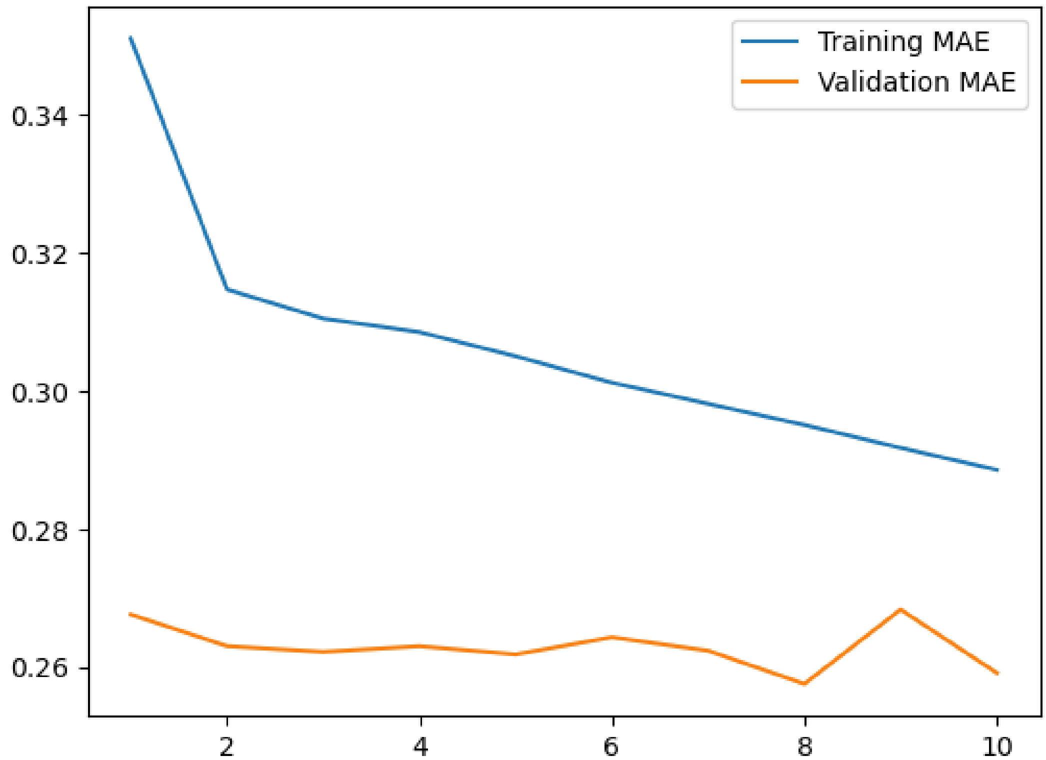
```
405/405              18s 42ms/step -
loss: 0.1257 - mae: 0.2767
```

[148]: [0.12531542778015137, 0.27656733989715576]

[149]:
```
loss = history.history['mae']
val_loss = history.history["val_mae"]
epochs = range(1, len(loss) + 1)
plt.figure()
plt.plot(epochs, loss,label="Training MAE")
plt.plot(epochs,val_loss,label="Validation MAE")
```

```
plt.legend()
```

[149]: <matplotlib.legend.Legend at 0x2e88c5c8e90>



Now, we will take measures to improve our accuracy.

We are using following points to increase our model's efficiency :

Adjusting the number of units in each recurrent layer in the stacked setup. Using layer_lstm() instead of layer_gru(). Using a combination of 1d_convnets and RNN.

Final Model Construction

[ ]:
```
inputs = keras.Input(shape=(sequence_length,dfs.shape[-1]))
# Using incresased units and applying lstm instead of gru
x=layers.LSTM(32,recurrent_dropout=0.5,return_sequences=True)(inputs)
# Adding more layers to turn it into a stacked model
x=layers.LSTM(32,recurrent_dropout=0.5,return_sequences=True)(x)
# Applying 1D convolution
x = layers.Conv1D(8, 24, activation="relu")(x)
x = layers.MaxPooling1D(2)(x)
x = layers.GlobalAveragePooling1D()(x)
# We are also using dropout layer to regularize our results
x=layers.Dropout(0.5)(x)
```

```
outputs=layers.Dense(1)(x)
model=keras.Model(inputs,outputs)

callbacks = [
    keras.callbacks.ModelCheckpoint("C:/Users/varshitha/Desktop/jena_dense/
 ↪model_5.keras",
                                    save_best_only = True)
]

model.compile(optimizer="rmsprop",loss="mse",metrics=["mae"])

history=model.fit(Train,
                 epochs=10,
                 validation_data=Validation,
                 callbacks=callbacks)
```

```
Epoch 1/10
819/819              274s 327ms/step -
loss: 0.4720 - mae: 0.5329 - val_loss: 0.2796 - val_mae: 0.4157
Epoch 2/10
819/819              277s 338ms/step -
loss: 0.3686 - mae: 0.4654 - val_loss: 0.2746 - val_mae: 0.4094
Epoch 3/10
819/819              274s 334ms/step -
loss: 0.3260 - mae: 0.4317 - val_loss: 0.2843 - val_mae: 0.4155
Epoch 4/10
819/819              301s 368ms/step -
loss: 0.3068 - mae: 0.4148 - val_loss: 0.2797 - val_mae: 0.4167
Epoch 5/10
819/819              306s 373ms/step -
loss: 0.2956 - mae: 0.4049 - val_loss: 0.2819 - val_mae: 0.4166
Epoch 6/10
819/819              296s 361ms/step -
loss: 0.2907 - mae: 0.4005 - val_loss: 0.3011 - val_mae: 0.4323
Epoch 7/10
819/819              304s 371ms/step -
loss: 0.2842 - mae: 0.3945 - val_loss: 0.2914 - val_mae: 0.4240
Epoch 8/10
819/819              301s 367ms/step -
loss: 0.2791 - mae: 0.3898 - val_loss: 0.2986 - val_mae: 0.4304
Epoch 9/10
819/819              302s 369ms/step -
loss: 0.2750 - mae: 0.3863 - val_loss: 0.3180 - val_mae: 0.4463
Epoch 10/10
819/819              0s 313ms/step -
loss: 0.2730 - mae: 0.3843
```

[ ]:

```
model=keras.models.load_model("C:/Users/varshitha/Desktop/jena_dense/model_5.
↪keras")
model.evaluate(Test)[1]
```