Arun Kumar Kudurumalla

Advanced Machine Learning (Time Series Data)

Assignment - 3

Importing the required libraries

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import datetime

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('C:/Users/Arun/Downloads/arun_AML_RNN'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you create a ver
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
C:/Users/YeswantH/Downloads/arun_AML_RNN\DailyDelhiClimateTest.csv
C:/Users/YeswantH/Downloads/arun_AML_RNN\DailyDelhiClimateTrain.csv
```

Reading the data into the train and test data frames

In [2]:
```
train_data = pd.read_csv('C:/Users/Arun/Downloads/arun_AML_RNN/DailyDelhiClimateTrain.csv')
test_data = pd.read_csv('C:/Users/Arun/Downloads/arun_AML_RNN/DailyDelhiClimateTest.csv')
```

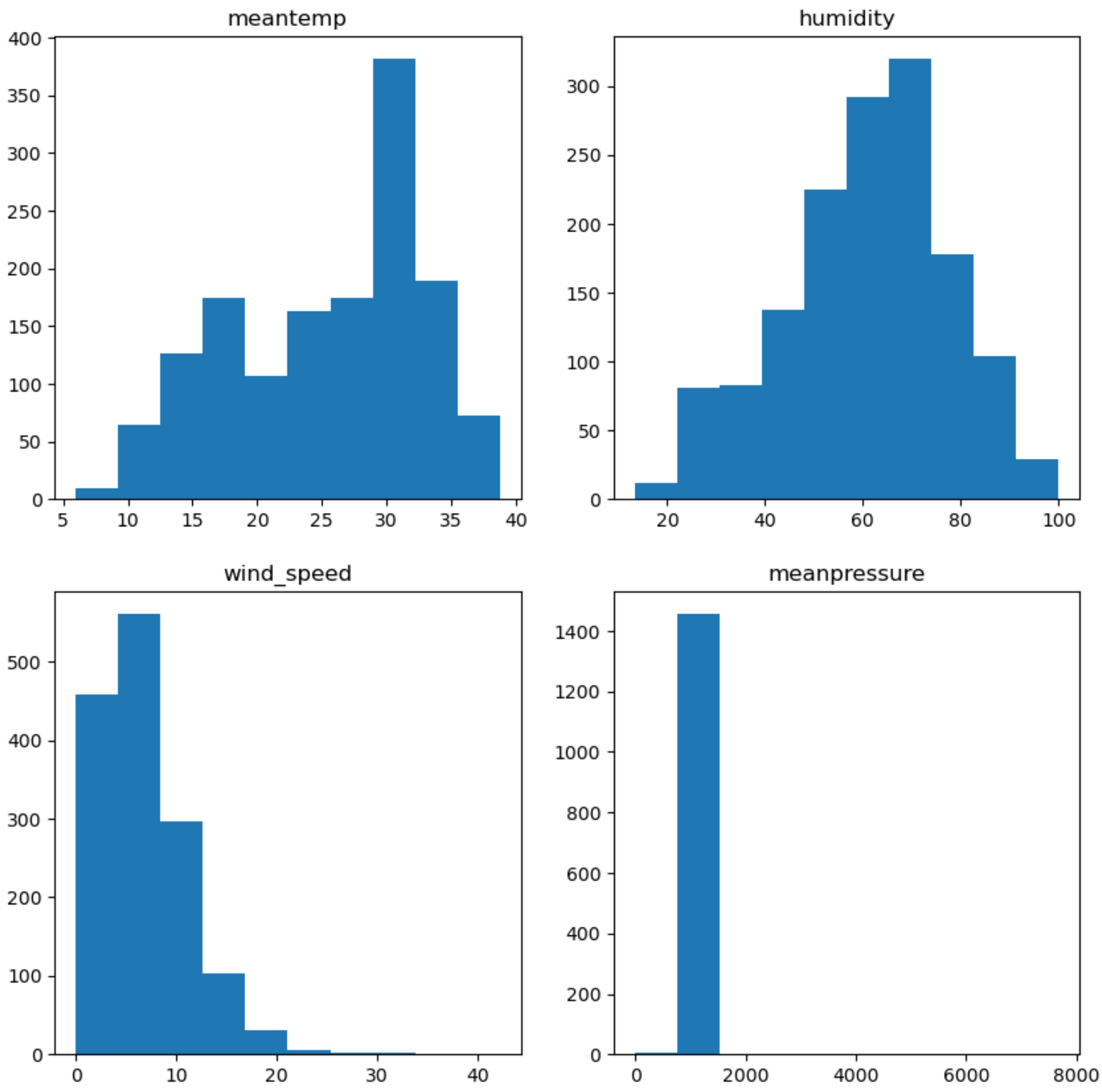
Converting the data into regular format

In [3]:
```
train_data['date'] = pd.to_datetime(train_data['date'])
test_data['date'] = pd.to_datetime(test_data['date'])
```

Visualizing the 4 diffferent categories of time-series data in the dataset

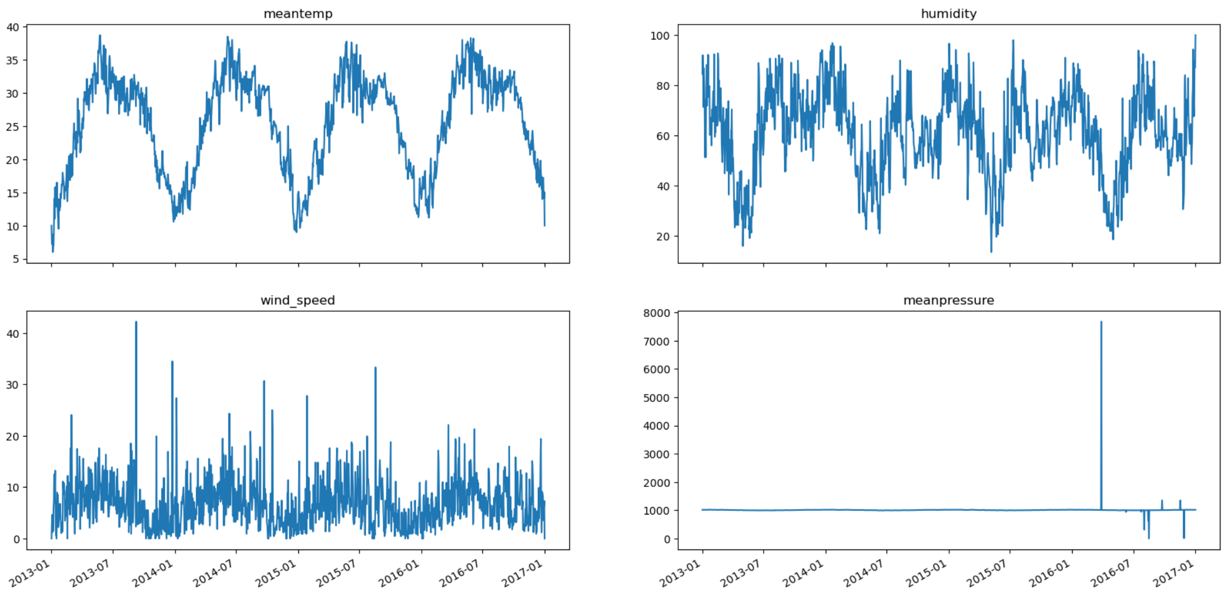
In [4]:
```
def Visualize(kind='hist',figsize=(10,10)):
    def Col():
        cols = ['meantemp','humidity','wind_speed','meanpressure']
        for c in cols:
            yield c

    fig,axes = plt.subplots(nrows=2,ncols=2,figsize=figsize)
    col = Col()
    for i in range(2):
        for j in range(2):
            curr = next(col)
            if kind == 'hist':
                axes[i,j].hist(train_data[curr])
            elif kind == 'plot':
                axes[i,j].plot(train_data['date'],train_data[curr])
                plt.gcf().autofmt_xdate()
            axes[i,j].set_title(curr)
    if kind=='hist':
        plt.savefig('data_hist.png')
    else:
        plt.savefig('data_plot.png')
```

In [5]:
```
Visualize('hist')
```



Plotting the data in a wave form

In [6]:
```
Visualize('plot',(20,10))
```



Pre-processing of the data

In [7]:
```
def process_data(data):
    x,y = [],[]
    switch = False

    if len(data)%2 == 1:
        last = False
    else:
        last = True

    for i in range(len(data)):
        if i == len(data)-1:
            if not last:
                break
        if switch:
            y.append(data[i])
            switch = False
        else:
            x.append(data[i])
            switch = True

    def reshape(d):
        d = np.array(d)
        d = np.reshape(d,(d.shape[0],1,1))
        return d
    return (reshape(x),np.array(y))
```

Applying the pre-processing function on train and test data

In [8]:
```
x_train_meantemp,y_train_meantemp = process_data(train_data.meantemp)
x_test_meantemp,y_test_meantemp = process_data(test_data.meantemp)
```

In [9]:
```
x_train_humidity,y_train_humidity = process_data(train_data.humidity)
x_test_humidity,y_test_humidity = process_data(test_data.humidity)
```

In [10]:
```
x_train_wind_speed,y_train_wind_speed = process_data(train_data.wind_speed)
x_test_wind_speed,y_test_wind_speed = process_data(test_data.wind_speed)
```

In [11]:
```
x_train_meanpressure,y_train_meanpressure = process_data(train_data.meanpressure)
x_test_meanpressure,y_test_meanpressure = process_data(test_data.meanpressure)
```

Models building and architecture

In [12]:
```
model_meantemp = keras.Sequential([
    keras.layers.GRU(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='relu'),
    keras.layers.Dense(32,activation='relu'),
    keras.layers.Dense(1)
])
```

In [13]:
```
model_humidity = keras.Sequential([
    keras.layers.GRU(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='relu'),
    keras.layers.Dense(32,activation='relu'),
    keras.layers.Dense(1)
])
```

In [14]:
```
model_wind_speed = keras.Sequential([
    keras.layers.GRU(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='relu'),
    keras.layers.Dense(32,activation='relu'),
    keras.layers.Dense(64,activation='relu'),
    keras.layers.Dense(1)
])
```

In [15]:
```
model_meanpressure = keras.Sequential([
    keras.layers.GRU(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='tanh'),
    keras.layers.Dense(32,activation='tanh'),
    keras.layers.Dense(64,activation='tanh'),
    keras.layers.Dense(1)
])
```

Compiling the models

In [16]:
```
model_meantemp.compile(loss='mse',optimizer='adam')
```

In [17]:
```
model_humidity.compile(loss='mse',optimizer='adam')
```

In [18]:
```
model_wind_speed.compile(loss='mse',optimizer='adam')
```

In [19]:
```
model_meanpressure.compile(loss='mse',optimizer='adam')
```

In [20]:
```
model_meantemp.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 8)                 264

 dense (Dense)               (None, 16)                144

 dense_1 (Dense)             (None, 32)                544

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 985 (3.85 KB)
Trainable params: 985 (3.85 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training the models with the training data and validating them

In [21]:
```
os.makedirs('logs',exist_ok=True)
logdir = os.path.join('logs',datetime.datetime.now().strftime('%Y%m%d-%H%M%S'))
```

In [22]:
```
callback = keras.callbacks.TensorBoard(logdir)
earlyStoping = keras.callbacks.EarlyStopping(monitor='loss',patience=3)
```

In [23]:
```
history_humidity = model_humidity.fit(x_train_humidity,y_train_humidity,epochs=100,verbose=2,batch_size=16,callbacks=[cal
```

```
Epoch 1/100
46/46 - 2s - loss: 3925.4617 - 2s/epoch - 51ms/step
Epoch 2/100
46/46 - 0s - loss: 3721.3130 - 122ms/epoch - 3ms/step
Epoch 3/100
46/46 - 0s - loss: 3197.3564 - 132ms/epoch - 3ms/step
Epoch 4/100
46/46 - 0s - loss: 2331.4836 - 133ms/epoch - 3ms/step
Epoch 5/100
46/46 - 0s - loss: 1376.0094 - 134ms/epoch - 3ms/step
Epoch 6/100
46/46 - 0s - loss: 645.8209 - 133ms/epoch - 3ms/step
Epoch 7/100
46/46 - 0s - loss: 324.4250 - 133ms/epoch - 3ms/step
Epoch 8/100
46/46 - 0s - loss: 225.6918 - 135ms/epoch - 3ms/step
Epoch 9/100
46/46 - 0s - loss: 155.9612 - 134ms/epoch - 3ms/step
Epoch 10/100
46/46 - 0s - loss: 108.8692 - 132ms/epoch - 3ms/step
Epoch 11/100
46/46 - 0s - loss: 87.4220 - 134ms/epoch - 3ms/step
Epoch 12/100
46/46 - 0s - loss: 77.9290 - 149ms/epoch - 3ms/step
Epoch 13/100
46/46 - 0s - loss: 74.5218 - 135ms/epoch - 3ms/step
Epoch 14/100
46/46 - 0s - loss: 71.8999 - 133ms/epoch - 3ms/step
Epoch 15/100
46/46 - 0s - loss: 71.2449 - 133ms/epoch - 3ms/step
Epoch 16/100
46/46 - 0s - loss: 71.4685 - 136ms/epoch - 3ms/step
Epoch 17/100
46/46 - 0s - loss: 67.7038 - 146ms/epoch - 3ms/step
Epoch 18/100
46/46 - 0s - loss: 65.9664 - 135ms/epoch - 3ms/step
Epoch 19/100
46/46 - 0s - loss: 68.2556 - 133ms/epoch - 3ms/step
Epoch 20/100
46/46 - 0s - loss: 69.9513 - 134ms/epoch - 3ms/step
Epoch 21/100
46/46 - 0s - loss: 65.0331 - 133ms/epoch - 3ms/step
Epoch 22/100
46/46 - 0s - loss: 67.5007 - 133ms/epoch - 3ms/step
Epoch 23/100
46/46 - 0s - loss: 67.5476 - 119ms/epoch - 3ms/step
Epoch 24/100
46/46 - 0s - loss: 64.2622 - 131ms/epoch - 3ms/step
Epoch 25/100
46/46 - 0s - loss: 65.3772 - 131ms/epoch - 3ms/step
Epoch 26/100
46/46 - 0s - loss: 67.4478 - 132ms/epoch - 3ms/step
Epoch 27/100
46/46 - 0s - loss: 64.6497 - 137ms/epoch - 3ms/step
```

In [24]:
```
history_meantemp = model_meantemp.fit(x_train_meantemp,y_train_meantemp,epochs=100,verbose=2,batch_size=16,callbacks=[cal
```

```
Epoch 1/100
46/46 - 2s - loss: 650.3699 - 2s/epoch - 51ms/step
Epoch 2/100
46/46 - 0s - loss: 516.8016 - 135ms/epoch - 3ms/step
Epoch 3/100
46/46 - 0s - loss: 291.8326 - 133ms/epoch - 3ms/step
Epoch 4/100
46/46 - 0s - loss: 97.4099 - 124ms/epoch - 3ms/step
Epoch 5/100
46/46 - 0s - loss: 47.1615 - 133ms/epoch - 3ms/step
Epoch 6/100
46/46 - 0s - loss: 29.0423 - 133ms/epoch - 3ms/step
Epoch 7/100
46/46 - 0s - loss: 10.4884 - 134ms/epoch - 3ms/step
Epoch 8/100
46/46 - 0s - loss: 5.1965 - 133ms/epoch - 3ms/step
Epoch 9/100
46/46 - 0s - loss: 3.9463 - 133ms/epoch - 3ms/step
Epoch 10/100
46/46 - 0s - loss: 3.7155 - 135ms/epoch - 3ms/step
Epoch 11/100
46/46 - 0s - loss: 3.2472 - 131ms/epoch - 3ms/step
Epoch 12/100
46/46 - 0s - loss: 3.0378 - 133ms/epoch - 3ms/step
Epoch 13/100
46/46 - 0s - loss: 3.0256 - 135ms/epoch - 3ms/step
Epoch 14/100
46/46 - 0s - loss: 2.8985 - 132ms/epoch - 3ms/step
Epoch 15/100
46/46 - 0s - loss: 2.9646 - 134ms/epoch - 3ms/step
Epoch 16/100
46/46 - 0s - loss: 2.7998 - 132ms/epoch - 3ms/step
Epoch 17/100
46/46 - 0s - loss: 2.7194 - 135ms/epoch - 3ms/step
Epoch 18/100
46/46 - 0s - loss: 2.7232 - 134ms/epoch - 3ms/step
Epoch 19/100
46/46 - 0s - loss: 2.6720 - 134ms/epoch - 3ms/step
Epoch 20/100
46/46 - 0s - loss: 2.7372 - 130ms/epoch - 3ms/step
Epoch 21/100
46/46 - 0s - loss: 2.7917 - 118ms/epoch - 3ms/step
Epoch 22/100
46/46 - 0s - loss: 2.7143 - 134ms/epoch - 3ms/step
```

In [25]:
```
el_wind_speed.fit(x_train_wind_speed,y_train_wind_speed,epochs=100,verbose=2,batch_size=16,callbacks=[callback,earlyStopi
```

```
Epoch 1/100
46/46 - 2s - loss: 49.0710 - 2s/epoch - 51ms/step
Epoch 2/100
46/46 - 0s - loss: 19.2668 - 137ms/epoch - 3ms/step
Epoch 3/100
46/46 - 0s - loss: 17.1051 - 130ms/epoch - 3ms/step
Epoch 4/100
46/46 - 0s - loss: 16.7571 - 136ms/epoch - 3ms/step
Epoch 5/100
46/46 - 0s - loss: 16.4721 - 135ms/epoch - 3ms/step
Epoch 6/100
46/46 - 0s - loss: 16.4099 - 131ms/epoch - 3ms/step
Epoch 7/100
46/46 - 0s - loss: 16.2295 - 135ms/epoch - 3ms/step
Epoch 8/100
46/46 - 0s - loss: 16.2948 - 131ms/epoch - 3ms/step
Epoch 9/100
46/46 - 0s - loss: 16.2587 - 134ms/epoch - 3ms/step
Epoch 10/100
46/46 - 0s - loss: 16.2888 - 133ms/epoch - 3ms/step
```

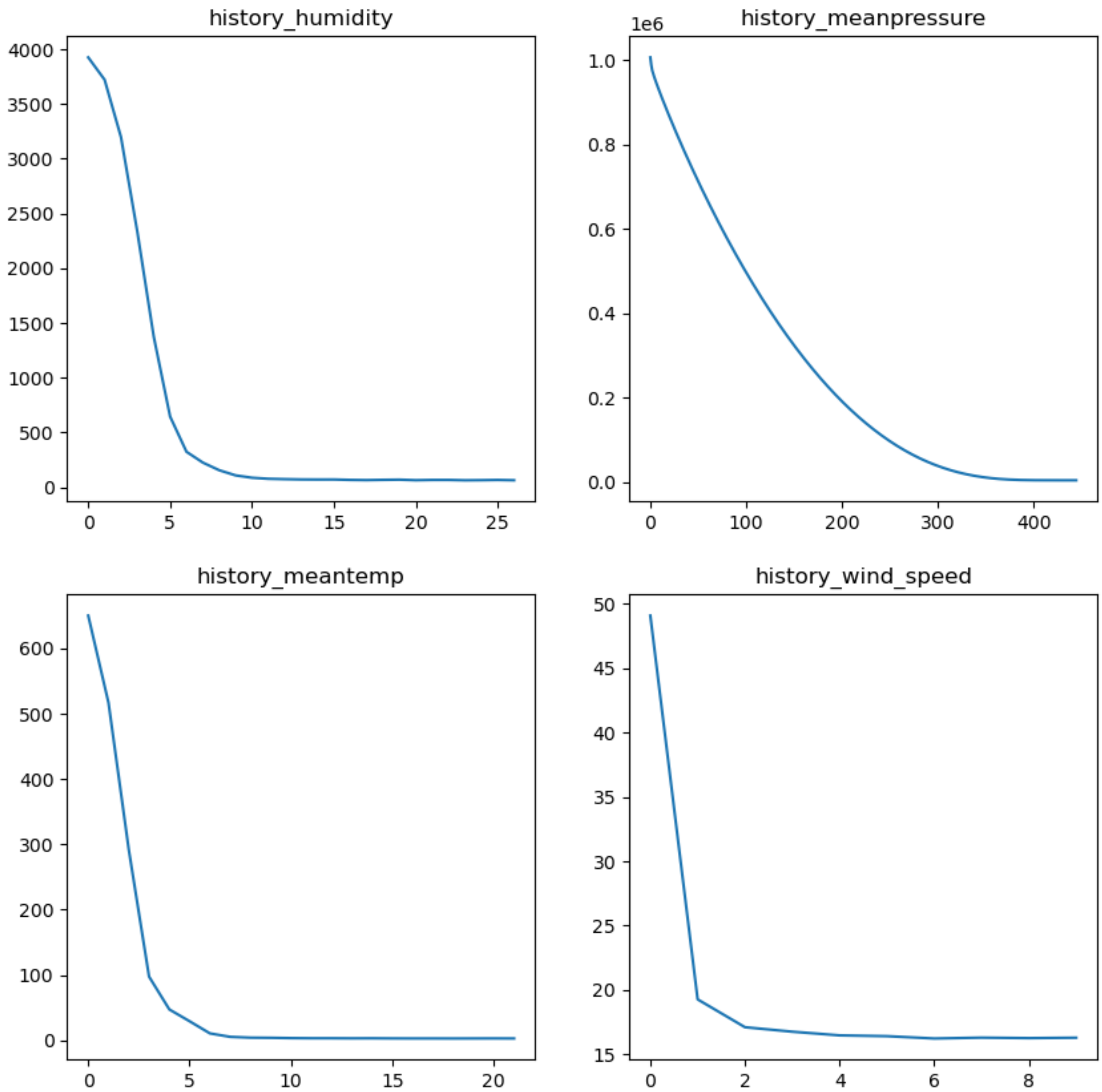
In [26]:
```
history_meanpressure = model_meanpressure.fit(x_train_meanpressure,y_train_meanpressure,epochs=1000,verbose=2,batch_size=
```

```
46/46 - 0s - loss: 619912.2500 - 133ms/epoch - 3ms/step
Epoch 72/1000
46/46 - 0s - loss: 615521.2500 - 134ms/epoch - 3ms/step
Epoch 73/1000
46/46 - 0s - loss: 611151.4375 - 133ms/epoch - 3ms/step
Epoch 74/1000
46/46 - 0s - loss: 606800.2500 - 133ms/epoch - 3ms/step
Epoch 75/1000
46/46 - 0s - loss: 602465.6250 - 129ms/epoch - 3ms/step
Epoch 76/1000
46/46 - 0s - loss: 598150.3750 - 133ms/epoch - 3ms/step
Epoch 77/1000
46/46 - 0s - loss: 593852.4375 - 118ms/epoch - 3ms/step
Epoch 78/1000
46/46 - 0s - loss: 589573.1875 - 135ms/epoch - 3ms/step
Epoch 79/1000
46/46 - 0s - loss: 585309.7500 - 131ms/epoch - 3ms/step
Epoch 80/1000
46/46 - 0s - loss: 581068.1875 - 133ms/epoch - 3ms/step
```

In [27]:
```
def Gen_hist():
    all_hist = ['history_humidity','history_meanpressure','history_meantemp','history_wind_speed']
    for hist in all_hist:
        yield hist
```

Plotting the loss graphs of the different models

In [28]:
```
gen_hist = Gen_hist()
fig,axes = plt.subplots(ncols=2,nrows=2,figsize=(10,10))
for i in range(2):
    for j in range(2):
        hist_now = next(gen_hist)
        axes[i,j].plot(eval(hist_now).history['loss'])
        axes[i,j].set_title(hist_now)
plt.savefig('loss_history.png')
```

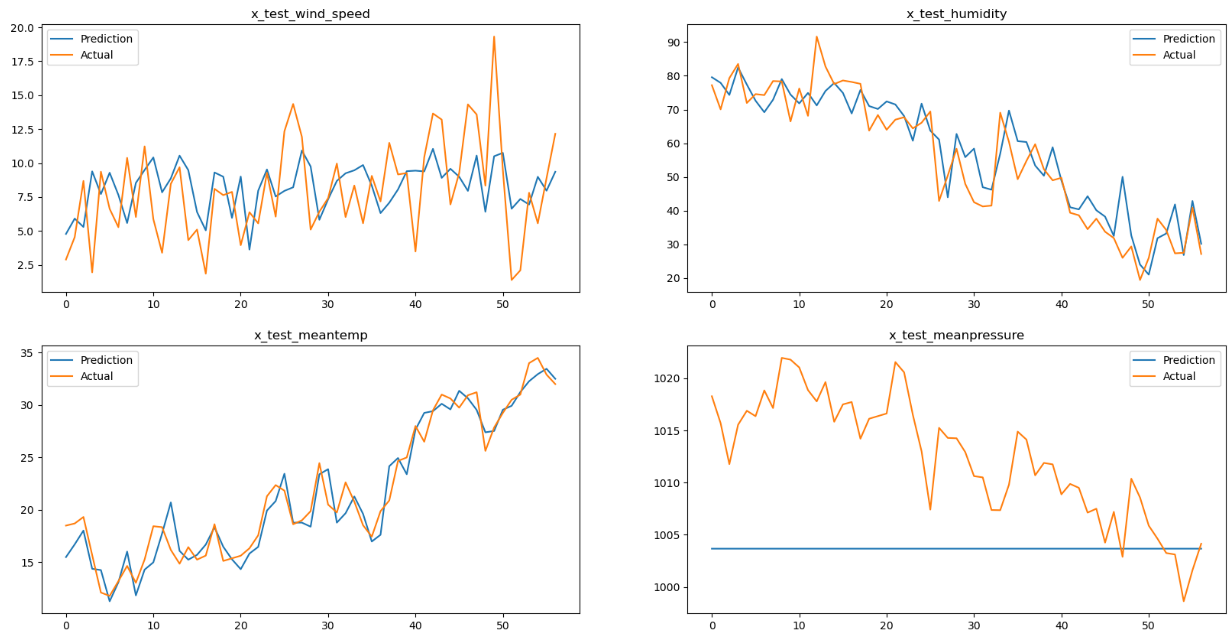


Predicting the weather using the time series data

In [29]:
```
def Gen_test():
    all_test = ['x_test_wind_speed','x_test_humidity','x_test_meantemp','x_test_meanpressure']
    all_y = ['y_test_wind_speed','y_test_humidity','y_test_meantemp','y_test_meanpressure']
    all_model = ['model_wind_speed','model_humidity','model_meantemp','model_meanpressure']
    for test in zip(all_test,all_y,all_model):
        yield test
```

In [30]:
```
gen_test = Gen_test()
fig,axes = plt.subplots(ncols=2,nrows=2,figsize=(20,10))
for i in range(2):
    for j in range(2):
        test_now = next(gen_test)
        axes[i,j].plot(eval(test_now[2]).predict(eval(test_now[0])),label='Prediction')
        axes[i,j].plot(eval(test_now[1]),label='Actual')
        axes[i,j].set_title(test_now[0])
        axes[i,j].legend()
plt.savefig('prediction.png')
```

```
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 0s/step
```



In [31]:
```
os.makedirs('models',exist_ok=True)
```

Improving Models performance

1) Using layer_lstm() instead of layer_gru()

In [32]:
```
model1_meantemp = keras.Sequential([
    keras.layers.LSTM(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='relu'),
    keras.layers.Dense(32,activation='relu'),
    keras.layers.Dense(1)
])
```

In [34]:
```
model1_humidity = keras.Sequential([
    keras.layers.LSTM(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='relu'),
    keras.layers.Dense(32,activation='relu'),
    keras.layers.Dense(1)
])
```

In [35]:
```
model1_wind_speed = keras.Sequential([
    keras.layers.LSTM(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='relu'),
    keras.layers.Dense(32,activation='relu'),
    keras.layers.Dense(64,activation='relu'),
    keras.layers.Dense(1)
])
```

In [36]:
```
model1_meanpressure = keras.Sequential([
    keras.layers.LSTM(8,input_shape=(1,1,)),
    keras.layers.Dense(16,activation='tanh'),
    keras.layers.Dense(32,activation='tanh'),
    keras.layers.Dense(64,activation='tanh'),
    keras.layers.Dense(1)
])
```

In [37]:
```
model1_meantemp.compile(loss='mse',optimizer='adam')
model1_humidity.compile(loss='mse',optimizer='adam')
model1_wind_speed.compile(loss='mse',optimizer='adam')
model1_meanpressure.compile(loss='mse',optimizer='adam')
```

In [38]:
```
history1_humidity = model1_humidity.fit(x_train_humidity,y_train_humidity,epochs=100,verbose=2,batch_size=16,callbacks=[c
```

```
Epoch 1/100
46/46 - 2s - loss: 3963.6135 - 2s/epoch - 48ms/step
Epoch 2/100
46/46 - 0s - loss: 3889.0544 - 151ms/epoch - 3ms/step
Epoch 3/100
46/46 - 0s - loss: 3500.6328 - 128ms/epoch - 3ms/step
Epoch 4/100
46/46 - 0s - loss: 2126.1179 - 131ms/epoch - 3ms/step
Epoch 5/100
46/46 - 0s - loss: 738.6169 - 122ms/epoch - 3ms/step
Epoch 6/100
46/46 - 0s - loss: 270.3648 - 140ms/epoch - 3ms/step
Epoch 7/100
46/46 - 0s - loss: 194.1878 - 130ms/epoch - 3ms/step
Epoch 8/100
46/46 - 0s - loss: 139.0719 - 129ms/epoch - 3ms/step
Epoch 9/100
46/46 - 0s - loss: 107.1884 - 124ms/epoch - 3ms/step
Epoch 10/100
46/46 - 0s - loss: 85.8247 - 119ms/epoch - 3ms/step
Epoch 11/100
46/46 - 0s - loss: 76.6262 - 134ms/epoch - 3ms/step
Epoch 12/100
46/46 - 0s - loss: 71.5407 - 141ms/epoch - 3ms/step
Epoch 13/100
46/46 - 0s - loss: 68.2660 - 126ms/epoch - 3ms/step
Epoch 14/100
46/46 - 0s - loss: 66.5848 - 135ms/epoch - 3ms/step
Epoch 15/100
46/46 - 0s - loss: 65.4801 - 133ms/epoch - 3ms/step
Epoch 16/100
46/46 - 0s - loss: 65.3950 - 136ms/epoch - 3ms/step
Epoch 17/100
46/46 - 0s - loss: 65.6845 - 124ms/epoch - 3ms/step
Epoch 18/100
46/46 - 0s - loss: 65.1627 - 135ms/epoch - 3ms/step
Epoch 19/100
46/46 - 0s - loss: 65.9406 - 138ms/epoch - 3ms/step
Epoch 20/100
46/46 - 0s - loss: 64.3730 - 130ms/epoch - 3ms/step
Epoch 21/100
46/46 - 0s - loss: 64.0230 - 129ms/epoch - 3ms/step
Epoch 22/100
46/46 - 0s - loss: 65.0695 - 130ms/epoch - 3ms/step
Epoch 23/100
46/46 - 0s - loss: 70.5483 - 134ms/epoch - 3ms/step
Epoch 24/100
46/46 - 0s - loss: 64.7856 - 144ms/epoch - 3ms/step
```

In [39]:
```
history1_meantemp = model1_meantemp.fit(x_train_meantemp,y_train_meantemp,epochs=100,verbose=2,batch_size=16,callbacks=[c
```

```
Epoch 1/100
46/46 - 2s - loss: 676.1584 - 2s/epoch - 48ms/step
Epoch 2/100
46/46 - 0s - loss: 580.0793 - 124ms/epoch - 3ms/step
Epoch 3/100
46/46 - 0s - loss: 381.2299 - 132ms/epoch - 3ms/step
Epoch 4/100
46/46 - 0s - loss: 132.2481 - 136ms/epoch - 3ms/step
Epoch 5/100
46/46 - 0s - loss: 43.4971 - 114ms/epoch - 2ms/step
Epoch 6/100
46/46 - 0s - loss: 25.7076 - 119ms/epoch - 3ms/step
Epoch 7/100
46/46 - 0s - loss: 10.9361 - 147ms/epoch - 3ms/step
Epoch 8/100
46/46 - 0s - loss: 5.4234 - 165ms/epoch - 4ms/step
Epoch 9/100
46/46 - 0s - loss: 4.0332 - 118ms/epoch - 3ms/step
Epoch 10/100
46/46 - 0s - loss: 3.5563 - 116ms/epoch - 3ms/step
Epoch 11/100
46/46 - 0s - loss: 3.3333 - 133ms/epoch - 3ms/step
Epoch 12/100
46/46 - 0s - loss: 3.0918 - 166ms/epoch - 4ms/step
Epoch 13/100
46/46 - 0s - loss: 2.9496 - 150ms/epoch - 3ms/step
Epoch 14/100
46/46 - 0s - loss: 2.8431 - 122ms/epoch - 3ms/step
Epoch 15/100
46/46 - 0s - loss: 2.8074 - 166ms/epoch - 4ms/step
Epoch 16/100
46/46 - 0s - loss: 2.7797 - 114ms/epoch - 2ms/step
Epoch 17/100
46/46 - 0s - loss: 2.8800 - 134ms/epoch - 3ms/step
Epoch 18/100
46/46 - 0s - loss: 2.8069 - 133ms/epoch - 3ms/step
Epoch 19/100
46/46 - 0s - loss: 2.6966 - 116ms/epoch - 3ms/step
Epoch 20/100
46/46 - 0s - loss: 2.9090 - 122ms/epoch - 3ms/step
Epoch 21/100
46/46 - 0s - loss: 2.6962 - 130ms/epoch - 3ms/step
Epoch 22/100
46/46 - 0s - loss: 2.6664 - 132ms/epoch - 3ms/step
Epoch 23/100
46/46 - 0s - loss: 2.7135 - 132ms/epoch - 3ms/step
Epoch 24/100
46/46 - 0s - loss: 2.7340 - 132ms/epoch - 3ms/step
Epoch 25/100
46/46 - 0s - loss: 2.6316 - 135ms/epoch - 3ms/step
Epoch 26/100
46/46 - 0s - loss: 2.7635 - 116ms/epoch - 3ms/step
Epoch 27/100
46/46 - 0s - loss: 2.6839 - 120ms/epoch - 3ms/step
Epoch 28/100
46/46 - 0s - loss: 2.6501 - 130ms/epoch - 3ms/step
```

In [40]:
```
history1_meanpressure = model1_meanpressure.fit(x_train_meanpressure,y_train_meanpressure,epochs=1000,verbose=2,batch_siz
```

```
Epoch 126/1000
46/46 - 0s - loss: 407055.9375 - 134ms/epoch - 3ms/step
Epoch 127/1000
46/46 - 0s - loss: 403595.5000 - 144ms/epoch - 3ms/step
Epoch 128/1000
46/46 - 0s - loss: 400152.8125 - 153ms/epoch - 3ms/step
Epoch 129/1000
46/46 - 0s - loss: 396726.6250 - 134ms/epoch - 3ms/step
Epoch 130/1000
46/46 - 0s - loss: 393315.7500 - 156ms/epoch - 3ms/step
Epoch 131/1000
46/46 - 0s - loss: 389923.3125 - 160ms/epoch - 3ms/step
Epoch 132/1000
46/46 - 0s - loss: 386544.2500 - 141ms/epoch - 3ms/step
Epoch 133/1000
46/46 - 0s - loss: 383184.7188 - 134ms/epoch - 3ms/step
Epoch 134/1000
46/46 - 0s - loss: 379838.0312 - 150ms/epoch - 3ms/step
Epoch 135/1000
```

In [41]:

```
history1_wind_speed = model1_wind_speed.fit(x_train_wind_speed,y_train_wind_speed,epochs=100,verbose=2,batch_size=16,call
```

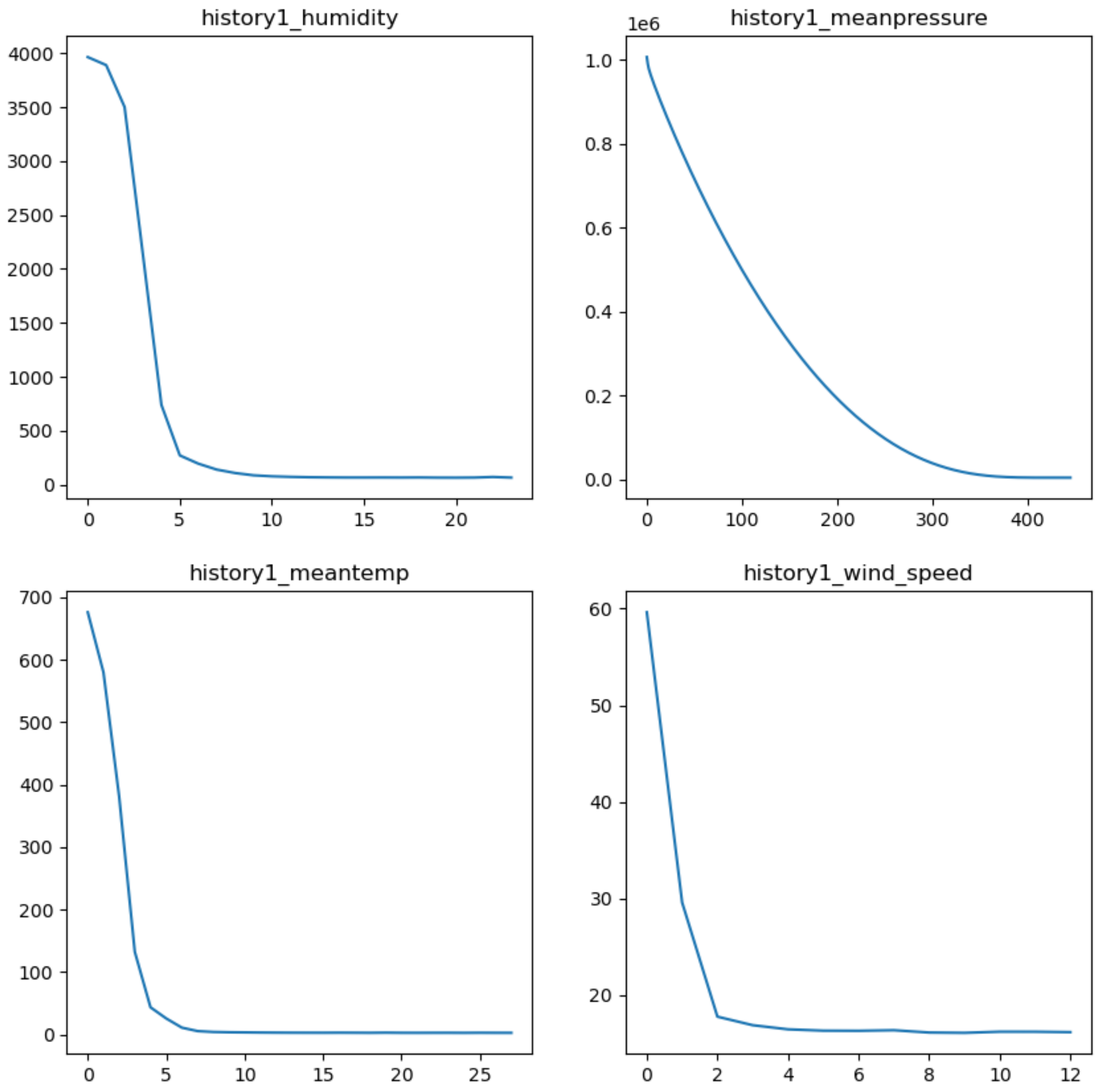
```
Epoch 1/100
46/46 - 3s - loss: 59.6198 - 3s/epoch - 55ms/step
Epoch 2/100
46/46 - 0s - loss: 29.5996 - 140ms/epoch - 3ms/step
Epoch 3/100
46/46 - 0s - loss: 17.7838 - 131ms/epoch - 3ms/step
Epoch 4/100
46/46 - 0s - loss: 16.8994 - 150ms/epoch - 3ms/step
Epoch 5/100
46/46 - 0s - loss: 16.4726 - 132ms/epoch - 3ms/step
Epoch 6/100
46/46 - 0s - loss: 16.3339 - 149ms/epoch - 3ms/step
Epoch 7/100
46/46 - 0s - loss: 16.3213 - 134ms/epoch - 3ms/step
Epoch 8/100
46/46 - 0s - loss: 16.3792 - 134ms/epoch - 3ms/step
Epoch 9/100
46/46 - 0s - loss: 16.1410 - 134ms/epoch - 3ms/step
Epoch 10/100
46/46 - 0s - loss: 16.1099 - 130ms/epoch - 3ms/step
Epoch 11/100
46/46 - 0s - loss: 16.2273 - 134ms/epoch - 3ms/step
Epoch 12/100
46/46 - 0s - loss: 16.2240 - 136ms/epoch - 3ms/step
Epoch 13/100
46/46 - 0s - loss: 16.1816 - 137ms/epoch - 3ms/step
```

In [42]:

```
def Gen_hist():
    all_hist = ['history1_humidity','history1_meanpressure','history1_meantemp','history1_wind_speed']
    for hist in all_hist:
        yield hist
```

In [43]:

```
gen_hist = Gen_hist()
fig,axes = plt.subplots(ncols=2,nrows=2,figsize=(10,10))
for i in range(2):
    for j in range(2):
        hist_now = next(gen_hist)
        axes[i,j].plot(eval(hist_now).history['loss'])
        axes[i,j].set_title(hist_now)
plt.savefig('loss_history.png')
```

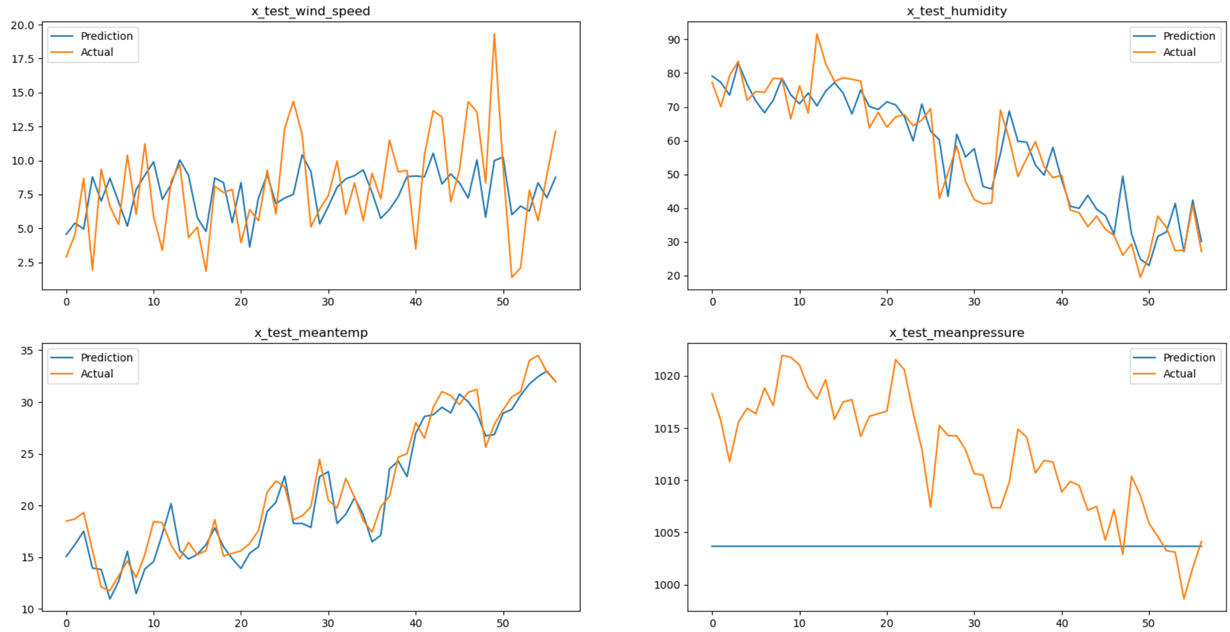


In [45]:

```
def Gen_test():
    all_test = ['x_test_wind_speed','x_test_humidity','x_test_meantemp','x_test_meanpressure']
    all_y = ['y_test_wind_speed','y_test_humidity','y_test_meantemp','y_test_meanpressure']
    all_model = ['model1_wind_speed','model1_humidity','model1_meantemp','model1_meanpressure']
    for test in zip(all_test,all_y,all_model):
        yield test
```

In [46]:
```
gen_test = Gen_test()
fig,axes = plt.subplots(ncols=2,nrows=2,figsize=(20,10))
for i in range(2):
    for j in range(2):
        test_now = next(gen_test)
        axes[i,j].plot(eval(test_now[2]).predict(eval(test_now[0])),label='Prediction')
        axes[i,j].plot(eval(test_now[1]),label='Actual')
        axes[i,j].set_title(test_now[0])
        axes[i,j].legend()
plt.savefig('prediction.png')
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_predict_function.<locals>.predict_function at 0x00
000132386BD090> 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 object
s instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_re
tracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/fun
ction#controlling_retracing (https://www.tensorflow.org/guide/function#controlling_retracing) and https://www.tensorflo
w.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.
2/2 [==============================] - 0s 3ms/step
WARNING:tensorflow:6 out of the last 11 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0
0000132386BED40> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be du
e to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python obje
cts 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/f
unction#controlling_retracing (https://www.tensorflow.org/guide/function#controlling_retracing) and https://www.tensorf
low.org/api_docs/python/tf/function (https://www.tensorflow.org/api_docs/python/tf/function) for  more details.
2/2 [==============================] - 0s 4ms/step
2/2 [==============================] - 0s 0s/step
2/2 [==============================] - 0s 3ms/step
```



The task involved training models using recurrent neural networks (RNNs) for forecasting weather-related parameters, focusing on humidity. Throughout the process, several steps were followed:

1. Model Training:

- Multiple models were trained using RNN architectures (specifically LSTM and GRU) to forecast weather parameters (humidity in this case).
- The training involved splitting the dataset into training and validation sets.
- Adjustments in RNN layers, hyperparameters, and callbacks like early stopping were used during the training process to optimize the models.

2. Evaluation on Validation Data:

- The models were evaluated on the validation data, calculating Mean Absolute Error (MAE) to assess their performance.
- The MAE is a metric indicating the average absolute difference between predictions and actual values. Lower MAE values signify better performance.

3. Evaluation on Testing Data:

- Finally, the trained models were evaluated on the unseen testing dataset to provide an assessment of their generalization performance.
- Evaluating the models on the testing set helps to understand how well they perform on new, unseen data, providing insights into real-world applicability.

Conclusion:

- The validation and testing evaluations, especially in terms of MAE, provide a measure of the models' performance and their ability to forecast humidity accurately.
- The comparison of models' performances on both validation and testing data helps in understanding their robustness and generalization capacity.

- Lower MAE on the testing data indicates that the models are better at predicting humidity and can potentially be more reliable in real-world applications.

The choice of the final model or the most suitable approach relies on its performance, specifically the MAE, on the testing data and its consistency with the performance observed during validation. Lower MAE on the testing set typically signifies a more reliable and effective model for weather forecasting

In [ ]: